Combining Cross-Validation and Confidence to Measure Fitness

D. Randall Wilson
fonix corporation
WilsonR@fonix.com

Tony R. Martinez
Brigham Young University
martinez@cs.byu.edu

Abstract

Neural network and machine learning algorithms often have parameters that must be tuned for good performance on a particular task. Leave-one-out cross-validation (LCV) accuracy is often used to measure the fitness of a set of parameter values. However, small changes in parameters often have no effect on LCV accuracy. Many learning algorithms can measure the confidence of a classification decision, but often confidence alone is an inappropriate measure of fitness. This paper proposes a combined measure of Cross-Validation and Confidence (CVC) for obtaining a continuous measure of fitness for sets of parameters in learning algorithms. This paper also proposes the Refined Instance-Based (RIB) learning algorithm which illustrates the use of CVC in automated parameter tuning. Using CVC provides significant improvement in generalization accuracy on a collection of 31 classification tasks when compared to using LCV.

1. Introduction

Inductive learning algorithms are typically presented with n training examples (*instances*) from a *training set*, T, during learning. In classification tasks each instance has an *input vector* $\boldsymbol{x}$, and an *output class* c. After learning is complete, these systems can be presented with new input vectors for classification, many of which were not in the original training set. The algorithm must *generalize* from the training set to determine the most likely output classification for each input vector $\boldsymbol{y}$.

Learning algorithms often have *parameters* which affect how well they are able to generalize on a particular task. Section 2 explains how leave-one-out cross-validation (LCV) can be used to tune parameters, and shows why it is limited in some cases. Section 3 proposes a continuous measure of fitness using both confidence and cross-validation (CVC). Section 4 proposes a distance-weighted *Refined Instance-Based* (RIB) learning algorithm that can use either LCV or CVC to refine various parameters. Section 5 presents empirical results on 31 classification tasks showing statistically significant improvemnt of CVC over LCV in tuning parameters in RIB.

2. Leave-one-out Cross-Validation (LCV)

One of the most popular methods of evaluating a set of parameter values is through the use of *cross-validation* [1, 2, 3]. In cross-validation, the training set T is divided into J partitions, $T_1...T_J$, and the instances in each of the partitions are classified by the instances in the remaining partitions using the proposed parameter setting. The average accuracy of these J trials is used to estimate what the generalization accuracy would be if the parameter value was used. The parameter value that yields the highest estimated accuracy is then chosen. When more than one parameter needs to be tuned, the combined settings of all of the parameters can be measured using cross-validation in the same way.

When J is equal to the number of instances in T, the result is *leave-one-out* cross-validation (LCV), in which each instance i is classified by all of the instances in T except for i itself, so that almost all of the data is available for each classification attempt. LCV has been described as being desirable but computationally expensive [2]. However, in some situations it can be performed efficiently, as illustrated in Section 4.

One problem with using LCV to fine-tune parameters in a classification system is that it can yield only a fixed number of discrete accuracy estimates. For each instance i in T, the accuracy is 1 if the instance is classified correctly and 0 if it is misclassified. Thus the average LCV accuracy over all n instances in T is r / n, where r is the number classified correctly. Since r is an integer from 0 to n, there are only $n + 1$ accuracy values possible with this measure, and often two different sets of parameter values will yield the same accuracy because they will classify the same number of instances correctly. This makes it difficult to tell which parameter values are better than another. This problem occurs quite frequently in some systems and limits the extent to which LCV can be used to fine-tune many classifiers.

3. Confidence and Cross-Validation (CVC)

An alternative method for estimating generalization accuracy is to use the *confidence* with which each instance is classified. The average confidence over all n instances in the training set can then be used to estimate which set of parameter values will yield better generalization. The confidence for each instance i is

$$conf_i = \frac{weight_{correct}}{\sum_{c=1}^{C} weight_c} \tag{1}$$

where $weight_{correct}$ is the weight received for the correct class of instance i, and $weight_c$ is the weight received for class c. This weight might be the sum from a weighted voting scheme as in distance-weighted instance-based learning algorithms or radial basis function neural networks. When the *weight* (i.e., summed activation or weighted votes) for each class is a continuous value, then confidence can provide a continuous measure of fitness for parameter settings.

