Distributed Collaborative Structuring — A Data Mining Approach to Information Management in Loosely Coupled Domains

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund

an der Fakultät Informatik

von

Michael Wurst

Dortmund

2008

Tag der mündlichen Prüfung:	18.06.2008
Dekan:	Prof. Dr. Peter Buchholz
Gutachter:	Prof. Dr. Katharina Morik Prof. Dr. Heinrich Müller

Danksagung

Ich möchte mich zunächst bei Katharina Morik bedanken, für die Betreuung meiner Arbeit, für ihre Anregungen, ihre Kritik, ihren Enthusiasmus, ihre Geduld und nicht zu letzt auch für die Freiheit, die Sie mir während meiner Zeit am Lehrstuhl gewährt hat. Bei Heinrich Müller bedanke ich mich herzlich für die Übernahme des Zweitgutachtens, für die Korrekturen der Arbeit und für die Anregungen, die mir dabei geholfen haben, die Arbeit klarer und verständlicher zu machen. Peter Padawitz und Eike Riedemann danke ich für ihre Bereitschaft an der Kommission mitzuwirken.

Des Weiteren gilt mein Dank auch meinen aktuellen und ehemaligen Kollegen am Lehrstuhl, die durch ihre offene, kollegiale Art stets ein produktives, positives Klima geschaffen haben, welches für die erfolgreiche wissenschaftliche Arbeit so wichtig ist. Besonders möchte ich dabei Ingo Mierswa danken, für zahlreiche Anregungen und Diskussionen, vor allem aber auch für die Entwicklung des RapidMiner Systems. Martin Scholz und Stefan Haustein danke ich mich für viele interessante und wichtige Diskussionen zum Thema Data Mining und darüber hinaus. Des Weiteren gilt mein Dank den zahlreichen Diplomanden mit denen ich in den letzten Jahren zusammenarbeiten durfte und den Mitgliedern der Projektgruppe 461, die mit viel Engagement den Nemoz Prototypen erstellt haben.

Auch außerhalb des Fachbereichs wurde ich in meiner Arbeit unterstützt. Hier gilt mein besonderer Dank Jasminko Novak, für die produktive Zusammenarbeit an dem Projekt Awake, sowie für viele interessante Diskussionen. Claus Weihs und seinem Team danke ich für die Bereitstellung ihrer Daten im Rahmen des SFB 475. Andreas Hotho danke ich für die Bereitstellung des Bibsonomydatensatzes, für Anregungen und für die gute Zusammenarbeit im Projekt KDubiq.

Contents

List of Figures

List of Tables

Overview of the Formalism

General

sim	similarity measure, defined on items, users or taxonomies
d	distance measure defined on items, users or taxonomies
f	a function, assumed true relationsship
h	hypothesis on a true relationship
L	loss function used to estimate the expected error
E	expected value of a random variable
p	probability of an event
i, j, k, l, m	index, meaning depends on context
$\overline{a}$	average value
$\sigma(a)$	standard deviation
α, β, γ	constant values, parameters
2^A	the power set of set A (set of all subsets of A)
$v \in \mathbb{R}^k$	a vector of values
$v_i \in \mathbb{R}$	the value of the i-th feature of a vector

Items, Users and Features

D	finite set of all items (identifiers)
$S \subseteq D$	specific set of items
$x, y, z \in D$	individual items
$X : D \to \mathbb{R}$	a feature
$X(x)$	value of feature $X \in \mathcal{X}$ for item $x \in D$
$\mathcal{X}$	all available features for a fixed set of items D
$\mathbf{X} \subseteq \mathcal{X}$	a set of features
$X_i \in \mathbf{X}$	the i-th feature in set $\mathbf{X}$
$X \sim X'$	denotes that two features $X, X' \in \mathcal{X}$ are alternative
U	set of users of the system
$u \in U$	a user

Distributed Feature Engineering

$mb(\mathbf{X}, X)$	denotes that $\mathbf{X} \subseteq \mathcal{X}$ is a Markov blanket for $X \in \mathcal{X}$
$q : 2^{\mathcal{X}} \to \mathbb{R}$	the accuracy by which a data mining task is achieved
T	a set of data mining tasks
$t_i \in T$	a single data mining task
$t_i \prec t_j$	task $t_i \in T$ must be solved before $t_j \in T$ is solved
$q_i : 2^{\mathcal{X}} \to \mathbb{R}$	the accuracy by which task $t_i \in T$ is achieved given a feature set
q^*	the accuracy by which a set of tasks T is achieved
$\mathbf{X}_{\mathrm{B}} \subseteq \mathcal{X}$	a set of base features
w_{il}	the weight of the l-th feature $X_l \in \mathbf{X}_B$ for a task $t_i \in T$
$\mathbf{X}_{\mathrm{T}} \subseteq \mathcal{X}$	the aggregated feature set of a set of tasks T
$\mathbf{X}_{cand} \subseteq \mathcal{X}$	a set of candidate features for feature selection
$E_{t,t'}$	expected loss if a hypothesis optimized for task t is applied to task t'

Tags, Aspects and Functions

C	a set concepts or tags
$c \in C$	an individual concept/tag
$ext : C \to 2^D$	extension function denoting which items are covered by a concept
$c \prec c'$	denotes that $c \in C$ is a sub concept of $c' \in C$
$C_x \subseteq C$	all concepts in C that contain item $x \in D$
$C_{xy} \subseteq C$	all concepts in C that contain both items $x, y \in D$
$\mathcal{C}_{\mathrm{S}}$	the set of all possible concept structures C on items in $S \subseteq D$
$cq : \mathcal{C}_{\mathrm{S}} \to \mathbb{R}$	a quality measure for cluster structures on $S \subseteq D$
$z_{ji} \in \mathbb{R}$	value of the i-th feature of the centroid assigned to cluster c_j
$\mathcal{I}(C, C')$	the mutual information of two sets of concepts C, C'
A	a set of aspects
$a \in A$	an individual aspect
$asp : C \to A$	function assigning each tag $c \in C$ to an aspect $a \in A$
$C(a) \subseteq C$	all concepts belonging to aspect $a \in A$
$x \sqsubset a$	item $x \in D$ is covered by aspect $a \in A$
$S_a \subseteq D$	items covered by an aspect $a \in A$
G	a finite set of classes
$g < g'$	denotes that g is a subgroup of g'
$\varphi : S \to 2^G$	a hierarchical classification function
$\varphi : S \to G$	a classification function
$\varphi_a : S_a \to 2^{C(a)}$	an aspect classification function
X_c	feature derived from tag c
$\mathbf{X}_{\mathrm{C}}$	feature space derived from a set of tags C
$spec : C \to \mathbb{R}_0^+$	the specificy of a tag
$spec_D : D \times A \to \mathbb{R}_0^+$	the specificy of a tag concerning an aspect

LACE

$x \sqsubset_\alpha \varphi$	a classification functions covers an item sufficiently
Z_φ	a set of points that represent a function φ
$h_i : S \to \mathbb{N}$	maps items in the i-th output function to the index of the corresponding bag
S_{ij}	the j-th partition of the i-th output function
$\varphi_i \prec \varphi_j$	denotes that two functions are in direct sub function relation
I	set of input functions (tags) to collaborative structuring
O	set of results of collaborative structuring
q^*	the quality of a single output clustering
q	the quality of a set of solutions O

Network

V	set of network nodes in the system
$v \in V$	a node
$F \subseteq V^2$	communication channels
$nbrs(v) \subseteq V$	set of neighbors of a node $v \in V$
$deg(v) \in \mathbb{N}$	degree of a node

Awake

$\mathcal{B}$	set of communities
$\mathbf{B} \in \mathcal{B}$	a single community, with $\mathbf{B} \subseteq U$
N_i	the set of predictors for user u_i
$u_i \prec_{ks} u_j$	denotes that $u_i \in U$ shares knowledge with $u_j \in U$

General Notes

- Throughout the work, the Big O notation is used to indicate the upper bound on the asymptotic behaviour of algorithms dependent on the input size.
- Online resources are cited directly in the text or in a footnote. If the actual version of an online resource could make a difference, a date is attached.
- External work, concepts, algorithms, systems, etc. are cited once at their first occurrence in the text.
- Page numbers are only indicated for articles in journals. Many current conference proceedings are not published in book form and do not have page numbers. All external work is referenced in such a way that this work is uniquely identified.

Overview

Making Internet resources available in a structured way is one of the most important and challenging problems today. While search and filtering technology is becoming more and more powerful, the development of explorative access methods lacks behind. A recent, very successful approach to this problem are social bookmarking systems. These systems allow users to annotate resources with arbitrary textual expressions, called *tags*. These tags can be used to navigate information collections in a very intuitive way. Their success can largely be attributed to three properties. They allow users to describe resources easily, without obeying rules or global schemata, they allow for heterogeneous views of domains and they make the underlying social structure explicit. A downside of this loosely-coupled approach is, however, that tag collection quickly become chaotic and hard to manage, which limits their applicability considerably. This work aims at exploring how data mining methods can be employed to achieve the best of both worlds. On the one hand, users should be free to apply arbitrary annotations to express their personal views. On the other hand, they should be supported in doing so by exploiting patterns in the underlying data and, even more important, by reusing structures in the annotations of other users. Beyond the current hype of the Web 2.0, this leads to several interesting, novel questions in the area of data mining and intelligent data analysis.

The amount of data on the Internet is growing at a speed that makes it impossible to even measure it accurately. Moreover, there is a clear shift from text related resources to multimedia data, such as pictures, audio or video. The shift from the traditional web to a participatory web, in which users with only minimal skills can add content, makes this process even faster. Furthermore, most resources on the Internet are very heterogeneous. The term „long tail content“[6] captures this effect very well. Traditional media could always only focus on popular resources. The Internet, in contrast, allows to share even content that is interesting for only very small niches. While each individual niche might be very small, all niches together make-up a considerable part of the available content.

Searching and navigating large and heterogeneous information spaces in a structured way is extremely challenging. There is a general agreement among researchers from different areas that the only way to cope with this problem is semantic annotation. The most ambitious project into this direction is the semantic web [20]. The basic idea is to describe resources in a way that allows machines to reason about them automatically. An image, for instance, would not be described by its individual pixels but by a set of logic assertions, describing the content of the image semanticlly. Given that all resources on the Internet would be described in such a way, logic based reasoning could be used assisting users in navigation and searching the web in any possible way. There are,

however, some essential problems. First, it is unclear how a semantic annotation of an enormous amount of heterogeneous, complex information on the web can be achieved. Second, the semantic web faces a considerable acceptance problem on the side of the users. Assuming a global semantic (and thus a global view of the world) does not reflect the very subjective way in which users structure their information. Just as anybody wants to furnish one's house or working place, the same holds for the aspects of living accessed through the Internet. Global approaches, such as the semantic web, do not reflect this natural heterogeneity. Therefore, many researchers nowadays doubt that the semantic web will become a reality. Still, we can hope to support users in navigating and searching the web.

However, there must be a paradigm shift, similar to the one experienced in robotics. Sebastian Thrun, one of the pioneers of adaptive robotics, points out that the key to build a car that drives autonomously through the desert was to develop pragmatic solutions. Given that each car is equipped with a camera, a long time vision in robotics was to automatically recognize all entities in the range of the camera in real-time and put them into relation with given background knowledge. Based on logical entailment and planning, optimal actions could then be chosen. However, till now, it is not possible to build pattern recognition systems that would recognize arbitrary entities. Moreover, logic based planning and entailment is often much too slow to enable acting in an environment that requires short response times. The key insight to solve this problem is that it is not necessary to recognize everything in the environment. To drive a car, the most important thing is to recognize the road and obstacles. This goal can be achieved efficiently by simply comparing the portions of the camera image that are known to contain no obstacles to the rest of the image [180].

On the web, we face a similar problem. There is a large amount of highly complex and heterogeneous data. Instead of trying to annotate this data in a complete and accurate way, we can choose task-oriented approaches that are based on the annotations that we are actually given, how noisy or incomplete they may be. A typical example for methods that are not based on a semantic annotation and do still deliver highly helpful assistance to users are social recommender systems [160]. Users rate items, such as books, and these ratings are then used to generate recommendations for other users that rated items similarly. Collaborative filtering methods do not try to annotate or analyze the underlying items. This technique can be applied to any kind of content. It allows different communities and individuals to have different views or preferences. Recommender systems are a cornerstone of the current Internet and can be found in a broad range of applications ranging from online stores, such as Amazon, over music organizers to web site recommenders [156].

However, merely filtering items is not enough. Rather, a way of annotation is needed that allows users to organize and navigate complex information spaces. In this work, such an approach is developed. The basic idea is to allow users to tag items with hierarchical classes instead of just selecting or rating them. This process can be rather work intensive, as it requires not only knowledge of the items but knowledge of the corresponding domain, as well. Therefore, the users should be supported in creating and maintaining

such personal access structures, just as collaborative filtering supports them in assessing the quality of items. Two major assistance tasks are automatic supervised tagging and automatic unsupervised tagging. For supervised tagging, the system automatically annotates new items with the tags defined by the current user. In terms of data mining, this is a classification task. Unsupervised tagging tries to find appropriate tags, if no user tagging is given. This corresponds to a clustering task. Both tasks should be achieved by exploiting the tags assigned by other users. The term *collaborative structuring* was developed for this approach to express the synthesis of information structuring with collaborative recommenders [198] and to cover classification and clustering.

A key to achieve collaborative structuring is data representation. First, the representation of data is essential for enabling successful data mining. Even very simple algorithms can be extended to capture arbitrarily complex structures by providing an adequate data representation [119]. For users, the representation of data is essential for accessing and navigating information structures.

The problem of finding an adequate representation will be denoted as representation problem. This representation problem is modeled as problem of selecting a subset of features from a possibly infinitely large number of features that describe the data. The problem of collaborative structuring is then mapped to another, more general problem, namely to the problem of finding an optimal representation for a set of data mining tasks. In contrast to the traditional, single-task representation problem, we now search for a adequate representations for several, heterogeneous, partially related data mining tasks. This problem is formalized and analyzed as combinatorial optimization problem. Approaches to solve this problem will be denoted as *distributed feature extraction*.

An important issue in implementing distributed feature extraction is the distributedness of many information collections in loosely coupled networks. While the Internet allows to exchange information between any two points, there are still several limitations. Mobile devices or cell phones are not connected to the Internet via a high speed connection. Sharing large amounts of data with such devices is prohibitively expensive. For these reasons, there is a trend to ad hoc and p2p networks, that allow for a better resource allocation and can be used with network technology such as Wireless LANs. As a consequence, sophisticated *distributed data mining* techniques are needed that can be applied on the Internet, as well as in ad hoc and p2p networks [41].

A major aim of this work is explore the interrelation of loosely-coupled information structuring with distributed data mining for multiple heterogeneous task. This work takes the following approach. First, the general representation problem for multiple tasks is defined and analyzed. This problem occurs in several current application scenarios and is, thus, not restricted to information structuring. Several methods for distributed feature extraction are developed and it is shown that they can be implemented efficiently into a p2p network. In a next step, the representation problem is put into relation with information structuring and it is shown how users can be supported in collaboratively structuring their information using data mining and distributed feature extraction. Finally, it is shown that the traditional formulations of the clustering problem are not

well-suited to structure information collaboratively in an unsupervised way. Based on this observation, a new formal problem definition is provided together with an efficient algorithm to solve this task. The contributions of this work are the following:

1. A summary of existing relevant research on information structuring, data mining and distributed computing, as well as the interrelations among these topics is given.
2. Based on existing research in the area of feature selection for single-task learning, the representation problem for multiple tasks is defined and analyzed from a combinatorial point of view. Since more than one task is involved, definitions, such as the minimality of a feature set, must be generalized. This leads to the novel, general challenge of distributed feature extraction. Several methods for distributed feature extraction are developed and analyzed. These methods are shown to improve over existing single task methods in two respects. First, they make the process of feature extraction more efficient, second, they yield smaller aggregated feature sets. Also, they make no assumptions on the temporal order of learning or on the learning algorithm that is applied, which is in contrast to methods of multi-task learning. Moreover, the proposed methods can be applied in a wide range of scenarios, such as p2p networks, as they make only minimal assumptions about the underlying network structure.
3. The problem of user-centric information structuring is analyzed, identifying two shortcomings of current approaches to social bookmarking. First, these approaches do not allow to separate tags into different aspects, making the organization of tags quite complicated and sensitive to errors. Second, they provide only poor support in assisting users in organizing tag collections.
   As a consequence, a novel representation mechanism for tags is developed, called *aspect-based tagging*. Based on this representation formalism, two especially important tasks to assist users in structuring items are identified, namely collaborative classification and collaborative structuring. Both allow to employ tags assigned by other users to provide assistance to a given user in finding optimal representations. For clustering, it can be shown that this approach yields an intuitive extension of cluster ensembles for hierarchical cluster models. An empirical study shows the benefits of using tags instead of using content only, as in the case of traditional classification and clustering approaches.
4. While collaborative classification and clustering produce sound and accurate results, it is argued why the traditional formulation of a clustering task does not yet meet the requirements of information structuring from a user's perspective. Therefore, a novel definition for this task is given and an algorithm to solve it, called localized, alternative cluster ensembles. This algorithm is shown to be superior to traditional clustering approaches in two respects. First, it guarantees the soundness of the results from a user's perspective, captured by the notion of structure preservation. Second, it yields better results on a quantitative level for two different application areas, one concerned with textual data and the second one concerned with multimedia data.

5. It is shown how the methods developed in this work were applied in two application areas, namely distributed media organization and distributed knowledge management in expert communities.

This work is structured into three parts. Part one presents background information and related work in the areas of data mining, (collaborative) information structuring and distributed systems. Part two first analyzes the general problem of feature extraction for several tasks and develops several algorithms and optimizations to solve it. Then, this framework is applied to information structuring. Part three contains two case studies for the developed methods, one in the domain of heterogeneous expert communities, the second one in the domain of distributed, collaborative media organization.

Part I

Background

1 Structuring Information

1.1 Introduction

Accessing information in a structured way is an essential part of almost every application based on information technology. This is especially true for the Internet. The large amount of information provided through the Internet can only be made utile, because we can rely on its underlying structure as guideline and orientation. Just imagine all websites would be printed out and all these printouts would be piled up in random order. Even for a small number of websites the resulting stack would be almost worthless.

The concept of a „structure“ is very general in this sense. You can structure your book collection by assigning it to different topics, you can structure your favorite webpages into bookmark folders or your music collections in a file system. In a more general sense, the ability to impose structures on entities is a precondition of knowledge and communication in general. Usually, structuring a set of items means putting them into some kind of a relation to each other. This may happen implicitly (without the user being aware of the underlying structure) or explicitly. This work is mostly concerned with the second case.

Why would we like to impose an explicit structure on an information collection? A typical example of the explicit use of structured information on the Internet are directories that allow users to find resources by browsing a set of hierarchically structured topics. This approach is taken by Web directories, such as Yahoo[1] or Open Directory[2], catalogues of online retailers, such as Amazon[3], topic structures used in (digital) libraries, such as the ACM classification[4], and many others. Besides query based search, browsing is currently the second major search strategy in large information collections. Users usually browse information collections because they cannot express their information need in terms of a search query or because they only have and unspecific information need, thus do not know exactly what they are looking for.

However, searching for items that fulfill a specific or unspecific information need is not the only use of explicitly imposing a structure on an underlying information collection. Often we face an unstructured set of items and need to get an exhaustive overview of them. Analyzing each item individually is not possible for any information collection that is not trivially small. We can, however, use algorithms that group a large set of items into

[1] http://www.yahoo.com
[2] http://www.dmoz.org
[3] http://www.amazon.com
[4] http://www.acm.org/class

a small number of groups of similar items. Instead of reviewing each item individually, the user must only analyze a small number of representative items of each group. This method is known in data analysis as clustering and is applied in many application areas ranging from document retrieval to customer segmentation for marketing purposes [81].

Finally, there is a third important reason why we would like information to be explicitly structured. While humans can deal with fairly unstructured information, such as natural language texts, automatic processing of such resources is very limited. Therefore, it is desirable to represent information in a highly structured way that allows processing it by (intelligent) software. A very ambitious project to represent resources on the Internet in a formal way is the semantic web [20]. Based on a formal description of all web resources, access could be much more specific than it is using term based queries. A common vision is, for instance, that users would simply ask questions, such as „How is the new film by Francis Ford Coppola called and where can I see it nearby on Saturday evening?“. The software would then gather all necessary information from different, semantically structured Internet sources, and produce a final answer. While this vision is very appealing, there a several serious obstacles that need to be overcome to make it a reality. They will be discussed below.

In this chapter we first review some basic formalisms for structuring information and how they are employed to make large information collections easier to access and organize. These formalisms are similarity, cluster structures, taxonomies and formal ontologies. For each of them, we will discuss the basic formalism, how they are used in information structuring (especially on the Internet) and how they can be obtained automatically from data. Obviously, not all of the existing methods can be described in detail here. Rather we focus on methods that play an important role in the subsequent chapters of this work.

1.2 Basic Concepts

One common basic concept in all formalisms presented below is the notion of a „thing“ that is put into relation with other things. This concepts reoccurs under different names in different approaches. In formal logic, the set of all things is usually called universe of discourse. In the context of the semantic web things are called *resources*. Object oriented approaches call them objects or instances. In the following we choose the term „item“ to talk about these basic entities, which is relatively neutral.

Definition 1.2.1. An *item* is a uniquely identified entity. The set of all possible items is denoted as D. For sake of formal simplicity, we assume D to be a finite set and to be totally ordered in some arbitrary way.

In the scope of this work, the items we deal with are mostly web pages, music files, scientific documents and similar things. The methods presented here are, however, not limited to this kind of information.

We assume that each item can be associated with a set of *features* or attributes. Formally, we define a feature as follows.

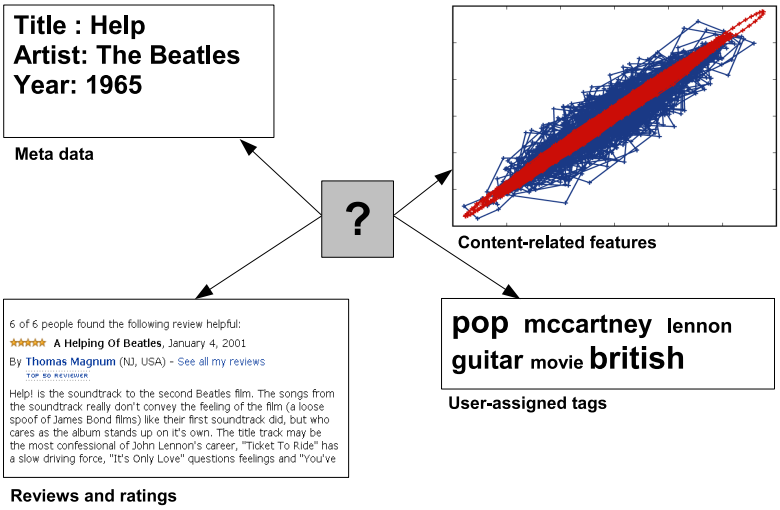


Figure 1.2.1: Items can often be described by various features of different type and heterogeneous source. In this example, music is described by structured metadata, by user-assigned tags, by ratings and reviews and by features extracted from the raw content of the music file.

Definition 1.2.2. A *feature* is a vector

$$X \in \mathbb{R}^{|D|}$$

that is indexed by the items in D,

$$X = \begin{pmatrix} X(x_1) \\ .. \\ X(x_i) \\ .. \\ X(x_{|D|}) \end{pmatrix}$$

where the items $x_i \in D$ are totally ordered by some arbitrary order (e.g. lexicographic) that is identical for all features. For convenience, we alternatively write $X : D \to \mathbb{R}$. Thus, $X(x_i)$ denotes the value of feature X for the item $x_i \in D$.

Note that D is assumed to be finite and fixed. A special case of features are *binary features*, that denote logical expression. For the sake of formal simplicity, we map binary features to a subset of $\mathbb{R}$ containing two elements, namely $\{0, 1\}$. These two values are associated with the logical expressions *true* $(= 1)$ and *false* $(= 0)$, as to denote whether the item has the associated feature, or not. Nominal features are not discussed here. The set of all features that are available for a fixed D is denoted as $\mathcal{X}$. We can assume that this set is totally ordered in some way (e.g. lexicographically). This set can be finite or infinite, depending on the application area. In the following we assume this set to be finite, unless stated otherwise.

Features can be of different type and origin. Music files, for instance, have features, such as the year they were published. Text documents are often automatically annotated by relevant terms they contain. Webpages, movies, etc. can be represented in similar ways. Figure 1.2.1 shows an example.

Some features are easy to understand for human users, such as meta data on the topic, and can directly be used to access and search an information collection. Others are rather abstract but can still help to structure an information space automatically.

Usually, we deal with a subset of all possible features $\mathbf{X} \subseteq \mathcal{X}$ that describe the items. Such a set will be denoted as *feature set*. The value that a given feature assigns to an item is denoted as *feature value*. The vector of all feature values for a given item is called a *feature vector*.

We assume features to be globally uniquely identified and static. Thus, each feature has a unique name and value $X(x)$ does not change over time for any $x \in D$ and $X \in \mathcal{X}$.

Items can be grouped into item collections. An *item collection* is simply a set of items, for instance the set of all scientific publications that occurred in a specific journal. This work is mostly concerned with personal item collections. A *personal item collection* is the set of items owned by a single user. The term may refer to different things depending on the context. A personal collection of music files could be all music files the user stores on her wearable player, the personal collection of webpages would mostly be a set of references to webpages that a user bookmarked etc. In most applications, we can assume that such personal item collections significantly overlap among users. Many users own the same music files, for instance.

In the following we do not make any further assumption about users other than assuming that they are uniquely identified. The set of all users is then simply denoted as U.

1.3 Similarity

1.3.1 Basic Formalism

Structuring means putting things into relation to each other. A very basic way to do so is to say that two items are similar or dissimilar. This can be expressed formally in different ways, depending on the scale that is used[5]

Definition 1.3.1. A *binary similarity relation* is a relation

$$SIM \subseteq D^2$$

Two items are similar exactly if they are in the relation SIM. Usually this relation is required to be reflexive and symmetric.

[5]The following presentation is adapted from [151]

In some cases it might make sense to introduce an additional relation $DISSIM \subseteq D^2$ that denotes that two items are dissimilar. In this case, pairs of items neither in SIM nor in $DISSIM$ have an undefined similarity.

Definition 1.3.2. A *similarity measure* is a function

$$sim : D^2 \rightarrow \mathbb{R}_0^+$$

assigning a degree of similarity to a pair of items. A high value of this function denotes that both items are similar. This representation usually assumes a rational scale. Also, in most cases, this function will be symmetric (thus $\forall x, y \in D : sim(x, y) = sim(y, x)$).

Opposite to the notion of similarity is the notion of distance.

Definition 1.3.3. A *distance measure* is usually expressed on a rational scale by

$$d : D^2 \rightarrow \mathbb{R}_0^+$$

Distance measures are often required to be metric.

Definition 1.3.4. A distance measure $d : D^2 \rightarrow \mathbb{R}_0^+$ is *a metric* if it fulfills for all $x, y, z \in D$ the following conditions:

1. $d(x, y) = 0 \Leftrightarrow x = y$
2. $d(x, y) = d(y, x)$
3. $d(x, z) \leq d(x, y) + d(y, z)$

These properties are used by many algorithms to process similarity information more efficiently. M-Trees [36], for instance, employ the properties of a metric for efficient similarity search.

A distance measure can be converted into a similarity measure and the other way around. Such conversion functions are usually required to at least preserve the ordinal quality of the original measure. This is fulfilled, for example, by

$$sim(x, y) = 1 - \frac{d(x, y)}{1 + d(x, y)}$$

In the following, the terms similarity and distance will be used interchangeable.

1.3.2 Structuring Information by Similarity

Many information systems structure items based on similarity. An important example are item based recommender systems. These systems are applied, for instance, by Amazon, to give recommendations of the form: „Users that bought book x also bought books y,

z“ [156, 155]. Users are often not able to express their information need in terms of a search query. They can, however, name items of which kind they would like additional ones.

Another example are Self Organizing Maps (SOM) [101] used in information search [79]. A SOM projects items in a high dimensional feature space to a two dimensional space preserving as much of the mutual similarity between items as possible. On the resulting map, two items are similar, if they are near to each other. Users can navigate such a map just like a real world map. An extension to basic SOM described in [185] even enriches the map with hills and valleys based on density estimates, making it even better suited for explorative data analysis.

Similarity is also a building block for many other representation mechanism, as clustering, which will be presented below.

1.3.3 Obtaining Similarity from Data

If the underlying domain D is finite, then a similarity measure can be defined explicitly by a symmetric matrix. Usually similarity is rather defined indirectly by referring to underlying properties and relations of the items in D. There is a huge number of different similarity measures for this purpose, of which only some basic ones will be discussed in the following.

In the simplest case, all features are binary. The most popular choice of comparing two binary feature vectors is to compare them bit-wise. Corresponding measures are called matching coefficients [5]. There are four possible cases for each feature. First, it can be true for both items (case a), second, it can be false for both items (case d) and third/forth, it can be true for one of the items and false for the other one (cases b and c). Similarity measures for binary data mostly differ in how they weight these three cases. Two important examples are simple matching and the Jaccard measure.

Definition 1.3.5. The similarity of two items $x, y \in D$ given a set of features $\mathbf{X} \subseteq \mathcal{X}$ using *simple matching* is given as

$$sim(x, y) = \frac{n_a(x, y) + n_d(x, y)}{n_a(x, y) + n_b(x, y) + n_c(x, y) + n_d(x, y)}$$

where $n_a(x, y)$ denotes the number of features that are true for both items, $n_b(x, y)$ denotes the number of features which are true for x and false for y, $n_c(x, y)$ denotes the number of features that are false for x and true for y and $n_d(x, y)$ denotes the number of features that are false for both items.

This measure weights cases a and d equally. This is not always appropriate. Given, for instance, a set of binary features, each of which denotes which items a specific user bought from an online vendor. The fact that a user did not buy any of two products

should not render these two products similar in the same way as if she actually bought both items. Adding new items, for instance, would make all existing items more similar to each other, which is quite unintuitive. A measure that is better suited in such cases is Jaccard's coefficient.

Definition 1.3.6. The similarity of two items $x, y \in D$ given a set of features $\mathbf{X} \subseteq \mathcal{X}$ using *Jaccard's coefficient* is given as

$$sim(x, y) = \frac{n_a(x, y)}{n_a(x, y) + n_b(x, y) + n_c(x, y)}$$

where $n_a(x, y)$ denotes the number of features that are true for both items, $n_b(x, y)$ denotes the number of features which are true for x and false for y and $n_c(x, y)$ denotes the number of features that are false for x and true for y. As can be seen, case d (false-false matches) does not play any role in the result.

In many domains, we face real valued features rather than binary ones. Two of the most important measures for this kind of features are the inner product and a family of measures called Minkowski or L_m distances of which the Euclidean distance is the most prominent representative.

Definition 1.3.7. The *Minkowski or L_m distance* of two items $x, y \in D$ given a set of features $\mathbf{X} \subseteq \mathcal{X}$ is

$$d(x, y) = \left(\sum_{X \in \mathbf{X}} (X(x) - X(y))^m \right)^{\frac{1}{m}}$$

where $m > 0$ is a natural number. For $m = 2$ we obtain the Euclidean distance, for $m = 1$ we obtain the absolute distance (also referred to as Manhattan distance).

The L_m distance shares some problems with simple matching. Imagine a feature set in which each feature denotes how often a given user visited a large set of webpages. The fact that a user never visited any of two webpages should not contribute much to their similarity. A measure, for which this problem does not occur, is the inner product.

Definition 1.3.8. The similarity of two items $x, y \in D$ using the *inner product* given a set of features $\mathbf{X} \subseteq \mathcal{X}$ is

$$sim(x, y) = \sum_{X \in \mathbf{X}} X(x) \cdot X(y)$$

We assume that for all $X \in \mathbf{X}$ and all $x \in D$, $X(x) \geq 0$. This ensures that the similarity never becomes negative.

This measure can additionally be normalized by the Euclidean length of the feature vectors corresponding to x and y. The resulting measure is called *cosine measure*. It is applied in cases, in which the length of feature vectors should not play an important role. In the above example, the total number of users that visited a webpage should not have a strong influence on its similarity to other webpages.

There are many other similarity measures for binary and real valued features, for a nice survey refer to [5].

Comparing items based on nominal and mixed attributes is a more challenging task and will not be discussed here. For a discussion on this problem, refer to [191].

1.3.4 Comparing and Evaluating Similarity Measures

As discussed above, there are many different approaches to measure similarity. Furthermore, similarity also depends on the feature space that is used. Comparing scientific publications according to their topic will lead to different results than comparing them according to the year in which they were published. Both issues lead to the question of how to evaluate, compare and optimize similarity measures.

To evaluate and optimize similarity measures, usually an external criterion is needed. In recommender systems, the accuracy of the recommendations can be used as performance criterion for the underlying similarity measure. Another possibility is to use explicit constraints provided by a user in form of pairs of items that should be similar/dissimilar. We can then derive feature weights that optimize for these constraints [202]. A similar method is to use pairwise comparisons of items as constraints [158, 86].

Also, we can use a reference similarity measure as ground truth, against which we evaluate a similarity measure in question. This leads to the question of how to compare two similarity measures. Two popular approaches are based on the Pearson correlation and on the absolute distance.

Both approaches are applied to each pair of items in a finite subset of $S \subseteq D$. Two similarity measures are similar, if they produce similar values over all pairs of items $(x, y) \in S^2$.

Definition 1.3.9. The *absolute distance of two similarity measures* $sim_i : S^2 \to \mathbb{R}_0^+$ and $sim_j : S^2 \to \mathbb{R}_0^+$ with $S \subseteq D$ is given as

$$\frac{1}{|S|^2} \sum_{(x,y) \in S^2} |sim_i(x, y) - sim_j(x, y)|$$

In some cases it is desirable that this measure is tolerant to linear transformations. In this case the Pearson correlation can be used.

Definition 1.3.10. The *correlation of two similarity measures* $sim_i : S^2 \to \mathbb{R}_0^+$ and $sim_j : S^2 \to \mathbb{R}_0^+$ with $S \subseteq D$ is given as

$$\frac{\sum_{(x,y)\in S^2}(sim_i(x,y)-\overline{sim_i})(sim_j(x,y)-\overline{sim_j})}{\sigma(sim_i)\sigma(sim_j)}$$

where $\overline{sim_i}$ and $\overline{sim_j}$ are the average similarities over all pairs of items in S. $\sigma(sim_i)$ and $\sigma(sim_j)$ are the corresponding standard deviations. Thus,

$$\overline{sim_i} = \frac{1}{|S|^2}\sum_{(x,y)\in S^2} sim_i(x,y)$$

and

$$\sigma(sim_i) = \left(\frac{1}{|S|^2}\sum_{(x,y)\in S^2}(sim_i(x,y)-\overline{sim_i})^2\right)^{\frac{1}{2}}$$

and correspondingly for $\overline{sim_j}$ and $\sigma(sim_j)$. To be applicable, $\sigma(sim_i) > 0$ and $\sigma(sim_j) > 0$ must hold, thus it is not allowed that all pairwise similarities are equal in any of both measures.

Both measures can be applied to distances as well.

1.4 Cluster Structures and Taxonomies

1.4.1 Basic Formalism

While similarity is already quite useful, it is not well suited to get an overview of a whole set of items. This can be achieved much better if the items are assigned to a small number of *clusters*, such that each cluster contains items that are similar to each other [5]. If each item is assigned to *exactly* one cluster, the resulting structure is called a *partition*. A partition can be encoded using the binary similarity relation SIM, which, however, must be an equivalence relation in this case. Below, some algorithms will be presented, that create a partition based on data and an arbitrary similarity measure. Cluster analysis is mostly used to get a quick survey of a set of items, because inspecting a small number of clusters is usually easier than inspecting each item individually.

However, the problem of grouping items can also be regarded from another perspective. Instead of thinking of clusters as sets of similar items, we can regard them as concepts that group items by a common intension. A scientific topic, e.g. „Artificial Intelligence“, would be a concept that groups a set of documents together that are connected by this topic.

It general, we can assume that there is a finite set of concepts or classes C and an extension function.

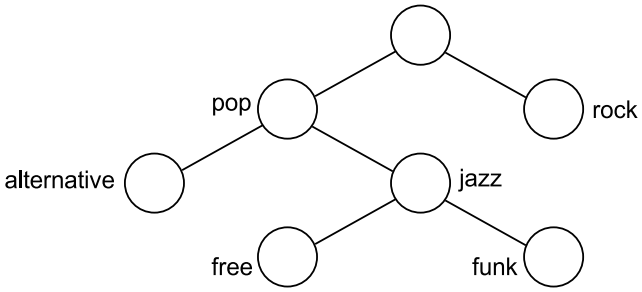


Figure 1.4.1: An example taxonomy

Definition 1.4.1. An *extension function*

$$ext : C \rightarrow 2^D$$

denotes which items in D belong to which concept $c \in C$.

This relation between item and concept is often denoted as „IS-A“ relation [150].

Definition 1.4.2. A set of concepts C is a *partition* of a set of items $S \subseteq D$ if

$$\forall x \in S, \exists c \in C : x \in ext(c)$$

and

$$\forall c, c' \in C : (S \cap ext(c) \cap ext(c') \neq \emptyset) \Rightarrow c = c'$$

thus every item in S is assigned to one concept and concepts are not allowed to overlap on S.

If items may belong to more than one concept, it is possible to impose a subset relation on them.

Definition 1.4.3. Two concepts $c, c' \in C$ are in *subconcept relation* $\prec \subseteq C^2$ if their extensions are in subset relation, thus:

$$ext(c) \subseteq ext(c') \Leftrightarrow c \prec c'$$

This relation is by definition restricted to directed acyclic graphs (DAG). Often it is further restricted to allow only tree like structures. Following the definition in [150], structures in which items are assigned to concepts using an extension function (IS-A relation) and in which concepts are arranged using the subset relation, are called *taxonomies*. Figure 1.4.1 shows an example of a taxonomy containing music.

1.4.2 Structuring Information by Taxonomies and Cluster Structures

As described above, taxonomies are used in many applications to allow users to browse information collections according to topic, genre, etc. Prominent examples are Yahoo!, the open directory, catalogues of online vendors, (digital) libraries and many others. The underlying structure is mostly a simple subset tree. In some cases the hierarchy is enriched with cross links, such as in the Yahoo! hierarchy, making it technically a DAG.

Taxonomies are much more powerful than similarity and recommendation systems, because of their ability to capture semantic. They are, for instance, well suited if users want to explore new topics. Taxonomies do not only point to relevant items, but also tell something about how a topic is structured into subtopics and how it relates to other topics. Thus, the use of taxonomies goes far beyond simple search by keywords or by similarity. Portals like Yahoo! combine keyword based search with taxonomies by returning not only relevant web pages upon a search query but also relevant concepts in the taxonomy.

Taxonomies, as described so far, are *global*. There is one taxonomy that should cover all items (e.g. all webpages) and that is shared by all users. The other extreme are bookmarks and directory structures. They represent taxonomies created by an individual user to cover exactly the personal item collection of the user. It is neither clear whether this structure would be valid for other items, nor if other users would share it. In this sense it is *local*. Such local, personal taxonomies are usually much smaller than global ones. Users employ their personal taxonomies not only for search but also for organizing items. They insert, for instance, new webpages into their bookmark tree, they create new nodes, rearrange the tree, etc. In a metaphorical sense one could say they furnish their personal information space, just like they would furnish their apartment.

Both global and local taxonomies are usually created and maintained manually, which is very unsatisfying. There are several approaches to create them dynamically from data. This is achieved by clustering methods. The most prominent application of clustering to interactive information retrieval is probably the scatter and gather approach [40]. A set of documents is clustered automatically into a partition. The user then chooses the most promising partition. Then the items in this partition are clustered. This process continues until there are only few items left, which the user can inspect manually.

There are also many approaches that automatically classify new items into an existing, fixed concept structure. Given, for instance, a set of music genres, audio classification methods annotate new music files automatically with genre information [68, 109]. This allows users to access so far unstructured items using a known class structure. A similar approach is to learn preferences of a user. Preferences can be represented as two concepts, one containing the items the user likes another one the items the user dislikes. One of the first approaches that applied this principle to Internet resources was the Letizia system [110]. Another, more sophisticated approach to support for browsing pages based on topics is WebWatcher [87]. It allows users to specify a topic and then guides them through the links on a web page.

Finally, there are hybrid systems which classify new items into an existing class structure only if they fit to one of the classes. If this is not the case, a new cluster is created. They are often referred to as incremental clustering systems. This approach is followed, e.g., by the FOCI system [138] or by the Newsgroup clustering system co-developed by the author and described in [74].

Most methods developed in this work are based on taxonomies and cluster structures, because these a very popular and widely used formalisms for structuring information, both at a global and at a local level. On the one hand, they provide more semantic information than simpler approaches based on similarity and recommendations. On the other hand, they are simple enough to be managed by casual users. Finally, there are many methods that allow to create and maintain such structures (semi-)automatically, some of which will be described in the next sections.

1.4.3 Creating Cluster Structures from Data

Taxonomies used on the Internet are often hand crafted. Usually, a very small team of people designs a concept hierarchy and assigns new items to these concepts manually, as they arrive. Faced with the vast amount of information on the Internet, this is not satisfying.

Creating cluster structures automatically from data is a well known task from explorative data analysis. We assume the subset of items to be clustered is $S \subseteq D$.

Definition 1.4.4. The task of *clustering* is to yield a finite set C of clusters, such that $\bigcup_{c \in C} ext(c) = S$, where S is a given subset of D.

Algorithms differ first of all in the kind of output they produce. As described above, there are basically three possibilities, the items in S can be partitioned, they can be grouped into a set of overlapping clusters or into a hierarchical cluster structure, which is a special case of overlapping clusters.

A second important question is what is regarded as a cluster. This is usually expressed by a set of constraints or by an objective function denoting a valid and optimal cluster structure. In the following, we will present some of the most important problem definitions, partially together with algorithms that solve them.

Definition 1.4.5. A *cluster quality function* is a function that assigns to each set of clusters C a numerical quantity that expresses the quality of the clustering concerning a given set of items $S \subseteq D$.

$$cq : \mathcal{C}_S \rightarrow \mathbb{R}$$

where $\mathcal{C}_S$ is the set of all possible cluster structures on S.

Often, clusters are defined in terms of pairwise distance. A basic approach is to minimize the mutual distance among items in the same cluster and maximize the distance between items in different clusters.

Definition 1.4.6. The *average within cluster distance* is defined as

$$cq_{wcd}(C) = \sum_{c \in C} \sum_{x \in ext(c)} \sum_{x' \in ext(c)} d(x, x')$$

where C is a fixed set of clusters, ext a corresponding extension function and d a distance measure.

The task is to find such an assignment of items to clusters that cq_{wcd} is minimized. It can be shown that by minimizing cq_{wcd} the mean pairwise difference between pairs of points in different clusters is maximized [73]. If the underlying distance measure is Euclidean distance, there is an efficient optimization strategy for this problem. The k-means algorithm, depicted in figure 1.4.2, applies an alternating optimization procedure [72]. For each cluster, a centroid is calculated, the point with the smallest average distance to all items in the cluster. Then every data point is assigned to the cluster with the nearest centroid. After this step, centroids are recalculated for each cluster and data points are newly assigned to clusters based on the new centroids. This alternation stops if there is no further change in cluster assignment or after a maximal number of steps. The centroids are initialized with random items. k-means does not guarantee to find an optimal solution and is sensitive to the choice of initial points. Therefore, a common strategy is to start k-means several times with different random initializations. For its simplicity and efficiency, k-means is one of the most popular clustering algorithms.

The k-means algorithm works only with Euclidean distance. Recent work shows that a highly similar optimization strategy works for a family of distance measures called Bregman divergences [12]. Another generalization of k-means is k-medoids, for which any similarity or distance measure can be used [73]. The idea is to select an item in each step, with the smallest average distance to all points in the cluster. This item then serves as centroid. Related to this approach are extensions of k-means using kernel methods [207, 44]. However, both, kernel k-means and k-medoids, suffer from the problem that they must compute similarities for all pairs of items in each step, while calculating centroids with k-means is linear in the number of items.

While k-means does only produce partitions in its base version, it can be easily extended to yield hierarchical structures as well [73]. The idea is to split the set of items to be clustered in a top-down manner. First, the set S is partitioned into k subsets. Then k-means is applied to each of these subsets. This procedure continues until a minimal cluster size is reached. This approach is very efficient. It shows good results, e.g. for hierarchical document clustering [167].

Approaches, such as k-means, model clusters as sets of items with minimal pairwise distance. This can be expressed in another way by assuming that items are the visible output of several different hidden processes. Items are grouped together if they are assumed to have been generated by the same hidden process. The optimization works similar as for k-means. In each step, parameters of the underlying distribution are

Input:
A set of k empty clusters C
A set of items S
A set of features $\mathbf{X}$

Output:
An extension function ext that assigns the items in S to clusters in C
A centroid $z_j \in \mathbb{R}^{|\mathbf{X}|}$ for each cluster

for $c_j \in C$ **do**
  Initialize $z_j \in \mathbb{R}^{|\mathbf{X}|}$ with random values
end for
$round = 0$;
while $round < maxRounds$ **do**
  for $x \in S$ **do**
    $l = argmin_j(\sum_{i=1}^{|\mathbf{X}|}(z_{ji} - X_i(x))^2)$;
    Assign x to c_l, thus $x \in ext(c_l)$;
  end for
  for $c_j \in C$ **do**
    for $X_i \in \mathbf{X}$ **do**
      $z_{ji} = \frac{1}{|ext(c_j)|} \sum_{x \in ext(c_j)} X_i(x)$
    end for
  end for
  $round = round + 1$;
end while

Figure 1.4.2: The k-means algorithm: Given a set of features and a number of k clusters, k-means reassigns items to clusters by alternatingly calculating the cendroid of each cluster (where z_{ji} denotes the value of feature X_i for the centroid corresponding to c_j), and assigning items to the cluster with nearest cendroid.

estimated for each cluster based on the items currently assigned to the cluster. Then items are reassigned to the cluster with the distribution that most likely produced them. This approach is denoted as expectation maximization clustering, named after the more general EM-optimization strategy [43].

Another view of the clustering problem is to assume that items are vertices in a graph. Two items are connected if they are sufficiently similar. The task of clustering is then to find connected components in the similarity graph. In this case, not the number of clusters is fixed, but a threshold that defines whether two items are connected and thus in the same cluster. This strategy is denoted as single link clustering [5]. Often it is used to produce a cluster tree instead of an item partition. The idea is to start with a clustering in which each item has its own cluster. Then clusters are consecutively merged, always

selecting the two most similar clusters. The similarity of clusters is assessed for single link clustering as the smallest distance between an item in one cluster to an item in the other cluster. In each step, the number of connected compounds is decreased by one, until all items are in the same cluster.

Single link clustering suffers from the problem of outliers. One faulty item is sufficient to join two otherwise clearly separated clusters. Obviously, this is not desirable. Density based clustering algorithms solve this problem by using another cluster definition. Two items are in one cluster, if they are connected by a dense area [52]. A dense area is basically one, that contains a minimum of data points per volume unit. There are also more sophisticated estimates of density, such as used for Support Vector Clustering [18], though the basic idea is the same. The benefit of density based approaches is that they are usually robust to outliers and are able to find clusters of any shape. A problem shared with pairwise similarity approaches are clusters of different density. Optics [9] is an approach to allow interactive density based clustering as a reaction to this problem.

Single-link clustering and density based approaches search for connected compounds in graphs. They differ in what is regarded as connection between two items. Another way to use an underlying graph of connected items is graph separation. Each edge in the graph has a weight that corresponds to the similarity of the items that it connects. The problem of how to find k clusters is then mapped to the problem of partitioning a graph into k subgraphs by deleting the set of edges that has a minimal sum of weights. Instead of usual graphs, often hypergraphs are used [71]. Hypergraphs may connect two items with more than one edge.

All of the above approaches differ in how they define the concept „cluster“ and what is regarded as correct or optimal clustering. They also differ in whether the search is performed bottom-up, top-down or by re-arranging items iteratively. Furthermore, they differ in an additional point which is more subtle and deserves some discussion: the representation of clusters. The representation of clusters determines which clusters can be found and is strongly connected to the definition of clusters, although not necessarily determined by it.

A common way of representing clusters, especially for pairwise similarity approaches, is by sets of points. A cluster can be represented by one point (the cluster centroid or another data item). This is the approach of k-means clustering and of its variants. A problem with this approach, at least if used with unweighted Euclidean distance, is that clusters have to be separable by spherical decision boundaries.

Another possibility is to represent a cluster by all its data points. Single link clustering applies this representation. A major drawback of this approach is the high computational effort to store all points and its sensitivity to outliers. An interesting alternative is to represent clusters by several well scattered points [66]. Such points capture, on the one hand, cluster boundaries even if they are non-spherical. On the other hand, they are less sensitive to outliers. A very similar approach are the „specific core points“ as applied in distributed density-based clustering [82].

Grid based clustering algorithms [188] represent clusters by a set of cells. These cells can be further aggregated and described in terms of attribute values by forming the maximal hyper rectangle that contains all the points. Finally, clustering algorithms working on probabilities and hidden concepts describe clusters by probability distributions or parameters of these distributions.

1.4.4 Classifying Items into a given Cluster Structure

Clustering imposes a new structure on a set of previously unstructured items by assigning them to groups. These groups are generated during the clustering process. Classification, on the other hand, assumes a given, fixed set of classes to which new items are assigned [125]. This is usually achieved by providing examples for each of the classes. Examples are items that have been tagged with the information to which class they belong. A typical scenario are music files which are tagged by a user according to whether she likes them or dislikes them. The task of classification is then to decide for new music files whether the user will probably like them, or not. This kind of learning is called supervised learning, because the algorithm is provided with training data. Clustering on the other hand is unsupervised, because it does not provide this kind of information.

Methods for classification differ significantly from those used for clustering. In clustering, the aim is usually to get a survey of the data. For classification, the aim is encoded into the training examples. In the example above, the aim is to find out which music the user could like. If the user had annotated the items according to genre instead, it would be the task of genre classification.

Definition 1.4.7. Given a finite set of classes C, the task of *classification* is to find an optimal function $h : \mathbb{R}^k \to C$ assuming a given function $f : \mathbb{R}^k \to C$ of true values and a set of examples, where each example has the form $(v_1, ...v_k, c)$ with $v_i \in \mathbb{R}$ and $c \in C$. A function h is optimal, if it minimizes the expected loss regarding the true concept f.

For a finite set C, which we assume here, and binary features, the expected loss can be formalized as follows.

Definition 1.4.8. The *expected loss* of a hypothesis $h : \{0,1\}^k \to C$ concerning a true concept $f : \{0,1\}^k \to C$ and a finite set of classes C is defined as

$$E(L) = \sum_{v \in \{0,1\}^k} L(h(v), f(v)) \cdot p(v)$$

where $L(a, b) = 0$, if $a = b$ and 1 otherwise. $p(v)$ is the probability by which the datapoint $v \in \{0,1\}^k$ is drawn from an underlying distribution.

This definition can be easily extended to continuous features as well. If the set C is binary, then we will write f as logical term, for sake of simplicity.

Given, for instance, the binary function $f(X_a, X_b) \equiv X_a + X_b > 5$. Labeled examples of this function could be $\{(1, 2, false), (2, 6, true)\}$. Based on these examples, we could induce, the not quite correct, hypothesis $h(X_a, X_b) \equiv X_b > 2$.

How is the problem of classification related to clusters and items? We again assume a fixed set of non-overlapping concepts (a partition) C. Also we assume a function $f : D \to C$ that assigns each element in D to exactly one concept in C. The function f can usually not be observed directly. Rather we know the value of f only for a subset $S \subseteq D$ of items, the examples. This can be written as a function $f' : S \to C$. The set S could be, for instance, a set of items already tagged by a user according to taste. The task of item classification is to yield a hypothesis that classifies the remaining items in D according to this concept.

Definition 1.4.9. The aim of *item classification* is to find a function $h : D \to C$ that approximates the true concept f given some examples $S \subseteq D$ with $f' : S \to C$ in such a way that expected loss is minimized.

We assume here that C is a finite set. Then, the expected loss can be rewritten for item classification as follows.

