PrefWork - a framework for the user preference learning methods testing[⋆]

Alan Eckhardt[1,2]

[1] Department of Software Engineering, Charles University,
[2] Institute of Computer Science, Czech Academy of Science,
Prague, Czech Republic
`eckhardt@ksi.mff.cuni.cz`

Abstract. *PrefWork is a framework for testing of methods of induction of user preferences. PrefWork is thoroughly described in this paper. A reader willing to use PrefWork finds here all necessary information - sample code, configuration files and results of the testing are presented in the paper. Related approaches for data mining testing are compared to our approach. There is no software available specially for testing of methods for preference learning to our best knowledge.*

1 Introduction

User preference learning is a task that allows many different approaches. There are some specific issues that differentiate this task from a usual task of data mining. User preferences are different from measurements of a physical phenomenon or a demographic information about a country; they are much more focused on the objects of interest and involve psychology or economy.

When we want to choose the right method for user preference learning, e.g. for an e-shop, the best way is to evaluate all possible methods and to choose the best one. The problems with the testing of methods for preference learning are:

– how to evaluate these methods automatically,
– how to cope with different sources of data, with different types of attributes,
– how to measure the suitability of a method,
– to personalise the recommendation for every user individually.

2 Related work

The most popular tool related to PrefWork is the open source projec t Weka [1]. Weka is in development for many years and has achieved to become the most widely used tool for data mining. It offers many classificators, regression methods, clustering, data preprocessing, etc. However this variability is also its weakness - it can be used for any given task, but it has to be customised, the developer has to choose from a very wide range of possibilities. For our case, Weka is too strong.

RapidMiner [2] has a nice user interface and is in a way similar to Weka. It is also written in Java and has source codes available. However the ease of use is not better than that of Weka. The user interface is nicer than in Weka but the layout of Weka is more intuitive (allowing to connect various components that are represented on a plane).

R [3] is a statistical software that is based on its own programming language. This is the biggest inconvenience - a user willing to use R has to learn yet another programming language.

There are also commercial tools as SAS miner [4], SPSS Clementine [5], etc. We do not consider these, because of the need to buy a (very expensive) licence.

We must also mention the work of T. Horváth - Winston [6], which was developed recently. Winston may suit our needs, because it is light-weighted, has also a nice user interface, but in the current stage there are few methods and no support for the method testing. It is more a tool for the data mining lecturing than the real world method testing.

We are working with ratings the user has associated to some items. This use-case is well-known and used across the internet. An inspiration for extending our framework is many other approaches to user preference elicitation. An alternative to ratings has been proposed in [7, 8] - instead of ratings, the system requires direct feedback from the user about the attribute values. The user has to specify in which values the given recommendation can be improved. This approach is called critique based recommendations.

Among other approaches, we should mention also work of Kiessling [9], which uses the user behaviour as the source for the preference learning.

We also need some implementations of algorithms of the user preference learning that are publicly available for being able to compare various methods among themselves. This is a strength of PrefWork - any existing method, which works with ratings, can be

[⋆] The work on this paper was supported by Czech projects MSM 0021620838, 1ET 100300517 and GACR 201/09/H057.

integrated into PrefWork using a special adaptor for each tool (see Section 4.3). There is a little bit old implementation of collaborative filtering Cofi [10] and a brand new one (released 7.4.2009) Mahout [11], developed by Apache Lucene project. Cofi uses Taste framework [12], which became a part of Mahout. The expectations are that Taste in Mahout would perform better than Cofi, so we will try to migrate our PrefWork adaptor for Cofi to Mahout. Finally there is IGAP [13] - a tool for learning of fuzzy logic programs in form of rules, which correspond to user preferences. Unfortunately, IGAP is not yet available publicly for download.

We did not find any other mining algorithm specialised on user preferences available for free download, but we often use already mentioned Weka. It is a powerful tool that can be more or less easily integrated into our framework and provide a reasonable comparison of a non-specialised data mining algorithm to other methods that are specialised for preference learning.