After learning is complete, the confidence can be used to indicate how confident the classifier is in its generalized output. In this case the confidence is the same as defined in Equation 1, except that $weight_{correct}$ is replaced with $weight_{out}$, which is the amount of voting weight received by the class that is chosen to be the output class by the classifier. This is often equal to the maximum number of votes (or maximum sum of voting weights) received by any class, since the majority class is typically chosen as the output.

Average confidence has the attractive feature that it provides a continuously valued metric for evaluating a set of parameter values. However, it also has drawbacks that make it inappropriate for direct use in measuring parameter fitness in some learning algorithms. For example, in the *Refined Instance-Based* learning algorithm presented in Section 4, using confidence alone favors any parameter settings that give nearer neighbors more weight than further ones, even if doing so degrades accuracy.

LCV works fairly well in general but suffers from not being a continuous measure, while confidence is continuous but suffers from problems of its own when used alone as mentioned above. We therefore combine *cross-validation and confidence* into a single metric called *CVC*. CVC uses LCV to count the number of instances r classified correctly using a particular set of parameter values. It then adds the average confidence of the correct class, $avgconf$ (where $0 \le avgconf \le 1$) to the LCV accuracy to get a continuous measure of fitness. Since $0 \le r \le n$ and $0 \le conf \le 1$, the maximum sum is $n + 1$. In order to get a normalized metric in the range 0..1, the sum is therefore divided by $n + 1$. Using CVC, the accuracy estimate CVC_T of the entire training set T is therefore given as

$$CVC_T = \frac{r + avgconf}{n+1} \tag{2}$$

The CVC accuracy cvc_i of a single instance i can also be computed as

$$cvc_i = \frac{n \cdot cv + conf_i}{n+1} \tag{3}$$

where n is the number of instances in the training set; $conf_i$ is as defined in Equation 1; and cv is 1 if instance i is classified correctly by its neighbors in T, or 0 otherwise. Note that CVC_T can also be obtained by averaging cvc_i for all n instances in T.

This metric weights the cross-validation accuracy more heavily than the confidence by a factor of n. The LCV portion of CVC can be thought of as providing the whole part of the score, with confidence providing the fractional part. This metric gives LCV the ability to make decisions by itself unless multiple parameter settings are tied, in which case the confidence makes the decision.

CVC can be used with a variety of classification algorithms in which parameters need to be tuned, a measure of confidence for the correct class is available and leave-one-out cross-validation is not computationally prohibitive. Section 4 presents an *instance-based learning* algorithm with automatically-tuned parameters to illustrate the use of CVC and to provide empirical data.

4. Instance-Based Learning

Instance-Based Learning (IBL) [4] is a paradigm of learning in which algorithms typically store some or all of the n available training examples (*instances*) from the training set T during learning. During *generalization*, these systems use a distance function to determine how close a new input vector $\boldsymbol{y}$ is to each stored instance and use the nearest instance or instances to predict the output class of $\boldsymbol{y}$ (i.e., to *classify* $\boldsymbol{y}$). Some of the earliest instance-based learning algorithms are referred to as nearest neighbor techniques [5, 6].

Instance-based learning algorithms often have a parameter k that determines how many instances are used to decide the classification, as in the k-nearest neighbor rule [5]. The use of $k > 1$ can lessen the effect of noise in the system, but it also introduces a parameter to the system which must be chosen.

Dudani [7] proposed a distance-weighted nearest neighbor algorithm in which neighbors nearer to the input vector get more weight. This can reduce the sensitivity of the system to the parameter k, though a suitable value for k must still be found.

