6th SIAM Conf. on Data Mining (SDM 06), Bethesda, MD, 2006

Advanced Prototype Machines: Exploring Prototypes for Classification

Hans-Peter Kriegel Matthias Schubert

Institute for Computer Science
University of Munich, Germany
{kriegel,schubert}@dbs.ifi.lmu.de

Abstract

In this paper, we propose advanced prototype machines (APMs). APMs model classes as small sets of highly descriptive prototypes which are well suited for interactive visualization. Thus, APMs offer a method to analyze class models, feature spaces and particular classification scenarios. To derive the prototypes, we introduce "Push and Grow", a classification algorithm which is based on a quality measure favoring maximal margins between classes. To explore the derived prototypes, we propose a visualization suite that adapts interactive multi-dimensional scaling to prototype models. The idea of this tool is to display the distance relationships between the prototypes and the objects to be classified. We distinguish three visualization tasks deriving different kinds of information. To shift the visualization error to the less important distance relationships as much as possible, the stress function is adjusted to each of these tasks. APMs achieve fast and accurate classification that is based on compact class models which can be explored by interactive visualization. Our experimental evaluation demonstrates on 14 data sets that APMs achieve better classification accuracy on much less data objects than other kNN-based classifiers. To demonstrate the value of our interactive exploration tool, we provide examples for the derived class models and classification scenarios.

Keywords

classification, visual data mining, multi-dimensional scaling, prototype models

1 Introduction

The most desirable ability of a good classifier is to offer reliable class predictions. However, in many applications another aspect of classification is very important as well, the understandability of the found class models. For example, a manager using a decision support system wants to understand the characteristics of the situation that led to the decision proposed by the computer. Thus, finding a class model that is understandable is an important aspect of designing a classification algorithm. However, most directions of classification neglect this aspect in favor of superior accuracy.

In this paper, we propose, advanced prototype machines (APM), an alternative approach to understandable classification that is based on nearest prototype classification and data visualization. The idea is to describe each class by a small set of highly descriptive prototypes. Each prototype represents a subset of the original training objects. Furthermore, the prototypes represent necessary concepts for each class and thus we can visualize the classes without the influence of noise objects. APMs consist of two components. A new algorithm, called "Push and Grow" (PAG) to derive prototype models which is focused on finding very small prototype models. Since the number of prototypes is very small, it is much easier to find a meaningful visualization. The second component of an APM is a visualization suite that adapts interactive multi-dimensional scaling on a hyperbolic plane to displaying prototype models. We distinguish three tasks of visualization: the visualization of feature spaces, prototype models and individual class decisions. Since not all distance relationships in a prototype model are equally important for each of these scenarios, we introduce different weightings to shift the visualization error to the less important distance relationships for that particular scenario. Additionally, the visualization suite allows to directly compare the feature vectors of objects and to analyze specific classification scenarios more closely.

The rest of the paper is organized as follows. In section 2, we will briefly survey related work. Section 3 provides an overview of the proposed system and makes some basic definitions. Section 3 describes the new classification algorithm called "Push and Grow". Weighted multi-dimensional scaling and its application to display class knowledge is introduced in section 5. Furthermore, this section contains a description of the features of the proposed visualization suite. The experimental evalu-

ation in section 5.3 describes the classification performance of "Push and Grow" compared to other classification algorithms, on multiple data sets. Furthermore, the section contains some examples for the usefulness of the introduced visualization tool. The paper concludes in section 7 with a short summary and some ideas for future work.

2 Related Work

The simplest method of prototype classification is centroid based classification as proposed in [7]. A more general approach to nearest prototype classification was proposed by Kohonen [9, 10]. Similar to his clustering approach, self-organizing-maps, he proposed "learning vector quantization" (LVQ) for optimizing the position of a given set of artificial prototypes with respect to nearest neighbor classification. Basically, LVQ performs the same task as the pushing step in our proposed algorithm PAG. However, LVQ does not determine the number of employed prototypes for each class on its own. Thus, it solves only one part of the problem. Furthermore, LVQ moves centroids with respect to the distance of a data object to the two closest prototypes. In our approach, we optimize the positions of prototypes with respect to the breadth of the margin between the classes, which is the core idea of our quality measure. To demonstrate the advantages of our method, we will compare it to a version of our method called "LVQ and Grow" that employs LVQ instead of the Push step in PAG.

Another method to shrink the training set of kNN classifiers is instance reduction [2, 15, 3]. All approaches try to reduce the number of instances in the training set in a way that the classifier provides comparable or even better accuracy and demands less processing time. Recent evaluations indicate that the methods RT-3 [15] and ICF [3] currently provide the best results. However, these methods delete instances from the training set and do not build artificial prototypes.

To extract knowledge from the found prototype models, we use visualization to get a good intuition about the classes. Though prototype models use only a fraction of the number of data objects of the original training set, they are still feature vectors in the original data space. Thus, prototypes are often very high-dimensional. To visualize high-dimensional feature vectors, we employ regularized multi-dimensional scaling. General multi-dimensional scaling was first proposed by Samon [13]. Since the data objects that MDS projects into a the 2D image space might provide very heterogeneous distances, zooming becomes an important problem. This problem was solved by Walters and Ritter in [14] proposing H-MDS, who solved the zooming problem by calculating the disparities of objects in an hyperbolic object space. Displaying the image point on a hyperbolic plane allows interactive zooming of very far and near objects. A difference to our use of MDS on a hyperbolic plane is that we first apply MDS and than transform the resulting configuration into the hyperbolic plane while H-MDS in [14] directly works in a hyperbolic space.

Since MDS was introduced for visualizing general high dimensional data sets, the distance relationships of all displayed data objects are considered as equally important. However, in the prototype model of an APM we have different types of objects, prototypes and test instances, and different types of distance relationships, between objects of different classes and between objects of the same class. Depending on the given visualization task, some of the distance relationships are less important. Therefore, we introduce different weightings that increases the influence of the more important distances to the stress function in MDS. Thus, our visualization suite is capable to adjust the image of the prototype model to the given scenario.

