Supporting KDD Applications by the *k*-Nearest Neighbor Join

Christian Böhm and Florian Krebs

University for Health Informatics and Technology,
Innrain 98,
6020 Innsbruck,
Austria
Christian.Boehm@umit.at
Krebs.Florian@symplex.de

Abstract. The similarity join has become an important database primitive to support similarity search and data mining. A similarity join combines two sets of complex objects such that the result contains all pairs of similar objects. Well-known are two types of the similarity join, the *distance range join* where the user defines a distance threshold for the join, and the *closest point query* or *k-distance join* which retrieves the *k* most similar pairs. In this paper, we propose an important, third similarity join operation called *k-nearest neighbor join* which combines each point of one point set with its *k* nearest neighbors in the other set. We discover that many standard algorithms of *Knowledge Discovery in Databases* (*KDD*) such as *k*-means and *k*-medoid clustering, nearest neighbor classification, data cleansing, postprocessing of sampling-based data mining etc. can be implemented on top of the *k*-nn join operation to achieve performance improvements without affecting the quality of the result of these algorithms. Our list of possible applications includes standard methods for all stages of the KDD process including preprocessing, data mining, and postprocessing. Thus, our method is turbo charging the complete KDD process.

1 Introduction

Knowledge Discovery in Databases (KDD) is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data [12]. The KDD process [3] is an interactive and iterative process, involving numerous steps including preprocessing of the data (data cleansing) and postprocessing (evaluation of the results). The core step of the KDD process is *data mining*, i.e. finding patterns of interest such as *clusters*, *outliers*, *classification rules* or *trees*, *association rules*, and *regressions*.

KDD algorithms in multidimensional databases are often based on similarity queries which are performed for a high number of objects. Recently, it has been recognized that many algorithms of similarity search [2] and data mining [4] can be based on top of a single join query instead of many similarity queries. Thus, a high number of single similarity queries is replaced by a single run of a *similarity join*. The most well-known form of the similarity join is the distance range join $R \underset{\varepsilon}{\bowtie} S$ which is defined for two finite sets

V. Mařík et al. (Eds.): DEXA 2003, LNCS 2736, pp. 504–516, 2003.

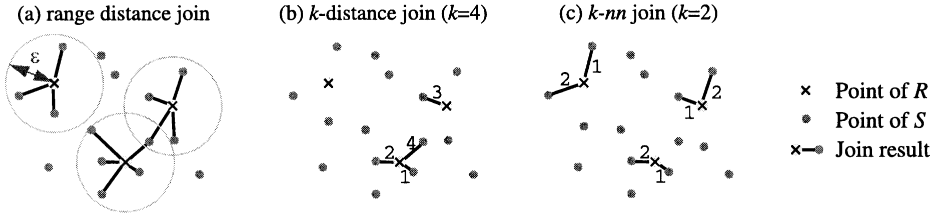


Fig. 1. Difference between similarity join operations

of vectors, $R = \{r_1,...,r_n\}$ and $S = \{s_1,...,s_m\}$, as the set of all pairs from $R \times S$ having a distance of no more than ε:

$$R \underset{\varepsilon}{\bowtie} S := \{(r_i,s_j) \in R \times S \mid \|p_i - q_j\| \le \varepsilon\}$$

E.g. in [4], it has been shown that density based clustering algorithms such as DBSCAN [21] or the hierarchical cluster analysis method OPTICS [1] can be accelerated by high factors of typically one or two orders of magnitude by the range distance join. Due to its importance, a large number of algorithms to compute the range distance join of two sets have been proposed, e.g. [23, 18, 6]

Another important similarity join operation which has been recently proposed is the incremental distance join [15]. This join operation orders the pairs from $R \times S$ by increasing distance and returns them to the user either on a give-me-more basis, or based on a user specified cardinality of *k* best pairs (which corresponds to a *k*-closest pair operation in computational geometry, cf. [19]). This operation can be successfully applied to implement data analysis tasks such as noise-robust catalogue matching and noise-robust duplicate detection [10].

In this paper, we propose a third kind of similarity join, the *k*-nearest neighbor similarity join, short *k*-nn join. This operation is motivated by the observation that a great majority of data analysis and data mining algorithms is based on *k*-nearest neighbor queries which are issued separately for a large set of *query points* $R = \{r_1,...,r_n\}$ against another large set of *data points* $S = \{s_1,...,s_m\}$. In contrast to the incremental distance join and the *k*-distance join which choose the best pairs from the complete pool of pairs $R \times S$, the *k*-nn join combines each of the points of *R* with its *k* nearest neighbors in *S*. The differences between the three kinds of similarity join operations are depicted in figure 1.