This section presents a new instance-based learning system called the *Refined Instance-Based* (RIB) learning algorithm that uses distance-weighted k-nearest neighbor voting. RIB automatically tunes several parameters, including the value of k and the kernel shape for the distance weighting function. The distance-weighted voting allows decision boundaries to be fine-tuned with more precision than is allowed with simple majority voting. RIB also uses a *heterogeneous* distance function described below in Section 4.1 that is appropriate for domains with nominal attributes, linear attributes, or both. Section 4.2 describes how distance-weighted voting is done in RIB, and Section 4.3 tells how parameters are automatically tuned in the system.

4.1. Heterogeneous Distance Function

A distance function is critical to the success of an instance-based algorithm. Euclidean Distance is the most commonly used distance function, but many applications have *nominal* (discrete, unordered) attributes for which Euclidean distance and other linear metrics are not appropriate.

The *Value Difference Metric* (VDM) [8, 9] is able to return a real-valued distance between each pair of values for nominal attributes based on statistics gathered from the training set. It does not, however, directly handle linear attributes but instead requires *discretization* [10].

We previously introduced several new heterogeneous distance functions that substantially improved average generalization accuracy on a collection of 48 different datasets [11]. RIB uses one of the most successful functions, the *Heterogeneous Value Difference Metric* (HVDM).

The HVDM distance function defines the distance between two input vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ as

$$HVDM(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{a=1}^{m} d_a(x_a, y_a)^2} \tag{4}$$

where m is the number of attributes. The function $d_a(x,y)$ returns a distance between the two values x and y for attribute a and is defined as

$$d_a(x,y) = \begin{cases} 1 & \text{if } x \text{ or } y \text{ is unknown; else...} \\ vdm_a(x,y) & \text{if } a \text{ is nominal} \\ \dfrac{|x-y|}{4\sigma_a} & \text{if } a \text{ is numeric} \end{cases} \tag{5}$$

where σ_a is the sample standard deviation of the numeric values occurring for attribute a. Since 95% of the values in a normal distribution fall within two standard deviations of the mean, the difference between numeric values is divided by four standard deviations to scale each value into a range that is usually of width 1. The function $vdm_a(x,y)$ returns the distance between two nominal attribute values x and y for attribute a as

$$vdm_a(x,y) = \sum_{c=1}^{C} \left(\frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}} \right)^2 \tag{6}$$

where $N_{a,x}$ is the number of times attribute a had value x in the training set, $N_{a,x,c}$ is the number of times attribute a had value x and the output class was c, and C is the number of output classes. Using this distance measure, two nominal attribute values are considered to be closer if they have more similar classifications, regardless of the order of the values. More details on this distance function are available in [11].

The HVDM distance function is used to find distances in the RIB algorithm, which in turn are used to weight voting and thus influence confidence, as discussed below.

4.2. Vote Weighting

Let $\boldsymbol{y}$ be the input vector to be classified and let $n_1...n_k$ be the k nearest neighbors of $\boldsymbol{y}$ in T. Let D_j be the distance from $\boldsymbol{y}$ to the jth neighbor using some distance function D (such as HVDM).

In the RIB algorithm, the voting weight of each of the k nearest neighbors depends on its distance from the input vector $\boldsymbol{y}$. The weight is 1 when the distance is 0 and decreases as the distance grows larger. The way in which the weight decreases as the distance grows depends on which *kernel function* is used. The kernel functions used in RIB are: *majority, linear, gaussian,* and *exponential*.

In majority voting, all k neighbors get an equal vote of 1. With the other three kernels, the voting weight of a neighbor n_j is 1 when the distance to n_j is 0 and drops to the value of a parameter w_k at a distance D_k, where D_k is the distance to the kth nearest neighbor.

Given w_k and D_k, the amount of voting weight w_j for the jth neighbor that is a distance D_j from the input vector for each kernel is given in Equations 7-10.

