# An Attribute Weight Setting Method for *k*-NN Based Binary Classification using Quadratic Programming

Lu Zhang, Frans Coenen and Paul Leng[1]

**Abstract.** In this paper, we propose a new attribute weight setting method for *k*-NN based classifiers using quadratic programming, which is particular suitable for binary classification problems. Our method formalises the attribute weight setting problem as a quadratic programming problem and exploits commercial software to calculate attribute weights. Experiments show that our method is quite practical for various problems and can achieve a competitive performance. Another merit of the method is that it can use small training sets.

## 1. INTRODUCTION

The *k*-Nearest Neighbour (*k*-NN) algorithm [7] is a typical lazy learning algorithm that has been intensively studied and applied to classification problems. When classifying an instance, *k*-NN selects *k* most similar instances in the training set of instances, and uses the *k* similar instances to determine the class of the instance under classification via some voting mechanism. Usually, each instance is described as a sequence of attributes, i.e. $\{A_1,A_2,\ldots A_q\}$ (where $q$ is the number of attributes), and the similarity between instance $X=\{X_1,X_2,..X_q\}$ and instance $Y=\{Y_1,Y_2,\ldots Y_q\}$ can be calculated through formula (1). In formula (1), $Simi(X,Y)$ is the similarity between instance $X$ and instance $Y$, and $Simi(X_i,Y_i)$ is the similarity on the $i$th attribute between $X$ and $Y$.

$$Simi(X,Y)=\sum_{i=1}^{q} Simi(X_i,Y_i) \qquad (1)$$

In formula (1), all the attributes for describing instances are equally treated. However, for a real world problem some attributes may be less important than others, and some attributes may even be irrelevant. Therefore, many *k*-NN based classifiers parameterise the similarity function (or the distance function) to deal with irrelevant attributes (see e.g. VDM [23], CCF [5], MVDM [4], MI [6], Relief-F [13], and *k*-NN$_{VSM}$ [25] etc.). Intuitively, more important attributes will be assigned higher weights, and less important attributes will be assigned lower weights. In reality, an attribute weight setting algorithm is needed for a weighted *k*-NN based classifier. In [26], a survey and empirical analysis of such algorithms is provided. For an attribute-weighted *k*-NN classifier, similarity between instances can be calculated through formula (2). In formula (2) $W_i$ is the weight on the $i$th attribute.

$$Simi(X,Y)=\sum_{i=1}^{q} W_i * Simi(X_i,Y_i) \qquad (2)$$

In this paper, we propose a novel attribute weight setting method using quadratic programming, which is particularly suitable for binary classification problems. Compared with previously proposed methods, our method has the following advantages. 1) Our method has a sound theoretical foundation, while most other methods are empirical. Other theoretical works on attribute weight setting can be found in [18], [20] and [14]. 2) From our experiments, the performance of our method is even higher than previously proposed attribute weight setting methods. 3) Our method can use a small-size training set, and still get a good performance.

[1] The authors are with the Department of Computer Science, the University of Liverpool, Liverpool L69 3BX, UK, emails: {lzhang, frans, phl}@csc.liv.ac.uk.

## 2. SETTING ATTRIBUTE WEIGHTS VIA QUADRATIC PROGRAMMING

### 2.1 Attribute Weight Setting Problem

The attribute weight setting problem in a weighted *k*-NN classifier can be described as follows. There are $n$ training instances in the training set, each having a value in each of $q$ attributes and being assigned to a class. These training instances will be used to calculate a set of attribute weights that will make the classifier achieve a high performance when using the weights and the training instances to classify new instances.

Supposing we already have a set of attribute weights, the set of weights will be used in formula (2) to calculate the similarity between two instances. The optimal set of attribute weights is a set of weights that when using the set of attribute weights to classify some new instances, the smallest number of misclassified instances can be achieved. Obviously, the optimal attribute weight setting is related to both the training instances and the instances under classification. Therefore, the optimal attribute weights cannot be calculated by only using the training set. However, if we can assume that the training instances can fully represent the instances under classification, it seems possible to get the set of optimal attribute weights that can achieve the smallest prediction error only using the training set itself. This is the attribute weight setting problem we will discuss in this paper.