Definition 1.4.10. The *expected loss for item classification* of a hypothesis h concerning a true concept f is defined as

$$E(L) = \sum_{x \in D} L(h(x), f(x)) \cdot p(x)$$

where $L(a, b) = 0$, if $a = b$ and 1 otherwise. $p(x)$ is the probability that item x is drawn from an underlying distribution.

The expected loss respects the distribution of the data. Misclassifying an item that is likely to occur is more severe than misclassifying an unlikely item.

Given a set of features $\mathbf{X}$, then a function $h : D \to C$ can be defined based on a traditional classification function, by

$h(x) = h'(X_1(x), .., X_i(x), .., X_{|\mathbf{X}|}(x))$ with $X_i \in \mathbf{X}$ and $h' : \mathbb{R}^{|\mathbf{X}|} \to C$ a classification function. Similarly, given $f' : S \to C$, we can easily transform this into examples $\{(X_1(x), .., X_i(x), .., X_{|\mathbf{X}|}(x), f'(x)) | x \in S\}$.

Usually, the distribution of the items and the true label are not known. Therefore, the expected error can only be estimated. A usual assumption is that the items in S obey the same distribution as the items in D. Under this assumption, the expected loss can be estimated by cross validation [73]. The idea is to subsequently delete examples $S' \subseteq S$ from S. The remainder of examples is used to generate a hypothesis h. Then h and f' are compared on the set S'. This process is repeated until each example in S was used for testing exactly once. The size of S' is called batch size and can be varied.

There are many different algorithms to create hypotheses or models given labeled examples. A very good survey can be found in [73] or in [125]. In the following, we will mostly use a simple but very flexible and powerful classification algorithm, namely *nearest neighbor*. This algorithm assumes a similarity function $sim : D^2 \rightarrow \mathbb{R}_0^+$ defining the pairwise similarity of all items to each other. If a new item is to be classified, the algorithm first looks for the k most similar items in the training set S. Then, a majority vote between the members of this set is performed, concerning the class label. The class label with the highest count is used as prediction. Ties are broken randomly. There are many variants and optimizations of this basic algorithm [192].

Until now, we assumed a partition of items. The approach can be extended to cover concept hierarchies as well. The problem of classifying items into a concept hierarchy is denoted as *hierarchical classification*. An obvious problem is that an item is in general not assigned to a single concept but to several ones. A first possibility to map this problem to a traditional, flat classification problem is to only regard the most specific concept to which an item is assigned. In this case, the function h, as well, should predict only this most specific concept. Evaluation and training are performed based on the most specific concept. Basically, all concepts are regarded as independent by this approach. The hierarchical structure among them is ignored.

Regarding all concepts independently can be suboptimal, especially if the number of leaf nodes is large. A better possibility to map hierarchical to flat classification is not to train a single classifier, but to train a classifier for each inner node in the concept hierarchy. Such a classifier then decides into which of the subtrees an item belongs. Items are classified in a top-down manner, thus they are inserted at the root. Then, at each step a classifier is used to decide how to propagate them down the tree, until a leaf node is reached. Using a classifier in each node has the advantage of dealing with much fewer classes in each individual classifier. For each classifier more examples are available, because for each class all items in the corresponding subtree serve as examples.

A second point that is not satisfying in regarding all concepts independently is evaluation. We usually would like to make a distinction between small mistakes (e.g. an item is falsely assigned to a sister or daughter concept) and a large mistakes (e.g. an item is falsely classified to a completely different subtree). This notion can be operationalized by, e.g., modeling the loss not binary but rather as tree distance between the true and the predicted concept.

Finally, there is still another possibility for evaluation. We can estimate the expected loss for each inner node (and its associated classifier) separately and then average all of these values. The idea behind this concept is that we often do not input items only at the root but at arbitrary inner nodes. A user can, for example, assign an item to the genre „rock“ and leave it to the algorithm into which sub genre to put the item. Therefore, all classifiers should be regarded equally important. Using tree distance, classifiers closer to the root of the concept hierarchy tend to obtain more weight.

At least equally important as the classification algorithm and its parameters is the set of features used. As discussed above, similarity can be defined under different aspects

represented by subsets of the features used to describe the items. Finding an optimal set of features is therefore a major challenge for automatic classification. This problem will be discussed in detail in chapter 4.

1.4.5 Comparing and Evaluating (Hierarchical) Cluster Structures

Comparing and evaluating cluster structures is important for several reasons. Especially, it allows to select optimal solutions in terms of the clustering algorithm, its parameters and the feature space. For classification, such an evaluation could be achieved by using cross validation to estimate the expected loss. For clustering, the situation is more complicated.

There is a distinction between *internal* and *external evaluation measures* for cluster structures. Internal measures can be derived from the result an algorithm produces. A typical example is the average within cluster distance cq_{wcd}, as used by k-means clustering.

Internal quality measures have the disadvantage that they are not well suited to optimize a clustering by parameter optimization and feature selection. Trivial choices (using only one nominal feature) often lead to optimal results. They are, however, almost certainly not intended by the user. The same holds for parameters, such as k in k-means. Using the number of items in S as value for k leads to the optimal result of $cq_{wcd} = 0$. A second drawback of internal evaluation measures is that they often do not allow to compare two different clustering schemes. The measures used by density based methods are, for example, very different from the ones used in single link clustering.

An alternative is to use external criteria to evaluate cluster structures. The simplest and most prominent method is to provide a reference clustering as ground truth. Cluster structures can then be evaluated by comparing them to this reference structure.

There is a large amount of research on how to compare two partitions of items. Most approaches are based on the notion of concordance. Two items assigned to the same cluster in the first partition should be in the same cluster in the second partition as well. Methods mostly differ in how they weight different forms of concordance and discordance. [117] provides an axiomatic characterization of desirable properties of functions comparing partitions and gives an impossibility result.

In this work, the focus is on comparing taxonomies and thus hierarchical cluster structures. This problem is considerably more complicated. In the following the most important approaches to compare taxonomies will be presented.

Theoretical computer science has contributed a lot of work on comparing tree structures. As hierarchical cluster models can be regarded basically as trees, such measures can be applied to compare them as well. The most important measure for structural tree similarity is tree edit distance. The similarity of two trees is derived by measuring the minimal number of steps (or more general costs) necessary to transfer one tree into the other one by deleting and creating nodes. The computational effort to compute the tree

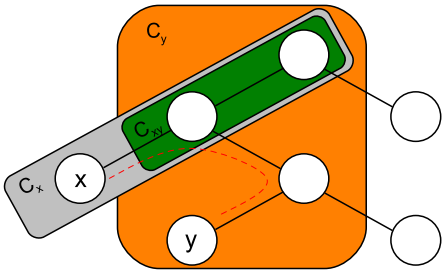


Figure 1.4.3: The treedistance of two items x and y. C_x contains all concepts that contain x, C_y contains all concepts that contain y and C_{xy} contains all concepts that contain both.

edit distance is quite high, especially, because items must be modeled as leaf nodes of the tree in this formalism. [22] gives a nice overview of different algorithms that calculate the tree edit distance.

As stated above, clusters and cluster structures are often defined by some form of similarity: clusters should contain similar items. This view can easily be reversed. Given a cluster structure, it is possible to define a similarity of items according to this cluster structure. The most popular measure of this kind is tree distance. The tree distance of two clusters is the length of the shortest path connecting them. The tree distance of two items is the tree distance of the most specific nodes they are associated with.

Definition 1.4.11. Given a cluster tree C, the *tree distance* of two items $x, y \in S$ is

$$d(x, y) = |C_x| + |C_y| - 2|C_{xy}|$$

where $C_x = \{c \in C \mid x \in ext(c)\}$ is the set of clusters containing item x, $C_y = \{c \in C \mid y \in ext(c)\}$ is the set of clusters containing item y and $C_{xy} = C_x \cap C_y$ the set of clusters containing both items. The set $S \subseteq D$ here denotes all items that are in the extension of any of the clusters in C, thus $S = \{x \in D | x \in ext(c), c \in C\}$.

Figure 1.4.3 illustrates the idea.

Two clustering can be compared by comparing the corresponding tree distance measures that are generated by each of them. The problem of comparing two similarity measures was described above. In this case, the correlation of tree distances is especially well suited, because more detailed cluster structures can still be similar to less detailed ones, as long as this can be captured as linear transformation of the tree distance.

Another popular measure to compare two cluster structures is f-measure, which often used in information retrieval and machine learning. In the following we will use f-measure to compare two individual clusters $c \in C$ and $c' \in C'$, where C and C' are two cluster structures. Given a query set of items and a result set of items (e.g. results delivered by a search engine and truly relevant items) precision denotes the fraction of items in the result set that are relevant.

Definition 1.4.12. The *precision* of a cluster $c \in C$ concerning another cluster $c' \in C'$ is

$$pr(c, c') = \frac{|ext(c) \cap ext(c')|}{|ext(c)|}$$

for $|ext(c)| > 0$.

Recall is the fraction of relevant items that are contained in the result set.

Definition 1.4.13. The *recall* of a cluster $c \in C$ concerning another cluster $c' \in C'$ is

$$re(c, c') = \frac{|ext(c) \cap ext(c')|}{|ext(c')|}$$

for $|ext(c')| > 0$.

F-measure is a (weighted) combination of both.

Definition 1.4.14. The *f-measure* of a cluster $c \in C$ concerning another cluster $c' \in C'$ is

$$fm(c, c') = \begin{cases} \frac{2 \cdot pr(c,c') \cdot re(c,c')}{pr(c,c') + re(c,c')} & \text{, if } pr(c, c') + re(c, c') > 0 \\ 0 & \text{, else} \end{cases}.$$

Two cluster structures are can be compared as follows [167]. For each cluster $c \in C$ in the first cluster structure C we search for the cluster $c' \in C'$ in the second cluster structure C' that has the highest f-measure concerning c. Thus, we search for each cluster its best match in the reference clustering. The similarity of two cluster structures is then the average f-measure over all clusters in C. Additionally, these clusters can be weighted by the number of items they contain. The overall measure is then defined as follows:

Definition 1.4.15. The *FScore* of a clustering C concerning a reference clustering C' is defined as

$$FScore(C, C') = \frac{1}{\alpha} \sum_{c \in C} |ext(c)| \cdot max_{c' \in C'} fm(c, c')$$

with

$$\alpha = \sum_{c \in C} |ext(c)|$$

and $fm(c, c')$ the f-measure between c and c', as defined above.

While this measure is asymmetric in general, it can easily be made symmetric by switching the roles of both cluster structures and using the average as result. This symmetric version of the FScore will be used in the empirical evaluation.

A popular measure to compare partitions of items is mutual information.

Definition 1.4.16. Given two partitions C and C' with respect to a common set of items $S \subseteq D$, the *mutual information* among both is defined as

$$\mathcal{I}(C, C') = \sum_{c \in C} \sum_{c' \in C'} \frac{|ext(c) \cap ext(c')|}{|S|} \log \frac{|ext(c) \cap ext(c')||S|}{|ext(c)||ext(c')|}.$$

While it can be extended to hierarchical structures as well, this step is not trivial and only applicable if all items reside in leaf nodes. In general this is not the case. Therefore this measure will not be used in the following.

1.4.6 Merging Cluster Structures

Classifier ensembles are one of the most important techniques in supervised learning [73]. The idea is to train several classifiers using different feature sets or different subsets of the training data. Then the individual classifiers are combined into a final classification function. Classifiers ensembles have been shown to be very accurate and robust to noisy data [73]. There are several attempts to transfer this success to unsupervised learning. Corresponding techniques are denoted as *cluster ensembles*. To date, research is only concerned with flat cluster ensembles and especially with partition ensembles. Another common assumption of all approaches is that all partitions cover the same set of items.

Several approaches were proposed to merge partitions. The most simple one uses the co-association of items in the given partitions to derive a binary similarity measure which is then used together with a traditional, similarity based clustering algorithm. The major advantage of this algorithm is its simplicity and the ability to plug it into any state of the art clustering algorithm. Empirical results suggest that it works very well on different problems [182]. A major drawback is the consumption of storage space, because an $|S|^2$ matrix has to be created as input to the clustering procedure. Another general approach is to search for a median partition among the input partitions, thus a partition that has a maximum average similarity to all other partitions.

Another possibility is to formulate the search for a common clustering as optimization problem.

Definition 1.4.17. Given a set of partitions $\Phi = \{C_1, .., C_i, .., C_k\}$ on a common set of of items $S \subseteq D$, find a *median partition* C_{avg}, that maximizes

$$\sum_{C \in \Phi} \mathcal{I}(C, C_{avg}).$$

The idea is to find the clustering that shares a maximum of information with all input cluster structures Φ.

[171] propose a hypergraph based algorithm to solve this problem. First, a hypergraph is generated from the input partitions. This hypergraph contains an edge between two items for each concept they both are assigned to. This hypergraph can then be clustered using a graph clustering algorithm, as described above.

While merging cluster structures is a very promising approach, the basic assumption that all input cluster structures as well as the output cluster structure must be global limit its applicability to clustering local item collections of users. Also, the limitation to flat structures makes them hard to apply in practical scenarios that often rather contain hierarchical structures. In chapter 5, we will propose an intuitive extension to flat cluster merging, called hierarchical cluster ensembles. This methods is global in the sense that all input cluster structures are assumed to cover all items. Also, it delivers only a single solution, which we will show is not appropriate for information structuring that naturally implies several, heterogeneous views on a domain. In chapter 6, another approach is developed that is local in this sense, called localized alternative cluster ensembles.

1.4.7 Non-redundant Clustering

A recent approach to extend the traditional clustering problem is non-redundant clustering [65]. The idea is to find a clustering that, on the one hand, is valid but, on the other hand, is dissimilar to a given, existing clustering.

Definition 1.4.18. Given a partition C on S the aim of *non-redundant clustering* is to find a partition C' on S such that the mutual information among both $\mathcal{I}(C, C')$ is minimized while a clustering criterion $cq(C')$ on the new clustering is maximized.

In [65] both criteria are combined into one criterion. The task is then solved by using an EM-like approach. Non-redundant clustering can be used to find several orthogonal structures in the data.

1.4.8 Feature Selection for Clustering and Subspace Clustering

As for similarities, a fundamental question for clustering is which features to use. For descriptive or explorative data analysis, there usually is no external performance criterion that could be used to answer this question. The aim is rather to describe the data, as it is. There are two approaches to perform feature selection for clustering despite these limitations. The first is called subspace clustering [1]. The idea is to identify clusters in subspaces of the data. Thus each cluster is not only defined by the items it contains, but also by the subset of features for which it is a valid cluster. The validity of clusters is computed based on density. The approach makes use of the fact that the density decreases monotonically as features are added.

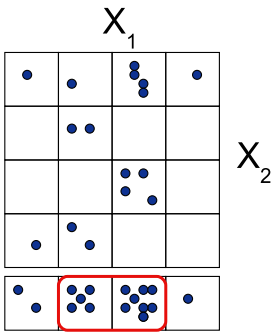


Figure 1.4.4: Subspace clustering. There is no unit that contains more than four items in the original space $\mathbf{X} = \{X_1, X_2\}$. Projecting units to the space $\mathbf{X}' = \{X_1\}$, there are two units that pass this threshold. These two units can be combined into a pattern.

The idea of subspace clustering is not to deliver a partition of items, but a set of possibly overlapping, maximal clusters.

Definition 1.4.19. Given a set of features $\mathbf{X} = \{X_1, .., X_i, .., X_n\}$, $n = |\mathbf{X}|$. A grid on this set of features is defined by segmenting the values of each feature in equal intervals. A *unit* is an intersection of intervals from each attribute, $\{[v_{r_1}, v'_{r_1}), .., [v_{r_i}, v'_{r_i}), .., [v_{r_k}, v'_{r_k})\}$, such that $v_{r_i}, v'_{r_i} \in \mathbb{R}$ and $1 \leq r_1 < .. < r_i < .. < r_k \leq |\mathbf{X}|$ and $k \leq n$ with $r_i \in \mathbb{N}$.

This leads to the notion of an item being contained in a unit.

Definition 1.4.20. An item $x \in S$ is *contained in a unit*, iff for all $1 \leq i \leq k$: $v_{r_i} \leq X_{r_i}(x) < v'_{r_i}$.

Subspace clustering aims to find maximal units, that contain more than a given fraction of the total number of items in S. Figure 1.4.4 shows an example.

An example output of a subspace cluster algorithm would be, for instance, $\{(1 \leq X_1 < 3) \wedge (2 \leq X_5 < 4), (1 \leq X_1 < 6)\}$. Note, that both clusters overlap and that not all features are used for clustering.

If the features are all binary, subspace clustering degenerates to the task of frequent itemset mining. A unit in this case has the form $\{X_{r_1} \wedge ..X_{r_i} \wedge ..X_{r_k}\}$ (if no negation is allowed), which can be seen as a frequent itemset [2]. The minimal density is in this case the exact support.

A highly similar approach is clustering based on frequent itemsets [16, 62, 187]. The idea is to first find combinations of features that co-occur very often. These sets are then arranged according to the lattice underlying them. This lattice is pruned by diverse heuristics to yield a tree. The number of clusters produced by both approaches can, however, be quite high, just as the number of frequent itemsets. This makes these clusters hard to interpret and to use for explorative data analysis.

Another approach is multi-objective optimization to find an appropriate set of features. Earlier works on this topic try to minimize the number of features while minimizing the average within cluster distance cq_{wcd} for k-means clustering [94, 95, 129]. In [122] we show that this is not appropriate. Instead we propose an approach that maximizes the number of features while minimizing cq_{wcd}. Thus, on the one hand, we want to obtain dense clusters, on the other hand we want the data to be described as completely as possible. Multi-objective optimization allows us to explore this tradeoff by means of a set of Pareto optimal solutions. In contrast to subspace clustering, this methods delivers several global results, in the sense that each cluster structure in the Pareto set covers all items. As will be shown in chapter 6, both are not well suited for clustering items locally. Subspace clustering does not deliver closed cluster structures but a set of clusters that is not easy to overview for the user. Multi-objective feature selection for clustering does not deliver results that are local. We will therefore propose a new method for feature selection in chapter 6 that is local, but still delivers several complete, sound cluster structures.

1.4.9 Other Variants of the Traditional Clustering Task

In some applications, items are described by features representing several different aspects. A typical example are web pages described by their content and by the link structure connecting them. Co-clustering exploits this input data by clustering the data items alternately according to each aspect. Corresponding algorithms, such as Co-EM [21], are reported to yield superior results in some areas. Another possibility to improve traditional clustering is to cluster the features simultaneously with the items. This can be especially beneficial if there are many, highly correlated features. In text clustering, for instance, terms can be clustered into prototypical topics, which again helps to cluster the documents, etc. Typical algorithms include latent semantic indexing [42] and its probabilistic counterpart [75], as well as the more general multi-way clustering [17].

Semi-supervised or constraint clustering algorithms allow users to pose constraints on the resulting cluster structure [39]. Constraints state, for instance, that two items must be assigned to the same cluster or that they cannot be assigned to the same cluster.

Definition 1.4.21. A *must-link constraint* for two items $x, y \in S$ with $S \subseteq D$ states that the clustering algorithm must deliver a partition C such that $\exists c \in C : x \in ext(c) \wedge y \in ext(c)$.

In a similar way, a cannot-link constraint can be defined.

Definition 1.4.22. A *cannot-link constraint* for two items $x, y \in S$ with $S \subseteq D$ states that the clustering algorithm must deliver a partition C such that $\neg\exists c \in C : x \in ext(c) \wedge y \in ext(c)$.

There are two general approaches to semi-supervised clustering. The first one is to learn a distance function that fulfills the constraints as good as possible and then to apply this distance function with a traditional clustering algorithm [202]. The resulting clustering

is not guaranteed to fulfill all constraints. The second approach is to incorporate the constraints directly into the clustering algorithm [186].

Other possible constraints include, e.g., that all items with different nominal values for a given feature must be assigned to different clusters. Also constraints on the size of the resulting clusters are possible.

A variant of semi-supervised clustering is supervised clustering [56]. The user provides a cluster structure on a small subset of items which is then used to cluster the resulting items.

Beside constraints, other kinds of background knowledge can be used to guide the clustering process. A typical example is presented in [80]. A text clustering algorithm is improved by enriching the original text documents with information from a thesaurus.

Incremental clustering refers to the task of clustering a stream of items, thus to adapt the cluster structure to new items automatically. A very simple method is to check whether a new item is sufficiently similar to one that is already assigned to an existing cluster. If this is the case, it is assigned to this cluster, otherwise a new cluster is created that contains only this item [81]. A more sophisticated approach can be found in [33]. Both approaches adopt a mostly data centric view, thus they try to achieve a high quality of clusters without performing the overall clustering process anew when items are added. In [74] we propose a user-centric view on the problem. If a cluster structure is employed for navigation by users, then each change to this structure is a critical step, because users must accommodate to the new, modified structure. Therefore, users prefer that their navigation structures are only altered if this is absolutely necessary. This leads to the interesting question of how to change a cluster structure minimally from a user's point of view, while preserving a high clustering quality.

1.5 Formal Ontologies

1.5.1 Basic formalism

Cluster structures can be further refined in several ways. First, directed relations between items and concepts of any type and any arity can be allowed. Especially important is, for instance, the support of part-of relationships, describing how an entity is internally structured (e.g. how the parts of a car form the whole car). Second, concepts may not only be described extensionally but also intensionally. For example, we can explicitly define a concept „bachelor“ as set of entities that are male and unmarried. The extension of this set is then implicitly inferred from these properties, instead of being explicitly assigned by a clustering or classification algorithm.

An early approach for allowing different relations between items and concepts are conceptual networks [164]. They consist of nodes, representing items, and of edges, representing relations among these items. There are many variants of such networks. A fundamental

problem with these early approaches is a rather unclear semantic [194]. Also, relations with arity more than two are not easily representable in such networks.

Description Logics (DL) consolidates much of the research on conceptual networks and provides a well defined semantic [11]. DL are a family of formalisms based on first order logic. The expressiveness of DL is usually restricted in order to make them (efficiently) decidable. Members of the DL family mostly differ in how they are positioned in the tradeoff between expressiveness and worst case complexity of inference.

In the following, we give a short sketch of a very simple description logic (following mostly the formalism and examples in [11]).

DL distinguish a TBox and an ABox. The ABox contains assertion about entities in the domain of interest, the TBox contains assertion about concepts.

The TBox contains expressions in a language denoted as $\mathcal{AL}$. The members of this language can be defined recursively. The language $\mathcal{AL}$ may contain a finite set of atomic concept expressions, of relation expressions and the symbols $\bot$ and $\top$ as terminal symbols.

Assume F, G to be compound concept expressions, A to be an atomic concept expression and R to be a relation expression in $\mathcal{AL}$. Then the following expressions are also in $\mathcal{AL}$:

$$\neg A | F \sqcap G | \forall R.G | \exists R.\top$$

$\bot$ denotes the empty concept, to which no entity is assigned, $\top$ denotes the universal concept, that contains all entities. A is a atomic concept and $\neg A$ is its negation (all entities not in A). $F \sqcap G$ is the set of entities contained in F and G. $\forall R.G$ denotes the entities to which all entities in relation R are in G. $\exists R.\top$ denotes the set of entities that are in relation R with at least one other entity. Formally, this can be expressed as follows.

Definition 1.5.1. Let Δ be a set that contains all entities of interest ($\Delta = D$, for instance). Then an *interpretation function* $ext : \mathcal{AL} \rightarrow 2^{\Delta}$ is defined as follows:

$ext(\bot) = \emptyset$

$ext(\top) = \Delta$

$ext(\neg A) = \Delta \backslash ext(A)$

$ext(F \sqcap G) = ext(F) \cap ext(G)$

$ext(\forall R.G) = \{x \in \Delta | \forall y \in \Delta : (x, y) \in ext(R) \rightarrow y \in G\}$

$ext(\exists R.\top) = \{x \in \Delta | \exists y \in \Delta : (x, y) \in ext(R)\}$

The extensions of all atomic concepts and of all relations are mapped directly by the extension function, just as described above for the simpler case of taxonomies.

Based on these expressions, terminological constraints can be stated.

If we assume, for instance, an atomic concept $Person$, then we can define $Parent \equiv Person \sqcap \exists hasChild.\top$. In this case, trivially, the terminological constraint $Parent \sqsubseteq Person$ holds, defining that $ext(Parent) \subseteq ext(Person)$.

The ABox contains assertions about individual entities, for instance $Person(peter)$ or $hasChild(peter, mary)$.

Based on a TBox, the consistency of the ABox can be checked. Also, it is possible to derive entailed assertions in the ABox, for instance, it would be possible to derive $Person(peter)$ from $Parent(peter)$.

The DL described above is called $\mathcal{AL}$. There are many extensions that allow more powerful expressions in the TBox. A DL with high expressiveness is, for instance, $\mathcal{SHOIN}(\mathcal{D})$ [11].

DL extend simple taxonomies in two ways. First, it is possible to state relations among entities. Second, terminological constraints can be stated. Simple taxonomies and cluster structures can therefore be regarded as ABox without assertions about relations.

1.5.2 Structuring Information by Formal Ontologies

Logic based formalism are used for a long time in a wide variety of domains. These domains include all kinds of expert systems, e.g. for medical diagnosis, configuration management, software engineering and many others. Knowledge in such systems is created by specially skilled experts in conjunction with domain experts, because creating and maintaining large knowledge bases is far from being trivial.

In the last years, a new application area for logic based descriptions emerged: the Internet. The current Internet is optimized for the access by human users. It contains mostly text and image data that can easily be processed by human users, that is, however, quite hard to process automatically. The increasing amount of highly interactive multimedia content makes this problem even more severe. Search engines, such as Google, derive semantic information from such data only indirectly, e.g. by extracting keywords from webpages. For many applications it would be desirable to obtain explicit, highly specific semantic information on web resources. Applying logic based representation mechanisms is the vision of the semantic web.

> „The semantic web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation“[20]

There are several languages on the semantic web. Most popular is still the resource description framework (RDF)[6]. It allows statements that represent named links between entities, very similar to conceptual networks. This makes it equivalent to concept networks. RDF Schema[7] allows to define simple ontologies for RDF. Because there were

[6] http://www.w3.org/RDF
[7] http://www.w3.org/TR/2002/WD-rdf-schema-20021112

still problems of finding a formal semantics for these approaches, most current semantic web representation mechanisms add an additional semantic layer that is derived from description logics. An example is the Web Ontology Language (OWL)[8], that is partially based on the $\mathcal{SHOIN}(\mathcal{D})$ description logic [11].

There are several problems that need to be solved before the semantic web may become a reality. First, much of the knowledge shared on the Internet is rather implicit and very hard to express explicitly. Even if this is possible, the costs to do so can be extremely high. This raises the question whether the benefits of the semantic web make up the costs of building it. Many applications work well enough with much simpler semantic descriptions (as based on natural text). A prominent example are approaches to build structured white pages on the Internet, that would contain names, phone numbers, mail addresses, etc. of users in a well defined format. None of these approaches was successful so far, because all users simply applied Google keyword search to find people on the Internet.

Finally, the semantic web depends on the active participation of large number of users. Otherwise it will not be possible to annotate the large amount of information the Internet offers. This participation does, however, require certain skills, because the aim is to create annotations in a formal language. It is not possible to make any strong assumption about how skilled users are. Experience shows that most of them are not really capable of dealing with simple logical expressions. Most users never even apply logic operators in keyword based search. This makes a participation in the semantic web a non-trivial task.

This problem can be solved in two ways. Either, we can use variants of logic that better fit to human thinking. Research on non-monotonic logics and reasoning goes into this direction [25]. The other possibility is to stick with simpler forms of structuring and combine them with advanced data mining technology. In this work, the second option is chosen. A point of departure are taxonomies and cluster structures that are defined by the user extensionally and by an informal textual description rather than formal descriptions. The success of bookmarking and tagging systems shows that this level of interaction is well accepted by most users. This topic will be discussed in some more detail in chapter 3.

1.5.3 Creating Formal Ontologies from Data

Formal ontologies can be created and populated (semi-) automatically, which is, however, more challenging than creating cluster structures, because beside the concept hierarchy, relations and formal descriptions have to be derived. As formal ontologies are logic expressions, many relevant work has its origin in the area of inductive logic programming. The idea is to induce domain knowledge from facts of the domain. In the terminology of description logics to infer a TBox from an ABox. This is usually a highly interactive

[8] http://www.w3.org/TR/2004/REC-owl-guide-20040210

process: the user might enter some knowledge into the knowledge base manually and then infer additional knowledge automatically using a data mining algorithm. In [128] the term „balanced cooperative modeling“ was coined for this process.

In the following we will exemplarily sketch the algorithm KLUSTER [93], that generates a TBox from an ABox.

KLUSTER works in several steps. In the following we assume $\Delta = D$, thus the underlying universe of discourse contains a predefined set of items.

In a first step, KLUSTER collects a set of candidate concepts C_{cand}. For each predicate symbol appearing in the ABox, all entities that are in the extension of the predicate are collected into a concept. For each role, two concepts are formed. One containing all entities that appear in any ABox assignment at the first place of the relation and a second concept for all entities that appear at the second place of the relation. These candidate concepts are ordered according to the subset relation. Additionally, the pairwise disjointness of these concepts is determined.

The result of the first step is a DAG with possible overlaps on each level. On the one hand, these overlaps are not desirable from the point of view of ontologies. Rather subconcepts of a superconcept should form a partition, thus should be pairwise disjoint. On the other hand, overlaps are necessary, because sometimes the same domain can be described from different viewpoints or aspects.

Music, for instance, could be described by the concepts „rock“, „pop“ and „jazz“ on the one hand and by „germany“, „uk“ and „usa“ to denote the country of origin. While we could assume that the first group of concepts forms a partition and that the second group of concepts forms a partition, it is very likely that there are overlaps between concepts of both groups (e.g. jazz music from Germany). An ontology learning method should be able to deliver concepts for both aspects. KLUSTER is such a method.

The key to solve this problem is to search for *mutually disjoint concepts.*

Definition 1.5.2. Given a set of concepts $C' \subseteq C_{cand}$, such that $\forall c \in C' : c \prec c_{super}$, where c_{super} is the direct ancestor of all concepts in C'. Such a set C' forms a set of *mutually disjoint concepts* (MDC) , iff

1. $\forall c \in C' : c \prec c_{super}$, where $c_{super} \in C_{cand}$
2. C' is a partition of the items in $ext(c_{super})$.
3. There is no $C'' \subseteq C_{cand}$ with $C' \subset C''$ and C'' also fulfills 1 and 2.

Thus an MDC is a partition of the subconcepts of a given concept. KLUSTER finds such MDCs using a top-down procedure for each inner concept in the hierarchy. This can be, however, quite costly in the worst case, because all subsets of concepts $C' \subseteq C_{cand}$ must be evaluated on whether they form an MDC (and in the worst case they do). This leads to computational costs that are exponential in the number of concepts in C_{cand}. If this set is rather small, then the task of finding MDCs is feasible.

In the next step, KLUSTER obtains terminological descriptions for the concepts in each MDC in a top-down manner. The idea is to first find for each concept in an MDC a most specific description, in terms of a DL expression, i.e. find a DL expression that has an extension that is a minimal superset of the extension of the concept. Such an expression is called *most specific generalization* (MSG). MSGs are, although they are maximally specific, not guaranteed to be perfect. Particularly, MSGs might not be able to perfectly separate concepts in an MDC, which might not be desirable, because concepts are overlapping in this case.

There are two possibilities to deal with concepts which are „ill-defined“ by their MSG. First, these concepts can be defined as atomic concepts, given by their ABox defined extension. Second, MSG can be made more specific by generating new concepts. These concepts can be subconcepts of concepts used originally in the MSG, or concepts derived by splitting the ranges of roles used in universal quantification.

All concepts that could be defined without further specialization can be further generalized. The idea is to find a DL expression that covers maximally many entities but does not overlap with other concepts in the MDC. MSGs are generalized to *most general discriminations* (MGD) by first dropping restrictions (thus conjunctive terms) from the expression. Then all restrictions are relaxed by replacing the original target concept by a superconcept such that the overlap within an aspect is excluded. This approach is exponential if an optimal solution must be found. In practice it is sufficient to use any solution. Other generalizations include the relaxation of number constraints.

There are approaches to learn ontologies from textual data [37] that work similar to approaches such as KLUSTER, require, however, considerable preprocessing.

In chapter 5 we will discuss the suitability of ontology learning for collaborative media organization.

1.6 Conclusion

Structuring items is an essential task in many application areas. There are several formalism for structuring items and representing knowledge. They differ in their syntax and semantic, in their expressiveness and in the kinds of inference they support. This chapter gave an overview of different formalisms, their variants and corresponding algorithms and evaluation measures. Furthermore, it was shown how these formalisms are interrelated and how they are applied in practical applications.

A focus of this chapter was on concept or cluster structures. Such structures are very well-suited for information structuring because they are simple to process and to comprehend and still powerful enough to express semantic meaning needed to navigate or search large information collections. The task of creating such concept structures from data is denoted as clustering. An overview of clustering algorithms was given, including many important variants, such as subspace clustering or non-redundant clustering. Also, the problem of evaluating clustering structures and algorithms was discussed. Complementary to

clustering is the task of classification, i.e. of assigning new items into a predefined set of classes, based on examples for such assignments. This problem was put into relation with the formalism of concept structures.

Clustering and classification on concept structures are the point of departure of this work concerning structuring information. They can be applied to assist users in creating and organizing information collections. It will be shown that the fact that several users create and organize (partially the same) information, using concept structures, leads to several new solutions for clustering and classification, that will be denoted as collaborative clustering and classification.

2 Distributed Computing

2.1 Introduction

The Internet enables users to share information independently of their geographical location. This is achieved by a variety of underlying network technologies and paradigms that allow for communication and cooperation in distributed systems. As most information collections are distributed, the question of how to (collaboratively) structure information is tightly coupled to the question of how to share information in a distributed system.

This chapter gives a brief overview of the challenges of distributed computing. The aim of this analysis is to get an insight into the requirements of collaborative information structuring methods from the point of view of distributed systems. This will help to choose an appropriate paradigm and technology on a network level and to derive requirements for collaborative structuring methods from this point of view.

2.2 Distributed Computing

The vision behind distributed systems and distributed computing is maybe best expressed in the definition by Andrew Tanenbaum [176]:

> „A distributed system is a collection of independent computers that appear to the users of the system as a single computer"

A natural question is in which ways distributed systems differ from traditional systems. One obvious difference between a local system and a distributed one is timing. While for a single control flow all actions and operations are completely ordered, this is usually not the case in a distributed system. Imposing a global ordering on events is only possible with additional effort and can only be achieved to a certain extent. Connected to the problem of global timing and event ordering, there is a second important difference, namely the failure model assumed in distributed systems. A failure model describes which kinds of errors and failures may occur in an information system. There are several failures in distributed systems, that cannot occur in local systems. These failures can be structured into two classes, link errors and node errors. Link errors include the modification or loss of a message, as well as permutations of the order of messages sent from a node A to a node B. Node failures are stopping failures (a node does not send any messages any more) and Byzantine failures (a node sends arbitrary messages possibly not complying to the protocol the nodes agreed upon).

Communication is not only a source of failure but also an additional cost factor. Costs refers to the networking resources that have to be provided as well as to network latency (time for executing a distributed algorithm). Therefore, the total amount of messages exchanged in distributed processing is an important issue in analyzing and developing distributed information systems and algorithms. This is especially true for real time systems in which small differences in the network latency make a huge difference concerning the proper functioning of the system.

Security is another issue that is much more problematic in distributed settings than it is in local ones. Messages cannot only get lost by failure, they can also be recorded, maliciously modified, deleted, redirected, or generated. Also, the sender of a message can differ from the one stated in the message. These issues are captured in a security model that, in analogy to the failure model, states all possible attacks on the system.

Another problem often occurring in distributed systems is heterogeneity. Although this is a problem of every information system, local systems, residing on a single node, are usually homogeneous at least in some major aspects (e.g. the operating system). Distributed systems, on the other hand, may contain components that differ in any possible way. Interconnecting and coordinating such resources is a major challenge.

Distributed systems also require some services that are not necessary for local systems. Most important, there is a need to identify resources uniquely and to locate them in a network. Depending on the setting, this can be a conceptually very hard task.

In short, when developing a distributed system there are several additional problems to deal with. These problems include the ordering of events, node and link failures, communication costs and latency, security issues, the naming and location of resources and heterogeneity.

On the other hand, distributed systems offer several benefits. An important issue is scalability, the ability of a system to be extended in order to deal with increasing work load. Also reliability and availability are important requirements that can often be met only by distributing a system over several nodes. Finally, many information systems are inherently distributed. The popularity of the Internet can be attributed to a large part to its ability to make resources accessible from almost every corner of the world.

2.3 Paradigms and Technologies for Distributed Computing

2.3.1 Basic Networking

A fundamental operation in all distributed systems is communication. Communication has first of all a physical aspect, namely transferring data between two geographically separated points. In a network, such points are denoted as *nodes*. There is a whole range of technologies to support the communication between nodes in a network on a physical layer. Typical examples are cable, fiber optic, satellites and wireless connections.

The physical layer represents the most fundamental layer in the OSI network model [177]. Additional functionality that is essential for almost every distributed system is added on top of this layer, namely unique identification of resources and exchange of data over arbitrary physical channels. The most prominent protocols for these tasks are IP and UCP/TCP. IP allows to uniquely identify nodes and to route messages from one node to another. UDP and TCP provide higher level communication functionality. For TCP this includes certain guarantees, e.g. that messages do not get lost and that the order of messages is preserved on a channel. Higher level protocols can rely on these guarantees and have usually a much simpler failure model.

2.3.2 Models for Resource Discovery and Access

„Service“ and „resource“ are basic terms to describe cooperation in distributed systems. A *resource* is a hardware or software „entity represented or shared on a distributed system" [178]. A *service* represents a resource in a software system. Correspondingly, a node that provides one or more services is denoted as *server*, a node that consumes services is denoted as *client*. Nodes acting as client and server are often called *peers* [178].

Three fundamental operations in the context of services are providing a service, discovering/locating a service and accessing a service. A traditional model to implement these operations is the client/server model. There are some designated nodes that provide services (servers) and other nodes that access these services (clients). Often one service is provided by exactly one server. A typical example is the world wide web (WWW). The service provided by servers are data files (e.g. HTML pages). This service is invoked by a client calling the server directly and receiving the desired file as response. Resources are discovered e.g. by a DNS lookup or by search engines, such as Google, or Internet directories, such as Yahoo.

The client/server model suffers from poor scalability and reliability. A centralized server can become a bottleneck and a single point of failure especially if each service is provided by exactly one server. Replication can help to some extent, is, however, conceptually limited. A good example for this limitation is the so called „Slashdot Effect“[1]. A web site referenced on „slashdot.org“ usually encounters an extreme increase of requests in the days following the publication. The request rate then usually returns to its regular height after a short period of time. Traditional sever replication is not a good solution, because request peaks usually cannot be forecasted. For small, unknown web sites, that are unable to quickly introduce several replication servers for some days, this is no solution at all.

This observation led to the emergence of a more flexible pattern for resource access in distributed systems: p2p networks. In p2p networks, each node may consume and serve resources, because a peer is server and client at once. Replication and reliability are achieved in a natural way. Very popular resources are accessed and copied by many

[1] http://ssadler.phy.bnl.gov/adler/SDE/SlashDotEffect.html (4.12.2007)

nodes, which in turn act as a server for these resources. The availability of a resource increases automatically with increasing demand. Current systems, such as Bittorrent, make use of this idea to enable efficient sharing of large files on the Internet.

A finer grained distinction between the client/server and the p2p model can be achieved by regarding resource access and resource discovery as two separated tasks. In the Napster system (see e.g. [178]), for example, resources are located by a centralized directory service. The resources are then accessed directly in a decentralized manner. Such p2p systems are called *brokered systems*. Systems like Gnutella (see e.g. [178]), in which resource discovery is performed by a range limited broadcast, are called fully distributed or *true decentralized systems*. Finally, there are p2p systems, such as some instant messengers, that rely on a central server that not only enables resource discovery but also acts as a proxy server for the peers. Thus, all requests to other peers are sent to the server and then forwarded to the actual peers. Such systems can be denoted as *fully centralized systems*.

Fully distributed systems are further devised into structured and unstructured p2p networks. *Structured p2p networks* employ a reference space that allows for efficient resource discovery. Examples are Distributed Hash Table (DHT) systems, such as Pastry [152] or Chord [169]. Structured p2p systems often become suboptimal in the presence of a high fluctuation of peers [34]. In *unstructured p2p networks*, resource discovery and propagation is based on range limited broadcast. There are several approaches to make unstructured p2p systems more scalable. The authors of [34], for instance, propose a combination of topology adaptation, flow control and look-aheads.

A closer analysis of decentralized p2p (file sharing) networks reveals three interesting observations that are essential to understand and improve p2p systems. The first observation is that although all peers are equal in principal, they can often be grouped into two categories. First, nodes that have a lot of resources and are online for a long time without disconnecting. Such nodes usually have a lot of neighbors and have a fundamental influence on the overall network structure. They are therefore often called super nodes or super peers. The second class of nodes are regular nodes, that join and leave the system very often and have very few networking resources and neighbors. Figure 2.3.1 shows such a structure, that is often referred to a power law network (as the node degrees can be described by a power law distribution, see below). Super nodes play a similar role like servers in brokered p2p systems. They are, however, not set up explicitly but emerge automatically. A second observation in p2p communities is the so called small world phenomenon [98]. The observation is that the average distance between nodes is surprisingly low even in very large networks. Based on this observation, distributed search becomes feasible. A third observation is that nodes cannot only be grouped concerning their capabilities and interests but also with respect to the amount of resources they provide. Especially, there is often a large group of free riders, that only consume services but do not provide any. Such free riders have a negative influence on the performance of the system, because they cause much traffic without contributing to the system [149].

Free riders are a severe problem in (true) p2p networks and are often dealt with by special

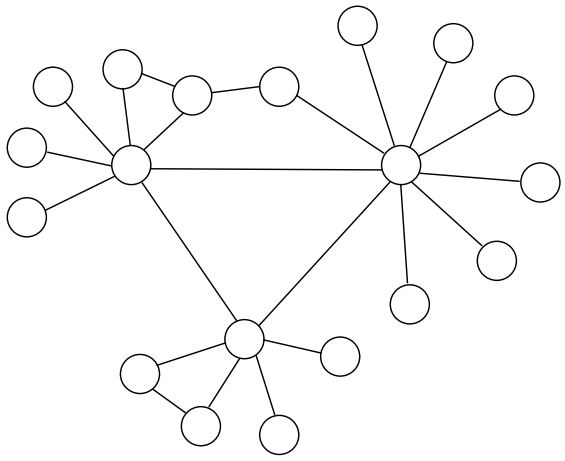


Figure 2.3.1: An example network with power law structure

policies defining a minimal ratio of services consumed and provided. Power law network structure and small world phenomena on the other hand allow for improved scalability in true p2p systems, while preserving the benefits of a self organizing, decentralized network structure. This combines the best of both worlds and is therefore the state of the art in p2p networking.

2.3.3 Paradigms for Higher-Level Cooperation

Communication and resource discovery/access provide the basic level for more sophisticated collaboration in distributed systems. While much of the details of this collaboration is application dependent, there are several paradigms that simplify the development and analysis of distributed applications by providing predefined building blocks, patterns and standards. Specific realizations of these conceptual building blocks are usually denoted as *middleware*.

In the following four such paradigmatic approaches (and corresponding middleware) will be discussed and compared: p2p computing, web services, grid computing and multi-agent systems.

P2P Computing

P2p networking breaks the traditional difference between clients and servers by allowing each node to consume and to provide services. The vision of *p2p computing* is more general and goes far beyond basic networking:

> „P2P is a class of applications that takes advantage of resources e.g. storage, cycles, content, human presence, available at the edges of the Internet“ [161]

What does „the edges of the Internet“ mean? In a client/sever architecture, servers are usually first class nodes that have many resources and are connected to other nodes through high speed connections. Clients on the other hand are computationally poor with very limited network connections. This limitation is constituted not only by a poor bandwidth. Many client nodes are behind network address translation (NAT) gateways and do not have an IP address on their own but share a single address with several other nodes. They are usually not directly reachable from outside. A similar problem are firewalls, that block incoming and often even outgoing connections. Finally, many clients are not online constantly but join and leave the system frequently. This is especially true for dial-up connections. Traditional network solutions never focused on such nodes *as servers* [178]. The major contribution of diverse p2p computing efforts is to incorporate such clients as servers by providing methods to deal with NATs and firewalls and by adopting distributed algorithms that can deal with a large number of nodes joining and leaving the system frequently and unpredictably. The focus on the development of such infrastructures and algorithms makes p2p computing different from many other paradigms described below, especially from large scale web services. Current applications based on the p2p paradigm go far beyond file sharing. Examples are the use of the p2p paradigm for web search [107] and distributed problem solving.

Web Services

A common way to think about services is as functions defined by an interface. Invoking a service is then similar to making a function call, only that the function is executed on a remote machine. This idea of cooperation underlies, for example, remote procedure call (RPC) systems. The shift to object orientation has led to several efforts to extend the RPC paradigm to deal with objects. Middleware, such as CORBA[2], allows, for example, to represent remote object references. Method invocations on such references are passed to a remote machine automatically. Object representations that are independent of concrete programming languages allow the integration of heterogeneous resources in a flexible way. Remote Method Invocation and Jini[3] go a step further. Because they are based on Java, not only data can be transferred between nodes but also executable byte code. A service in Jini is invoked by first obtaining an executable stub class which than handles the communication with the actual service. Both CORBA and Jini provide lookup services to discover resources in distributed systems based on interface descriptions.

While these paradigms are extremely powerful, web services may seem to go a step back. Web services enable applications to perform service invocations similar to remote procedure calls by exchanging XML documents using standard Internet protocols. The use of Internet protocols, such as HTTP, has several consequences. Most importantly, these protocols do not support the representation of state information directly. Correspondingly, there is no built-in support for remote object handlers. Also, the transfer of executable

[2]http://www.corba.org
[3]http://www.java.com

code is not covered by these protocols. While these are severe limitations, using Internet protocols has important advantages. First, stateless systems are often more robust and much easier to develop and test. Second, there is a lot of reliable technology, as web servers, that can be directly used as web service infrastructure. Finally, the orientation on standards, as e.g. XML, helps when integrating heterogeneous resources.

Web Services can be described by the Web Service Description Language (WSDL)[4]. This language allows to define which services are offered and how to invoke them. Services are invoked by Simple Object Access Protocol (SOAP)[5] messages. There are facilities to discover web services based on their description by the Universal Description, Discovery and Integration (UDDI)[6] service.

Resource and service description are usually low level, describing the interface of a service. The semantic web is an effort to describe resources on a semantic level. By giving resources an exact description, it will be possible to discover them with much higher precision, even if the requirements are expressed on a high level (e.g. „find a cheap flight from Dortmund to Paris“). Clear semantic descriptions should also enable automatic interaction among resources on the Internet, without explicit coordination. The semantic web pushes forward several standards, such as RDF, to allow for universal resource descriptions. This is in contrast to WSDL that allows for interface descriptions only.

Grid Computing

The vision of grid computing partially overlaps with the one of p2p computing, the focus is, however, very different. The idea of grid computing stems from power networks and electricity. Just as any electrical device can be plugged into a wall socket and works with electricity, the idea is to plug any computer or device to a network from which it then obtains services on demand. There are several aspects to this. First, each application should obtain all services automatically, without bothering the user with details, just as electronic devices work by plugging them in. Cooperation should be ad hoc: a service should be chosen that suits the need of the consumer in a given situation. This ad hoc interaction is in contrast to static cooperation underlying many current web service applications. Second, there should be an accounting and security infrastructure that delivers services and accounts them on a per use basis. Similarly, service providers should get paid based on how often their services are invoked. Finally, there should be a task allocation mechanism that allocates resources in an optimal way.

This vision is very general. In [60] Ian Forster gives three more specific conditions of what can be regarded as a grid. First, a grid „coordinates resources that are not subject to centralized control“. Second, it „uses standards, open, general purpose protocols and interfaces“ and third, „delivers non-trivial qualities of service“. While the first point is rather obvious, the second point disqualifies many p2p applications, because they usually

[4] http://www.w3.org/TR/wsdl
[5] http://www.w3.org/TR/soap/
[6] http://www.uddi.org

incorporate special purpose protocols for a given application or application area. The third point focuses on the definition of quality requirements (in p2p systems this would be e.g. the transfer rate).

There are several obstacles that have to be removed in order to make the vision of grid computing become true. Accounting and security are well known from previous frameworks and do not pose severe conceptual problems that are special to grid computing. A much more essential problem is the semantic description and interoperability of services. Nodes automatically have to find the services that are needed to achieve a given task. This search is often constrained by quality of service requirements and costs. The envisioned services are extremely general and should support tasks ranging from file download over high performance calculations to tasks such as travel booking. The proposed frameworks, such as the Globus toolkit [59], are very complex, because they must provide a wide range of functionality. This is probably the major difference to p2p middleware like JXTA, that provides only minimal support for semantic description and discovery and no support at all for aspects like accounting. On the other hand, p2p solutions are often more focused and easier to implement. Also, they directly address the problem of small nodes, as described above. This aspect is covered to some extend by the large grid frameworks but does not seem to be a major focus.

While Globus and similar systems represent efforts to create a general grid system, there are many frameworks for specialized tasks that resemble p2p computing frameworks. An example is high performance computing by utilizing computation time of many personal computers (e.g. Seti@Home [7]). Although such applications do not directly comply with the three conditions of grid computing, they are often seen as grid applications.

Grid computing envisions providing services in a network just as power networks deliver electricity. This has led on the one hand to rather complex frameworks that support various aspects, such as security, accounting, load balancing and semantic description and discovery. These efforts move into the direction of Service Oriented Architectures and are driven by large organizations. A more general vision is to use semantic description, in the sense of the semantic web to allow for universal interoperation of resources on a high level. This effort is often denoted as semantic grid [64]. On the other hand, there are several specialized applications and frameworks that try to accomplish resource intensive tasks by combining many low performance nodes on the Internet. These efforts ressemble current efforts in the p2p computing community.

Agents and Multi-Agent Systems

There are many different definitions for the term „agent“ and „multi-agent system“ (MAS) [84]. It is important to notice that the notion of agents comes from at least three different sources to understand this diversity. First, „agent“ is often used in the sense of personal assistant [105]. An agent is a software that supports its user by performing diverse assistance tasks, e.g. negotiating appointments. This often presumes some degree of personalization, making the agent aware of the preferences and interests of the user.

Most research on agents in the sense of personal assistants is contributed in the area of intelligent human computer interfaces.

A second notion of agents and multi-agent systems has its origins in research on emerging behavior. Ant colonies are often referenced as a typical example of very simple entities achieving rather complex overall behavior. The idea of intelligence as emergent behavior is in strong opposition to the idea of intelligence that assumes that intelligent behavior is achieved by maintaining a representation of the world and then using reasoning (planning) to choose the right actions [26].

A third notion of an agent comes from robotics and refers to physical agents moving in a real world environment, possibly interacting intelligently with the physical world and other agents [153].

Beside all these differences, agents are assumed to have at least three properties. They should be autonomous in the sense that their internal behavior is not fully determined from outside. Rather, an agent makes commitments to the user and other agents and these commitments define its behavior. Agents are rational, which means that they have explicit goals and follow them in a rational way. Third, agents are social, they communicate and collaborate with the user and with other agents.

Modeling distributed components as agents goes far beyond the paradigms presented so far. Since agents can be described by goal driven behavior, they allow for much more general and complex representations than e.g. web services do by offering simple interface descriptions. The research in this area correspondingly focuses on rather high level semantic cooperation. There is a large variety of topics ranging from trust, negotiations, emerging coalitions, adaptive behavior, game theory, agent reasoning about other agents and themselves, cooperative language learning and a lot of other topics. Much of this research is rather visionary and theoretical and is not incorporated in any large scale application. On the other hand, an effort, as is the semantic web, makes extensive use of existing research in the area of MAS, especially concerning issues like semantic descriptions of resources or interoperability.

Overlaps and Differences

All the paradigms and technologies described above reflect in some sense the specific nature of distributed systems. They provide diverse mechanisms to provide, discover, access and combine services. Also they usually allow for the integration of heterogeneous resources in flexible ways and provide layers for security, accounting etc.