3 Describing Class Knowledge as Prototypes

In this section, we want to give a brief overview of the described classification approach. We model classes as sets of prototypes or prototype models. Each prototype represents a subset of the training objects for one class and is basically a weighted centroid that is built from a set of feature vectors. Formally, a prototype is defined as follows:

DEFINITION 3.1. ***prototype***
Let $S_i \subset \mathbb{R}^d$ be a set of training objects for class c and $w(o)$ be a weight function determining the importance for this object $o \in S_i$ in the prototype. The prototype $PT(S_i)$ is defined as:

$$PT(S_i) = \frac{\sum_{o \in S_i} w(o) \cdot o}{\sum_{o \in S_i} w(o)}$$

DEFINITION 3.2. ***prototype model***
Let $S = \{S_1, \ldots, S_n\}$ be a disjoint decomposition of the set $TR_c = \bigcup_{S_i \in S} S_i$ of all training objects for class $c \in C$. Then $PM_c(S) = \{x_i | \forall S_i \in S : x_i = PT(S_i)\}$ is called a prototype model of c with respect to S. A set of prototype models for a given training set TR belonging to classes C is denoted as $PMs_C(TR)$.

A prototype model is a condensed version of the training data for one class. Since a prototype is usually built from several objects, it is a generalization and thus contains information about what is typical for

the underlying training objects. A good prototype model should contain a well-positioned prototype for all training objects that are based on a similar concept. Nearest neighbor classification is handled the same way as on the original training set. However, since an object usually is dependent on a single mechanism, we consider only the nearest neighbor for each class. As distance measure, we employ the well established Euclidian distance on feature vectors $fv \in \mathbb{R}^d$.

Another aspect of prototypes is that they are well suited for visualization. First of all, it is possible to find very small prototype models for most applications, containing only a fraction of the number of original training objects. Thus, a visualization of prototype models is faster to calculate, more accurate with respect to visualization errors and easier to understand for a human user. The simplest method for deriving a set of prototype models is to describe each class by the unweighted centroid of the training objects. In [7] this method was described to perform well for text classification.

A more sophisticated method for deriving prototype models are clustering algorithms like k-Means, DBSCAN or EM [8]. By clustering the training objects for each of the classes and condense the resulting clusters to their centroids, it is possible to find a set of meaningful centroids for each class. The problem of this approach is that the resulting prototype models describe the training set without considering any relationships to other classes. Thus, the location of prototypes mirrors only the data distribution in the training set of a class. Another problem with using clustering algorithms for generating prototypes is that most clustering algorithms are dependent on the given input parameters. k-Means for example, needs the expected number of clusters as input parameter. This has the advantage that we can select the size of the resulting prototype model for each class. However, since the number of necessary objects for each class is unknown, it is very difficult to find a suitable parametrization for each of the classes.

4 Push and Grow

In this section, we will introduce a new algorithm to find small prototype models that achieve high classification accuracy. Our algorithm is driven by two important observations of classification. The first is minimum description length. A classifier depending on a small number of parameters has a smaller tendency to overfit than a classifier using complex class models. For prototype models, this means that a classifier depending on a small number of prototypes is considered to be described by less information. The second observation is that maximizing the margin between the classes improves the performance of the classifier on unknown data objects. This second observation is one of the main ideas of support vector machines as proposed in [5].

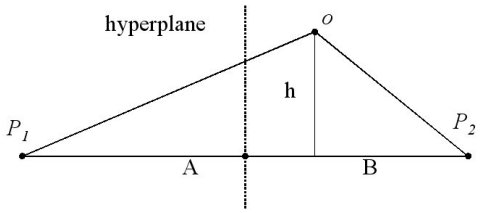

Figure 1: Sketch for proof of lemma 1.

4.1 The Quality of Prototype Models

Before describing the algorithm, we will define a quality measure for sets of prototype models that is based on the second observation. For each object o, we calculate the distance to the closest class border by calculating the distance to each hyperplane that is defined by a pair of prototypes (P_1, P_2) where P_1 belongs to c and P_2 belongs to any of the other classes $\hat{c} \in C \setminus c$. Let us note that the hyperplane that is given by two prototypes can be derived by taking the vector between both prototypes as normal vector of the plane. Additionally, the point in the middle of this vector must be placed on the separating hyperplane. To compute the distance of object o to a hyperplane that is given by prototypes P_1, P_2, we introduce the following lemma.

LEMMA 4.1. *Let o be an object belonging to class $c \in C$ and let $PMs_C(TR)$ be a set of prototype models with respect to a training set TR. Furthermore, let P_1, P_2 be prototypes with $P_1 \in PM(Tr_c)$ and $P_2 \in PM(Tr_{\hat{c}})$ with $\hat{c} \in C \setminus c$ and $d(x_1, x_2)$ be the Euclidian distance in $\mathbb{R}^d$. Then, $dist_{P_1,P_2}(o)$ is the distance of o to the hyperplane that is defined by P_1 and P_2:*

$$dist_{P_1,P_2}(o) = \frac{d(P_1,P_2)^2 + d(P_1,o)^2 - d(P_2,o)^2}{2d(P_1,C_2)} - \frac{d(P_1,P_2)}{2}$$

Proof. cf. figure 1

Since $d(P_1,o)^2 = A^2 + h^2$ and $d(P_2,o)^2 = B^2 + h^2$, we can derive that $d(P_1,o)^2 = A^2 - B^2 + d(P_2,o)^2$. Furthermore, we know that $A + B = d(P_1,P_2)$. Together, $d(P_1,o)^2 = d(P_1,P_2) \cdot (2A - d(P_1,P_2))) + d(P_2,o)^2$.

Thus

$$A = \frac{d(P_1,o)^2 + d(P_1,P_2)^2 - d(P_2,o)^2}{2 \cdot d(P_1,P_2)}$$

Since we want to calculate the distance to the middle of $\overline{P_1P_2}$, we have to subtract $d(P_1,P_2)/2$. q.e.d.

DEFINITION 4.1. ***Minimal Distance to Class Border***

$$d_{min}^{PMs_C(TR)}(o) = \min_{P_1 \in PM(Tr_c), P_2 \in PM(Tr_{\hat{c}})} dist_{P_1,P_2}(o)$$

where $\hat{c} \in C \setminus c$.