(a) Majority: $$w_j = 1 \tag{7}$$

(b) Linear: $$w_j = w_k + \frac{(1-w_k)(D_k - D_j)}{D_k} \tag{8}$$

(c) Gaussian: $$w_j = w_k^{D_j^2 / D_k^2} \tag{9}$$

(d) Exponential: $$w_j = w_k^{D_j / D_k} \tag{10}$$

Note that the majority voting scheme does not require the w_k parameter. Also note that if $k = 1$ or $w_k = 1$, then all four of these schemes are equivalent. As D_k approaches 0, the weight in Equations 8-10 all approach 1. Therefore, if the distance D_k is equal to 0, then a weight of 1 is used for consistency and to avoid dividing by 0. These four kernels are illustrated in Figure 1.

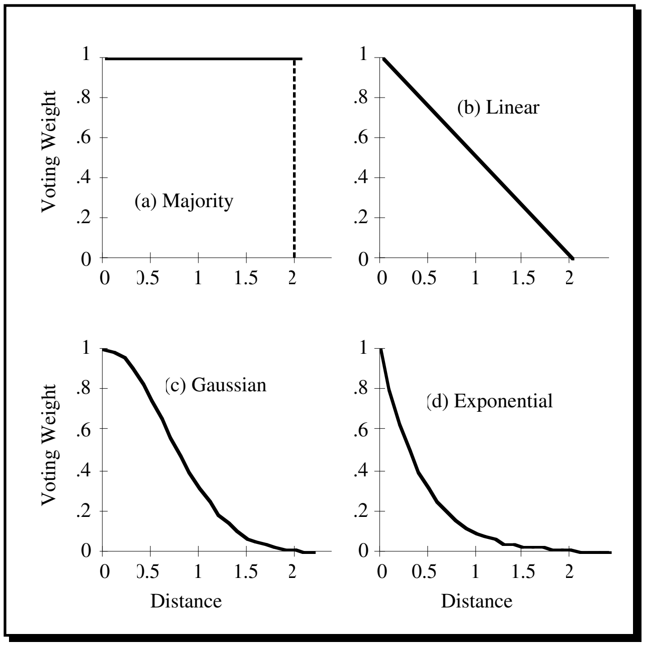

Figure 1. Distance-weighting kernels, shown with $D_k = 2.0$ and $w_k = .01$.

Sometimes it is preferable to use the *average* distance of the k nearest neighbors instead of the distance to the kth neighbor to determine how fast voting weight should drop off. This can be done by computing what the distance D'_k of the kth nearest neighbor would be if the neighbors were distributed evenly. This can be done by setting D'_k to

$$D'_k = \frac{2 \cdot \sum_{i=1}^{k} D_i}{k+1} \tag{11}$$

and using D'_k in place of D_k in Equations 8-10. When $k = 1$, Equation 11 yields $D'_k = 2D_k/2 = D_k$, as desired. When $k > 1$, this method can be more robust in the presence of changes in the system such as changing parameters or the removal of instances from the classifier.

4.3. RIB Learning Algorithm

Several parameters in have been mentioned in the above discussion without specifying how they are set. Specifically, for a given classification task, RIB must set k, the number of neighbors that vote on the class of a new input vector; *kernel*, the kernel of the distance-weighted voting function; w_k, the weight of the kth neighbor (except in majority voting); and *avgk*, the flag determining whether to use Dk or D'_k.

These parameters are set as described in the remainder of this section. RIB begins by finding the first *maxk* nearest neighbors of every instance i, where *maxk* is the maximum value of k being considered. (In our experiments we used *maxk* = 30 to leave a wide margin of error, and values of k greater than 10 were rarely if ever chosen by the system.) The nearest neighbors of each instance i, notated $i.n_1...i.n_{maxk}$, are stored in a list ordered from nearest to furthest for each instance, so that $i.n_1$ is the nearest neighbor of i and $i.n_k$ is the kth nearest neighbor. The distance $i.d_j$ to each of instance i's neighbors is also stored to avoid continuously recomputing this distance. This is the most computationally-intensive step of the process and takes $O(mn^2)$ time, where m is the number of input attributes and n is the number of instances in the training set.