Despite these common goals, the focus of the individual paradigms described above differs very strongly. P2p computing focuses on algorithms and technological frameworks to interconnect a large amount of small, computationally poor nodes. Web services rather target large scale integration of existing services on the Internet based on Service Oriented Architectures and are therefore often not well suited for fully distributed p2p computing. The semantic web tries to develop an universal description language for resources to

enable ubiquitous integration of services. This is a vision shared with the multi-agent community that focuses on complex high level cooperation among nodes. Finally, current grid applications move into the direction of state enabled web services, on the one hand, and the semantic grid, on the other hand. Also, there is a clear focus on integration, load balancing and accounting.

2.4 Description and Analysis of Distributed Systems

2.4.1 Formal Models

Basic models for distributed computing capture the communication and coordination among a set of nodes. Nodes are modeled as processes that have the ability to exchange messages with other processes to accomplish a (common) task. Such models are used to obtain complexity or (im-)possibility results for distributed systems. Distributed systems can be modeled as synchronous or asynchronous systems [113]. A synchronous system can be described as follows.

Definition 2.4.1. A *network* is a an undirected graph (V, F), where V denotes the set of nodes. A communication channel between node $v \in V$ and $v' \in V$ is denoted as $(v, v') \in F$.

In such a network, we can define the notion of neighbors of nodes.

Definition 2.4.2. The set of *neighbors* of a node $v \in V$ in a network (V, F) is defined as

$$nbrs(v) = \{v' \in V | (v, v') \in F\}$$

containing all nodes that are directly connected to v.

In the following it is assumed that edges are undirected and that communication channels are bi-directional.

In general, the network graph may be of arbitrary topology. Some of these topologies are especially popular.

Definition 2.4.3. In a *client/server network* (V, F), there is a designated node $v_s \in V$, the server, that is connected to all other nodes via a bi-directional communication channel, thus

$$F = \{(v, v_s) | (v \in V) \wedge (v \neq v_s)\}.$$

Usually, there are no other connections.

A second model that is very popular are fully connected networks.

Definition 2.4.4. In a *fully connected network* (V, F), each node is connected with each other node, thus

$$F = \{(v, v') | (v, v' \in V) \wedge (v \neq v')\}.$$

Finally, nodes can be partially connected. This is a structure found in many p2p and ad-hoc networks. An important characteristic of such networks is the distribution of node degrees.

Definition 2.4.5. The *degree of a node* $v \in V$ is the number of its neighbors, thus

$$deg(v) = |nbrs(v)|$$

Using these node degrees, we can calculate the distribution of node degrees for a network.

2.4.2 Network Topologies and Simulation

Simulation is a standard instrument to analyze distributed systems. One of the most important tools for such simulations are random networks or graphs that represent the topology of the network. As mentioned above, many networks expose a power law structure. They contain a small number of nodes with high degree and a large number of nodes with low degree. This can be captured formally in the following way.

Definition 2.4.6. A network is said to obey a *power law distribution* if the probability of node degrees follows the following relationship

$$p(k) \approx \beta k^{-\alpha}$$

where $\beta > 0$ and $\alpha > 0$ are constants and $k > 0$ is a node degree [108].

We can assume that $\alpha \in [2, 3]$ for most real world networks [14].

There are several models that allow to construct graphs and networks following the power law. They also try to give insight into mechanisms that lead to the emergence of such networks in various domains. In [14], much of the previous research on this topic is consolidated. The authors introduce the model of preferential attachment and growth. The idea is to start with an empty set of nodes V_0. At step l the network contains nodes V_l. Then, at each step, a new node is added to the network, connecting it to a node v' that is already in the network. The node $v' \in V_l$ is chosen randomly according to the probability

$$p(v') = \frac{deg(v')}{\sum_{v \in V_l} deg(v)}$$

This principle is denoted as „preferential attachment“. In a more general sense it is referred to as „rich get richer“ or „the winner takes it all“. Nodes that already have many

connections are likely to get many additional ones while such with few connections are likely to obtain few new or no links. It is interesting that if the node is chosen at random with equal probability at each step, the resulting network does not obey a power law distribution of node degrees.

The simulation procedure described above can be shown to yield a network in which the distribution of degrees follows a power law distribution with $\alpha \approx 2.9$ [14]. Several variants of this basic simulation procedure were proposed to allow for other values of α. In [4] connections between existing nodes can be added. In [143], it is argued that often two processes mix in the emergence of random networks. New nodes either connect according to preferential attachment to some „popular“ node or according to some other preference to a random node. They show that the resulting distribution of node degrees reflects the structure of the current WWW much better than the simple model presented in [14].

2.5 Conclusion

This chapter described the foundations of distributed systems as well as challenges faced in distributed applications that do not occur in local applications. These challenges concern, for instance, different kinds of failures in the network or synchronization of nodes. Also several paradigms for creating distributed systems were discussed in detail. These are p2p computing, grid computing, web services and agent systems. It was shown that each of them has an own focus and solves certain aspects connected to distributed computing. Almost all of these systems rely on a client/server model.

The paradigm of p2p computing is particularly powerful. It can be applied in any kind of networks, making only minimal assumptions on bandwidth, structure or reliability of the network. Given that many current media organization systems are based on mobile devices and ad hoc networks, p2p computing will play an important role in this area. A downside of this flexibility is that algorithms for p2p computing must be able to operate in such an environment in a robust way. This is especially challenging for data mining algorithms, that usually require rather complex computations. Most distributed data mining algorithms are based on a large amount of messages that must be exchanged among nodes in a synchronized way. Achieving synchronization in a p2p network is very costly and sometimes even impossible without a central server. We will come back to this problem later in chapter 4.

In part two of this work, algorithms for distributed feature extraction will be developed, that can be efficiently implemented in p2p networks. This is a prerequisite of enabling collaborative structuring in p2p networks. We will see that current models using an epidemic dissemination approach are not well-suited in heterogeneous settings and should be replaced with approaches that are selectively exchange only relevant information.

3 Distributed Information Structuring

3.1 Introduction

So far, we discussed several formalisms for structuring sets of items, reviewed how they can be used to facilitate the access to information collections and presented different algorithms for structuring items (semi-) automatically based on features of these items. Also, we discussed basic concepts of distributed systems and corresponding technology. How can both be combined? Or, to put it another way, how can users, that are scattered over a loosely-coupled network, structure and represent a set of common items?

In the following, we apply a notion of distributedness that goes beyond the distribution in a network. A major characteristic of distributed systems is that the components are only loosely-coupled and that several processes run independently of each other. In the same sense, information can be structured in a loosely-coupled way. There are two aspects to this. First, methods for structuring, such as are clustering or classification, can be applied to distributed data. This could be denoted as data perspective to *distributed structuring*. The area concerned with research on this topic is *distributed data mining*. In section 3.2, a short survey of corresponding methods is given.

Second, we can regard structuring from a user perspective. Structures that serve a user to access an information collection must reflect the knowledge and preferences of this user to some extent. So if there are several users, how should a set of items be structured to suite the needs of all of them? And how does the information structuring process proceed if there are several users involved? These and other questions will be discussed in the second part of this chapter.

3.2 Distributed Data Mining

As mentioned above, most current information collections are inherently distributed. Data mining was shown to be a key technology for automatic structuring of information collections. This leads to the problem of how data mining can be applied to distributed information. Research in the area of distributed data mining has contributed several methods to solve this problem.

The focus of most of the work in this area has been on scenarios in which *global* patterns or concepts should be identified from large, distributed and often heterogeneous data sources. The aim is to achieve a result that is as accurate as if all data was processed

at a single node. This is achieved by sharing only relevant information. What kind of information is shared depends on the particular algorithm.

In systems that perform distributed classification, nodes usually exchange models of the relationship between the features and target value. A typical example is distributed Naive Bayes [103] for which only conditional probabilities are exchanged. A more general approach is to train an arbitrary classifier on each of the distributed data bases and then to combine them [170, 146]. Research on ensemble classifiers has shown that iterative approaches, such as boosting, often improve the accuracy of the classifier considerably. This has led to approaches, such as distributed boosting [106]. All of these approaches aim at training a single global classifier.

Similar approaches exist for clustering as well. In distributed k-means [41], for instance, nodes simply exchange their local centroids in each iteration. A distributed variant of the dbscan algorithm [52] can be implemented by allowing nodes to send points that lay in a dense area to a central node, that then performs dbscan on the union of points received from the individual nodes [82]. [89] describes an approach to cluster data hierarchically at each node and to combine the resulting dendograms at a central server. In a similar way, flat cluster ensembles can be used. Each node clusters the data using any algorithm and a central node then combines these cluster structures.

Finally, there is a large amount of research on learning (association) rules from distributed data [206]. Corresponding algorithms mostly share frequent itemsets. [201] describes an approach to distributed rule discovery that is based on sharing counts and pruning information to enable a distributed breadth-first search in the hypothesis space.

We exemplify the idea of distributed data mining on the distributed k-means algorithm (as described in [92]). A central coordination node is assumed, as well as some distributed nodes V that have access to the data. Each node l contains a portion of the data space $D_l \subseteq D$ and we assume that these subsets do not overlap. The coordination node randomly calculates centroids for k clusters and sends them to all other nodes. Now, each node assigns each of its local data points to the nearest centroid and calculates local centroids for each cluster. It then sends these local centroids together with counts on how many items were assigned to each cluster to the coordinating node. The coordinating node merges the centroids received from all nodes. This can be achieved easily by

$$z_{ij} = \frac{\sum_{l=1}^{|V|} z_{ijl}}{\sum_{l=1}^{|V|} n_{jl}}$$

where z_{ijl} is the centroid value of feature X_i for the jth cluster as calculated by node l and n_{jl} is the number of items from D_l that where assigned to cluster c_j.

The local centroids are calculated as follows

$$z_{ijl} = \sum_{x \in ext(c_j) \cap D_l} X_i(x)$$

Note that $ext(c_j)$ is a global property: all nodes use the same centroid to determine which item belongs to which cluster. The actual assignment is, however, performed in a decentralized way at each node locally.

The coordinating node then sends the new global centroids to all nodes, that again respond with local centroids, until the algorithm converges or a maximum number of steps is reached.

Most distributed data mining approaches assume a set of fully connected nodes and a reliable communication network. However, the currently emerging field of p2p data mining tries to allow for distributed data mining even in loosely coupled networks. Almost all approaches to p2p data mining are based on epidemic dissemination protocols that use simple base operations, such as distributed majority vote or averaging. In [103] an approach based on the newcast model of computation is presented. It is based on the dissemination of information by probabilistic broadcast [53] and applied to share conditional probabilities for distributed Naive Bayes. In [193] an algorithm for distributed averaging in p2p networks is proposed, that is used to evaluate locally generated rules under the global distribution. [41] proposes a distributed k-means algorithm that uses this strategy to obtain globally valid centroids from local ones by distributed averaging. These approaches aim at establishing a kind of an equilibrium among the nodes, such that all peers share the global model (or its approximation). While dissemination is well suited in a homogeneous settings, it can produce a considerable overhead in heterogeneous settings, in which each peer aims at developing an own local model. In the worst case, each node faces a data mining problem that is completely independent of all other problems. Then, a large amount of network bandwidth and local resources is wasted, because only irrelevant information is shared among nodes. This demands for models and methods that are targeted at the heterogeneous case.

In this work, we therefore discuss an alternative scenario. In this scenario, a large number of loosely coupled nodes applies data mining to different, usually very small and overlapping subsets of the entire data space. The aim is not to learn a general, global model to cover all data, but to learn a set of *local* concepts. Thus, each node learns an own concept and cooperates with other nodes only to improve local learning. Still, tasks can be related, but instead of assuming that all tasks are identical, cooperation and information flow among tasks should emerge.

The areas most engaged in research on how to transfer information among heterogeneous learning tasks are Multi-task Learning and Multi-Agent Learning. In both areas, the key is to measure the relatedness of different data mining tasks and to share information only if the tasks are similar. This topic will be discussed in detail in chapter 4.

3.3 Distributed Structuring From a User Perspective

The problem of a group of users structuring information occurs in many scenarios. Examples are knowledge engineers that collaboratively design a domain ontology, different

vendors consolidating their product descriptions or users tagging websites using a social bookmarking system. In the following, we will make some basic distinctions that help to classify these scenarios. Relevant aspects concern the desired outcome of the process as well as its nature. Also, the number and the skills of the participating users must be regarded.

An obvious question is whether the result of the structuring process should be a single taxonomy or ontology for all users or a set of personal taxonomies or ontologies for each user individually. The first option will be denoted as *global structure*, the second one as *local structure*. Between these two extremes, group specific structures are also possible. They will be in the following denoted as local structures as well.

Popular examples for global structures are web directories or taxonomies used in (digital) libraries. Examples for local structures are personal bookmarks or directory structures to organize files.

A clear advantage of global structures is that they can be created and maintained by a small number of administrators. These experts use their extensive domain knowledge to produce a model which they think most users can accept. A large number of usual users then profits from this effort by simply using the resulting structure. They do not need to participate in its creation and maintenance.

Another advantage of global structures is that they may serve as a means of communication. Imagine one user asking another one where to find information on „spatial data mining“. The second user could then respond with specific topics in the Yahoo! hierarchy containing interesting references (instead of sending the references themselves).

Low effort for casual users and standardized communication are certainly two important advantages of using global structures. They are, however, very limited in capturing any personal preferences and background of individual users. Information spaces (as the Internet) are sometimes compared with real spaces. Many people take their participation in various applications on the Internet as essential part of their living. In this sense, structuring information for better access can be seen as similar task to furnishing a house or arranging cooking tools in a kitchen. While the suitability of this metaphor maybe questionable in general, it makes one point quite obvious: users like to personalize their access to information. Just as most people would not like to live in an apartment that is furnished according to some global standard, they do not want to organize their information in a globally standardized way. In fact, many people really like to structure their information to some extent, despite the additional work. The success of (social) tagging and bookmarking systems stands evidence for this claim.

The same holds for companies or departments in companies. Even when dealing with similar issues, different groups tend to prefer their own conceptualization. An engineer, for instance, is often not willing to use business terminology. In such cases, it is not only desirable but essential to allow for different conceptualizations, because otherwise the system will not be accepted.

Coupled with the issue of local and global structures is the question of the *scope* of

a structure. Again, this scope can be global or local. The scope of a standard genre classification for music are all possible musical expressions. The scope of a directory tree in which a user organizes her mp3 files is only a small subset of this global set (e.g. only jazz music). This example already suggests that mostly a local structure comes with a its own corresponding scope. This must, however, not be the case. A user might classify the same set of items using different, local taxonomies, all of which have the same scope.

Another important distinction for local structures is whether they are coupled. Coupling refers to the question whether it is possible to make relations that connect two such structures. It would be, for example, possible to define a personal concept as follows:

$$rock_{u1} \equiv metal_{u2} \sqcap alternative_{u3}$$

The concept „rock“ of user u1 would contain all items that are contained in the concept „metal“ of user u2 and in the concept „alternative“ of user u3. In uncoupled systems, such connections cannot be made.

So far, distinctions were concerned with the outcome of the structuring process. This process itself is, however, equally important.

First, structuring can be coordinated or implicit. If the structuring process is coordinated, users must explicitly communicate with others users to achieve a (common) result. Examples of such explicit approaches are collaborative ontology editors. In implicit systems, users must not coordinate their actions with others. In the extreme, they must not even be aware that other users are using the system and can still profit from their work. Examples for this kind of implicit communication are given below.

If structuring is coordinated, a second question is whether the aim is to reach an agreement. If an agreement must be reached, the outcome of the process is a single agreed upon structure. Corresponding systems must provide mechanisms to deal with all kinds of conflicts and ways to resolve them.

Finally, an important question is the number of users, as well as their knowledge, skills and motivation to use the system. The applicability of the methods strongly depends on this question. A system designed for a small group of knowledge engineers is most probably not well suited for casual Internet users that want to share their bookmarks.

In the following, several approaches to structuring information in a collaborative way will be discussed concerning the dimensions presented above.

3.3.1 Collaborative Ontology Engineering

Tools that help users to work together on a common object are usually denoted as *groupware*. Examples of groupware are collaborative text editors or distributed software development tools. They coordinate the actions of several users such that the object they are working on remains in a consistent state. Corresponding methods can be conceptually

rather simple, such as mutual exclusion on a common resource, or very sophisticated (e.g. enabling „undo“ in a collaborative word processor). The research area concerned with such systems is called *computer supported collaborative work*.

Creating and maintaining large ontologies and knowledge bases is a very complex task. Also, there are often several people involved. Therefore, it is desirable to provide tools that enable these users to coordinate their actions (semi-) automatically. This has led to a large number of collaborative ontology editing tools.

First of all, a collaborative ontology editor allows for common access to a central ontology. In many systems (e.g [55], [13]), this access is web based, an approach that has proved to be very successful in other groupware systems, such as in the popular BSCW system [19].

Second, these tools help to coordinate the actions of different users. This is achieved on different levels. One aspect is rights management. It is possible, for instance, to allow users only restricted access to a common ontology depending on their access rights. In this way, important parts of an ontology can be protected, while other parts can be made available for change and addition. Related to this problem is the problem of consistency and synchronization. Often, it is desirable that two users are not allowed to simultaneously make changes to the same part of the ontology, because this may lead to severe consistency problems. Therefore, some systems, such as OntoEdit [174], allow to lock concepts, such that only one user at a time can edit them. Related to this approach is the use of transactions. If a user performs several related changes, transactions assure that either all changes are performed or none of them. This avoids situations in which a user changes the ontology at two different points of which only the changes at the first point are committed (leading to inconsistencies). All of these approaches assume that users work tightly coupled on the same global copy of the ontology. Another approach is to allow users to work on local copies of the common ontology. Users obtain a copy, modify it locally and then synchronize the result with the central repository. Changes are usually logged, allowing to recover older version of an ontology in a convenient way. Corresponding systems, such as OKNI [102], use methods that are very similar to the methods used in CVS [58]. Similar to this approach is the use of a private and public area, such as in Co-Protegé [45]. Users make changes in the private area and then publish the result in the public area. The problem of ontology change in collaborative ontology editing is discussed in detail in [137].

Most of these methods are quite generic. They are applied in many application domains (e.g. software engineering) as well. An important advantage of formal descriptions is that they can be used to automatically detect inconsistencies. This allows, for instance, to first check whether changes made by a user would leave the ontology in a sound state. This approach is supported, e.g., by Ontosaurus [175]. However, consistency checks maybe quite inefficient in a large ontology. Therefore, an important addition is the use of a module concept. Modules allow to encapsulate parts of the ontology and to link them to other parts by well defined interfaces. This makes consistency checks, even in a distributed system, feasible. An example of a system implementing such modular

ontologies is Wiki@nto [13].

While access management and consistency checks help to avoid accidental inconsistencies, this is often not enough. Another important issue to enable successful collaborative ontology engineering is communication. Multiple purpose communication can be achieved by any tool, such as email or instance messaging. There are, however, tools that are tailored directly to ontology editing. An example is Tadzebao [49] that allows users not only to communicate texts but also sketches and partial ontologies. In this way, they may discuss intended changes. Another important mechanism to support communication is a notification system. These systems allow users to register for certain parts of the ontology and the users are then informed about any changes performed by other users. To make such notifications manageable in large systems, tools, such as the Ontolingua Server [55], allow to create sessions. Users can join a session or create a new session. Notifications are only sent among members of a session. This method can also be used to achieve mutual exclusion on a group level, such that nobody outside the group can make any changes.

Another aspect of collaborative ontology editors is process support. The creation of an ontology can be seen as a complex process including several different steps and stages. The OntoEdit [174] system supports these stages by providing several tools for different steps, e.g. a concept map tool for brainstorming. The most important aspect, however, is the problem of achieving agreement among a set of users about how a domain should be modeled. The approach described in [76] makes use of the Delphi methology for this task. Each user proposes changes. An editor collects all proposals and consolidates them in a global document. This document is then passed back to the users, which may modify or retract their proposals. To keep the process focused, specific speech acts are used (e.g. revision, addition, clarification, etc.). A very similar approach is described in [144]. Here, however, the focus is on an evolving system, in which users make their modifications locally and synchronize with a global copy only from time to time.

There are many different tools that support (in parts) the functionality described above. One of the first and most powerful systems is the Ontolingua Server [55], that already provided a lot of innovative functionality, such as a separation of presentation and representation of ontologies, access control and session based synchronous editing. A system that focuses on the process of ontology engineering is the OntoEdit tool [174]. Its major contribution is a fine grained locking mechanism to allow for mutual exclusion on a concept basis. The Tadzebao system [49] introduced several new concepts concerning communication. It allows users to exchange multimedia data consisting of text, hand written sketches and partial ontologies. The Wiki@nt system [13] makes use of an advanced web based interface inspired by Wiki based systems. It introduces module based editing and consistency checks.

Collaborative ontology engineering is targeted at small groups of highly skilled experts that want to coordinate their actions in maintaining large, complex knowledge bases. The aim is always to reach one global, explicitly agreed upon structure.

3.3.2 Ontology Reuse

The aim of collaborative ontology engineering, as described above, is to support a group of users to create a new ontology. There is, however, another form of collaboration that is rather indirect. Users create ontologies for their specific purpose. Then they can make these ontologies publically available. Now, users who need to model a domain may first look whether they find a (partial) ontology that already fits their needs. If this is the case, they may reuse it as a whole or in parts. The overall vision of these approaches is to create a library of ontologies which can be reused in many applications, reducing the effort in creating new ontologies. This vision is similar to the vision of creating reusable software components. Most of the systems mentioned in the last section provide an ontology library as well.

The major question is how to find a relevant ontology. Most approaches currently rely on simple keyword search. Users provide an expression and the system then matches this expression against the concept names in the ontologies. The Ontolingua server [55], for instance, provides this kind of functionality. An extension to this simple mechanisms in presented in [28]. Ontologies can be evaluated by the users according to different aspects, such as readability or level of formality. If a user searches for an ontology, the result of this search is filtered by looking for evaluations by other users that fit the query. An average score of these ratings is then used to re-rank the results. While the applicability of this approach is questionable in the context described in [28], it correctly focuses on two aspects. First, the evaluation of ontologies is a process that depends heavily on human judgment. Therefore, ratings given by users are an important source of information, when recommending ontologies. Second, the criteria used to judge on the quality of ontologies are subjective as well. Therefore, it is important to find the combination of criteria that matches the need of the current user.

Ontology re-use is a very powerful concept. It is a loosely-coupled approach that usually does not demand any coordination among users or user groups. Also, no common result has to be achieved, because each user may copy and modify the ontologies in a library to suit her own needs, leaving the original ontology untouched. Afterwards, she might possibly write the result back to the library as additional entry that others may reuse.

3.3.3 Ontology Merging

Often it is not possible to create a new ontology from scratch. Rather, one is faced with different ontologies that emerged in parallel over time and must be consolidated into a common ontology. Optimally, the resulting ontology should contain all concepts the individual input ontologies contain, without any redundant concepts. This task is called ontology merging [144]. Popular examples are catalogues of different vendors that are consolidated because one company acquired the other.

Ontology merging is usually a two-step process. First, equivalent concepts and relations in all input ontologies are identified. This step is denoted as ontology matching and will

be discussed in the next section. In a second step, corresponding concepts and relations are consolidated.

Integrating ontologies is extremely challenging even in seemingly simple cases. One ontology may contain, for instance, the concept „adult“ denoting persons aged 18 and more. In another ontology, persons are „adults“ if they are aged 21 and more. Also concepts are often constrained by their relations to other concepts. A person in the US may have a social security number, while a person in Europe does not. How such conflicts are resolved and how resulting concepts look like cannot be inferred from the input structures alone. It rather depends on the application at hand. For these reasons, ontology merging is a task that must in almost all cases be performed manually.

Systems, such as SMART [135], support the user in this process. The system can be used in two modes. In the first mode, called „merge“, both ontologies are assumed to be on the same generality level. The system tries to find equivalence matches between concepts and then either merges them or removes one of them. If the confidence in the match is not sufficient, the user is prompted for input. The „align“ mode works similar, however the assumption is that one ontology is less general than the other one, and the aim is to find a superconcept for the concepts in the less general ontology. The system employs a todo list to point out unresolved issues (as unmatched concepts) to the user.

Another approach to merge ontologies is based on formal concept analysis [173]. The algorithm requires a set of documents as items. The authors assume that the overlap between two ontologies may not be large enough to allow for an extensional matching. The approach therefore first creates two term lattices from the original ontologies and a document collection that must be representative for both ontologies. These two lattices are then merged and pruned. In a manual step, the user must resolve conflicts (e.g. a target concept having two different labels or a source concept without target concept).

Besides these approaches, there are several formal approaches that merge ontologies by making the context of a conceptualization explicit. This helps to resolve conflicts that occur because concepts are used differently in two contexts. As pointed out in [61], a more severe problem is that users have conflicting opinions about how to conceptualize a domain. A possible solution is to retain several conceptualizations in the same system but to annotate them with meta information on how they are used by which users. This process is called „social reification“ in [130]. It is, however, unclear, how to exploit these annotations in a real world application. Another approach is used in the APECKS system [179]. This system is based on frames and allows for conflicting ontologies. It compares these ontologies with respect to consensus, conflict, correspondence and contrast. A consensus is achieved if two concepts use the same terminology and attributes (slots). The use of the same terminology but different attributes corresponds to a conflict, the use of the same attributes but different terminology is a correspondence and if both differ, there is a contrast. These properties of concepts are used in structured discussions, that could, e.g., lead a user to rename a concept to a common name. An interesting approach to automate this process is taken in the KnowCat system [38]. Users can add concepts to a common taxonomy. They can also give feedback on concepts added by other users.

Concepts that get positive feedback receive a high degree of „crystallization“. Concepts with a low degree of crystallization are removed after some time. While the methods used in the system are rather basic, the idea of emerging concept hierarchies is a quite appealing alternative to ontology merging approaches.

In terms of the above characterization, ontology merging again aims at a global structure with global scope, however, without the explicit agreement. The resulting structure is rather chosen to capture as much of the input structures as possible and to suit the needs of the given application.

3.3.4 Ontology Mapping

As described above, integrating ontologies is very complex and often leads to contradictions that have to be resolved based on specifications taken from the application. On the other hand, merging ontologies is often not really necessary or even desirable. Organizations and individual users often prefer to structure information in their own way. Still these structures should be interoperable. An example are timetables of railway companies. Each company may prefer an own format to represent their timetables and still all of them should be combinable, such that the search system of one railway company finds connections of the others. The task to achieve this is denoted as *ontology mapping*. The idea is to find for each concept and each relation in one ontology corresponding concepts and relations in another ontology. The complexity of this task depends highly on the underlying ontologies. Today, ontology mapping is mostly performed manually. Technology supports this process by providing convenient low level representations like, e.g., XML.

Ontology mapping is sometimes distinguished from ontology alignment, denoting the task of making ontologies with different domains interoperable. This task will not be considered here.

Ontology mapping can be used in two modes [35]. Given k local ontologies, we can either first create an integrated global ontology to which all k ontologies map, or we can mutually map each local ontology to each other. The advantage of the first approach is that only k ontology mappings are needed. Also, if a new ontology is added, only a mapping for this ontology has to be created, which is very natural (e.g. a new railway company joining). Mapping all k ontologies to each other results in $k(k-1)$ mappings. Each time a new ontology is added, $(k-1)$ mappings need to be created, which is a clear disadvantage concerning scalability. On the other hand, there is no need to create and maintain a global ontology, which is a challenging problem in itself (as discussed above). Another benefit of not using a central ontology as mediator is that usually more information can be preserved. For instance, some of the railway companies could provide estimates on train delays. They would be hard to represent in systems using a common ontology, as not all participants provide this information. For each new company joining the system, the common ontology would have to be extended to reflect such particular

information. However, if we map such information directly, it can be optimally preserved, where this is possible, without any changes to a global information exchange format.

Another question concerns the type of links that ontology mapping establishes among concepts and relations. The most simple approach is to find corresponding pairs of concepts and corresponding pairs of relations in both ontologies. This semantic equivalence automatically also entails that both concepts have the same extension. An extension of this basic mechanism is to allow to express not only equivalence among concepts but also other semantic relations, such as subset. Thus, a mapping can define that concept c in the first ontology is a subconcept of concept c' in the second. Such a semantic mapping [162] is, for instance, used in the C-OWL [172] methology. This can be extended further by allowing arbitrary logical expressions in a mapping. In [32], such an approach is proposed, based on semantic and syntactic rewriting. Finally, the mappings themselves can be described in terms of an ontology. An example for such an approach is the Semantic Bridging Ontology used in the MAFRA framework [115]. It allows to specify information, such as the cardinality of the mapping or constraints that must be fulfilled to actually perform it.

In principle, a mapping between ontologies can be created manually. Quite appealing, but more challenging, is to find a mapping between ontologies automatically. This task is denoted as *ontology matching*. Solving this task is a prerequisite to make ontology mapping scalable. It has been intensively discussed in research on formal ontologies. However, the problem occurs quite prominently in yet another area, namely database and information systems. The challenge in this area is to find mappings between different database and xml schemes [148]. Methods in both areas resemble each other to some extent. The largest difference between both is that schema mapping approaches can not rely on a formal semantic description. Therefore, semantic relations, e.g. that two concepts are in subset relation, cannot be easily achieved by these approaches. In the following, a brief overview of approaches to schema matching is given. For a more detailed discussion refer to [91, 136, 148, 162].

Most matching approaches deliver a function that calculates for any concept in the first ontology and any concept in the second ontology a degree of how similar these two concepts are. This function is then used to guide the user through a semi-automatic process in linking corresponding concepts. Approaches differ mostly in the way this similarity is calculated. [148] makes the following distinctions. Approaches either regard each concept individually or they regard the relations among concepts. The first approach is denoted as element-level matching, the second one as structure-level matching. Mostly both are used in combination. In the CUPID system [114], a bottom-up tree matching procedure is used to match hierarchical concepts. In [118] an approach known as similarity flooding is used. Based on an initial element-based matching, similarities are propagated using a fix point like computation. Element matching methods can be classified into instance-level and schema-level methods. The latter ones use only schema information, such as the name of a concept, its datatype, distributions and similar information. Additionally, often external resources are employed, such as thesauri or lexicons. Instance based methods use the actual data items assigned to a concept as well. A very simple and

efficient approach is to compare the extensions of two concepts. This works well only if there is a considerable overlap between both ontologies. An approach that can be applied even if both ontologies contain disjoint sets of items is implemented in the GLUE system [47]. The idea is obtain an estimate on the probability that a given item is relevant to a concept c, denoted as $p(x \in ext(c))$. Given a concept c in the first ontology C and another concept c' in the second ontology C' the aim is to obtain the joint probability distribution, especially the probability of $p(x \in ext(c) \wedge x \in ext(c'))$ for a given $x \in D$. Given this joint distribution, any similarity measure can be applied. In [47] a generalized version of the Jaccard coefficient is used. The major challenge is how to obtain the joint distribution. The approach proposed in their work is basically to train a classifier on concept c and to apply it to the union of items in both ontologies. Then, the roles of c' and c are switched. This gives an estimate for any item contained in any of the two ontologies by which of the two concepts it is covered. The joint probability distribution is then obtained simply by counting. An important advantage of this approach is that any classification scheme can applied. The authors propose a multi-strategy approach, using instance attributes, name similarity, etc. In [48] a similar approach for schema matching is proposed. Most approaches combine different techniques. COMA [46], for instance, offers the possibility, to plug any schema matcher into a flexible meta schema matcher.

The approaches described so far deliver only suggestions on similar concepts in two ontologies. There are some approaches that are able to find semantic relations among concepts as well. An example is the s-match algorithm [63]. The basic idea is the following. In a first step, the approach heuristically finds a set of relations among concepts. This is achieved by using string matching (e.g. „P.O.“ and „Post Office“ are equivalent) and background knowledge, especially generality relations from Wordnet[1] (e.g. „Europe“ is more general than „Italy“). Based on these semantic relations and the relations that are present in the original ontologies, the missing relations are found by a constraint satisfaction procedure.

Ontology mapping is used in many web applications to achieve interoperability between different, heterogeneous systems. These mapping are, however, almost in all cases derived manually. An example of ontology matching applied in end user information access is described in [104]. This system uses a central taxonomy, namely the open directory for webpages (http://www.dmoz.org). It then maps the concepts in personal bookmark folders to concepts in this taxonomy. By coupling a personal concept and a global concept, users get recommendations on new items that are added to the global ontology.

In terms of the above characterization, ontology matching is based on local structures, usually with a local scope. The concepts between different structures are coupled either mutually or via a concept in a global ontology.

Most ontology mapping approaches are targeted at system administrators that maintain heterogeneous information systems, create data warehouses or mash-ups based on web services. These are usually experts. Only a minority of systems is targeted at casual users.

[1] http://wordnet.princeton.edu

3.3.5 Social Tagging and Bookmarking

A key advantage of ontology mapping approaches is that they do not require all users to obey the same structuring scheme. Still, individual structures should be coherent and it should be possible to map between them. Social bookmarking systems go a step further concerning loosely-coupledness. In contrast to the other formalisms discussed so far, they represent a very simple form of information structuring. Users may simply assign arbitrary textual descriptions, called tags, to items. Tags assigned by different users may differ in any possible way. They must not even be consistent for a single user.

By making tags assigned by one user visible to all other users, still a kind of agreement can be reached, as users add additional tags only if they find the existing tags not satisfying. This keeps the vocabulary small, without any explicit coordination among users. Coordination is achieved simply by visibility of what other users did.

Social bookmarking systems are a point of departure for the methods developed in this work and will be discussed in detail in chapter 5.

3.4 Conclusion

Distributed data structuring can be analyzed from a data and from a user point of view. Concerning a data point of view, the aim is to apply methods like data clustering or classification to data that is not stored in a central database but is distributed over a network. The result should be (approximately) the same as processing all data at a central node.

From a user point of view, distributedness does also entail that different users may have different, diverging opinions on how the items in a given domain should be structured. For this heterogeneity, allowing users to collaboratively structure a domain of interest is still a challenging task. On the one hand, each user or group of users should be able to define one or even several views on a domain that can be completely independent of each other and of the views of other users and user groups. On the other hand, a maximum of agreement should be achieved, where this is possible, to make sharing information and communication easier.

This chapter gave an overview of existing approaches to deal with this heterogeneity. On the one hand, there are approaches based on formal ontologies, that allow a group of users to create a common knowledge structure based on collaborative editing or merging. If a common structure is not desirable, ontology mapping and matching approaches allow to make heterogeneous ontologies interoperable. These systems are primarily targeted at experts. On the other hand, there are social bookmarking systems, that allow casual users to structure a domain in an intuitive way by allowing them to attach arbitrarily chosen tags to items. While these systems are very popular, tag collections tend to be hard to manage when the number of tags and items increases. One of the biggest social bookmarking systems, del.icio.us (http://www.del.icio.us), contains currently more than

three million tags. Allowing users to navigate and organize items using this huge tag collection is far from being trivial. The future success of these systems therefore depends on methods that help to organize such tag collections and that support the emergence of common tag structures among users and user groups.

In part two of this work, social bookmarking systems are discussed in more detail. Based on methods of information structuring and distributed feature engineering, these systems will be extended to support users in tagging items by exploiting the tags of other users. This approach allows users to profit from the work of others, still not forcing them to agree on a common structure. In this way, an implicit agreement among groups of users is achieved, that does not require any explicit coordination.

Part II

Problem Definition, Methods and Evaluation

4 Distributed Feature Extraction

4.1 Introduction

Finding an adequate data representation is a key to successful information retrieval and data mining. Such a representation is often expressed in terms of a feature space, the space in which items in the universe of discourse are described. The problem of finding an optimal feature space can be denoted as *representation problem*. Methods that solve this problem are referred to as *feature extraction* or *feature construction* methods [69]. This term covers such diverse approaches as feature extraction from raw data, feature selection, feature space transformation, dimensionality reduction, meta data extraction, etc.

Feature extraction is a very general approach. By constructing appropriate features, simple data mining algorithms can be extended to handle complex concepts, such as trigonometric relationships, without modifying the algorithm [119]. Kernel methods, that have found increasing attention in the recent years, can be regarded as feature transformation methods as well [159]. They do, however, not create the target feature space explicitly but through an implicit mapping. While this can be efficient, it has the severe drawback that the resulting models are hard to interpret. The same holds for dimensionality reduction methods, such as Principal Component Analysis (PCA) or Singular Value Decomposition (SVD) (see e.g. [73]). Interpretable models are, however, crucial in many application areas. This is especially true for information structuring, because user interaction plays a central role in this process. In the following, we will discuss only such methods, that explicitly transform the feature space without altering the original features.

A family of methods that are particularly powerful for feature extraction are so called *wrapper approaches* [100]. The idea of wrapper approaches is to invoke the target (data mining) method several times with different feature sets and to apply an objective function to measure how well suited a feature set is for solving the problem. Optimization algorithms can then be used to select an optimal feature set. If the task is to automatically classify items into predefined classes, for instance, this objective function would be the estimated accuracy of the classifier, thus the expected ability to classify new items correctly. An advantage of wrapper approaches is that they directly optimize an objective function, instead of relying on heuristics that optimize this function indirectly. As wrappers treat the inner data mining task as black-box, these approaches are very generally applicable. Particularly, they do not make any assumptions on the underlying task

and the objective function that is optimized. A downside of this generality is, however, that the optimization process can be computationally very demanding.

In many current scenarios, not a single, separated data mining task is solved, but rather several, partially related ones. Corresponding approaches are called *multi-task learning* [30, 31]. „Partially related“ refers to the fact that these tasks are allowed to differ in any possible way, but that we can hope that some of them resemble each other to some extend. An example for an application in which multi-task learning could be applied, is collaborative media organization. Users arrange their media items (documents, video, audio, etc.) into categories by tagging them. These taggings define data mining tasks, for instance, the task of tagging new items automatically with user defined categories. While it would be possible that the categories defined by different users differ completely, this is not very likely. Many users will, for instance, arrange their music according to genre. These tasks are then *related*. Other, similar application areas include, for instance, robotics (several autonomous agents that recognize similar entities in different environments) or business intelligence (data from different branches of company is analyzed).

Almost all approaches to multi-task learning make very restrictive assumptions. They assume that all learners use the same underlying model class, the same set of features and that all learning tasks are solved at once in parallel. These assumption do not reflect the practical constraints posed by many application domains. First, using the same underlying learning algorithm is a strong practical restriction in areas where many different, heterogeneous components must be connected. Second, using the same feature set for all tasks is often not optimal. Actually, a key to improve the accuracy by which individual tasks can be solved is to select an optimal feature set for each of the tasks, as we will see below. This is especially true if not only feature selection but feature construction is used. Finally, tasks are often solved in some kind of a sequential order instead of being solved all at once. As will be shown below, this is not just an implementation detail but has an essential influence of the solveability of the problem.

In this chapter, a different approach is chosen. Based on the paradigm of wrapper approaches, the problem of selecting appropriate features is analyzed as a *combinatorial optimization problem*. We will analyze how concepts, such as feature relevance, redundancy and minimality, well known from single-task learning, can be generalized to the case of multi-task learning. This will lead to the *distributed representation problem,* thus the problem of finding an optimal local feature set for each task in a set of tasks. It will be shown that constraints on the order in which tasks are solved have an influence on whether we can guarantee to find an optimal solution. Based on this analysis, a simple, heuristic algorithm, called *prioritized forward selection,* is developed. This algorithm is applicable to very general problem classes. We show how this algorithm can be optimized for special cases, such as classification. Also, two frameworks are proposed to implement this algorithm and its optimizations efficiently in a p2p network. Based on the analysis of the algorithm and its variants concerning computation time and message size, we will show that the methods allow for a new solution in the tradeoff between response time, message size and accuracy. Finally, to show the feasibility of the proposed methods, an empirical evaluation is given.

4.2 (Generalized) Feature Relevance and Redundancy

In almost all data mining applications, items are represented by features of some kind. Usually, there are many different ways to describe the entities in an application domain. A multimedia item can, for instance, be described by an infinite number of content related features, by textual features, ratings, etc. Furthermore, existing features can be combined by applying arithmetic operations to them. The choice of features influences the accuracy that can be achieved solving the data mining task to a large extent.

On the one hand, the feature set should contain only such features that are relevant for the task, i.e. omitting this information would lead to an inferior accuracy. On the other hand, the feature set should not contain redundant features, thus features which express the same information already expressed by another feature, or by a set of features in the feature set. The latter point is important for efficiency reasons and to improve the understandability of derived models.

In this section, we will first briefly review feature relevance and redundancy for classification tasks. We will then generalize this framework to cover other data mining tasks as well.

4.2.1 Feature Relevance and Redundancy for Classification Tasks

Classification learning was defined as the task of finding a decision function that automatically assigns items to a finite set of classes, given example assignments. This decision function should be selected in such a way that the expected error with respect to an assumed true relationship is minimized.

An important concept in this context is the one of conditional class probabilities. In the following, we will present this concept with respect to binary features only. It can, however, easily be extended to continuous features.

Definition 4.2.1. The *probability of a feature vector* with respect to a set of binary features $\mathbf{X}$ and an arbitrary item $x \in D$ is denoted by $p(X_1(x) = v_1, .., X_i(x) = v_i, .., X_n(x) = v_n)$ with $\mathbf{X} = \{X_1, ..X_i, .., X_n\}$ and $v_i \in \{0, 1\}$. We use $p(\mathbf{X})$ to express the probability distribution of feature vectors for arbitrary items.

In other words, $p(\mathbf{X})$ describes a probability distribution on the values for the features in $\mathbf{X}$ for all possible items D.

Definition 4.2.2. The *conditional class probability* with respect to a set of binary features $\mathbf{X}$ and an arbitrary item $x \in D$ is denoted by $p(x \in ext(c)|X_1(x) = v_1, .., X_i(x) = v_i, .., X_n(x) = v_n)$ with $\mathbf{X} = \{X_1, ..X_i, .., X_n\}$, $v_i \in \{0, 1\}$, $c \in C$, where C is a finite set of classes. We use $p(C|\mathbf{X})$ to express the probability distribution of classes, given a feature vector.

The conditional class probabilities describe how likely each class occurs, given a vector of feature values.

In [88], a systematic analysis of feature relevance is given, based on the idea of conditional class probabilities. The authors argue that concerning a (classification) task, features can not only be relevant or irrelevant, but relevant features must be further divided into weakly and strongly relevant features. The following formalization follows the presentation in [204].

Definition 4.2.3. A feature $X \in \mathbf{X}$ is s*trongly relevant for classification* with respect to features $\mathbf{X} \subseteq \mathcal{X}$ if

$$p(C|\mathbf{X}) \neq p(C|\mathbf{X}\backslash\{X\})$$

Thus omitting feature X changes the conditional probability distribution concerning the class value.

Definition 4.2.4. A feature $X \in \mathbf{X}$ is *weakly relevant for classification* with respect to features $\mathbf{X} \subseteq \mathcal{X}$ if

$$p(C|\mathbf{X}) = p(C|\mathbf{X}\backslash\{X\})$$

and

$$\exists \mathbf{X}' \subseteq (\mathbf{X}\backslash\{X\}) : p(C|\mathbf{X}' \cup \{X\}) \neq p(C|\mathbf{X}')$$

For features that are weakly relevant, omitting them may lead to a change in the conditional probability, depending on which other features are selected.

Definition 4.2.5. A feature $X \in \mathbf{X}$ is *irrelevant for classification* with respect to features $\mathbf{X} \subseteq \mathcal{X}$ if

$$\forall \mathbf{X}' \subseteq (\mathbf{X}\backslash\{X\}) : p(C|\mathbf{X}' \cup \{X\}) = p(C|\mathbf{X}')$$

If a feature can be omitted without affecting the conditional distribution not regarding which other features are selected, it is irrelevant for the classification task.

Complementary to the view of features as relevant or irrelevant is the concept of feature redundancy. Feature redundancy is usually defined based on the concept of a Markov blanket.

Definition 4.2.6. Given a set of classes C in a classification task, a feature $X \in \mathcal{X}$, a set of features $\mathbf{X} \subseteq \mathcal{X}$ and a subset $\mathbf{X}' \subseteq (\mathbf{X}\backslash\{X\})$ of $\mathbf{X}$ that does not contain X. Then $\mathbf{X}'$ is said to be a *Markov blanket concerning classification* for X if

$$p(\mathbf{X}\backslash(\{X\}\cup\mathbf{X}'), C|\mathbf{X}', X) = p(\mathbf{X}\backslash(\{X\}\cup\mathbf{X}'), C|\mathbf{X}')$$

Thus, a Markov blanket replaces a feature X in such a way that the conditional probability of the class C *and* the remaining features $\mathbf{X}\backslash(\{X\}\cup\mathbf{X}')$ does not change. In other words, the information contained in X does not have an influence of the joint probability of the remaining features and the class, as long as we are given the information in the Markov blanket $\mathbf{X}'$.

Based on this definition, redundant features can be defined as follows.

Definition 4.2.7. A feature $X \in \mathcal{X}$ is *redundant* with respect to a feature set $\mathbf{X} \subseteq \mathcal{X}$ if it is weakly relevant in $\mathbf{X}$ and has a Markov blanket in $\mathbf{X}$.

If we try to learn the concept $f(X_a, X_b, X_c, X_d) \equiv X_a + X_b > 3$ and we know that $X_b = X_c$, then X_d is clearly irrelevant, because it is not related to the target concept in any way. X_a is strongly relevant, because omitting it would not permit to capture the target concept correctly. X_b and X_c are weakly relevant, because each of them could be removed while learning the concept correctly, but not both of them can be removed. In fact, X_b and X_c are redundant to each other.

Note that features can be weakly relevant but still not have a Markov blanket. The reason is that a Markov blanket not only requires them to be replaceable with respect to the class value but also to be replaceable with respect to the other features. Thus, four groups of features can be defined: strongly relevant, weakly relevant but non-redundant, redundant and irrelevant features.

Definition 4.2.8. An *optimal set of features for classification* is a set of features $\mathbf{X} \subseteq \mathcal{X}$ that contains no redundant and no irrelevant features and that minimizes the expected classification error.

4.2.2 Existing Approaches to Single-task Feature Selection

In this section we will give a very brief summary of existing approaches to feature set optimization for single learning tasks. This will be the point of departure to extend these methods to several tasks.

Actually, we have to distinguish two different types of problems.

Definition 4.2.9. *Feature selection* is the problem of finding an optimal set of features $\mathbf{X} \subseteq \mathcal{X}$, where $\mathcal{X}$ is a finite set of given features.

This problem is more specific than the general problem of feature extraction.

Definition 4.2.10. *Feature extraction or construction* is the problem of finding an optimal feature subset $\mathbf{X} \subseteq \mathcal{X}$ from an infinite set of possible features $\mathcal{X}$.

In the following, we will not make a distinction among both on a *conceptual level*, assuming that the number of features that we could reasonably construct is always finite. On an *algorithmic level*, we have to be aware that this set can be extremely large.

Existing approaches to feature selection can be roughly classified in wrappers, filters and embedded methods. *Wrappers* use an inner evaluation function that assesses the estimated accuracy of a classifier. Based on some search strategy, different feature subsets are evaluated until (an optimal) one is found. *Filters* select features without invoking the actual learning process. They mostly produce a ranking of the features, according to their relevance. While these methods are very efficient, they usually cannot capture feature interaction appropriately (like redundant features), because features are regarded only individually and not in combination to each other. *Embedded methods* are feature selection methods that are part of an existing data mining algorithm. An example are decision tree learners [147].

Filter Approaches for Feature Extraction

Most filter approaches aim at classification and regression tasks. They usually measure the relatedness of a given feature to the concept that should be learned. They then select the k features with the highest relatedness. A key advantage of such filter approaches is that they are usually extremely efficient. Most algorithms are linear in the number of features and items in question. See [69], chapter three, for an overview.

A limitation of filters is that they regard each feature individually and thus cannot capture any feature interaction. Especially, they tend to select several weakly relevant features. An extreme case are two identical features, that would obtain the same score.

Other filter approaches, such as RelieF [97], are more sophisticated, though less efficient.

Wrapper Approaches for Feature Extraction

Wrapper approaches treat the inner data mining task as black box and assume only that there is a possibility to assess the accuracy by which a certain data mining task can be solved. Searching for an optimal subset then becomes a straight forward optimization problem. The task is to select a subset $\mathbf{X} \subseteq \mathcal{X}$ with optimal performance with respect to an arbitrary objective function q (this will be defined in a formal way below, see definition 4.2.12). In general, it would be possible to evaluate all possible subsets. This can be denoted as *optimal feature selection*, because it guarantees to yield an optimal and minimal solution. If we do not make any restrictive assumptions about the objective function, then the process of finding an optimal feature set involves in the worst case the evaluation of $2^{|\mathcal{X}|}$ feature subsets. Based on this observation, it can be shown that the problem of optimal feature selection is NP-complete, in general, by reducing it to the boolean satisfiability problem.

Therefore, several heuristics were developed to solve the representation problem more efficiently. Probably best known are *forward selection* and *backward elimination*. The forward selection algorithm is shown in figure 4.2.1. In each round, the algorithms adds a feature to the current feature set that increases the accuracy q maximally. If no

Input:
A set $\mathcal{X}$ of features from which to select
A task with evaluation function q

Output:
A subset of features $\mathbf{X} \subseteq \mathcal{X}$ for the task

$\mathbf{X} = \emptyset$;
$\mathbf{X}_{open} = \mathcal{X}$;
$max = q(\emptyset)$;
repeat
  $improve = false$;
  for $X \in \mathbf{X}_{open}$ **do**
    $score = q(\mathbf{X} \cup \{X\})$;
    if $score > max$ **then**
      $max = score$;
      $X_{max} = X$;
      $improve = true$;
    end if
  end for
  if $improve$ **then**
    $\mathbf{X} = \mathbf{X} \cup \{X_{max}\}$;
    $\mathbf{X}_{open} = \mathbf{X}_{open} \setminus \{X_{max}\}$;
  end if
until $((\mathbf{X}_{open} = \emptyset) \vee \neg improve)$

Figure 4.2.1: Forward Selection: In each step, it is evaluated, whether adding one of the not yet selected features increases the accuracy. A feature that increases the accuracy maximally is then added to the feature set. If no such feature exists, the process terminates.

such feature exists or if all features have been added to the feature set, the algorithm terminates.

It is well known and easy to show, that forward selection has a worst-case complexity of $O(|\mathcal{X}|^2)$ in the number of necessary evaluations.

Lemma 4.2.11. *Forward selection has a worst-case complexity of $O(|\mathcal{X}|^2)$ in the number of evaluations.*

Proof. Sketch: In each round, the number of not yet selected features is reduced by one, therefore the maximal number of rounds is $|\mathcal{X}|$. In each step, all remaining features have to be tried, upper-bounded by $|\mathcal{X}|$. □

Backward elimination starts with a given feature set $\mathcal{X}$ and removes in each round the

feature for which the resulting feature set leads to an optimal performance. If no such feature exists or if all features were deselected, the algorithm terminates.

Beside these two basic approaches, several other search strategies are possible, for instance evolutionary approaches, simulated annealing, beam search, etc. See [69], chapter 4, for a recent overview.

Many approaches to feature selection can be extended to the case of feature construction as well. Because the number of features that can be constructed is not finite (or at least very large), it is usually necessary to provide an additional operation that delivers for each set of given features $\mathbf{X} \subseteq \mathcal{X}$ a finite set of candidate features $\mathbf{X}_{\text{cand}}$, such that $\mathbf{X} \cap \mathbf{X}_{\text{cand}} = \emptyset$. Instead of selecting from the remaining features in each search step (like in feature selection), the algorithm selects from this finite candidate set. This approach can be referred to as *forward generation*. Other approaches, such as genetic search strategies, can be extended in a similar way.

Most approaches to feature construction restrict the space of possible features to be searched in several ways. First, a threshold can be set, such that if a task is solved by at least a minimal accuracy q_{min}, the procedure terminates. Second, the number and kinds of possible feature sets that are regarded can be constrained. Finally, a common strategy is to use a time-constraint, accepting the best solution that can be found in some predefined period of time. This is mostly what happens in practice.

An important question for all wrapper approaches is how to assess the accuracy by which an inner data mining task is solved. For classification problems, cross-validation can be used to estimate the expected error of the classifier (see section 1.4.4). An extension to this basic approach is the use of multi-objective optimization, thus applying two or more different criteria. For instance, it is possible to select a feature set that optimizes the accuracy *and* minimizes the number of features *not preserving the accuracy* [50]. This leads to a set of Pareto-optimal solutions and the user can choose between a minimal set of features or a set of features that optimizes the accuracy.

Another application area for multi-objective methods for feature selection and construction is clustering. Feature selection for clustering is much more challenging, because unsupervised machine learning differs essentially from supervised learning. The aim is usually rather to *describe* the data set, and thus to automatically find inherent, natural patterns in the data.