3 User model

For making this article self-contained, we describe in brief our user model, as in [14]. In this section, we describe our user model. This model is based on a scoring function that assigns the score to every object. User rating of an object is a fuzzy subset of X(set of all objects), i.e. a function $R(o) : X \rightarrow [0, 1]$, where 0 means the least preferred and 1 means the most preferred object. Our scoring function is divided into two steps.

Local preferences In the first step, which we call local preferences, all attribute values of object o are normalised using fuzzy sets $f_i : D_{A_i} \rightarrow [0, 1]$. These fuzzy sets are also called objectives or preferences over attributes. With this transformation, the original space of objects' attributes $X = \prod_{i=1}^{N} D_{A_i}$ is transformed into $X' = [0, 1]^N$. Moreover, we know that the object $o \in X'$ with transformed attribute values equal to $[1, \ldots, 1]$ is the most preferred object. It probably does not exist in the real world, though. On the other side, the object with values $[0, \ldots, 0]$ is the least preferred, which is more probable to be found in reality.

Global preferences In the second step, called global preferences, the normalised attribute values are aggregated into the overall score of the object using an aggregation function $@ : [0, 1]^N \rightarrow [0, 1]$. Aggregation function is also often called utility function.

Aggregation function may have different forms; one of the most common is a weighted average, as in the following formula:

$$@(o) = (2 * f_{Price}(o) + 1 * f_{Display}(o) + 3 * f_{HDD}(o) + 1 * f_{RAM}(o))/7\,,$$

where f_A is the fuzzy set for the normalisation of attribute A.

Another totally different approach was proposed in [15]. It uses the training dataset as partitioning of normalised space X'. For example, if we have an object with normalised values $[0.4, 0.2, 0.5]$ with rating 3, any object with better attribute values (e.g. $[0.5, 0.4, 0.7]$) is supposed to have the rating at least 3. In this way, we can find the highest lower bound on any object with unknown rating. In [15] was also proposed a method for interpolation of ratings between the objects with known ratings and even using the ideal (non-existent) virtual object with normalised values $[1, ..., 1]$ with rating 6.

4 PrefWork

Our tool PrefWork was initially developed as a master thesis of Tomáš Dvořák [16], who has implemented it in Python. In this initial implementation, only Id3 decision trees and collaborative filtering was implemented. For better ease of use and also for the possibility of integrating other methods, PrefWork was later rewritten to Java by the author. Many more possibilities were added until the today state. In the following sections, components of PrefWork are described.

Most of the components can be configured by XML configurations. Samples of these configurations and Java interfaces will be provided for each component. We omit methods for configuration from Java interfaces such as `configTest(configuration,section)` which is configured using a configuration from a section in an XML file. Also data types of function arguments are omitted for brevity.

4.1 The workflow

In this section a sample of workflow with PrefWork is described.

The structure of PrefWork is in Figure 1. There are four different configuration files - one for database access configuration (confDbs), one for datasources (confDatasources), one for methods (confMethods) and finally one for PrefWork runs (confRuns). A run consists of three components - a set of methods, a set of datasets and a set of ways to test the method. Every method is tested on every dataset using every way to