The value $d_{min}^{PMs_C(TR)}(o)$ is positive, if o is placed inside the Voronoi cell of prototype P_1 and negative if it is positioned outside. To derive a confidence value, we apply a sigmoid function to $d_{min}^{PMs_C(TR)}(o)$:

DEFINITION 4.2. ***confidence value***

$$Conf_{PMs_C(TR)}(o) = \frac{1}{1 + \exp^{-\alpha d_{min}^{PMs_C(TR)}(o)}}$$

If $Conf_{PMs_C(TR)}(o)$ displays a value smaller than 0.5, o is placed in a Voronoi cell belonging to the class of P_2 instead of its own class and is thus misclassified. Note that depending on parameter α the confidence $Conf_{PMs_C(TR)}(o)$ does not instantly display a large value if o is placed in the correct Voronoi cell. If o is close to the class border, $Conf_{PMs_C(TR)}(o)$ might still display a value of almost 0.5.

To measure the quality of a given set of prototype models, $Conf_{PMs_C(TR)}(o)$ is determined for every object in the training set and the average is computed.

DEFINITION 4.3. ***quality function***
Let $PMs_C(TR)$ *be a set of prototype models describing the set of classes* C *with respect to a training set* $TR = \{TR_{c_1}, \ldots, TR_{C_m}\}$. *Then, the quality of* $PMs_C(TR)$ *is defined as follows:*

$$quality(PMs_C(TR)) = \frac{\sum_{o \in TR} Conf_{PMs_C(TR)}(o)}{|TR|}$$

This quality measure displays large values if the training objects are correctly classified and have a significant distance to the class border. A set of prototype models having a considerably large portion of misclassifications or many objects that are close to the class margins, will display only a low quality when using this measure. By considering the distance to the closest class border, a set of prototype models achieving a broader margin between classes offers a higher quality as well. Thus, our quality measure favors maximum margins between classes. By defining a quality measure, we now have a measure for comparing different prototype models with respect to their potential to offer accurate classification.

4.2 Push and Grow To derive descriptive models with a minimal number of prototypes, we now introduce a novel algorithm that is based on the quality measure described above. The idea of our algorithm is to begin with a minimum number of prototypes, namely one for each class. Then, we are improving the prototype models by adjusting the positions of the prototypes. This step is called the push step. After we reach a maximum of the quality measure, we split some of the prototypes in the grow step. Afterwards, some of the classes are represented by more than one prototype. Then, we again apply the push step in the next iteration to improve the positioning of the prototypes and split again. The algorithm terminates when the quality of a the new prototype model does not significantly exceed the quality of the previous model. To explain the exact algorithm, we will first describe the push and the grow step separately and afterwards we will present the complete algorithm combining both steps.

The Push Step

The goal of the push step is to find a weighting of the training objects that moves the prototypes into a position increasing the margin between the classes and thus increasing the quality of the set of prototype models.

The push step in "Push and Grow" (PAG) performs several iterations over the training data. In each iteration, the weight of object o is increased with respect to $(1 - Conf_{PMs_C(TR)}(o))$. This way, misclassified objects and objects that are close to the class margin are weighted higher than objects that are classified correctly with high confidence. At the beginning of the push step, the weight of each object is 1. After each iteration, the weight is updated as follows:

$$weight(o) = weight(o) \cdot (2 - Conf_{PMs_C(TR)})(o))$$

This way each weight is steadily increased and the weights of objects that are close to the class borders have an increased influence on the position of the prototype. As a result, the prototypes are slowly moving in the direction of the classification errors. As can be seen from lemma 1, increasing the weight of an object o increases the distance to the class border for correctly classified objects and decreases it if o is on the wrong side of the border. However, since the weights are simultaneously increased for several objects, it is not guaranteed that the distance to the class border is indeed improving for all objects. However, it is improved for the majority of objects. After performing some iterations (about 5-15), the quality measure is more or less stable and does not display any significant improvement.

The Grow step

While the push step tries to find a more suitable

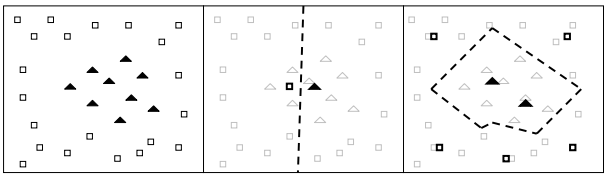

Figure 2: Left: Illustration of two classes (triangles and rectangles). Middle: A prototype model employing a single centroid for each class. Right: A prototype model using several prototypes per class.

position for a given set of prototypes, the grow step tries to find the right number of prototypes. After the algorithm starts with a single prototype for each class and optimizes the position of these prototypes using the push step, a set of prototype models might still display only an unsatisfying quality. For many real-world problems, it is not sufficient to model each class by a single prototype, i.e. a single Voronoi cell. Figure 2 displays a 2D example where a single prototype per class is insufficient. However, by choosing several prototypes it is possible to find a good approximation of the class border. Thus, for some applications simply moving the prototypes cannot achieve a good classification model. When classifying more complex classes, the number of prototypes representing some of the classes has to be increased. Let us note that the same problem occurs for other linear classification methods like support vector machines using a linear kernel or linear discriminant analysis.

First of all, the grow step selects certain prototypes to be split. Therefore, the quality of the complete set of prototype models is compared to the quality of each of the single prototypes. If the quality per object of a prototype is smaller than the quality per object for the complete set of prototype models, the prototype is selected for splitting. This heuristic is based on the assumption that classification errors are caused by prototypes that do not model the underlying training objects well enough.

To split a cluster, we proceed as following. We determine the two data objects having the largest distance from each other. Afterwards, we assign the remaining data points to the closer of these two objects and build prototypes from the resulting groups. Then, we reassign each point to the closer prototype and again calculate the prototypes. The algorithm terminates if the prototypes are stable for two iterations. The algorithm is basically k-means ($k = 2$) [8] with one small modification: we use the weights determined in the push step instead of using unweighted mean values. The advantage of this approach is that the resulting clusters are built with respect to the misclassified and difficult objects.

```
algorithm push(PM[], TrainSet[])
  initWeights()
   old_q = ∞
   oldPM[] = PM[]
  while(quality(PM[]) > old_q)
    old_q = quality(PM[])
     oldPM[] = PM[]
    foreach pt in TrainSet[] do
      pt.setWeight(weight(pt) · 2 − Conf_PM[](pt))
    next
  endwhile
  return oldPM[]
end.

algorithm PAG(TrainSet[])
  // init PrototypeModels PM[]
  for each class c do
    insert centroid(TrainSet[c]) into PM[c]
  next
  PM[] = push(PM[])
  //start PAG
  old_q = ∞
  oldPM[] = PM[]
  while(quality(PM[] − old_q > ϵ) do
    old_q = quality(PM[])
    oldPM[] = PM[]
    //grow step
    candidates = calculateSplitCandidates(PM[])
    foreach cand in candidates do
      remove cand fromPM[]
      newProtoypes[] = split(cand)
      insert(newProtoypes[]) into PM[]
    next
    //Push Step
    PM[] = push(PM[])
  endwhile
  return oldPM[]
end.
```

Figure 3: Pseudo code of Push & Grow.