Feature space transformation is important for unsupervised learning as well. Noise, sparsity and redundancy can hide the natural patterns in a data set, just as they can hide the relationship of the data points to a target function in supervised learning. Feature space transformation for unsupervised learning can be performed manually based on domain knowledge. This procedure is very time and labor intensive, especially as unsupervised learning is often applied for data exploration and few or nothing is known about the data and the domain.

Several multi-objective optimization schemes for unsupervised feature selection were proposed in [94, 95, 129]. These approaches minimize the number of features. Simultane-

ously, the accuracy of the identified patterns should be maximized. This idea is directly transferred from supervised feature selection. While these approaches are very promising, they are limited in two points. First, minimizing the number of features, just as in supervised learning, is not robust for the unsupervised setting. Under very weak assumptions we can show that the set of Pareto optimal solutions collapses into one singular point that represents a trivial solution [122]. The accuracy of a clustering algorithm, for example, can trivially be optimized by selecting a single feature only. This, however, leads in virtually every case to a completely inadequate and useless description of the original data set.

In [122] an improved approach is proposed. This approach is based on maximizing the accuracy of the clustering and the number of clusters simultaneously. The accuracy of clusters is measured by within-cluster distance cq_{wcd}, as described in section 1.4.5. This approach leads to much more complete sets of solutions than the existing approaches and is robust against nominal features. In [123] this framework is extended to cover feature construction as well.

4.2.3 Generalized Feature Relevance and Redundancy

Almost all existing work on feature relevance and redundancy has been focusing on classification tasks. The concept is, however, much more general and can be applied to other tasks as well. Following the paradigm of wrapper approaches, we give a very general definition of a data mining task. We consider a data mining task a black box that gets a feature set as input and delivers a quality measure as output. We will denote this quality as *accuracy* in the following.

Definition 4.2.12. A *data mining task* $t \in T$ is any task for which we can define an accuracy measure $q : 2^{\mathcal{X}} \to R$ that depends on the subset of features $\mathbf{X} \subseteq \mathcal{X}$ used to solve the task. R denotes an ordered set. In the following $R = \mathbb{R}$ is assumed. T denotes a set of data mining tasks.

Given a set of features $\mathbf{X} \subseteq \mathcal{X}$ we define the following.

Definition 4.2.13. A feature $X \in \mathbf{X}$ is s*trongly relevant* with respect to features $\mathbf{X} \subseteq \mathcal{X}$ if

$$\forall \mathbf{X}' \subseteq \mathbf{X} \backslash \{X\} : q(\mathbf{X}') < q(\mathbf{X}' \cup \{X\})$$

Thus omitting the feature X leads always to a performance that is inferior than the one achieved if X is part of the feature space.

Definition 4.2.14. A feature $X \in \mathbf{X}$ is *weakly relevant* with respect to features $\mathbf{X} \subseteq \mathcal{X}$ if it is not strongly relevant and

$$\exists \mathbf{X}' \subseteq \mathbf{X} : q(\mathbf{X}' \backslash \{X\}) < q(\mathbf{X}')$$

For features that are weakly relevant, omitting them may lead to a decrease in performance dependent on the which other features are used for data mining. A typical case for weakly relevant features are features that are alternative.

Definition 4.2.15. Two features $X \in \mathcal{X}$ and $X' \in \mathcal{X}$ are called *alternative,* denoted as $X \sim X'$ if $X(x) = \alpha + \beta \cdot X'(x)$ for all $x \in D$ with $\beta > 0$.

Many learners can easily deal with linear transformations of features (e.g. SVM [73]). For this reason, both features X and X' are not strongly relevant, because either one can be omitted, as long as the other one is used.

Definition 4.2.16. A feature $X \in \mathbf{X}$ is *irrelevant* with respect to features $\mathbf{X} \subseteq \mathcal{X}$ if

$$\neg\exists \mathbf{X}' \subseteq \mathbf{X} : q(\mathbf{X}'\backslash\{X\}) < q(\mathbf{X}')$$

If a feature can be omitted without affecting the accuracy not regarding which other features are used, it is irrelevant for the task, because it can be omitted under all circumstances.

The concept of feature redundancy can be captured in a more general way as well.

Definition 4.2.17. Given a feature $X \in \mathcal{X}$, a set of features $\mathbf{X} \subseteq \mathcal{X}$ and a subset of this set $\mathbf{X}' \subseteq (\mathbf{X}\backslash\{X\})$ then $\mathbf{X}'$ is said to be a *Markov blanket* for X, denoted as $mb(\mathbf{X}', X)$ if

$$p(\mathbf{X}\backslash(\{X\}\cup\mathbf{X}')|\mathbf{X}', X) = p(\mathbf{X}\backslash(\{X\}\cup\mathbf{X}')|\mathbf{X}')$$

This definition is the same as for the classification case (Def. 4.2.6), we do, however, not make any assumption on whether there is class information given, or not.

Again, a feature $X \in \mathbf{X}$ is redundant if it has a Markov blanket in $\mathbf{X}$.

Definition 4.2.18. An *optimal feature set* is such a set of features $\mathbf{X} \subseteq \mathcal{X}$ that $q(\mathbf{X})$ is maximized and $\mathbf{X}$ does not contain redundant and irrelevant features with respect to $\mathbf{X}$.

This is a generalization of definition 4.2.8 that covered only feature selection for classification. By definition, irrelevant features cannot lead to an improvement in performance. As redundant features are replaceable by their Markov blanket, their removal should have no impact on the performance as well.

We can still give a slightly stronger characterization, by referring to the concept of a minimal and optimal feature set.

Definition 4.2.19. An *optimal, minimal feature set* is such a set of features $\mathbf{X} \subseteq \mathcal{X}$ that $q(\mathbf{X})$ is maximized and there is no $\mathbf{X}' \subseteq \mathcal{X}$, such that $q(\mathbf{X}) = q(\mathbf{X}')$ and $|\mathbf{X}'| < |\mathbf{X}|$.

This is also sometimes referred to as *minimal sufficient feature subset* for the case of classification (see e.g. [69], page 20). It is obvious, that this definition covers the above definition of a optimal feature set as well.

It is not necessary to state that a minimal feature set should not contain any redundant or irrelevant features, because this is entailed by the minimality condition.

4.3 Feature Relevance and Redundancy for Several Tasks

In traditional scenarios, only a single task is considered. In many current scenarios, this is not adequate. Rather we often face several related data mining tasks that are solved in parallel or subsequentially. This problem is often denoted as *multi-task learning* [30, 31]. The idea is that it could be easier to learn several tasks, that are related to each other, at once, than learning them individually. This assumption is based on the observation that often several learning tasks in the same domain resemble each other to some extent and that parameters to solve these tasks resemble each other as well. First approaches to multi-task learning used, for instance, a common set of inner neurons in a neural net which is shared among several tasks [30]. These common neurons encode common parameters among different tasks. More current methods for multi-task learning make use of SVM, kernel methods [54, 83] and Gaussian processes [203] to achieve a corresponding effect. The idea in all cases is to have a class of models (for instance SVM) and several learning tasks. Then a distribution among the parameters of the models is assumed that allows to group tasks into clusters of tasks with similar parameters. There are several variants to this basic idea. In [8] it is discussed how unlabeled data can be incorporated into the process of multi-task learning. [205] discusses the problem of outlier tasks and how to robustify existing approaches to multi-task learning. Finally, [10] proposes an interesting variant to the problem by allowing to calculate a small number of artificial features that are common to all tasks.

All of these approaches make several assumptions. First, only classification tasks are regarded, with the standard expected loss function. Second, all approaches assume that the same model class is used for each of the tasks. Third, a common set of features is assumed for all learning tasks. Forth, all approaches assume that model parameters cluster well. While [205] considers outliers, it is still assumed that clusters can be found. Finally, it is assumed that all tasks are solved at once in a single optimization process.

In the following, we take a more general view on this problem. The idea is to make none of the assumptions above, but to assume only the minimalistic definition of a data mining task given above. Tasks are only related in that the solution of each task relies on a subset of features $\mathbf{X} \subseteq \mathcal{X}$ of the same superset of features $\mathcal{X}$. Which subset $\mathbf{X}$ of features is relevant to solve a given task and how an optimal model looks like may differ for each of the tasks in any possible way. This allows us to mix not only different data mining algorithms but even different data mining tasks. Also, we do not make any restrictive assumption on how the tasks clusters. Finally, we also analyze the influence of the order in which the tasks are solved, which is a crucial problem in practice.

A downside of making only minimal assumptions is that it is harder to obtain specific analytical results or to find optimizations. As will be shown, analyzing the problem on an abstract level will still yield several interesting insights. Also, by making some more assumptions later, it will be possible to optimize the basic approach proposed here.

A set of data mining tasks will be denoted as T. Each task $t_i \in T$ is connected to a accuracy function q_i, as described above.

Definition 4.3.1. The *overall accuracy of a several data mining tasks* T is the sum over all individual qualities

$$q^*(\mathbf{X}_1, .., \mathbf{X}_\mathrm{i}, ., \mathbf{X}_{|\mathrm{T}|}) = \sum_{i=1}^{|T|} q_i(\mathbf{X}_\mathrm{i})$$

where $\mathbf{X}_\mathrm{i} \subseteq \mathcal{X}$ denotes the feature set used to solve task t_i and q_i the corresponding accuracy.

Given more than one data mining task, we can generalize the notion of feature relevance, redundancy and optimality.

To do so, we regard the accumulated feature set that contains exactly the features that are used for at least one task in T.

Definition 4.3.2. The *accumulated set of features,* given a set of tasks T, denoted as $\mathbf{X}_T$, is the union of all locally selected features

$$\mathbf{X}_\mathrm{T} = \bigcup_{i=1}^{|T|} \mathbf{X}_\mathrm{i}$$

where $\mathbf{X}_\mathrm{i} \subseteq \mathcal{X}$ denotes the feature set used to solve task t_i.

Assume an application that learns personal preferences of users concerning music. Also assume that we can extract three features from each music clip that represent, for instance, loudness X_l, rhythm X_r and color X_c of the music, thus $\mathcal{X} = \{X_l, X_r, X_c\}$. Each user defines a learning task by tagging music clips with a positive or a negative label, according whether she likes them, or not. This results in one learning task per user, for which we assume a true but unknown concept f_i. For each of the users, another subset of $\mathcal{X}$ may be relevant. For the first user, preferences may only depend on the loudness, which should be high, thus $f_1 \equiv X > 2.3$. For a second user, preference may depend on the loudness and the rhythm, e.g. $f_2 \equiv X_l + X_r > 3$. For a third user, the preferences could depend on the loudness again, which should, however, be small in this case, e.g. $f_3 \equiv X_l < 1$. The task is to find for each user a subset of features and a predictive model that captures her preferences in a way that allows predictions with a high accuracy. If the true concepts were known, optimal feature subsets $\mathbf{X}_1 = \{X_l\}$, $\mathbf{X}_2 = \{X_l, X_r\}$ and $\mathbf{X}_3 = \{X_l\}$ could be selected easily. The accumulated set of features is then $\mathbf{X}_\mathrm{T} = \{X_l, X_r\}$. In a real application, we can only approximate the real concepts f_i by hypothesis, as described above.

We can now generalize the notions of strongly and weakly relevant features in different ways by regarding properties of features that hold for all, some or for none of the tasks.

Table 4.1 gives an overview.

	$\forall t \in T$	$\exists t \in T, \neg\forall t \in T$	$\neg\exists t \in T$
irrelevant	globally irrelevant	relevant	relevant
redundant	globally redundant	partially redundant	not part. redundant
weakly rel.	globally weakly rel.	globally weakly rel.	not weakly rel.
strongly rel.	globally strongly rel.	globally strongly rel.	not strongly rel.

Table 4.1: This table gives an overview of how feature relevance and redundancy can be generalized to several tasks.

Definition 4.3.3. A feature $X \in \mathbf{X}_\mathrm{T}$ is *globally strongly relevant* with respect to features $\mathbf{X}_\mathrm{T}$ if it is strongly relevant for at least one task $t \in T$.

If X is deleted from $\mathbf{X}_\mathrm{T}$ the overall accuracy decreases, because it must by definition decrease for at least one task.

Definition 4.3.4. A feature $X \in \mathbf{X}_\mathrm{T}$ is *globally weakly relevant* with respect to features $\mathbf{X}_\mathrm{T}$ if it is not globally strongly relevant and is weakly relevant at least for one task $t \in T$.

Globally weakly relevant features can be omitted, as long as there are alternative features left that replace them in a way that does not affect the global accuracy.

Definition 4.3.5. A feature $X \in \mathbf{X}_\mathrm{T}$ is *globally irrelevant* with respect to features $\mathbf{X}_\mathrm{T}$ if it is irrelevant for every task.

Globally irrelevant features can be omitted from $\mathbf{X}_\mathrm{T}$ in any case without affecting the accuracy at all, because, by definition, they do not affect the accuracy of any individual data mining task.

Now, we can extend the notion of redundancy and Markov blankets to the case of several tasks. We have different options here. A feature can be regarded as redundant if it is redundant for at least one task, for all tasks or with respect to the accumulated feature set $\mathbf{X}_\mathrm{T}$.

Definition 4.3.6. A feature $X \in \mathbf{X}_\mathrm{T}$ is *globally redundant* if it contains a Markov blanket for each individual task $t \in T$, thus:

$$\forall i : 1 \leq i \leq |T| : \exists \mathbf{X}'_\mathrm{i} \subseteq (\mathbf{X}_\mathrm{i} \backslash \{X\}) : mb(\mathbf{X}'_\mathrm{i}, X)$$

If a feature is redundant for only some tasks, this is denoted as partially redundant.

Definition 4.3.7. A feature $X \in \mathbf{X}_\mathrm{T}$ is *partially redundant* if it contains a Markov blanket for some task $t \in T$:

$$\exists i : 1 \leq i \leq |T| : \exists \mathbf{X}'_\mathrm{i} \subseteq (\mathbf{X}_\mathrm{i} \backslash \{\mathrm{X}\}) : mb(\mathbf{X}'_\mathrm{i}, X)$$

Note, that these Markov blankets are not necessarily the same for all tasks.

Finally, we can define redundancy against the accumulated feature set.

Definition 4.3.8. A feature $X \in \mathbf{X}_\mathrm{T}$ is *redundant with respect to the accumulated feature set* $\mathbf{X}_\mathrm{T}$ if

$$\exists \mathbf{X}'_\mathrm{T} \subseteq (\mathbf{X}_\mathrm{T} \backslash \{X\}) : mb(\mathbf{X}'_\mathrm{T}, X)$$

How are these definitions related? Obviously, if a feature is globally redundant, it is also locally redundant. Furthermore, we can show the following relationship.

Lemma 4.3.9. *If a feature $X \in \mathbf{X}_T$ is partially redundant, then it is also redundant with respect to $\mathbf{X}_T$.*

Proof. There is a task $t_i \in T$ and a $\mathbf{X}'_\mathrm{i} \subseteq \mathbf{X}_\mathrm{i} \backslash \{X\}$ such that $mb(\mathbf{X}'_\mathrm{i}, X)$. $\mathbf{X}'_\mathrm{i}$ is also a Markov blanket for X in $\mathbf{X}_\mathrm{T}$, as $\mathbf{X}'_\mathrm{i} \subseteq \mathbf{X}_\mathrm{i} \subseteq \mathbf{X}_\mathrm{T}$. □

Therefore, being redundant with respect to $\mathbf{X}_\mathrm{T}$ is the weakest possible restriction.

It is easy to show that if $\mathbf{X}_\mathrm{T}$ does not contain any redundant features, none of the individual feature sets can contain redundant features.

Lemma 4.3.10. *If a feature $X \in \mathbf{X}_T$ is non-redundant in $\mathbf{X}_T$, then it is non-redundant for all tasks in T.*

Proof. By definition, there is no $\mathbf{X}' \subseteq \mathbf{X}_\mathrm{T} \backslash \{X\}$, such that $mb(\mathbf{X}', X)$ (there is no Markov blanket in $\mathbf{X_T}$**).** Then, there cannot be a Markov blanket for X in any $\mathbf{X}_\mathrm{i}$, because for all tasks $t_i \in T$, $\mathbf{X}_\mathrm{i} \subseteq \mathbf{X}_\mathrm{T}$ holds. □

Please note that the other direction does not hold. The fact that all local feature sets do not contain any redundant features does not entail that the accumulated feature set does not contain redundant features.

Given these definitions, we can now define optimal feature sets for more than one task.

Definition 4.3.11. A *set of optimal feature sets* selects for each task $t_i \in T$ a set of features $\mathbf{X}_\mathrm{i} \subseteq \mathcal{X}$ in a way that q^* is maximized and the accumulated feature set $\mathbf{X}_\mathrm{T}$ does not contain any features that are redundant or irrelevant with respect to $\mathbf{X}_\mathrm{T}$ and T.

Again, this can be strengthened to the concept of an optimal minimal set of feature sets.

Definition 4.3.12. A *set of optimal, minimal feature sets* selects for each task $t_i \in T$ a set of features $\mathbf{X}_\mathrm{i} \subseteq \mathcal{X}$ in a way that $q^*(\mathbf{X}_1, ..., \mathbf{X}_{|\mathrm{T}|})$ is maximized and that there does not exist another set of features $\mathbf{X}'_1 ... \mathbf{X}'_{|\mathrm{T}|}$ for each task, such that $q^*(\mathbf{X}_1, ..., \mathbf{X}_{|\mathrm{T}|}) = q^*(\mathbf{X}'_1, ..., \mathbf{X}'_{|\mathrm{T}|})$ and $|\mathbf{X}_\mathrm{T}{}'| < |\mathbf{X}_\mathrm{T}|$, where $\mathbf{X}'_\mathrm{T} = \bigcup_{i=1}^{|T|} \mathbf{X}'_\mathrm{i}$.

Why should we bother to find a minimal optimal feature set? Assume an example in which a first agent tries to induce the binary concept given by $f_1(X_a, X_b, X_c, X_d) \equiv X_a + X_b > 4$ from the data, a second agent tries to induce $f_1(X_a, X_b, X_c, X_d) = X_b + X_d < 2$. We assume that $X_b = X_c$ $(\forall x \in D : X_b(x) = X_c(x))$. In this case, it would

be perfectly reasonable, from an accuracy point of view, to select $\mathbf{X}_1 = \{X_a, X_b\}$ and $\mathbf{X}_2 = \{X_c, X_d\}$. Both feature sets are locally optimal and do not contain redundant features. The accumulated feature set $\mathbf{X}_\mathrm{T} = \{X_a, X_b, X_c, X_d\}$ does, however, contain redundancy. This is not desirable for several reasons. First, if the feature sets are inspected by a human user, it is hard for her to see the relationship between the tasks. Assume an analyst that tries to discovery patterns in several branches of a large company. It is much easier to recognize patterns or influence factors known from another case than being faced with completely new features. Second, selecting different features can lead to a higher storage requirements. Finally, as we will see, selecting redundant features leads also to higher communication costs if features are shared among agents. Thus, it would be much better if both agents chose X_b (or X_c) leading to an accumulated feature set $\mathbf{X}'_\mathrm{T} = \{X_a, X_b, X_d\}$ or $\mathbf{X}''_\mathrm{T} = \{X_a, X_c, X_d\}$ respectively, both of which do not contain any redundant features.

4.4 Distributed Feature Extraction

In this section, several approaches for distributed feature extraction are proposed, that all aim at finding optimal subsets of features for each task in a set of tasks. We use single-task feature selection as point of departure and discuss how these methods can be extended to be applicable to the case of several data mining tasks. A key point is to share features among agents in order to improve the efficiency of the data mining process. The idea is that features found to be successful for some of the tasks, can be promising for new tasks as well. Tasks can be heterogeneous. Therefore, a crucial optimization of this approach is not to regard all features that were extracted or generated by any other agent, but to retrieve only the relevant ones.

4.4.1 The Problem of Distributed Feature Extraction

In the following, we will analyze how methods for single task feature extraction can be generalized to the problem of feature extraction given several tasks. As described above, existing solutions to multi-task learning make restrictive assumption on the underlying model classes, the data mining tasks and how they are related. In the following, we seek general solutions that do not depend on these assumptions. These solutions should be especially applicable to different models and data mining tasks, and they should not force all tasks to use the same features (indeed they should allow to construct special features for individual tasks).

This leads to the basic problem of distributed feature extraction.

Definition 4.4.1. *Distributed feature extraction* denotes the task of finding for each task $t_i \in T$, an optimal feature set $\mathbf{X}_\mathrm{i} \subseteq \mathcal{X}$, such that $q^*(\mathbf{X}_1, .., \mathbf{X}_i, ..\mathbf{X}_{|T|})$ is maximized and $|\mathbf{X}_T|$ is minimized.

A very obvious way to do distributed feature extraction would be to simply use any optimization or feature transformation algorithm to optimize the feature set for each task individually. There are two reasons why not to do so:

1. It would not necessarily yield an accumulated feature set that is optimal *and* minimal. This is, however, a desirable property, as was shown above.
2. Solving several learning tasks in a sequence or in parallel can be computationally less expensive than solving them independently if it is possible to share information among tasks. In the remainder of this chapter we will show different methods of how to do so.

In this section we present a simple, generalized forward selection algorithm that will be the point of departure for further optimizations. First, however, we must further specify the scenario we are talking about in terms of the temporal order in which tasks are solved. While for stating the distributed representation problem the order in which the tasks are solved did not have any significance, this order plays an important role for solving it.

In the following, we assume an order relation on tasks that denotes that one task must be solved before another one is solved. In many applications, such an order is implicit. Assume, for instance, the media organization examples. Usually, not all user will classify their media items in parallel. Still, the system has to deliver a solution for the first users, even if the tasks that will be submitted by other users are still not known. This can be regarded as constraint on the order in which the tasks must be solved.

Definition 4.4.2. The *temporal order relation for tasks* $\prec \subseteq T^2$ represents constraints on the order in which the tasks have to be solved, i.e. $t_i \prec t_j$, denotes that task t_i must be solved before t_j is solved, which, as a consequence, means that the feature set $\mathbf{X}_\mathrm{i} \subseteq \mathcal{X}$ must be chosen before $\mathbf{X}_\mathrm{j} \subseteq \mathcal{X}$ is chosen. $\prec$ is assumed to be transitive and anti-symmetric.

In many applications, $\prec$ can be assumed to be a total order as well, which leads to the concept of a sequence of tasks.

Let us first consider the option of performing optimal feature selection at each agent independently. If we do not consider the restriction of minimality, this would already solve the problem.

Lemma 4.4.3. *Given a set of tasks T, selecting $\mathbf{X}_i \subseteq \mathcal{X}$ by optimal feature selection for each $t_i \in T$ yields an overall optimal set of feature sets $\mathbf{X}_1 ... \mathbf{X}_{/T/}$.*

Proof. The accuracy is measured by $q^*(\mathbf{X}_1, .., \mathbf{X}_\mathrm{i}, .., \mathbf{X}_{|\mathrm{T}|}) = \sum_{i=1}^{|T|} q_i(\mathbf{X}_\mathrm{i})$ and optimizing each q_i also optimizes q^*. □

Obviously, optimal feature selection for several tasks inherits its computational complexity from optimal feature selection for single tasks and is therefore NP-complete as well.

Also, different agents may select different weakly relevant features, making the accumulated set of features larger than necessary. Thus, this procedure does not yield solutions that fulfill the minimality condition.

It may seem that this problem can be solved by simply imposing some kind of total priority order on the features in $\mathcal{X}$, and then to select features that are equally well suited to solve the data mining task always according to this preference order. This, however, does not solve the problem of minimality.

Lemma 4.4.4. *Given a set of tasks T, selecting $\mathbf{X}_i \subseteq \mathcal{X}$ by optimal feature selection for each $t_i \in T$ does not yield a minimal accumulated feature set $\mathbf{X}_T$, in general.*

Proof. We give a counterexample. Consider the following two learning tasks: $f_1(X_a, X_b) \equiv (X_a < 4)$ and $f_2(X_a, X_b) \equiv X_b > 10$. Also assume, that $X_a = X_b$, for $X_b < 4$ and $X_a = 4$, else. For the first task, it does not make any difference if we choose X_a or X_b. For the second learning task, we must choose X_b. Let us assume that for the first task X_a is selected, because this feature has a higher priority than X_b. For the second task X_b has to be chosen. The resulting feature set is $\{X_a, X_b\}$, which is suboptimal, because $\{X_b\}$ would lead to the same performance with a smaller feature set. □

A simple approach to find a minimal subset of accumulated features is to solve all tasks at once. Let us assume an agent that enumerates all possible subsets $\mathbf{X} \subseteq \mathcal{X}$. For each subset $\mathbf{X}$, optimal feature selection is applied for each task separately and the accumulated accuracy and the size of the accumulated feature set is measured. Similar to optimal feature selection for a single task, this would also lead to an optimal and minimal feature set for several tasks (although it would be computationally extremely expensive).

However, in many applications we cannot assume that all tasks are solved at once. Mostly, tasks are solved in some kind of a temporal order. Again, assume the collaborative media organization example which is the focus of this work. Usually not all users organize their items at the same time. Also, in a real world scenario, new tasks are added continuously. The same holds for business intelligence applications. Data to be analyzed usually emerges over time and is not available at once. Still, the system has to output some result for the first task it faces.

Given constraints on the order in which task are solved, can we still find an optimal and minimal feature set? Surprisingly, the answer is no, in general. In fact, there is no algorithm that guarantees to find such a set in general, given at least two tasks $t_i, t_j \in T$ that are in temporal relation to each other.

Theorem 4.4.5. *There is no algorithm that can guarantee to find an optimal, minimal set of feature sets, given that $\exists t_i, t_j \in T : t_i \prec t_j$.*

Proof. We construct a counter example. Assume that the feature selection algorithm would first face t_i, for which the concept $f_i(X_a, X_b) \equiv (X_a < 4)$ should be induced. Again assume that $X_a = X_b$, for $X_b < 4$ and $X_a = 4$, else. For this task, either $\{X_a\}$

or $\{X_b\}$ can be chosen to optimize it. Let us assume that $\{X_a\}$ is chosen. The second task is t_j and the concept $f_j(X_a, X_b) \equiv X_b > 10$ is to be captured. In this case, $\{X_b\}$ must be chosen. This leads to a non-minimal overall feature set $\mathbf{X}_\mathrm{T} = \{X_a, X_b\}$. If for t_i $\{X_b\}$ would have been chosen, then we could construct a similar example leading to the non-minimal subset $\mathbf{X}_\mathrm{T} = \{X_a, X_b\}$. Thus any choice for the first task can later turn out to be suboptimal. Therefore, no algorithm that solves t_i and t_j in sequential order can guarantee to yield a minimal and optimal set of features $\mathbf{X}_\mathrm{T}$. □

4.4.2 Prioritized Forward Selection

While this is somewhat discouraging, it still allows us to look for heuristic approaches. In the following, an extension to forward selection is proposed, that is denoted as *prioritized forward selection*. If we solve a task t, we first take a look at all tasks $T' \subseteq T$ that are already solved. This leads to the accumulated feature set $\mathbf{X}_{\mathrm{T}'} = \bigcup_{t_i \in T'} \mathbf{X}_\mathrm{i}$ containing all features used in at least one task, already solved or currently solved. Then we perform forward selection twice. First, we perform forward selection using only the features in $\mathbf{X}_{\mathrm{T}'}$. Only after no further improvement is possible, we perform a second forward selection on the remaining features in $\mathcal{X}$. This can be combined with a threshold $\varepsilon \in \mathbb{R}$, such that new features are only added if this increases the performance above a certain level.

Figure 4.4.1 shows the algorithm. The computational complexity of this algorithm is the following.

Lemma 4.4.6. *The computational complexity of prioritized forward selection is $O(|\mathcal{X}|^2|T|)$ in the number of inner evaluations.*

Proof. For each task, two forward selection problems are solved, one on the feature set accumulated so far, the other one on the remaining features in $\mathcal{X}$. This leads to the following upper bound $|\mathbf{X}_{\mathrm{T}'}|^2 + |\mathcal{X} \backslash \mathbf{X}_{\mathrm{T}'}|^2 \leq (|\mathbf{X}_{\mathrm{T}'}| + |\mathcal{X} \backslash \mathbf{X}_{\mathrm{T}'}|)^2$. Trivially, $(|\mathbf{X}_{\mathrm{T}'}| + |\mathcal{X} \backslash \mathbf{X}_{\mathrm{T}'}|)^2 = |\mathcal{X}|^2$ for all $\mathbf{X}_{\mathrm{T}'} \subseteq \mathcal{X}$. This procedure is repeated exactly once for each task, and therefore has the complexity given above. □

Prioritized forward selection serves two purposes. First, it helps to find small accumulated feature sets, which is an important property when facing feature extraction for several tasks. A second purpose, at least equally important, is that efficiency and even accuracy can be increased.

The idea is the following. We assume that in most application areas some of the tasks in T resemble each other to some extend and require similar feature sets to be solved optimally. If good solutions for some tasks were already identified, applying these „know-to-work-well“ features on new tasks seems quite promising, because if the new task is similar to an existing one, the set $\mathbf{X}_{\mathrm{T}'}$ will already contain all relevant features. Selecting features from $\mathbf{X}_{\mathrm{T}'}$ is always more efficient than selecting from a set of possible features $\mathcal{X}$, because $\mathbf{X}_{\mathrm{T}'} \subseteq \mathcal{X}$. Especially, if we consider feature construction, where $\mathcal{X}$ might

Input:
A set $\mathcal{X}$ of features from which to select and a set $\mathbf{X}_{cand} \subseteq \mathcal{X}$ of candidate features
A task t_i with evaluation function q_i

Output:
A subset of features $\mathbf{X}_i \subseteq \mathcal{X}$ for task t_i

$\mathbf{X}_i = \emptyset$;
$\mathbf{X}_{open} = \mathbf{X}_{cand}$;
$max = q(\emptyset)$;
repeat
  $improve = false$;
  for $X \in \mathbf{X}_{open}$ **do**
    $score = q_i(\mathbf{X}_i \cup \{X\})$;
    if $score > max$ **then**
      $max = score$; $X_{max} = X$;
      $improve = true$;
    end if
  end for
  if $improve$ **then**
    $\mathbf{X}_i = \mathbf{X}_i \cup \{X_{max}\}$;
    $\mathbf{X}_{open} = \mathbf{X}_{open} \setminus \{X_{max}\}$;
  end if
until $((\mathbf{X}_{open} = \emptyset) \vee \neg improve)$
$max = max + \varepsilon$;
$\mathbf{X}'_{open} = \mathcal{X} \setminus \mathbf{X}_{cand}$;
repeat
  $improve = false$;
  for $X \in \mathbf{X}'_{open}$ **do**
    $score = q_i(\mathbf{X}_i \cup \{X\})$;
    if $score > max$ **then**
      $max = score$; $X_{max} = X$;
      $improve = true$;
    end if
  end for
  if $improve$ **then**
    $\mathbf{X}_i = \mathbf{X}_i \cup \{X_{max}\}$;
    $\mathbf{X}'_{open} = \mathbf{X}'_{open} \setminus \{X_{max}\}$;
  end if
until $((\mathbf{X}'_{open} = \emptyset) \vee \neg improve)$

Figure 4.4.1: Prioritized Forward Selection for task t_i: In a first forward selection, only the candidate features are tried. Only if no further improvement is reached, the algorithm tries additional features from the set $\mathcal{X} \setminus \mathbf{X}_{cand}$. A threshold ε can be used to allow only to add features that lead to an improvement of at least ε over selecting the features from $\mathbf{X}_{cand}$ only.

contain millions of possible features, this can make the problem for subsequent tasks much easier to solve.

Furthermore, it may even lead to better solutions. As stated above, users often terminate the feature extraction process after a given amount of time, or after the performance is „good enough“, thus a given accuracy threshold is exceeded. Finding relevant features as early as possible is therefore crucial, because it can speed up the process of finding a good solution. Given a maximum number of evaluations a user is willing to allow, it can even improve the solutions. If good solutions are evaluated late in the feature extraction process, they might not be found at all if the process is terminated early on.

4.4.3 Feature Filtering

In the last section, prioritized forward selection was presented as a way to speed up (and even improve) feature selection if at least some of the tasks in T resemble each other. This is achieved, because the agents first perform feature selection on a set of features that are at least known to work for some tasks. Still, if the number of tasks already solved $T' \subseteq T$ is very large, the set $\mathbf{X}_{\mathrm{T'}}$ can be very large as well, which would decrease its utility, because in the worst case $\mathbf{X}_{\mathrm{T'}} = \mathcal{X}$, which would lead to the traditional feature extraction problem.

The key to solve this problem is to share only some feature among tasks, i.e. not to use all features in $\mathbf{X}_{\mathrm{T'}}$ in the forward selection process but only a subset $\mathbf{X}_{cand} \subseteq \mathbf{X}_{\mathrm{T'}}$ of them. This process will be denoted as *feature queries* in the following. Given a yet unsolved task $t \in T$, the idea is to query already solved tasks for features $\mathbf{X}_{cand}$ that are promising for t.

Definition 4.4.7. A *feature query* for task $t \in T$ yields a subset of features $\mathbf{X}_{\mathrm{cand}} \subseteq \mathbf{X}_{\mathrm{T'}}$ that are assumed to be relevant to solve task t from the set of features $\mathbf{X}_{\mathrm{T'}}$ used to solve a set of tasks $T' \subseteq T$.

In the following, several approaches of how to select such „promising“ features will be discussed. The first group of approaches is based on efficiently identifying feature subsets $\mathbf{X}_{\mathrm{cand}} \subseteq \mathbf{X}_{\mathrm{T'}}$ directly by evaluating the features against the current task t. The second approach is based on imposing a similarity relation on tasks. Based on this similarity, a set of tasks T_{cand} that are similar to t are identified and $\mathbf{X}_{T_{cand}}$ is used for prioritized forward selection instead of $\mathbf{X}_{\mathrm{T'}}$.

Both approaches are targeted mostly at classification tasks.

Selecting Features based on Relevance

In this approach, first, each feature is weighted by a score. Then, the subset of features with the highest scores are selected and used as input into forward selection. This leads to a combination of filter and wrapper approach. Any filter approach can easily be applied

to this problem. This works, however, only for classification. Also, using filters to select promising features inherits some of the problems of the corresponding filter approaches, such as selecting redundant features and not regarding feature interaction. There is also a more subtle but important limitation when it comes to the distributed scenario. Some of the feature values might be, while being defined in general, not known to a given a given agent. This will be discussed below.

Selecting Features based on Task Similarity

Task relatedness The filter approach selects features individually. A very promising alternative is not to analyze each feature in $\mathbf{X}_{\mathrm{T'}}$ on whether it is relevant for a new task $t \in T$, but to analyze tasks in $T' \subseteq T$ whether they are similar to task t.

Definition 4.4.8. A *task similarity measure* is defined as function $sim : T^2 \rightarrow \mathbb{R}$, that measures the similarity of two data mining tasks.

In contrast to item similarity, as defined in definition 1.3.2, we assume here that the similarity of tasks is expressed by values in $\mathbb{R}$. Again, larger values of this function denote higher similarity.

Given a task similarity measure sim, we obtain a set of candidate features $\mathbf{X}_{\mathrm{cand}} \subseteq \mathbf{X}_{\mathrm{T'}}$ by the following procedure.

Definition 4.4.9. We select a candidate set of features $\mathbf{X}_{\mathrm{cand}} \subseteq X_{T'}$ based on *task similarity selection* by assigning $\mathbf{X}_{\mathrm{cand}} = \mathbf{X}_{\mathrm{T''}}$, with $T'' = \{t' \in T' | sim(t, t') \geq \beta\}$, where t is the task that we are facing and β is a threshold.

Thus, we select tasks with a maximal similarity to the given task t and use the corresponding features as input to prioritized forward selection.

The open question is how to compare two data mining tasks. In the following, we will only regard classification tasks.

Most approaches to multi-task learning define an implicit task similarity measure by assuming distributions on the underlying model parameters. Unfortunately, these methods are only applicable if we assume that all learners use the same model class and if learning happens in parallel at each agent. One exception is the task clustering approach proposed in [181]. While the application to different model classes is not mentioned in this work, the basic method proposed there could be extended in this way.

The similarity of two learning tasks t and t' is calculated as follows. First, for each task a hypothesis h, respectively h' is identified individually. Then the hypothesis h is applied to the examples of t', leading to the expected loss $E_{t,t'}(L)$.

Definition 4.4.10. Given two data mining tasks $t, t' \in T$. The *expected loss of applying a classification model optimized for t to t'* is denoted as

$$E_{t,t'}(L) = \sum_{x \in D} L(h(x), f'(x)) \cdot p(x)$$

where f' is the assumed true relationship underlying classification task t' and h is an hypothesis that minimizes the expected loss for task t assuming a true relationship f. $p(x)$ is the probability of an item $x \in D$.

Based on this expected loss, a task similarity function can be defined.

Definition 4.4.11. The *relatedness of two tasks* t and t' is defined as

$$sim(t, t') = -E_{t,t'}(L)$$

Please note that this similarity measure is not symmetric in general.

In a second step, tasks can be clustered into groups of similar tasks. It is assumed that the number of such task clusters is fixed. The algorithm thus yields a partition $\{T_1, ..T_m\}$ of the overall set of tasks T. The following objective function is used to find an optimal task clustering.

$$\frac{1}{m} \sum_{i=1}^{m} \sum_{t \in T_i} \frac{1}{|T_i|} \sum_{t' \in T_i} E_{t,t'}(L)$$

This denotes the average loss if models are pairwise applied to the training data of other tasks in the same cluster. Obviously, this quantity should be minimized. The optimization problem can be solved, for instance, by using the k-medoids algorithm (see section 1.4.3).

Once such a solution is identified, a model is searched that minimizes the loss for all tasks in a task cluster.

Just as for incremental clustering in general, clusters need to be adapted from time to time. For this reason, a complete re-clustering is performed at regular intervals. Clustering tasks has two advantages. Learning a hypothesis based on a small training set can easily lead to overfitting. If several tasks are grouped together, this problem less severe, because information is shared among the tasks in one cluster. Second, finding an optimal hypothesis for new tasks can be achieved very efficiently, because it is only necessary to select among m solutions.

Applying this approach leads to the following computational effort applied to all tasks in T, where we assume that the underlying model is derived by a SVM [73] on a set of features that is common to all tasks and denoted as $\mathbf{X}_B$ (see below).

Lemma 4.4.12. *Feature queries based on task relatedness for all tasks in* T *can be achieved in* $O(|\mathbf{X}_B||D||T|^2 + |\mathbf{X}_B||T||D|^3)$

Proof. Generating base feature weights for each task in T has to be performed exactly once for each task, leading to $|\mathbf{X}_B||T||D|^3$. The weight vector corresponding to each task has to be applied to each other task, leading to $|\mathbf{X}_B||T|^2|D|$. □

This approach has, however, several severe drawbacks. First, the similarity measure for tasks is not metric and not even symmetric. Many optimization strategies can therefore not be applied. Second, similar tasks cannot be found in a query-like way easily. We will come back to this problem, when we will discuss the implementation of distributed feature sharing in p2p networks in section 4.5.

Comparing Tasks by Base-Weights In the following a method is proposed that compares feature weights directly, without using the underlying training data. We assume a set of features that is common to all tasks in T. This common set of features $\mathbf{X}_B \subseteq \mathcal{X}$ will be denoted as *base features.* Also, we assume a weighting function that assigns a weight to each base feature, given a specific task $t_i \in T$.

Definition 4.4.13. A *feature weighting function* is a function

$$w_t : \mathbf{X}_B \rightarrow \mathbb{R}$$

that assigns a weight to each feature in $\mathbf{X}_B$ reflecting its relevance for a task $t \in T$. We assume the features in $\mathbf{X}_B$ to be totally ordered in an arbitrary way, thus $\mathbf{X}_\mathrm{B} = \{X_1, .., X_l, .., X_k\}$. Then, w_{il} denotes the weight of feature $X_l \in \mathbf{X}_B$ for task t_i.

The weights assigned to these features are denoted as *base feature weights.* Two learning tasks are compared by directly comparing their base feature vectors. This representation of learning tasks is motivated by the idea that a given learning scheme approximates similar constructed features by a set of base features in a similar way. The approach works as follows: for a given learning task $t_i \in T$ we first calculate the relevance of all base features $\mathbf{X}_\mathrm{B}$ concerning t_i. This vector is used to compare this task to other tasks. We use a distance function $d(t_i, t_j)$ to decide whether two learning tasks are similar.

Still, there are two problems to be solved. First, how to assign the base weights given a set of labeled examples and second, how to compare two base weights?

We assume that for a modified set of base features $\mathbf{X}'_\mathrm{B} = \{X_1 ... X_k, X_{k+1}\}$ the function w' denotes the weighting function for $\mathbf{X}'_\mathrm{B}$ and d' denotes the distance measure for $\mathbf{X}'_\mathrm{B}$.

In a first step, we define a set of conditions which must be met by feature weighting schemes. In a second step, a set of conditions for learning task distance is defined which makes use of the weighting conditions.

Condition 4.4.14. Assume a task $t_i \in T$, a set of base features $\mathbf{X}_\mathrm{B} = \{X_1, .., X_l, .., X_k\}$ and an altered set of base features $\mathbf{X}'_\mathrm{B} = \{X_1, .., X_l, .., X_k, X_{k+1}\}$. Also assume that $\mathbf{X}_B$ does not contain any pairwise alternative features. Then the following must hold:

(W1) If $X_l \in \mathbf{X}_B$ is irrelevant for task t_i, then $w_{il} = 0$ and $w_{il} > 0$, otherwise

(W2) If there is a $1 \leq l \leq k$ such that $X_{k+1} \sim X_l$, then $w'_{il} + w'_{ik+1} = w_{il}$

(W3) If there is a $1 \leq l \leq k$ such that $X_{k+1} \sim X_l$, then $w'_{il} = w'_{ik+1}$

(W4) For all $1 \leq l \leq k$ holds that if not $X_l \sim X_{k+1}$, then $w'_{il} = w_{il}$

These conditions state that irrelevant features have weight zero (W1) and that the sum of weights of alternative features must be constant (W2). Also, alternative features should get equal weights (W3). Condition (W4) states that the weight of a feature is not affected if any features are added that are not alternative to it.

We can now define a set of conditions which must be met by distance measures for learning tasks which are based on feature weights only.

Condition 4.4.15. Assume two task $t_i, t_j \in T$, a set of base features $\mathbf{X}_\mathrm{B} = \{X_1, .., X_l, .., X_k\}$ and an altered set of base features $\mathbf{X}'_\mathrm{B} = \{X_1, .., X_l, .., X_k, X_{k+1}\}$. Also assume that $\mathbf{X}_B$ does not contain any pairwise alternative features. Then the following must hold:

(D1) $d(t_i, t_j) = 0 \Leftrightarrow t_i = t_j$

(D2) $d(t_i, t_j) = d(t_j, t_i)$

(D3) $d(t_i, t_k) \leq d(t_i, t_j) + d(t_j, t_k)$

(D4) If X_{k+1} is irrelevant to both tasks t_i and t_j, then $d(t_i, t_j) = d'(t_i, t_j)$

(D5) If there is a $1 \leq l \leq k$ such that $X_{k+1} \sim X_l$, then $d(t_i, t_j) = d'(t_i, t_j)$

(D1) - (D3) represent the conditions of a metric. These conditions are required for efficient case retrieval and indexing, using e.g. M-Trees [36]. (D4) states that irrelevant features should not have an influence on the distance. Finally, (D5) states that adding alternative features should not have an influence on distance.

In this section, we will show that many feature weighting approaches do not fulfill conditions (W1) - (W4). Furthermore, one of most popular distance measures, the Euclidean distance, cannot be used as a learning task distance measure introduced above.

Lemma 4.4.16. *No feature selection fulfills the conditions (W1) - (W4), in general.*

Proof. For feature selection methods, weights are by definition binary, i.e. $w_{il} \in \{0, 1\}$. We assume a learning task $t_i \in T$ and an $\mathbf{X}_\mathrm{B} = \{X_1, .., X_l, .., X_k\}$ with no alternative and irrelevant features. We add a feature X_{k+1} which is alternative to a feature $X_l \in \mathbf{X}_B$, obtaining a feature space $\mathbf{X}'_\mathrm{B} = \{X_1, .., X_l, .., X_k, X_{k+1}\}$. Then, either $w'_{il} = w'_{ik+1} = w_{il} = 1$, leading to a contradiction with (W2), or $w'_{il} \neq w'_{ik+1}$ leading to a contradiction with (W3). □

Lemma 4.4.17. *Any feature weighting method for which w_{il} is calculated independently of all other features does not fulfill the conditions (W1) - (W4).*

Proof. We assume a learning task $t_i \in T$ and a $\mathbf{X}_\mathrm{B} = \{X_1, .., X_l, .., X_k\}$ with no alternative and irrelevant features. We add a feature X_{k+1} which is alternative to a feature

$X_l \in \mathbf{X}_B$, obtaining the feature space $\mathbf{X}'_{\mathrm{B}} = \{X_1, .., X_l, .., X_k, X_{k+1}\}$. If w is independent of other features, then adding X_{k+1} would not change the weight w_{il} in the new feature space $\mathbf{X}'_{\mathrm{B}}$, thus $w'_{il} = w_{il}$. From (W3) follows that $w'_{ik+1} = w'_{il}$ which is a violation of (W2). □

This lemma essentially covers all feature weighting methods that treat features independently, such as information gain [147] or Relief [96].

The next theorem states that the Euclidean distance cannot be used as a distance measure based on feature weights.

Theorem 4.4.18. *Euclidean distance does not fulfill the conditions (D1) - (D5).*

Proof. We give a counterexample. We assume that a weighting function w is given which fulfills the conditions (W1) - (W4). Further assume that learning tasks $t_i, t_j \in T$ are given, as well as $\mathbf{X}_{\mathrm{B}} = \{X_1, .., X_l, .., X_k\}$ with no alternative and irrelevant features. We add a feature X_{k+1} which is alternative to a feature $X_l \in \mathbf{X}_B$, obtaining a feature space $\mathbf{X}'_{\mathrm{B}} = \{X_1, .., X_l, .., X_k, X_{k+1}\}$.

We infer from conditions (W2) and (W3) that

$$w'_{ik+1} = w'_{il} = \frac{w_{il}}{2} \quad \text{and} \quad w'_{jk+1} = w'_{jl} = \frac{w_{jl}}{2}$$

and from condition (W4) that

$$\forall 1 \leq m \leq k, m \neq l : (w'_{im} = w_{im}) \wedge (w'_{jm} = w_{jm})$$

In this case the following holds for the Euclidean distance

$$\begin{aligned} d'(t_i, t_j) &= \sqrt{\theta + 2\left(w'_{ik+1} - w'_{jk+1}\right)^2)} = \sqrt{\theta + 2\left(\frac{w_{il}}{2} - \frac{w_{jl}}{2}\right)^2} \\ &= \sqrt{\theta + \frac{1}{2}\left(w_{il} - w_{jl}\right)^2} \neq \sqrt{\theta + \left(w_{il} - w_{jl}\right)^2} \\ &= d(t_i, t_j) \end{aligned}$$

with

$$\theta = \sum_{m=1, m\neq l}^{k} \left(w'_{im} - w'_{jm}\right)^2 = \sum_{m=1, m\neq l}^{k} \left(w_{im} - w_{jm}\right)^2$$

□

Theorem 4.4.19. *The feature weight calculation of SVMs with linear kernel function meets the conditions (W1) - (W4).*

Proof. See [121, 120]. □

In order to calculate the distance of learning tasks based only on a set of base feature weights we still need a distance measure that met the conditions (D1) - (D5).

Theorem 4.4.20. *Manhattan distance does fulfill the conditions (D1) - (D5).*

Proof. The conditions (D1) - (D3) are fulfilled due to basic properties of the Manhattan distance. We assume that a weighting function w is given, that fulfills the conditions (W1) - (W4). Further, we assume that learning tasks $t_i, t_j \in T$ are given, as well as $\mathbf{X}_\mathrm{B} = \{X_1, .., X_l, .., X_k\}$ with no alternative and irrelevant features. We add a feature X_{k+1} obtaining a feature space $\mathbf{X}'_\mathrm{B} = \{X_1, .., X_l, .., X_k, X_{k+1}\}$.

We prove (D4) by observing, that $w'_{ik+1} - w'_{jk+1} = 0$ if X_{k+1} is irrelevant to t_i and t_j. This yields the following:

$$\begin{aligned} d'(t_i, t_j) &= \left(\sum_{r=1}^{k} |w'_{ir} - w'_{jr}| \right) + \underbrace{|w'_{ik+1} - w'_{jk+1}|}_{0} \\ &= d(t_i, t_j) \end{aligned}$$

from (W4).

We prove *(D5)* by assuming that X_{k+1} is alternative to a feature $X_l \in \mathbf{X}_\mathrm{B}$.

We infer from conditions (W2) and (W3) that

$$w'_{ik+1} = w'_{il} = \frac{w_{il}}{2} \quad \text{and} \quad w'_{jk+1} = w'_{jl} = \frac{w_{jl}}{2}$$

and from condition (W4) that

$$\forall 1 \leq r \leq k, r \neq l : (w'_{ir} = w_{ir}) \wedge (w'_{jr} = w_{jr})$$

Then we can show, that

$$\begin{aligned} d'(t_i, t_j) &= \left(\sum_{r=1, r\neq l}^{k} |w'_{ir} - w'_{jr}| \right) + 2 \cdot |w'_{ik+1} - w'_{jk+1}| \\ &= \left(\sum_{r=1, r\neq l}^{k} |w_{ir} - w_{jr}| \right) + |w_{il} - w_{jl}| \\ &= d(t_i, t_j) \end{aligned}$$

from (W4) and (W2). □

SVM feature weights in combination with Manhattan distance fulfill the necessary constraints for a learning task distance measure based on feature weights.

How efficient is feature pre-selection based on this approach?

Lemma 4.4.21. *Feature selection for all tasks in T can be achieved in $O(|\mathbf{X}_B||T|^2 + |\mathbf{X}_B||T||D|^3)$.*

Proof. Generating base feature weights for each task in T has to be performed exactly once for each task, leading to $|\mathbf{X}_\mathrm{B}||T||D|^3$. The weight vector corresponding to each task has to be compared to each other, leading to $|\mathbf{X}_\mathrm{B}||T|^2$. □

This approach is very appealing for its efficiency. While training the SVM can be costly in some cases, it has only to be performed once for each task. Comparing the vectors is then very efficient.

4.5 Distributed Feature Extraction in Peer-to-Peer Networks

In the application scenarios on which this work focuses, tasks are not solved at a central place but are distributed in a network. In collaborative media organization, users might, for instance, apply their handhelds or cell phones to share their information in a fully distributed way.

4.5.1 Existing Approaches to Data Mining in Peer-to-Peer Settings

Our aim is to enable distributed feature extraction in a loosely coupled network. Two important paradigms for such systems are global computing (GC) and p2p computing. GC aims at distributing computational demanding tasks to a high number of nodes over a network. An example is Xtreme Web [29], that uses a coordination service to actively distribute computation tasks among nodes. P2p networks are well known for file sharing applications but are applicable in many other scenarios as well.

P2p and GC systems can be centralized (all communication is routed through a server), brokered (resource discovery is centralized, service invocation is distributed) or fully distributed. Fully distributed systems are further devised into structured and unstructured p2p networks. Structured networks employ a reference space that allows for efficient resource discovery. Examples are Distributed Hash Table (DHT) systems, such as Chord [169]. They often become suboptimal in the presence of a high fluctuation of peers [34]. Also sophisticated search strategies are not supported directly. Therefore, virtually all approaches to p2p data mining focus on unstructured networks.

In unstructured systems, resource discovery and propagation is based on range limited broadcast. The most popular example is the Gnutella network. There are several approaches to make (unstructured) p2p systems more scalable through the use of explicit super nodes or, as in Gia [34], by a combination of topology adaptation, flow control and look-aheads.

Current approaches to p2p data mining are based on epidemic dissemination protocols that use simple base operations, such as distributed majority vote or averaging. In

[103] an approach based on the „newcast model of computation“ is presented. It is based on the dissemination of information by probabilistic broadcast [53]. In [193] an algorithm for distributed averaging in p2p networks is proposed that is used to evaluate locally generated rules under the global distribution. [41] proposes a distributed k-means algorithm that uses this strategy to obtain globally valid centroids from local ones by distributed averaging. These approaches aim at establishing an equilibrium among the nodes, such that all peers share the global model (or its approximation). While dissemination is well suited in a homogeneous settings, it can produce a considerable overhead in heterogeneous settings, in which each peer aims at developing an own local model.

