Considering Decision Cost During Learning of Feature Weights

Wolfgang Wilke and Ralph Bergmann

University of Kaiserslautern
Centre for Learning Systems and Applications (LSA)
Department of Computer Science
P.O. Box 3049, D-67653 Kaiserslautern, Germany
e-Mail: {wilke,bergmann}@informatik.uni-kl.de

Abstract. This paper is to present a new algorithm, called KNN_{cost}, for learning feature weights for CBR systems used for classification. Unlike algorithms known so far, KNN_{cost} considers the profits of a correct and the cost of a wrong decision. The need for this algorithm is motivated from two real-world applications, where cost and profits of decisions play a major role.
We introduce a representation of accuracy, cost and profits of decisions and define the decision cost of a classification system. To compare accuracy optimization with cost optimization, we tested KNN_{acc} against KNN_{cost}. The first one optimizes classification accuracy with a conjugate gradient algorithm. The second one optimizes the decision cost of the CBR system, respecting cost and profits of the classifications. We present experiments with these two algorithms in a real application to demonstrate the usefulness of our approach.

1 Introduction

Developing a case-based reasoning system for a real-world application requires a lot of effort. For example, features must be selected to represent cases, the types of the features must be defined, and the similarity measure for all instances of the attributes must be selected by an expert. One step of the last task of this development process deals with the determination of feature weights. These weights can be acquired from an expert or can be determined by a learning algorithm that tries to extract the importance of the features for a given set of cases. Learning algorithms are required, if no natural weighting can be given for the domain or the weights given by an expert need an improvement. There are several different learning algorithms for feature weights. We focus in this work on algorithms using feedback for learning.
For example, Salzberg's EACH (Salzberg, 1991) proceed as follows: if a correct classification occurs, the weights of the matching features are incremented, while those of mismatching features are decremented of the new query by a fixed amount. If an incorrect classification occurs, matching feature weights are decremented and mismatching feature weights are incremented. Other approaches like VDM (Stanfill and Waltz, 1986), IB4 (Aha, 1989; Aha, 1991), RELIEF-F

(Kononenko, 1994), PADEX/INRECA (Wess, 1993; Wess, 1995), VSM (Lowe, 1995) or $k - NN_{vsm}$ (Wettschereck, 1994; Wettschereck and Aha, 1995) mostly differ in the way feature weights are modified, but share the main criterion for changing weights, namely: the correctness of the classification. As a consequence, these algorithms optimize classification accuracy only. We argue that this is not an appropriate learning goal for many real-world applications, because *decision cost* play a major role in several domains.
Our view of the problem is motivated by experiences with the following two applications:

Credit Scoring:
The goal of this task is to decide about a business bank customer's creditworthyness. Available cases consist of the main balance items from the balance sheet, financial index numbers, and other relevant enterprise data, together with the credit-rating. The CBR system classifies new business customers to decide whether the enterprise is creditworthy (class A), or not (class B). In this application, the profits and cost of a right or wrong decision are asymmetric. If the system classifies a creditworthy customer (class A) as not creditworthy (class B) and the bank rejects the credit for the customer, the bank looses only the interest income. On the other hand, if a class B customer is classified as a member of class A, the bank looses the whole credit sum, which is a considerably higher loss. Here, the decision cost of a wrong decision depend strongly on the predicted and the correct class of the customer.

Diagnosing Cases of Poisoning by Psychotropes:
In the INRECA+ project (Althoff et al., 1996), we built a CBR system for diagnosing cases of poisoning by psychotropes. The cases[1] consist of 86 attributes that have been identified as useful for this diagnosis task by experts. The decision classes describe 8 different kinds of possible therapies for treating the poisoning. Of course, the different therapies cause different effects on the patient's constitution. In the worst case a wrong therapy can be mortal for the patient. Thus, it is important to come up with the right diagnosis as quick as possible. However, for a number of diagnoses the respective therapy is identical or, at least, very similar. Therefore making a wrong diagnosis leading to the same (or nearly the same) therapy as the correct diagnosis is no problem. So, in this application decision cost differ significantly.