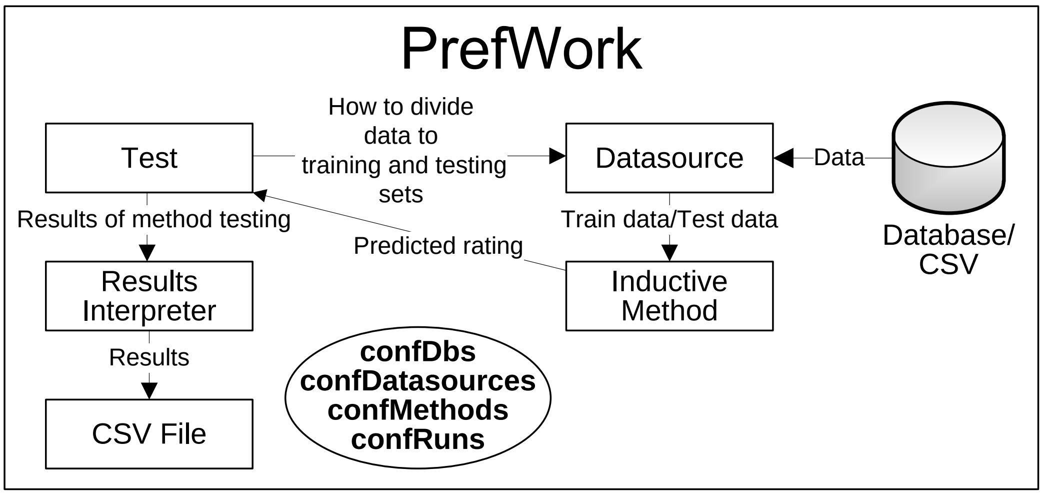

Fig. 1. PrefWork structure.

test. For each case, results of the testing are written into a csv file.

A typical situation a researcher working with PrefWork finds himself in is: "I have a new idea X. I am really interested, how it performs on that dataset Y."

The first thing is to create corresponding Java class X that implements interface InductiveMethod (see 4.3) and add a section X to confMethods.xml. Then copy an existing entry defining a run (e.g. IFSA, see 4.5) and add method X to section `methods`. Run ConfigurationParser and correct all errors in the new class (and there will be some, for sure). After the run has finished correctly, process the csv file with results to see how X performed in comparison with other methods.

A similar case is when introducing a new dataset into PrefWork - confDatasets.xml and confDBs.xml have to be edited if the data are in SQL database or in a csv file. Otherwise a new Java class (see 4.2) able to handle the new type of data has to be created. For example, we still have not implemented the class for handling of arff files - these files have the definition of attributes in themselves, so the configuration in confDatasets.xml would be much more simple (see Section 4.2 for an example of a configuration of a datasource with its attributes).

4.2 Datasource

Datasource is, as the name hints, the source of data for inductive methods. Currently, we are working only with ratings of objects. Data are vectors, where the first three attributes typically are: the user id, the object id and the rating of the object. The attributes of the object follow. There is a special column that contains a random number associated to each rating. Its purpose will described later.

Every datasource has to implement the following methods:

```
interface BasicDataSource{
  boolean hasNextRecord();
  void setFixedUserId(value);
  List<Object> getRecord();
  Attribute[] getAttributes();
  Integer getUserId();
  void setLimit(from, to,
        recordsFromRange);
  void restart();
  void restartUserId();
}
```

There are two main attributes of datasource - a list of all users and a list of ratings of the current user. `getUserId` returns the id of the current user. The most important function is `getRecord`, which returns a vector containing the rating of the object and its attributes. Following calls of `getRecords` return all objects rated by the current user. A typical sequence is:

```
int userId = data.getUserId();
data.setFixedUserId(userId);
data.restart();
while(data.hasNextRecord()){
  List<Object> record =
                data.getRecord();

  // Work with the record
  ...
}
```

Another important function is `setLimit`, which limits the data using given boundaries `from` and `to`. The random number associated to each vector returned by `getRecord` has to fit into this interval. If `recordsFromRange` is false, then the random number should be outside of the given interval on the contrary. This method is used when dividing the data into training and testing sets. For example, let us divide the data to 80% training set and 20% testing set. First, we call `setLimit(0.0,0.8,true)` and let the method train on these data. Then, `setLimit( 0.0,0.8,false)` is executed and vectors returned by the datasource are used for the testing of the method.