Therefore, we propose an approach that is rather based on a „pull“ paradigm than on a „push“ paradigm. Agents query other agents to obtain a set of candidate features as input for prioritized forward selection. This approach leads to much fewer communication costs than a dissemination approach, because only agents with relevant features respond.

4.5.2 Differences to the Non-distributed Case

In the following, we will use term „distributed feature extraction" to denote approaches that solve the distributed representation problem in a distributed scenario. The basic procedure is still the one of a prioritized forward selection. The only difference is that we propose methods on how the set of candidate features $\mathbf{X}_{\text{cand}}$ is actually collected from different agents in a p2p network in an efficient way.

Each agent is assumed to perform the following steps to solve its local data mining task $t \in T$:

1. Query other agents for features to solve task t.
2. Integrate the features that were received from different agents into a set of candidate feature $\mathbf{X}_{\text{cand}}$.
3. Apply prioritized forward selection based on $\mathbf{X}_{\text{cand}}$.

The third step was presented and analyzed above. Feature integration (step 2) is trivial, because we assume that features are static and have globally unique names. The open question is how to query other agents for features in an efficient way.

In section 4.4.3, we already introduced several possibilities to query other tasks for features: filter approaches, task relatedness and base features weights. All of these three approaches and their variants can be used to retrieve features in a distributed system.

How does the distributed scenario differ from the non-distributed one?

First, if all tasks are solved at the same node, then feature sharing among tasks comes without any additional costs. If tasks are solved at different nodes of a network, this is no longer the case. On the one hand, communication itself can be expensive. On the other hand, if communication channels are slow, it might be possible that it is faster to

perform feature selection only locally, because waiting for candidate features delivered by other agents may take much longer.

Overall, we face a trade-off between three quantities:

1. *The response time*
   This time is constituted by the time used querying for features and performing feature selection.
2. *The accuracy of the feature sets*
   This criterion denotes the overall accuracy q^* by which the tasks are achieved.
3. *The communication costs*
   The communication costs of feature sharing are measured as the number of values that are transferred over the network.

These criteria are pair-wise conflicting. In order to obtain a response quickly, heuristic optimizations must be used, that, however, on average lead to an inferior accuracy (just as for the traditional representation problem). An optimal performance can be reached by exhaustive search, which in turn leads to high response times. If no features are obtained from other agents (feature selection is performed locally only), then there are no communication costs but possibly a higher response time, because local feature selection may take longer without input from the other nodes. Note that while we are facing a multi-objective optimization problem, it is not possible to apply multi-objective optimization methods here, because the process is executed in a decentralized way without any central control over the actions of individual agents.

A second aspect that distinguishes the distributed scenario from the non-distributed one is that feature values maybe known only to certain agents of a network. An important example is collaborative media organization. If a user tags an item with a textual description, then this can be perceived as an important feature of this item. Initially, the value of this feature is only known to the agent at which this tagging takes place. To make this feature applicable for other agents as well, the feature values have to be shared with other agents.

In this sense, distributed feature extraction can actually be seen as a kind of *dynamic p2p data warehousing*, where relevant data (in this case the feature values) is shared and aggregated just-in-time to process a task.

Third, some feature values may not be known to all agents, even if these feature values are defined. Assume the following example.

A first user has a music collection consisting only of 70's music. Another user only owns current chart entries. Even if some of the features of the 70's collector would be relevant for the second user, the feature values for these features may not be extracted from the audio files of the second user yet. To apply any of the filter approaches proposed above, these feature values would have to be determined first, before we can decide whether the feature is promising.

Thus, some of the feature query approaches presented above are only efficiently applicable among nodes that share the same items with the same features.

To capture the notion of communication costs in a formalized way, we define the following message types and state the costs connected with sending them from one node to another node in a network.

Definition 4.5.1. A *feature vector message* is a message that contains the values of k different features for a subset of items in $D' \subseteq D$. The communication costs are assumed to be in $O(k \cdot |D'|)$.

In many cases it is sufficient to transfer only the name of a feature, respectively a description of how it can be obtained from the raw data.

Definition 4.5.2. A *feature name message* is a message that contains the unique name of a feature. The communication costs of transferring k feature names are assumed to be in $O(1 \cdot k)$.

Finally, we will regard messages that contain only item identifiers.

Definition 4.5.3. An *item id message* is a message that contains a set of identifiers for items in $D' \subseteq D$. The communication costs of a message containing ids for the items in D' are assumed to be in $O(|D'|)$.

Finally, to analyze the base-weight approach presented in section 4.4.3, we define a feature weight message.

Definition 4.5.4. A *feature weight message* is a message that contains a vector of feature weights. The communication costs of a message containing k weights are assumed to be in $O(k)$.

For the sake of simplicity, we assume that each agent faces exactly one data mining task $t \in T$ and that each node in the network $v \in V$ contains exactly one agent.

4.5.3 Sharing Features in a Peer-to-Peer Scenario

In this section, we propose two frameworks for feature sharing as solutions to the distributed representation problem. The first one is based on a fully distributed p2p system and a second one is based on a client/server architecture.

In the fully distributed model, an agent sends a query for features to all its neighboring agents. This query is then distributed within the network by range limited broadcast. Each agent that stores feature information that is relevant to a query, responds to the requesting agent. The requesting agent incorporates all feature information and performs prioritized forward selection.

We also propose a simplified model. It is not based on a fully distributed p2p network but on a central server that facilitates communication and acts as a cache for feature information. The overall procedure is the same, the difference is that agents actively push their locally stored features to the server and that all queries are sent to and responded by the server.

Orthogonal to the network topology are the different ways of pre-selecting candidate features that are then used for prioritized forward selection. First, this can be achieved by any filter approach. The queries contain a set of item ids, possibly together with class information (a label for each item id). The response to such a query can either be a set of feature names or a matrix of feature values. Feature names can be applied if the requesting agent is able to extract the feature values locally from the data. If this is not possible, feature values must be sent. A third possibility for feature pre-selection are base-weights. The request contains a base-weight feature vector. The response contains a set of feature names. This approach is only applicable to binary classification tasks. In this case, feature names are returned in any case, as the query does not contain any item information. These three approaches can be applied in the p2p and in the centralized model, which leads to six different approaches, that will be analyzed below.

In the following, we first present the client/server based framework. We then analyze how this framework can be altered to be applicable in the fully distributed scenario.

The Feature Facilitator Approach

We first describe how features can be shared based on a central agent that collects these features. We then discuss how this approach can be combined with the feature filtering methods proposed in section 4.4.3.

Features can be shared easily by using a central shared data space or blackboard architecture [190]. Each agent stores all features that were selected for a data mining task in a local database (step 1).

The agent sends these features (or the corresponding feature names) to a centralized service called *feature facilitator* (step 2).

The facilitator stores the information, indicating the agent that provided them.

When an agent encounters a new data mining task, it queries the facilitator and receives a set of features names or feature vectors (step 4).

Then, the agent uses these features in a prioritized forward selection to accomplish the given data mining task (step 5).

Any new features created or selected during this process are stored in the local database and are sent to the feature facilitator (step 1'). After a modification (items or features were added or the label structure was changed), agents upload these changes only. The feature facilitator does not perform any processing that is specific to a data mining task. It merely collects feature information.

This approach can be combined with candidate selection based on a filter approach. In this case, an agent that encounters a new data mining task sends a vector containing all ids of corresponding items to the feature facilitator. For classification based filters, also the class labels for these items are included. The facilitator applies the filter to rank all features in the database with respect to the new task and selects the k best features. These features are then returned to the agent that initiated the query, either as feature values or as features names. This leads to the following communication costs.

Lemma 4.5.5. *The accumulated message size of the filter-based feature candidate selection for a set of tasks T using a feature facilitator is in $O(|T|(|D'| + |\mathcal{X}|))$ if feature names are shared and $D' \subseteq D$ is the set of query items.*

Proof. Each agent sends at most one item id message (possibly with class information). The size of this message is in $O(|D'|)$. Also, each agent receives at most one feature name message with a size in $O(|\mathbf{X}_{\text{cand}}|)$ and sends one with size in $O(|\mathcal{X}|)$ to the feature facilitator. □

Lemma 4.5.6. *The accumulated message size of the filter-based feature candidate selection for a set of tasks T using a feature facilitator is in $O(|T| \cdot |D'| \cdot |\mathcal{X}|))$ if feature vectors are shared and $D' \subseteq D$ is the set of query items.*

Proof. Each agent sends at most one item id message (possibly with class information). The size of this message is in $O(|D'|)$. Also, each agent receives at most one feature vector message as result with a size in $O(|\mathbf{X}_{\text{cand}}| \cdot |D'|)$. Uploading feature vectors to the facilitator is in $O(|D'| \cdot |\mathcal{X}|)$ for each task. □

For the base weight approach, each agent uploads the selected feature set, together with the base weight vector. The feature facilitator stores both in a central database. On a request, an agent first calculates the base weights corresponding to t and sends them to the feature facilitator. The feature facilitator selects the k most similar tasks and responds with the set of feature names that are used in these tasks.

Note that the feature facilitator cannot respond with feature values in this case, because the set of items $D' \subseteq D$ is not known at this point.

The communication costs are in this case the following.

Lemma 4.5.7. *The accumulated message size of base weight feature candidate selection for a set of tasks T using a feature facilitator is in $O(|T| \cdot |\mathcal{X}|)$ if feature names are shared.*

Proof. Each agent sends at most one base weight message of size $|\mathbf{X}_{\text{B}}|$ as query. Also, each agent receives at most one feature name message with size in $O(|\mathcal{X}|)$. Uploading features to the feature facilitator consists of a base weight message and a feature name message with size at most $O(|\mathcal{X}|)$. □

Fully Distributed Feature Sharing

The feature facilitator approach is not fully satisfying. Depending on the scenario, agents may not be connected to a global network, such as the Internet, but only to diverse local networks. In such scenarios, a more flexible solution is needed. We do not make strict assumptions about the underlying network and simply assume that each agent keeps reliable bidirectional communication channels to a set of neighbors.

Distributed feature extraction in a fully distributed setting is much more challenging than by using a central server in several respects:

1. Search cannot be directed to one single agent but has to be performed in a distributed way. Therefore, traditional search or brokering strategies cannot be applied.
2. Redundancy cannot be controlled easily (agents partially store the same information). This leads to potential inconsistencies and inefficiency, because the same or similar information is returned by several agents on a request.
3. Agents join and leave the network very often. This can lead to a considerable overhead, as agents frequently need to detect new neighbors in the network.
4. Communication is often more expensive and latency is higher, because messages are routed through several nodes.

A straightforward way to share features would be a dissemination approach, such as used in current p2p data mining applications. Each agent would send feature information to each neighbor until the system (temporarily) converges. While such an approach is well suited for some applications, such as the ones described in [41] or [193], it is not well suited for the given setting. The basic difference is that these approaches assume that a single global model is approximated by distributed data mining. In contrast to this problem, we assume that we face different data mining tasks locally. This leads to two problems. First, nodes must in this case store and forward information on items that are not relevant to them. Second, features can be filtered at the processing node only and not at the sender. This leads to a high communication overhead that a typical user will not accept.

We therefore propose a request-based approach. Instead of sending queries to a central server, agents broadcast a query to their direct neighbors (step 1).

Each agent that receives a query forwards this query to its own neighbors. A time-to-live tag, max_{hops}, is used to limit the number of times a query can be forwarded. If the network is a connected graph, then a query is received by all agents that can be reached with at most max_{hops} hops. All agents that receive a query evaluate it and reply to the requesting agent directly if this is possible, or by routing the response back along the way the query was received (step 2).

A response message from a peer has the same format as a response from the feature facilitator. The information received from several peers is aggregated in each individual agent (step 3).

An agent uses this extended feature set in a prioritized forward selection to solve the given data mining task (step 4) and stores the features selected to solve the task locally.

Again, this algorithm can be combined with all filter-based feature candidate selection approaches and with base feature weights.

In the filter-based approach, each agent disseminates an item id message to all agents in the maximum range. Each agent that receives such a message responds with relevant features, if there are any. The agent then performs prioritized forward selection. Instead of sending the features to a facilitator, they are stored locally.

We assume that an optimal broadcast strategy is used. The following holds (see [113]).

Lemma 4.5.8. *Broadcast in a synchronous network can be achieved in* $O(|V|size_{load} + |E|size_{control})$*, where* V *denotes the nodes in the network and* E *the edges.* $size_{load}$ *denotes the size of the message content distributed with the message and* $size_{control}$ *denotes the size of the administrative messages used to distribute the query.*

Proof. see [113]. □

We assume that the control messages are negligible, and that disseminating a message leads to an accumulated message size that is linear in the message size and the number of receivers.

Lemma 4.5.9. *The accumulated message size of the filter-based feature candidate selection for a set of tasks* T *using a p2p approach is in* $O(|T|^2(|D'| + |\mathcal{X}|))$ *if feature names are shared and* $D' \subseteq D$ *is the set of query items.*

Proof. Each agent sends at most one item id message (possibly with class information) to all other agents. The size of this message is in $O(|D'|)$. Also, each agent receives at most one feature name message with a size in $O(|\mathcal{X}|)$ from all other agents. □

Lemma 4.5.10. *The accumulated message size of the filter-based feature candidate selection for a set of tasks* T *using a p2p approach is in* $O(|T|^2 \cdot |D'| \cdot |\mathcal{X}|))$ *if feature vectors are shared and* $D' \subseteq D$ *is the set of query items.*

Proof. Each agent sends at most one item id message (possibly with class information) to all other agents. The size of this message is in $O(|D'|)$. Also, each agent receives at most one feature vector message from each other agent with a size in $O(|\mathcal{X}| \cdot |D'|)$. □

Base weight queries can be applied in a similar way. Again, base weights can only return feature names, not values.

In this case, we obtain the following accumulated message sizes.

Lemma 4.5.11. *The accumulated message size of base weight feature candidate selection for a set of tasks* T *using a p2p approach is in* $O(|T|^2 \cdot |\mathcal{X}|)$ *if feature names are shared.*

	Message size (facilitator)	Message Size (p2p)
BW+names	$O(\|T\|\cdot\|\mathcal{X}\|)$	$O(\|T\|^2\cdot\|\mathcal{X}\|)$
Filter+names	$O(\|T\|(\|D'\|+\|\mathcal{X}\|))$	$O(\|T\|^2(\|D'\|+\|\mathcal{X}\|))$
Filter+values	$O(\|T\|\cdot\|D'\|\cdot\|\mathcal{X}\|)$	$O(\|T\|^2\cdot\|D'\|\cdot\|\mathcal{X}\|)$

Table 4.2: This table compares the message size complexity of the base weight approach (BW) and the filter approach (Filter) in combination with sharing features names (+names) or feature values (+values).

Proof. Each agent sends at most one base weight message of size $|\mathbf{X}_\mathrm{B}|$ as query to each other agent. Also, each agent receives at most one feature name message with size in $O(|\mathcal{X}|)$ from all other agents. □

4.5.4 Comparison of the Approaches

The approaches presented above behave quite differently concerning communication costs and computational complexity. Tables 4.2 and 4.3 give an overview.

The approach that produces the smallest amount of communication costs is querying for relevant features using task similarity with base feature weights. A disadvantage of the approach is that only feature names can be returned, not feature values (at least not at a reasonable price). The reason is that the set of query items $D' \subseteq D$ is not transmitted in the query and the server would not know for which items to return feature values. Returning feature values for all items in D is in almost all cases not a reasonable option.

For filter approaches, we have the choice of returning feature values or feature names. The choice depends mostly on whether the feature values can be extracted at the client node (which is the case, for instance, for features extracted from raw multimedia data, if agents share the same items), or whether this is not possible (which is the case, for instance, for features derived from different user annotations/tags). Returning feature names leads to much lower communication costs in almost all cases.

Comparing the p2p and the facilitator approach, disseminating a query based on broadcast leads to communication costs that is squared in the number of tasks (as we assume that each node contains one task). The computation time is the same for facilitator and p2p approach for the base weight approach. For the filter approach, the facilitator leads to higher computation costs, because possibly (the same) features are filtered at each node.

4.5.5 Further Optimizations

The above algorithm is basically the same as the Gnutella algorithm for search in unstructured networks (see e.g. [34]). A difference is that the algorithm does return feature information directly, instead of first returning a set of search results and then querying

	Computation (facilitator)	Computation (p2p)
BW+names	$O(\|\mathbf{X}_B\|\|T\|^2 + \|\mathbf{X}_B\|\|T\|\|D'\|^3)$	$O(\|\mathbf{X}_B\|\|T\|^2 + \|\mathbf{X}_B\|\|T\|\|D'\|^3)$
Filter+names	$O(\|T\|\|\mathcal{X}\|\|D'\|)$	$O(\|T\|^2\|\mathcal{X}\|\|D'\|)$
Filter+values	$O(\|T\|\|\mathcal{X}\|\|D'\|)$	$O(\|T\|^2\|\mathcal{X}\|\|D'\|)$

Table 4.3: This table compares the computational complexity of the base weight approach (BW) and the filter approach (Filter) in combination with sharing features names (+names) or feature values (+values).

individual agents for the actual data. This is motivated by the constraint that feature information should be made available as quickly as possible. Second, feature information produces much smaller message sizes than encountered in usual p2p file sharing systems. Several standard techniques from the area of distributed systems can be applied to improve this framework.

Supernodes can be used to distribute queries more efficiently. Every two agents that are connected directly to a super node have a distance of at most three hops if all supernodes are fully interconnected.

Other optimizations, not discussed here, but possibly relevant in practice, are compression and and topology adaptation. Compression helps to reduce the size of query and response messages. Topology adaptation helps to restructure the network in an optimal way [34]. Such approaches, as studied in the area of distributed systems, are beyond the scope of this work. The same holds for the use of structured p2p networks, such as Chord [169] or Pastry [152]. A basic limitation of such systems is that they use keys to identify items. These keys do not allow to search for items content based (e.g. by substring). However, such approaches could be easily combined with the methods proposed here.

4.6 Empirical Results

4.6.1 Overview

In the following, the methods proposed above will be evaluated using different criteria. To guide the evaluation, we derive several assertions from the first part of this chapter, that are then empirically validated. The first set of assertions corresponds to the general framework of the distributed representation problem and the prioritized forward selection (PFS) algorithm:

(P1) In general, there is a conflict between the overall accuracy and the size of the accumulated feature set. Thus, sacrificing some accuracy can lower the overall number of features.

(P2) PFS is able to reduce the size of the accumulated feature set.

(P3) PFS leads to a faster increase in accuracy compared to traditional forward selection. This is essentially means that if the algorithm is invoked in a best effort manner, good results are achieved faster.

(P4) The number of times the inner learning and evaluation procedure is invoked is smaller for PFS than for traditional forward selection. This in general also entails that the running time is smaller in this case.

(P5) The above effects become stronger with an increasing amount of irrelevant and alternative features.

These properties concern the basic version of PFS. A second set of assertions can be posed for the pre-selection of candidate features by a filter approach and by task similarity.

(F1) Pre-selecting features based on a filter approach reduces the number of invocation of the inner learning and evaluation procedure and leads to a faster increase in accuracy.

(F2) Pre-selecting features using base-weight task similarity reduces the number of invocation of the inner learning and evaluation procedure and leads to a faster increase in accuracy.

(F3) The performance and accuracy of base-weight task similarity does not decrease in the presence of many alternative and irrelevant features.

The last set of properties concerns the distributed algorithms and their properties:

(D1) Feature pre-selection by filters reduces the amount of communication costs compared to actively disseminating features.

(D2) Feature pre-selection by task similarity reduces the amount of communication costs compared to actively disseminating features and over pre-selection by filters.

In the following, the experimental setup used to validate these assertions will be described.

4.6.2 Experimental Setup

For all datasets, the evaluation procedure is the following. The tasks in a data set are solved one by one in a random order. To solve them, traditional forward selection, prioritized forward selection and prioritized forward selection with feature pre-selection is used.

Each feature set is evaluated by applying 10-fold stratified cross-validation to the corresponding task. As learning algorithm, nearest neighbor is used. This choice is motivated by the fact that nearest neighbor can handle multi-class problems very easily and does not perform implicit feature selection.

The following properties were measured for each set of tasks:

- The average accuracy over all tasks
- The number of times the inner evaluation procedure was invoked
- The size of the accumulated feature set
- The average increase in performance depending on the number of steps performed in the selection procedure

The last property yields a graph, with the number of steps on the x-axis and the current best accuracy on the y-axis. This graph is created by first creating a graph for each task in a set of tasks. Then these graphs are averaged.

For the evaluation of the distributed algorithms, an additional criterion was used, namely the accumulated message size. We assume that each task is assigned to exactly one agent or node in a network. This network is assumed to be connected such that each agents can exchange messages with each other agent. For the running time an abstract value was used instead of actual message sizes, to avoid effects of operating system or network type, which are not in the focus of this work. Therefore, a value of 1 was amounted for transferring a single, atomar value. The cost for a vector of values was set correspondingly equal to the length of the vector. The transfer of a feature names/feature construction description was set to 10. This reflects that unique feature names and construction descriptions are usually much larger than atomar values. All administrative messages were neglected. Especially, the cost for a full broadcast is assumed to be the size of the broadcasted message multiplied by the number of receivers.

All experiments were performed with Rapid Miner 4.0 and the distributed data mining simulator, both of which can be obtained as open source software[1]

The following methods were used in the evaluation:

1. *Traditional Forward Selection*
   For traditional forward selection, the features are arranged in an arbitrary total order.
2. *Prioritized Forward Selection, as proposed above*
   The parameter ϵ was varied. Below, this value is always indicated explicitly.
3. *Prioritized Forward Selection with a filter-based approach*
   The one-attribute rule learner was used [77] as filter method. This learner generates a rule from each attribute and scores them according to their expected performance. The threshold for feature selection was set to zero. Thus only features were omitted for which a single rule performed worse than always choosing the majority class (the one with the most examples).
4. *Prioritized Forward Selection with task-similarity*
   The base weight approach above was applied by using a random subset of the

[1] http://www.rapid-i.de and http://www-ai.cs.uni-dortmund.de/SOFTWARE/ddm_simulator

	tasks	examples	classes	feat.	rel. feat.	funct.	ref.
synth1	5	93 - 108	2	7	4	1	
synth2	5	90 - 111	2	20	4	1	
synth3	10	86 - 112	2	15	10	3	
synth4	10	87 - 116	2	30	10	3	
garageband	39	16 - 674	2-9	49	?	?	[78]
register	9	92 - 777	2	26	?	?	[189]

Table 4.4: Overview of the datasets: the number of examples for each task (examples), the number of different classes for the tasks (classes), the number of features (feat.), the number of relevant features (rel. feat.), the number of underlying generic functions (funct.) and a reference of the datasets (ref.). The number of relevant features and the number of generative functions are only known for the synthetic datasets.

available features. The task similarity threshold was optimized semi-automatically by analyzing the distribution of task similarity values. It is indicated below for each dataset.

These algorithms were combined with the two feature sharing strategies in distributed systems described above:

1. *Feature facilitator*
   Each agent is assumed to send the features it selected to a centralized data space. Subsequent agents query this centralized data space for features using an empty query, for simple PFS, or corresponding queries for task-similarity and filter-based pre-selection.
2. *Fully distributed feature sharing*
   For simple PFS, we assume that each agent disseminates all features it selected for a task to all other agents. For task-similarity and for filter-based pre-selection, the methods described in section 4.5.3 are applied.

The algorithms are thus applied always in two variants: in combination with the feature facilitator and with the fully distributed feature sharing.

A combination of synthetic and real datasets is used for evaluation. This allows to validate the applicability of the proposed methods in real-world applications and to analyze the influence of irrelevant and alternative features.

The most important properties of the individual datasets are described in table 4.4.

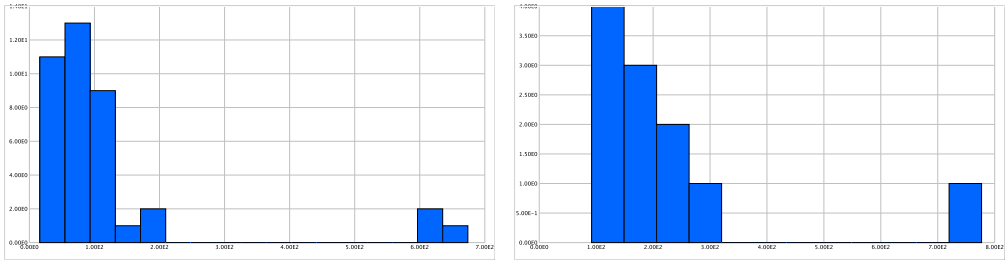
- *Synthetic Dataset I (synth1)*
  This dataset was created by the following procedure. First, an arbitrary, multidimensional, polynomial function was chosen. This function was applied to 200 examples with random attribute values, each of which based on a Gaussian-distribution

with zero mean and standard deviation one. Of 6 attributes, 4 are relevant to the function, two are irrelevant. Using a threshold, a binary label for each example was inferred. This threshold was chosen such that the amount of positive and negative examples was about equal. Then, each individual data set (each task) was created by first selecting a random subset of examples. The probability of selecting an example was set to 0.5. For each data set, a noise term was applied to the label attribute by flipping a label value with probability 0.05. Five tasks were generated in this way.

- *Synthetic Dataset II (synth2)*
  For the second dataset, the same procedure was used as for the first one, but 4 irrelevant and 3 alternative features were added. An alternative feature is generated by first choosing a random feature that was already generated and then applying additively a Gaussian noise term to the feature value with zero mean and standard deviation 0.1.
- *Synthetic Dataset III (synth3)*
  For the third dataset, the same procedure was applied as for the first one. However, instead of using a single function, three generative functions were used, depending on different subsets of 10 attributes. Of these 10 attributes, 9 were relevant for at least one of the three generative functions. There were no alternative attributes. Datasets (tasks) are generated by first randomly choosing a generative function and then applying the same procedure as for the first dataset.
- *Synthetic Dataset IV (synth4)*
  The forth data set is generated just as the third one, but there were 4 irrelevant and 3 alternative features added.

The two real-world datasets are the following:

- *The Garage Band Benchmark Dataset (garageband)*
  This dataset contains the 39 tag structures on 1886 audio files. Each tag structure is a hierarchical taxonomy in which each inner node or leaf node may contain references to a subset of the underlying audio files. These tag structures reflect the ways in which the students organized their files. Each audio file is described by a set of 49 features. These features were extracted from the raw wave data using the method described in [119] and were shown to work well in a wide variety of applications. For the given experiments, only the top level of each taxonomy was used. This leads to 39 tasks. The number of classes is, in this case, not two, as for the synthetic datasets, but varies between 2 and 9. The number of examples varies from 5 to 200. A histogram of dataset sizes is depicted in figure 4.6.1. The dataset is publicly available. Details can be found in [78].
- *The Register Classification Dataset (register)*
  This dataset contains 9 different tasks. Each tasks corresponds to a single musical instrument and contains examples for notes played in alto and bass. The aim is to separate high and low notes for each instruments without taking the pitch into



(a) garageband (b) register

Figure 4.6.1: This plot shows the distribution of the number of examples per task for the garageband and for the register dataset. The synthetic datasets contain about the same number of examples for each task by construction.

	avg. acc.	#features	#cycles
FS	0.79	5	127
PFS $\epsilon = 0.0$	0.79	5	91
PFS task sim $\epsilon = 0.0$	0.79	5	91
PFS filter $\epsilon = 0.0$	0.78	6	83

Table 4.5: Feature sharing results for synth1

account. All notes are stored as wave forms. From these waveforms 26 features were extracted. The distribution of the number of examples to tasks is depicted in figure 4.6.1. The dataset is described in detail in [189] and can be obtained from the corresponding authors.

All methods were applied to all six datasets, with the single exception of the garageband dataset to which the base weight approach could not be applied, because it contains more than two classes in general.

4.6.3 Results

In the following, first the results for accuracy, number of cycles and number of features are presented. Then the results for communication costs concerning the two models of distributed feature sharing are discussed.

Table 4.5 to table 4.10 show the results for each of the six datasets concerning average accuracy, number of features and number of inner cycles. Figure 4.6.2 shows the averaged performance curves for all six datasets.

We can see the following. Traditional forward selection produces the largest sets of aggregated features over all six datasets. Also, the number of invocations of the inner learning and evaluation procedure was the highest among all methods on all datasets. Simple PFS selects on five of six datasets less features than traditional forward selection.

	avg. acc.	#features	cycles
FS	0.79	6	284
PFS $\epsilon = 0.05$	0.79	4	179
PFS task sim $\epsilon = 0.05$	0.79	4	179
PFS filter $\epsilon = 0.05$	0.72	4	176

Table 4.6: Feature sharing results for synth2

	avg. acc.	#features	cycles
FS	0.76	15	789
PFS $\epsilon = 0.0$	0.74	9	493
PFS task sim $\epsilon = 0.02$	0.78	10	396
PFS filter $\epsilon = 0.02$	0.74	8	432

Table 4.7: Feature sharing results for synth3

	avg. acc.	#features	cycles
FS	0.79	24	1863
PFS $\epsilon = 0.0$	0.79	20	1533
PFS task sim $\epsilon = 0.03$	0.79	15	966
PFS filter $\epsilon = 0.0$	0.76	12	952

Table 4.8: Feature sharing results for synth4

	avg. acc.	#features	cycles
FS	0.61	43	7302
PFS $\epsilon = 0.0$	0.59	27	3009
PFS task sim	-	-	-
PFS filter $\epsilon = 0.02$	0.58	23	3767

Table 4.9: Feature sharing results for garageband

	avg. acc.	#features	cycles
FS	0.96	16	742
PFS $\epsilon = 0.0$	0.96	11	580
PFS task sim $\epsilon = 0.01$	0.96	6	482
PFS filter $\epsilon = 0.0$	0.96	13	571

Table 4.10: Feature sharing results for register

The number of cycles is always smaller. Concerning accuracy, PFS mostly performs equally well or only slightly worse. This supports propositions (P2) and (P4). In figure 4.6.2 we see that the increase in performance is much faster for PFS than for traditional forward selection, just as claimed in (P3). Later tasks can exploit the work done on preceding tasks by directly focusing on promising features. This works even if the tasks are heterogeneous. With an increasing number of alternative and irrelevant features, the difference between PFS and traditional forward selection may become even stronger, as can be seen on the dataset synth2.

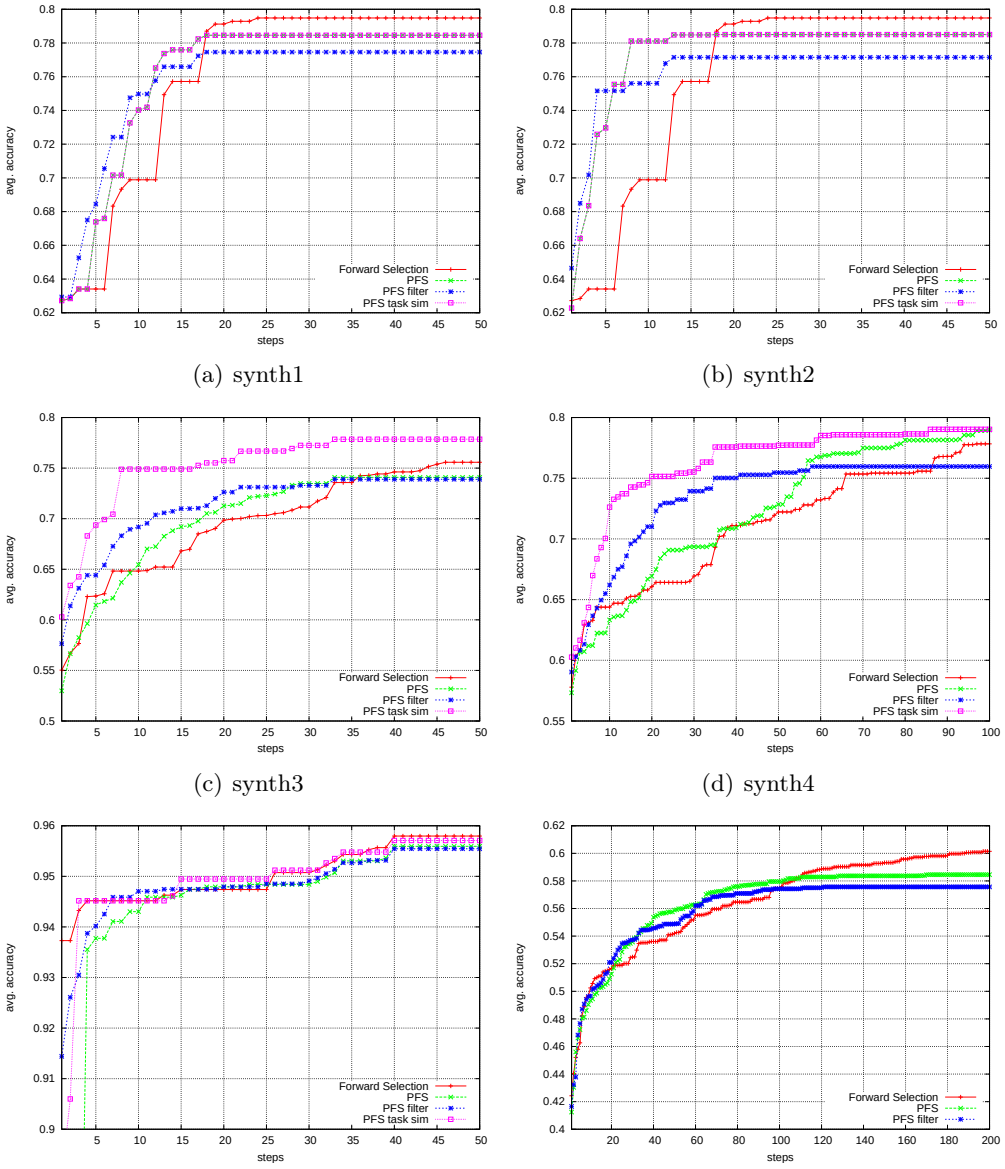
Pre-selecting features with a filter approach mostly leads to small feature sets and fewer invocations of the inner learning and evaluation procedure. This comes, however, at the price of a reduced accuracy on almost all datasets. The reason for this behavior is the same as the reason why simple filter approaches in general lead to reduced accuracy compared with wrappers, namely that feature interactions are not taken into account. Somewhat surprising is the fact that on several datasets, simple PFS performed better than PFS with pre-selection. By greedily selecting only few features in the first step of PFS, more features have to be selected in the second step, which may lead a higher number of features and decreased accuracy. This supports that assertion (F1) does not hold.

While pre-selecting features with a filter approach suffers from the limitation of ignoring feature interactions, this is not the case for pre-selection based on task similarity. In this case, features are rather selected in blocks of features that were selected together. This is also reflected in the results. Using pre-selection based on task similarity never leads to a lower accuracy than simple PFS. Indeed, it may lead to higher accuracy. The same holds for the number of features and the number of invocations of the inner learning and evaluation procedure. The reason is that PFS, just as traditional forward selection, works greedily. This may lead to sub-optimal solutions, because there is no „backtracking“ step. By selecting promising features in blocs, the search for optimal features is guided in a more effective way. This supports claim (F2). Alternative and irrelevant features do not have a negative effect on this result (see datasets synth2 and synth4), which supports assertion (F3).

A critical step for the use of task similarity is to adjust the task similarity threshold. In this work, this was done semi-automatically based on the similarity histogram. Figure 4.6.3 shows this histogram for dataset synth3. There is a clear gap between 2.3 and 3.5, which is then the choice for an optimal split point. In this case, the threshold was therefore chosen to 3.

Another parameter that plays an important role is ϵ. This parameter can be used to trade-off accuracy and the size of the aggregated feature set. Figure 4.6.4 shows this exemplarily for the dataset garageband. This supports claim (P1), namely that the size of the accumulated feature set and the overall accuracy are in conflict to each other.

In a second step, the communication costs produced by the feature facilitator and by the p2p feature sharing approach were analyzed. Table 4.11 and table 4.12 show the results.



(a) synth1 (b) synth2

(c) synth3 (d) synth4

(e) register (f) garageband

Figure 4.6.2: The development of the accuracy in the course of feature selection. PFS helps to increase the accuracy quickly. Later, traditional feature selection usually leads to a slightly better final accuracy, because PFS additionally tries to reduce the size of the accumulated feature set, sacrificing some accuracy.

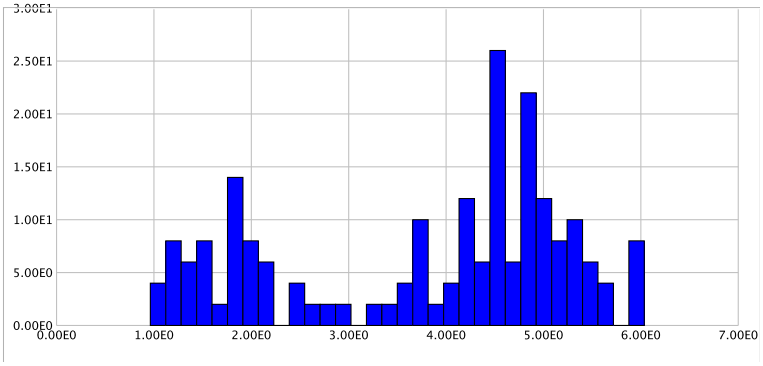


Figure 4.6.3: The histogram of task similarities for synth3. This graph can be used as a way to find a suitable task similarity threshold.

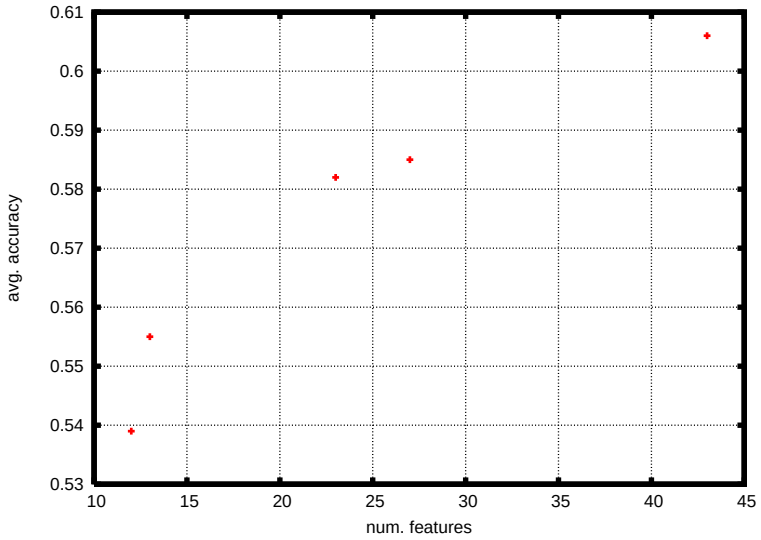


Figure 4.6.4: A plot of Pareto optimal points for the garageband dataset that shows that accuracy and the number of features in the accumulated feature set can be balanced using the parameter ϵ.

	synth1	synth2	synth3	synth4	garageband	register
FS	0	0	0	0	0	0
PFS	400	220	1420	2930	840	8400
PFS task sim	406	226	1045	1560	314	-
PFS filter	923	727	3348	3233	3293	11584

Table 4.11: Communication costs of the different approaches for the feature facilitator case. The parameter settings are the same as indicated above.

	synth1	synth2	synth3	synth4	garageband	register
FS	0	0	0	0	0	0
PFS	410	200	3050	4830	1270	20380
PFS task sim	500	290	2515	4170	1046	-
PFS filter	1349	1212	12336	13232	7364	110848

Table 4.12: Communication costs of the different approaches for the p2p case. The parameter settings are the same as indicated above.

We can see the following. In general, costs are much higher for the p2p approach. This is inherent to this approach, because usually all agents have to communicate with all other agents, such that the communication costs increase non-linearly with the number of tasks (which is here assumed to be the number of agents). Communication costs are also higher for datasets that involve more attributes, which is also inherent to the approach. More precisely, the costs increase with the number of selected attributes. These are, for instance, less for synth2 than for synth1. This explains why synth2 requires less communication costs than synth1, although synth1 contains fewer attributes.

The filter approach produces very high communication costs, because values have to be transferred for each example. Pre-selection based on task-similarity, on the other hand, produces very few costs. This is because the overhead of broadcasting a single vector (the base weights) is quite small. On the other hand, only the features of a subset of agents have to be transferred. This approach is only slightly worse compared to PFS on synth1 and synth2, because in these cases, all other tasks are relevant, such that there is no advantage over simple PFS.

Based on these results, (D1) is supported, however, in case that all other tasks are relevant there will be no savings. (D2) is not supported by these results. Filter-based pre-selection leads to higher costs than actively disseminating features in almost all cases.

4.6.4 Summary

To summarize the evaluation, we can say that if the total number of features and the time in which an optimal feature set is found play a role, then PFS should be used instead of traditional forward selection. The performance of PFS can be further increased by pre-selecting features using task-similarity. Especially in fully distributed settings, this may lead to a substantial decrease of communication costs.

4.7 Conclusion

Data representation is a key to successful and accurate data mining. In some special cases, there is a representation that works for a wide range of tasks in a domain. An example is text classification, where tf/idf weighting yields high accuracy for many different tasks

[85]. In most application areas, this is not the case. Mostly, optimal features depend on the characteristics of a specific task and a specialized representation has be found for each task individually in order to increase the accuracy. Still, one can often expect that some tasks in a domain are related to each other.

Based on existing research on the feature extraction problem for classification, a generalization was given, that treats the data mining task as black-box, assuming only that it is possible to asses the accuracy by which this task is achieved. This extension makes feature extraction applicable not only to classification but to other tasks as well. Notions, such as feature relevance or redundancy, were generalized accordingly.

In a next step, the case of several learning tasks was analyzed. Again, the notion of feature relevance and redundancy were generalized. Then, the problem of finding an optimal and minimal set of feature sets for several tasks was formulated. Features for each task should be selected in such a way that the sum of accuracies is maximized while the union of the features used for each task is as small as possible. It was shown that this problem is solvable but in the worst case exponential in the number of potential features. Furthermore, if we assume constraints on the order in which tasks are solved, then we can show that there is no algorithm that can guarantee to yield an optimal solution.

Based on this analysis, a heuristic algorithm was proposed, called prioritized forward selection. This algorithm first selects features that were already selected by other tasks and adds novel features only if they lead to a considerable improvement. This algorithm serves two purposes. First, it helps to find a minimal set of feature sets, second it can make the process of feature extraction more efficient, because features that are known to work well for other tasks are tried early in the extraction process.

The basic prioritized forward selection algorithm was optimized by pre-selecting only relevant features or only features from similar data mining tasks. To this end, a similarity measure for data mining tasks was developed, that allows for efficient implementation.

Finally, it was shown how the proposed methods can be implemented in a p2p network. The first approach to do so is based on a central node, called feature facilitator, that collects feature information from the individual nodes. The second approach is fully distributed and based on range limited broadcast. Both approaches were analyzed concerning computational effort and network traffic. They were empirically compared to active information dissemination, which is the current state-of-the art in p2p data mining.

Distributed feature extraction is directly related to collaborative structuring. In both cases, the representation of items is essential, and, in both cases, we must assume that this representation is different for each user or each task. Still, we would like to share information among tasks and users. Therefore, this chapter provides the theoretical and practical foundation for implementing collaborative structuring in the subsequent chapters.

5 Social Annotation and Feature Extraction

5.1 Introduction

In the last chapter, we analyzed the general problem of finding an optimal representation for several, partially related data mining tasks. In this chapter, we will focus more specifically on a special kind of features, namely *social annotations*. Most current web based information systems allow users to annotate items with arbitrary textual descriptions, called tags. These tags complement structured meta data. A system for audio organization, for instance, would usually store artist names and song titles as part of the meta data. Users could then additionally assign tags, such as „rock“, „party“, „smooth“ etc., to the audio items. These tags can be highly personal and situation dependent. The simplicity to add tags in an ad hoc manner, without having to obey an existing ontology, makes tagging systems extremely successful.

While tagging is very attractive for its simplicity, tags quickly become chaotic and hard to manage as the size of an information collection and the number of tags grow. This motivates the development of new formalisms and intelligent methods that help users to create, organize and maintain tag structures in a semi-automatic way. As we will see below, this can best be achieved by exploiting tags and tag structures created by other users. This approach is denoted as *collaborative structuring* in this work. This term is derived from collaborative filtering, just that collaborative structuring does not focus on recommending items but on recommending structure on items.

In the following, we will first give a short introduction to tagging and discuss why tagging offers some unique advantages from a user point of view (Section 5.2). We will then discuss the limitations of current tagging formalisms, and present an extended formalism, called *aspect-based tagging* (Section 5.3). We will argue that social annotations can be seen as features extracted by users and discuss how social annotations can be transformed into sets of binary or continuous features (Section 5.4). These features could be used in any data mining task, just as described in more general in chapter 4. For the case of social annotations, we identify two especially interesting tasks. The first one is to assign existing tags to new items automatically, based on supervised classification (Section 5.5). The second task is to structure items, not sufficiently structured yet, in an unsupervised way (Section 5.6). As will be shown, this problem can be formulated as a variant of cluster ensembles described in section 1.4.6. Both methods can be applied in a peer-to-peer network as specializations of the approach proposed in chapter 4 and show superior results

to using only content-based features. We will, however, argue that defining clustering in the traditional way is still not sufficient to support social annotations. Therefore, we will give a novel definition of the clustering problem, based on existing social annotations in chapter 6. The chapter is concluded by an empirical evaluation of the classification and the clustering approach (Section 5.7) and a summary (Section 5.8).

5.2 Social Bookmarking vs. the Semantic Web

Both collaborative filtering and link analysis are limited in their ability to capture semantic properties of underlying resources. There are two major approaches to overcome this limitation, the semantic web and web 2.0 based tagging systems.

5.2.1 Social Bookmarking Systems

Social bookmarking is an approach that allows users to semantically annotate items in a very loosely coupled way. The idea is that users tag items, such as webpages or music, with arbitrary chosen textual descriptions. These descriptions might be highly personal, such as „music for work“. Tags are globally assigned to items, such that other users can see them and exploit them to search for specific items or to browse them. Also, items are shown in conjunction with all tags that were assigned to them. In this way, a user can survey under which tags an item was classified by other users. Social bookmarking systems are widely used for two reasons. The first one is rather selfish. Users want to organize their own item collection in a convenient and intuitive way on the Internet (across different hardware or software platforms). The second one is benevolent. Users want to contribute to the common knowledge about web resources in order to help other users to find interesting information. Often, both processes are mixed up in a system, or even in the tags of a single user [70]. It is, for instance, quite usual that tags reflect rather personal concept, e.g. „music_for_michael“. Such tags are not of direct value for other users. If we apply corresponding data mining methods, still such tags can be used to support other users. This will be discussed below.

Many systems have emerged since the first tagging systems became popular around 2004. Currently the most popular system is del.icio.us[1]. It allows users to save their bookmarks on a central server, making them available from any Internet enabled computer. The main feature of the system is, however, the ability to annotate links with tags and descriptions. Both can be chosen arbitrarily. Entering tags is supported by an auto completion feature, based on tags that have already been entered. The system allows to classify links as public or private. The default is public, thus other users can see all tags and notes on a link. Users can search the global set of tags for keywords or, alternatively, just their personal collection of tags. Del-icio.us also allows users to directly recommend links to other users. Furthermore, users can be grouped into networks. It is then possible to share bookmarks

[1] http://www.del.icio.us

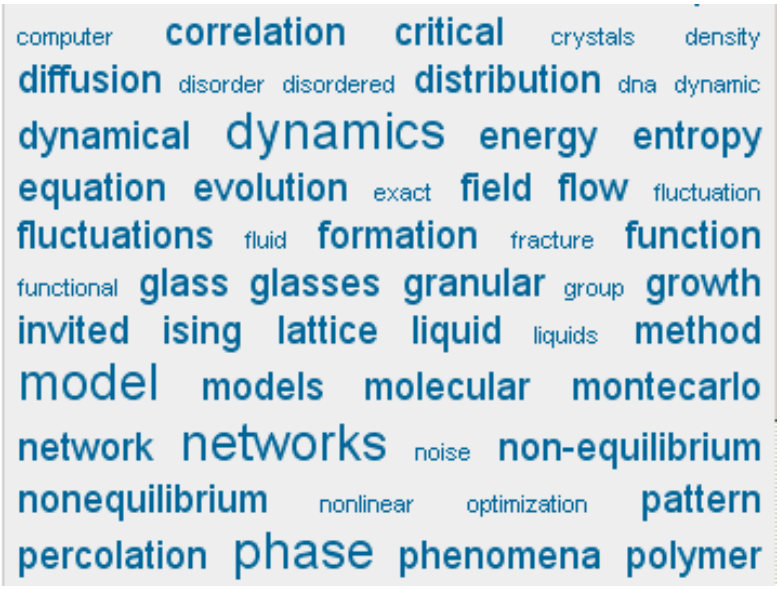


Figure 5.2.1: Tag clouds are used to visualize tags. The bigger the font, the more users assigned this tag.

only with the members of this group. Another interesting feature is a subscription and notification service. Users can subscribe for bookmarks in a network but also for very specific bookmarks, e.g. all links tagged with „kdd“ by a colleague. Del.icio.us also offers a back-end API for access by external programs, making it very convenient to link it to other applications.

Many other social bookmarking systems build upon this basic functionality. They do, however, provide some additional features. The Wink[2] system, for instance, aggregates results from several external sources in a mash-up like way. This usually leads to search results that are better than the results found by only searching tags. Furthermore, Wink uses a relevance feedback mechanism, allowing users to rate results. On future searches for the same tags, results are ranked according to this feedback. This approach is sometimes referred to as „PeopleRank“.

The Furl[3] system allows users to edit additional meta information on links, such as author, source, etc. Also, it allows users to leave comments on other peoples bookmarks. This can be convenient if the user wants to annotate links with information of the kind „if you liked this, be sure to take a look at that page too“. Also, Furl provides a page cache, such that there is copy of the original page. This can be very helpful for pages that change quickly. Finally, Furl offers to export bookmarks to different formats, e.g. for local backup.

While the systems mentioned so far focus on managing and searching bookmarks, Stum-

[2] http://wink.com
[3] http://www.furl.net

bleUpon[4] directly recommends web pages to the user, based on the tags the user entered and on explicit categories provided through a preferences dialog. Items are categorized by their type, e.g. audio, video, etc. They can also be filtered, e.g. by criteria, such as how the user found the page, whether it is a page the user liked, etc.

Other systems provide even more features, such as thumbnails of pages and support for blogs. One feature that is missing in many systems is the ability to create folders. In a recent Wired review of social bookmarking systems[5], this is found one of the most unsatisfying points. An exception is the Spurl[6] system, allowing for the creation of simple categories. An overview on the functionality of different social bookmarking systems can be found on the social bookmarks' charts[7].

Besides tagging bookmarks, there are systems that support sharing and annotating multimedia data. Examples are flickr[8] for sharing images and YouTube[9] for sharing video clips. They differ from other tagging systems in that the provider of the content is usually the same person as the one that creates the (initial) tags. This focus on very personal content also leads to more personal description of the content in terms of tags.

From a point of view of information structuring and retrieval, tags are not a new concept. In digital libraries, for instance, documents can often be annotated by „open keywords“ in addition to a predefined classification scheme.

Indeed, on a formal level, a tag is nothing more than a concept $c \in C$ such that $ext(c)$ contains exactly all items that were tagged with c. The set of all tags C may contain tags that overlap in any possible way. Therefore, tagging is more flexible than partitions or taxonomies.

The major difference to traditional approaches is the availability of tagging systems, allowing a wide range of users to tag items in a very convenient way. The ease of use and the ability to create arbitrary descriptions without the need of explicit coordination with other users make them very appealing for „ad hoc“ use. On the other hand, social structure is made explicit, such that users can see what tags other users assigned, subscribe for tags, form user groups, etc. The simplicity and the social aspects make the rather simple concept of tags a powerful tool for collaborative information organization.

5.2.2 Key Advantages of Social Bookmarking Systems

The most important approach to a formal representation of Internet resources is the semantic web [20]. It is mostly based on Description Logics, as described in section 1.5. The semantic web has shown to be not well suited as representation mechanism for

[4] http://www.stumbleupon.com
[5] http://wired.com/news/technology/Internet/0,72070-0.html
[6] http://www.spurl.net
[7] http://www.roxomatic.de/archives/985/Social-bookmarks-review-version-3.5
[8] http://www.flickr.com
[9] http://www.youtube.com

collaborative information organization. It is quite complex and requires explicit coordination among users. The co-existence of views and emerging views is supported only indirectly. Also, because the representation mechanism is quite powerful, operations may become inefficient. It is based on logical entailments that are not always comprehensible for regular users.