These two examples show that decision cost may play a major role for the optimization of a CBR system. For that purpose, we developed a feature weight learning algorithm KNN_{cost}, which optimizes the feature weights, with regard to cost and profits of the decisions of a case-based classification system. To compare KNN_{cost} to an algorithm that optimizes classification accuracy only, we first introduce KNN_{acc}, a learning algorithm that uses the same mechanism

[1] The application data was provided by the Toxicology Information and Advisory Centre of the Russian Federation Ministry of Health and Medical Industry

as KNN_{cost}, with respect to the accuracy only, yet with another optimization criterion.
The paper is organized as follows: In the next section, we give a short introduction into $k-$nearest neighbor classification and illustrate our definition of accuracy and cost for a classification system. Section 3 introduces two algorithms KNN_{acc} and KNN_{cost}. After that, we describe an empirical evaluation of the algorithms in the credit scoring application in section 4. We then add a discussion and, finally, give an outlook on future work.

2 Basics

2.1 Case-Based Classification with $k-$Nearest Neighbors

In a CBR system used for classification tasks, a case $c = (f_1, \ldots, f_n, t_c)$ consists of n describing features f and of the desired class t_c. The set $T = \{t_1, \ldots, t_l\}$ denotes all possible classes in the domain. The case base CB is defined as a set of known cases from the past. Given a new query $q = \{q_1, \ldots, q_n\}$ and a case base CB, the k most similar cases are retrieved to predict the real class t_q of a query q. The similarity $sim(q, c)$ between a query q and a case c from the case base is defined as:

$$sim(q, c) = \sum_{a=1}^{n} w_a * sim_a(q_a, f_a) \tag{1}$$

where w_a is the weight for feature f_a and sim_a is the local similarity measure for attribute a. Further, we assume that $\sum_{a=1}^{n} w_a = 1$ and $sim_a(q_a, f_a) \in [0, 1]$, for all features, yielding the similarity between a query and a case $sim(q, c) \in [0, 1]$. The CBR system predicts the class of the query by retrieving the $q's$ $k-$nearest neighbors $K = \{r_1, \ldots, r_k\}$ and applying a majority vote method on them, e.g.: Let $p_{q,t}$ denote the probability[2] that a query q is a member of the class $t \in T$ defined as:

$$p_{q,t} = \frac{\sum_{r \in K} \delta_{r,t} * sim(q, r)^2}{\sum_{r \in K} sim(q, r)^2} \tag{2}$$

where $\delta_{r,t}$ is defined as follows:

$$\delta_{r,t} = \begin{cases} 1 & : \quad t_r = t \\ 0 & : \quad t_r \neq t \end{cases} \tag{3}$$

and where $t_r \in T$ denotes the class of case r. Then, the prediction of the CBR system is the class with the highest probability calculated from the set of the $k-$nearest neighbors.
There are other voting algorithms for the $k-$nearest neighbor classification, for example the single majority vote (Michie et al., 1994)[p. 10 pp], (Weiss and

[2] If the numerator in the definition of $p_{q,t}$ is 0, then $p_{q,t}$ is set to 0

Kulikowski, 1991) [p. 70] that only considers the frequencies of the different classes in the set of nearest neighbors. However, the weighted majority vote has the main advantage that the distance of the neighbors is taken into account for the prediction.

2.2 How to Present Accuracy and Decision Cost

A common representation for the classification accuracy of a system is a confusion matrix (Weiss and Kulikowski, 1991)[p. 18]. For a given set of cases, the entry of such a matrix counts the number of classifications of cases from a class as a member of the predicted class by the system, for a given set of cases. For

$P_{i,j}$	**predicted class**	
correct class	**A**	**B**
A	0.55	0.45
B	0.25	0.75

Table 1. A sample confusion matrix for the credit scoring application

our approach, we modify the entries of the confusion matrix with respect to the probability $p_{q,t}$ from equation (2) to classify that a query q is a member of the class t. So our matrix contains the probabilities for every possible decision of the system for a given set of cases.
For measuring the accuracy of a CBR system during learning, one can use a leave-one-out test (Weiss and Kulikowski, 1991)[p. 31] over a given training set L. Every case from L is used as a query q and classified with the CBR system including all other cases from L except the query q. For every query q, the probabilities $p_{q,t}$ from the retrieved k-nearest neighbors using equation (2) can be calculated for every $t \in T$. This is done for every case from L and the resulting probabilities are combined for every entry of the confusion matrix. After that we normalize every row of the confusion matrix with respect to the occurrences of the correct classes in the given training set. The resulting confusion matrix represents the probabilities for the different outcomes of a classification with the training set L. This matrix is an approximated confusion matrix, because we use probabilities for the decisions, instead of counting the occurrence of the different decisions[3]. Table 1 shows an example of such a matrix for the credit scoring application with the two different classes and predictions. The probability of the correct predictions for each class can be found in the diagonal of the matrix, here 55 and 75 percent. All other entries represent the probabilities of errors for a particular type of misclassification, here 45 and 25 percent.
To represent cost, (Weiss and Kulikowski, 1991)[p. 21] and (Michie et al., 1994)[p.