Let us show a sample configuration of a datasource that returns data about notebooks:

```
<NotebooksIFSA>
  <attributes>
   <attribute><name>userid</name>
     <type>numerical</type>
   </attribute>
   <attribute><name>notebookid</name>
     <type>numerical</type>
   </attribute>
   <attribute><name>rating</name>
     <type>numerical</type>
    </attribute>
   <attribute><name>price</name>
     <type>numerical</type>
   </attribute>
   <attribute><name>producer</name>
     <type>nominal</type>
   </attribute>
   <attribute><name>ram</name>
     <type>numerical</type>
   </attribute>
   <attribute><name>hdd</name>
     <type>numerical</type>
   </attribute>
  </attributes>
  <recordsTable>
    note_ifsa
  </recordsTable>
  <randomColumn>
    randomize
  </randomColumn>
  <userID>userid</userID>
  <usersSelect>
  select distinct userid from note_ifsa
  </usersSelect>
</NotebooksIFSA>
```

First, a set of attributes is defined. Every attribute has a name and a type - numerical, nominal or list. An example of list attribute is actors in a film. This attribute can be found in the IMDb dataset [17].

Let us also note the select for obtaining the user ids (section `usersSelect`) and the name of the column that contains the random number used in setLimit (`randomColumn`).

Other types of user preferences. PrefWork as it is now supports only ratings of objects. There are many more types of data containing user preferences - user clickstream, user profile, filtering of the result set etc.

PrefWork does not work with any information about the user, either demographic like age, sex, place of birth, occupation etc. or his behaviour. These types of information may bring a large improvement in the prediction accuracy, but they are typically not present - users do not want to share any personal information for the sole purpose of a better recommendation. Another issue is the complexity of user information; a semantic processing would have to be used.

4.3 Inductive method

InductiveMethod is the most important interface - it is what we want to evaluate. Inductive method has two main methods:

```
interface InductiveMethod {
  int buildModel(trainingDataset,
        userId);
  Double classifyRecord(record,
        targetAttribute);
}
```

`buildModel` uses the training dataset and the userId for the construction of a user preference model. After having it constructed, the method is tested - it is being given records via method `classifyRecord` and is supposed to evaluate them.

Various inductive methods were implemented. Among the most interesting are our method Statistical ([18, 15]) and Instances ([15]), WekaBridge that allows to use any method from Weka (such as Support vector machine) and ILPBridge that transforms data to a prolog program and then uses Progol [19] to create the user model. CofiBridge allows to use Cofi as a PrefWork InductiveMethod.

A sample configuration of method Statistical is:

```
<Statistical>
  <class>Statistical</class>
  <rater>
   <class>WeightAverage</class>
   <weights>VARIANCE</weights>
  </rater>
  <representant>
   <class>AvgRepresentant</class>
  </representant>
  <numericalNormalizer>
       Linear
  </numericalNormalizer>
  <nominalNormalizer>
       RepresentantNormalizer
  </nominalNormalizer>
  <listNormalizer>
       ListNormalizer
  </listNormalizer>
</Statistical>
```

Every method requires a different configuration, only the name of the class is obligatory. Note that the methods based on our two-step user model (Statistical and Instances for now) can be easily configured to test different heuristics for the processing of different types of attributes. Configuration contains three sections: `numericalNormalizer`, `nominalNormalizer` and `listNormalizer` for the specification of the method for the particular type of attribute. Also see Section 4.5 for an example of this configuration.