CVC is used in the RIB system to evaluate values for the parameters k, w_k, *kernel*, and *avgk*. None of these parameters affect the distance between neighbors but only affect the amount of voting weight each neighbor gets. Thus, changes in these parameters can be made without requiring a new search for nearest neighbors or even an update to the stored distance to each neighbor. This allows a set of parameter values to be evaluated in $O(kn)$ time instead of the $O(mn^2)$ time required by a naive application of LCV.

To evaluate a set of parameter values, cvc_i as defined in Equation 3 is computed as follows. For each instance i, the voting weight for each of its k nearest neighbors is found according to its stored distance and the current settings of k, w_k, *kernel* and *avgk*, as described in Section 4.2. These weights are summed in their respective classes, and the confidence of the correct class is found as in Equation 1. If the majority class is the same as the true output class of instance i, cv in Equation 3 is 1. Otherwise, it is 0. The average value of cvc_i over all n instances is used to determine the fitness of the parameter values.

The search for parameter values proceeds in a greedy manner as follows. For each iteration, one of the four parameters is chosen for adjustment, with the restriction that no parameter can be chosen twice in a row, since doing so would simply rediscover the same parameter value. The chosen parameter is set to various values as explained below while the remaining parameters are held constant. For each setting of the chosen parameter, the CVC fitness for the system is calculated, and the value that achieves the highest fitness is chosen as the new value for the parameter.

At that point, another iteration begins, in which a different parameter is chosen at random and the process is repeated until 10 attempts at tuning parameters does not improve the best CVC fitness found so far. In practice, only a few iterations are required to find good settings, after which improvements cease and the search soon terminates. The set of parameters that yield the best CVC fitness found at any point during the search are used by RIB for classification. The four parameters are tuned as follows.

1. Choosing *k*. To pick a value of k, all values from 2 to *maxk* (=30 in our experiments) are tried, and the one that results in maximum CVC fitness is chosen. Using the value $k = 1$ would make all of the other parameters irrelevant, thus preventing the system from tuning them, so only values 2 through 30 are used until all iterations are complete.

2. Choosing a kernel function. Picking a vote-weighting kernel function proceeds in a similar manner. The kernel functions *linear*, *gaussian*, and *exponential* are tried, and the kernel that yields the highest CVC fitness is chosen. Using *majority* voting would make the parameters *wk* and *avgk* irrelevant, so this setting is not used until all iterations are complete. At that point, majority voting is tried with values of k from 1 to 30 to test both $k = 1$ and majority voting in general, to see if either can improve upon the tuned set of parameters.

3. Setting *avgk*. Selecting a value for the flag *avgk* consists of simply trying both settings, i.e., using D_k and D'_k and seeing which yields higher CVC fitness.

4. Searching for w_k. Finding a value for w_k is more complicated because it is a real-valued parameter. The search for a good value of w_k begins by dividing the range 0..1 into ten subdivisions and trying all eleven endpoints of these divisions. For example, on the first pass, the values 0, .1, .2, ..., .9, and 1.0 are used. The value that yields the highest CVC fitness is chosen, and the range is narrowed to cover just one division on either side of the chosen value, with the constraint that the range cannot go outside of the range 0..1. For example, if .3 is chosen in the first round, then the new range is from .2 to .4. The process is repeated three times, at which point the effect on classification becomes negligible.

Pseudo-code for the parameter-finding portion of the learning algorithm is shown in Figure 2. This routine assumes that the nearest *maxk* neighbors of each instance T have been found and returns the parameters that yield the highest CVC fitness found during the search. Once these parameters have been found, the neighbor lists can be discarded, and only the raw instances and best parameters need to be retained for use during subsequent classification.