Figure 3 displays the complete algorithm in pseudo code. PAG starts by building the centroids for each class. Afterwards the Push step improves the position of the prototypes with respect to the quality measures. After the quality cannot be improved any further with the push step, any candidate having a quality that is below the quality of the complete set of prototypes is split in the grow step. Afterwards the weights are reset and the push step is started again. The algorithm terminates if the quality does not improve after an additional iteration of growing and pushing. Thus, increasing the number of prototypes is only accepted

if the margin between the classes is increased and the error confidence is reduced. In section 5.3, we will demonstrate that PAG is capable to derive very small prototype models that display high accuracy compared to other methods of classification and kNN classification in particular.

5 Visualizing Prototype Models

After deriving small and descriptive prototype models, the next step is to exploit these class models for interactive exploration of classification knowledge. Therefore, we will adapt multi-dimensional scaling on the hyperbolic plane to displaying prototype models. After generally introducing this visualization technique, we will turn to describe several possibilities for interactive exploration of prototype models and adjust the visualization to these scenarios.

5.1 Weighted Multi-dimensional Scaling Our goal is to visualize a set of feature vectors of arbitrary dimensions on a two dimensional canvas or a screen. For most high-dimensional data sets, it is impossible to find a two dimensional image that exactly mirrors the distance relationships between each of the data points. However, to get a useful impression of a data distribution, it is often enough to display an image that is resembling the distance relationships as good as possible. Multi-dimensional scaling (MDS) as proposed by Samon [13], uses optimization techniques to move a set of 2D image points I into positions that resemble the distance relationships for a given data set DB in a best possible way. The principle idea of the algorithm is to minimize a so-called stress function that compares the distance matrix of DB to the distance matrix of the image points I. If both matrices are the same with respect to some scaling factor the stress is 0 and the mapping is optimal. The stress functions used so far were designed for visualizing a set of data objects that are equally important. However, in our application of visualizing prototype models, the distance relationships that have to be displayed, provide different levels of importance and thus not all distances should influence the stress function in the same way. For example, since the class membership of a test object only depends on the prototypes, the distances between prototypes and test objects should influence the stress function to a higher degree than the distances to the other test objects. To model such influences, we introduce a weight factor $w_{i,j}$ for each distance in the distance matrix. These weights are set with respect to the individual visualization task and will be discussed in the next subsection. To achieve a better distribution of the data points for visualization, we use quadratic Euclidian distance for calculating the distance matrix on the original data points.

DEFINITION 5.1. ***weighted stress function***
Let D be the distance matrix over all elements $o \in DB \subset \mathbb{R}^n$ where $D_{i,j} = d(o_i, o_j)^2$ and let $f^t(o) : DB \to I$ be a function assigning each element $o \in DB$ an image point $i \in \mathbb{R}^2$ at time t. Furthermore, let W be an $|D| \times |D|$ weight matrix with $w_{i,j} = w_{j,i}$ and $w_{i,j} \geq 0$. The weighted stress function σ_W for f^t is defined as follows:

$$\sigma_W(f^t) = \sum_{i=1}^{n} \sum_{j>i} \frac{w_{i,j} \cdot (d(f^t(o_i), f^t(o_j)) - D_{i,j})^2}{D_{i,j}}$$

An optimal mapping f^{opt} can now be considered as a mapping providing a minimal value for the stress function σ_W. The weights in our distance function now achieve that some of the considered distances have an increased or decreased influence on the quality of the visualization. To calculate a good approximation of f^{opt} even for complex cases, we now apply the following iteration method that is based on gradient descent:

We start with an arbitrary mapping of the data object f^0 and calculate the stress function. Afterwards, we derive a new position for each image point $f^0(o_i)$ by adding a correction factor $V_{o_i,d}$ to each of the d components. $V_{o_i,d}$ is defined as following:

$$V^t_{o_i,d} = \sum_{j=i}^{|DB|} \left[\left(1 - \frac{D_{i,j}}{d(f^t(o_i), f^t(o_j))} \right) \cdot (o_{j_d} - \bar{o_{i_d}}) \cdot w_{i,j} \right]$$

Formally, the d-th component of the image point $f^t(o_i)$ is calculated for iteration $t+1$ in the following way.

$$f^{t+1}(o_i)_d = f^t(o_i)_d + V^t_{o_i,d}$$

After the update is calculated for each image point, the stress function σ_1 is calculated again. The optimization ends if the last iteration was not able to decrease the stress function. Let use note that this method does not necessarily find the global minimum. Thus, we have to rerun the procedure several times with varying start configurations and than choose the method providing the smallest stress value.

5.2 MDS on the Hyperbolic Plane Though MDS usually finds useful mappings into an two dimensional image space, we still might not be able to display these image points in a meaningful way. The distances calculated on the image points might still display a large variation of distance values. Thus, if we want to display all data objects including distant ones, we might not be able to distinguish close objects. On the other hand, if we zoom onto two very close objects, we cannot

display the distant objects at the same time. A method that is capable of overcoming this problem is interactive zooming on a hyperbolic plane as proposed in [14].

The hyperbolic plane (H2) is based on non-Euclidian geometry which does not use the parallel axiom. This axiom states that given a point and a straight line there exists only one parallel line containing the point. The H2 has a constant negative curvature and provides an infinite number of parallel lines through a particular point and for a particular line.

For visualization, the use of the hyperbolic space is very interesting because it allows us to display all data objects and to zoom onto a particular area at the same time. This is achieved by using the Poincaré or fisheye projection. The idea of the Poincaré projection is to shrink the hyperbolic space into the unit circle. The infinite H2 fits into this finite circle by exponentially decreasing the size of displayed objects with the distance to the middle of the circle, which is called the focus point F. Thus, objects in the middle of the circle are zoomed in for closer investigation while all objects are still visible, since even infinitely far away objects would be displayed on the border of the circle. To interactively explore a set of data objects, we can reassign the focus point and thus zoom different areas of the data space.

