Database Implementation of a Model-Free Classifier*

Konstantinos Morfonios

Department of Informatics and Telecommunications, University of Athens
kmorfo@di.uoa.gr

Abstract. Most methods proposed so far for classification of high-dimensional data are memory-based and obtain a model of the data classes through training before actually performing any classification. As a result, these methods are ineffective on (a) very large datasets stored in databases or data warehouses, (b) data whose partitioning into classes cannot be captured by global models and is sensitive to local characteristics, and (c) data that arrives continuously to the system with pre-classified and unclassified instances mutually interleaved and whose successful classification is sensitive to using the most complete and/or most up-to-date information. In this paper, we propose LOCUS, a scalable model-free classifier that overcomes these problems. LOCUS is based on ideas from pattern recognition and is shown to converge to the optimal Bayes classifier as the size of the datasets involved increases. Moreover, LOCUS is data-scalable and can be implemented using standard SQL over arbitrary database tables. To the best of our knowledge, LOCUS is the first classifier that combines all the characteristics above. We demonstrate the effectiveness of LOCUS through experiments over both real-world and synthetic datasets, comparing it against memory-based decision trees. The results indicate an overall superiority of LOCUS over decision trees on both classification accuracy and data sizes that it can handle.

Keywords: Lazy Classification, Scalable Classification, Disk-Based Classification, Optimal Bayes.

1 Introduction

Consider a collection of "labeled" objects available, often called a *training set*. The objects are described by a number of attributes, called *features*, and are modeled as *feature vectors*. The object labels are used to categorize each object into one of several predefined *classes*. Assuming that the class of every object can be expressed as a function of the object's attributes, features are often called *predictor attributes*, while the corresponding class is the *dependent attribute*. Classification is the task of labeling new objects, whose class is unknown, using the a-priori labeling in the training set as a basis. Some example applications of common classification tasks

* The project is co-financed within Op. Education by the ESF (European Social Fund) and National Resources.

Y. Ioannidis, B. Novikov, and B. Rachev (Eds.): ADBIS 2007, LNCS 4690, pp. 83–97, 2007.

include automated medical diagnosis, target group identification, email filtering, character or speech recognition, and fraud detection.

Interestingly, traditional classification algorithms mainly focus on providing high accuracy even at the cost of iterative traversals of the training data, neglecting the potential access costs of such operations. Clearly, this is acceptable only for small datasets that fit in main memory: the majority of the literature involves datasets with only a few hundreds of instances at most.

Nevertheless, increasing use of computers and database management systems (DBMSs) in modern business environments, decreasing storage costs, and other similar trends have generated a wealth of data stored in databases and data warehouses. *Data mining* over such databases can discover useful information that can help their owners predict future events based on past observations. As a special case of data mining, classification presents new challenges when applied to training sets that are orders of magnitude larger than the ones typically considered. Revisiting traditional classification algorithms to make them applicable in today's large-scale environment is highly desirable.

Existing classifiers can be placed into two main categories: *eager* and *lazy*. Eager classifiers use an off-line training phase during which they build a model of the training set. During classification they use this model to label unknown objects. Popular eager classifiers include Decision Trees [18] and Support Vector Machines [5]. On the contrary, lazy classifiers need no training as they do not rely on a global model. For every unknown object, they search the training set to find the objects that are most similar to the unknown one (under various similarity measures on the objects' features, e.g., Euclidian distance). Then, they classify the given object based on the classes of these most similar objects. The most widely known lazy classifier is Nearest Neighbors [12].

Comparing the aforementioned general classification schemes, we can see that eager classifiers pay a considerable cost for (off-line) training but provide faster answers during decision, exploiting their use of a global model. On the contrary, lazy methods spend no time in training at the expense of increased costs during decision, due to on-line searching. Besides trivial training costs, another great advantage of lazy classifiers is that they exhibit greater accuracy on complex datasets when a global model is too hard to find, since they exploit only local information. Furthermore, they need no incremental maintenance when the training set is expanded with new known instances. This is extremely important in applications that involve a continuous flow of new training data and classification tasks that are sensitive to the most up-to-date information. For example, automatically deciding whether to buy or sell a given share in a stock-market application seems to be more effective if based on a window consisting of the most recent transactions, rather than on a global model built on older observations. Clearly, keeping such a window up to date is much easier in lazy schemes.