Furthermore, many operations provided by DL are not fully relevant for information organization. First, the representation of complex relationships is not of essential importance. Regular users are not capable of dealing with such complex relationships (the large majority of Google users never even apply simple logical operators in their search requests). Also, the domains in question do often not contain complex relations. Most properties can be simply expressed by pairs of attribute and value (artist, year of publication, ...). Furthermore, sophisticated logical inference is often not useful because most users express their knowledge rather ad hoc and do not accept logical entailment of what they expressed. We do not claim, however, that these properties are irrelevant in general, we only claim that they are not essential for user-driven web information organization.

In the following, we will argue why social bookmarking systems are very well-suited for collaborative media organization, which in turn also explains their success in this application domain.

We can identify the following key advantages of social bookmarking systems over logic-based approaches:

1. *No explicit coordination*
   The growth of the Internet can be attributed largely to its loosely-coupled character. If, for instance, every owner of a site would have to agree that someone else links to her site, the Internet would probably not have grown as fast as it did, nor would approaches like link analysis be as powerful as they are. Tags can be assigned, just as weblinks, without any coordination with other users or existing tags.
2. *Co-existence of different views*
   Often, users do not agree on how to structure a certain set of items. Moreover, a single user may structure the same set of items using different aspects. It is therefore essential that different representations of the same items may co-exist. In the extreme, each user should be allowed to create views completely independently of all other users. This allows for bottom-up innovation, because each user is capable of creating novel views. Which views become popular should emerge automatically, just like popular web pages emerge automatically if many other pages link to them. Tagging supports arbitrary views on item collections by not constraining the tags users choose.
3. *Support for data mining and mediation*
   While using loosely coupled representations is very attractive, the question is how to derive useful information from such heterogeneous views and to allow users to profit from what other users did. A representation mechanism should therefore allow for the successful application of data mining and mediation methods. Tags

can be incorporated into data mining easily. This will be shown in the remainder of this chapter.

4. *Efficiency*
   Relevant operations must be executable efficiently. For information management, the most important operations are the retrieval of items, basic consistency checks and the application of data mining methods, such as automatic classification. Tags are very efficient to manage, even if their number grows to several billions. This is not true for DL based mechanisms.

5. *Ubiquitous environment*s
   The mechanism must be applicable in highly distributed environments. It must not expect that all nodes are connected to the network all the time. Also, distributed data mining and retrieval methods must be applicable in a way that keeps the overall effort in communication time and cost low, because information is often organized on computationally poor devices connected by loosely-coupled networks (such as p2p or ad hoc network).

Still, tagging approaches must be extended to make them maximally useful. First, users should be supported in creating and maintaining tag structures. Tagging some items is usually fun, but tagging large sets of items can be rather labour intensive. Automatic support includes, for instance, tagging new items automatically with user defined tags or with tags obtained from other users. This will be described in section 5.5 and section 5.6. Another extension, helping users to organize large tag collections, is to allow users to arrange tags into groups of tags. This approach will be described in the next section.

5.3 Aspect-Based Tagging

A major drawback of tagging as a formalism is that tags tend to become rather chaotic and hard to manage as the number of tags and the number of items increases. We therefore describe two extensions to the basic tagging formalism that help to make tags more manageable:

1. We allow tags to be arranged *hierarchically*. Such a hierarchical structure enables users to overview even large numbers of tags without getting lost.
2. We allow to group tags into sets of tags called *aspects*. An aspect could, for instance, be „genre" for audio organization. Tags can be overviewed much easier if they are filtered by individual aspects. Also data mining methods profit from this concept (as will be shown below).

We denote the resulting mechanism as aspect-based tagging. Figure 5.3.1 depicts the differences between both approaches. A formal description of aspect based tagging will be given in the next section[10].

[10]Recently, some social bookmarking system came up with similar ideas. del.icio.us (http://www.del.icio.us) supports, for instance, tags of tags, that are similar to aspects. Some sys-

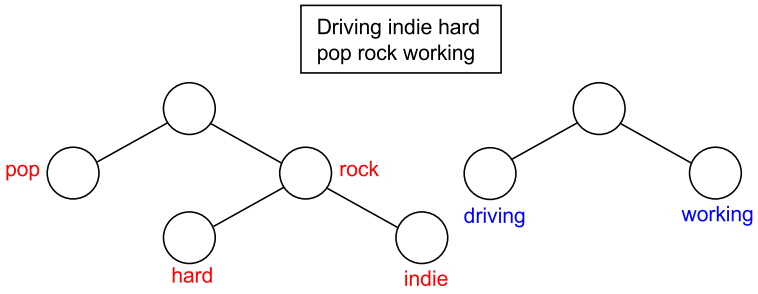


Figure 5.3.1: Aspect-based tagging allows to structure tags hierarchically into different aspects. This makes them easier to overview and organize.

5.3.1 (Personal-) Hierarchical Tag Structures

As described in section 1.2, we assume a set of items D. These items can be annotated by features and can be grouped into item collections. Also, we assumed a globally uniquely identified set of users. Users assign tags to items in order to structure them. These tags are arbitrary textual descriptions. Furthermore, users assign exactly one aspect to each tag (see figure 5.3.1). In the following, we will formalize these notions and show how they relate to traditional description logics and current web 2.0 tagging applications.

All tags are assumed to be private, thus even if two tags have the same syntactic name, they are assumed to be different if they are owned by different users.

Definition 5.3.1. A *tag* is a uniquely identified term that is owned by exactly one user. The set of all possible tags is denoted as $\mathcal{C}$. At each point in time, we assume a finite set of actually existing tags $C \subseteq \mathcal{C}$.

A tag is assigned to a set of items by a *tag mapping function*.

Definition 5.3.2. A *tag mapping function ext* assigns a set of items to a tag $c \in C$:

$$ext : C \rightarrow 2^D$$

Beside assigning tags, users are allowed to create hierarchical relations among tags.

Definition 5.3.3. A *sub-tag relation* is a transitive, strict partial ordering

$$\prec \subseteq C \times C$$

where $c \prec c'$ denotes, that c' is a *super-tag* of c (and c is a *sub-tag* of c').

For tags that are in subset relation, the following additional conditions must hold.

tems, such as bibsonomy (http://www.bibsonomy.org), introduced hierarchical relations.

Condition 5.3.4. All items that belong to a sub-tag must be assigned to the super-tag as well.

$$\forall c, c' \in C : c \prec c' \Rightarrow ext(c) \subseteq ext(c')$$

Note that the other direction must not hold, in general. Thus, the fact that two extensions are in subset relation does not yet entail that they are in sub-tag relation. Note that this assumption is weaker than the one made for general taxonomies in chapter 1.

We also assume a set of *aspects*. For these aspects, the same holds as for tags. They are assumed to be private. If two aspects share the same name, they are still treated distinctly if they are owned by two different users.

Definition 5.3.5. An *aspect* is an uniquely identified entity that is assigned to exactly one user. The set of all aspects is denoted as $\mathfrak{A}$. At each point in time, we assume a finite set of actually existing aspects $A \subseteq \mathfrak{A}$.

Each tag is assigned to exactly one aspect by an *aspect mapping function*.

Definition 5.3.6. An *aspect mapping function* is a function that assigns each tag $c \in C$ to exactly one aspect in A:

$$asp : C \rightarrow A$$

As an inverse to this function we define

$$C(a) = \{c \in C | asp(c) = a\}$$

thus the set of all tags belonging to a given aspect.

We require tags to fulfill some additional conditions.

Condition 5.3.7. Given an aspect $a \in A$, the tags belonging to this aspect, $C(a)$, must form a tree concerning the sub-tag relation.

Thus, all tag structures must form tag trees.

Condition 5.3.8. There may not be any cross links among tags belonging to different aspects.

$$\forall c, c' \in C : c \prec c' \Rightarrow asp(c) = asp(c')$$

Thus, only such subset relations among tags are allowed that belong to the same aspect. This condition has an important implication. Relations among tags of different users are not allowed, aspects are private to users. This condition is essential to ensure the loosely

coupled character of the approach. Coupling tags of different users would entail coupling network nodes, which leads to severe problems as pointed out above.

Aspects partition the set of tags C into several sets of tags. Given the above restrictions, each such set forms a tree of tags under the sub-tag relation. We will refer to these trees as *tag trees* or *tag structures*. An individual tag in such a tree is denoted as *tag node*. From the above it is obvious that each tag structure belongs to exactly one user. A user may, however, own more than one tag structure.

Below, we will need still another concept, the one of a tag tree *covering* an item.

Definition 5.3.9. An aspect or tag structure $a \in A$ *covers* an item $x \in D$ if any tag node in the tag tree covers the item

$$x \sqsubset a \equiv \exists c \in C(a) : x \in ext(c)$$

Then we can define the set $S_a \subseteq D$, as $S_a = \{x \in D | x \sqsubset a\}$, thus the set of all items covered by an aspect.

As each tag tree must have a root node, this is equivalent to saying that the item must be assigned to the root node of a tag tree.

A stricter restriction is that if two tags have a common predecessor (are in the same tag structure) and overlap (there are items assigned to both) they must be in relation to each other.

Condition 5.3.10. Tag nodes $c, c' \in C$ may not overlap, unless they are in sub-tag relation:

$$\forall c, c' \in C : asp(c) = asp(c') \wedge ext(c) \cap ext(c') \neq \emptyset \Rightarrow (c \prec c') \vee (c' \prec c)$$

This condition expresses that if two tags belonging to the same aspect have overlapping extensions, they must be in relation to each other. This is not a severe restriction, because users are allowed to create arbitrary many aspects.

5.3.2 Classification Functions

Based on the above assumptions and conditions, we can write tag structures in another way, that will be more convenient for the presentation of data mining algorithms in the second part of this chapter. Instead of considering tags that are assigned to aspects, we now consider aspects built up by tags. All tags belonging to an aspect form a tree structure. We can now rewrite an aspect as *classification function*.

Definition 5.3.11. A *classification function* is a function $\varphi : S \rightarrow G$ that maps items $S \subseteq D$ to one member of a (finite) set of groups G.

Groups are very similar in notion to tags or concepts. They must, however, not necessarily obey the same restrictions that were defined for tags and are therefore more general. In the following, we will usually deal with groups that are in a hierarchical relation to each other.

Definition 5.3.12. A classification function $\varphi : S \rightarrow 2^G$ induces a *group hierarchy* $<\subseteq G^2$ on the groups in G, with $\forall g, g' \in G : (g < g') \Leftrightarrow (\forall x \in S : g \in \varphi(x) \Rightarrow g' \in \varphi(x))$. $(G, <)$ must be a tree.

The function φ is then called a hierarchical function.

Definition 5.3.13. A *hierarchical classification function* is a classification function $\varphi : S \rightarrow 2^G$ that maps items $S \subseteq D$ to a subset of groups in a group hierarchy G.

Now, we can rewrite aspects as *hierarchical classification functions*.

Definition 5.3.14. An *aspect function* is a hierarchical classification function that maps items to tags associated with an aspect $a \in A$. Thus, $\varphi_a : S_a \rightarrow 2^{C(a)}$.

Regarding tag trees as classification functions helps to link them to more traditional data mining tasks, such as clustering or classification, which is the main intent of this work.

5.3.3 Aspect-based Tagging and Description Logics

As argued above, social bookmarking is much better suited and more successful for collaborative media organization than the semantic web, that is based on Description Logics.

On a formal level, a very interesting question is how both formalisms are connected.

It is easy to see that traditional tagging approaches, such as described in section 1.5, are basically a subset of Description Logics (DL). They simply support only atomar ABox entries of the form $c(x)$, denoting that $x \in D$ is tagged with the concept $c \in C$. A TBox is not supported. In this sense, a DL with an empty TBox could be used to implement a tagging system, it would, however, not make much sense to do so, because some operations can be quite expensive in DL, that are trivial if we face only a set of overlapping, atomic concepts.

For aspect-based tagging, the situation is slightly different. There is no simple way to implement aspect-based tagging using a DL. The reason is the co-existence of assertions like:

$$x \in ext(c)$$

denoting that x is tagged with c, and propositions like

$$asp(c) = a$$

denoting that c belongs to aspect a.

The only possibility to denote this in an traditional DL is to use two pairs of TBox and ABox. A first one using the set of items D as universe of discourse and the second one using the set of possible tags C as universe of discourse. The second TBox would contain meta-logic expressions, thus terminological constraints on terminology.

5.4 Feature Extraction from Social Annotations

User implicitly define features on items by assigning tags. These features will be denoted as *collaborative features* in the following. Such features are very powerful for many tasks, because they directly describe items from a point of view of the user. In this section, we will analyze how user annotations can be converted into feature spaces to be used in various data mining tasks.

5.4.1 (Mixed) Collaborative Feature Spaces

Definition 5.4.1. Given a tag $c \in C$, a *collaborative feature* is a vector

$$X_c : D \to \mathbb{R}$$

Note that D is assumed to be finite.

Given more than one concept, we derive a *collaborative feature space*.

Definition 5.4.2. Given a set of tags $C \subseteq \mathcal{C}$, the set

$$\mathbf{X}_{\mathrm{C}} = \{X_c : D \to \mathbb{R} | c \in C\}$$

is called a *collaborative feature space* derived from C.

Naturally, features extracted from tag structures can be combined with features obtained from other sources, e.g. meta data. We refer to such a feature space as *mixed collaborative feature space*.

Definition 5.4.3. Given a set of tags $C \subseteq \mathcal{C}$ and a set of features $\mathbf{X}$, the set

$$\mathbf{X}_{\mathrm{C}}{}^* = \{X_c : D \to \mathbb{R} | c \in C\} \cup \mathbf{X}$$

represents a *mixed collaborative feature space*.

In the following, we will discuss how to derive collaborative features from a given set of tag structures.

5.4.2 Representation Axioms

A mapping from a set of tags C to a set of collaborative features can be seen as a *representation function*. A representation function is a mapping from one language $\mathcal{L}_1$ to another language $\mathcal{L}_2$. Usually, such a function should fulfill certain constrains, such as being bijective. But there are also additional restrictions possible.

Restrictions on the source language (together with assumptions on the representation function) lead to restrictions of what constructs can occur in the target language. If we know, for example, that every tag structure will include only one tag assigned to all items, this certainly constraints the possible values of the features.

Restrictions on the target language work just the other way round. If the target language should fulfill some constraints, then some other constraints have to be fulfilled by the source language. A famous example are the rationality axioms [127]. The task is to map ordinal preferences to an interval scale. To ensure that the target is actually interval scaled, the ordinal relations in the source language have to fulfill the rationality axioms.

In the following, some basic conditions are assumed that constrain the possible representation functions for tag structures.

Condition 5.4.4. A representation for tags must fulfill the following conditions given a tag $c \in C$ and a corresponding feature X_c:

1. $x \notin ext(c) \Rightarrow X_c(x) = 0$
2. $x \in ext(c) \Rightarrow X_c(x) > 0$
3. $0 \leq X_c(x) \leq 1$

The third condition ensures that extracted features have a range between zero and one, which is an important property for the application of many methods to the extracted features. The second assumption is important to ensure reversibility of the interpretation (see below). Finally, the first condition encodes the closed world assumption. If a user does not assign a tag to an item, this tag is not considered relevant for the item. The underlying assumption is that the user knows all items in D. While this is often not true, the closed world assumption was shown to be useful in many applications, such as in the well-known vector space model [154] for text processing, an area closely related to tagging.

In the following, several mappings fulfilling these basic conditions will be presented and analyzed.

5.4.3 Extracting Binary Features

A simply representation function fulfilling the above constraints is the extension function itself.

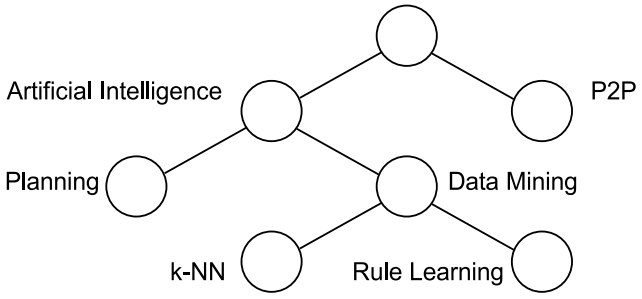


Figure 5.4.1: Topics that are positioned lower in the tree are assumed to be more specific. Data Mining, for instance, can be considered a subtopic of Artificial Intelligence and is thus more specific.

Definition 5.4.5. A *binary representation function* maps all tags in C to binary features on items $x \in D$

$$X_c(x) = \begin{cases} 1, x \in ext(c) \\ 0, else \end{cases}$$

Lemma 5.4.6. *Assuming that every tag has an extension larger than one and that every inner tag node has at least two children with different extensions, then the binary representation function is bijective.*

Proof. (Idea) With the assumed properties for tags and aspects, the above mapping is injective. An inverse function can be constructed as follows. Given a set of collaborative features, construct C such that there is exactly one concept for each feature, containing all items for which this feature has value one. The sub-tag relations can then be derived from the subset relations on *ext* for each aspect separately. Please note that the aspect information is crucial in this process. □

A restriction of this simple interpretation function is that it does not make use of the relative position of items and tags in the tag tree. The relation between a tag and an item might be more or less strong, depending on how close they are.

5.4.4 Making Use of the Position in the Tag Tree

Items, as well as tags, can be more or less general. A general scientific document, for instance, would give an overview on a whole research area, while a specialized one only focuses on a very specific problem.

Definition 5.4.7. The *specificy of a tag* is defined as function

$$spec : C \rightarrow \mathbb{R}_0^+$$

This notion is rather abstract in general. For an illustration, take a look at figure 5.4.1. Instantiations will be given below. Similar to tag specificy, we can define a degree of specificy for items

Definition 5.4.8. The *specificy of an item* is defined as function

$$spec_D : D \times A \to \mathbb{R}_0^+$$

Items can be covered by more than one aspect. Thus, the specificy of items, in general, depends on the aspect. In the following, we assume that the specificy of an item x is equal to the one of the most specific tag that is assigned to it, given a tag structure $a \in A$.

$$spec_D(x, a) = max_{c \in \varphi_a(x)}\{spec(c)\}$$

We assume the following condition about tag specificy.

Condition 5.4.9. For each pair of tags $c, c' \in C$ the following must hold

$$c \prec c' \Rightarrow spec(c') \leq spec(c)$$

Thus, the specificy must increase monotonically with the tree depth of a tag in the tag tree.

There are two popular measures of the specificy of nodes in a subset hierarchy: tree depth and information content.

Definition 5.4.10. The *tree depth based specificy* of a tag $c \in C$ is defined as

$$spec(c) = |\{c' \in C | c \prec c'\}| + 1$$

The tag specificy axioms are trivially fulfilled. The assumption is that the specificy rises linearly with the tree depth. A tag node twice as distant from the root is twice as specific.

Definition 5.4.11. The *information content based specificy* of a tag $c \in C$ is defined as

$$spec(c) = -log(\frac{|ext(c)|}{|ext(c_{root})|})$$

where c_{root} denotes the root tag of the tag structure that belongs to the same aspect as c. Furthermore it is assumed that $\forall c \in C : ext(c) \neq \emptyset$.

The idea of information content based specificy is that a tag is more specific if it is assigned to less items. The root node reaches a minimum of specificy, as it is assigned to all items. See [111] or [150] for further discussion of information based measures in information organization.

Lemma 5.4.12. *The information content measure fulfills the tag specificy axioms.*

Proof. From $c \prec c'$ we obtain $ext(c) \subseteq ext(c')$ and thus $|ext(c)| \leq |ext(c')|$. □

This measure may also fail. If, for example, a given topic is very general, but there are only few items available for this topic, then users can only assign the corresponding tag to few items and it will seem very specific, while it actually is not.

Based on specificy, the binary interpretation can be refined in several ways, two of which will be discussed in the following.

1. *Tag Weighting*

$$X_c(x) = \begin{cases} spec(c), x \in ext(c) \\ 0, else \end{cases}$$

   Only the specificy of the tag is regarded as weight for the degree of relevance. The consequence is the following: If an item is specific, a specific tag gets a higher weight than a general one. If the tag is general, then a specific item gets the same weight as a general one. This measure is derived from the family of similarity measures discussed in [150].

2. *Tag/Item Weighting*

$$X_c(x) = \begin{cases} \frac{spec(c)}{spec_D(x, asp(c))}, x \in ext(c) \\ 0, \ else \end{cases}$$

   We assume that $spec(c) \leq spec_D(x, asp(c))$ and $spec_D(x, asp(c)) > 0$. This measure reaches its maximum if the specificy of both, the tag and the item, are the same. If the specificy is measured by information content, this measure can be derived from the axioms stated in [111]).
   For the special case $spec(c) = spec_D(x, asp(c)) = 0$ it is assumed that $X_c(x) = 1$ if tag c is assigned to x.

Lemma 5.4.13. *If $spec(c) > 0$ for all tags, both representation functions are bijective under the same conditions as described above.*

Proof. See above. □

Lemma 5.4.14. *Tag based weighting does in general not fulfill the third condition for representation functions*

Proof. Counter example: Assume an item is assigned a most specific tag c. Unless $spec(c) = 1$, the third condition is violated. □

There is a problem of scale variance in collaborative filtering systems. Different users exploit the scale of possible ratings in different ways. A similar problem exists for the interpretation of tag structures.

1. *Tag structures as a whole can be specific or general*
   One user could create a tags covering the whole area of computer science, while another user creates tags denoting sub-topics of artificial intelligence. Both tag structures may have the same number of tags and the same depth.
2. *Tag structures can be more or less detailed*
   Even if tag structures start on the same level of specificy, they can still be more or less detailed. One tag structure could contain many layers between root and leafs, while another one may contain only few.

It would be desirable to use methods that are invariant against such effects. Using tag/item based weighting, at least the following holds:

Lemma 5.4.15. *Given two concepts $c, c' \in C$ for which $spec(c) = \beta spec(c')$ and $ext(c) = ext(c')$ holds, with $\beta > 0$. Then, for features extracted by tag/item based weighting, $X_c(x) = X_{c'}(x)$ holds, for arbitrary items $x \in D$.*

Proof. $X_{c'}(x) = \frac{\beta spec(c)}{\beta spec_D(asp(c),x)} = \frac{spec(c)}{spec_D(asp(c),x)} = X_c(x)$ □

Thus, a tag that is more detailed than another one may still deliver the same feature values and is thus invariant concerning the second problem. This does not hold for tag based weighting.

5.4.5 Tag Aggregation

Up till now, tags were treated as independent, private structures. This is, however, often not appropriate. If two users both create tags labeled „computer science“, then they will probably not have exactly the same concept in mind, though both tags will be usually somewhat related. We capture this relation by allowing to merge two or more features derived from different tags as to replace them by an aggregated feature.

Feature aggregation has been applied successfully in many applications. For text clustering, generalizing terms by adding superordinate terms can improve the quality of the result significantly [80]. The same holds for association rule mining. Adding generalized features, that combine individual items to classes, enables the algorithm to find patterns that would not be valid in the original data space [165]. A similar approach can be applied to merge features derived from tags. Again, the idea is to summarize several tags to create a more general tag.

The constructed feature space should still be easily interpretable in order to allow for a quick inspection of the results.

Definition 5.4.16. A *feature aggregation function* is a function $aggr : \mathbb{R}^2 \to \mathbb{R}$ that maps two features to a new feature.

Please note that the newly aggregated feature replaces the arguments.

In [123] a simple and intuitive axiomatic is given that constraints feature aggregation functions. The maximum function is the only function that fulfills all of the conditions. This is very intuitive, because maximum is a fuzzy t-co-norm [27] and represents an „or“ operation. Summarizing tags by maximum makes tags more general in the sense that the new tag covers more items than the original ones.

An open question is, however, which tags should be merged. In the following, we will distinguish between supervised and unsupervised feature aggregation.

Supervised Feature Aggregation

Supervised feature aggregation is based on a wrapper approach. Features are merged if this increases the performance of an inner data mining scheme, e.g. the estimated accuracy of a classification algorithm. As the number of possibilities to merge features grows exponentially with the number of features, usually efficient optimization algorithms will be applied for this task (as e.g. genetic algorithms).

Unsupervised Feature Aggregation

Unsupervised feature aggregation does not take any data mining task into account. Features are merged based on heuristics. We assume the following two heuristics:

1. Features derived from tags with the same syntactic name are merged. This assumes that all users have the same notion of the concepts connected to a tag. This is often to a large degree fulfilled, may, however, be misleading in some cases (e.g. concerning homonyms).
2. Two features are only considered for merging if they are derived from tags that have aspects with the same name. This is essentially the same assumption as for the first heuristic, but on an aspect level. As we assume the number of aspects usually much smaller than the number of tags, this heuristic can be applied in more cases in general.

Unsupervised feature aggregation is heuristic. Its applicability depends on the domain and has to be evaluated for each new domain. On the other hand, it is much more efficient than supervised feature aggregation.

Lemma 5.4.17. *Features can be aggregated by unsupervised aggregation in* $O(|C|\,|D|)$

Proof. Adding a feature value requires an access to the hash and exactly one maximum calculation. As the number of features is bound by $|C|$ and the number of items by $|D|$, the assertion holds. □

5.5 Collaborative Classification With Social Annotations

While tagging some items is often perceived as „fun“ by the users, tagging a complete collection of items is often rather seen as „work“. Automatic tagging is a solution to this problem. Users may only tag a small number of items and then leave it to the system to assign tags to the rest of the items. This is especially useful because items arrive one-by-one and are automatically added to the existing tag structures. This functionality can be easily achieved by creating one classification model per aspect. These functions can then be applied to any new item. The fact that tags are arranged into aspects is essential at this point, as we do not have to deal with one classification problem per tag but one per aspect.

5.5.1 Problem Definition

In the following we give a definition of the task of collaborative classification.

Definition 5.5.1. Assume that we are given a set of aspects A and a corresponding (mixed) collaborative feature space $\mathbf{X}_{\mathrm{C}}$. Also assume a set of aspects A' (e.g. all aspects of a given user), with $\varphi_a : S_a \rightarrow 2^{C(a)}$ for each $a \in A'$. The task of *collaborative classification* is to yield for each $a \in A'$ a function $\varphi'_a : D \rightarrow 2^{C(a)}$.

Thus collaborative classification uses a set of tagged examples for each given aspect a and a (mixed) collaborative feature space to derive for each $a \in A'$ a function φ'_a that is able to assign the tags associated with a to any item in D.

Note that if we were not given any aspect information, we would be forced to create a classification model for each user tag, instead of one for each aspect.

5.5.2 Collaborative Classification

Thus collaborative classification can simply be mapped to the known problem of hierarchical classification, just as described in section 1.4.4. As the classification problems induced by user tags are supervised, we can apply all available methods for feature extraction. Especially, methods for distributed feature extraction and feature sharing, as proposed in chapter 4, are directly applicable to the problem. Thus, an agent can profit in two ways from the presence of other agents. First, by querying other agents for collaborative features, it can extract new, mostly highly relevant collaborative features to enrich the item representation. Second, by using methods for feature filtering, agents can solve the representation problem connected with each classification task more efficiently.

Decision tree learners [147] are especially well suited to deal with data of the kind described above. Partially missing data can be handled easily. If, e.g., a given feature is only applicable to jazz music, and a first globally defined criterion separates these jazz items from the rest of the items, the specific jazz features can be applied even if they are

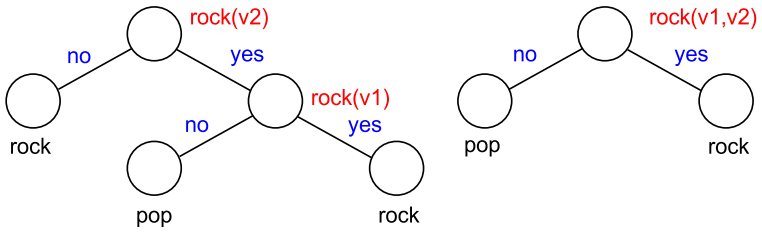


Figure 5.5.1: Decision trees to deal with disjoint concepts

undefined for the remainder of the items. Thus, decision trees are able to accommodate to local properties of the item space.

A second important advantage of decision trees is their ability to learn with disjunctions to some extent. Tags often cover different items belonging to the same underlying concept from the perspective of the classification. E.g. „rock(v1)“ defined by a first user might cover some of the items and „rock(v2)“ the remainder of the items in the training set. A decision tree, such as the one shown in figure 5.5.1 on the left, could deal with this problem directly. The decision tree on the left is based on two user defined tags rock(v1) and rock(v2) that are combined. The decision tree on the right is a simplified tree, obtained if rock(v1) and rock(v2) are aggregated into a new combined feature rock(v1,v2). This kind of aggregation is constructed in a data driven way that is described in section 5.4.5.

5.6 Clustering with Social Annotations

Features derived from tags may not only be used for classification but also for clustering. In this section, we show that doing so results in a natural extension of the co-association cluster ensemble approach described in section 1.4.6.

Clustering items can serve several purposes:

1. *Tagging a set of unstructured items*
   Given a set of items for which no tags are available, collaborative clustering can yield new tag structures for these items. This is very useful if no personal tags are available yet.
2. *Finding alternative descriptions of a set of items*
   Even if a set of items is already tagged, the user might want to explore additional possibilities to structure these items.
3. *Top-down refinement of tag structures*
   Users often implicitly adopt a top-down approach in structuring items. They start out with a set of items and then devise them recursively into subsets by assigning hierarchical tag structures. This process is supported by allowing to cluster items in a tag leaf by collaborative clustering.

In the following, we adapt the traditional clustering problem, as introduced in section 1.4.3, to the application on social features.

5.6.1 Problem Definition

We can define the task of collaborative clustering as follows.

Definition 5.6.1. The task of *collaborative clustering* is, given a set of aspects A, a corresponding collaborative feature space $\mathbf{X}_{\mathrm{C}}$ and a set of items to cluster S, to produce a set of concepts C' that covers the items in S ($\bigcup_{c \in C'} ext(c) = S$) and optimizes some objective function $cq(C')$ with respect to the feature space $\mathbf{X}_{\mathrm{C}}$.

Thus the aim of collaborative clustering is defined just as usual clustering with the exception that the feature space used is derived from social annotations.

As for classification, this is directly combinable with distributed feature extraction as proposed in chapter 4. In this case, however, we do not assume that candidates are selected or that prioritized forward selection is applied, because assessing the accuracy of a feature set given a clustering task is non-trivial (see section 4.2.2). Distributed feature extraction is in this case only used to build-up the collaborative feature space $\mathbf{X}_{\mathrm{C}}$ in a distributed way.

If we use only collaborative features, then collaborative clustering should intuitively deliver a result that reflects the input clusterings. More precisely, we would expect that all input clustering are somehow merged into the output clustering. In the following, we will show that collaborative clustering can be seen as an intuitive kind of cluster ensemble for hierarchical clusterings.

Cluster ensembles based on co-association count for every pair of items in how many input clusterings they appear in the same cluster. The resulting matrix is used as similarity measure for any similarity based clustering algorithm. As tag structures are hierarchical, this idea must be extended. A natural extension of the co-occurrence of two items in a cluster is the tree distance. It allows for a smooth transition between being in the same cluster and not being in the same cluster. The resulting distance measure could be chosen, for instance, as the average tree distance of each pair of items over all input cluster trees. While this would be a sound approach, it has the disadvantage that we have to deal with a matrix that has $|D|^2$ entries. Furthermore, there is few reason to assume that this matrix is sparse (a zero entry only occurs if two items are in the same cluster in every input clustering).

In the following, we show that we can achieve something similar by first extracting features using the approach presented in the last section and then to apply a standard similarity measure to the resulting collaborative feature space (section 5.6.2). We then show how this can be used for collaborative clustering (section 5.6.3).

5.6.2 Collaborative Similarities

Many clustering and classification schemes are based on distance or similarity measures on the underlying items. We can apply such a measure to a collaborative feature space as well. The question is whether this results in a sound measure, from the point of view of collaborative structuring. In the following, we will analyze this question for two particularly popular measures, namely squared Euclidean distance and inner product.

Euclidean Distance

Many algorithms for classification and clustering (such as k-means) are based on the squared Euclidean distance. If the squared Euclidean distance is applied to a collaborative feature space, the following is obtained.

Lemma 5.6.2. *Given a set of tag structures A that all cover exactly the same set of items S (thus $\forall a \in A : S = \{x \in D | x \sqsubset a\}$). $C = \bigcup_{a \in A} C(a)$ is the set of tags belonging to the aspects in A. Then, the squared Euclidean distance applied to the binary collaborative feature space $\mathbf{X}_C$ (denoted as $d_{\mathbf{X}_C}$) is identical to the sum of tree distances d_a over all tag trees $a \in A$ for any pair of items $x, y \in S$.*

$$\forall x, y \in S : d_{\mathbf{X}_C}(x, y) = \sum_{a \in A} d_a(x, y)$$

where $d_a(x, y)$ is the distance of x and y in the tag tree associated with a.

Proof. We first show that this property holds for a single tag tree. Given two items $x, y \in S$, tags in a single tag tree $a \in A$ can be classified into four sets. The ones that contain both items, $C_{xy} \subseteq C(a)$, the ones that contain x only, denoted as $(C_x \setminus C_{xy}) \subseteq C(a)$, the ones that contain only y denoted as $(C_y \setminus C_{xy}) \subseteq C(a)$ and the nodes containing neither x nor y. Features derived from the first and from the last set of tags are equal for x and y. They do not contribute to the distance. Features derived from the other two sets lead to a difference of 1 each, such that all features derived from the tag structure a lead to a distance of $|C_x \setminus C_{xy}| + |C_y \setminus C_{xy}|$. This is exactly the tree distance. As features that are derived from different tag structures in A are independent and the Euclidean distance is additive, it can easily be shown by induction over the set of tag structures that the resulting distance is the sum of all individual tree distances. □

Squared Euclidean distance leads to a result that is very close to the idea of using the average tree distance as similarity measure to cluster a set of items. More precisely, we can normalize the above distance measure by the number of aspects and obtain exactly the average tree distance.

Inner Product

Another popular similarity measure is the inner product. What happens if we apply this measure to a binary collaborative feature space?

Lemma 5.6.3. *Given a set of tag structures A that all cover exactly the same set of items S (thus $\forall a \in A : S = \{x \in D | x \sqsubset a\}$). $C = \bigcup_{a \in A} C(a)$ is the set of tags belonging to the aspects in A. Then the inner product applied to the binary collaborative feature space $\mathbf{X}_C$ (denoted by $sim_{\mathbf{X}_C}$) is identical to the sum of the tree depths of the most specific common tag node in each tag tree $a \in A$ for any pair of items $x, y \in S$.*

$$\forall x, y \in S : sim_{\mathbf{X}_C}(x, y) = \sum_{a \in A} sim_a(x, y)$$

where $sim_a(x, y) = |\{c \in C(a) \mid x \in ext(c) \wedge y \in ext(c)\}|$ (tree depth of the most specific common node).

Proof. The proof of this property works similar to the one for the Euclidean distance. We divide the tags in a tag tree into four sets. In the case of the inner product, only the set $C_{xy} \subseteq C(a)$ contributes to the similarity concerning a single tag structure $a \in A$. This is exactly the tree depth of the most specific tag that is assigned to both items x and y. Again, as features obtained from different tag structures are independent and the inner product is additive, the lemma above can be shown by induction. □

Using the inner product on a collaborative feature space leads to another sound similarity measure, which is a natural extension of tag based weighting method (based on tree depth) for individual tag structures.

Inner product has, however, the disadvantage of not being metric and thus being less efficient to be used for clustering and classification.

5.6.3 Hierarchical Cluster Ensembles

Applying the inner product and Euclidean distance to a collaborative feature space both lead to reasonable notions of a collaborative similarity measure. Both similarity measures can be used to create cluster ensembles, just as the co-occurrence measure is used to create ensemble partitions.

The application of top-down k-means seems especially attractive, as this algorithm is very efficient and produces good results in the similar context of text clustering [167]. Any other hierarchical clustering scheme that is based on the above similarity measures can be applied as well.

However, there is still one problem. Above, we assumed that all tag structures cover the same set of items. In practice, this is seldom true. There are two ways to deal with this problem. Either, it can be ignored, hoping that any imbalances cancel each other out.

The other possibility is to classify the missing items into all tag structures in order to ensure that all tag structures cover the same set of items. This approach is feasible, because we usually have a query set S of items to cluster. Only these items have to be inserted into all tag structures. Actually, this problem reflects an underlying problem of cluster ensembles as such, namely that they aim at a global consensus, such that each input clustering decides for all items how they are structured. For collaborative structuring, this is unnatural, as, for instance, some input clustering could be well suited to structure jazz music, while another one could be well suited for rock music. Applying the jazz structure to rock items and vice versa would not make much sense. Another disadvantage of cluster ensembles is that they destroy the underlying structure of the input tags including the labels associated with the tags. Also, if there are opposing opinions of how to structure a set of items, they are not returned both, but an average of them, which could be completely meaningless. An approach that preserves the structure of the underlying tags and may return different, conflicting tag structures is presented in the next chapter.

5.7 Evaluation

5.7.1 Overview

The aim of the evaluation is to validate a central hypothesis of this work: that exploiting user assigned tags and tag structures helps to increase the accuracy of clustering and classification algorithms in the context of information structuring. Based on the definitions and findings above, the following hypothesis can be formulated for the task of classification:

(C1) Using tags improves the average classification accuracy.

(C2) Weighting tags leads to superior accuracy than using binary values to represent tags.

(C3) Feature selection on tags improves the accuracy.

(C4) Tag aggregation improves the accuracy.

(C5) Supervised tag aggregation improves the accuracy even further.

For the task of clustering, the following hypothesis can be stated:

(C6) Cluster ensembles perform better than content-based clustering.

(C7) Unsupervised aggregation improves the accuracy of tag clustering.

To validate these hypothesis, we introduce a novel evaluation procedure called „leave-one-structure-out“. This strategy works similar to „cross-validation“ evaluation for classification (see chapter 1). Given a set of tag structures A, this set is split into a training set A_{train} and a test set A_{test}, with $A_{train} \cap A_{test} = \emptyset$. From the training set, features are

extracted, that are then used to enrich the feature space for classification or clustering. The latter leads to hierarchical cluster ensembles, as presented above.

In the given case, A_{test} is chosen such that it contains exactly one tag structure and A_{train} contains all remaining tag structures. Evaluation is the performed n-times, such that each tag structure in A is used exactly once for testing. The result for all test sets is then averaged to produce an estimation for the expected accuracy of the algorithm. The evaluation of a given test or training set is based on traditional methods of evaluating classification and clustering algorithms, as presented in chapter 1. As evaluation criteria for classification the expected accuracy, as measured by 10-fold cross-validation, is used. Clustering is evaluated by comparing the tag structure in the test set to the tag structure produced by the clustering algorithm using FScore and the correlation and distance of the tree distances.

5.7.2 The Datasets

The above procedure was applied to two datasets. The first one is the garageband dataset already used in chapter 4. Again, we use only the top-level of the tag structure to obtain a flat classification problem. For clustering, the full hierarchical tag structures were used. The second dataset was obtained from the Awake system (see chapter 7). In this case, users structured conference papers. The dataset contains 70 tag structures. For each of the underlying conference papers, textual features were extracted from the abstracts. These features are calculated using standard stopword filtering, a stemmer for English language texts and tf/idf weighting [154].

Beside these two datasets, that directly contain tag structures, a third data set was used. This dataset was derived from the Bibsonomy[11] system. Bibsonomy is a social bookmarking system for BibTEX entries and for web pages. Users can tag URLs and BibTEX keys with arbitrary tags and optionally with a textual description. For some of the BibTEX entries, an abstract exists. For the experiments in this chapter, each resource was represented by all tags that were assigned to it. Additionally, a word vector representation was calculated from the textual description associated with each resource. These latter features are again calculated using standard stopword filtering, a stemmer for English language texts and tf/idf weighting. As this dataset does not contain any tag structures in the sense of this work, the evaluation procedure slightly differs from the one used for the first two datasets. We first create three classification problems using tag pairs. These tag pairs are „software vs. education“ (bib1), „research vs. education“ (bib2) and „web vs. software“ (bib3). These pairs were chosen as the corresponding tags are among the most popular ones (and thus contain many examples). The tags „software“, „education“, „research“ and „web“ were removed from the datasets. The three classification problems are solved using a classification algorithm and evaluated by traditional cross-validation.

[11] http://www.bibsonomy.org

	avg. accuracy	
	garageband	awake
audio/text	0.45	0.64
tags binary	0.68	0.78
tag depth	0.67	0.78
tag inf.	0.68	0.80
tag/item depth	0.68	0.78
tag/item inf.	0.69	0.80
audio/text + tags	0.71	0.81

Table 5.1: Comparison of content and tag based classification for the garageband and the awake datasets. The numbers show the average accuracy over all tasks in each dataset.

5.7.3 Collaborative Classification

For the evaluation of collaborative classification, three basic approaches were compared: classifying items using content related features, using tags and using both, tags and content features. Additionally, different weighting schemes for extracting features from tags were used. „tag depth“ and „tag inf.“ denote tag based weighting with tree depth and information content respectively. „tag/item depth“ and „tag/item inf.“ denote tag/item based weighting with tree depth and information content respectively. For „audio/text + tags“ content-based features and tags were used. Tags are weighted by tag/item weighting using information content in this case. As learning algorithm for the awake and the garageband dataset, nearest neighbor was used, for the bibsonomy dataset the support vector machine [85] was used. The result is presented in tables 5.1 and 5.2.

We can see the following. Using tags clearly improves the result over using content related features only. This improvement is stronger for the garageband dataset, that is based on audio features, than for the other two datasets that are based on textual features. This confirms hypothesis (C1). The way tags are weighted has some influence on the result. This influence is not very strong, however. Information based weighting is slightly superior to depth based weighting and tag/item weighting is slightly superior to tag weighting (hypothesis (C2)).

In a second set of experiments, the influence of tag selection and tag aggregation was evaluated. Tag selection is performed using traditional forward selection on a mixed collaborative feature space. While PFS could have been used as well here, the focus of this chapter is on the role of tags in collaborative classification. Feature aggregation is performed unsupervised and supervised, as described above. Finally, a decision tree learner [147] is used, which implicitly performs feature selection and some kind of aggregation, as argued in chapter 1. The results are depicted in table 5.3. Feature selection clearly improves the result, as expected (hypothesis(C3)). Unsupervised tag aggregation does improve the result only for garageband. For the awake dataset, accuracy decreases. Thus hypothesis (C4) is not fully supported. This approach remains valid as a heuristic that

	avg. accuracy		
	bib1	bib2	bib3
text	0.87	0.73	0.75
tags binary	0.92	0.82	0.84
tag depth	0.92	0.82	0.84
tag inf.	0.93	0.85	0.84
tag/item depth	0.92	0.82	0.84
tag/items inf.	0.93	0.85	0.84
text + tags	0.93	0.79	0.81

Table 5.2: Comparison of content and tag based classification for the three tasks in the bibsonomy dataset. The numbers show the average accuracy over all tasks in a dataset.

	avg. accuracy	
	garageband	awake
forward tag selection	0.84	0.90
unsupervised tag aggr.	0.69	0.70
supervised tag aggr.	0.85	0.92
C4.5	0.80	0.74

Table 5.3: Influence of tag selection and aggregation on the accuracy of collaborative classification

has to be evaluated on new datasets. Supervised tag aggregation improves the result, is, however, computationally extremely expensive (hypothesis (C5)). Decision trees work well on the garageband dataset. On the awake dataset they produce inferior results.

5.7.4 Collaborative Clustering

To evaluate the hypothesis on collaborative clustering, we use two algorithms, namely top-down k-means and bottom-up agglomerative clustering, both in conjunction with Euclidean distance. For top-down clustering, the number of child nodes was set to five for both datasets. The maximal number of items in a leaf node was set to 10 for garageband and to 5 for the awake dataset.

These algorithms are applied to three different features spaces: one consisting of content related features only, one derived from tags only (hierarchical cluster ensembles) and one with a mixture of both. Tags are weighted always binary here, as suggested by the theoretical findings above (see lemma 5.6.2). The accuracy is measured by comparing the tag structure in the test set with the one produced by the algorithm in terms of symmetric FScore and the correlation and absolute distance of tree distances (as presented in section 1.4.5).

Tables 5.4 and 5.4 show the result. As can be seen, collaborative features (thus hierar-

	Correlation	Absolute distance	FScore
TD audio	0.13	1.9	0.40
TD ensemble	0.22	1.7	0.54
TD ensemble (aggr.)	0.23	1.9	0.55
TD mixed	0.15	1.9	0.41
single-link audio	0.065	28	0.35
single-link ensemble	0.11	25	0.39
single-link ens. (aggr.)	0.12	28	0.39
single-link mixed	0.086	30	0.35

Table 5.4: Comparison for the garageband of top-down (TD) and bottom-up (single-link) clustering based on audio features (audio), based on tags (hierarchical ensemble) and based on both (mixture). The table also shows the influence of using aggregation on hierarchical cluster ensembles (aggr.).

	Correlation	Absolute distance	FScore
TD text	0.41	0.65	0.67
TD ensemble	0.55	0.62	0.71
TD ensemble (aggr.)	0.56	0.58	0.73
TD mixed	0.57	0.54	0.72
single-link text	0.46	3.9	0.79
single-link ensemble	0.60	3.9	0.81
single-link ensemble (aggr.)	0.59	3.9	0.81
single-link mixed	0.60	3.9	0.82

Table 5.5: Comparison for the awake dataset of top-down (TD) and bottom-up (single-link) clustering based on text features (text), based on tags (hierarchical ensemble) and based on both (mixture). The table also shows the influence of using aggregation on hierarchical cluster ensembles (aggr.).

chical cluster ensembles) perform mostly better than clustering based on content-related features only, which confirms hypothesis (C5). Tag aggregation does improve the result for garageband, but not for the awake dataset. For the text based awake dataset, a mixed collaborative feature space outperforms both, content based and collaborative features. For the garageband dataset this is not the case. Using audio features in addition to collaborative features decreases the accuracy.

5.8 Conclusion

Current tagging system, such as last.fm or flickr, are very promising in enriching web resources with semantic information. In contrast to more complex approaches, such as the semantic web, they are widely used and accepted. The major reason for this popularity is the loosely coupled way in which items can be tagged. However, as the number of items and tags grows, they become hard to manage. The user should therefore be assisted in the task of organizing tags.

This chapter gave a survey of current social bookmarking systems and their relation to the semantic web. It was argued why social bookmarking is very well-suited for collaborative media organization and in which way current social bookmarking systems are still limited. Based on this analysis, an extended formalism for tagging was given, called aspect-based tagging. This formalism allows users to create hierarchical tags and to group tags into aspects.

In a second step, it was shown that social bookmarking and annotations can be seen as a very powerful kind of feature extraction for items. Based on these collaborative features, methods for collaborative classification and collaborative clustering were formulated. Collaborative classification allows users to automatically annotate new items with predefined tags. Users may only tag a small amount of their items and the system then tags the remaining items automatically. Exploiting tags of other users is essential in this process, as tags often contain highly personal views, that are not reflected in the content or the meta data of these items. Collaborative clustering allows to assign tags if the user did not assign any tags yet. In this case, the tags that were assigned by other users can be used to structure these items. It was shown that applying traditional clustering algorithms to a collaborative feature space leads to cluster ensembles that merge all tag information into an average cluster structure.

Both collaborative clustering and classification were empirically compared to their traditional counterparts on several datasets. We applied a novel evaluation procedure, called leave-one-structure-out. In this procedure, we temporarily delete a tag structure from a set of test structures and try to reconstruct it, using the remaining tag structures. This is done for each tag structure in the test set and the result is the average over all individual results. It was shown that exploiting tag structures of other users leads to a superior result for both tasks, as compared to using content-based features only.

Exploiting tags created by other users through collaborative feature spaces has the advantage that all methods for distributed feature extraction can be directly applied to collaborative structuring. This makes collaborative structuring applicable in any kind of network.

While collaborative clustering yields good results, the output is not fully satisfying from a user's perspective. Especially, the fact that the resulting clustering can be hard to interpret and non-concise is a major problem. Also, from a point of view of aspect based tagging, it would be desirable to output more than one solution. In the next chapter, we will give an alternative problem definition for collaborative clustering that regards these constraints and propose a clustering algorithm that solves this problem.

6 Localized Alternative Cluster Ensembles

6.1 Introduction

The last chapter discussed the use of features gathered from user annotations for classification and clustering. It was argued why applying traditional clustering methods to such a collaborative feature space would lead to results that can be interpreted as hierarchical cluster ensembles. This approach was shown to be superior to clustering items based on content features only, in that it reflects the view of the users on a given domain.

Still, this approach shows several weaknesses, that correspond to the definition of collaborative clustering given in chapter 5:

1. Only a single cluster model is returned. Following the paradigm of aspect-based tagging, the algorithm should return several, alternative solutions, from which the user can pick one or several that best suit her needs and preferences.
2. User-created structures are not preserved. A major issue with automatically created cluster structures is that they are hard to interpret (the clusters have no labels) and can be non-concise (merging several heterogeneous cluster structures that are concise in themselves may lead to an average non-concise cluster structure)
3. The features space used is not adapted to items that are clustered. In cluster ensembles, it is expected that all input clusterings and the output clustering share the same set of items. This is, however, almost never true for collaborative media organization, because personal item collections may have very different extensions.
4. The algorithm is not directly applicable if some of the items were not yet tagged. Cluster ensembles do not allow to cluster items that are not yet tagged by anybody.

In the following, we will convert this into a set of conditions that a clustering scheme should fulfill to be applicable to collaborative structuring.

6.2 Problem Definition

Reflecting the problem identified above, we give the following extended definition of the collaborative clustering task.

Definition 6.2.1. The task of *collaborative clustering* is, given a set of aspects A (with associated aspect functions $\varphi_a : S_a \rightarrow 2^{C(a)}$ for each $a \in A$) and a set of items to cluster S, to produce a set of aspect functions $O \subseteq \{\varphi_{a'} : S_{a'} \rightarrow 2^{C(a')}\}$, with $S \subseteq S_{a'}$. Thus, the aim is to deliver several functions that are at least defined on the set of items that should be clustered.

This definition states that a set of tag structures (aspects) A are used as input and a set of hierarchical classification functions O is returned that are defined at least on the items to cluster S. Note that the input clusterings are not required to be defined on S.

An important constraint from a user's point of view is the preservation of structure. By merging several heterogeneous cluster or tag structures the outcome maybe a mix of different tag structures that does not make up a sound clustering. We therefore require that the inner structure beneath any tag that is used in the output is preserved.

In the following, we will review several existing approached concerning their ability to yield clusterings that fulfill these conditions.

6.3 Applicability of Existing Clustering Approaches

6.3.1 Cluster Ensembles and Constrained Clustering

In section 1.4.6, the problem of merging partitions was presented. This problem was generalized to the problem of merging hierarchical cluster structures in section 5.6.

Cluster ensembles do not comply with the problem definition above for three reasons. First, they yield only one solution, while we require the clustering algorithm to produce several alternative solutions. Second, they require that all input clusterings are defined on at least the set of items S to be clustered. Such input functions do possibly not exist. Third, they do not necessarily preserve the structure of the input clusterings.

For distributed clustering approaches (see section 3.2) the same holds because they deliver a hierarchical cluster ensemble just only in a distributed way.

While this was not discussed in detail so far, the same holds for constrained clustering. Instead of using the input clusterings in an ensemble like way, they could be used to constraint the output clustering. Constraints usually include must-link and cannot-link constraints. We could now define must-link constraints for all pairs of items that have at least one tag in common and cannot-link constraints for all items that do not have any tag in common.

While this would be an interesting variant to cluster ensembles, it has the same weaknesses concerning the conditions stated above: it yields a single, global cluster structure, that is not guaranteed to preserve the structure of the input clusterings. We will therefore not discuss this possibility further here.

6.3.2 Subspace Clustering

As seen in section 1.4.8, subspace clustering applied to binary features is equivalent to frequent itemset mining. A first reason why this does not fulfill the conditions above is that the algorithm returns individual clusters and not classification functions. The lattice formed by the frequent itemsets could be seen as a result clustering. This result clustering would, however, not fulfill the above conditions. First, it is a DAG and not a classification function (which is technically a tree). Second, it mixes concepts of different aspects into one structure, violating the structure preservation condition.