Let F be the focus point and let x be an image point. The transformation τ of x into the H2 is calculated as follows:

$$\tau(x) = F + \tanh\left(s \cdot \|(F-x)\|\right) \cdot \frac{(F-x)}{\|(F-x)\|}$$

where s is a scaling factor describing the degree of the curvature of space. A more extensive description of interactive zooming on hyperbolic planes, is given in [11]. Now, we can map any 2D data point derived by multi-dimensional scaling into the hyperbolic space.

5.3 Visually Exploring Prototype Models After we generally have described the most important underlying technique of our visualization suite, we will now discuss the information that can be gained by the visual exploration of prototype models and how weights in the stress functions should be set to emphasize the important distances. The knowledge that can be gained from our visualization suite can be categorized into three different levels:

1. **The Feature Space Level**
   At this level, we try to get information about the general characteristics of a chosen feature space. For many applications, the used feature transformation yields a very important influence on classification. If the provided features are not somehow correlated to the classes of the objects, even very sophisticated classification algorithms will not find an accurate class model. Thus, making observations of a feature space without respecting a special classification paradigm, yields important implications for classification.

2. **The Class Level**
   This level deals with analyzing the model for each class. The key question here is: What is characteristic for each class and what features are crucial when distinguishing the classes. The rules derived by decision trees partly answer this question because they describe the class borders. However, a decision rule describes the class borders and thus, this description might not be very informative if we want to find out what is typical for a certain class.

3. **The Object Level**
   The object level deals with particular classification decisions. What is most important here is to answer, why a certain object has been classified to a certain class. Another important question is, how likely is it that the classifier made a mistake. By analyzing particular decisions, we also achieve a better understanding of the classes.

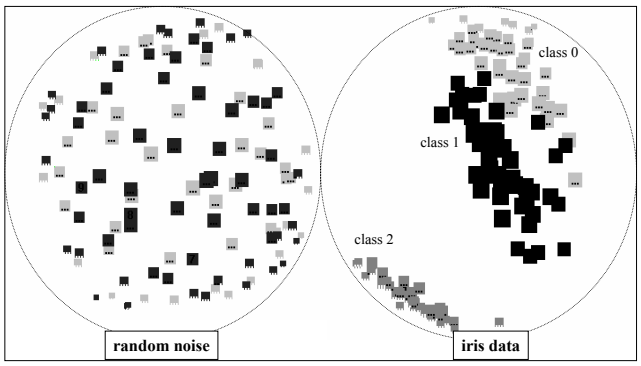

Figure 4: Visualization of two complete training sets. The left set can be recognized as noise. The right data set is the well-known iris set that displays well defined class borders.

Our proposed visualization suite allows to display training and test objects offering the following possibilities.

To gain knowledge about the complete feature space, we can display the complete training set using MDS. Since we want to judge the characteristics of the complete feature space, we apply a uniform weight of 1 to each distance relationship. We use the same weight regardless of the class or type of each displayed object

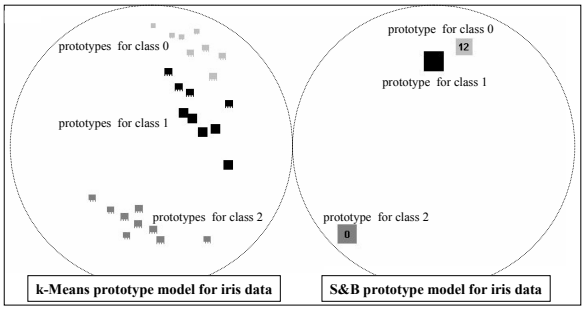

Figure 5: Two different compression levels for the iris data. Left: 10 times compressed data with k-means ($k = 10\% \cdot |TR_c|$). Right: Prototype model derived by PAG.

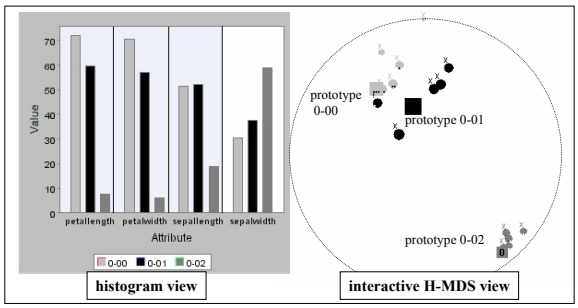

Figure 6: Exploring prototype and class knowledge with APMs.

because shifting the error to a certain kind of distance might mislead the viewer. For example, emphasizing the distances between the objects of different classes might cause the impression the data objects are well-separable even if they are not. If the number of training objects gets rather large, we also offer the possibility to reduce the training objects to a number that is easier to visualize and examine. We achieve this by running k-means clustering on the objects for each class. To determine the parameter k for each class, we multiply the number of data objects for that class with a compression factor. Thus, the percentage of used data objects is the same for each class. Figure 4 shows two example data sets. When looking at the left data set, it becomes quite obvious that the objects of both classes are mixed in all areas of this feature space. Thus, trying out a modified feature transformation, might not be a bad idea. On the other hand, we can recognize that it should be easy to find a classifier separating the classes for the iris data set displayed on the left side.

Though this method often offers useful insight, it is not descriptive enough to explore classification knowledge as demanded on the class level. For this task, displaying the complete training data set offers too many instances for finding out what is typical for a certain class. To allow a general understanding of a class, we need to reduce the training data to the concepts hidden within each class. Therefore, we apply PAG to derive small and highly descriptive prototype models. Due to the small number of prototypes for each class, we can now further analyze each prototype to understand each concept characterizing each class. For this task, we employed a weight of 1 for distances between prototypes of different classes and distances between test objects and prototypes. For the distance of the additional test instances to each other and distances between the prototypes of the same class, we assign a small weight for example 0.0001. These rather small weights are used because the distance relationships of the test instances to each other are usually rather unimportant. On the other hand the distances to prototypes should be modelled as accurately as possible since these distance relationships are responsible for the classification result. Neglecting the distances between the prototypes of the same class, turned out to be beneficial because we are usually not interested in the exact distance of the prototypes for the same class. In our experiments, it turned out that the distances to the prototypes of other classes are usually sufficient for understanding the structure of classes. However, if we want to get an impression of the relative distance of the prototype within the same class as well, we can increase the weight to 1 as well.

In figure 5, we compare the prototype models derived by k-Means for the iris data set to those derived by PAG. Let us note that both models achieved the same classification accuracy, however PAG succeeded to reduce each class to a single concept for each class.