[3] For simplification, we further use the term confusion matrix

224] use a similar matrix called *cost matrix*, in which non-diagonal entries represent cost of a specific misclassification and the fields in the diagonal represent the benefits of a correct classification, explicitly set to 0. We extend this approach by allowing positive and negative entries for representing cost *and* profits in a single matrix. As a result these *decision value matrix* represents the cost and profits for every possible decision of the system. Table 2 shows an example for such a decision value matrix in the credit scoring application. The values

$C_{t_q,t}$	**predicted class**	
correct class	**A**	**B**
A	-1	1
B	10	-10

Table 2. A sample decision value matrix for the credit scoring application

represent the relation between the two possible errors that might occur. Here the cost of misclassifying a bad customer are 10-times higher than misclassifying a good one[4].
In the first case, the bank looses the credit volume, but in the second case only the interest rate is lost. Consequently the profit of detecting a bad one is here 10-times higher than correctly classifying a good one[5].
So, this decision value matrix represents the cost and profits of the different decisions of the classification in the bank application.
With the representation of accuracy from the confusion matrix and the cost from the decision value matrix we can define the *decision cost* of a classification system as:

$$Cost_{decision} = \sum_{i \in T} \sum_{j \in T} P_{i,j} * C_{i,j} \tag{4}$$

where the $P_{i,j}$ are the probability from the confusion matrix and the $C_{i,j}$ are the corresponding entries from the decision value matrix.

3 Incremental Learning of Feature Weights

In this section, we first describe the general algorithm for learning feature weights with a conjugate gradient algorithm. In the next two subsections we specialize this algorithm to the algorithms KNN_{acc} and KNN_{cost}.

[4] See the relation between the entries $C_{B,A}$ and $C_{A,B}$ in Table 2.
[5] See the entries $C_{B,B}$ and $C_{A,A}$ in Table 2.

3.1 Learning Weights Guided by the Conjugate Gradient

The conjugate gradient method is a generate-and-test search algorithm with feedback from the test procedure. The algorithm tries to optimize a system with respect to an error function E by adjusting the weights $w = \{w_1, \ldots, w_n\}$. This optimization is realized by an iterative search for a local minimum of the error function E. So E has to be chosen according to the learning goal. A learning rate λ is used to influence the step width for a learning step in the direction of the conjugated gradient. The basic algorithm looks at follows:

1. initialize the weight-vector w and the learning rate λ
2. compute the error $E(w)$ of the initial system
3. **while** $not(stop - criterion)$ **do**
   learning step: $\forall\ a\ \hat{w_a} := w_a - \frac{\partial E}{\partial w_a} * \lambda$
   compute $E(\hat{w})$
   if $E(\hat{w}) < E(w)$ **then** $w := \hat{w}$ **else** $\lambda := \frac{\lambda}{2}$
4. output w

First, the weights w_i are either initialized to random values, set to the constant $\frac{1}{n}$, or given by an expert. After the calculation of the initial E, the algorithm does a number of learning steps, depending on the stop-criterion. If a learning step is successful ($E(\hat{w}) < E(w)$), the weights are modified to guide E in the direction of a local minimum. Otherwise, the learning rate is decreased. The algorithm terminates and outputs a weight vector as the result of the learning.

