Learning Weight Intervals for Classification

Derek Bridge Alex Ferguson

Department of Computer Science,
University College, Cork
d.bridge@cs.ucc.ie a.ferguson@cs.ucc.ie

Abstract. In this paper, we introduce a new case-based classifier. Its novelty lies in that it uses a *pair* of weight vectors when computing and comparing distances between cases. Given a case base and a query, the classifier induces a partial order on the case base and returns the maxima of the partial order. The maxima is a set whose size reflects the uncertainty in the weights. This means we do not need to fix a value of k as is done in traditional k-NN-classifiers. The paper also describes a learning algorithm which is capable of learning the pair of weight vectors. We compare our classifier with a k-NN-classifier that uses a vector of weights learned by a variant of the RELIEF-F algorithm.

1 Introduction

We begin, in Section 2, by describing k-NN classifiers, as a way of laying down the notation used in this paper. In Section 3, we describe a weight learning algorithm from the literature. It learns a weight vector that can be used by the k-NN-classifier. We describe it in detail since our new weight learning algorithm is based on it. Section 4 contains a description of our new classifier, which uses a *pair* of weight vectors. An algorithm for learning the pair of weight vectors is given in Section 5. In Section 6, we discuss some experimental results.

2 k-Nearest Neighbours Classification

We assume each case $x = \langle x_1, \ldots, x_n, x_c \rangle$ is defined by n descriptive attributes and a target class, $x_c \in C$. In some cases, the values of one or more attributes x_i may be missing, for which we write $x_i = \perp$. A case base CB is a set of cases. Given a query $q = \langle q_1, \ldots, q_n \rangle$ and a case base CB, a k-NN classifier retrieves the k least distant cases to predict the class of q.

Total distance is a weighted sum of local (attribute-specific) distances:

$$\text{dist}(x, q) \hat{=} \frac{\sum_{i=1}^{n} (w_i \times \text{dist}_i(x_i, q_i))}{\sum_{i=1}^{n} w_i} \tag{1}$$

You can see that we are using Manhattan distances. You can also see that, in this paper, we are looking only at global, *per-attribute* weights. The ideas that we introduce might extend naturally to finer-grained, local weighting schemes [3], [7], but this has not been explored yet.

In the experiments reported in Section 6, we compute normalised local distances differently for symbolic (including Boolean) attributes

$$\text{dist}_i(x_i, q_i) \hat{=} \begin{cases} 0 & : \quad x_i \neq \perp \text{ and } q_i \neq \perp \text{ and } x_i = q_i \\ 1 & : \quad \text{otherwise} \end{cases} \tag{2}$$

and for continuous numerical attributes

$$\text{dist}_i(x_i, q_i) \hat{=} \begin{cases} \frac{\text{abs}(x_i - q_i)}{\text{range}_i} & : \quad x_i \neq \perp \text{ and } q_i \neq \perp \\ 1 & : \quad \text{otherwise} \end{cases} \tag{3}$$

(range_i is used to normalise the distance and is defined as the difference between the maximum and minimum values observed in the data set for attribute i.)

There are, of course, many other ways of defining distance (see, e.g., [10]) and many other ways of handling missing attribute values (see, e.g., [6]).

The distance function and query q define an ordering over cases in the case base: case x is higher in the ordering than case y iff the distance between x and q is less than the distance between y and q:

$$x > y \text{ iff } \text{dist}(x, q) < \text{dist}(y, q) \tag{4}$$

Let *Max*, where $|Max| = k$, be the k highest cases in this ordering of the case base. The class of q is predicted from *Max*. Here are two, among many similar, approaches for doing this:

- A majority vote can be used, i.e. the class occurring most often among the cases in *Max* is predicted to be the class of q.
- Alternatively, this voting can be weighted so that cases closer to q have a higher vote. In this case, the vote for each class $c \in C$ is computed as follows:

$$\frac{\sum_{x \in Max \wedge x_c = c} K(x, q)}{\sum_{x \in Max} K(x, q)} \tag{5}$$

where the kernel function K is defined as follows:

$$K(x, q) \hat{=} \frac{1}{\text{dist}(x, q)} \tag{6}$$

In our experiments, reported in Section 6, we will use the second of these approaches as we found it to give equal or marginally better performance on all data sets.

By default, k-NN classifiers use equal, non-zero weights: $\forall w_i \forall w_j (w_i > 0 \wedge w_i = w_j)$. If we allow the weights to differ, we can increase the contribution of more predictive attributes and decrease the contribution of less relevant, including redundant, attributes. We look now at one way of learning such a weight vector.