### 2.2 Quadratic Programming

A quadratic programming (QP) problem is a particular case of an optimisation problem, which is to calculate the maximum or minimum value of an objective function of a set of variables subject to a set of constraints on the variables. For a quadratic programming problem, each of the constraints is a linear equation or a linear inequality, and the objective function is at most quadratic [9]. Therefore, a QP problem can be represented in the following form.

$$\begin{aligned}
&\text{maximise or minimise } \sum_{j=1}^{n} c_j x_j + \sum_{j=1}^{n}\sum_{k=j}^{n} C_{jk} x_j x_k \\
&\text{subject to } \text{Constraint}_i \ (i = 1,2,\ldots m) \\
&\qquad \text{Constraint}_i \text{ is of one of the three forms :} \qquad (3)\\
&\qquad \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \text{ or} \\
&\qquad \sum_{j=1}^{n} a_{ij} x_j = b_i, \text{ or} \\
&\qquad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \\
&\qquad x_j \geq 0 \ \ (j = 1,2,\ldots n)
\end{aligned}$$

In (3), $x_1, x_2, \ldots x_n$ are the *variables*; $\sum_{j=1}^{n} c_j x_j + \sum_{j=1}^{n}\sum_{k=j}^{n} C_{jk} x_j x_k$ is the *objective function*; $n$ is the number of *variables*; and $m$ is the number of *constraints*.

Quadratic programming is a well-studied area in optimisation, and there have already been commercial software packages (such as IBM OSL [12]) to solve quadratic programming problems. Although there is lack of theoretical analysis of the complexity of quadratic programming, it is shown in [17] that current quadratic programming software is able to solve problems with several thousand variables and several thousand constraints.

## 2.3 Formalising Attribute Weight Setting for Binary Classification as Quadratic Programming

In this paper, our focus is on a particular subset of classification problems – binary classification. In a binary classification problem, each instance will be classified between two classes. In this section, we will demonstrate how the attribute weight setting problem in *k*-NN based binary classification can be reduced to a quadratic programming problem.

### 2.3.1 *Assumption for the method*

When using a *k*-NN based classifier to classify instances, the classifier will classify an instance to the class of a similar instance. However, if there are many irrelevant attributes and/or different attributes have different importance, the calculated similarities may not reflect the real similarities. It is the responsibility of an attribute weight setting method to acquire the set of weights that can make the similarities calculated from formula (2) approximate to the real similarities.

For a binary classification problem, let us assume that the real similarity between instances in the same class is 1, and the real similarity between instances in different classes is 0. Then, the training process is to seek a set of weights, which, when applied to instances in the training set through formula (2), will lead to similarities that approximate to the real similarities.

### 2.3.2 *Formalisation*

Based on the assumptions, the attribute weight setting problem in *k*-NN based binary classification can be viewed as an optimisation problem whose aim is to minimise the differences between the similarities calculated from formula (2) and the real similarities obtained by comparing the classes. However, as there is only one objective function in an optimisation problem, all the differences between corresponding pairs of similarities should be summed into the objective function. As the aim of the optimisation problem is to minimise all the individual differences, we use the sum of the square of the differences instead of the arithmetic sum of the differences. Therefore, the optimisation problem can be summarised as the following quadratic programming problem.