The choice of the learning rate The choice of a good value for λ is difficult, because it depends on many unknown domain properties. If λ is quite small, many learning steps are needed to find the next local minimum in the neighborhood of the starting point. However, it is guaranteed that the algorithm finds it,

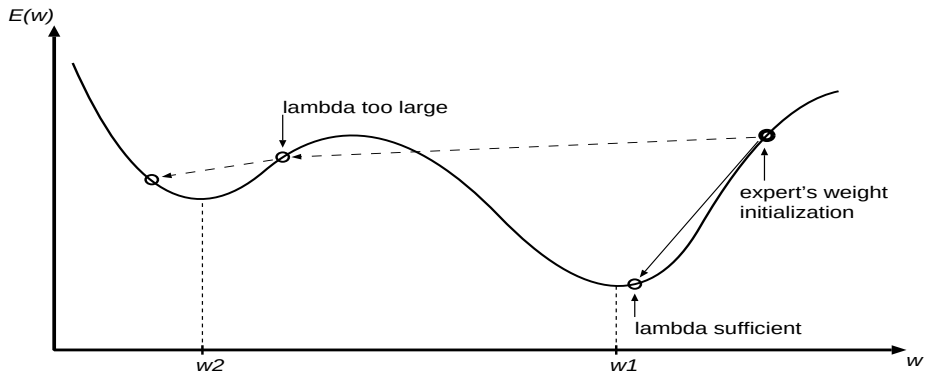

Fig. 1. The effects of different values for the learning rate

respecting the initial starting point. If λ is too large, misleading steps may happen and the algorithm very often decrements the value of λ, before it improves E. It is also possible that a learning step with a large λ leads to an improvement of E, but this improvement is based on a different minimum. So, the algorithm finds a local minimum of E, not respecting to the initial weights. This behavior is illustrated in figure 1. The doted line shows learning steps which lead to the minimum $w2$, because the value of λ is too large. The solid line shows a learning step with a sufficiently small learning rate. This leads step wards a local minimum $w1$ that respects the expert initialization. Thus, if the learning rate is too large, the behavior of the algorithm is not respecting the initialization, because the founded minimum $w2$ is not related to weights given by an expert.

The choice of the stop-criterion The stop-criterion has to ensure the termination of the algorithm. Possible stop criteria for the algorithm are:

- a fixed number of learning steps
- a minimal change of the error function: $\mid E(\hat{w}) - E(w) \mid \leq \epsilon$
- a minimal change of the feature weights: $\sum_{a=1}^{n} \mid \frac{\partial E}{\partial w_a} \mid \leq \epsilon$
- a minimal learning rate: $\lambda \leq \epsilon$

It is also possible to combine these criteria.

The choice of k A different value of k for the amount of the nearest neighbors taken into account leads to different probabilities for the classifications. The optimal value of k could be computed by a simple generate and test procedure. Here, optimal means a minimal value for the decision cost from equation (4). We used a fixed k from our experience with the domain for the whole learning procedure, like VSM (Lowe, 1995). (Wettschereck and Aha, 1995) calculate an optimal k prior to the first learning step and fix it for the rest of the learning procedure. A computational expensive approach calculates a new optimal k after each learning step. This is necessary, because in general the new weights after a learning step could effect the optimality of the old value of k. The new calculation of an optimal k after every step is costly, but should lead to the best results.

3.2 KNN_{acc} for optimizing classification accuracy

Now we specialize the basic algorithm of the conjugated gradient to KNN_{acc}, which optimizes classification accuracy. This is done by defining a special error functions E for this purpose, together with the required derivation $\frac{\partial E_{acc}}{\partial w_a}$. During the learning we use the leave-one-out test, described in section 2.2 to calculate the value for $E(w)$.
KNN_{acc} is similar to the VSM (Lowe, 1995) and $k-NN_{vsm}$ (Wettschereck and Aha, 1995) algorithms. They also determinate feature weights with a conjugate gradient algorithm. Unlike other approaches, we use a similarity measure to quantify the distance of cases. This measure is not fixed for different types of

features.
The error function E_{acc} for optimizing the classification accuracy E_{acc} can be defined as:

$$E_{acc} = \sum_{q \in CB} \sum_{t \in T} (\delta_{q,t} - p_{q,t})^2 \tag{5}$$

For a learning step with the conjugate gradient method, we need the derivation of E_{acc} with respect to the weights w_a from equation (5):

$$\frac{\partial E_{acc}}{\partial w_a} = -2 * \sum_{q \in CB} \sum_{t \in T} (\delta_{q,t} - p_{q,t}) * \frac{\partial p_{q,t}}{\partial w_a} \tag{6}$$

where the derivation of $p_{q,t}$ for the weights w_a is given by[6]:

$$\frac{\partial p_{q,t}}{\partial w_a} = \frac{2 * \sum_{r \in K} [(\delta_{r,t} - p_{q,t}) * sim(q,r) * sim_a(q_a, r_a)]}{\sum_{r \in K} sim(q,r)^2} \tag{7}$$

The KNN_{acc} algorithm is the conjugate gradient algorithm together with the error function from (5) and the derivation given in (6). Minimizing the error function means maximizing the overall probabilities for the prediction of the correct classes and minimizing the overall probabilities of a misclassification for the training set. In the confusion matrix (see Table 1) KNN_{acc} tries to maximize the diagonal and to minimizes the non-diagonal entries.