To explore single prototypes and compare them to other objects we display a feature vector as histogram where each feature value is represented by a bin. A comparison can now be done by displaying the bins for each object next to each other as can be seen in figure 6. For the iris example, we now recognize that class one is characterized by rather small values for the first three attributes while the other two classes provide rather high values for these attributes. Thus, we have drawn general conclusions from a kNN based classifier. The data objects to be displayed can be chosen by simply clicking the image points in the MDS view of the prototype model. The smaller objects around the prototypes are test instances that can be displayed to get an impression of how prototypes represent the training set.

Set	PAG	LVQaG	ICF	k-means	centroid	kNN	NB	J48	SVM
bal.-scale	**91.2**	80.2	81.5	83.5	73.8	90.4	90.6	76.3	**91.2**
breast-w	**97.1**	96.7	95.1	96.6	96.1	**97.1**	96.0	94.3	96.0
colic	79.3	74.4	78.8	75.3	70.1	80.4	73.1	84.0	**84.2**
credit-g	70.9	74.9	75.4	70.1	67.3	75.3	74.4	71.8	**77.1**
diabetis	74.5	76.0	74.9	71.5	72.9	75.3	76.0	73.8	**76.8**
hepatitis	**86.5**	81.2	82.3	85.8	79.4	83.2	83.9	76.8	81.9
ionosphere	87.7	88.6	88.9	88.0	74.1	87.5	81.8	89.2	**89.7**
iris	96.7	95.3	92,6	**97.3**	92.7	96.7	94.7	94.7	95.3
labor	**94.7**	89.4	82.5	82.4	73.7	84.2	86.0	86.0	91.2
sonar	80.3	77.4	81.3	**87.0**	69.7	86.1	68.8	74.5	83.7
soybean	91.7	90.2	90.8	91.9	79.2	92.5	92.5	93.4	**94.6**
lyase	80.0	75.9	65.5	77.7	71.7	83.2	59.5	47.6	**80.1**
cell growth	71.2	70.7	65.0	70.3	36.4	**73.0**	35.4	43.7	69.1
Signal Transd.	**72.0**	70.4	61.3	70.7	49.6	68.2	44.7	49.1	63.9

Table 1: Comparison of classification accuracies in percent.

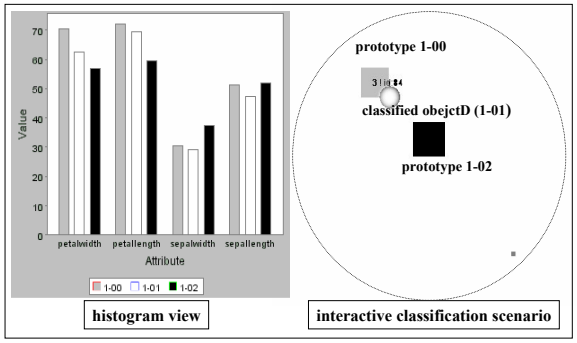

Figure 7: Analyzing a classification decision by analyzing the neigborhood of the classified instance.

To analyze particular classification decisions, we display the data objects within the prototype model with MDS. However, since MDS is not exact, we cannot rely on the visualization of the distance relationships between the prototypes and classified objects, especially if the prototype model contains various prototypes for each class. Thus, we can select to display an MDS visualization containing only the data object and the closest prototypes from each of the classes. Since only these prototypes are relevant for the classification decision, we do not need any other objects to understand the decision. For this visualization, we emphasize the weights of the distances between the data objects and the prototypes by a weight of 1 and shift the visualization error to the distances between the prototypes by weighting them with a small value like 0.0001. The zoomed MDS visualization only displays the number of classes $|C|$ plus one data object and thus the stress function of MDS is now mainly dependent on $|C|+1$ distances. Thus, the visualization of the classification scenario usually gives a quite good impression of the distance relationships leading to a class decision. To get a verification that the displayed image gives a correct impression, we additionally provide the possibility to calculate the exact distances in the original feature space as well. In the iris example displayed in figure 7, we can now recognize that the data object is closest to the prototype of class 1. A comparison using the histogram view reveals that the proximity to the closest prototype is mainly based on the first three attribute values because the next nearest prototype has similar values for the fourth feature.

6 Experimental Evaluation

6.1 Test bed Our experiments were performed on 14 different data sets. To have a broad and comparable set of well known classification tasks, we employed 11 data sets from the UCI [12] machine learning repository. However, since many of these problems are described by a comparably small number of training objects, these problems are not well suited to demonstrate the capability of PAG to describe even complex, multimodal classes. Thus, we additionally used 3 amino-acid-sequence data sets that were taken from the Swissprot [1] protein database and were labelled with respect to Gene Ontology [4]. For these datasets, we employed the feature transformation proposed in [6]. The data sets and their characteristics are displayed in table 2.

6.2 Performance of Push and Grow In this section, we compare the classification performance of PAG to 8 other classification methods. The first 5 approaches are kNN-based methods like PAG. The first is a basic kNN classifier. To compare PAG with a state-of-the art instance reduction technique, we use ICF as proposed in [3]. As stated before, prototype models can be derived by other methods than PAG as well. Thus, we

Name	classes	# Objects
balance-scale	3	625
breast-w	2	699
colic	2	368
credit-g	2	1000
diabetis	2	768
hepatitis	2	155
ionosphere	2	351
iris	3	150
labor	2	57
sonar	2	208
soybean	19	683
lyase sequ.	35	1640
cell growth	36	4401
signal transd.	39	2208

Table 2: Summary of the used datasets

Set	PAG	LVQaG	ICF	k-means
bal.-scale	187.5 (1)	175.8 (1)	6.1 (34)	23.4 (9)
breast-w	321.4 (1)	314.6 (1)	21.1 (17)	11.5 (30)
colic	166.7 (1)	166.7 (1)	5.9 (31)	3.0 (62)
credit-g	450.0 (1)	450.0 (1)	7.9 (63)	20.5 (24)
diabetis	346.2 (1)	345.6 (1)	5.9 (65)	11.5 (33)
hepatitis	46.5 (2)	69.8 (1)	5.5 (14)	20.0 (4)
ionosphere	160.7 (1)	143.6 (1)	3.0 (58)	3.3 (53)
iris	45.0 (1)	45.0 (1)	2.1 (23)	22.5 (2)
labor	25.7 (1)	22,3 (1)	1.7 (17)	2.2 (13)
sonar	93.8 (1)	72 (1)	2.6 (41)	2.7 (39)
soybean	26.7 (1)	5.8 (6)	2 .6 (14)	4.4(8)
lyase	12.5 (4)	2.1(45)	1.4 (33)	10.1 (5)
cell growth	11.7 (10)	7.4(11)	2.5 (49)	11.1 (11)
Signal Tra.	4.3 (13)	2.1(24)	1.6 (35)	2.9 (19)