We could extend this approach by constraining the subspace clusters in a way that does not allow to mix concepts of different aspects. This would deliver several lattices, one for each aspect. These lattices are, however, DAG and do not fulfill the requirement of a classification function. Also, it would be trivial, as instead of using the lattices corresponding to each input aspect, we could directly use the classification function associated with each aspect, not performing subspace clustering at all. In this case, the number of solutions would be much to high.

6.3.3 Non-redundant Clustering

Non redundant clustering is a good candidate to produce alternative clustering solutions. The idea is to start by using a traditional clustering algorithm, say k-means based on collaborative features. This delivers a first result. Then, we may invoke this algorithm a second time, this time searching for solutions that are maximally different from the first result.

While the original algorithm described in [65] supports only one reference clustering, the algorithm could be extended to find clusterings that are maximally different to two or more reference clusterings. In this way, the algorithm could be invoked iteratively delivering a sequence of non-redundant solutions.

Unfortunately, non-redundant clustering does not deliver hierarchical solutions. Just as cluster ensembles, it does not guarantee to preserve the structure of the input clusterings.

6.3.4 Ontology Learning

The assignment of items to concepts by the extension function ext can be seen as entries in an DL ABox. Therefore, it would be possible to apply ontology learning to derive a TBox for these items.

In the following, we assume the KLUSTER algorithm. As we do not have any roles in our application, KLUSTER is mostly reduced to a substep, the Sort Taxonomy Tool (STT). KLUSTER would first collect all possible concept extensions into a lattice C_{all} by set inclusion. From this lattice C_{all}, we can remove all concepts $c \in C_{all} : ext(c) \cap S = \emptyset$, thus all concepts that are disjoint with the items to cluster, as these clusters would not

contribute to a final classification function anyway. The resulting structure C_S is not guaranteed to be a tree, however.

The next step would be to find MDCs, thus partitions of the items in S on the top level. Let us assume that in addition to what KLUSTER does, we restrict the MDCs such that they must be disjoint on S. Thus an MDC $C \subseteq C_S$ must fulfill the additional condition,

$$\forall c, c' \in C : ext(c) \cap ext(c') \cap S = \emptyset$$

KLUSTER does not enforce this condition, it is needed, however, to generate clusterings that are disjoint at each level.

For each of these MDCs, we could now learn a classification function. There are no roles or relations, in our case. We can, however, apply an attribute based learner to obtain discriminative descriptions. This learner can now be applied to classify the items in S, not yet covered by top level MDCs, into the result structure. Then, each MDC forms a classification function on at least the items in S.

These MDCs can be returned as result. There are, however, still some problems.

The concepts of different user created aspect could be mixed up if their extensions are identical. This could be easily fixed, however, by introducing additional constraints.

A second problem is that the number of MDCs could exceed the number of desired results. In this case, a ranking on MDCs would be desirable, such that the k best ones could be returned only.

A third problem is that the algorithm is not applicable in cases in which all concepts are disjoint with S. This could happen if the input space is very sparse.

In the next section, we present the LACE algorithm that resolve these issues.

6.4 Localized, Alternative Cluster Ensembles

In the following, we describe a method for collaborative clustering that is based on the idea of bags of clusterings. The idea is to derive a new tag structure from existing ones by extending existing tag structures and combining them such that each of them covers a subset of the items to be clustered. As this method is applicable in a more general context as well, we will refer to input and output clusterings and functions instead of tag structures. In section 6.4.4, we explain why this method is especially well suited for collaborative structuring.

6.4.1 Bags of clusterings

We define the extension of functions φ_i.

Definition 6.4.1. Given a function $\varphi_i : S_i \rightarrow G_i$, the function $\varphi_i' : S_i' \rightarrow G_i$ is an *extended function* for φ_i, if $S_i \subset S_i'$ and $\forall x \in S_i : \varphi_i(x) = \varphi_i'(x)$.

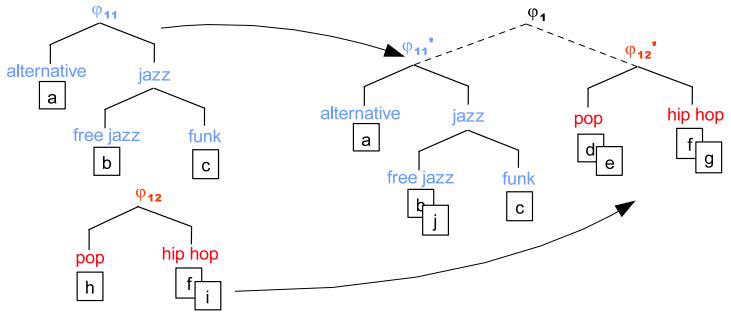


Figure 6.4.1: A bag of clusterings composed of two input clusterings

Extended functions allow us to define a bag of extensions of non-overlapping originally labeled subsets that covers the entire collection:

Definition 6.4.2. Given a set I of functions. A *bag of clusterings* is a function

$$\varphi_i(x) = \begin{cases} \varphi'_{i1}(x), & \text{if } x \in S'_{i1} \\ \vdots & \vdots \\ \varphi'_{ij}(x), & \text{if } x \in S'_{ij} \\ \vdots & \vdots \\ \varphi'_{im}(x), & \text{if } x \in S'_{im} \end{cases}$$

where each φ'_{ij} is an extension of a $\varphi_{ij} \in I$ with $\varphi_{ij} : S_{ij} \to G_{ij}$ and $\{S'_{i1}, \ldots, S'_{im}\}$ is partitioning S with $S_{ij} \subset S'_{ij}$.

Figure 6.4.1 shows an example.

Now, we can define the objective function for our bag of clusterings approach to local alternative clustering ensembles.

Definition 6.4.3. The *quality of an individual output function* φ_i, with respect to a set of input functions I and a set of items S to be clustered, is measured as

$$q^*(I, \varphi_i, S) = \sum_{x \in S} \max_{x' \in S_{ij}} sim(x, x') \text{ with } j = h_i(x)$$

where sim is a similarity function $sim : D^2 \to \mathbb{R}_0^+$ and h_i assigns each example to the corresponding function in the bag of clusters $h_i : S \to \{1, \ldots, m\}$ with

$$h_i(x) = j \Leftrightarrow x \in S'_{ij}.$$

The *quality of a set of output functions* now becomes

$$q(I, O, S) = \sum_{\varphi_i \in O} q^*(I, \varphi_i, S).$$

Input:
A set of items S
A set of input functions I

Output:
A set of output functions O

$O = \emptyset$;
$I' = I$;
while $(I' \neq \emptyset) \wedge (|O| < max_{alt})$ **do**
  $S' = S$;
  $B = \emptyset$;
  $step = 0$;
  while $((S' \neq \emptyset) \wedge (I' \neq \emptyset) \wedge (step < max_{steps}))$ **do**
    $\varphi_i = \arg\max_{\varphi \in I'} q_f^* (Z_\varphi, S')$;
    $I' = I' \setminus \{\varphi_i\}$;
    $B = B \cup \{\varphi_i\}$;
    $S' = S' \setminus \{x \in S' | x \sqsubset_\alpha \varphi_i\}$;
    $step = step + 1$;
  end while
  $O = O \cup \{bag(B, S)\}$;
end while

Figure 6.4.2: The sequential covering algorithm finds bag of clusterings in a greedy manner. max_{alt} denotes the maximum number of alternatives in the output, max_{steps} denotes the maximum number of steps that are performed during sequential covering. The function *bag* constructs a bag of clusterings by assigning each item $x \in S$ to the function $\varphi_i \in B$ that contains the item most similar to x.

Besides optimizing this quality function, we want to cover the set S with a bag of clusterings that contains as few clusterings as possible.

Note the connection to the KLUSTER algorithm. A bag of clusterings serves a similar aim as a MDC. It is, however, not required that the individual members of a bag of clusterings do not overlap. Also, it is not assumed that there is a perfect extensional match, but rather a similarity-based match is sufficient. Finally, bags of clusterings are rated by a quality function that allows to rank them, such that only the top k results are returned.

6.4.2 The LACE Algorithm

In the following, we present a greedy approach to optimizing the bag of clusterings problem. The main task is to cover S by a bag of clusterings φ. The basic idea of this

approach is to employ a sequential covering strategy. In a first step, we search for a function φ_i in I that best fits the set of query items S. For all items not sufficiently covered by φ_i, we search for another function in I that fits the remaining points. This process continues until either all items are sufficiently covered, a maximal number of steps is reached, or there are no input functions left covering the remaining items. All data points that could not be covered are assigned to the input function φ_j containing the item which is closest to the one to be covered. Alternative clusterings are produced by performing this procedure several times using each input function at most once.

We now have to formalize the notion of a function sufficiently covering an item and a function fitting a set of items such that the quality function is optimized. When is a data point sufficiently covered by an input function so that it can be removed from the query set S? We define a threshold based criterion for this purpose.

Definition 6.4.4. A function φ sufficiently covers an item $x \in S$ (written as $x \sqsubset_\alpha \varphi$), iff $\max_{x' \in Z_\varphi} sim(x, x') \geq \alpha$, where $Z_\varphi \subseteq D$ is a set of points that represent function φ (see below).

The set Z_φ of items represents φ. This concept will be discussed below. The threshold α allows us to balance the quality of the resulting clustering and the number of input clusters. A small value of α allows a single input function to cover many items in S. This, on average, reduces the number of input functions needed to cover the whole query set. However, it may also reduce the quality of the result, as the algorithm covers many items in a greedy manner, which could be covered better using an additional input function.

Turning it the other way around: when do we consider an input function to fit the items in S well? First, it must contain at least one similar item for each item in S. This is essentially what is stated in the quality function q^*. Second, it should cover as few additional items as possible. This condition follows from the locality demand. Using only the first condition, the algorithm would not distinguish between input functions which span a large part of the data space and those which only span a small local part. This distinction, however, is essential for treating local patterns in the data appropriately. The situation we are facing is similar to that in information retrieval. The target concept S, the ideal response, is approximated by φ delivering a set of items, the retrieval result. If all members of the target concept are covered, the retrieval result has the highest recall. If no items in the retrieval result are not members of S, it has the highest precision. We want to apply precision and recall to characterize how well φ covers S. We can define

$$prec(Z_{\varphi_i}, S) = \frac{1}{|Z_{\varphi_i}|} \sum_{z \in Z_{\varphi_i}} max \{sim(x, z) | x \in S\}$$

and

$$rec(Z_{\varphi_i}, S) = \frac{1}{|S|} \sum_{x \in S} max \{sim(x, z) | z \in Z_{\varphi_i}\}.$$

Please note that using a similarity function which maps identical items to 1 (and 0 otherwise) leads to the usual definition of precision and recall. The fit between an input function and a set of items now becomes a continuous f-measure:

$$q_f^*(Z_{\varphi_i}, S) = \frac{(\beta^2+1)rec(Z_{\varphi_i}, S)prec(Z_{\varphi_i}, S)}{\beta^2 rec(Z_{\varphi_i}, S) + prec(Z_{\varphi_i}, S)}$$

Recall directly optimizes the quality function q^*, precision ensures that the result captures local structures adequately. The fitness $q_f^*(Z_{\varphi_i}, S)$ balances the two criteria using a parameter $\beta \in \mathbb{R}_0^+$. A smaller value of β gives a stronger weight on the recall, a higher value gives a stronger weight on the precision. Note that for $\beta = 1$ we receive the original f-measure.

Deciding whether φ_i fits S or whether an item $x \in S$ is sufficiently covered requires to compute the similarity between an item and a cluster. If the cluster is represented by all of its items ($Z_{\varphi_i} = S_i$, as usual in single-link agglomerative clustering), this central step becomes inefficient. If the cluster is represented by exactly one point ($|Z_{\varphi_i}| = 1$, a centroid in k-medoids clustering), the similarity calculation is very efficient, but sets of items with irregular shape, for instance, cannot be captured adequately. Hence, we adopt the representation by „well scattered points“ Z_{φ_i} as representation of φ_i [66], where $1 < |Z_{\varphi_i}| < |S_i|$. These points are selected by stratified sampling according to G.

We can compute the fitness q_f^* of all $\varphi_i \in I$ with respect to a query set S in order to select the best φ_i for our bag of clusterings. The whole algorithm works as depicted in figure 6.4.2. We start with the initial set of input functions I and the set S of items to be clustered. In a first step, we select an input function that maximizes $q_f^*(Z_{\varphi_i}, S)$. φ_i is removed from the set of input functions leading to a set I'. For all items S' that are not sufficiently covered by φ_i, we select a function from I' with maximal fit to S'. This process is iterated until either all items are sufficiently covered, a maximal number of steps is reached, or there are no input functions left that could cover the remaining items. All input functions selected in this process are combined to a bag of clusters, as described above. Each item $x \in S$ is assigned to the input function containing the item being most similar to x. Then, all input functions are extended accordingly, again by nearest-neighbor classification (cf. definition 6.4.1). We start this process anew with the complete set S and the reduced set I' of input functions until the maximal number of alternatives is reached.

Lemma 6.4.5. *The number of similarity calculations necessary to perform LACE is in* $O(|I||S||Z_{\varphi_i}|)$.

Proof. Each function is represented by a fixed number of representative points Z_{φ_i}. For each input function, the algorithm must compare all items in S with the representative points of all input function in I. □

6.4.3 Hierarchical Matching

A severe limitation of the algorithm described so far is that it can only combine complete input clusterings. In many situations, a combination of partial clusterings or even individual clusters would yield a much better result. This is especially true if local patterns are to be preserved being captured by maximally specific concepts. Moreover, the algorithm does not yet handle hierarchies. Our motivation for this research was the structuring of media collections. Flat structures are not sufficient with respect to this goal. We cannot use a standard hierarchical clustering algorithm since we still want to solve the new task of local alternative cluster ensembles. In the following, we extend our approach to the combination of partial hierarchical functions.

It should be possible to match functions that correspond to only a partial group hierarchy. We formalize this notion by defining a hierarchy on functions that extends the set of input functions such that it contains all partial functions.

Definition 6.4.6. Two hierarchical functions $\varphi_i : S_i \rightarrow 2^{G_i}$ and $\varphi_j : S_j \rightarrow 2^{G_j}$ are in direct *sub function relation* $\varphi_i \prec \varphi_j$, iff $G_i \subset G_j$, $\forall x \in S_i : \varphi_i(x) = \varphi_j(x) \cap G_i$, and $\neg\exists \varphi_i' : G_i \subset G_i' \subset G_j$.

Let the set I^* be the set of all functions which can be achieved following the direct sub function relation starting from I and the function in I itself, thus

$$I^* = \{\varphi_i | \exists \varphi_j \in I : \varphi_i \prec^* \varphi_j\} \cup I$$

where $\prec^*$ is the transitive hull of $\prec$. While it would be possible to apply the same algorithm as above to the extended set of input functions I^*, this would be rather inefficient, because the size of I^* can be considerably larger than the one of the original set of input functions I. We therefore propose an algorithm which exploits the function hierarchy and avoids multiple similarity computations. Each function $\varphi_i \in I^*$ is again associated with a set of representative items Z_{φ_i}. We additionally assume the standard taxonomy semantics:

$$\varphi_i \prec \varphi_j \Rightarrow Z_{\varphi_i} \subseteq Z_{\varphi_j}$$

Now, the precision can be calculated recursively in the following way:

$$prec(Z_{\varphi_i}, S) = \frac{|Z^*_{\varphi_i}|}{|Z_{\varphi_i}|} prec(Z^*_{\varphi_i}, S) + \sum_{\varphi_j \prec \varphi_i} \frac{|Z_{\varphi_j}|}{|Z_{\varphi_i}|} prec(Z_{\varphi_j}, S)$$

where $Z^*_{\varphi_i} = Z_{\varphi_i} \setminus \bigcup_{\varphi_j \prec \varphi_i} Z_{\varphi_j}$. For recall a similar function can be derived. Note that neither the number of similarity calculations is greater than in the base version of the algorithm nor are the memory requirements increased.

Moreover, the bottom-up procedure also allows for pruning. We can optimistically estimate the best precision and recall that can be achieved in a function hierarchy using all representative items Z_e for which the precision is already known. The following holds:

$$prec(Z_{\varphi_i}, S) \leq \frac{|Z_e| \cdot prec(Z_e, S) + |Z_{\varphi_i} \setminus Z_e|}{|Z_{\varphi_i}|}$$

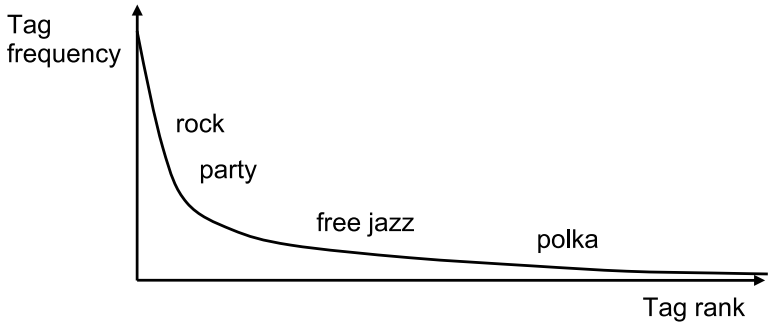


Figure 6.4.3: Long tail in the tag distribution

with $Z_e \subset Z_{\varphi_i}$. An optimistic estimate for the recall is one. If the optimistic f-measure estimate of the hierarchy's root node is worse than the current best score, this hierarchy does not need to be processed further. This is due to the optimistic score increasing with $|Z_{\varphi_i}|$ and $|Z_{\varphi_i}| > |Z_{\varphi_j}|$ for all sub functions $\varphi_j \prec \varphi_i$. No sub-function of the root can be better than the current best score if the score of the root is equal or worse than the current best score.

6.4.4 Using LACE for Collaborative Clustering

The LACE approach is very well suited for collaborative structuring for several reasons. First, it leads to a comprehensible, sound result, as the structure and labels of the input tags are preserved. Second, it accommodates to the local properties of the item space, returning only such tags that are relevant to the items that should be clustered. Third, it allows new structures to emerge from existing ones, which was one of the aims of collaborative structuring and corresponds to the mash-up paradigm of the web 2.0. Furthermore, it allows for evolving tag spaces. Users copy tag structures they like, which are then more likely recommended. Still, as several solutions are returned, this does not lead to only one dominant tag structure but to a set of several popular tag structures for a given set of items.

The claim is that by recommending tag structures we achieve both, accuracy (in the sense of performance as defined in section 5.7) and diversity. Both are essential properties for a collaborative structuring algorithm. To show this relationship, we assume a fixed set S for which the users try to find an optimal tag structure. We further assume that the set S can be covered by a single input cluster. In principle, each user could prefer an own structure on S that is completely uncorrelated to the structures all other users would choose. In this case, the approach would not work. In reality, this does not happen. On contrary, structures and tags are all but equally distributed. Figure 6.4.3 shows a typical distribution of tags in a tagging system. As can be seen, there are few tags preferred by many users and many tags that are somewhat preferred by only a small number of users. How does this distribution affect the accuracy of the approach? As tag structures

are more likely recommended if many users share them, for the large majority of tag structures, LACE will deliver a perfect result. Only for the content of the long tail, this result maybe inferior. As, however, the expected error is weighted by the number of users that share a tag structure, this long tail will affect the result to a much lower extent. Returning several solutions still enables several structures to co-exist and users can freely choose among them. This was one of the most important properties that we derived for a desirable representation and recommendation mechanism for structures in section 5.2.

Please note that the above scenario assumes a fixed set of items. Actually, we face two long tail distributions, one concerning items and the second concerning tags (given a fixed set of items). The locality of LACE allows to adequately cluster items even if they are not globally popular (thus even if they are part of the long tail). The long tail of tag structures can be supported by increasing the number of alternative solutions that are returned, at the price of an increased effort for the user to evaluate these alternatives.

6.5 Empirical Evaluation

Beside this qualitative evaluation of the approach, an empirical evaluation was performed on the two real world datasets garageband and awake that were already described in the last chapters.

6.5.1 Evaluation Criteria

The evaluation criterion used for simple collaborative clustering must be extended to reflect the fact that several solutions are returned. As described above, the idea of returning more than one result is that the user may choose the clustering that most suits her needs and preferences. Therefore, a natural choice is to assume that the quality of a set of solutions O is the quality of the best solution in O.

Definition 6.5.1. The *quality of a set of clustering solutions* O is the maximum quality of any member of O, thus

$$q^*(\varphi_a, O) = max_{\varphi_{a'} \in O}\{sim(\varphi_a, \varphi_{a'})\}$$

where sim is any measure to compare two clusterings (see section 1.4.5). For the absolute distance of tree distances, this maximum is replaced by the minimum, as in this case the distance between two cluster models should be minimized.

The overall performance is the average performance over all tag structures $a \in A$.

The number of clusterings returned should be, however, small, such that the user can easily evaluate all of them. In the remainder of this work, a maximum of $|O| \leq 5$ result structures is used, if not stated differently.

	Correlation	Absolute distance	FScore
LACE	0.45	0.65	0.74
TD audio	0.13	1.9	0.40
TD ensemble	0.22	1.7	0.54
single-link audio	0.065	28	0.35
single-link ensemble	0.11	25	0.39

Table 6.1: Results of LACE on the garageband dataset

6.5.2 Empirical Evaluation

We use two datasets for the evaluation of LACE: the garageband dataset and the awake dataset. Both datasets have already been described in the evaluation sections of the last chapters. To guide the evaluation, we formulate some hypothesis:

(L1) LACE yields an accuracy that is better or at least comparable to the one produced by traditional, content-based clustering algorithms.

(L2) LACE yields an accuracy that is better or at least comparable to the one produced by hierarchical cluster ensembles.

(L3) The number of representative points has an influence on the accuracy, which is, however, not severe.

(L4) The number of results returned by LACE has an influence on the accuracy.

In the following, LACE is compared with single-link agglomerative clustering using Euclidean distance, top down divisive clustering based on recursively applying k-means (TD). The results for these clustering algorithms were already reported in section 5.7 and are here only shown to ease the comparison. LACE was applied using Euclidean distance as inner similarity measure. The LACE parameter β was set to 1.

The overall evaluation procedure is the same as in chapter 5. We use all but one tag structure for training LACE (or for creating a hierarchical cluster ensemble). Then the items covered by the remaining tag structure are clustered using LACE. The result is compared to the tag structure that was left out using symmetric FScore and the correlation and the absolute distance of tree distances. As LACE delivers several results, the best of these results is used. The rationale behind this procedure is that a user would be able to choose the „best“ among five results manually without major problems.

Table 6.1 and table 6.2 show the results for the garageband and for the awake dataset. As can be seen, LACE performs best in all cases.

A second experiment inspects the influence of the representation on the accuracy. The results of LACE with different numbers of instances at a node are shown in tables 6.3 and 6.4. Representing functions by all points performs best. Well scattered points perform well. We obtain good results even for a very small number of representative items at each node of the cluster model.

	Correlation	Absolute distance	FScore
LACE	0.75	0.26	0.89
TD text	0.41	0.65	0.67
TD ensemble	0.55	0.62	0.71
single-link text	0.46	3.9	0.79
single-link ensemble	0.60	3.9	0.81

Table 6.2: Results of LACE on the awake dataset

Representation	Correlation	Absolute distance	FScore
all points	0.45	0.65	0.74
$\|Z\| = 10$	0.41	0.70	0.74
$\|Z\| = 5$	0.39	0.71	0.74
$\|Z\| = 2$	0.39	0.72	0.73

Table 6.3: Influence of the concept representation in LACE for the garageband dataset.

Representation	Correlation	Absolute distance	FScore
all points	0.75	0.26	0.89
$\|Z\| = 10$	0.75	0.26	0.89
$\|Z\| = 5$	0.74	0.28	0.89
$\|Z\| = 2$	0.73	0.28	0.89

Table 6.4: Influence of the concept representation in LACE for the awake dataset.

Alternatives	Correlation	Absolute distance	FScore
5	0.45	0.65	0.74
3	0.40	0.70	0.73
1	0.33	0.80	0.68

Table 6.5: The influence of response set cardinality in LACE for the garageband dataset

Alternatives	Correlation	Absolute distance	FScore
5	0.75	0.26	0.89
3	0.70	0.30	0.89
1	0.61	0.42	0.86

Table 6.6: The influence of response set cardinality in LACE for the awake dataset

We also evaluated how the number of output functions influences the quality of the result. The result should be clearly inferior with a decreasing number. Tables 6.5 and 6.6 show the results. On one hand, we observe that even with just one model, i.e. $|O| = 1$, LACE still mostly outperforms the other methods. On the other hand, the results are, indeed, getting worse with less alternatives. Providing alternative solutions seems to be essential for improving the quality of results, at least in heterogeneous settings as the one discussed here. Probably, the performance would increase even further for more output clusterings. Although a user still would select the best available clustering from all alternatives, which motivates this form of evaluation, the number of solutions should be rather small and was restricted to 5 in this setting.

6.6 Implementing LACE in a Peer-to-Peer Environment

In the last section, we showed that collaborative classification and hierarchical cluster ensembles can easily be combined with distributed feature extraction, as both assemble a set of features from user annotations. This makes both approach directly applicable in a peer-to-peer setting, using the methods proposed in chapter 4.

A crucial question is whether this is true for the LACE algorithm as well. In the following we show that this is the case. The key to this problem is the fact that the representation functions for user annotations proposed in section 5.4 are bijective. Thus given a set of collaborative features $\mathbf{X}_{\mathrm{C}}$, we can retransform these features into a set of concepts C. We assume that the aspect a concept belongs to is encoded in the feature name. As concepts cannot be shared among aspects, it is always possible to name features in such a way. This reduces the problem to recovering a tag structure from features belonging to exactly one aspect. It was shown in section 5.4 that this is always possible, given the restriction of tag structured introduced in section 5.3.1.

In this sense, LACE can be seen as a feature selection algorithm, selecting all the features that represent the concepts that make up the bags of clusterings returned as a result.

6.7 Conclusion

Cluster ensembles show several weaknesses when it comes to clustering items based on existing tag structures. In this chapter, these weaknesses were analyzed. First, cluster ensembles are not able to produce more than one solution. This, however, is essential, as the same domain can usually be described from different perspectives. Second, they do not preserve the structures created by other users. This often leads to average clusterings, that are not concise and hard to comprehend.

Based on this analysis, a new definition of the collaborative clustering task was given. In this definition, a set of given tag structures and a set of items to cluster is taken as input, and a set of proposed tag structures on the items in question is delivered as output. This learning task was defined in a formal way and it was shown that existing clustering approaches are not able to solve this task adequately. Then, a greedy algorithm was developed, called Localized Alternative Cluster Ensembles (LACE). LACE combines given tag structures without altering their inner structure. Several enhancements for hierarchical structures were proposed as optimization of LACE. The algorithm was analyzed and it was shown that it works linear in the number of items and given tag structures. Also, it was shown that LACE can be interpreted as a special form of feature selection from $\mathbf{X}_{\mathrm{C}}$. This allows us to apply the algorithm in the general p2p feature extraction framework proposed in chapter 4.

The accuracy of algorithms solving the collaborative clustering problem can be measured by a leave-one-structure-out approach, as proposed in the last chapter. The LACE algorithm outperforms standard clustering schemes on two real-world datasets. We also investigated the influence of the number of representative points and the influence of response set cardinality on the result.

Collaborative classification, as proposed in the last chapter, and localized alternative cluster ensembles complement each other. They provide two powerful tools to enable users to fully profit from what other users have already done. Integrated into the framework of distributed feature extraction, they allow to perform collaborative structuring in an ubiquitous way.

Part III

Application and Integration

7 Supporting Heterogeneous Expert Communities

7.1 Introduction

The widespread use of the Internet enables radically new forms of collaboration. Maybe the most prominent ones concern managing and sharing knowledge and information. Knowledge management is a very broad area. It is often mentioned in the context of enterprises and intellectual capital. So far, almost all approaches focus explicitly or implicitly on sharing knowledge within groups of similar users. This is however not appropriate. Expert work requires a constantly growing degree of specialization. Many practical problem on the other hand can only solved by intensive collaboration across domain boundaries. A mutual understanding and awareness among experts from different domains is a prerequisite for this kind of collaboration. While this fact is widely acknowledged, there is a need to enable knowledge management in an interdisciplinary context on a technical level.

This chapter shows how aspect based tagging and collaborative structuring can be used to achieve this goal. The basic idea is the following. Individual experts are allowed to create personal tags as representation of their view on a topic. By recommending and copying such tags, communities of users with similar views emerge. These views can be made explicit by analyzing them using data mining. These data mining algorithms are modified in a way that recommendations correspond to certain modes of knowledge sharing, such as inner-community or cross-community knowledge sharing. This helps users to structure and navigate complex information spaces either according to their own views and terminology or to explore views on a topic that correspond to experts from other domains.

First, we give a very brief overview of knowledge management and sharing in section 7.2. In section 7.3 we discuss the problem of heterogeneous user groups, together with benefits, challenges and approaches. Section 7.4 provides a more focused discussion on the problem of how data mining relates to the problem of heterogeneity. It is shown that traditional data mining approaches, applied in most knowledge management systems lead to results that support only within community knowledge sharing. In section 7.6, the basic concepts of the Knowledge Explorer are discussed, a collaborative platform for knowledge sharing in heterogeneous expert communities. Section 7.6 provides details on how collaborative structuring is applied in the Knowledge Explorer.

7.2 Information and Knowledge Management

The key idea of knowledge management and sharing is that a very valuable part of an organization is its knowledge and that this knowledge should be maintained and applied optimally.

According to a popular definition, the of goal knowledge management is to

> "Improve organizational performance by enabling individuals to capture, share and apply their collective knowledge to make optimal decisions" [163]

This definition covers several aspects:

- *Sharing knowledge* refers to communicating knowledge among several people.
- A very important concept of knowledge management is a *community*, thus a group of people that work together or have similar interests and goals.
- Knowledge is *applied*, thus it leads to decisions.
- The goal of knowledge management is to improve *performance* at the given task. In fact, the success of knowledge management often depends on improving the performance not only globally but for each participant, as otherwise people refuse to use the system. Performance and evaluation are of significant importance for knowledge management.
- To *capture knowledge* indicates that knowledge is often implicit, and in order to share it, it has to be made explicit first.

Knowledge Management consists of many different processes that make up the so called „knowledge life cycle" [116]

- *Capturing and formalizing knowledge*
  Having an explicit representation of knowledge is often a prerequisite for sharing it in a formal and systematic way, as well as for any kind of automatic processing. Therefore, the process of knowledge externalization is of special importance for many (technical) knowledge management solutions.
- *Structuring knowledge*
  Knowledge that should be shared in a formal and systematic way must be represented using a common formalism. This representation determines the ways in which the knowledge can be added and retrieved. Representation mechanisms range from text documents to highly structured data, such as data and knowledge bases.
- *Sharing knowledge*
  Users can share knowledge asynchronously through a common repository or by direct communication. The actual mode depends on the task at hand and knowledge management systems will usually provide several modes of communication.
- *Applying knowledge*
  Knowledge is usually applied to support some problem solving or decision making

task. Applications may range from very unstructured tasks, such as writing an essay, to very structured decisions, e.g. whether or not to buy a certain cellphone.

- *Evaluating knowledge*
  Measuring the performance of any technical system is a key to improve it. The same is true for knowledge management. Beside evaluating a system globally, individual pieces of knowledge can be evaluated. In many help desk applications, for example, users have the possibility to rate individual articles on solutions according to how helpful they are.

- *Introducing and maintaining knowledge management in an organization*
  The whole process of choosing, installing, maintaining and administrating a knowledge management system is an important practical issue for successful knowledge management. This task includes choice of technology, teaching, nominating knowledge managers and similar tasks.

This enumeration of basic knowledge- and information management processes cannot claim to be exhaustive. Though, it covers the essential aspects relevant for this work.

7.3 Heterogeneous Expert Communities

7.3.1 The Problem of Heterogeneity

From the point of view of knowledge sharing, heterogeneity is first of all a problem. Heterogeneity has many aspects in this context: technological ones (e.g. different hardware and software platforms), syntactical ones (different data formats and terminology), semantical ones (different concepts) and pragmatical ones (different use of concepts, workflows, etc.). Heterogeneity on lower levels can often be resolved by enabling global standards. The use of web based technology (as presented in section 2.3.3) is an important example for such a solution. Syntactical heterogeneity can often be resolved by schema or ontology mapping approaches (as discussed in section 3.3.4).

While standards and ontology mapping work well on a syntactical level, the semantical and pragmatical level are often much more complicated to handle. First, the semantic of concepts is often not explicitly formalized. Second, concepts often do not differ only by their label but semantically as well.

The lack of explicit formalization and semantical differences make a mapping on a semantical level very difficult or even impossible. This is especially true for expert communities, in which the members are strongly influenced by an educational background that determines these concepts. To enable knowledge sharing in heterogeneous expert communities novel approaches are needed.

7.3.2 The Importance of Heterogeneity

While heterogeneity is a problem on a technological level, it is of essential importance on a user level for several reasons.

First, heterogeneity is often the key to user acceptance. If users have to obey a formalism that does not fully reflect their needs, they will not use it. Actually, they tend to use highly suboptimal channels of knowledge sharing instead (as e-mail), just as they do not like the representation mechanism of a given knowledge management system.

Second, many current knowledge sharing scenarios are inherently heterogeneous, such as systems in which experts from different domains collaborate. Heterogeneity here is a prerequisite to solve complex problems.

Third, heterogeneity is an important source of innovation [24]. Often, knowledge creation and technical development can be structured in two phases: innovation and consolidation. While traditional knowledge management solutions support communities in a consolidated phase, they do not support innovation. Actually, a successful system should support both, consolidation and innovation.

Forth, heterogeneity often improves quality, as the co-existence of different views naturally leads to a competition, and only such views will be adopted on the long run that are accepted by many users.

To fully profit from these benefits, a knowledge management system must support heterogeneous user groups adequately. In the next section, we give an overview on existing approaches to this problem.

7.3.3 Approaches to Cross Community Knowledge Sharing

The model of perspective making and perspective taking provides a theoretical foundations for dealing with cross community knowledge sharing. This model describes the processes of knowledge exchange between different „communities of knowing“ [23]. Perspective making refers to intra-community development and refinement of knowledge, whereas perspective taking refers to making the thought worlds of different communities visible and accessible to each other. [23] propose that these processes are intrinsically connected: a community develops new knowledge both through social exchanges and knowledge discourses between its members, as well as by taking on perspectives of others. The interplay of these two processes then provides the ground for allowing knowledge to be exchange between different communities.

Another model related to the problem of cross community knowledge sharing are „boundary objects“ [166]. Boundary objects are knowledge artifacts that embody different perspectives and can be interpreted in different ways. Such boundary objects are seen as essential means for supporting cooperation between different communities in a way that allows each community to retain local perspectives and yet these perspectives to become interconnected.

Different authors have emphasized the largely tacit nature of human knowledge [23] and the difficulties of codifying and formalizing socially distributed knowledge in communities. Existing solutions to this problem can be roughly classified into three main approaches: the „internalization“ model based on individual reflection on the community discourse, the „socialization“ model based on direct interaction mediated by CSCW technologies and the „externalization“ model based on the explicit construction of shared conceptualizations.

The internalization model is the only model supported by basic community technologies such as mailing lists, bulletin boards and discussion forums. The development of a shared context requires members' extensive and active participation in the community exchange. There is no mode for the shared understanding of the community to be expressed, and the repository of the collective memory is an unstructured space of many interrelated but rather isolated pieces of information. Context is very difficult to establish.

The socialization model is connected to approaches that aim at supporting the sharing of social knowledge through a shared virtual space (e.g. [51]). This is the so-called awareness and knowledge socialization approach, which can be related to two basic premises. The first is that by providing mutual awareness of spatially distributed but contextually related users (e.g. working on same task, or belonging to same community) by means of a shared virtual space, the cognitive distance between them is bridged. The second is that once this cognitive distance is bridged, the conditions are established for the users to enter into conversations through which they exchange otherwise inaccessible personal knowledge. There are several variants of this basic model. Some of them are, for example, connected to the constructionist theory of learning [142], others focus on the establishment of identity and the self-organizing of social norms (e.g. [183]). The main shortcoming of computer-mediated socialization approaches is that the sharing of implicit knowledge requires extensive manual interaction between individual members, and the resulting exchange still resides only in individual users. There is no possibility to visualize or access the resulting structure of shared understanding or to use it to support new users.

The externalization model is addressed by approaches aiming at supporting the explicit formulation of shared conceptualizations in form of knowledge ontologies. Ontologies represent models for formal descriptions of concepts and named relationships between them, that describe how a given individual or a group of people understands a particular domain of knowledge. Ontologies often have to be created explicitly by hand and require a process of explicit community negotiation for achieving a consensus about the shared understanding that is to be expressed (corresponding tools of collaborative ontology engineering are described in section 3.3.1). Once created they can be used to access and navigate the community information pool, as well as to visualize the semantic structure of the shared community understanding. An example of existing efforts for building such ontologies in different disciplines but interrelated to each other is the DublinCore initiative (http://www.dublincore.org). The Open Directory Project aims at a collaborative definition of a somewhat simpler taxonomy for manually mapping the content of the whole Web (http://dmoz.org). Such approaches to creating externalized representations

of a shared conceptual structure require explicit negotiation for achieving consensus between the members. There is no or little support for expressing the personal points of view of individual users and putting them in relation to the shared structure. At the same time, one of the essential mechanisms of knowledge creation is the ability to change perspective and see the world with „different eyes". Finally, the challenge remains of how to provide insight into the underlying values and beliefs shared by a group of users, as fundamental elements influencing their thinking, judgment and the creation of new knowledge.

While the aim of ontologies and other forms of knowledge externalization usually is to create a formalized common understanding, a radically different approach is to allow different knowledge structures to co-exist and to mediate between them automatically by means of a mapping between different taxonomies, categorization structures or ontology schemes (as described in section 3.3.4). These approaches offer the benefits of allowing a decentralized creation and maintenance of knowledge (and thus personal views on a domain) with little explicit coordination. Finding an intentional mapping between conceptualizations is, however, far from being trivial and usually depends on a logical description of concepts. Thus ontology mapping also depends on the assumption that the meaning of concepts and thought worlds of communities can be codified in a formal representation and therefore suffers from the same basic problem as the other knowledge externalization approaches.

While explicit externalization is often costly and unsuitable for capturing tacit and social knowledge, an alternative is to infer the common understanding of groups of users from their interactions. This is the approach taken here. The basic idea is to operationalize the concepts of perspective taking and perspective making and of boundary objects by means of data mining.

7.4 Cross Community Knowledge Sharing and Data Mining

The aim of this work is to support users in organizing and searching information across community boundaries. Basically, the collaborative structuring approach is very well suited for this task, as it allows for emerging concepts, such that communities with different views can co-exist in a single system. Recommendations help to consolidate the system, as popular views within a community automatically become dominant. This corresponds to the perspective making metaphor. Applying tag structures to arbitrary sets of items corresponds to the perspective taking metaphor. In this way, knowledge within one community becomes accessible to members of other communities. Boundary objects are tag structures covering items stemming from different communities. As users may combine arbitrary tag structures, such boundary objects can be easily identified by collaborative structuring. To fully profit from this idea, we extend the basic mechanism presented in part two of this work in one point.

Collaborative filtering systems [160] are designed to enable knowledge sharing exactly among users with similar preferences. A cineaste probably does not want to be even

aware of all the action film fans that also utilize the system. Most collaborative filtering algorithms therefore apply a measure that decides which users are similar to a given user. Only ratings of these similar users (or predictors) are applied to make recommendations. For sharing knowledge in heterogeneous expert communities the situation is different. The idea is often the other way around: we want to enable experts to find connections between domains, relevant research in another domain, basic relationships of one domain to another, etc. In the following, we describe how the methods presented in the second part of this work must be extended to support this kind of exploration.

We assume a set of communities $\mathcal{B}$, each consisting of a set of users, thus for all $\mathbf{B}_i \in \mathcal{B}$: $\mathbf{B}_i \subseteq U$. Each user belongs to a least one community, users may however belong to more than one community. Following the idea of collaborative filtering, each user has a set of predictors. To make recommendation for a user, only information provided by this set of predictors is employed. Let again N_i denote the set of predictors of a users u_i. Then, knowledge sharing can easily be characterized by the following definition.

Definition 7.4.1. User u_i *shares knowledge* with user u_j, denoted by $u_i \preceq_{ks} u_j$, iff $u_i \in N_j$.

This definition captures the notion of knowledge sharing from a point of view of (instance-based) data mining. For collaborative structuring this means that only the tag structures of users are applied that are in the set of predictors corresponding to the current user.

Now, the partial order denoting knowledge sharing can be classified into different modes of knowledge sharing, depending on the membership of users to communities:

1. ***Reflexive "knowledge sharing"***
   $u_i \preceq_{KS,r} u_j$ iff $u_i = u_j$ and $u_i \preceq_{ks} u_j$
   Only information by the active user herself is employed for learning. This form of knowledge sharing is useful to get an overview of one's own structures.
2. ***Inner-community knowledge sharing***
   $u_i \preceq_{KS,p} u_j$ iff $\forall \mathbf{B} \in \mathcal{B} : (u_j \in \mathbf{B}) \Rightarrow (u_i \in \mathbf{B})$ and $u_i \preceq_{ks} u_j$
   Sharing knowledge is only allowed between members of the same community. As described before, this is useful if users want to ignore users from other communities altogether. This will be also denoted as *restrictive personalization*.
3. ***Cross community knowledge sharing***
   $u_i \preceq_{KS,f} u_j$ iff $\neg\exists \mathbf{B} \in \mathcal{B} : u_i \in \mathbf{B} \wedge u_j \in \mathbf{B}$ and $u_i \preceq_{ks} u_j$
   This form of knowledge sharing is useful if users want to gain insight in how other communities label and structure items, as to find, e.g., relevant research and terminology in other domains.
4. ***Knowledge sharing by cross domain users***
   $u_i \preceq_{KS,cross} u_j$ iff $|\{\mathbf{B} \in \mathcal{B} | u_j \in \mathbf{B}\}| > 1$ and $u_i \preceq_{ks} u_j$
   Users belonging to more than one community are of special importance, as they can serve as bridges between domains, highlighting boundary objects and bringing into relation item and topics that have no obvious connection.

Usually the set of predictors is determined automatically by some kind of optimization procedure (as for collaborative filtering) and we can only analytically determine to which mode of knowledge sharing this corresponds. In the approach presented in this chapter, we turn this relationship around. We start with a desired mode of knowledge sharing and then select the set of predictors correspondingly. If, for instance, knowledge should be shared within a given community only (mode 2), then we use only tags created by members of this community in the data mining process.

This procedure enables us to support different modes of knowledge sharing without altering the actual data mining algorithms presented in part two. We simply impose additional constraints on the subset of features that can be selected during feature selection.

7.5 The Knowledge Explorer

7.5.1 Basic Concepts

The main aim of the Knowledge Explorer is to allow heterogeneous expert communities to collaboratively navigate and explore cross community document collections. Scientific publications are the most important carrier of knowledge in this context. In the following, we assume that documents are uniquely identified items from which textual features (using the bag of words approach [154]) can be extracted. They correspond to the concept of items as introduced in section 1.2. Additionally, there are document pools, corresponding to item collections. Document pools are sets of documents connected by their common origin. Typical examples are proceedings of conferences, journals, announcements of a given institutions and so on. Additionally, documents can be annotated by user specific tags, called personal concepts. These tags are assigned by creating personal knowledge maps, which are described in the next section.

7.5.2 The Knowledge Map Metaphor

The most important concept in the Knowledge Explorer system is the knowledge map metaphor. A knowledge map is a two dimensional area on which items are arranged. We distinguish two kinds of knowledge maps, document maps and concept maps (see Fig. 7.5.1). *Document maps* contain individual documents each represented by a point in a two-dimensional space. Similar items are located close to each other. *Concept maps* contain tags. Similar tags are located close to each other.

A document map displays each document together with its meta data and descriptive terms. These terms are derived either from the text itself or from user tags. As the user selects a document, particularly similar documents are highlighted. Additionally, areas of a document map can be tagged with a label, denoting that the documents in this area belong to a concept.

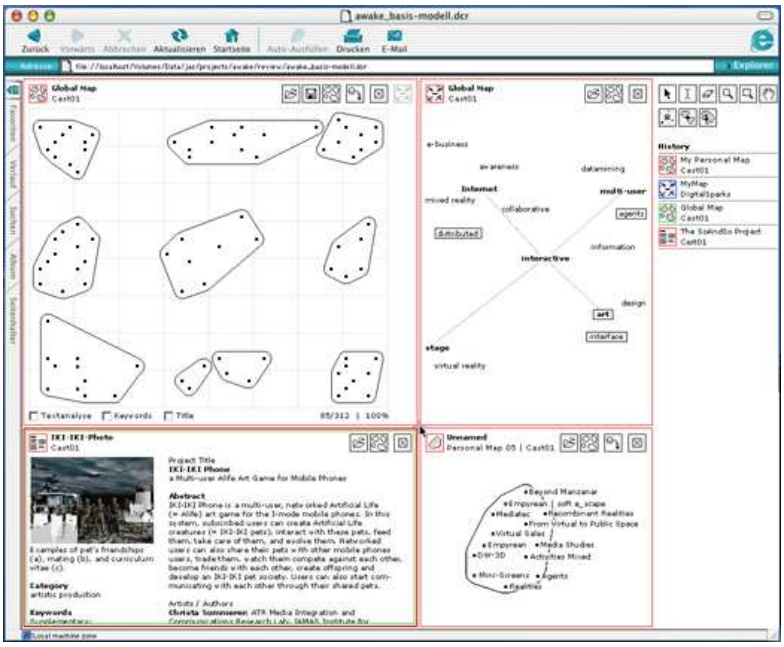


Figure 7.5.1: The interface of the knowledge explorer. The upper left window shows a document map, the upper right window shows a concept map.

Document maps can be created by the system automatically or manually by the user. In the latter case, they are denoted as *personal document maps*. These personal document maps allow users to express their view on a set of documents by arranging the documents and by assigning them to personal concepts. Each user may create several personal document maps.

System generated document maps are created automatically by applying a SOM algorithm [101]. The underlying feature space initially consists of text features extracted from the documents. Later, this feature space can be enriched with features extracted from personal document maps. Enriching the feature space can be performed in different ways, depending on the desired mode of knowledge sharing (see below).

Concept maps are always automatically generated and correspond to one or several document maps. They show the major concepts that underlie the document map and how they are related. The following types of concept maps are supported:

- *Personal Concept Map*
  The map corresponds to a single, personal document map.
- *Gesamt Personal Concept Map*
  The map corresponds to all maps created by a given user.
- *Community Concept Map*
  The map corresponds to all maps created by users that exactly belong to a given community.

- *Overall Concept Map*
  The map corresponds to all document maps.

The creation of concept maps is described below.

How does this model relate to the basic model presented in section 5.3.1? As described above, the concept of users and items is a special case of the general model. Personal concepts correspond to the user assigned tags. Personal document maps correspond to aspects. Personal concepts are not allowed to overlap. This entails that only flat tag structures can be created. The Knowledge Explorer model is richer in that each item does not only have user assigned tags but also a user assigned position for each knowledge map it appears on. We discuss the consequences of this extension below.

7.5.3 Operations on Knowledge Maps

Beside interactive browsing, the system supports the following operations on knowledge map:

1. *Generating a document map*
   Given a set of documents, such as a document pool, a document map can be created automatically. This is usually a point of departure if no additional information is available.
2. *Generating a concept map*
   For each document map, a concept map can be created. Additionally, the system allows to create concept maps that correspond to all maps created by a single user, by a community or by all users.
3. *Applying a document map*
   Documents can automatically be classified into any document map by learning a classifier function. This operation serves two purposes. First, a personal document map can be populated with additional relevant documents automatically. Second, users may view arbitrary documents collections through the eyes of other users or communities by applying a foreign personal document map. They can even take a look at their own personal document maps from a different perspective.
4. *Searching for documents*
   The Knowledge Explorer supports standard keyword based search for documents. Result documents are highlighted in all knowledge maps. User tags are considered as well.
5. *Searching for personal document maps*
   Users may not only search for individual documents but for complete personal document maps. This enables them for instance to discover alternative contexts in which a document is used.

Operation 1 corresponds to the basic clustering task (possibly based on a collaborative feature space). Operation 3 corresponds to a classification task and operation 5 corresponds to the LACE approach (as described in section 6.4). These operations were

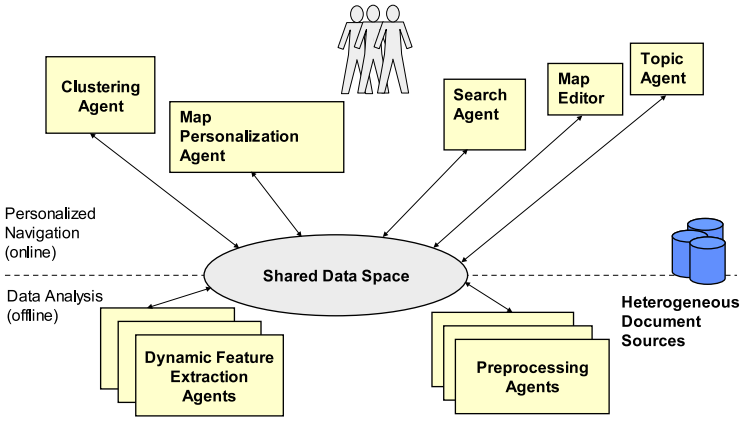


Figure 7.6.1: The Knowledge Explorer architecture: The lower part depicts the feature extraction and preprocessing agents that work asynchronously with the user interaction agents depicted in the upper part.

already covered in the second part of this work and are therefore not discussed here in detail. Operation 4 is a standard information retrieval task and will not be discussed either. The only specific functionality is the concept map creation, which will be covered below. In the following we also discuss how these operations can be combined with different modes of knowledge sharing by community specific feature selection.

7.6 Knowledge Explorer Data Mining

7.6.1 Basic Architecture

The Knowledge Explorer data mining system consists of two different kinds of agents (Fig 7.6.1). One group of agents is concerned with responding to user requests. These agents have to work very efficiently, as interactive work requires very short response times. To achieve this, we use a second group of agents, which asynchronously preprocess data and stores it in intermediate data structures. These agents take much of the work load from the first group of agents. Using this strategy, we can use sophisticated and costly data and interaction analysis methods and even so have short response times. In the following, we will roughly describe some of the system's components.

Data Preprocessing and Feature Extraction Agents

Preprocessing agents allow the user to create a pool of documents by connecting data-sources to the system. The user can either choose between readily available data sources or manually connect other structured data-sources (such as databases and semi-structured document repositories).

Feature extraction includes a text-analyzer for encoding semantic properties of texts into a vector space model [154] and the extraction of meta data, such as information about the author or the year of publication.

Dynamic Feature Extraction Agents

This layer contains agents for semantic processing of user created, personal maps. While preprocessing is performed only once for an item, interaction analysis is performed at regular intervals, as the set of personal maps changes.

Personal Information Agents

Personal information agents perform all data mining tasks that are directly related to user requests. This includes all functionality described above, such as classification, clustering, key word extraction, etc.

Visualization Agents

The visualization agents provide post-processing of the data and of the interaction-analysis done by the personal information agents. They take care of collecting all necessary information from different agents, needed to construct all the information layers of the document map and the concept map described in the previous section. Based on the selected visualization model, the visualization agents then retrieve information stored by the data integration assistant and preprocessing agents in order to fill in additional information (e.g. titles, abstracts, term-document frequencies etc.).

Agent Communication and Coordination

We use two simple techniques for agent communication and coordination. The exchange of data between agents is realized as feature facilitator described in section 4.5.3.

This approach allows that on the one hand there are possibly several agents working on preprocessing in parallel. On the other hand, the preprocessing agents can provide data for the request processing agents asynchronously, without direct communication or coordination. Though within each group of agents, there is a need for a tighter form of coordination. In the Knowledge Explorer framework, this is achieved by a simple event service based on XML and SOAP.

7.6.2 Knowledge Sharing and Feature Engineering

As described above, almost all intelligent operations provided by the Knowledge Explorer can be reduced to data mining tasks presented in chapter 4 and chapter 5. The most im-

portant addition to this basic framework is the support for different modes of knowledge sharing. How this can be achieved by feature selection will be described in the following.