Applications of the *k*-nn join include but are not limited to the following list: *k*-nearest neighbor classification, *k*-means and *k*-medoid clustering, sample assessment and sample postprocessing, missing value imputation, *k*-distance diagrams, etc. We briefly discuss using these algorithms on top of a join operation in section 4.

Our list of applications covers all stages of the KDD process. In the preprocessing step, data cleansing algorithms are typically based on *k*-nearest neighbor queries for each of the points with NULL values against the set of complete vectors. The missing values can be computed e.g. as the weighted means of the values of the *k* nearest neighbors. A *k*-distance diagram can be used to determine suitable parameters for data mining. Additionally, in the core step, i.e. data mining, many algorithms such as clustering and classification are based on *k*-nn queries. As such algorithms are often time consuming and have at least a linear, often $n \log n$ or even quadratic complexity they typically run on a sample set rather than the complete data set. The k-nn-queries are used to assess the quality of the sample set (preprocessing). After the run of the data mining algorithm, it is necessary to relate the result to the complete set of database points [9]. The typical method for doing that is again a k-nn-query for each of the database points with respect to the set of classified sample points.

In all these algorithms, it is possible to replace a large number of k-nn queries which are originally issued separately, by a single run of a k-nn join. Therefore, the k-nn join gives powerful support for all stages of the KDD process.

The remainder of the paper is organized as follows: In section 2, we give a classification of the well-known similarity join operations and review the related work. In section 3, we define the new operation, the k-nearest neighbor join. Section 4 is dedicated to applications of the k-nn join. We show exemplary, that typical data mining methods can be easily implemented on top of the join. The experimental evaluation of our approach is presented in section 5 and section 6 concludes the paper.

2 Related Work

In the relational data model a join means to combine the tuples of two relations R and S into pairs if a *join predicate* is fulfilled. In multidimensional databases, R and S contain points (feature vectors) rather than ordinary tuples. In a *similarity join*, the join predicate is similarity, e.g. the Euclidean distance between two feature vectors.

2.1 Distance Range Based Similarity Join

The most prominent and most evaluated similarity join operation is the distance range join. Therefore, the notions *similarity join* and *distance range join* are often used interchangeably. Unless otherwise specified, when speaking of *the similarity join*, often the distance range join is meant by default. For clarity in this paper, we will not follow this convention and always use the more specific notions.

As depicted in figure 1a, the distance range join $R \underset{\varepsilon}{\bowtie} S$ of two multidimensional or metric sets R and S is the set of pairs where the distance of the objects does not exceed the given parameter e. Formally:

Definition 1 Distance Range Join (e-Join)

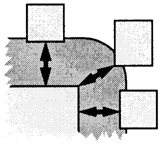
The distance range join $R \underset{\varepsilon}{\bowtie} S$ of two finite multidimensional or metric sets R and S is the set

$$R \underset{\varepsilon}{\bowtie} S := \{(r_i, s_j) \in R \times S : \|r_i - s_j\| \leq \varepsilon\}$$

The distance range join can also be expressed in a SQL like fashion:

SELECT * FROM R, S **WHERE** $\|R.\text{obj} - S.\text{obj}\| \leq \varepsilon$

In both cases, $\|\cdot\|$ denotes the distance metric which is assigned to the multimedia objects. For multidimensional vector spaces, $\|\cdot\|$ usually corresponds to the Euclidean distance.



$$\text{mindist}^2 = \sum_{0 \le i < d} \begin{cases} (R.\text{lb}_i - S.\text{ub}_i)^2 & \text{if } R.\text{lb}_i > S.\text{ub}_i \\ 0 & \text{otherwise} \\ (S.\text{lb}_i - R.\text{ub}_i)^2 & \text{if S.lb}_\text{i} > \text{R.ub}_\text{i} \end{cases}$$

Fig. 2. mindist for the similarity join on R-trees

The distance range join can be applied in density based clustering algorithms which often define the local data density as the number of objects in the e-neighborhood of some data object. This essentially corresponds to a self-join using the distance range paradigm.