3 Learning a Weight Vector

There are many weight learning algorithms. We choose to use an online optimizer [9]. An early example of such a learner is Salzberg's EACH algorithm [8]. This influenced Aha's IB4 [1]. And this, in turn, influenced Kira's and Rendell's RELIEF algorithm [5], which removed IB4's assumption of a uniform distribution of irrelevant attribute values. Kononenko experimented with variants of the RELIEF algorithm [6]. A minor variant of Kononenko's RELIEF-F algorithm was adopted for use in Wettschereck et al's classic lazy learner study [9], and it is this that we use here (although we make one further small change, reported below). This algorithm is also the basis for our new algorithm (Section 5).

Assume to begin with that there are only two classes, $|C| = 2$. This simplifies the presentation, and we will lift this assumption shortly.

In the RELIEF algorithm, each weight is initially 0.5. A training set T of size $|T|$ is used for supervised learning of the weights. Each training case $t \in T$ is used to query a case base. The algorithm retrieves the closest positive case (the one that is least distant and has the *same* class as the training case) and the closest negative case (the one that is least distant and has a *different* class from the training case). Call these the *nearest hit* and *nearest miss* and denote these cases by h and m, respectively.

Weights are updated using the following rule:

$$w_i := w_i - \frac{\text{dist}_i(t_i, h_i)}{2|T|} + \frac{\text{dist}_i(t_i, m_i)}{2|T|} \tag{7}$$

(Given that weights are initialised to 0.5 and that distances are normalised to $[0, 1]$, division by $2|T|$ ensures that, even after weights have been updated $|T|$ times, they will remain in $[0, 1]$.[1])

The rationale behind this update rule is to change the weights to decrease distances to cases having the same class, and increase distances to cases having the other class.

The original RELIEF algorithm selects training examples from the training set at random. We follow [9] in choosing each training example only once, as shown in Figure 1.

for $i := 1$ to n
  $w_i := 0.5$
$CB := T$
for each t in T
  $h := t$'s nearest hit in $CB \setminus \{t\}$
  $m := t$'s nearest miss in $CB \setminus \{t\}$
  for $i := 1$ to n
    $w_i := w_i - \frac{dist_i(t_i, h_i)}{2|T|} + \frac{dist_i(t_i, m_i)}{2|T|}$

Fig. 1. Sequential RELIEF Algorithm

Kononenko investigates variants of the original RELIEF algorithm, which he refers to as RELIEF-A through RELIEF-F [6]. Kononenko's RELIEF-B through RELIEF-D provide alternative ways of handling missing attribute values. We do not adopt any of his methods here. We indicated in the previous section the rudimentary treatment we give to undefined attributes. Investigating the best ways of handling undefined attributes is not a focus of our paper. Instead, we take ideas from Kononenko's RELIEF-A and RELIEF-F.

In RELIEF-A, Kononenko proposes that the reliability of the learned weights can be improved if the system retrieves the k ($k > 1$) nearest hits and misses, instead of a single hit and miss. Average distances are then used to update

[1] In fact, in the original RELIEF algorithm, weights are initially 0.0 and the weight update rule divides only by $|T|$, not $2|T|$. Hence, weights in that algorithm are in $[-1, 1]$.

weights. So, let H be t's k nearest hits and let M be t's k nearest misses, then the weight update rule becomes:

$$w_i := w_i - \frac{\text{avg}_{h \in H} \text{dist}_i(t_i, h_i)}{2|T|} + \frac{\text{avg}_{m \in M} \text{dist}_i(t_i, m_i)}{2|T|} \tag{8}$$

In RELIEF-E and RELIEF-F, Kononenko removes the assumption that there are only two classes, $|C| = 2$. Only RELIEF-F is of interest, since Kononenko found that it outperformed RELIEF-E.

To handle more than two classes in RELIEF-F, the nearest miss *for each class* is retrieved. Weights are updated using the average distance between t and each different nearest miss, weighted by the prior probability of the class of the miss, Prob(c). But note that RELIEF-F, as Kononenko defines it, uses only one nearest hit and only one nearest miss per class.