3.3 KNN_{cost} for Optimizing Decision Cost

The idea of KNN_{cost} is to minimize the decision cost as defined in equation (4). This is done by integrating the decision value matrix (Table 2) into the error function. So every probability in the error function is judged by its respective cost from the decision value matrix. The resulting error function E_{cost}, which defines the error for the respective decision cost, is given by:

$$E_{cost} = \sum_{q \in CB} \sum_{t \in T} sgn(C_{t_{q,t}}) * C_{t_{q,t}}^2 * p_{q,t}^2 \tag{8}$$

where $C_{t_{q,t}}$ is an entry from the decision value matrix and $sgn(C_{t_{q,t}})$ is the sign of this entry. The derivation $\frac{\partial E_{cost}}{\partial w_a}$ used to calculate the new weights can be computed as follows:

$$\frac{\partial E_{cost}}{\partial w_a} = 2 * \sum_{q \in CB} \sum_{t \in T} sgn(C_{t_{q,t}}) * C_{t_{q,t}}^2 * p_{q,t} * \frac{\partial p_{q,t}}{\partial w_a} \tag{9}$$

where $\frac{\partial p_{q,t}}{\partial w_a}$ is the formula from equation (7). Thus, our algorithm KNN_{cost} is the conjugate gradient algorithm from the first part of this section together with

[6] Please note that these equations are similar to those shown in (Wettschereck, 1994), except for the replacement of distances by similarities

the error function E_{cost} and its derivation $\frac{\partial E_{cost}}{\partial w_a}$. With this error function the algorithm minimizes the decision cost of the CBR system. In our representation of accuracy and cost this means that KNN_{cost} tries to minimize/maximize the products of every entry of the confusion matrix with a corresponding entry of the decision value matrix, depending on cost and profits[7].

4 Empirical Evaluation of KNN_{acc} and KNN_{cost}

We now present the results of an empirical evaluation of KNN_{acc} and KNN_{cost}. The algorithms were implemented as part of the INRECA[8] CBR - shell. The goal of these tests is to verify or reject our hypothesis:
Classification based on weights learned by KNN_{cost} leads to lower decision cost than classification based on weights learned by KNN_{acc}.

4.1 Experimental Settings

For our experiments we used the the credit scoring domain with 685 cases. A case consists of 136 different features and a class description. There are 20 numeric attributes while the remaining have symbolic values. About 5 percent of the attributes were unknown in each case description.
To test our learning algorithms, we made 5 independent runs. In every run, we divided the case base in a training set with 70 percent randomly selected cases and use the remaining 30 percent as a test set. In each run, we used the training set as case base for the CBR system[9]. The weights were initialized randomly. In each run we learned feature weights using the KNN_{acc} and KNN_{cost} algorithms. The learning process was stopped when a change less than $5 * 10^{-3}$ occured. The initial learning rate λ was set to $5 * 10^{-4}$. In our tests we took $k = 11$ nearest neighbors into account.

4.2 Results

Now we compare the decision cost of the initial CBR systems to those of the resulting systems after learning feature weights with both algorithms. First, we exemplarily show how to calculate the decision cost. We classify the 206 cases from the test set with the initial CBR systems. The left side of the Table 3 shows a confusion matrix with the classification accuracy averaged over the 5 initial systems. Contrary to the matrix introduced in Table 1, the entries denote the average occurrence of classifications for the cases of the test set. We choose

[7] This is equal to minimizing the decision cost, because they are computed as the sum of the single products. The squares in equation (8) were only introduced to speed up the learning process.