Supposing there are $n$ training instances in the training set, each having $q$ attributes, the constraints in the problem can be presented in (4), in which, $S_{ijk}$ is the similarity on the $k$th attribute between instance $i$ and instance $j$, $\sum_{k=1}^{q} S_{ijk} W_k$ is the similarity between instance $i$ and instance $j$ calculated using formula (2), $R_{ij}$ is the real similarity between instance $i$ and instance $j$, $L_{ij}$ is the value by which the calculated similarity is less than the real similarity, and $M_{ij}$ is the value by which the calculated similarity is greater than the real similarity. We will call the $L_{ij}$ and $M_{ij}$ as difference variables.

$$\sum_{k=1}^{q} S_{ijk} W_k + L_{ij} - M_{ij} = R_{ij} \quad (i, j = 1...n, i < j) \tag{4}$$

As analysed above, the objective function in this problem is to minimise the sum of the square of each $L_{ij}$ and $M_{ij}$. Actually, our aim is to minimise each $L_{ij}$ and $M_{ij}$. However, as we have to express our aim in one objective function, we choose the sum of the square of each $L_{ij}$ and $M_{ij}$ to prevent any of them to be too large. Therefore, the objective function can be presented in (5).

$$\text{minimise} \sum_{i=1}^{n} \sum_{j=i+1}^{n} (L_{ij}^2 + M_{ij}^2) \tag{5}$$

### 2.3.3 *Complexity analysis*

We provide here a brief analysis of the size of this quadratic programming problem. As there is still no comprehensive analysis of the complexity of quadratic programming problems, this analysis can only indicate how large this quadratic programming problem can be, but not the actual complexity of the problem.

From the above formalisation, there are $q$ weight variables and $n*(n-1)$ difference variables. The number of the constraints in (4) is $n*(n-1)$. Therefore, the above quadratic programming problem is a quadratic programming problem with $n*(n-1)+q$ variables and $n*(n-1)$ constraints.

### 2.3.4 *Simplification*

From the above complexity analysis, the size of the formalised quadratic programming problem is not linear to the number of training instances. Therefore, with the increase of the number of the training instances, the formalised quadratic programming problem may become unmanageable. In such a case, some simplification mechanism can be applied to reduce the size of the quadratic programming problem, which may also reduce the performance of the classifier in some extent.

In the above formalisation, each instance is compared with all the other instances. If we only compare each instance with a subset of other instances, we can reduce the size of the formalised quadratic programming problem. Supposing an instance is only compared with another $p$ ($p \le n-1$) instances, we will have $q$ weight variables and $n*p$ difference variables, and the number of constraints will be $n*p$. Therefore, it becomes a quadratic programming problem with $n*p+q$ variables and $n*p$ constraints.

# 3. EXPERIMENTAL RESULTS

## 3.1 Experimental Method

To test the performance of our method, we have applied our method to four binary classification data sets, acquired from UCI Machine Learning Repository [1]. For each data set, we randomly choose 100 instances in the data set as the training data. Based on the 100 instances, we calculate the attribute weight setting using our method without any simplification. Then, the calculated setting will be used to classify the remaining instances in the data set. The accuracy of classifying the remaining instances is recorded, and as the comparison, the accuracy of standard *k*-NN classification without using any attribute weights is also recorded. When calculating the attribute weight setting, we generate the corresponding quadratic programming problem in MPS format [19], and use IBM OSL [12] to solve the problem and get the weight setting. To avoid occasional results, we perform the experiment ten times for each data set. All the experiments are performed on a Pentium III 500MHz PC with 128M RAM running Windows NT 4.0. The CPU time of each weight setting calculation

is also recorded as the training time to indicate the manageability of each formalised quadratic programming problem. In a *k*-NN based classifier, the value of *k* will also affect the accuracy of classification. As we are not aiming at investigating the effect of *k* to our method, we just select different *k* for each different data set to let the basic non-parameterised classifier get the best result in the first experiment for the data set.

## 3.2 Descriptions of the Tested Data Sets

The four data sets are the Mushroom data set, the Congressional Voting data set, the Breast Cancer data set, and the Pima Indians Diabetes data set. The characteristics of the four problems are summarised in table 1.