We take this a stage further. In our implemented learning algorithm, we retrieve H, the k nearest hits and, for each class $c \in C$ that is different from t's class, we retrieve M_c, the k nearest misses. Our update rule averages over the k retrieved case distances, weighted by prior probabilities for the different misses.

$$w_i := w_i - \frac{\text{avg}_{h \in H} \text{dist}_i(t_i, h_i)}{2|T|} + \frac{\sum_{c \in C \wedge t_c \neq c} (\text{Prob}(c) \times \text{avg}_{m \in M_c} \text{dist}_i(t_i, m_i))}{2|T|} \tag{9}$$

(In the experiments reported in Section 6, estimates of the prior probabilities are computed from the datasets.)

Henceforth, we refer to this learning algorithm as Sequential k-RELIEF-F Learning (SKRFL), and we will refer to a k-NN-classifier whose weights have been learned using SKRFL as a SKRFL-classifier.

Let us now turn to our new classification algorithm and our new weight learning algorithm.

4 Classification using a Pair of Weight Vectors

Our new classifier is a case-based classifier and so the idea, as usual, is to compute distances between queries and cases in a case base; to use these to define an ordering over the case base; to retrieve the maxima of this ordering of the case base; and to predict the class of the query from these maxima. Where we differ substantially from methods in the literature is in the way we define the ordering. The novel consequence of the way we do this is that we do not need to supply a value k.

Our classifier uses two weight vectors, $v = \langle v_1, \ldots, v_n \rangle$ and $w = \langle w_1, \ldots, w_n \rangle$. We require that for all i, $v_i \leq w_i$. Hence, we will refer to v as the *lower weight vector* and w as the *upper weight vector*. The difference between a pair of weights, $w_i - v_i$, can reflect our uncertainty about the predictive power of attribute i.

Case x will be higher in the ordering than case y iff the distance between x and q *using the upper weights*, $\text{dist}^w(x, q)$, is less than the distance between y and q *using the lower weights*, $\text{dist}^v(y, q)$:

$$x > y \text{ iff } \text{dist}^w(x, q) < \text{dist}^v(y, q) \tag{10}$$

Equation 10 is 'conservative' in the following sense. For case x to be higher than case y requires that case x's total distance using the upper weights, which are the larger of the weights, nevertheless be small enough to fall below case y's total distance using the lower (smaller) weights.

A crucial observation about this ordering is that it is a *partial order*, whereas Equation 4 defines a *total order*. This is illustrated with an example. Suppose three cases, x, y and z, are compared to query q. Assume that the cases have two Boolean-valued attributes; local distances will either be 0 (equal) or 1 (different):

	x: T F	
q: T T	y: F T	
	z: F F	

Using a single weight vector, $w = \langle 0.8, 0.6 \rangle$, we get the following distances between q and the cases:

$$\begin{aligned}
\text{dist}^w(x, q) &= 0.8 \times 0 + 0.6 \times 1 = 0.6 \\
\text{dist}^w(y, q) &= 0.8 \times 1 + 0.6 \times 0 = 0.8 \\
\text{dist}^w(z, q) &= 0.8 \times 1 + 0.6 \times 1 = 1.4
\end{aligned}$$

So Equation 4 induces a total ordering on x, y and z based on closeness to q:

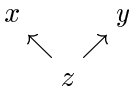
$$\begin{array}{c} x \\ \uparrow \\ y \\ \uparrow \\ z \end{array}$$

Suppose instead we use two weight vectors, $v = \langle 0.5, 0.4 \rangle$ and $w = \langle 0.8, 0.6 \rangle$:

$$\begin{aligned}
\text{dist}^v(x, q) &= 0.5 \times 0 + 0.4 \times 1 = 0.4 \\
\text{dist}^w(x, q) &= 0.8 \times 0 + 0.6 \times 1 = 0.6 \\
\text{dist}^v(y, q) &= 0.5 \times 1 + 0.4 \times 0 = 0.5 \\
\text{dist}^w(y, q) &= 0.8 \times 1 + 0.6 \times 0 = 0.8 \\
\text{dist}^v(z, q) &= 0.5 \times 1 + 0.4 \times 1 = 0.9 \\
\text{dist}^w(z, q) &= 0.8 \times 1 + 0.6 \times 1 = 1.4
\end{aligned}$$

A partial order is induced. Case x is higher in the ordering than case z because x's distance from q using the upper weights is less than z's distance from q using the lower weights ($0.6 < 0.9$). Similarly, case y is higher in the ordering than case z ($0.8 < 0.9$).