4.4 Ways of the testing of the method

Several possible ways for the testing of methods can be defined, the division to training and testing sets is the most typically used. The method is trained on the training set (using buildModel) and then tested on the testing set (using classifyRecord). Another typical method is k-fold cross validation that divides data into k sets. In each of k runs, one set is used as the testing set and the rest as the training set.

```
interface Test {
  void test(method, trainDataSource,
         testDataource);
}
```

When the method is tested, the results in the form userid, objectid, predictedRating, realUserRating have to be processed. The interpretation is done by a TestResultsInterpreter. The most common is DataMiningStatistics, which computes such measures as correlation, RMSE, weighted RMSE, MAE, Kendall rank tau coefficient, etc. Others are still waiting to be implemented - ROC curves or precision-recall statistics.

```
abstract class TestInterpreter {
  abstract void writeTestResults(
                     testResults);
}
```

4.5 Configuration parser

The main class is called ConfigurationParser. The definition of one test follows:

```
<IFSA>
  <methods>
   <method>
     <name>Statistical</name>
     <numericalNormalizer>
       Standard2CPNormalizer
     </numericalNormalizer>
   </method>
   <method><name>Statistical</name>
   </method>
   <method><name>Mean</name></method>
   <method><name>SVM</name></method>
  </methods>
  <dbs>
   <db>
    <name>MySQL</name>
    <datasources>NotebooksIFSA
    </datasources>
   </db>
  </dbs>
  <tests>
   <test>
    <class>TestTrain</class>
    <ratio>0.05</ratio>
    <path>resultsIFSA</path>
    <testInterpreter>
      <class>DataMiningStatistics
      </class>
    </testInterpreter>
   </test>
   <test>
    <class>TestTrain</class>
    <ratio>0.1</ratio>
    <path>resultsIFSA</path>
    <testInterpreter>
      <class>DataMiningStatistics
      </class>
    </testInterpreter>
   </test>
  </tests>
</IFSA>
```

First, we have specified which methods are to be tested - in our case it is two variants of Statistical, then Mean and SVM. Note that some attributes of Statistical, which was defined in confMethods, can be "overridden" here. The basic configuration of Statistical is in Section 4.3. Then the datasource for testing of the methods is specified – we are using MySql database with datasource NotebooksIFSA. Several datasources or databases can be specified here. Finally, the ways of the testing and interpretation are given in section `tests`. TestTrain requires ratio of the training and the testing sets, the path where the results are to be written, and the interpretation of the test results.

```
date;Ratio;dataset;method;userId;mae;rmse;weightedRmse;monotonicity;tau;weightedTau;correlation;buildTime;
testTime;countTrain;countTest;countUnableToPredict
28.4.2009
12:18;0,05;NotebooksIFSA;Statistical,StandardNorm2CP;1;0,855;0,081;1,323;1,442;0,443;0,358;0,535;94;47;10;188;0;
28.4.2009
12:18;0,05;NotebooksIFSA;Statistical,StandardNorm2CP;1;0,868;0,078;1,216;1,456;0,323;0,138;0,501;32;0;13;185;0;
28.4.2009
12:18;0,05;NotebooksIFSA;Statistical,StandardNorm2CP;1;0,934;0,083;1,058;1,873;0,067;0,404;0,128;31;16;12;186;0;
28.4.2009 12:31;0,025;NotebooksIFSA;Statistical,Peak;1;0,946;0,081;1,161;1,750;0,124;0,016;0,074;15;16;4;194;0
28.4.2009 12:31;0,025;NotebooksIFSA;Statistical,Peak;1;0,844;0,076;1,218;1,591;0,224;0,215;0,433;0;16;6;192;0
28.4.2009 12:31;0,025;NotebooksIFSA;Statistical,Peak;1;1,426;0,123;1,407;1,886;0,024;0,208;-0,063;16;0;4;194;0
```

Fig. 2. A sample of results in a csv file.

The definitions of runs are in confRuns.xml in section `runs`. The specification of the run to be executed is in section `run` of the same file.

4.6 Results of testing

In Figure 2 is a sample of the resulting csv file. In our example, there are three runs with method Statistical with normaliser StandardNorm2CP and three runs with normaliser Peak. Runs were performed on different settings of the training and the testing sets, so the results are different even for the same method.

The results contain all necessary information required for generation of a graph or a table with the results. Csv format was chosen for its simplicity and wide acceptance, so any other possible software can handle it. We are currently using Microsoft Excel and its Pivot table that allows aggregation of results by different criteria. Among other possibilities is also the already mentioned R [3].