Like for plain range queries in multimedia databases, a general problem of distance range joins from the users' point of view is that it is difficult to control the result cardinality of this operation. If ε is chosen too small, no pairs are reported in the result set (or in case of a self join: each point is only combined with itself). In contrast, if ε is chosen too large, each point of R is combined with every point in S which leads to a quadratic result size and thus to a time complexity of any join algorithm which is at least quadratic; more exactly $o\,(|R| \cdot |S|)$. The range of possible ε-values where the result set is non-trivial and the result set size is sensible is often quite narrow, which is a consequence of the curse of dimensionality. Provided that the parameter his chosen in a suitable range and also adapted with an increasing number of objects such that the result set size remains approximately constant the typical complexity of advanced join algorithms is better than quadratic.

Most related work on join processing using multidimensional index structures is based on the *spatial join*. The spatial join operation is defined for 2-dimensional polygon databases where the join predicate is the intersection between two objects. This kind of join predicate is prevalent in map overlay applications. We adapt the relevant algorithms to allow distance based predicates for multidimensional point databases instead of the intersection of polygons.

The most common technique is the *R-tree Spatial Join (RSJ)* [8] which processes R-tree like index structures built on both relations R and S. RSJ is based on the lower bounding property which means that the distance between two points is never smaller than the distance (the so-called mindist, cf. figure 2) between the regions of the two pages in which the points are stored. The RSJ algorithm traverses the indexes of R and S synchronously. When a pair of directory pages (P_R, P_S) is under consideration, the algorithm forms all pairs of the child pages of P_R and P_S having distances of at most ε. For these pairs of child pages, the algorithm is called recursively, i.e. the corresponding indexes are traversed in a depth-first order. Various optimizations of RSJ have been proposed.

Recently, index based similarity join methods have been analyzed from a theoretical point of view. [7] proposes a cost model based on the concept of the Minkowski sum [5] which can be used for optimizations such as page size optimization. The analysis reveals a serious optimization conflict between CPU and I/O time. While the CPU requires fine-grained partitioning with page capacities of only a few points per page, large block sizes of up to 1 MB are necessary for efficient I/O operations. The authors propose the *Multipage Index* (*MuX*), a complex index structure with large pages (optimized for I/O) which accommodate a secondary search structure (optimized for maximum CPU efficiency). It is shown that the resulting index yields an I/O performance which is similar (*equal*, up to a small additional overhead by the more complex structure) to the I/O optimized R-tree similarity join and a CPU performance which is close to the CPU optimized R-tree similarity join.

A join algorithm particularly suited for similarity self joins is the *ε-kdB-tree* [23]. The basic idea is to partition the data set perpendicularly to one selected dimension into stripes of width ε to restrict the join to pairs of subsequent stripes. The join algorithm is based on the assumption that the database cache is large enough to hold the data points of two subsequent stripes. In this case, it is possible to join the set in a single pass. To speed up the CPU operations, for each stripe a main memory data structure, the ε-kdB-tree is constructed which also partitions the data set according to the other dimensions until a defined node capacity is reached. For each dimension, the data set is partitioned at most once into stripes of width ε. Finally, a tree matching algorithm is applied which is restricted to neighboring stripes.

Koudas and Sevcik have proposed the *Size Separation Spatial Join* [17] and the *Multidimensional Spatial Join* [18] which make use of space filling curves to order the points in a multidimensional space. Each point is considered as a cube with side-length ε in the multidimensional space. The cube is assigned a value l (level) which is essentially the size of the largest cell (according to the Hilbert decomposition of the data space) that contains the point. The points are distributed over several *level-files* each of which contains the points of a level in the order of their Hilbert values. For join processing, each subpartition of a level-file must be matched against the corresponding subpartitions at the same level and each higher level file of the other data set.

An approach which explicitly deals with massive data sets and thereby avoids the scalability problems of existing similarity join techniques is the *Epsilon Grid Order (EGO)* [6]. It is based on a particular sort order of the data points which is obtained by laying an equi-distant grid with cell length hover the data space and then compares the grid cells lexicographically.