Our study of related work has revealed that most disk-based classification methods proposed so far mainly focus on the development of eager classifiers. To overcome this drawback, in this paper, we propose LOCUS (**L**azy **O**ptimal **C**lassification of **U**nlimited **S**calability), an effective, efficient, and disk-based

lazy classifier. LOCUS is data-scalable and can be implemented using standard SQL over arbitrary database tables. It overcomes the efficiency problems of existing lazy methods providing very fast on-line answers, while it still enjoys all the advantages of laziness described above. To the best of our knowledge, LOCUS is essentially the first disk-based lazy classifier with the following properties:

- It exhibits good classification accuracy, which improves as training sets become larger. This can be justified theoretically based on its convergence to the optimal Bayes classifier, which minimizes the classification error probability. The same is also verified experimentally in comparison to Decision Trees, a very popular and accurate existing classifier.
- It is database-friendly and, to the best of our knowledge, the only lazy method that uses a small and constant number of highly selective range queries[1] in order to classify unknown objects. Such queries actually need to access a very small part of the underlying database and have been well studied in the database literature. They can be expressed in standard SQL and existing query optimizers guarantee their fast response times with the use of traditional indices, e.g., B^+-Trees.

The rest of this paper is organized as follows: In Section 2, we review related work and, in Section 3, we describe LOCUS and show its optimality with respect to classification error probability. In Section 4, we present the results of our experimental evaluation using both real-world and synthetic datasets and, finally, we conclude in Section 5 and present our directions for future work.

2 Related Work

The problem of classification over small datasets that fit in main memory has been thoroughly studied in the past and there are a very large number of related papers. The most influential methods have been well described in existing textbooks related to pattern recognition [21] and machine learning [15]. Although LOCUS is a disk-based approach built on fundamental ideas originally conceived in these areas, surveying existing memory-based techniques in detail exceeds our purpose.

The need for scalable and database-friendly data mining techniques over extremely large datasets has given thrust to revisiting traditional methods under new constraints, in an attempt to transform them into scalable solutions. General ideas towards this end can be found in related surveys [6,17].

Interestingly, a large number of these efforts have focused on Decision Trees, mainly due to their ability of learning faster than other eager classifiers and their accuracy, which has been found comparable or superior to other classification models [8]. Moreover, every path in a Decision Tree can be easily converted into an SQL statement that can be used to access databases efficiently [1]. Popular

[1] The number of queries depends on the training set. In all cases we have seen in practice one or two queries are enough.

methods in this category include SLIQ [13], SPRINT [19], RainForest [9], MedGen [11], and BOAT [8]. Note that these methods do not actually propose a new classification model. Instead, they provide a disk-based implementation of a popular one, i.e., Decision Trees. The rationale behind LOCUS is similar. It provides a disk-based implementation for a rather familiar classifier based on counting training vectors in a fixed neighborhood centered at an incoming unknown vector. The main differences between LOCUS and the other techniques is that LOCUS is lazy and converges to an optimal solution with respect to classification error probability as the training dataset becomes larger.

Another popular approach that can be combined with known classifiers to reduce resource requirements is sampling and dataset size reduction. For example, CB-SVM [23] first applies BIRCH [24], a disk-based clustering algorithm, and then trains a classifier based on support vector machines using the centroids of the identified clusters instead of the original data. Such methods induce potentially expensive pre-processing that can be thought of as another form of training. Moreover, they leave a taste of defeat as they depend on lossy techniques. LOCUS considers every training vector as valuable. Both theoretical and experimental results show that its accuracy improves as datasets become larger, rendering it superior to sampling.

As we have already mentioned, lazy classifiers do not build a general model in a training stage, but access the training data on-line for every decision. Relying on local rather than global rules, they promise higher accuracy in inherently complex situations. Popular lazy classifiers include Nearest Neighbors [12], IB1-IB5 [3], and LazyDT [7]. Clearly, evaluating a similarity measure between an unknown vector and every training vector for ranking the latter accordingly is impractical in very large datasets. To overcome this drawback, several approaches have been proposed, based on specialized multidimensional indices for accelerating the search without scanning the entire training set [12]. (These methods mainly focus on Nearest Neighbors.) However, it has been recently shown [20] that accessing the data using such techniques can be even worse than a sequential scan through the entire dataset under very broad conditions. This is due to the fact that the distance difference between the nearest and the furthest neighbor is often so small that it turns methods based on Nearest Neighbors inaccurate [4]. Based on these conclusions, we consider LOCUS as essentially the first disk-based lazy classifier that guarantees both speed and accuracy over any large dataset.

In order to achieve this, LOCUS uses an SQL Interface Protocol (SIP) [10] and is based on highly selective range queries leaving the burden of data access to the DBMS. Using SQL for implementation of data mining methods has been proposed elsewhere [10] as a general hint. Querying the data on the server where it is originally stored (a) provides efficiency, (b) saves time from expensive export operations, (c) provides increased security, since data is available only to authorized users, and (d) enables scientists from different fields, potentially unwilling to learn the usage of new software, to use familiar interfaces on top of a database system. SQL interfaces have also been applied on construction of Decision Trees

[10] as well as in DBPredictor [14]. The latter is a lazy method that builds a custom model consisting of an IF-THEN rule for every unknown vector. To do this, it queries the training database iteratively, until some stopping criteria are met. Executing an arbitrary number of queries per instance for building a local model generates concerns about the performance of this method. Furthermore, the model constructed is rather ad-hoc and its accuracy is not based on a mathematical foundation. Unlike DBPredictor, LOCUS uses a small fixed number of queries per instance. Moreover, it builds no model but is based on raw counting, which is proven to converge to the optimal Bayes classifier.