However, case x is neither higher, lower nor equal to case y. They are simply incomparable. Case x's distance from q using the upper weights is not less than case y's distance from q using the lower weights ($0.6 \not< 0.5$). But case y's distance from q using the upper weights is not less than case x's distance from q using the lower weights ($0.8 \not< 0.4$). They are incomparable, and, in some sense, equally good. Both are in the maxima of this ordering:



Call the maxima of this partial ordering *Max*. *Max* will comprise cases that are judged to be equal or incomparable in the ordering. The 'conservativeness' of Equation 10 increases the likelihood that cases will be incomparable. Therefore, our classifier does not use a value k. The fact that the maxima is a set introduces, we believe, the same kind of robustness that use of $k > 1$ introduces to k-NN.

The class of q is predicted from *Max*. For consistency with our k-NN-classifier, we use a weighted majority vote, as per the penultimate paragraph of Section 2, in which cases that are closer to the query get bigger votes. But there is a problem with this: because we have a *pair* of weight vectors, we do not have a single notion of distance to use in the kernel function. So what we have done is to compute distances using the lower weights and also using the upper weights. Their average is then used in the kernel function.

5 Learning a Pair of Weight Vectors

We have designed an algorithm, based on the one presented in Section 3, for learning the pair of weight vectors.

We initialise the weight vectors, $v_i = w_i = 0.5$. Then, as before, for each training case $t \in T$, we retrieve a set of nearest hits and, for each class different from t's class, we retrieve a set of nearest misses. Of course, in retrieving these sets, we do not use a value k. We plug the current values of v and w into Equation 10 and retrieve the maxima of the partial order that this induces on the case base.

We need two update rules, one for the lower weights and one for the upper weights. To update the lower weights, we subtract the maximum distance among the nearest hits and, for each class different from t's class, we add the minimum distance, weighted by the class probability, among the nearest misses.

$$v_i := v_i - \frac{\max_{h \in H} \mathrm{dist}_i(t_i, h_i)}{2|T|} + \frac{\sum_{c \in C \wedge t_c \neq c} (\mathrm{Prob}(c) \times \min_{m \in M} \mathrm{dist}_i(t_i, m_i))}{2|T|} \tag{11}$$

To update the upper weights, we use minimum hit distances and maximum miss distances:

$$w_i := w_i - \frac{\min_{h \in H} \mathrm{dist}_i(t_i, h_i)}{2|T|} + \frac{\sum_{c \in C \wedge t_c \neq c} (\mathrm{Prob}(c) \times \max_{m \in M} \mathrm{dist}_i(t_i, m_i))}{2|T|} \tag{12}$$

After these two rules are applied, it is still guaranteed that for all i, $v_i \leq w_i$. The lower weights have as much subtracted from them and as little added to them as possible; the upper weights have as much added and as little subtracted as possible.

Henceforth, we refer to this learning algorithm as Sequential Weight Interval Learning (SWIL) and we will refer to a classifier whose pair of weight vectors has been learned using SWIL as a SWIL-classifier.

6 Experimental Results

We compare our new SWIL-classifier to both a k-NN-classifier (whose weights are equal) and a SKRFL-classifier. These comparisons are the most meaningful ones we can carry out: the strong conceptual similarities between the three classifiers minimise the number of independent variables that might otherwise contribute to differences in the results.

The datasets come from the UCI repository [2], and their characteristics are show in Table 1. For all datasets, we used 70% of the cases as the training set and 30% as the test set.

Dataset	Training Set Size	Test Set Size	Number of Attributes	Missing Values	Number of Classes
Ecoli	235	101	7C	No	8
Glass	149	65	9C	No	7
Hepatitis	108	47	13B, 6C	Yes	2
Iris	105	45	4C	No	3
LED-7 (10% noise)	252	108	7B	No	10
LED-7 (20% noise)	252	108	7B	No	10
LED-24 (10% noise)	252	108	7B	No	10
LED-24 (20% noise)	252	108	7B	No	10
Votes	304	130	16B	Yes	2

B = Boolean; C = Continuous

Table 1. Dataset Characteristics.

Dataset	k-NN	SKRFL	SWIL
Ecoli	0·80±0·02	0·8	0·8
Glass	0·69±0·02	0·69	0·69
Hepatitis	0·80±0·02	0·8	0·82
Iris	0·94±0·02	0·94	0·94
LED-7 (10% noise)	0·63±0·02	0·63	0·5
LED-7 (20% noise)	0·37±0·02	0·35	0·29
LED-24 (10% noise)	0·38±0·02	0·47	0·47
LED-24 (20% noise)	0·24±0·02	0·26	0·27
Votes	0·91±0·0	0·93	0·93

Table 2. Experimental Results: Classification Accuracy

The classification accuracy results are shown in Table 2. The accuracies are averaged over 20 cross-validation runs. The same training sets and test sets were used for each algorithm. In the case of the k-NN-classifier, 95% confidence intervals are shown.