[8] INRECA (ESPRIT contract P6322, the INRECA project) with the partners: AcknoSoft (prime contractor, France), tecInno (Germany), Irish Medical Systems (Ireland) and the University of Kaiserslautern (Germany)

[9] In the following we name these systems initial CBR systems

	predicted class			
	accuracy		*cost*	
correct class	**A**	**B**	**A**	**B**
A	48.6	36.4	-48.6	36.4
B	27.4	93.6	274	-936.0
	decision cost: **-674.2**			

Table 3. The average decision cost of the CBR systems before learning

this different representation for the classifications to make the entailed cost of the systems more explicit. These entries were multiplied with the respective cost from the cost matrix (Table 2). The resulting cost for every possible decision are shown on the right side of the Table 3. The bottom line of the table shows the average decision cost for the 5 initial systems as a result of the sum of all entries of the decision value matrix[10] .

Comparing KNN_{acc} and KNN_{cost} In the following we compare our initial systems with the systems after learning feature weights using KNN_{acc} and KNN_{cost}. After the learning phase, we test the resulting systems with the test set. For both learning algorithms, we calculate the average decision cost of the resulting systems. Table 4 summarizes the entire decision cost of the initial systems and the resulting systems after learning. Both learning algorithms lead to

	Decision cost
Before Learning:	**-674.2**
Learning with KNN_{acc}:	**-756.2**
Learning with KNN_{cost}:	**-1040.6**

Table 4. The decision cost of the CBR systems before and after learning feature weights

an improvement of the decision cost. The average decision cost of the systems with the weights obtained from KNN_{acc} are 82 (= 756.2 − 674.2) lower than the decision cost before learning. We associated profits with right and cost with wrong decisions. So, this improvement represents the benefit of the better classification accuracy after learning. The weights extracted with KNN_{cost} lead to an improvement of 366.4 (= 1040.6 − 674.2) for the decision cost. Especially, the decrease of the decision cost of 284.4 (= 1040.6 − 756.2) obtained by using

[10] Here, a negative value denotes the profit of the classification systems

KNN_{cost} when compared to the results obtained by KNN_{acc} is remarkable. The classification accuracy of the systems that use weights learned by KNN_{acc} and of those using weights learned by KNN_{cost} were nearly the same. The reason for the improvement of the decision cost is that KNN_{cost} prefers to classify more costly cases correctly than KNN_{acc}. So, our hypothesis that classification based on weights learned by KNN_{cost} leads to lower decision cost than classification based on weights learned by KNN_{acc}, has been empirically verified in this domain.

5 Summary and Discussion

In this paper, we presented two feature weight learning algorithms, one optimizing classification accuracy only and one optimizing decision cost. Both algorithms use the conjugate gradient to optimize the feature weights for the different criteria. We empirically evaluated these two algorithms in the domain of credit scoring with real bank-customer data. In this evaluation we could verify our hypothesis that cost optimization could be more profitable than classification accuracy optimization.

It should also be possible to integrate cost into other algorithms for learning feature weights to improve a CBR system. Especially, the extension of algorithms using feedback seems easy in some cases. Here, the feedback denotes the amount of change for every weight after a learning step to improve the system.

As already stated, EACH (Salzberg, 1991) changes the feature weights by a fixed value depending on whether correct or incorrect classification occured and whether a feature matches or mismatches. To introduce cost, this value could vary according to the decision of the system. EACH has two disadvantages: it is very sensitive to the order of the presentation of the examples and all weights were simultaneously changed by a fixed amount in one learning step.

RELIEF (Kira and Rendell, 1992) selects a random training case c, the most similar case p of the same class, and the most similar case n of a different class. The new feature weights are calculated by:

$$\hat{w_a} = w_a - difference(c_f, p_f) + difference(c_f, n_f) \tag{10}$$

An approach two introduce decision cost in the feedback of RELIEF (in equation 10) could be to judge the differences with the desired cost of the training case c classified as the class of a similar case, here the class of p or n. The original version of RELIEF is limited to two-class problems only, yet this restriction has been removed by (Kononenko, 1994) in RELIEF-F. He takes all different classes in the feedback into account. The extension to decision cost would be quite similar. As in EACH, the problem with these algorithms is the sensitivity to the order of the presentation of the examples.

IB4 (Aha, 1989; Aha, 1991) takes the distribution of the different classes in the case base into account for changing weights. The amount of change is judged with a factor $(1 - \Delta)$ that represents the observed frequency among the different classes. This is a promising approach to optimizing the classification accuracy.