3 LOCUS Classification

In this section, motivated by the need for a scalable and accurate disk-based lazy classifier, we propose LOCUS and argue that, to the best of our knowledge, it is the first lazy classifier with these properties that converges to optimality with respect to minimizing the classification error probability for large training sets. In principle, given an unlabeled feature vector $\mathbf{x}$, LOCUS counts the number of neighbors per class that reside in a fixed neighborhood around $\mathbf{x}$. Although this idea is not novel (counting neighbors is common in pattern recognition), its naive implementation overlooks that selecting the training vectors that live in the given neighborhood of an arbitrary $\mathbf{x}$ by scanning the entire training set is very expensive in large datasets. Moreover, although reasonable, it seems to be rather ad-hoc. In the following subsections, we provide answers for both: First, we show that our simple counting method can be based on a sound mathematical background [21] and converges to the optimal Bayes classifier. Second, we provide a very efficient disk-based method to implement the otherwise familiar task of counting. This is analogous to the contribution of algorithms like RainForest [9] that provide efficient disk-based algorithms for the construction of well-known Decision Trees in the area of eager classifiers.

3.1 Intuition of LOCUS

Assume a classification task in a D-dimensional feature space. For the sake of simplicity, let all features be numerical with discrete domains (LOCUS can also work with continuous and categorical features, as shown later). If C_i denotes the number of distinct values of the i-th feature ($i \in [1, D]$), then there are $C = C_1 \times C_2 \times \cdots \times C_D$ possible different feature vectors. Suppose that our training set is very dense and large, so that it contains an instance of every possible vector. Then, intuitively, when an unknown vector $\mathbf{x}$ comes, it seems reasonable to classify it according to the class of the training vector $\mathbf{y}$ that matches $\mathbf{x}$ exactly (i.e., $\mathbf{y} = \mathbf{x}$).

Unfortunately, this ideal scenario is very unrealistic, mainly due to two reasons: the number of features is usually large and most features have large domains. Both factors result in an increase of C. For example, if $D = 10$ and $C_i = 10 \; \forall \; i \in [1, D]$, then $C = 10^{10}$. Hence, even in this rather simple case an ideal training set should consist of 10 billion feature vectors, which seems impractical.

In practice, the available training set is usually sparse, i.e., it contains a very small subset of all possible feature vectors. Hence, finding an exact match to $\mathbf{x}$ has very low probability. To overcome this drawback, a reasonable solution is to loosen the condition of "exact match". Since finding a training vector $\mathbf{y}$ such that $\mathbf{y} = \mathbf{x}$ is usually infeasible, we can alternatively search for training vectors in a narrow neighborhood Y centered at $\mathbf{x}$. Formally, we can say that $\mathbf{y} \in Y$ iff $y_i \in [x_i\text{-}\delta_i, x_i\text{+}\delta_i]\ \forall\ i \in [1, D]$, where y_i (x_i) is the value of $\mathbf{y}$ ($\mathbf{x}$) for the i-th feature and δ_is denote our tolerance regarding the extent of the neighborhood Y around $\mathbf{x}$. If δ_i values are small enough to ensure a small deviation from the ideal scenario described above and large enough to make Y non-empty with high probability, then classifying $\mathbf{x}$ according to the majority class in Y seems intuitively reliable.

In the 2-dimensional example of Fig. 1, we would classify the unknown vector $\mathbf{x} = <x_1, x_2>$ as "+", as indicated by the majority of training points that fall within a small area centered at $\mathbf{x}$.

Clearly, loosening the "exact match" condition as described above is possible for all numerical features (either discrete or continuous). Furthermore, it is possible for categorical features, if their values can be ordered. In such cases the range $[x_i\text{-}\delta_i, x_i\text{+}\delta_i]$ must be substituted by a range of contiguous values around x_i according to the defined order. On the other hand, such loosening is not possible in non-ordered categorical features, in which case, we leave the equality condition unmodified. In general, this is of minor importance for two reasons: (a) Usually, there is a mixture of numerical and categorical features and loosening the equality condition for those that include ordering is just enough. (b) Categorical attributes have usually small domains, which makes the satisfaction of an exact match condition highly possible.

In the following subsection, we flesh out a known argument from pattern recognition [21] that proves optimality for the classifier described above, showing that it converges to the optimal Bayes classifier as training datasets become larger.