**Table 1.** Characteristics of the four tested problems

| Data Set | Number of Instances | Number Attributes |
|---|---|---|
| Mushroom | 8124 | 22 |
| Congressional Voting | 435 | 16 |
| Breast Cancer | 286 | 9 |
| Pima Indians Diabetes | 768 | 8 |

In the Mushroom data set, an instance represents one type of mushrooms. Each instance is characterised by 22 attributes, and each instance is classified as either edible or poisonous. There are 8124 instances in the data set in total. In the Congressional Voting data set, an instance represents one vote. Each instance is characterised by 16 attributes, and each instance is classified as either for democrats or for republicans. There are 435 instances in the data set in total. In the Breast Cancer data set, an instance represents one case of breast cancer. Each instance is characterised by 9 attributes, and each instance is classified as either recurrent or not recurrent. There are 286 instances in the data set in total. In the Pima Indians Diabetes data set, an instance represents one report of diabetes test. Each instance is characterised by 8 attributes, and each instance is classified as either positive or negative. There are 768 instances in the data set in total.

## 3.3 Results on Less Noisy Data Sets

For the Mushroom data set, and the Congressional Voting data set, there is not much noise in the data sets. Therefore, there have been highly accurate methods reported in the literature, and the standard instance based classification can also solve the problems with high accuracy. Previous results on the Mushroom data set can be found in [21] and [11]. Previous results on the Congressional Voting data set can be found in [21] and [26]. The results on the Mushroom data set are summarised in Table 2. We select *k* as 1 when experimenting on this data set.

**Table 2.** Results on the Mushroom data set

| Quadratic Programming | Standard *k*-NN | Training Time (seconds) |
|---|---|---|
| 99.40% | 95.96% | 1225 |
| 98.95% | 98.77% | 2291 |
| 99.50% | 96.83% | 2155 |
| 98.50% | 96.07% | 1628 |
| 98.60% | 97.54% | 2110 |
| 99.00% | 98.90% | 2240 |
| 98.95% | 98.40% | 2057 |
| 98.75% | 98.73% | 2292 |
| 96.11% | 95.78% | 1586 |
| 98.90% | 98.02% | 1819 |

On average, the accuracy of using quadratic programming while setting attribute weights is 98.67%, while the accuracy of not using attribute weights is 97.50%. The standard deviations are 0.9 and 1.2 percentage points respectively. There is an increase in accuracy of 1.17 percentage points. The average training time for the quadratic programming approach is 1940 seconds with the standard deviation of 343 seconds. The results on the Congressional Voting data set are summarised in Table 3. We select *k* as 9 when experimenting on this data set.

**Table 3.** Results on the Congressional Voting data set

| Quadratic Programming | Standard k-NN | Training Time (seconds) |
|---|---|---|
| 97.01% | 92.54% | 990 |
| 94.93% | 91.64% | 742 |
| 95.52% | 88.96% | 1232 |
| 94.93% | 87.76% | 1467 |
| 95.82% | 90.45% | 1068 |
| 96.12% | 92.54% | 1320 |
| 96.42% | 93.13% | 1418 |
| 95.52% | 89.25% | 1181 |
| 95.22% | 90.45% | 1163 |
| 95.22% | 89.55% | 1011 |

On average, the accuracy of using quadratic programming while setting attribute weights is 95.67%, while the accuracy of not using attribute weights is 90.63%. The standard deviations are 0.6 and 1.7 percentage points respectively. There is an increase in accuracy of 5.04 percentage points. The average training time for the quadratic programming approach is 1159 seconds with the standard deviation of 206 seconds.

## 3.4 *Results on Noisy Data Sets*

There has been much noise in the Breast Cancer data set and the Pima Indians Diabetes data set. Therefore, there has been no accurate method reported in the literature and the standard *k*-NN classifier can only solve the problems with low accuracy. Previous results on the Breast Cancer data set can be found in [16], [2], [3] and [24]. Previous results on the Pima Indians Diabetes data set can be found in [22]. The results on the Breast Cancer data set are summarised in Table 4. We select *k* as 17 when experimenting on this data set.