In Figure 2, to "try" a parameter value means to set the parameter to that value, find the CVC fitness of the system, and, if the fitness is better than any seen so far, set *bestCVC* to this fitness, and remember the current set of parameter values in *bestParams*.

The time spent tuning parameters is done just once during learning, and is dominated by the first $O(mn^2)$ step required to find the nearest neighbors of each instance. During execution, classification takes $O(mn)$ time, which is the same as the basic nearest neighbor rule.

```
FindParams(maxTime, training set T): bestParams
  Assume that the maxk nearest neighbors have been
     found for each instance i in T.
  Let timeSinceImprovement=0.
  Let bestCVC=0.
  While timeSinceImprovement < maxTime
     Choose a random parameter p to adjust.
     If (p="k") try k=2..30, and set k to best value found.
     If (p="shape") try linear, gaussian, and exponential.
     If (p="avgk") try D_k and D'_k.
     If (p="w_k")
        Let min=0 and max=1
        For iteration=1 to 3
           Let width=(min-max)/10.
           Try w_k=min..max in steps of width.
           Let min=best w_k-width (if min<0, let min=0)
           Let max=best w_k+width (if max>1, let max=1)
        Endfor
     If bestCVC was improved during this iteration,
        then let timeSinceImprovement=0,
        and let bestParams=current parameter settings.
  Endwhile.
  Let shape=majority, and try k=1..30.
  if bestCVC was improved during this search,
     then let bestParams=current parameter settings.
  Return bestParams.
```

Figure 2. Learning algorithm for RIB.

5. Experimental Results

The Refined Instance-Based (RIB) learning algorithm was implemented and tested on 31 applications from the Machine Learning Database Repository at the University of California, Irvine [12]. RIB was compared to a static instance-based learning algorithm that is identical to RIB except that it uses $k = 3$ and majority voting and thus does not fine-tune parameters. RIB was also compared to an otherwise identical algorithm that uses leave-one-out cross-validation (LCV) instead of CVC to decide on the various parameters. (Experiments were also run using confidence alone to decide on parameters, but as expected, values were almost always chosen that favored nearer neighbors, i.e., $k = 1$, $w_k = 0$, and an exponential kernel. Results using confidence alone were thus worse than doing no parameter tuning at all, and are therefore not included here.)

For each dataset each algorithm was trained using 90% of the available data. The remaining 10% of the data was classified using the instances in T and the best parameter settings found during training. The average accuracy over 10 such trials (i.e., 10-fold cross-validation accuracy) is reported for each dataset in Table 1.

RIB (using CVC) had the highest generalization accuracy in 18 out of these 31 datasets, LCV was highest in 10 datasets and the static majority-voting algorithm was highest in 7 cases. RIB was an average of over 1% higher than the static algorithm in generalization accuracy on these datasets. LCV fell almost exactly halfway between the static and RIB methods. All of these algorithms have substantially higher generalization accuracy than the basic nearest neighbor rule using a Euclidean distance function [11].

In order to see if the average generalization accuracy for CVC was significantly higher than the others, a *Wilcoxon signed ranks* test [13] was used on the accuracy values listed in Table 1. As shown at the bottom of Table 1, CVC had a significantly higher average generalization accuracy on this set of classification tasks than both the static and LCV methods at a 99% confidence level or higher.