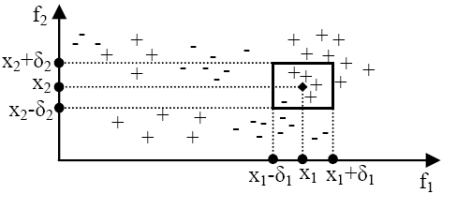

Fig. 1. Classification example of vector $\mathbf{x} = <x_1, x_2>$

3.2 Optimality

Assume that $\mathbf{x}$ is an unlabelled feature vector and that there are M possible classes $\omega_1, \cdots, \omega_M$ to which it can be classified. Then our task is to classify $\mathbf{x}$ into *the most probable* of these classes. Naturally, the term "most probable" brings in

mind the conditional probabilities $P(\omega_i \mid \mathbf{x})$, $i \in [1, M]$. Hence, it seems reasonable to classify $\mathbf{x}$ as ω_i if $P(\omega_i \mid \mathbf{x}) > P(\omega_j \mid \mathbf{x})\ \forall\ j \in [1, M]$ such that $j \neq i$. This rule is known as the *Bayes classification rule*. Actually, this reasonable rule, which seems rather empirical, turns out to minimize the classification error probability [21]. Hence, the Bayes classifier, which performs classification according to the Bayes classification rule, is optimal with respect to minimizing the classification error probability.

So, our original classification problem has now been transformed into that of comparing the conditional probabilities $P(\omega_i \mid \mathbf{x})$, $i \in [1, M]$. In order to fulfill this task, let us recall the Bayes rule from the probability theory basics:

$$P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(\mathbf{x})} \tag{1}$$

In this formula $P(\omega_i)$ is the a priori probability of class ω_i, $p(\mathbf{x} \mid \omega_i)$ is the class-conditional probability density function[2], and $p(\mathbf{x})$ is the pdf of $\mathbf{x}$.

For the sake of simplicity, let us focus on the 2-class case (M=2). Then, using formula (1) transforms the Bayes classification rule as follows: classify $\mathbf{x}$ to ω_1 if the following holds

$$\begin{gathered} P(\omega_1|\mathbf{x}) > P(\omega_2|\mathbf{x}) \Rightarrow \\ \frac{p(\mathbf{x}|\omega_1)P(\omega_1)}{p(\mathbf{x})} > \frac{p(\mathbf{x}|\omega_2)P(\omega_2)}{p(\mathbf{x})} \Rightarrow \\ \frac{p(\mathbf{x}|\omega_1)}{p(\mathbf{x}|\omega_2)} > \frac{P(\omega_2)}{P(\omega_1)} \end{gathered} \tag{2}$$

Let N denote the total number of training vectors (N_1 of which belong to ω_1 and N_2 to ω_2), V the volume of some neighborhood centered at $\mathbf{x}$, and n_1 (n_2) the number of training vectors that belong to ω_1 (ω_2) and reside in the given neighborhood. Then the elements of formula (2) can be estimated as follows [21]:

$$P(\omega_i) \approx \frac{N_i}{N} \text{ and } p(\mathbf{x}|\omega_i) \approx \frac{1}{V} \times \frac{n_i}{N_i}\ (i \in [1, 2])$$

These estimators converge to the real values when $N_i \to \infty$, provided that $V \to 0$, $n_i \to \infty$, and $\frac{n_i}{N_i} \to 0$ ($i \in [1, 2]$) at the same time.

In other words, these conditions indicate that using the aforementioned estimators is more reliable when (a) the number of training vectors is very large, (b) the neighborhood around $\mathbf{x}$ is rather small, and (c) there is a large number of training vectors within the borders of the neighborhood, which is though much smaller than the total number of training vectors.

Replacing the elements of formula (2) with their estimators gives:

$$\frac{\frac{1}{V} \times \frac{n_1}{N_1}}{\frac{1}{V} \times \frac{n_2}{N_2}} > \frac{\frac{N_2}{N}}{\frac{N_1}{N}} \Rightarrow n_1 > n_2$$

[2] We assume that $\mathbf{x}$ can take any value in the D-dimensional feature space. In the case that feature vectors can take only discrete values, pdfs become probabilities.

The last inequality implies classifying $\mathbf{x}$ to ω_1 if $n_1 > n_2$. This simplified criterion is an estimator of the Bayes classification rule and can be generalized for the multi-class case as follows: classify $\mathbf{x}$ to ω_i if $n_i > n_j \ \forall \ j \in [1, M]$ such that $j \neq i$.

This result proves that the intuitive classifier described in the previous subsection, which simply relies on counting training vectors that fall inside a given neighborhood around $\mathbf{x}$, converges to the optimal Bayes classifier, which minimizes the classification error probability.