2.2 Closest Pair Queries

It is possible to overcome the problems of controlling the selectivity by replacing the range query based join predicate using conditions which specify the selectivity. In contrast to range queries which retrieve potentially the whole database, the selectivity of a (k-) closest pair query is (up to tie situations) clearly defined. This operation retrieves the k pairs of $R \times S$ having minimum distance. (cf. figure 1b) Closest pair queries do not only play an important role in the database research but have also a long history in computational geometry [19]. In the database context, the operation has been introduced by Hjaltason and Samet [15] using the term (k-) *distance join*. The (k-)closest pair query can be formally defined as follows:

Definition 2 (k-) Closest Pair Query $R \bowtie_{k\text{-CP}} S$

$R \bowtie_{k\text{-CP}} S$ is the smallest subset of $R \times S$ that contains at least
k pairs of points and for which the following condition holds:

$$\forall\, (r,s) \in R \bowtie_{k\text{-CP}} S,\ \forall\, (r',s') \in R \times S \setminus R \bowtie_{k\text{-CP}} S\colon \|r-s\| < \|r'-s'\| \qquad (1)$$

This definition directly corresponds to the definition of (*k*-) nearest neighbor queries, where the single data object *o* is replaced by the pair (*r*,*s*). Here, tie situations are broken by enlargement of the result set. It is also possible to change definition 2 such that the tie is broken non-deterministically by a random selection. [15] defines the closest pair query (non-deterministically) by the following SQL statement:

SELECT * FROM *R*, *S*
ORDER BY ||*R*.obj - *S*.obj||
STOP AFTER *k*

We give two more remarks regarding self joins. Obviously, the closest pairs of the selfjoin $R \bowtie_{k\text{-CP}} R$ are the *n* pairs (r_i,r_i) which have trivially the distance 0 (for any distance metric), where $n = |R|$ is the cardinality of *R*. Usually, these trivial pairs are not needed, and, therefore, they should be avoided in the **WHERE** clause. Like the distance range selfjoin, the closest pair selfjoin is symmetric (unless nondeterminism applies). Applications of closest pair queries (particularly self joins) include similarity queries like

- find all stock quota in a database that are similar to each other
- find music scores which are similar to each other
- noise-robust duplicate elimination of any multimedia application

For plain similarity search in multimedia-databases, it is often useful to replace *k*-nearest neighbor queries by ranking queries which retrieve the first, second, third,... nearest neighbor in a one-by-one fashion. The actual number *k* of nearest neighbors to be searched is initially unknown. The user (or some application program on top of the database) decides according to a criterion which is unknown to the DBMS whether or not further neighbors are required. This kind of processing called *incremental* distance join can also be defined on top of the closest pair query, e.g. by cancelling the STOP AFTER clause in the SQL statement above. The query results are passed to the application program using some cursor concept. It is important to avoid computing the complete ranking in the initialization phase of the cursor, because determining the complete ranking is unnecessarily expensive if the user decides to stop the ranking after retrieving only a few result pairs. Hjaltason and Samet [15] also define the distance semijoin which performs a GROUP BY operation on the result of the distance join. All join operations, *k*-distance join, incremental distance join and the distance semijoin are evaluated using a pqueue data structure where node-pairs are ordered by increasing distance.

The most interesting challenge in algorithms for the distance join is the strategy to access pages and to form page pairs. Analogously to the various strategies for single nearest neighbor queries such as [20] and [14], Corral et al. propose 5 different strategies including recursive algorithms and an algorithm based on a pqueue [11]. Shin et al. [22] proposed a plane sweep algorithm for the node expansion for the above mentioned pqueue algorithm [15, 11]. In the same paper [22], Shim et al. also propose the *adaptive multi-stage algorithm* which employs aggressive pruning and compensation methods based on statistical estimates of the expected distance values.

3 The *k*-Nearest Neighbor Join

The range distance join has the disadvantage of a result set cardinality which is difficult to control. This problem has been overcome by the closest pair query where the result set size (up to the rare tie effects) is given by the query parameter *k*. However, there are only few applications which require the consideration of the *k* best pairs of two sets. Much more prevalent are applications such as classification or clustering where each point of one set must be combined with its *k* closest partners in the other set, which is exactly the operation that corresponds to our new *k*-nearest neighbor similarity join (cf. figure 1c). Formally, we define the *k*-nn-join as follows:

Definition 3 *k*-nn-Join $R \ltimes_{k\text{-}nn} S$

$R \ltimes_{k\text{-}nn} S$ is the smallest subset of $R \times S$ that contains for each point of R at least k points of S and for which the following condition holds:

$$\forall\, (r,s) \in R \ltimes_{k\text{-}nn} S,\ \forall\, (r,s') \in R \times S \setminus R \ltimes_{k\text{-}nn} S\colon \|r-s\| < \|r-s'\| \tag{2}$$