As described above, we start with a mode of knowledge sharing and then select a set of predictors for a user accordingly. This selection is based on the information which user belongs to which communities. In many applications, this information is part of the available domain knowledge. If this is not the case, it can easily be extracted by data mining. The following procedure is proposed. First, the documents in all available pools are clustered using k-means. Users are then assigned to a community if they have selected documents from the corresponding document cluster. In many applications, the step of document clustering can be omitted, as documents are assigned to communities by their origin, e.g. as conference proceedings, journals, etc., which are in most cases community specific.

Given the information which users belong to which communities, feature selection can be applied to extract features from only a subset of personal maps, corresponding to the desired mode of knowledge sharing.

This functionality can be added easily to the feature facilitator. The feature selection according to the community and the mode of feature sharing is very efficient and does not affect the response time, as features are selected without analyzing their values.

Using community memberships offers an additional benefit concerning distributed feature extraction. Unsupervised feature aggregation (see section 5.4.5) can be performed more selectively by only aggregating tags provided by users that belong to the same community. This helps to save storage space and computation time at a lower chance of adding false information to the system.

7.6.3 Personal Agents

In the following, the most important intelligent functions of the Knowledge Explorer are briefly discussed. As they mostly correspond to operations covered in detail in the second part of this work, only specific issues are discussed here.

Classification

Classification is used to add new items to a given personal map. This function is used to allow users to view arbitrary document collections „through the eyes“ of a personal map (perspective taking). Also, new documents can be added to a personal map automatically, such that the user does not have to insert these items manually.

The classification algorithm is based on nearest neighbor (see section 1.4.4). For each document, a set of similar documents is selected. An item is assigned the class that a majority of predictors is labeled with. This choice is motivated by two characteristics of nearest neighbor. First, it is easy to explain to the user the result of a classification, especially as similarity information is displayed in the user interface anyway. Second,

items do not only need a class label, they need a position on the two dimensional map as well. Using nearest neighbor, this position is calculated easily as the component-wise average of the positions of all predictors.

The similarity measure used for classification is derived by applying cosine similarity to the corresponding feature space, as described above.

Clustering

Clustering is achieved by applying a Self Organizing Map [101] to a set of items. These items are again described by a feature space selected according to the mode of feature sharing. Additionally, for each point on the SOM, representative tags are selected to support the navigation [79].

Concept Map Creation and Similarities

A concept map contains terms, each represented by a point. These terms are connected to relevant documents and to related terms.

In a first step, a set of main concepts is derived from the document maps that correspond to the concept map (depending on the type of concept map). This is achieved by calculating for each tag to how many documents it is assigned. If a tag is assigned to a document twice (by two different users), both occurrences are counted. The tags that achieve the highest number of assigned documents are selected as root concepts.

In a second step, similarities among the root concepts and between root concepts and other tags are determined using cosine measure on the corresponding collaborative feature space. In a third step, relevant document are determined for each term on the map by document frequency (the number of times a document is assigned a given term).

Matchmaking

Matchmaking is based on the LACE algorithm (see section 6.4). Results are limited to the personal maps that correspond to the feature set that is used. In this way, users may search for maps in other communities or maps created by interdisciplinary users. Another restriction of the Knowledge Explorer system is that only individual maps are recommended. A combination of maps is not supported.

7.7 Conclusion

Sharing knowledge in heterogeneous expert communities is important and highly challenging. Current knowledge management systems „naively" apply standards data mining

algorithms for search and navigation in document collections. These algorithms implicitly support only within community knowledge sharing. To enable knowledge sharing between different communities, novel algorithms are needed. In this chapter, it was shown that collaborative structuring can be easily extended to allow for different modes of knowledge sharing. The key is to filter tags and tag structures from which features are derived according to these modes. In this way, existing algorithms do not need to be modified.

8 Collaborative Media Organization

8.1 Introduction

The aim of data mining in general is to find useful patterns in large data collections. Data mining was applied to multimedia data in diverse areas ranging from robotics to multimedia information retrieval. Distributed computing plays an important role in this process for several reasons. First, mining multimedia data often requires huge amounts of resources in storage space and computation time. To make systems scalable, it is important to develop mechanisms that distribute the work load among several sites in a flexible way. Second, multimedia data is often inherently distributed into several databases, making a centralized processing of this data very inefficient and prone to security risks. In this chapter, we discuss a particular interesting application of distributed multimedia processing: the organization of private media collections. Popular file sharing applications allow user to search and to exchange media files in loosely coupled domains. Media organizers as iTunes[1] or amaroK[2] offer some limited intelligent functionality through the creation of intelligent playlists. However, these playlists basically are dynamic database views. Applying data mining methods to the field of personal music management offers many new opportunities. Typical applications include the classification of music items according to predefined schemes like genres [68, 99, 109, 145, 184, 208, 67], automatic clustering and visualization of music [126, 139, 141, 157], recommendations of songs [168, 15], as well as the automatic creation of playlists based on audio similarity and user feedback [140, 112].

A general lack of these approaches is that they do consider media items and views on these items as global entities. In fact, media items are usually highly personal and subjective and must thus rather be considered as local entities [90]. Therefore, users should be allowed to manage items in a personal way using own tags. Still, they should profit from what other users did, such that the effort to develop and maintain a personal view is minimized. Collaborative structuring allows for this balance between creating structures by hand and copying them from other users.

We created Nemoz, a prototypical system that applied collaborative structuring in the context of media organization. The aim of Nemoz, as application, is to support users in organizing their media files in a distributed, collaborative way. Peers in the Nemoz system are assumed to be connected over a loosely-coupled p2p network.

[1] http://www.apple.com/itunes
[2] http://amarok.kde.org

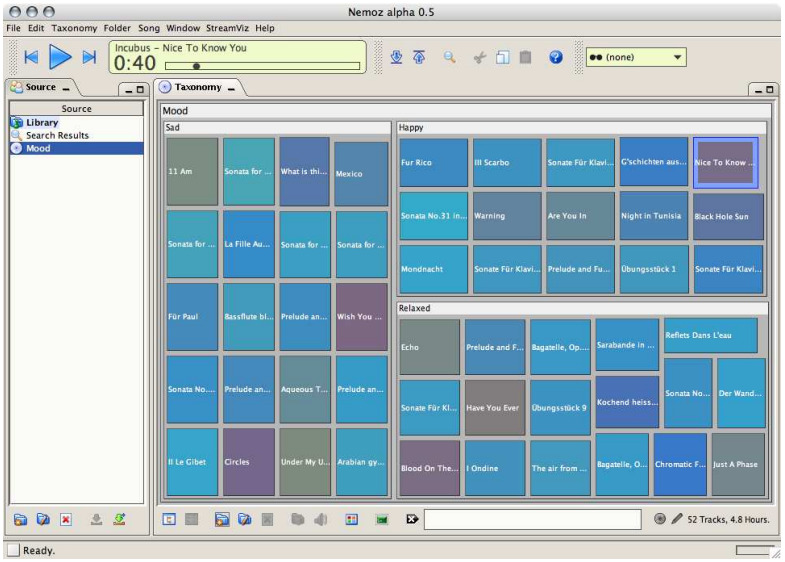


Figure 8.1.1: The interface of the Nemoz application

8.2 Challenges for Distributed Multimedia Mining

Distributed multimedia data occurs in many current applications, ranging from media organizer applications, such as Nemoz, over mobile robots to sensor networks. Applying data mining to such data is very challenging for several reasons. Multimedia data, such as images, audio, video or general time series, are hard to handle with traditional data mining techniques. The underlying data representation is very complex and large. Furthermore, different data mining tasks require completely different preprocessing. For example, the characteristics needed to separate different jazz styles from each other differ completely from the ones needed to separate styles of rock music. The data volumes in multimedia applications are usually huge. Distributed analysis of such data is very restricted, as even efficient networks usually do not allow to transfer large amounts of data just for the purpose of analyzing it [41]. Unlike business data, multimedia data is typically stored and processed on computationally weak devices that do not necessarily have a high speed Internet connection. Examples are wearable media players, cell phones or distributed sensor networks. Processing in such settings cannot rely on expensive calculations and on reliable high speed Internet connections. Typical network architectures are rather based on ad hoc or p2p networks.

P2p and ad hoc networks are characterized by a low reliability of individual nodes that usually leave or join the system at high pace. Also, such networks contain many nodes that are only loosely connected. Traditional distributed algorithms, that are based on global agreements, will likely fail in such a setting. Other characteristic of p2p and ad hoc networks require special treatment. The structure of p2p networks is usually such that there are few nodes with rich resources and many connections and many nodes with

few resources and few connections. Data mining algorithms must respect this natural unbalance instead of forcing the work load to be equally distributed. Free riders are nodes that only consume services without providing any. Such nodes have a negative influence on performance of the whole system [149]. Traditional data mining applications are usually based on a data warehouse specially built for data analysis. Distributed data mining application must, on the other hand, operate incrementally. As, for instance, users of the Nemoz system add new items, these items must be analyzed as they arrive.

Connected to this issue is the problem that the system must perform data mining in near real time. This is obvious for sensor networks that must react on unexpected events as quickly as possible. However, even a typical Nemoz user is not willing to wait for data mining process more than some seconds. Many traditional data mining methods are not able to deal with such tight restrictions on processing time.

Collaborative structuring and distributed feature extraction are promising approaches to cope with several of the above challenges. The complex nature of media data can be handled by user assigned tags, that employ the common knowledge of a user base instead of extracting semantic information from the media data itself. Such tags can be extracted and shared on a wide variety of devices by the distributed feature extraction methods proposed in chapter 4. These methods save computation time and can easily be applied even in networks that allow only minimal assumptions concerning connectivity. Even if user tags are not sufficient, content based features can be shared efficiently, without actually sharing any data. The proposed methods make use of the natural unbalance of nodes in computational power by allowing each node to decide locally which features to extract and to share. The simplicity of the feature sharing approach allows for simple, general protocols that do not make any assumptions on the underlying system and data mining methods. This approach is well suited to create a general platform that allows the interoperability of a large variety of devices. Incremental processing is supported, as new items can be annotated with existing tags or can be structured from scratch. Features for new items can be obtained easily by querying only for the subset of items that is actually new.

In the following, we show how these methods are actually implemented in Nemoz.

8.3 Nemoz Concepts

8.3.1 Basic concepts

Items in Nemoz are media files. We assume that these items are globally, uniquely identified and static over time. We further assume that arbitrary features can be extracted from these files. Such features can be content-based, as audio features from music files. They may also stem from other sources, as web pages or user reviews [15, 157]. This corresponds to the basic model introduced in part two of this work. The same holds for the concept of users in Nemoz. The most important concept in Nemoz are user tags, that will be described in the next section.

8.3.2 Nemoz Tagging

Nemoz tagging basically works as introduced in section 5.3.1. Users may assign arbitrary tags to media items. Such tags can be organized hierarchically. Furthermore, we assume a set of aspects that are used to arrange and manage tags.

8.3.3 Nemoz (Intelligent) Operations

The aim of Nemoz, as an application, is to support users in organizing and exploring media collections. Nemoz clients offer a user friendly, state of the art interface that allows users to organize media collections very conveniently. Nemoz offers basic player and file sharing operations. Users can browse media collections of other peers, download files or play them remotely.

The major focus of Nemoz is, however, on operations that help users to create and maintain personal tag structures. Nemoz supports the following intelligent operations on personal tags:

1. *Assign tags automatically*
   Given an item and a set of user defined tags, the system can assign tags to these items automatically. This process is guided by the aspects. For each aspect, exactly one most specific concept is selected. This kind of classification has two aims. First, it helps the user to maintain her tag structures. She might only label a small part of all items explicitly and then the system assigns the remainder of the items automatically. Second, assigning tags to items temporarily allows a user to browse a remote media collection using the own tags.
2. *Automatically structure a set of items*
   Given a set of items without a tag, this set of items can be structured automatically by exploiting tags created by other users and/or content related features. The system provides several solutions (aspects). A very common application of this function is to refine overfull nodes in a hierarchy. In this way, a user can iteratively structure a set of items in a top-down manner.
3. *Search*
   The systems provides extended search functionality. Users may, for example, search for a tags, users, meta data and any combination of this information. Also, similarity search is supported. This search can be based on any feature space. This allows to search for similar items according to an aspect by applying exactly the feature space that is associated with this aspect.
4. *Visualization*
   Based on a feature set, visualization methods can be invoked, that, for instance, map all items to a two dimensional space. As the feature space may correspond to an aspect, items can be visualized according to several different, user defined aspects in this way.

These functions largely correspond to the basic functions of collaborative structuring as proposed in chapter 5. Operation 1 is simply achieved by classification. Operation 2 is achieved by clustering and especially the by the LACE approach (see chapter 6). Search is based on traditional information retrieval methods and visualization on dimensionality reduction methods.

8.4 The Nemoz Framework

8.4.1 Data Mining

Nemoz uses a query-based cooperation pattern, as presented in chapter 4. As Nemoz is a p2p system, we assume the fully distributed approach to feature sharing, based on range limited broadcast.

This approach is very well-suited for two reasons. First, an important restriction in Nemoz is that active task distribution (distributing data mining tasks among nodes in some optimal way) is not applicable. This is due to the fact that media items cannot be sent easily over a network. Second, many features are only relevant to a subset of nodes. Disseminating them to all nodes would be highly suboptimal.

In our approach, each node decides which calculations to perform and which features to store, depending on its resources and the current work load. This especially includes local feature aggregation and the extraction of new features using sophisticated but costly methods, as e.g. described in [119]. Nodes shares the results of their calculations with other nodes. If a node encounters a new data mining task, it first queries the other nodes, to find useful features for this task. These features are then obtained and incorporated into the local feature space.

The actual data mining is performed using standard methods. Again, nearest neighbor is used for classification. Collaborative clustering is based on the LACE approach, presented in section 6.4. Similarities are calculated using Euclidean measure on mixed feature space of (aggregated) user tags and other features (as described in section 5.4).

8.4.2 System Architecture

Nemoz, as a framework, offers the basic building blocks to develop and evaluate distributed data mining methods for collaborative media organization. The architecture of the framework is depicted in figure 8.4.1.

The Graphical User Interface is responsible for all human computer interaction. The application layer contains basic functionality of the system that is part of the Nemoz core. Plugin and scripts can be added on top of this layer to extend the system. The data mining service backs the intelligent functionality. It makes use of RapidMiner (Yale) [124], an extendable platform for data mining, offering diverse classification, clustering

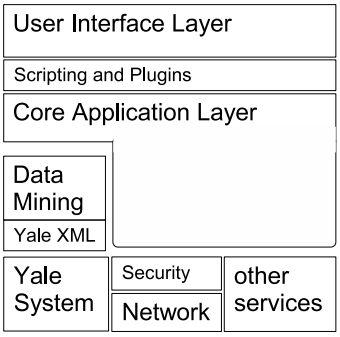


Figure 8.4.1: The Nemoz architecture

and preprocessing operators. RapidMiner is accessed by a scripting layer that allows to represent complex processing trees in an open XML format. The network service offers all necessary functionality to communicate with other nodes, using data access objects to allow for a flexible XML serialization. The privacy layer allows to restrict access to the local resources. Users can define the accessibility and visibility of items and structures to other users and which information can be used for data mining. Finally, there are several other local services responsible for playing files, persistency, etc.

Nemoz as a platform allows for the combination of different components, connected over a common cooperation protocol. This makes the whole system extensible on a node level. It would be for example easy to add new nodes performing additional forms of multimedia analysis.

8.5 Conclusion

Processing multimedia data is very challenging. This is especially true if this data is distributed over a loosely coupled network, as it is the case in many current media organization applications. Media items are stored locally on devices that differ strongly in their capabilities. Tagging and collaborative structuring allow to annotate multi-media items in a very flexible way making only minimal assumptions about the computational power or the network connection of the devices. This helps to solve the problem of automatic annotation of such data. The co-existence of different aspects and views is very natural for media organization and therefore essential of the acceptance of the system. Distributed feature extraction allows to share tags and features easily. As it is based on standard queries, it can be combined with state-of-the-art p2p resource location software, such as distributed hash tables. This makes the Nemoz approach scalable and applicable in a very broad range of underlying networks.

9 Conclusion

Every new technology raises the question whether it simply makes some aspects of our life more convenient, or whether it actually opens completely new possibilities. This is especially true for the Internet. On a very basic level, the Internet is a mechanism to provide access to information in a transparent, location independent way. However this might not look very impressive on a first sight, the implications are enormous. The ability to exchange any kind of data and information, independent of one's current geographical location, paves the way for patterns of cooperation that have not been possible before. Although the Internet, as mass medium, is relatively young, a huge range of applications stands evidence for these achievements. The collaborative dictionary Wikipedia contains more articles than the Encyclopedia Britannica. On the Usenet, each day 3 Terabyte of information is exchanged in more than 60.000 discussion groups. The number of World Wide Web pages is growing too fast to even be measured accurately. People share knowledge, experiences, news, recommendations. If someone faces a problem, e.g. which cell phone to buy, she can profit from the experience users made world wide. The same holds for many technical or scientific problems.

Successful navigation, search and collaboration on the Internet depend on mechanisms that allow users to access large and complex information spaces in a structured way. A key to this problem is to respect and reflect the natural heterogeneity of users and information on the Internet. Heterogeneity is a very broad concept, including differences in topic, language, preferences and interests, pre-knowledge and use context. Actually, the ability to serve very heterogeneous user groups and interests is one of the main achievements of the Internet and one of the reasons for its current success. It is often expressed by the term „long tail“, denoting that all „less-popular“ resources together can make up an at least equally important part of the content as the popular ones. Still, when it comes to information structuring, this heterogeneity is very challenging. Imposing a globally agreed upon structure on all information is not very promising. First, it does not reflect the very fine grained, partially quickly emerging (and vanishing) topics and resources that make up the long tail. Second, there are often different, diverging views on the same topic, that cannot be captured by a global structure. The other extreme, namely leaving the structuring to individual users and small user groups, is also not very promising. It leads to a high work load and possibly to poorly annotated resources. Moreover, it may cause that unnecessarily different names for the same underlying concept are introduced. The paradigm of collaborative structuring allows to combine the best of both worlds. By finding patterns in user annotations, it allows to recommend not only resources but structure on these resources. This enables users to position themselves between two extremes. Either, they can simply use the annotations and structures recommended by

the system, or they can create a new structure completely from scratch. New concepts are then only introduced if the existing concepts or concept structures do not fit to a given domain or to the preferences of the current user. In contrast to other approaches, such as ontology matching, collaborative structuring does not require a one-to-one matching among concepts but is able to recombine existing concepts into new ones dynamically.

Based on this analysis and aims, the given work developed for the first time several strategies and algorithms for collaborative structuring that go byeond what current methods are able to achieve. These algorithms are based on the fact that most structuring operations can be mapped to the well-known data mining tasks of classification and clustering. The accuracy of clustering and classification (especially when applied to multi-media data) strongly depends on the representation of the underlying data. In this way, collaborative structuring could be mapped to a more general problem, namely the problem of finding adequate representations, not for a single, but for a set of data mining tasks. This problem, labeled distributed feature extraction, is not limited to information structuring but appears in many different domains ranging from robotics to business intelligence. In this work, this problem was analyzed from a combinatorial point of view. Given a (possibly infinite) set of features, select for each task in a set of tasks a subset of these features in such a way that the overall accuracy is optimized. Additionally, the union of all feature sets should have a minimal size. It was shown that constraints on the order in which the data mining tasks are solved have an influence on whether it can be guaranteed that an optimal result can be found. In fact, it turned out that no algorithm can guarantee to find an optimal solution if there is an order constraint between any two tasks. Still, a heuristic algorithm, called prioritized forward selection, was proposed, together with several optimizations. This algorithm helps to find minimal global feature sets and can speed up the process of feature extraction considerably. As most problems that include several data mining tasks appear in distributed environments, a p2p implementation of the approach was presented, that allows to apply the algorithms efficiently in networks with only minimal assumption on the network capabilities.

Distributed feature extraction provides the theoretical and practical base for collaborative structuring methods. Exploiting social annotations can be interpreted as a sophisticated feature extraction method and can be integrated into the general distributed feature extraction framework. The key is to regard user annotations, applied in many social bookmarking systems, as features. Based on these collaborative features, classification and clustering algorithms can be applied. It was shown that applying traditional clustering algorithms to user annotations efficiently yields hierarchical cluster ensembles. While the proposed methods outperform methods that apply clustering only to content based features, the result is still not fully satisfying from a user perspective. The reason for this is that traditional clustering algorithms return only a single solution and that cluster ensembles are not local. This led to a reformulation of the clustering problem and to the development of the localized, alternative cluster ensembles approach.

The proposed methods are inspired and applied in two application areas. The first one are heterogeneous expert communities. Collaborative structuring enables experts from different domains to cooperate and gain a mutual understanding of the other domains.

This is achieved by using an additional feature selection model, based on community membership. Users can either receive recommendations from other members of their own community or from members of other communities. This can be implemented easily as a filtering step for collaborative features. It allows to switch between a more restrictive personalization mode and an explorative mode of knowledge sharing in heterogeneous domains.

The second application area is distributed media organization. This application area is very interesting, as media organization naturally integrates many heterogeneous areas of interest, preferences and backgrounds. Also, processing multi-media by data mining methods is very challenging. Distributed feature extraction can help to speed-up the process of finding an adequate representation for multi-media items for the application of data mining. Collaborative structuring allows users to collectively organize their media items in a very loosely-coupled way. The proposed methods can be applied, for instance, in any p2p or ad hoc network. They do not require users to be connected to a global network.

Collaborative information organization is a very promising but also challenging approach. In order to make this approach applicable in large scale information systems, data mining methods are needed. Collaborative structuring tries to bridge the gap between current social media systems and data mining methods in a way that respects the natural diversity in such systems.

Acknowledgements and Joint Work

Some parts of this work are adapted versions of joint publications. In the following, the individual contributions are described in detail.

Chapter 4

The concept of distributed feature extraction was developed in cooperation with Katharina Morik [197, 196]. The generalization of feature relevance and corresponding definitions and proofs are sole work of the author, published in [195].The approach to query for features by base weights (paragraph 4.4.3) is joint work with Ingo Mierswa [121, 120]. The weighting axioms and the corresponding proofs are sole work of Ingo Mierswa. The distance axioms and all corresponding proofs are sole work of the author. The remainder of the paragraph was developed by the author and Ingo Mierswa to equal parts. Multi-objective feature selection and construction for clustering (as mentioned and referenced in Section 4.2.2 and published in [122, 123]) is joint work with Ingo Mierswa. All other parts of chapter 4 are sole work of the author, if not indicated otherwise in the text.

Chapter 5 and Chapter 6

The requirements for representation mechanisms (Section 5.2) are part of a common publication with Andreas Kaspari, Oliver Flasch and Katharina Morik [57]. The list of requirements is however sole work of the author. Feature aggregation (as mentioned in section 5.4.5) is part of a common publication with Ingo Mierswa [123]. The LACE approach (Section 6.4) is joint work with Ingo Mierswa and Katharina Morik [198]. The (distributed) LACE algorithm, its implementation and evaluation are work of the author. All other parts of chapter 5 and chapter 6 are sole work of the author, if not indicated otherwise in the text.

Chapter 7

The work described in chapter 7 was developed partially in the project AWAKE, funded by the BMBF 2001-2003. The conceptual model of the Knowledge Explorer (section 7.5) and the related research (sections 7.3, 7.2) are joint work with Jasminko Novak. The data mining architecture and the data mining methods, as well as their implementation (Sections 7.4, 7.6) are the sole work of the author (with exception to the SOM clustering). The content of this chapter is based on several common publications: [131, 132, 133, 134, 200, 199].

Chapter 8

The Nemoz system was initially developed in the course of a student project supervised by Katharina Morik and the author. The data mining methods described in this chapter (Section 8.4.1) are based on the methods described in part two of this work. Section 8.2 is the sole work of the author. The Nemoz interaction model and system architecture (Sections 8.3, 8.4.2) were developed in cooperation with Katharina Morik, Oliver Flasch, Andreas Kaspari [57] and the other members of student project Nemoz (Metin Aksoy, Dominique Burgard, Matthias Lüttgens, Maxim Martens, Bülent Möller, Umut Öztürk and Philip Thome) [3].

Bibliography

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD Record*, 27(2):94–105, 1998.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM International Conference on Management of Data*, 1993.

[3] M. Aksoy, D. Burgard, O. Flasch, A. Kaspari, M. Lüttgens, M. Mertens., B. Möller, U. Öztürk, and P. Thome. Kollaboratives Strukturieren von Multimediadaten für Peer-to-Peer Netze. Technical report, Fachbereich Informatik, Universität Dortmund, 2005. Endbericht der Projektgruppe 461.

[4] R. Albert and A.-L. Barabasi. Topology of evolving networks: local events and universality. *Physical Review Letters*, 85(24):5234–5237, 2000.

[5] M. R. Anderberg. *Cluster Analysis for Applications*. Academic Press, 1973.

[6] C. Anderson. The long tail. *Wired*, 12(10), 2004.

[7] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. Seti@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[8] R. K. Ando and T. Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853, 2005.

[9] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: Ordering points to identify the clustering structure. In *Proceedings of the ACM International Conference on Management of Data*, 1999.

[10] A. Argyriou, T. Evgeniou, and M. Pontil. Multi-task feature learning. In *Advances in Neural Information Processing Systems*, 2007.

[11] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, 2003.

[12] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with bregman divergences. *JMLR*, 6:1705–1749, 2005.

[13] J. Bao and V. Honavar. Collaborative ontology building with Wiki@nt - a multi-agent based ontology building environment. In *Proceedings of the International Workshop on Evaluation of Ontologybased Tools*, 2004.

[14] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[15] S. Baumann and O. Hummel. Using cultural metadata for artist recommendations. In *Proceedings of the International Conference on WEB Delivering of Music*, 2003.

[16] F. Beil, M. Ester, and X. Xu. Frequent term-based text clustering. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2002.

[17] R. Bekkerman, R. El-Yaniv, and A. McCallum. Multi-way distributional clustering via pairwise interactions. In *Proceedings of the International Conference on Machine Learning*, 2005.

[18] A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik. Support vector clustering. *Journal of Machine Learning Research*, 2:125–137, 2001.

[19] R. Bentley, T. Horstmann, and J. Trevor. The World Wide Web as enabling technology for CSCW: The case of BSCW. *Computer-Supported Cooperative Work: Special issue on CSCW and the Web*, 6:111–134, 1997.

[20] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web : A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, 284(5):34–43, 2001.

[21] S. Bickel and T. Scheffer. Multi-view clustering. In *Proceedings of the International Conference on Data Mining*, 2004.

[22] P. Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3):217–239, 2005.

[23] J. R. Boland and R. V. Tenkasi. Perspective making and perspective taking in communities of knowing. *Organization Science*, 6:350–372, 1995.

[24] M. Bonifacio, A. Dona, G. Mameli, and M. Nori. A technological solution for distributed knowledge management. In *Proceedings of the I-Know Conference*, 2004.

[25] G. Brewka. *Nonmonotonic Reasoning: Logical Foundations of Commonsense*. Cambridge University Press, 1991.

[26] R. A. Brooks. Intelligence without reason. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1991.

[27] J. Buckley and E. Eslami. *An Introduction to Fuzzy Logic and Fuzzy Sets*. Springer, 2002.

[28] I. Cantador, M. Fernandez, and P. Castells. A collaborative recommendation framework for ontology evaluation and reuse. In *Proceedings of the International ECAI Workshop on Recommender Systems*, 2006.

[29] F. Cappello, S. Djilali, G. Fedak, T. Herault, F. Magniette, V. Néri, and O. Lodygensky. Computing on large-scale distributed systems: Xtremweb architecture, programming models, security, tests and convergence with grid. *Future Generation Computer Systems*, 21(3):417–437, 2005.

[30] R. Caruana. Multitask learning: A knowledge-based source of inductive bias. In *International Conference on Machine Learning*, pages 41–48, 1993.

[31] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[32] H. Chalupsky. Ontomorph: a translation system for symbolic knowledge. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning*, 2000.

[33] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing*, 33(6):1417–1440, 2004.

[34] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker. Making gnutella-like p2p systems scalable. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2003.

[35] N. Choi, I.-Y. Song, and H. Han. A survey on ontology mapping. *SIGMOD Records*, 35(3):34–41, 2006.

[36] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of International Conference on Very Large Data Bases*, 1997.

[37] P. Cimiano, A. Hotho, and S. Staab. Learning concept hierarchies from text corpora using formal concept analysis. *Journal of Artificial Intelligence Research*, 24:305–339, 2005.

[38] R. Cobos and X. Alaman. KnowCat: A knowledge crystallisation tool. In *Proceedings of the eBusiness and eWork Conference*, 2000.

[39] D. Cohn, R. Caruana, and A. McCallum. Semi-supervised clustering with user feedback. Technical Report TR2003-1892, Cornell University, 2000.

[40] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/gather:a cluster-based approach to browsing large document collections. In *Proceedings of the International ACM SIGIR Conference*, 1992.

[41] S. Datta, K. Bhaduri, C. Giannella, R. Wolff, and H. Kargupta. Distributed data mining in peer-to-peer networks. *IEEE Internet Computing*, 10(4):18–26, 2006.

[42] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[43] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[44] I. S. Dhillon, Y. Guan, and B. Kulis. Kernel k-means: spectral clustering and normalized cuts. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2004.

[45] A. Diaz, G. Baldo, and G. Canals. Co-Protege: Collaborative ontology building with divergences. In *Proceedings of the International Conference on Database and Expert Systems Applications*, 2006.

[46] H.-H. Do and E. Rahm. COMA—a system for flexible combination of schema matching approaches. In *Proceedings of the International Conference on Very Large Databases*, 2002.

[47] A. Doan, P. Domingos, and A. Halevy. Learning to match the schemas of data sources: A multistrategy approach. *Machine Learning*, 50(3):279–301, 2003.

[48] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. *Handbook on Ontologies*, chapter Ontology matching: A machine learning approach, pages 385–404. 2004.

[49] J. Domingue. Tadzebao and WebOnto: Discussing, browsing, and editing ontologies on the web. In *Proceedings of the Workshop on Knowledge Acquisition for Knowledge-Based Systems*, 1998.

[50] C. Emmanouilidis, A. Hunter, and J. MacIntyre. A multiobjective evolutionary setting for feature selection and a commonality-based crossover operator. In *Proceedings of the Congress on Evolutionary Computation*, 2000.

[51] T. Erickson and W. Kellogg. Knowledge communities: Online environments for supporting knowledge management and its social context. In *Beyond Knowledge Management: Sharing Expertise*. MIT Press, 2001.

[52] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 1996.

[53] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight probabilistic broadcast. *ACM Transactions on Computer Systems*, 21(4):341–374, 2003.

[54] T. Evgeniou, C. A. Micchelli, and M. Pontil. Learning multiple tasks with kernel methods. *Journal of Machine Learning Research*, 6:615–637, 2005.

[55] A. Farquhar, R. Fikes, and J. Rice. The Ontolingua server: A tool for collaborative ontology construction. Technical report, Stanford, 1996.

[56] T. Finley and T. Joachims. Supervised clustering with support vector machines. In *Proceedings of the International Conference on Machine Learning*, 2005.

[57] O. Flasch, A. Kaspari, K. Morik, and M. Wurst. Aspect-based tagging for collaborative media organisation. In *Proceedings of the ECML/PKDD workshop on Ubiquitous Knowledge Discovery for Users*, 2006.

[58] K. F. Fogel. *Open Source Development with CVS*. Coriolis Group Books, 1999.

[59] I. Foster and C. Kesselman. The globus toolkit. In *The grid: blueprint for a new computing infrastructure*. 1999.

[60] I. T. Foster. The anatomy of the grid: Enabling scalable virtual organizations. In *Proceedings of the International Euro-Par Conference on Parallel Processing*, 2001.

[61] T. Froehner, M. Nickles, G. Weiss, W. Brauer, and R. Franken. Integration of ontologies and knowledge from distributed autonomous sources. *Kuenstliche Intelligenz*, 19(1):18–23, 2005.

[62] B. C. M. Fung, K. Wang, and M. Ester. Hierarchical document clustering using frequent items. In *Proceedings of the SIAM International Conference on Data Mining*, 2003.

[63] F. Giunchiglia, P. Shvaiko, and M. Yatskevich. S-match: an algorithm and an implementation of semantic matching. In *Proceedings of the Dagstuhl Seminar on Semantic Interoperability and Integration*, 2005.

[64] C. A. Goble and D. De Roure. The semantic grid: Myth busting and bridge building. In *Proceedings of the European Conference on Artificial Intelligence*, 2004.

[65] D. Gondek and T. Hofmann. Non-redundant data clustering. In *Proceedings of the International Conference on Data Mining*, 2004.

[66] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. In *Proceedings of International Conference on Management of Data*, 1998.

[67] G. Guo and S. Z. Li. Content-based audio classification and retrieval by support vector machines. *IEEE Transaction on Neural Networks*, 14(1):209–215, 2003.

[68] F. Guoyon, S. Dixon, E. Pampalk, and G. Widmer. Evaluating rhytmic descriptors for musical genre classification. In *Proceedings of the International AES Conference*, 2004.

[69] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh. *Feature Extraction: Foundations and Applications (Studies in Fuzziness and Soft Computing)*. Springer-Verlag New York, Inc., 2006.

[70] T. Hammond, T. Hannay, B. Lund, and J. Scott. Social bookmarking tools (i) - a general review. *D-Lib Magazine*, 11(4), 2005.

[71] E.-H. Han, G. Karypis, V. Kumar, and B. Mobasher. Clustering based on association rule hypergraphs. In *Proceedings of the Workshop on Research Issues on Data Mining and Knowledge Discovery*, 1997.

[72] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *Applied Statistics*, 28(1):100–108, 1979.

[73] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

[74] S. Hennig and M. Wurst. Incremental clustering of newsgroup articles. In *Proceedings of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, 2006.

[75] T. Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the conference on Uncertainty in Artificial Intelligence*, 1999.

[76] C. W. Holsapple and K. D. Joshi. A collaborative approach to ontology design. *Communications of the ACM*, 45(2):42–47, 2002.

[77] R. C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11(1):63–90, 1993.

[78] H. Homburg, I. Mierswa, B. Möller, K. Morik, and M. Wurst. A benchmark dataset for audio classification and clustering. In *Proceedings of the International Conference on Music Information Retrieval*, 2005.

[79] T. Honkela, S. Kaski, K. Lagus, and T. Kohonen. Exploration of full-text databases with self-organizing maps. In *Proceedings of the International Conference on Neural Networks*. 1996.

[80] A. Hotho, S. Staab, and G. Stumme. Ontologies improve text document clustering. In *Proceedings of the International Conference on Data Mining*, 2003.

[81] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 3(31):264–323, 1999.

[82] J. E. Januza, H.-P. Kriegel, and M. Pfeifle. Towards effective and efficient distributed clustering. In *ICDM Workshop on Clustering Large Data Sets*, 2003.

[83] T. Jebara. Multi-task feature and kernel selection for svms. In *Proceedings of the International Conference on Machine Learning*, 2004.

[84] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[85] T. Joachims. *Learning to Classify Text using Support Vector Machines*. Kluwer, 2002.

[86] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2002.

[87] T. Joachims, D. Freitag, and T. M. Mitchell. Web watcher: A tour guide for the world wide web. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1997.

[88] G. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the International Conference on Machine Learning*, 1994.

[89] E. L. Johnson and H. Kargupta. Collective, hierarchical clustering from distributed, heterogeneous data. In *Proceedings of the Workshop on Large-Scale Parallel KDD Systems*, 2000.

[90] S. Jones, S. J. Cunningham, and M. Jones. Organizing digital music for use: an examination of personal music collections. In *Proceedings of the International Conference on Music Information Retrieval*, 2004.

[91] Y. Kalfoglou and M. Schorlemmer. Ontology mapping: The state of the art. *The Knowledge Engineering Review*, 18(1):1–31, 2003.

[92] R. Kashef and M. Kamel. Distributed cooperative hard-fuzzy document clustering. In *Proceedings of the Annual Scientific Conference of the LORNET Research Network*, 2006.

[93] J.-U. Kietz and K. Morik. A polynomial approach to the constructive induction of structural knowledge. *Machine Learning*, 14(2):193–217, 1994.

[94] Y. Kim, W. N. Street, and F. Menczer. Feature selection in unsupervised learning via evolutionary search. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2000.

[95] Y. Kim, W. N. Street, and F. Menczer. Evolutionary model selection in unsupervised learning. *Intelligent Data Analysis*, 6:531–556, 2002.

[96] K. Kira and I. A. Rendell. The feature selection problem: Traditional methods and a new algoirthm. In *Proceedings of the National Conference on Artificial Intelligence*, 1992.

[97] K. Kira and L. A. Rendell. A practical approach to feature selection. In *Proceedings of the International Workshop on Machine Learning*, 1992.

[98] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *Proceedings of the ACM Symposium on Theory of Computing*, 2000.

[99] P. Knees, E. Pampalk, and G. Widmer. Artist classification with web-based data. In *Proceedings of the International Conference on Music Information Retrieval*, 2004.

[100] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[101] T. Kohonen. *Self-Organizing Maps*. Springer, 1995.

[102] V. Komulainen, A. Valo, and E. Hyvoenen. A tool for collaborative ontology development for the semantic web. In *Proceedings of International Conference on Dublin Core and Metadata Applications*, 2005.

[103] W. Kowalczyk, M. Jelasity, and A. E. Eiben. Towards data mining in large and fully distributed peer-to-peer overlay networks. In *Proceedings of the Belgian-Dutch Conference on Artificial Intelligence*, 2003.

[104] M. S. Lacher, W. Woerndl, M. Koch, and Brede H. Ontology mapping in community support systems. Technical report, Technische Universitaet Muenchen, Lehrstuhl Informatik XI, 2000.

[105] Y. Lashkari, M. Metral, and P. Maes. Collaborative interface agents. In *Proceedings of the National Conference on Artificial Intelligence*. 1994.

[106] A. Lazarevic and Z. Obradovic. The distributed boosting algorithm. In *International Conference on Knowledge Discovery and Data Mining*, 2001.

[107] J. Li, B. Loo, J. Hellerstein, F. Kasshoek, D. Karger, and R. Morris. On the feasibility of peer-to-peer web indexing and search. In *International Workshop on Peer-to-Peer Systems*, 2003.

[108] L. Li, D. Alderson, J. C. Doyle, and W. Willinger. Towards a theory of scale-free graphs: Definition, properties, and implications. *Internet Mathematics*, 2(4):431–523, 2005.

[109] T. Lidy and A. Rauber. Evaluation of feature extractors and psycho-acoustic transformations for music genre classification. In *Proceedings of the International Conference on Music Information Retrieval*, 2005.

[110] H. Lieberman. Letizia: An agent that assists web browsing. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.

[111] D. Lin. An information-theoretic definition of similarity. In *Proceedings of the International Conference on Machine Learning*, 1998.

[112] B. Logan. Content-based playlist generation: Exploratory experiments. In *Proceedings of the International Symposium on Music Information Retrieval*, 2002.

[113] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[114] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of the International Conference on Very Large Data Bases*, 2001.

[115] A. Maedche, B. Motik, N. Silva, and R. Volz. Mafra - an ontology mapping framework in the context of the semantic web. In *Proceedings of the ECAI Workshop on Knowledge Transformation for the Semantic Web*, 2002.

[116] R. Mayer. *Knowledge Management Systems*. Springer, 2004.

[117] M. Meila. Comparing clusterings: an axiomatic view. In *Proceedings of the International Conference on Machine Learning*, 2005.

[118] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *International Conference on Data Engineering*, 2002.

[119] I. Mierswa and K. Morik. Automatic feature extraction for classifying audio data. *Machine Learning Journal*, 58:127–149, 2005.

[120] I. Mierswa and M. Wurst. Efficient case based feature construction for heterogeneous learning tasks. In *Proceedings of the European Conference on Machine Learning*, 2005.

[121] I. Mierswa and M. Wurst. Efficient feature construction by meta learning – guiding the search in meta hypothesis space. In *Proceedings of the International Conference on Machine Learning, Workshop on Meta Learning*, 2005.

[122] I. Mierswa and M. Wurst. Information preserving multi-objective feature selection for unsupervised learning. In *Proceedings of the International Conference on Genetic and Evolutionary Computation*, 2006.

[123] I. Mierswa and M. Wurst. Sound multi-objective feature space transformation for clustering. In *Proceedings of the German Conference on Knowledge Discovery, Data Mining, and Machine Learning*, 2006.

[124] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, and T. Euler. YALE: Rapid prototyping for complex data mining tasks. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2006.

[125] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[126] F. Moerchen, A. Ultsch, M. Thies, I. Loehken, M. Noecker, C. Stamm, N. Efthymiou, and M. Kuemmerer. Musicminer: Visualizing perceptual distances of music as topograpical maps. Technical report, Department of Mathematics and Computer Science, University of Marburg, Germany, 2004.

[127] O. Morgenstern and J. Von Neumann. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[128] K. Morik. Balanced cooperative modeling. *Machine Learning*, 11:217–235, 1993.

[129] M. Morita, R. Sabourin, F. Bortolozzi, and C. Y. Suen. Unsupervised feature selection using multi-objective genetic algorithms for handwritten word recognition. In *Proceedings of the International Conference on Document Analysis and Recognition*, 2003.

[130] M. Nickles, T. Froehner, and G. Weiss. Social annotation of semantically heterogeneous knowledge. In *Proceedings of the International Workshop on Knowledge Markup and Semantic Annotation*, 2004.

[131] J. Novak and M. Wurst. Discovering, visualizing and sharing knowledge through personalized learning knowledge maps. In *Agent Mediated Knowledge Management*, 2003.

[132] J. Novak and M. Wurst. Supporting communities of practice through personalisation and collaborative structuring based on capturing implicit knowledge. In *Proceedings of the International Conference on Knowledge Management*, 2003.

[133] J. Novak and M. Wurst. Supporting knowledge creation and sharing in communities based on mapping implicit knowledge. *j-jucs*, 10(3):235–251, 2004.

[134] J. Novak and M. Wurst. Collaborative knowledge visualisation for cross-community learning. In *Knowledge and Information Visualization*. Springer, 2006.

[135] N. Noy and M. Musen. SMART: Automated support for ontology merging and alignment. In *Proceedings of the Workshop on Knowledge Acquisition, Modeling and Management*, 1999.

[136] N. Noy and H. Stuckenschmidt. Ontology alignment: An annotated bibliography. In *Proceedings of the Dagstuhl Seminar on Semantic Interoperability and Integration*, 2005.

[137] N. F. Noy, A. Chugh, W. Liu, and M. A. Musen. A framework for ontology evolution in collaborative environments. In *Proceedings of the International Semantic Web Conference*, 2006.

[138] H.-L. Ong, A.-H. Tan, J. Ng, H. Pan, and Q.-X. Li. FOCI: Flexible organizer for competitive intelligence. In *Proceedings of the International Conference on Information and Knowledge Management*, 2001.

[139] E. Pampalk, S. Dixon, and G. Widmer. Exploring music collections by browsing different views. In *Proceedings of the International Symposium on Music Information Retrieval*, 2003.

[140] E. Pampalk, T. Pohle, and G. Widmer. Dynamic playlist generation based on skipping behavior. In *Proceedings of the International Conference on Music Information Retrieval*, 2005.

[141] E. Pampalk, G. Widmer, and A. Chan. A new approach to hierarchical clustering and structuring of data with self-organizing maps. *Intelligent Data Analysis*, 8(2):131–149, 2005.

[142] S. Papert. *Introduction: Constructionist Learning.* Cambridge University Press, 1990.

[143] D. M. Pennock, G. W. Flake, S. Lawrence, E. J. Glover, and C. L. Giles. Winners don't take all: Characterizing the competition for links on the web. *Proceedings of the National Academy of Sciences*, 99(8):5207–5211, 2002.

[144] H. S. Pinto, A. Gomez-Perez, and J. P. Martins. Some issues on ontology integration. In *Proceedings of the Workshop on Ontologies and Problem Solving Methods*, 1999.

[145] T. Pohle, E. Pampalk, and G. Widmer. Evaluation of frequently used audio features for classification of music into perceptual categories. In *Proceedings of the International Workshop on Content-Based Multimedia Indexing*, 2005.

[146] A. Prodromidis and P. Chan. Meta-learning in distributed data mining systems: Issues and approaches. In *Advances of Distributed Data Mining*, 2000.

[147] R. J. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[148] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10:334–350, 2001.

[149] L. Ramaswamy and L. Liu. Free riding: A new challenge to peer-to-peer file sharing systems. In *Proceedings of the International Conference on System Sciences*, 2003.

[150] P. Resnik. Semantic similarity in a taxonomy. *Journcal of Artificial Intelligence Research*, 11:95–130, 1999.

[151] M. M. Richter. Classification and learning of similarity measures. Technical Report SR-92-18, University of Kaiserslautern, 1992.

[152] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[153] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice-Hall, 2003.

[154] G. Salton. *Automated Text Processing.* Addison-Wesley, 1989.

[155] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the International World Wide Web Conference*, 2001.

[156] J. B. Schafer, J. A. Konstan, and J. Riedl. Recommender systems in e-commerce. In *Proceedings of the ACM Conference on Electronic Commerce*, 1999.

[157] M. Schedl, P. Knees, and G. Widmer. Discovering and visualizing prototypical artists by web-based co-occurrence analysis. In *Proceedings of the International Conference on Music Information Retrieval*, 2005.

[158] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2004.

[159] B. Schölkopf and A. J. Smola. *Learning with Kernels — Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[160] U. Shardanand and P. Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of ACM Conference on Human Factors in Computing Systems*, 1995.

[161] C. Shirky. What is p2p ... and what isn't. *OpenP2P*, 2000.

[162] P. Shvaiko. A classification of schema-based matching approaches. Technical report, Informatica e Telecomunicazioni, University of Trento, 2004.

[163] R. G. Smith and A. Farquhar. The road ahead for knowledge management: An AI perspective. *AI Magazine*, 21(4):17–40, 2000.

[164] J. F. Sowa. *Conceptual structures: information processing in mind and machine*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[165] R. Srikant and R. Agrawal. Mining generalized association rules. In *International Conference on Very Large Databases*, 1995.

[166] S. L. Star. The structure of ill-structured solutions: Boundary objects and heterogeneous distributed problem solving. In *Readings in Distributed Artificial Intelligence*. Morgan Kaufman, 1989.

[167] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. In *Proceedings of the KDD Workshop on Text Mining*, 2000.

[168] R. Stenzel and T. Kamps. Improving content-based similarity measures by training a collaborative model. In *Proceedings of the International Conference on Music Information Retrieval*, 2005.

[169] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2001.

[170] S. J. Stolfo, A. L. Prodromidis, S. Tselepis, W. Lee, D. W. Fan, and P. K. Chan. JAM: Java agents for meta-learning over distributed databases. In *Proceedings of the Internation Conference on Knowledge Discovery and Data Mining*, 1997.

[171] A. Strehl and J. Ghosh. Cluster ensembles – a knowledge reuse framework for combining partitionings. In *Proceedings of the Conference on Artificial Intelligence*, 2002.

[172] H. Stuckenschmidt, F. van Harmelen, P. Bouquet, F. Giunchiglia, and L. Serafini. Using C-OWL for the alignment and merging of medical ontologies. In *Proceedings of the International Workshop on Formal Biomedical Knowledge Representation*, 2004.

[173] G. Stumme and A. Maedche. FCA-merge: Bottom-up merging of ontologies. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2001.

[174] Y. Sure, J. Angele, and S. Staab. Ontoedit: Guiding ontology development by methodology and inferencing. In *International Conference on Ontologies, Databases and Applications of Semantics for Large Scale Information Systems*, 2002.

[175] B. Swartout, R. Patil, K. Knight, and T. Russ. Toward distributed use of large-scale ontologies. In *Proceedings of the Workshop on Knowledge Acquisition*, 1996.

[176] A. S. Tanenbaum. *Distributed Operating Systems.* Prentice Hall, 1995.

[177] A. S. Tanenbaum. *Computer Networks, Fourth Edition.* Prentice Hall, 2002.

[178] I. J. Taylor. *From P2P to Web Services and Grids — Peers in a Client/Server World.* Springer, 2005.

[179] J. Tennison and N. R. Shadbolt. Apecks: a tool to support living ontologies. In *Proceedings of the Workshop on Knowledge Acquisition, Modeling and Management*, 1998.

[180] S. Thrun. Winning the DARPA grand challenge: A robot race through the mojave desert. In *Proceedings of the International Conference on Automated Software Engineering*, 2006.

[181] S. Thrun and J. O'Sullivan. Discovering structure in multiple learning tasks: The TC algorithm. In *Proceedings of the International Conference on Machine Learning*, 1996.

[182] A. P. Topchy, A. K. Jain, and W. F. Punch. Combining multiple weak clusterings. In *Proceedings of the International Conference on Data Mining*, 2003.

[183] S. Turkle. *Life on the Screen: Identity in the Age of the Internet.* Simon and Schuster, 1995.

[184] G. Tzanetakis. *Manipulation, Analysis and Retrieval Systems for Audio Signals.* PhD thesis, Computer Science Department, Princeton University, 2002.

[185] A. Ultsch and F. Moerchen. ESOM-maps: tools for clustering, visualization, and classification with emergent SOM. Technical Report 46, Deptartment of Mathematics and Computer Science, University of Marburg, Germany, 2005.

[186] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained k-means clustering with background knowledge. In *Proceedings of the International Conference on Machine Learning*, 2001.

[187] K. Wang, C. Xu, and B. Liu. Clustering transactions using large items. In *Proceedings of the International Conceference on Information and Knowledge Management*, 1999.

[188] W. Wang, J. Yang, and R. R. Muntz. STING: A statistical information grid approach to spatial data mining. In *Proceedings of the International Conference on Very Large Data Bases*, 1997.

[189] C. Weihs, G. Szepannek, U. Ligges, K. Luebke, and N. Raabe. Local models in register classification by timbre. In *Data Science and Classification*, 2006.

[190] G. Weiß, editor. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

[191] D. R. Wilson and T. R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

[192] D. R. Wilson and T. R. Martinez. An integrated instance-based learning algorithm. *Computational Intelligence*, 16(1):1–28, 2000.

[193] R. Wolff and A. Schuster. Association rule mining in peer-to-peer systems. In *International Conference on Data Mining*, 2003.

[194] W. A. Woods. What's in a link: foundations for semantic networks. In *Representation and Understanding*. 1975.

[195] M. Wurst. Multi-agent learning by distributed feature extraction. In *Adaptive and Learning Agents and Multi-Agent Systems III*, 2008 (to appear).

[196] M. Wurst and K. Morik. Multi-agent learning by feature sharing. In *Proceedings of the European Symposium on Adaptive Learning Agents and MAS*, 2006.

[197] M. Wurst and K. Morik. Distributed feature extraction in a p2p setting - a case study. *Future Generation Computer Systems, Special Issue on Data Mining*, 23(1):69–75, 2007.

[198] M. Wurst, K. Morik, and I. Mierswa. Localized alternative cluster ensembles for collaborative structuring. In *Proceedings of the European Conference on Machine Learning*, 2006.

[199] M. Wurst and J. Novak. Knowledge sharing im heterogeneous expert communities based on personal taxonomies. In *Proceedings of the ECAI Workshop on Agent Mediated Knowledge Management*, 2004.

[200] M. Wurst, J. Novak, and M. Schneider. Integrating different machine learning methods to support search in cross-domain information sources - the project Awake. In *Proceedings of the FGML Workshop*, 2002.

[201] M. Wurst and M. Scholz. Distributed subgroup discovery. In *Proceedings of the European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2006.

[202] E. Xing, A. Ng, M. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2002.

[203] K. Yu, V. Tresp, and A. Schwaighofer. Learning gaussian processes from multiple tasks. In *Proceedings of the International Conference on Machine Learning*, 2005.

[204] L. Yu and H. Liu. Efficient feature selection via analysis of relevance and redundancy. *Journal of Machine Learning Research*, 5:1205–1224, 2004.

[205] S. Yu, V. Tresp, and K. Yu. Robust multi-task learning with t-processes. In *Proceedings of the International Conference on Machine Learning*, 2007.

[206] M. J. Zaki. Parallel and distributed association mining: A survey. *IEEE Concurrency*, 7(4):14–25, 1999.

[207] R. Zhang and A. Rudnicky. A large scale clustering scheme for kernel k-means. In *Proceedings of the International Conference on Pattern Recognition*, 2002.

[208] T. Zhang and C. Kuo. Content-based classification and retrieval of audio. In *Proceedings of the Conference on Advanced Signal Processing Algorithms, Architectures, and Implementations*, 1998.