**Table 4.** Results on the Breast Cancer data set

| Quadratic Programming | Standard k-NN | Training Time (seconds) |
|---|---|---|
| 75.27% | 74.73% | 373 |
| 73.12% | 70.43% | 574 |
| **67.74%** | **67.74%** | **640** |
| 72.58% | 71.51% | 498 |
| 71.51% | 70.97% | 530 |
| 75.27% | 74.19% | 396 |
| 76.34% | 74.73% | 454 |
| 72.58% | 69.89% | 541 |
| 76.88% | 72.04% | 553 |
| 74.19% | 72.04% | 559 |

On average, the accuracy of using quadratic programming while setting attribute weights is 73.55%, while the accuracy of not using attribute weights is 71.83%. The standard deviations are 2.6 and 2.1 percentage points respectively. There is an increase in accuracy of 1.72 percentage points. The average training time for the quadratic programming approach is 512 seconds with the standard deviation of 78 seconds. The results on the Pima Indians Diabetes data set are summarised in Table 5. We select *k* as 13 when experimenting on this data set.

**Table 5.** Results on the Pima Indians Diabetes data set

| Quadratic Programming | Standard k-NN | Training Time (seconds) |
|---|---|---|
| 75.60% | 74.10% | 334 |
| 73.80% | 71.86% | 380 |
| 74.25% | 72.46% | 412 |
| 73.05% | 69.91% | 385 |
| 74.40% | 72.01% | 380 |
| 70.51% | 69.76% | 396 |

| 72.60% | 74.10% | 390 |
|---|---|---|
| 73.80% | 70.96% | 401 |
| 74.55% | 70.96% | 429 |
| 73.50% | 69.61% | 431 |

On average, the accuracy of using quadratic programming while setting attribute weights is 73.61%, while the accuracy of not using attribute weights is 71.57%. The standard deviations are 1.3 and 1.6 percentage points respectively. There is an increase in accuracy of 2.04 percentage points. The average training time for the quadratic programming approach is 394 seconds with the standard deviation of 27 seconds.

## 3.5 Analysis of the Experimental Results

Based on the above experimental results, we can find the following merits of our method.

### *3.5.1 Stable increase in accuracy*

For an attribute setting method for *k*-NN, the basic evaluation is the increase over the standard *k*-NN. For the two less noisy data sets, our method achieves increase in accuracy in all the 20 experiments unanimously. For the two noisy data sets, our method achieves increase in accuracy in 18 experiments, zero increase in one experiment and some decrease in another experiment. On average, our method achieves increase in accuracy for all the four data sets. It has been shown in [26] that any of the previous weighted approaches analysed in that paper may achieve decrease in accuracy for some data sets. However, our method seems more stable in achieving increase in accuracy for different data sets. It should be noted here that as there is no irrelevant attribute in any of the four data sets, it is natural that our method does not achieve dramatic increase in accuracy for any of them. In [26], six data sets with no irrelevant attribute are tested against six methods. The accuracies of the standard *k*-NN for the six data sets are respectively 72.7±0.4 (LED-7), 82.1±0.4 (Waveform-21), 82.4±0.8 (Cleveland), 92.6±0.7 (Congressional Voting), 84.2±0.3 (Isolet), and 69.6±0.2 (NETtalk). A summary of the increases achieved in [26] is shown in Table 6. The average increases in accuracy in percentage points for the six data sets are respectively 1.88 (Relief-F [13]), 1.75 (*k*-$NN_{VSM}$ [25]), -0.22 (CCF [5]), 0.55 (VDM [23]), 1.88 (MVDM [4]), and 2.00 (MI [6]). Our method achieves an average increase of 2.49 percentage points on the tested four data sets.