In the case of the k-NN-classifier, a value of k must be selected. An 'optimal' value was estimated by leave-one-out-cross-validation (LOOCV), where accuracy ties were broken in favour of smaller values of k. LOOCV might recommend a

	k-NN				SKRFL				SWIL			
Dataset	Min.	Max.	Mean	Mode	Min.	Max.	Mean	Mode	Min.	Max.	Mean	Mode
Ecoli	1	1	1·0	1	1	1	1·0	1	1	4	1·14	1
Glass	1	1	1·0	1	1	1	1·0	1	1	3	1·12	1
Hepatitis	1	2	1·3	1	1	2	1·35	1	1	18	2·87	1
Iris	1	1	1·0	1	1	1	1·0	1	1	4	1·04	1
LED-7 (10% noise)	1	2	1·65	2	1	2	1·6	2	1	31	9·43	1
LED-7 (20% noise)	1	3	1·6	2	1	3	1·85	2	1	20	5·08	1
LED-24 (10% noise)	1	1	1·0	1	1	1	1·0	1	1	10	1·47	1
LED-24 (20% noise)	1	1	1·0	1	1	1	1·0	1	1	8	1·35	1
Votes	1	2	1·3	1	1	2	1·45	1	1	304	4·76	1

Table 3. Experimental Results: Retrieval Set Sizes

different k on each of the 20 validation runs and so some data about the values of k is given in Table 3. Specifically, the minimum, maximum, mean and mode values are shown. (Where the distribution has more than one mode, the lower value is shown.)

Similar considerations apply in the case of the SKRFL-classifier. Table 3 shows the values estimated by LOOCV. However, there is an additional consideration here. As well as needing a value of k for classification (Table 3), SKRFL needs a value of k for use during learning (see Section 3). The optimal value for learning need not be the same as the optimal value for classification. This required us to use the training set to estimate optimal values of k both for learning and then for classification. The Table reports data only about the k used for classification.

The SWIL-classifier, of course, does not need a value of k, either for learning or for classification. As we have discussed in Section 4, the result of a query is, quite naturally, a set (the maxima of the partial order), whose size will vary from one query to the next. Table 3 shows the minimum, maximum, mean and mode values for the sizes of the retrieved sets for SWIL-classification.

In looking at Table 3, some care must be taken when comparing the values of k for the k-NN-classifier and SKRFL-classifier with the set sizes reported for the SWIL-classifier. For k-NN-classification and SKRFL-classification, k might differ between validation runs, but the value of k used for each query within any one run remains fixed: the same value is used for each query. So the Table reports the minimum, maximum, mean and mode values over the different validation runs.

However, for SWIL-classification, the set size can be different for each query: each query in the test set may have a different number of nearest neighbours. The Table reports the minimum, maximum, mean and mode values over the different queries, irrespective of validation run.

Looking at Table 2, there is no reason to expect the classification accuracy from our new SWIL-classifier to be better than that of the SKRFL-classifier. There is nothing in our new learning or classification algorithms that would lead us to expect higher accuracy. (The advantages lie elsewhere: in particular, the fact that we do not need to set a value of k.) Indeed, the fact that SKRFL-classification is optimised (by estimating 'optimal' k), but there is no equivalent

optimisation step in SWIL-classification, might lead us to expect lower accuracy values.

In fact, we see that SWIL performs as well as SKRFL on all but two of the datasets (LED-7 with 10% and 20% noise). Given the lack of optimisation, this is surprising and reassuring. Its lower performance on the two LED-7 datasets might be explained by the fact that these datasets are noisy. It is possible that our new weight update rules (Equations 11 and 12) are overly sensitive to this noise. They use minimum and maximum values where the original SKRFL update rule (Equation 9) uses averages. SWIL may therefore be overly sensitive to outliers in the noisy data.

This does not explain why SWIL's performance on the two other noisy datasets, LED-24 with 10% and 20% noise, is as good as that of SKRFL. The difference between the LED-7 and LED-24 datasets is the addition in LED-24 of 17 totally irrelevant attributes. SKRFL and SWIL outperform k-NN on these datasets because they learn smaller weights for these redundant attributes. Deprecating these redundant attributes seems to be the most important factor for predictive accuracy on these datasets (after all, much of the noise is dispersed into these redundant attributes).