Let us revisit the pdf estimator:

$$p(\mathbf{x}|\omega_i) \approx \frac{1}{V} \times \frac{n_i}{N_i}$$

This can be written as

$$p(\mathbf{x}|\omega_i) \approx \frac{1}{V} \times \frac{\sum_{j=1}^{N_i} \phi(\mathbf{x}_j)}{N_i} \tag{3}$$

where $\phi(\mathbf{x}_j)$ is a kernel function that returns 1 if $\mathbf{x}_j$ resides in the given neighborhood, or 0 otherwise. In the literature [21], there are smoother kernel functions that better estimate continuous pdfs. However, in our case count seems to be enough, since it is easy to implement and our goal is not to find the absolute values of pdfs, but to identify the class that maximizes the pdf in the given neighborhood.

3.3 Disk-Based Implementation

LOCUS is based on the criterion described above fixing a small neighborhood around an incoming feature vector $\mathbf{x}$ and counting the number of its neighbors for every class in a lazy fashion. The majority class wins. Note that this strategy differs from that of the K-Nearest Neighbors, where the number of neighbors is fixed instead of the volume of the neighborhood. As we have shown, this intuitive alternative converges to an optimal solution. Furthermore, lazily counting seems to be very attractive, due to its simplicity. However, a naive implementation can be very expensive in terms of memory requirements and I/O costs. Recall that, formula (3) includes an invocation of the kernel function ϕ for every training vector. This implies a complete scan of the training set for every incoming unknown object, which is unacceptable for very large datasets, on which we focus.

A straightforward solution would be to apply sampling in order to keep only a small fragment of the original data that fits in main memory. However, as we have shown, LOCUS converges to optimality when the underlying dataset is as large as possible. Hence, sampling is out of question.

An alternative solution is to implement LOCUS over a DBMS using standard SQL. This approach is popular in data mining and it has been shown effective in different mining tasks [10]. Note that counting neighbors in a small area can be easily transformed into a highly selective range query that accesses directly the feature vectors (stored as tuples) that reside in the given ranges. Such queries

have been thoroughly studied in the database literature and existing query optimizers guarantee high efficiency with the use of traditional indexing techniques. This property actually turns LOCUS into an efficient disk-based approach that is database-friendly.

Back to the example in Fig. 1, assume that known objects are stored in a relation R with the following schema R(f_1, f_2, class). Then the number of objects of each class that fall within the given volume can be found by the following query:

```
SELECT class, count(*)
FROM R
WHERE f1 ≥ x1-δ1 AND f1 ≤ x1+δ1 AND f2 ≥ x2-δ2 AND f2 ≤ x2+δ2
GROUP BY class
```

Since the volume V is chosen small so that $n_i/N_i \to 0$, the number of tuples accessed by the query above is actually orders of magnitude smaller than the total number of tuples in R. This property makes LOCUS scalable, enabling it work over very large datasets, which minimizes the classification error probability, as shown in the previous subsection.

If in our previous example feature f_2 was categorical with no ordering defined for its values, the corresponding query would be:

```
SELECT class, count(*)
FROM R
WHERE f1 ≥ x1-δ1 AND f1 ≤ x1+δ1 AND f2=x2
GROUP BY class
```

As we have already explained, the existence of categorical features does not generate problems.

3.4 Selection of Neighborhood Volume

As shown in Section 3.2, a proper value for the neighborhood volume V depends on the characteristics of the underlying training set. Clearly, V must be smaller when the dataset is denser and vice versa. A dataset becomes denser as (a) the size N of the training set increases, (b) the number of relevant features D decreases, and (c) the number of different values of every feature C_i decreases.

Choosing a proper value for the neighborhood volume V can be automated by using cross-validation for minimizing the classification error. Generally, if V is too small the result set of the queries used by LOCUS is empty and classification accuracy is marginal. The execution time of such queries is very fast, since they actually access a limited number of tuples, if any, making initial experimentation with small values of V very cheap. As V increases, the probability that a result set is empty decreases and this can be easily observed, since in this case, LOCUS starts returning useful results and classification accuracy increases. Finally, when V becomes too large, a considerable proportion of feature vectors lies in the

corresponding neighborhood, which invalidates the preconditions of optimality and results once more in a decrease of classification accuracy overall. Hence, we propose selecting V with cross-validation, i.e., by identifying a value that strikes a balance between the two trends and minimizes classification error.