**Table 6.** Previous results on data sets without irrelevant attributes

| Data Set | Relief-F | *k*-$NN_{VSM}$ | CCF | VDM | MVDM | MI |
|---|---|---|---|---|---|---|
| LED-7 | -1.0 | 0.0 | -1.5 | -1.4 | -1.3 | -1.2 |
| Waveform-21 | 0.3 | -0.5 | -6.1 | -3.7 | -3.9 | 0.5 |
| Cleveland | -0.5 | 0.0 | -1.3 | 0.2 | 0.7 | -0.6 |
| Congressional Voting | 2.9 | 2.5 | 1.0 | 2.1 | 2.1 | 2.0 |
| Isolet | 0.4 | 1.9 | -1.1 | -3.9 | 1.6 | 1.6 |
| NETtalk | 9.2 | 6.6 | 7.7 | 10.0 | 12.1 | 9.7 |

### *3.5.2 Bearable training time*

The idea of using optimisation for machine learning has already been proposed in the literature (see e.g. [15] and [10]). The main drawback of this kind of approaches is they usually need much computation, which may mean long training time. In our experiments, the average training times for the four data sets are respectively 32.3 minutes (Mushroom), 19.3 minutes (Congressional Voting), 8.5 minutes (Breast Cancer), and 6.6 minutes (Pima Indians Diabetes). Any of the above training times is bearable, and it can be predicted that the training time of our method for larger training set and/or data sets with more attributes should also be bearable.

### *3.5.3 Competitive performance*

In both [21] and [11], the Mushroom data set is used to evaluate the methods proposed in the corresponding papers. Both accuracies acquired in the papers are approximately 95%. Both are lower than the accuracy achieved by our method, which is 98.67%. However, our result is a little lower than the result achieved by RISE [8], which is 100%.

In [21], the STAGGER method is also applied to the Congressional Voting data set and the accuracy is between 90% and 95%. In [26], six weighted *k*-NN approaches are tested on the Congressional Voting data set and the accuracies are respectively 95.5% (Relief-F [13]), 95.1% (*k*-$NN_{VSM}$ [25]), 93.6% (CCF [5]), 94.7% (VDM [23]), 94.7% (MVDM [4]) and 94.6% (MI [6]). The accuracy of RISE [8] is 95.2%. All the above accuracies are somewhat lower than the accuracy achieved by our method, which is 95.67%.

The Breast Cancer data set is tested in [16], [2], [3] and [24]. The accuracy range acquired in [16] is 66%-72%. The accuracy range acquired in [3] is 65%-72%. The accuracy range acquired in [24] is 68%-73.5%. The accuracy acquired by RISE [8] is 67.7%. All the above accuracies are somewhat lower than the accuracy achieved by our method, which is 73.55%. The accuracy achieved in [2] is 78%, a little higher than that of our method.

The accuracy acquired on the Pima Indians Diabetes data set in RISE [8] is 70.4%, which is lower than our method. The accuracy in [22] is 76%, which is a little higher than that of our method, which is 73.61%. Please note that *k*-NN based approaches usually have lower performance than other approaches [26]. The above comparison of performance is summarised in Table 7.

**Table 7.** Comparison of average performance

| Method | Mushroom | Congressional Voting | Breast Cancer | Pima Indians Diabetes |
|---|---|---|---|---|
| Ours | 98.67% | 95.67% | 73.55% | 73.61% |
| [11] | 95% | - | - | - |
| [21] | 95% | 90%-95% | - | - |
| [26] | - | 93.6%-95.5% | - | - |
| [16] | - | - | 66%-72% | - |
| [3] | - | - | 65%-72% | - |
| [24] | - | - | 68%-73.5% | - |
| [2] | - | - | 78% | - |
| [22] | - | - | - | 76% |
| [8] | 100% | 95.2% | 67.7% | 70.4% |

### *3.5.4 Small training sets*

Intuitively, our method can take all the comparisons between any two training instances into consideration when training, and therefore, our method may acquire enough knowledge from less training instances to get a good performance. The experimental results also indicate this merit.