In contrast to the closest pair query, here it is guaranteed that each point of *R* appears in the result set exactly *k* times. Points of *S* may appear once, more than once (if a point is among the *k*-nearest neighbors of several points in *R*) or not at all (if a point does not belong to the *k*-nearest neighbors of any point in *R*). Our *k*-nn join can be expressed in an extended SQL notation:

```
SELECT * FROM R,
   (  SELECT * FROM S
      ORDER BY ||R.obj – S.obj||
      STOP AFTER k )
```

The closest pair query applies the principle of the nearest neighbor search (finding *k* best *things*) on the basis of the pairs. Conceptually, first all pairs are formed, and then, the best *k* are selected. In contrast, the *k*-nn join applies this principle on a basis "per point of the first set". For each of the points of *R*, the *k* best join partners are searched. This is an essential difference of concepts.

Again, tie situations can be broken deterministically by enlarging the result set as in this definition or by random selection. For the selfjoin, we have again the situation that each point is combined with itself which can be avoided using the WHERE clause. Unlike the ε-join and the *k*-closest pair query, the *k*-nn selfjoin is not symmetric as the nearest neighbor relation is not symmetric. Equivalently, the join $R \ltimes_{k\text{-}nn} S$ which retrieves the *k* nearest neighbors for each point of *R* is essentially different from $S \ltimes_{k\text{-}nn} R$ which retrieves the nearest neighbors of each *S*-point. This is symbolized in our symbolic notation which uses an *asymmetric* symbol for the *k*-nn join in contrast to the other similarity join operations.

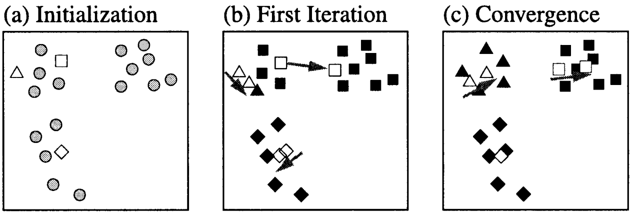


Fig. 3. *k*-Means Clustering

4 Applications

4.1 *k*-Means and *k*-Medoid Clustering

The *k*-means method [13] is the most important and most widespread approach to *clustering*. For *k*-means clustering the number *k* of clusters to be searched must be previously known. The method determines *k* cluster centers such that each database point can be assigned to one of the centers to minimize the overall distance of the database points to their associated center points.

The basic algorithm for *k*-means clustering works as follows: In the initialization, *k* database points are randomly selected as tentative cluster centers. Then, each database point is associated to its closest center point and, thus, a tentative *cluster* is formed. Next, the cluster centers are redetermined as the means point of all points of the center, simply by forming the vector sum of all points of a (tentative) cluster. The two steps (1) point association and (2) cluster center redetermination are repeated until convergence (no more considerable change). It has been shown that (under several restrictions) the algorithm always converges. The cluster centers which are generated in step (2) are artificial points rather than database points. This is often not desired, and therefore, the *k*-medoid algorithm always selects a database point as a cluster center.

The *k*-means algorithm is visualized in figure 3 using $k = 3$. At the left side (a) $k = 3$ points (white symbols ◇ △ □) are randomly selected as initial cluster centers. Then in figure 3(b) the remaining data points are assigned to the closest center which is depicted by the corresponding symbols (◆ ▲ ■). The cluster centers are redetermined (moving arrows). The same two operations are repeated in figure 3(c). If the points are finally assigned to their closest center, no assignment changes, and, therefore, the algorithm terminates clearly having separated the three visible clusters. In contrast to density-based approaches, *k*-means only separates compact clusters, and the number of actual clusters must be previously known.

It has not yet been recognized in the data mining community that the point association step which is performed in each iteration of the algorithm corresponds to a ($k = 1$) nearest neighbor join between the set of center points (at the right side) and the set of database points (at the left side of the join symbol) because each database point is associated with its nearest neighbor among the center points:

$$\text{database-point-set} \ltimes_{\text{1-NN}} \text{center-point-set}$$

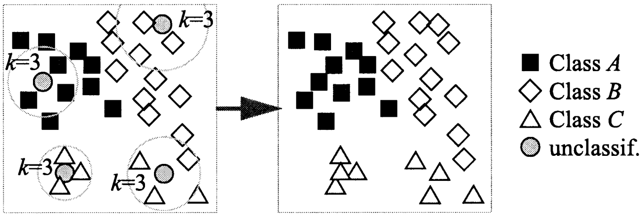


Fig. 4. *k*-Nearest Neighbor Classification

During the iteration over the cursor of the join, it is also possible to keep track of changes and to redetermine the cluster center for the next iteration. The corresponding pseudocode is depicted in the following:

```
repeat
  change := false ;
  foreach (dp,cp) $\in$ database-point-set $\ltimes_{1\text{-NN}}$ center-point-set do
    if dp.center $\neq$ cp.id then change := true ;
    dp.center := cp.id ;
    cp.newsum := cp.newsum + dp.point ;
    cp.count := cp.count + 1 ;
  foreach cp $\in$ center-point-set do
    cp.point := cp.newsum / cp.count ;
until $\neg$ change ;
```