Table 3: Comparison of data compression rates.

compare to a centroid based classifier and derive prototype models using k-Means clustering. Let us note that we used a wide variety of values for k to find an optimal setting for all comparison partners. Last but not least, we tested a variant of PAG using learning vector quantization [10] instead of the proposed push step. We called this variant "LVQ and Grow" (LVQaG). The remaining 3 classifiers were taken from the weka machine learning software [16] to demonstrate that PAG offers comparable accuracies to other established classification paradigms. Therefore, we compare to naive Bayes, a J48 decision tree and a support vector machine using a polynomial or a linear kernel function. Table 1 displays the achieved classification accuracy in percent for 10-fold cross validation.

The results demonstrate that PAG offered highly accurate classification on all 14 data sets. For 5 datasets, PAG offered the best performance compared to all 8 other classifiers. Let us note that only support vector machines dominated more problems, i.e. 7 data sets. On 10 out of 14 data sets PAG outperformed ICF and on 9 out of 14 data sets PAG offered better accuracy than using prototype models that were derived by k-Means. Using simple centroids as prototypes always provided a lesser accuracy for any of the data sets. Another very important result is that PAG performed better than LVQaG on 11 out of 14 data sets. Thus, the push step demonstrated to be better suited for optimizing the position of data objects. To conclude, PAG provided the best classification performance for all NN-based classification approaches.

This observation is especially interesting when comparing the compression ratio of all four instance reduction techniques. Table 3 contains the compression ratio of PAG, LVQaG, k-Means and ICF, i.e. the factor by which the method decreases the number of instances in the training base. Additionally, the average number of instances/prototypes per class is displayed in brackets. The parameter settings for all methods were chosen to provide maximum accuracy. For all 15 data sets, PAG represented the given training data while achieving a higher compression level than any of the other methods. For the first 9 data sets, it even achieved a maximum compression by representing each class by a single prototype. The only data sets where PAG was forced to use a significantly larger number of prototypes, are the three protein sequence problems. However, these data sets contain complex, multi-modal classes. Since PAG increased the number of used prototypes for these data sets, the found prototype models demonstrate the capability of PAG to react to multi-modal data and thus to increase the size of the prototype model, if necessary.

6.3 Further Examples for Visual Class Knowledge In this section, we present some additional examples for the visualization of class knowledge. All three examples where taken from the 14 training data sets to ensure that APMs offer high classification accuracy. This is important because analyzing a classifier that is performing badly, might lead to misleading assumptions.

The first example, displayed in figure 9, is the labor data set distinguishing good jobs from bad jobs. We chose this example to demonstrate the necessity of generalization and instance reduction in order to understand the classes. The left side displays the complete data set using MDS. Though we see that feature vectors for bad jobs are mainly displayed on the right side of the visualization, it is difficult to recognize a clear pattern for the classes. The reason for this problem is that visualizing a large number of vectors often displays high stress values. Thus, even if there are well-defined class borders, the probability that these borders are clearly displayed in MDS decreases with the

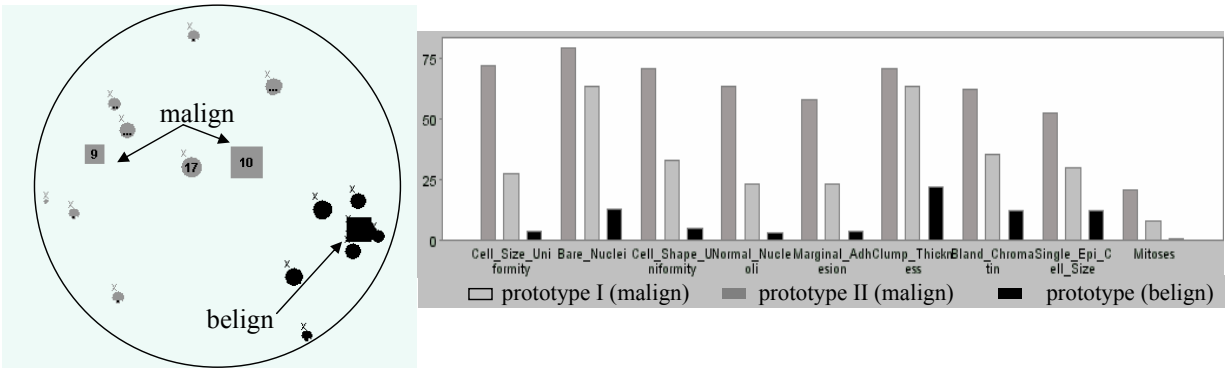

Figure 8: The left circle displays the prototype model and some example objects for the breast-w data set. The histogram displays the feature values of the prototypes.

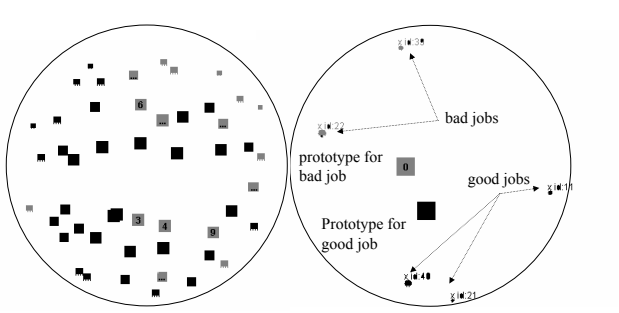

Figure 9: The left image is a MDS display of the labor data set. The right image displays the corresponding prototype model and some example objects.