The fact that SKRFL and SWIL rarely outperform k-NN is not too surprising. Except in the case of the LED-24 datasets, where we know that some attributes are much more predictive than others (due to these 'others' being spurious attributes), there are not stark differences in the relevances of the different attributes in the other datasets. Furthermore, Holte observes that for some of the datasets in the UCI Repository, even simple-minded schemes can perform well, and improvements on these already good results tend to be small [4].

Table 3 is also of interest. It shows that, on these mostly simple datasets, most of the time the optimised values of k used by k-NN and SKRFL are 1, and the sets sizes retrieved by SWIL are also 1.

The mean set sizes are higher for SWIL because, every now and then, a query gives rise to a much larger set of results. The starkest example of this is in the Votes dataset, where, on at least one occasion, the maxima is the whole dataset. In fact, this is explicable. This dataset records the voting histories of U.S. House of Representatives Congressmen on 16 key votes, and classifies the congressmen as Republican or Democrat. One of the congressmen has an unknown disposition recorded for all votes, and so the case records missing values ($\perp$) for all attributes. With our simple-minded treatment of missing values, this case will be equidistant from all other cases in the case base. In this situation, k-NN and SKRFL will, of course, still only retrieve k cases; SWIL will, arguably more correctly, retrieve all cases.

In fact, this example reveals another possible advantage of the SWIL-classifier. When SWIL makes a classification, it can give the user some indication of the certainty of the classification, based, in part, on the size of the retrieved set. For example, for most congressmen, SWIL's result set is of size 1 and we can have confidence in its classification. For our indolent and indecisive congressman, the result set is the whole case base: SWIL is showing that a nearest neighbour approach does not help it to make a classification. (Of course, this is a bit simplistic: the number of different classes within the result set, as well as its size, is of relevance to ascertaining SWIL's certainty about any classification.) k-NN

and SKRFL do not have this feature because they have a value of k which is fixed for all queries.

7 Conclusions

We believe that this paper introduces a promising line of research. Our accuracy results are, for the most part, as good as those for the conceptually closest competitor algorithms. We have the advantages of not needing to estimate a value of k for learning and of not needing to estimate a value of k for classification. We have the potential, when classifying any particular new instance, to give some information about the certainty of that classification, based on the size of the retrieved set and the number of distinct classes within that set.

We are not yet satisfied with the SWIL weight update rules (Equations 11 and 12). They lack proper motivation and they may make the system overly sensitive to outliers. We would hope to find a better-motivated way of learning the two weight vectors. If this is done, we can then proceed to bring in more sophisticated ways of defining distance (especially for symbolic attributes); more sophisticated treatments of missing attribute values; and local, as opposed to global, forms of weighting.

References

1. Aha, D.: Tolerating Noisy, Irrelevant, and Novel Attributes in Instance-Based Learning Algorithms, *International Journal of Man-Machine Studies*, vol.36, pp.267-287, 1992.
2. Blake, C., & Merz, C.J.: *UCI Repository of Machine Learning Databases*, `www.ics.uci.edu/~mlearn/MLRepository.html`, University of California, Department of Information and Computer Science, 1998
3. Hastie, T. & Tibishirani, R.: Discriminant Adaptive Nearest Neighbor Classification, *IEEE PAMI*, vol.18, pp.607-616, 1996
4. Holte, R.C.: Very Simple Classification Rules Perform Well on Most Commonly Used Datasets, *Machine Learning*, vol.3, pp.63-91, 1993
5. Kira, K. & Rendell, L.: A Practical Approach to Feature Selection, *Procs. of the Ninth International Conference on Machine Learning*, pp.249-256, Morgan Kaufmann, 1992
6. Kononenko, I.: Estimating Attributes: Analysis and Extensions of RELIEF, *Procs. of European Conference on Machine Learning*, pp.171-182, Springer, 1994
7. Ricci, F. & Avesani, P.: Learning a Local Similarity Metric for Case-Based Reasoning, *Procs. of First International Conference on Case-Based Reasoning*, pp.301-312, Springer, 1995
8. Salzberg, S.L.: A Nearest Hyperrectangle Learning Method, *Machine Learning*, vol.6, pp.251-276, 1991
9. Wettschereck, D., Aha, D.W., & Mohri, T.: 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
10. Wilson, D.R. & Martinez, T.R.: Improved Heterogeneous Distance Functions, *Journal of Artificial Intelligence Research*, vol.6, pp.1-34, 1997