4.2 *k*-Nearest Neighbor Classification

Another very important data mining task is classification. Classification is somewhat similar to clustering (which is often called *unsupervised classification*). In classification, a part of the database objects is assigned to class labels. For classification, also a set of objects without class label (new objects) is given. The task is to determine the class labels for each of the unclassified objects by taking the properties of the classified objects into account. A widespread approach is to build up tree like structures from the classified objects where the nodes correspond to ranges of attribute values and the leaves indicate the class labels (called *classification trees* [13]). Another important approach is *k*-nearest neighbor classification [16]. Here, for each unclassified object, a *k*-nearest neighbor query on the set of classified objects is evaluated (*k* is a parameter of the algorithm). The object is assigned to the class label of the majority of the resulting objects of the query. This principle is visualized in figure 4. As for each unclassified object a *k*-nn-query on the set of classified objects is evaluated, this corresponds again to a *k*-nearest neighbor join:

$$\text{unclassified-point-set} \ltimes_{k\text{-}nn} \text{classified-point-set}$$

4.3 Sampling Based Data Mining

Data mining methods which are based on sampling often require a *k*-nearest neighbor join between the set of sample points and the complete set of original database points. Such a join is necessary, for instance, to assess the quality of a sample. The *k*-nearest neighbor join can give hints whether the sample rate is too small. Another application is the transfer of the data mining result onto the original data set after the actual run of the data mining algorithm [9]. For instance, if a clustering algorithm has detected a set of clusters in the sample set, it is often necessary to associate each of the database points to the cluster to which it belongs. This can be done by a *k*-nn join with $k = 1$ between the point set and the set of sample points:

$$\text{sample-set} \ltimes_{k\text{-}nn} \text{point-set}$$

4.4 *k*-Distance Diagrams

The most important limitation of the DBSCAN algorithm is the difficult determination of the query radius e. In [21] a method called *k*-distance diagram is proposed to determine a suitable radius e. For this purpose, a number of objects (typically 5-20 percent of the database) is randomly selected. For these objects, a *k*-nearest neighbor query is evaluated where *k* corresponds to the parameter MIN_PTS which will be used during the run of DBSCAN. The resulting distances between the query points and the *k*-th nearest neighbor of each are then sorted and depicted in a diagram (cf. figure 5). Vertical gaps in that plot indicate distances that clearly separate different clusters, because there exist larger *k*-nearest neighbor distances (inter-cluster distances, noise points) and smaller ones (intra-cluster distance). As for each sample point a *k*-nearest neighbor query is evaluated on the original point set, this corresponds to a *k*-nn-join between the sample set and the original set:

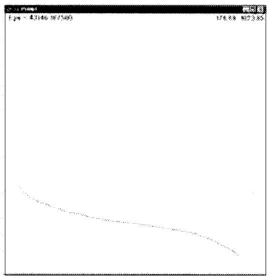


Fig. 5. *k*-Distance Diagram

$$\text{sample-set} \ltimes_{k\text{-}nn} \text{point-set}$$

If the complete data set is taken instead of the sample, we have a *k*-nn self join:

$$\text{point-set} \ltimes_{k\text{-}nn} \text{point-set}$$

5 Experimental Evaluation

We implemented a *k*-Means clustering algorithm and a *k*-Nearest Neighbor classification algorithm in both versions traditionally with single similarity queries (nearest neighbor queries) as well as on top of our new database primitive, the similarity join. Our *k*-NN similarity join algorithm used the Multipage Index [7] which allows a separate optimization of CPU and I/O performance. The competitive technique, the evaluation on top of single similarity queries, was also supported by the same index structure which is traversed using a variation of the nearest neighbor algorithm by Hjaltason and Samet [14] which has been shown to yield an optimal number of page accesses.