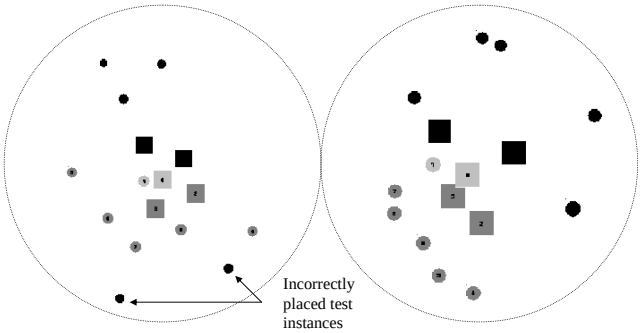

Figure 10: Unweighted (left side) and weighted (right side) MDS visualization of a prototype model for the balance scale data set. The test instances are only displayed on the correct side for the weighted MDS.

number of data objects. An additional problem that is independent of visualization, are noise objects blurring the class borders. The right side of figure 9 displays the prototype model. In this view, it becomes quite clear that a linear separation works out quite well for this data set.

Figure 10 is an example for the impact of the introduced weighted stress function. The class model on the left set displays a classification model of the balance-scale dataset comprising 5 prototypes. Additional to the prototypes, we added 11 test instances that are correctly classified. The left side displays an unweighted MDS visualization. Two of the instances are displayed in an area that is nearer to some prototype of the incorrect class. Let us note that even trying several start configuration did not help to correct this misleading effect. The right side, on the other hand, displays a correct image with respect to the class assignment by assigning small weights (w=0.0001) to the distances between test instances and prototypes of the same class. Of course, the right side does not display the exact distance relationships as well, but it can shift error to unimportant information. Thus, the displayed image of the class model is easier to understand.

Our last example, is the "breast-w" data set distinguishing belign from malign tumors of breast cancer patients. Figure 8 displays the prototype models containing two prototypes for class "malign" and one for class "belign". Since we achieved a classification accuracy of 97.1 %, we can assume that the class model is a quite good description of the data. To analyze all three prototypes, we employ the histogram view. From this view of prototype models, we can derive that a patient whose tumor displays small values in most of the features has a very good chance that the tumor is belign. The prototype for the class "belign" displays rather small values

for all features compared to those of the malign class.

7 Conclusions

In this paper, we proposed APMs a classification system that is based on nearest prototype classification and a visualization suite that is capable to display high-dimensional data sets for classification. The idea of APMs is to describe each class by a minimal set of prototypes that are generalized from the objects of the training set. Working with small prototype models increases the speed of classification, achieves accurate classification and allows us to browse and understand the class models with data visualization techniques. Therefore, we introduce the algorithm "Push and Grow" (PAG) which is based on a quality measure that favors large margins between classes and thus prefers general class models. The algorithm employs two different steps increasing the model quality. The second part of an APM is a visualization suite to interactively explore the prototypes and data objects. This suite employs weighted interactive multi-dimensional scaling (MDS) and projects high dimensional feature vectors into the hyperbolic space. MDS is needed to project high-dimensional data objects and prototypes into a 2D image. By assigning weights to distances in MDS, we achieve that the visualization emphasizes the more important distance relationships. Our visualization suite allows us to analyze the data space, class models and individual classification scenarios. In our experimental evaluation, we demonstrate the good classification accuracy of PAG compared to 8 other classification methods on 14 data sets. Furthermore, we show that PAG provides smaller class models than the compared methods of instance reduction and prototype classification. Finally, we display some examples of how APMs can be used to derive knowledge about classes and data distributions.

For future work, we plan to extend our visualization suite to directly display the data objects that are represented by the feature vectors. This is an interesting task for prototypes because there is no single underlying object, but a complete set of objects. Another interesting idea is visualizing the closest points on the class border for a given data object. This way, it is possible to derive the smallest modification of a feature vector that would cause another classification result. To extend PAG for other data spaces, we plan to apply PAG to text vectors and graph-represented data objects.

References

[1] B. Boeckmann, A. Bairoch, R. Apweiler, M.-C. Blatter, A. Estreicher, E. Gasteiger, M. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider. "The SWISS-PROT Protein Knowledgebase and its Supplement TrEMBL in 2003". *Nucleic Acid Research*, 31:365–370, 2003.

[2] H. Brighton and C. Mellish. On the consistency of information filters for lazy learning algorithms. In *PKDD*, pages 283–288, 1999.

[3] H. Brighton and C. Mellish. Advances in instance selection for instance-based learning algorithms. *Data Mining and Knowledge Discovery*, 6:153–172, 2002.

[4] T. G. O. Consortium. "Gene Ontology: Tool for the Unification of Biology". *Nature Genetics*, 25:25–29, 2000.

[5] C. Cortes and V. Vapnik. "Support-Vector Networks". *Machine Learning*, 20(3):273–297, 1995.

[6] M. Deshpande and G. Karypis. Evaluation of techniques for classifying biological sequences. In *PAKDD '02: Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 417–431, London, UK, 2002.

[7] E.-H. Han and G. Karypis. "Centroid-Based Document Classification: Analysis and Experimental Results". In *Proc. 4th European Conf. on Principles of Data Mining and Knowledge Discovery (PKDD'00), Lyon, France*, Lecture Notes in Computer Science (LNCS), Springer, pages 1910: 424–431, 2000.

[8] J. Han and M. Kamber. *"Data Mining Concepts and Techniques"*. Morgan Kaufmann Publishers, 2001.

[9] T. Kohonen. "Learning Vector Quantization". Technical Report, Helsinki Univ. of Tech., 1986.

[10] T. Kohonen. "Improved versions of learning vector quantization". In *IJCNN International Joint Conference in Neural Networks,Washington DC., USA*, pages 545–550, 1990.

[11] R. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for viewing large hierarchies. In *in Proc. ACM SIGCHI Conference on Human Factors in Computer Systems,Denver, CO, USA*, pages 401–408, 1995.

[12] U. of Irvine. Uci machine learning repository. "http://www.ics.uci.edu/ mlearn/MLRepository.html", 2005.

[13] J. Samon Jr. A none-linear mapping for data structure analysis. *IEEE Transactions on Computers.*, 18:401–409, 1969.

[14] J. Walter and H. Ritter. On interactive visualization of high-dimensional data using the hyperbolic plane. In *Proc. 8th int. Conference on KDD SIGKDD'02, Edmonton, Alberte,CA*, pages 123–132, 2002.

[15] H. Wilson and T. Martinez. Instance pruning techniques. In *Proc. 14th Int. Conf. on Machine Learning*, pages 403–411. Morgan Kaufmann Publishers, 1997.

[16] I. Witten and E. Frank. *"Data Mining: Practical Machine Learning Tools and Techniques with Jave Implementations"*. Morgan Kaufmann, 1999.