Example figures of the output of PrefWork are in Figures 3 and 4. The lines represent different methods, X axis represents the size of the training set and the Y axis the value of the error function. In Figure 3 the error function is Kendall rank tau coefficient (the higher it is the better) and in Figure 4 is RMSE weighted by the original rating (the lower the better). The error function can be chosen, as is described in Section 4.4.

It is impossible to compare PrefWork to another framework generally. A simple comparison to other such systems is in Section 2. This can be done only qualitatively; there is no attribute of frameworks that can be quantified. The user itself has to choose among them the one that suits his needs the most.

4.7 External dependencies

PrefWork is dependent on some external libraries. Two of them are sources for inductive methods - Weka [1] and Cofi [10]. Cofi also requires `taste.jar`.

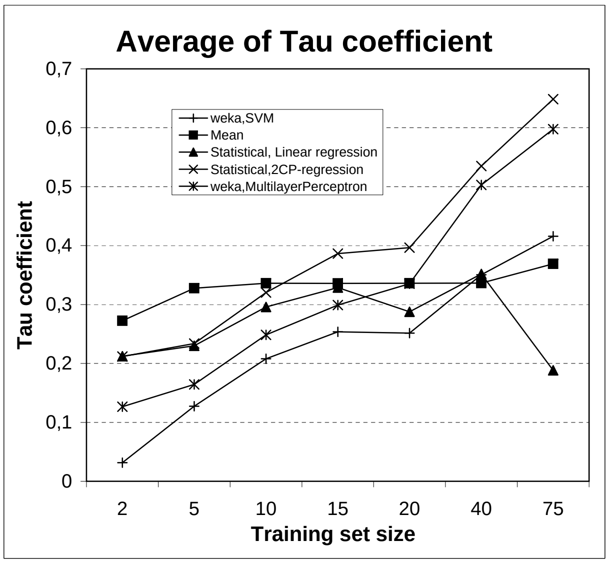

Fig. 3. Tau coefficient.

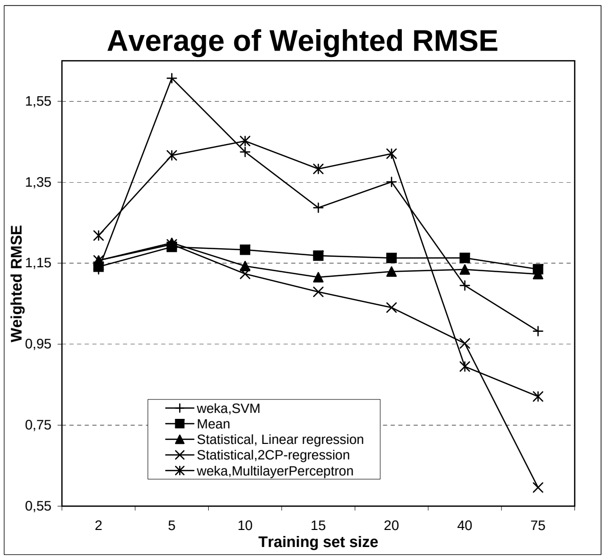

Fig. 4. Weighted RMSE.

PrefWork requires following jars to function correctly:

Weka	weka.jar
Cofi	cofi.jar
Cofi	taste.jar
Logging	log4j.jar
CSV parsing	opencsv-1.8.jar
Configuration	commons-configuration-1.5.jar
Configuration	commons-lang-2.4.jar
MySql	mysql-connector-java-5.1.5-bin.jar
Oracle	ojdbc1410.2.0.3.jar

Tab. 1. Libraries required by PrefWork.

5 Conclusion

PrefWork has been presented in this paper with a thorough explanation and description of every component. Interested reader should be now able to install PrefWork, run it, and implement a new inductive method or a new datasource.