All our experiments were carried out under Windows NT4.0 SP6 on Fujitsu-Siemens Celsius 400 machines equipped with a Pentium III 700 MHz processor and at least 128 MB main memory. The installed disk device was a Seagate ST310212A with a sustained transfer rate of about 9 MB/s and an average read access time of 8.9 ms with an average latency time of 5.6 ms. We allowed about 20% of the database size as cache or buffer for either technique and included the index creation time for our k-nn join and the hs-algorithm. We used large data sets from various application domains, in particular:

- 5-dimensional feature vectors from earth observation. These data sets have been generated from the well known SEQUOIA benchmark
- 20-dimensional data from astronomy observations
- 64-dimensional feature vectors from a color image database (color histograms)

In our first set of experiments, we tested the *k*-nearest neighbor classification method where we varied the number of training objects (cf. figure 6) as well as the number of objects which have to be classified (cf. figure 7). The superiority of our new method becomes immediately clear from both experiments. The improvement factor over the simple *k*-NN approach is high over all measured scales. It even improves for an increasing number of training objects or classified objects, respectively, and reaches a final factor of 9.1 in figure 6 (factor 2.0 in figure 7).

Figure 8 varies over our various data sets and shows that the improvement factor also grows with increasing data space dimension. Our new database primitive outperforms the well-known approach by factors starting with 1.8 at the 5-dimensional space up to 3.2 at the 64-dimensional space. In our last experiment, depicted in figure 9, we tested the *k*-nearest neighbor clustering method. In the depicted experiment, we varied the number of clusters to be searched. Again, the improvement factor grows from 1.4 for the smallest number of clusters to 5.1 for our largest number of clusters.

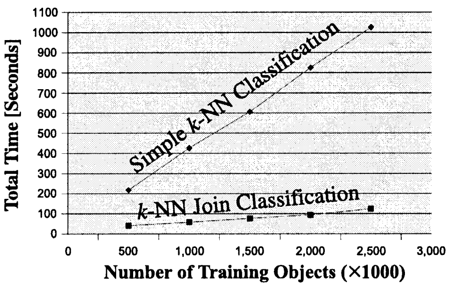


Fig. 6. Classification (5-dim Sequoia data, 1000 classified objects)

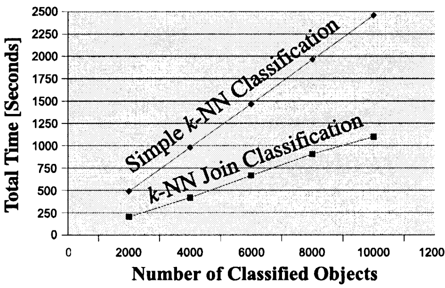


Fig. 7. Classification (64D image data, 100,000 training objects)

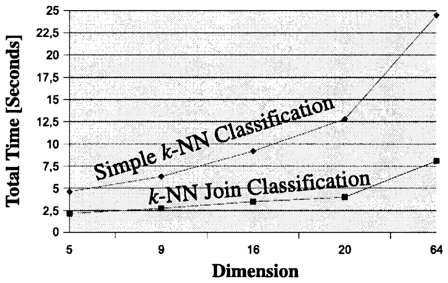


Fig. 8. *k*-NN Classification (100,000 training objects; 100 classified)

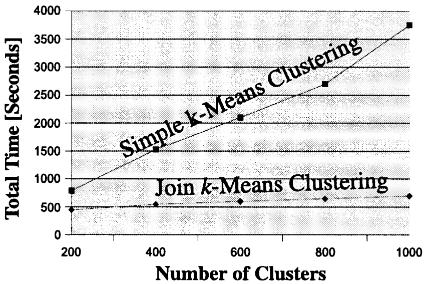


Fig. 9. *k*-Means Clustering (5d SEQUOIA data, 100,000 feature vectors)

6 Conclusions

In this paper, we have proposed a new kind of similarity join which is based on a *k*-nearest neighbor join predicate. In contrast to other types of similarity joins such as the distance range join, the *k*-distance join (*k*-closest pair query) and the incremental distance join, our new *k*-nn join combines each point of a point set *R* with its *k* nearest neighbors in another point set *S*. We have demonstrated that many standard algorithms of *Knowledge Discovery in Databases* (*KDD*) such as *k*-means and *k*-medoid clustering, nearest neighbor classification, data cleansing, postprocessing of sampling-based data mining etc. can be implemented on top of the *k*-nn join operation to achieve performance improvements without affecting the quality of the result of these algorithms. Our list of possible applications includes standard methods for all stages of the KDD process including preprocessing, data mining, and postprocessing. In our experimental evaluation, we have demonstrated the performance potential of the new database primitive. The results yield high performance gains for classification and clustering algorithms.