Dataset	Static	LCV	RIB
Anneal	93.11	94.49	**94.62**
Australian	84.78	85.08	**85.36**
Breast Cancer(WI)	96.28	**96.71**	96.42
Bridges	**66.09**	65.09	65.09
Crx	83.62	84.78	**85.07**
Echocardiogram	94.82	**96.07**	**96.07**
Flag	61.34	62.39	**63.37**
Glass	**73.83**	67.81	69.20
Heart	81.48	81.85	**83.34**
Heart(Cleveland)	81.19	81.48	**83.15**
Heart(Hungarian)	79.55	80.60	**80.93**
Heart(Long Beach)	70.00	**73.50**	73.00
Heart(More)	73.78	77.03	**78.52**
Heart(Swiss)	**92.69**	92.63	92.63
Hepatitis	80.62	**83.00**	81.79
Horse Colic	57.84	59.51	**65.15**
Image Segmentation	**93.10**	91.67	91.91
Ionosphere	84.62	**86.91**	86.62
Iris	94.00	**95.33**	**95.33**
LED Creator+17	67.10	71.80	**71.90**
LED Creator	**73.40**	72.30	72.90
Liver (Bupa)	**65.57**	61.77	61.41
Pima Diabetes	73.56	73.58	**75.26**
Promoters	93.45	**94.09**	93.09
Sonar	**87.55**	83.57	84.10
Soybean (Large)	88.59	90.54	**90.86**
Vehicle	71.76	**72.13**	71.54
Voting	95.64	**95.85**	**95.85**
Vowel	96.57	**98.29**	**98.29**
Wine	94.93	96.01	**97.71**
Zoo	94.44	94.44	**97.78**
Average	**82.11**	**82.59**	**83.17**
Wilcoxon	**99.50**	**99.00**	*n/a*

Table 1. Generalization accuracy of IBL algorithms using static majority voting (*Static*), cross-validation to make decisions (*LCV*), and CVC as used in RIB.

6. Conclusions

The RIB learning algorithm combines the use of cross-validation accuracy and confidence (CVC) to generate an evaluation function that returns real-valued differences in fitness in response to even small changes in parameters. It avoids the problem of frequent ties that occurs when using cross-validation alone. It also does not suffer from the strong bias towards heavily weighting nearer neighbors that occurs when using confidence alone.

In our experiments on a collection of 31 datasets, RIB was able to successfully use the new CVC evaluation method in conjunction with a distance-weighted voting scheme to improve average accuracy over a static majority-voting algorithm or a distance-weighted algorithm using only cross-validation to make decisions.

References

[1] Schaffer, Cullen. 1993. Selecting a Classification Method by Cross-Validation. Machine Learning.13-1.

[2] Moore, Andrew W., and Mary S. Lee. 1993. Efficient Algorithms for Minimizing Cross Validation Error. In Machine Learning: Proceedings of the Eleventh International Conference, Morgan Kaufmann.

[3] Kohavi, Ron. 1995. A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection, In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95).

[4] Aha, David W., Dennis Kibler, Marc K. Albert. 1991. Instance-Based Learning Algorithms. Machine Learning. 6, pp. 37-66.

[5] Cover, T. M., and P. E. Hart. 1967. Nearest Neighbor Pattern Classification, Institute of Electrical and Electronics Engineers Transactions on Information Theory. 13-1, January 1967. pp. 21-27.

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

[7] Dudani, Sahibsingh A. 1976. The Distance-Weighted k-Nearest-Neighbor Rule, IEEE Transactions on Systems, Man and Cybernetics. 6-4, 1976. pp. 325-327.

[8] Stanfill, C., and D. Waltz. 1986. Toward memory-based reasoning. Communications of the ACM. 29, 1986. pp. 1213-1228.

[9] Cost, Scott, and Steven Salzberg. 1993. A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features, Machine Learning. 10. pp. 57-78.

[10] Lebowitz, Michael. 1985. Categorizing Numeric Information for Generalization. Cognitive Science. 9. pp. 285-308.

[11] Wilson, D. Randall, and Tony R. Martinez. 1997. Improved Heterogeneous Distance Functions. Journal of Artificial Intelligence Research. 6-1. pp. 1-34.

[12] Merz, C. J., and P. M. Murphy. 1996. UCI Repository of Machine Learning Databases. Irvine, CA: University of California Irvine, Department of Information and Computer Science. http://www.ics.uci.edu/~mlearn/

[13] Conover, W. J. 1971. Practical Nonparametric Statistics. New York: John Wiley, pp. 206-209, 383.