In [21], 1000 instances are used as training instances for the Mushroom data set. In our experiments, our method uses 100 training instances and the accuracy is higher that the accuracy achieved in [21].

In [26], 305 instances are used as training instances for the Congressional Voting data set to test the six weighted *k*-NN approaches. In our experiments, our method uses 100 training instances and the accuracy of our method is higher than that of any of the six methods.

In [22], 576 instances are used as training instances for the Pima Indians Diabetes data set to predict the rest 192 instances. In our experiments, our method uses only 100 training instances and achieves a little lower accuracy. In Table 8, there is a summary of the above comparison.

**Table 8.** Comparison of training set sizes

| Data Set | Previous Methods | Our Method |
|---|---|---|

| | Training Set Size | Average Accuracy | Training Set Size | Average Accuracy |
|---|---|---|---|---|
| Mushroom | 1000 | 95% | 100 | 98.67% |
| Congressional Voting | 305 | 93.6%-95.5% | 100 | 95.67% |
| Pima Indians Diabetes | 576 | 76% | 100 | 73.61% |

## 4. LIMITATION

The main limitation of our method may be the assumption that the real similarity between instances in the same class is 1 and the real similarity between instances in different classes is 0. Although this seems natural for many problems, it is not always the case. For example, when classifying instances as normal or abnormal, those normal instances should be similar in nature, but those abnormal instances may not be similar at all. Furthermore, instances in different classes may not be totally dissimilar to each other. This is more the case when the problem is not a binary classification problem. It is always in nature that some classes are somewhat similar to each other while other classes are dissimilar to each other. Due to this reason, we think this method can achieve good performance only on binary classification problems, and there should be some adaptation before applying it to other classification problems.

## 5. CONCLUSION

There have been quite a few attribute weight setting algorithms for *k*-NN reported in the literature. In this paper, we proposed a new attribute weight setting method for *k*-NN based binary classification using quadratic programming. We also performed a series of experiments on four previously known binary classification problems. Besides that our method has a solid theoretical foundation, the experimental results also show that our method has the following merits: a stable increase in accuracy over standard *k*-NN, bearable training time, a good performance compared with other methods, and the ability to deal with small training sets.

## ACKNOWLEDGEMENTS

The work in this paper was supported by the UK DTI under the Foresight 'LINK' programme (FLA009). Our thanks are due to Colin Johnson and John Huckssteppe of Stoves PLC, Mike Delves of NA Software Ltd., and Stan Price for their assistance with the project.

## REFERENCES

[1] C. L. Blake and C. J Merz, UCI Repository of Machine Learning Databases [http://www.ics.uci.edu/~mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998.

[2] G. Cestnik, I. Konenenko, and I. Bratko, 'Assistant-86: A Knowledge-Elicitation Tool for Sophisticated Users', In Proceedings of the 2nd European Working Session on Learning. I. Bratko & N. Lavrac (Eds.) Progress in Machine Learning, 31-45, Sigma Press. (1987).

[3] P. Clark, and T. Niblett, 'Induction in Noisy Domains', In Proceedings of the 2nd European Working Session on Learning. I. Bratko & N. Lavrac (Eds.) Progress in Machine Learning, 11-30, Bled, Yugoslavia: Sigma Press. (1987).

[4] S. Cost, and S. Salzberg, A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. Machine Learning, 10, 57-78. (1993).

[5] R. H. Creecy, B. M. Masand, S. J. Smith, and D. L. Waltz, 'Trading Mips and Memory for Knowledge Engineering', Communications of the ACM, vol. 35, pp. 48-64, (1992).

[6] W. Daelemans, S. Gills, and G. Durieux, Learnability and Markedness in Data-Driven Acquisition of Stress (Technical Report 43). Tilburg, Netherlands: Tilburg University, Institute for Language Technology and Artificial Intelligence, 1993.

[7] B. V. Dasarathy, Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques. Los Alamitos, CA: IEEE Computer Society Press, 1991.