References

1 Ankerst M., Breunig M. M., Kriegel H.-P., Sander J.: *OPTICS: Ordering Points To Identify the Clustering Structure*, ACM SIGMOD Int. Conf. on Management of Data, 1999.

2 Agrawal R., Lin K., Sawhney H., Shim K.: *Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases*, Int. Conf on Very Large Data Bases (VLDB), 1995.

3 Brachmann R., Anand T.: *The Process of Knowledge Discovery in Databases,* in: Fayyad et al.: *Advances in Knowledge Discovery and Data Mining*, AAAI Press, 1996.

4 Böhm C., Braunmüller B., Breunig M. M., Kriegel H.-P.: *Fast Clustering Based on High-Dimensional Similarity Joins*, Int. Conf. on Information Knowledge Management (CIKM), 2000.

5 Berchtold S., Böhm C., Keim D., Kriegel H.-P.: *A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space*, ACM Symposium on Principles of Database Systems (PODS), 1997.

6 Böhm C., Braunmüller B., Krebs F., Kriegel H.-P.: *Epsilon Grid Order: An Algorithm for the Similarity Join on Massive High-Dimensional Data*, ACM SIGMOD Int. Conf. on Management of Data, 2001.

7 Böhm C., Kriegel H.-P.: *A Cost Model and Index Architecture for the Similarity Join,* IEEE Int. Conf on Data Engineering (ICDE), 2001.

8 Brinkhoff T., Kriegel H.-P., Seeger B.: *Efficient Processing of Spatial Joins Using R-trees*, ACM SIGMOD Int. Conf. on Management of Data, 1993.

9 Breunig M. M., Kriegel H.-P., Kröger P., Sander J.: *Data Bubbles: Quality Preserving Performance Boosting for Hierarchical Clustering*, ACM SIGMOD Int. Conf. on Management of Data, 2001.

10 Böhm C.: *The Similarity Join: A Powerful Database Primitive for High Performance Data Mining*, tutorial, IEEE Int. Conf. on Data Engineering (ICDE), 2001.

11 Corral A., Manolopoulos Y., Theodoridis Y., Vassilakopoulos M.: *Closest Pair Queries in Spatial Databases*, ACM SIGMOD Int. Conf. on Management of Data, 2000.

12 Fayyad U. M., Piatetsky-Shapiro G., Smyth P.: *From Data Mining to Knowledge Discovery: An Overview*, in: Fayyad et al.: *Advances in Knowledge Discovery and Data Mining*, AAAI Press, 1996.

13 Han J, Kamber M.: *Data Mining: Concepts and Techniques*, Morgan Kaufmann, 2000.

14 Hjaltason G. R., Samet H.: *Ranking in Spatial Databases*, Int. Symp. on Large Spatial Datab. (SSD), 1995.

15 Hjaltason G. R., Samet H.: *Incremental Distance Join Algorithms for Spatial Databases*, SIGMOD Int. Conf. on Management of Data, 1998.

16 Hattori K., Torii Y.: *Effective algorithms for the nearest neighbor method in the clustering problem.* Pattern Recognition, 26(5), 1993.

17 Koudas N., Sevcik C.: *Size Separation Spatial Join*, ACM SIGMOD Int. Conf. on Managem. of Data, 1997

18 Koudas N., Sevcik C.: *High Dimensional Similarity Joins: Algorithms and Performance Evaluation*, IEEE Int. Conf. on Data Engineering (ICDE), Best Paper Award, 1998.

19 Preparata F. P., Shamos M. I.: *Computational Geometry*, Springer 1985.

20 Roussopoulos N., Kelley S., Vincent F.: *Nearest Neighbor Queries*, ACM SIGMOD Int. Conf. on Management of Data, 1995.

21 Sander J., Ester M., Kriegel H.-P., Xu X.: *Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Applications*, Data Mining and Knowledge Discovery, Kluwer Academic Publishers, Vol. 2, No. 2, 1998.

22 Shin H., Moon B., Lee S.: *Adaptive Multi-Stage Distance Join Processing*, ACM SIGMOD Int. Conf. on Management of Data, 2000.

23 Shim K., Srikant R., Agrawal R.: *High-Dimensional Similarity Joins*, IEEE Int. Conf. on Data Engin. 1997.