To introduce decision cost in this approach could be difficult, because weights are optimized according to two conflicting criteria. If a costly class has a low frequency, the benefits of the two criteria could disappear. The cost criterion argues for a massive change of the feature weights, but the frequency criterion argues for a moderate modification. Otherwise, the factor to change a highly frequent class with low cost is also contrary. So, the feedback of these two criteria would compensate, because the criteria accuracy and cost, as already stated could be contradictory.
Other learning algorithms could also be modified in order to take decision cost into account. For example, an instance-based learning algorithm(IBL) (Aha, 1989; Aha, 1990) (IBL) could be extended. Roughly speaking in IBL the cases are rated with their effects on arbitrary classifications and misclassifications. If cases often cause wrong classifications and seldom ensure a right classification, they are removed from the case base. The goal is to keep only those cases in the case base, which ensure a correct classification. To integrate cost, it is possible to rate the cases not only with the classification rating, but with the respective entry from the cost matrix. This will be a topic of our further research in this area.
The price of our approach is that the decision cost must be acquired additionally. However, experts often know the cost and profits for the different outcomes of a classification.

Acknowledgements

The authors would like to thank Prof. Michael M. Richter, Dr. Klaus-Dieter Althoff, Harald Holz and Ivo Vollrath for the useful discussions and for their implementation work. This work was partially funded by the Commision of the European Communities (ESPRIT contract P6322, the INRECA project) with the partners: AcknoSoft (prime contractor, France), tecInno (Germany), Irish Medical Systems (Ireland) and the University of Kaiserslautern (Germany) and partially funded by the "Stiftung Innovation für Rheinland-Pfalz".

References

Aha, D. W. (1989). Incremental, instance-based learning of independend and graded concepts. In *Proceedings of the 6th international Workshop on Machine Learning*, pages 387–391.

Aha, D. W. (1990). *A Study of Instance-Based Algorithms for supervised Learning*. PhD thesis, University of California at Irvine.

Aha, D. W. (1991). Case-Based Learning Algorithms. In Bareiss, R., editor, *Proceedings CBR Workshop 1991*, pages 147–158. Morgan Kaufmann Publishers.

Althoff, K.-D., Bergmann, R., Wess, S., Manago, M., Auriol, E., Larichev, O. I., Bolotov, A., and Gurov, S. I. (1996). Integration of Induction and case-Based Reasoning for Critical Decision Support Tasks in Medical Domains:

The INRECA Approach. *Centre for Learning Systems and Applications Technical Report*, 96-03E.

Kira, K. and Rendell, L. A. (1992). A practical approach to feature selection. In *Proccedings of the Ninth International Conference on Machine Learning*, Aberdeen, Scotland. Morgan Kaufmann.

Kononenko, I. (1994). Estimating attributes: Analysis and extensions of relief. In *Proceedings of the 1994 European Conference on Machine Learning*, pages 171–182. Springer Verlag.

Lowe, D. (1995). Similarity metric learning for a variable-kernel classifier. *Neural Computation*, 7:72–85.

Michie, D., Spiegelhalter, D., and Taylor, C. C. (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood.

Salzberg, S. (1991). A nearest hyperrectangle learning method. *Machine Learning*, 6:277–309.

Stanfill, C. and Waltz, D. (1986). Toward Memory-Based Reasoning. *Communications of the ACM*, 29(12):1213–1229.

Weiss, S. and Kulikowski, C. (1991). *Computer Systems That Learn – Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. Morgan Kaufmann.

Wess, S. (1993). PATDEX - Inkrementelle und wissensbasierte Verbesserung von Ähnlichkeitsurteilen in der fallbasierten Diagnostik. In *Tagungsband 2. deutsche Expertensystemtagung XPS-93*, Hamburg. Springer Verlag.

Wess, S. (1995). *Fallbasiertes Problemlösen in wissensbasierten Systemen zur Entscheidungsunterstützung und Diagnostik*. PhD thesis, University of Kaiserslautern.

Wettschereck, D. (1994). *A Study of Distance-Bases Machine Learning Algorithms*. PhD thesis, Oregon State University.

Wettschereck, D. and Aha, D. W. (1995). Weighting features. In Veloso, M. and Aamodt, A., editors, *Case-Based Reasoning Research and Development*, pages 347–358. Springer.

This article was processed using the LaTeX macro package with LLNCS style