[8] Domingos P. (1996). Unifying Instance-Based and Rule-Based Induction, Machine Learning, 24(2), pages 141-168.

[9] R. Fourer, Nonlinear Programming Frequently Asked Questions, [http://www-unix.mcs.anl.gov/otc/Guide/faq/nonlinear-programming-faq.html], 2001.

[10] A. J. Grove, N. Littlestone, and D. Schuurmans, 'General Convergence Results for Linear Discriminant Updates', In Proceedings of the COLT 97, pp. 171-183, ACM Press. (1997).

[11] W. Iba, J. Wogulis, and P. Langley, 'Trading off Simplicity and Coverage in Incremental Concept Learning', In Proceedings of the 5th International Conference on Machine Learning, 73-79. Ann Arbor, Michigan: Morgan Kaufmann. (1988).

[12] IBM, Optimization Solutions and Library (Version 3), [http://www6.software.ibm.com/sos/osl/optimization.htm], 2001.

[13] I. Kononenko, 'Estimating Attributes: Analysis and Extensions of RELIEF', In Proceedings of the 1994 European Conference on Machine Learning, pp. 171-182, Catania, Italy: Springer Verlag. (1994).

[14] C. X. Ling and H. Wang, 'Computing Optimal Attribute Weight Settings for Nearest Neighbor Algorithms', Artificial Intelligence Review, vol. 11, pp. 255-272, (1997).

[15] N. Littlestone, 'Learning Quickly When Irrelevant Attributes Abound: A New Linear-Threshold Algorithm', Machine Learning, vol. 2, no. 4, pp. 285-318, (1988).

[16] R.S. Michalski, I. Mozetic, J. Hong, and N. Lavrac, 'The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains', In Proceedings of the Fifth National Conference on Artificial Intelligence, pp. 1041-1045, Philadelphia, PA: Morgan Kaufmann. (1986).

[17] H. D. Mittelmann, 'Benchmarking Interior Point LP/QP Solvers', Optimization Methods and Software 12, 655-670, (1999).

[18] M. Mohri and H. Tanaka, 'An Optimal Weighting Criterion of Case Indexing for Both Numeric and Symbolic Attributes', In Aha (ed.) Case-Based Reasoning: Papers from the 1994 Workshop, Menlo Park, CA: AAAI Press. (1994).

[19] B. Murtagh, Advanced Linear Programming: Computation and Practice. McGraw-Hill, 1981.

[20] K. Satoh and S. Okamoto, 'Toward PAC-Learning Of Weights from Qualitative Distance Information', In Aha (ed.) Case-Based Reasoning: Papers from the 1994 Workshop, Menlo Park, CA: AAAI Press. (1994).

[21] J.S. Schlimmer, Concept Acquisition Through Representational Adjustment (Technical Report 87-19). Doctoral dissertation, Department of Information and Computer Science, University of California, Irvine. 1987.

[22] J.W. Smith, J.E. Everhart, W.C. Dickson, W.C. Knowler, and R.S. Johannes, 'Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus', In Proceedings of the Symposium on Computer Applications and Medical Care, pp. 261--265. IEEE Computer Society Press. (1988).

[23] C. Stanfill and D. Waltz, 'Toward Memory-Based Reasoning', Communications of the ACM, vol. 29, pp. 1213-1228, (1986).

[24] M. Tan, and L. Eshelman, 'Using Weighted Networks to Represent Classification Knowledge in Noisy Domains', In Proceedings of the Fifth International Conference on Machine Learning, pp. 121-134, Ann Arbor, MI. (1988).

[25] D. Wettschereck, A Description of the Mutual Information Approach and the Variable Similarity Metric (Technical Report 944). Sankt Augustin, Germany, German National Research Center for Computer Science, Artificial Intelligence Research Division. 1995.

[26] D. Wettschereck, D. W. Aha, and T. Mohri, 'A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms', Artificial Intelligence Review, vol. 11, pp. 273-314, 1997.