4 Experimental Evaluation

To evaluate the efficiency of the proposed techniques, we have compared LOCUS with Decision Trees (DTs) [18], as they have been popular and effective, and have been widely used for benchmarking in the past. The actual implementation of DTs we have used is J48, which is a variation of C4.5 [18] offered in weka [22], a standard, open-source suite of machine learning algorithms. We have also run some initial experiments with Nearest-Neighbors, implemented in weka as well. Its scalability has been found poor, even for moderate-sized datasets, and its accuracy comparable to that of DTs. Hence, we have excluded it from further investigation. We have implemented LOCUS in C++ and we have used an open-source DBMS for query processing. We have run our experiments on a Pentium 4 (2.8 GHz) PC with 512 MB memory under Windows XP. Below, we present the results of our experimental evaluation of the algorithms of interest over appropriate subsets of the features of both synthetic and real-world datasets.

Synthetic datasets: Due to lack of publicly available very large real-world datasets we have used ten functions, first proposed elsewhere [2], that have been widely used in the past for generating synthetic data proper for evaluating disk-based classification algorithms. The resulting datasets consist of nine predictor attributes (6 numerical, 3 categorical) of various domain sizes and two classes. Please refer to the bibliography [2] for more information.

We have generated datasets of various sizes and devoted a reasonable size of 200 MB of memory to memory-based DTs, which have managed to run over small and medium datasets (of 5×10^3 and 5×10^4 tuples) but not over larger ones (of 5×10^5 tuples or more). We have further tested LOCUS with datasets two orders of magnitude beyond this limit (of 5×10^5 and 5×10^6 tuples). All test sets have consisted of 10^3 tuples. Below, we illustrate the most indicative results.

Fig. 2 shows the error rates generated by LOCUS and DTs, respectively, for medium datasets (N=5×10^4 tuples) generated by the ten functions mentioned above. We see that LOCUS wins in four cases (2, 7, 8, 9), is equal with DTs in three (1, 3, 10), and loses in three (4, 5, 6). Its greatest success over DTs occurs in function 2, which is complicated enough to prevent DTs from building a global model. LOCUS overcomes this based on local information only.

Expectedly, the accuracy of LOCUS improves converging to optimality as datasets become larger. We show this in Fig. 3, which presents the error rates generated by LOCUS for all ten functions when the number of training tuples varied from 5×10^3 to 5×10^6. Clearly, error rates tend to decrease and finally LOCUS reaches the accuracy of DTs for all cases it lost in Fig. 2. These results

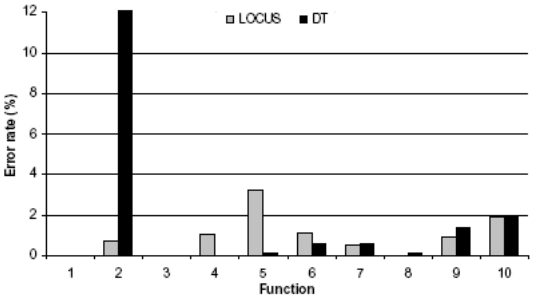

Fig. 2. Error rate using synthetic data (N=5×10^4)

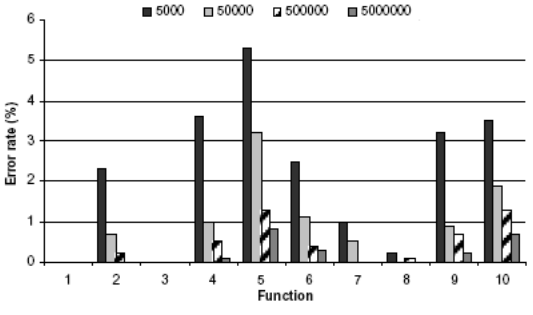

Fig. 3. Error rate using synthetic data wrt dataset size

show the potentiality of LOCUS, mainly in very large datasets, for all of which error rates have dropped under 1%.

As an example of how the prerequisites (mentioned in Section 3.2) for convergence to Bayes optimality hold when the size of the training dataset grows, Fig. 4 shows the size of the neighborhood volume V used by LOCUS as a function of the number of tuples stored in the database. The specific volumes have been found with cross-validation over datasets generated with function 5 (behavior is similar for all ten functions). The values indicated are the normalized volumes (i.e., the fractions of the volumes over the volume corresponding to the largest training set) and clearly show that V tends to zero as the size of the dataset increases.