The software can be downloaded at `http://www.ksi.mff.cuni.cz/~eckhardt/PrefWork.zip` as an Eclipse project containing all java sources and all required libraries or can be downloaded as SVN checkout at [20]. The SVN archive contains Java sources and sample configuration files.

5.1 Future work

We plan to introduce time dimension to PrefWork. Netflix [21] datasets uses a timestamp for each rating. This will enable to study the evolution of the preferences in time, which is a challenging problem. However, the integration of the time dimension into PrefWork can be done in several ways and the right one is yet to be chosen.

Allowing other sources of data apart from the ratings is a major issue. The clickthrough data can be collected without any effort of the user and can be substantially larger than the number of ratings. But its integration into PrefWork would require a large reorganisation of existing methods.

References

1. I.H. Witten, E. Frank: *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd Edition. Morgan Kaufmann, San Francisco (2005).
2. I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, T. Euler: *Yale: Rapid prototyping for complex data mining tasks.* In Ungar, L., Craven, M., Gunopulos, D., Eliassi-Rad, T., eds.: KDD'06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, New York, NY, USA, ACM (August 2006), 935–940.
3. R-project. `http://www.r-project.org/`.
4. SAS enterprise miner. `http://www.sas.com/`.
5. SPSS Clementine. `http://www.spss.com/software/modeling/modeler/`.
6. Š. Pero, T. Horváth: *Winston: A data mining assistant.* In: To appear in proceedings of RDM 2009, 2009.
7. P. Viappiani, B. Faltings: *Implementing example-based tools for preference-based search.* In: ICWE'06: Proceedings of the 6th international conference on Web engineering, New York, NY, USA, ACM, 2006, 89–90.
8. P. Viappiani, P. Pu, B. Faltings: *Preference-based search with adaptive recommendations.* AI Commun. **21**, 2-3, 2008, 155–175.
9. S. Holland, M. Ester, W. Kiessling: *Preference mining: A novel approach on mining user preferences for personalized applications.* In: Knowledge Discovery in Databases: PKDD 2003, Springer Berlin / Heidelberg, 2003, 204–216.
10. Cofi: A Java-Based Collaborative Filtering Library. `http://www.nongnu.org/cofi/`.
11. Apache Mahout project. `http://lucene.apache.org/mahout/`.
12. Taste project. `http://taste.sourceforge.net/old.html`.
13. T. Horváth, P. Vojtáš: *Induction of fuzzy and annotated logic programs.* In Muggleton, S., Tamaddoni-Nezhad, A., Otero, R., eds.: ILP06 - Revised Selected papers on Inductive Logic Programming. Number 4455 in Lecture Notes In Computer Science, Springer Verlag, 2007, 260–274.
14. A. Eckhardt: *Various aspects of user preference learning and recommender systems.* In Richta, K., Pokorný, J., Snášel, V., eds.: DATESO 2009. CEUR Workshop Proceedings, Česká technika - nakladatelství ČVUT, 2009, 56–67.
15. A. Eckhardt, P. Vojtáš: *Considering data-mining techniques in user preference learning.* In: 2008 International Workshop on Web Information Retrieval Support Systems, 2008, 33–36.
16. T. Dvořák: *Induction of user preferences in semantic web*, in Czech. Master Thesis, Charles University, Czech Republic, 2008.
17. The Internet Movie Database. `http://www.imdb.com/`.
18. A. Eckhardt: *Inductive models of user preferences for semantic web.* In Pokorný, J., Snášel, V., Richta, K., eds.: DATESO 2007. Volume 235 of CEUR Workshop Proceedings., Matfyz Press, Praha, 2007, 108–119.
19. S. Muggleton: *Learning from positive data.* 1997, 358–376
20. PrefWork - a framework for testing methods for user preference learning. `http://code.google.com/p/prefwork/`.
21. Netflix dataset, `http://www.netflixprize.com`.