Fig. 5 illustrates the scalability of LOCUS with respect to the number of tuples stored in the database (x-axis is logarithmic). Each point represents the average time for making a decision. Averages refer to all ten functions. Performing classification in approximately half a second over datasets consisting of 5 $\times$ 10^6 tuples is very promising. Note that in these experiments we have used a single B^+-Tree for each dataset, indexing the corresponding attribute with the largest domain. Optimistically, since the classification algorithm can be applied directly over the database relation R that holds the original data, it is highly likely that R is already indexed for other purposes, implying that effective reuse

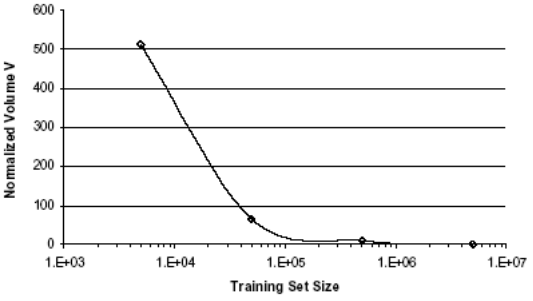

Fig. 4. Normalized volume V used by LOCUS on synthetic data wrt dataset size

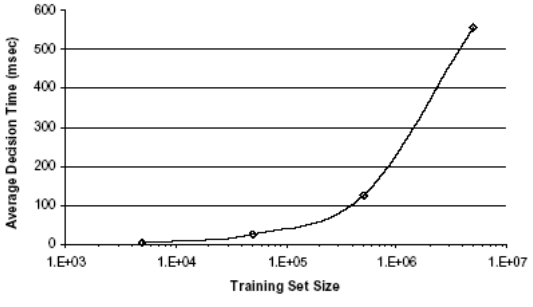

Fig. 5. Time scalability of LOCUS on synthetic data wrt dataset size

Dataset	F	N	M
Patient	8	90	3
Glass	9	214	7
Liver	6	345	2
BreastCancer	9	699	2
Diabetes	8	768	2
Letters	16	2×10^4	26
CovType 5000	10	5×10^3	7
CovType 50000	10	5×10^4	7
CovType 500000	10	5×10^5	7

Fig. 6. Properties of the real datasets we have used

of existing resources may just be enough. On the other hand, multidimensional index structures like R-trees and techniques like pre-sorting stored data or materializing views could be used to generate better results. The effect of these techniques in this case is identical to that when applied to any (simple) queries expressed in standard SQL, so it is orthogonal to and beyond the scope of this paper.

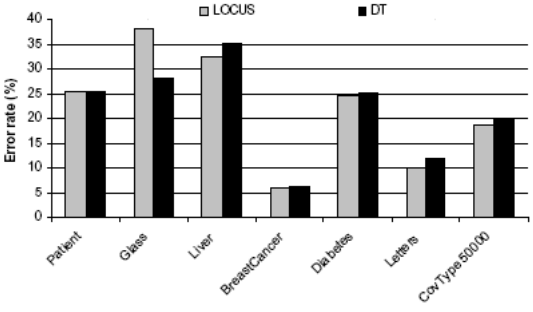

Fig. 7. Error rate using real datasets

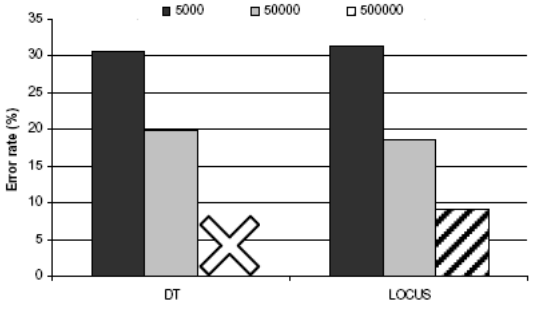

Fig. 8. Error rate using CovType wrt dataset size

Real datasets: As we have already mentioned, LOCUS behaves best under large datasets; hence, testing its behavior over real data, which are rather small, is really challenging. In our study, we have chosen commonly used publicly available real-world datasets [16]. Fig. 6 shows the properties of the real datasets we have used (number of features F, number of tuples N, and number of classes M, respectively). Datasets appear ordered according to the average number of tuples per class they contain. We expect better results as this number increases. In small datasets (up to Diabetes), we have treated 2/3 of all tuples as known and 1/3 as unknown (separation has been random), which is common. In Letters and CovType we have used 1,000 unknown tuples instead. All results are averages of five different experiments per dataset.

Fig. 7 shows the error rates generated by LOCUS and DTs respectively. Error rates are in general high over most of these datasets, which exhibit inherent difficulty in being used for prediction tasks. LOCUS performs worst over Glass, which is too sparse, consisting of only 214 tuples for 7 classes. Nevertheless, surprisingly, it outperforms DTs in most cases overall, even over very small datasets. As datasets become larger (mainly in Letters and CovType) its superiority increases. This is also clear in Fig. 8, which shows the accuracy improvements over CovType with respect to the training set size. The symbol "X" denotes that DTs failed to deal with the case of 5×10^5 tuples.

Overall, we have shown that LOCUS is scalable, its accuracy is comparable to that of eager DTs in small datasets and becomes superior when datasets become larger. Hence, we have provided strong evidence that it is a promising classifier, mainly suited for datasets with large and constantly growing sizes.

5 Conclusions and Future Work

In this paper, we proposed LOCUS, an accurate and efficient disk-based lazy classifier that is data-scalable and can be implemented using standard SQL. We have shown that in most cases it exhibits high classification accuracy, which improves as training sets become larger, based on its convergence to the optimal Bayes. Overall, the results are very promising with respect to the potential of LOCUS as the basis for classification, mainly over large or inherently complex datasets.

Note that, in this thread of our work, we have focused on classification scalability with respect to the number of known vectors N and have deliberately neglected scalability with respect to the number of dimensions D. The latter problem is tightly related to methods for feature selection, which are orthogonal to our work presented here. In the future, we plan to investigate the applicability of similar techniques for feature selection as well. Furthermore, we intend to implement a parallel version of LOCUS and study its effectiveness on regression problems (possibly replacing the "count" aggregate function with "average").

References

1. Agrawal, R., Ghosh, S.P., Imielinski, T., Iyer, B.R., Swami, A.N.: An Interval Classifier for Database Mining Applications. In: VLDB 1992 (1992)
2. Agrawal, R., Imielinski, T., Swami, A.N.: Database Mining: A Performance Perspective. IEEE Trans. Knowl. Data Eng. 5(6), 914–925 (1993)
3. Aha, D.W., Kibler, D.F., Albert, M.K.: Instance-Based Learning Algorithms. Machine Learning 6, 37–66 (1991)
4. Beyer, K.S., Goldstein, J., Ramakrishnan, R., Shaft, U.: When Is "Nearest Neighbor" Meaningful? In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, Springer, Heidelberg (1998)
5. Burges, C.J.C.: A Tutorial on Support Vector Machines for Pattern Recognition. Data Min. Knowl. Discov. 2(2), 121–167 (1998)
6. Chen, M.S., Han, J., Yu, P.S.: Data Mining: An Overview from a Database Perspective. IEEE Trans. Knowl. Data Eng. 8(6), 866–883 (1996)
7. Friedman, J.H., Kohavi, R., Yun, Y.: Lazy Decision Trees. In: AAAI/IAAI, vol. 1, pp. 717–724 (1996)
8. Gehrke, J., Ganti, V., Ramakrishnan, R., Loh, W.Y.: BOAT-Optimistic Decision Tree Construction. In: SIGMOD 1999 (1999)
9. Gehrke, J., Ramakrishnan, R., Ganti, V.: RainForest - A Framework for Fast Decision Tree Construction of Large Datasets. In: VLDB 1998 (1998)
10. John, G.H., Lent, B.: SIPping from the Data Firehose. In: KDD 1997 (1997)
11. Kamber, M., Winstone, L., Gon, W., Han, J.: Generalization and Decision Tree Induction: Efficient Classification in Data Mining. In: RIDE 1997 (1997)

12. Katayama, N., Satoh, S.: The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In: SIGMOD 1997 (1997)
13. Mehta, M., Agrawal, R., Rissanen, J.: SLIQ: A Fast Scalable Classifier for Data Mining. In: Apers, P.M.G., Bouzeghoub, M., Gardarin, G. (eds.) EDBT 1996. LNCS, vol. 1057, Springer, Heidelberg (1996)
14. Melli, G.: A Lazy Model-Based Algorithm for On-Line Classification. In: Zhong, N., Zhou, L. (eds.) Methodologies for Knowledge Discovery and Data Mining. LNCS (LNAI), vol. 1574, Springer, Heidelberg (1999)
15. Mitchel, T.: Machine Learning. McGraw-Hill, New York (1997)
16. Newman, D.J., Hettich, S., Blake, C.L., Merz, C.J.: UCI Repository of machine learning databases, `http://www.ics.uci.edu/~mlearn/MLRepository.html`
17. Provost, F.J., Kolluri, V.: A Survey of Methods for Scaling Up Inductive Algorithms. Data Min. Knowl. Discov. 3(2), 131–169 (1999)
18. Quinlan, J.R.: Induction of Decision Trees. Machine Learning 1(1), 81–106 (1986)
19. Shafer, J.C., Agrawal, R., Mehta, M.: SPRINT: A Scalable Parallel Classifier for Data Mining. In: VLDB 1996 (1996)
20. Shaft, U., Ramakrishnan, R.: When Is Nearest Neighbors Indexable? In: Eiter, T., Libkin, L. (eds.) ICDT 2005. LNCS, vol. 3363, Springer, Heidelberg (2004)
21. Theodoridis, S., Koutroumbas, K.: Pattern Recognition, 3rd edn. Academic Press, London (2005)
22. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)
23. Yu, H., Yang, J., Han, J.: Classifying large data sets using SVMs with hierarchical clusters. In: KDD 2003 (2003)
24. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: An Efficient Data Clustering Method for Very Large Databases. In: SIGMOD 1996 (1996)