# Stretch: A Feature Weighting Method for The k Nearest Neighbor Algorithms

by

Mehmet Ali Yatbaz

A Thesis Submitted to the
Graduate School of Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Electrical & Computer Engineering

Koç University

October, 2007

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Mehmet Ali Yatbaz

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

Assist. Prof. Deniz Yuret

Assist. Prof. Metin Türkay

Assist. Prof. Alper Tunga Erdoğan

Date:

*Aileme*

*To my parents*

# ABSTRACT

The k nearest neighbor learning algorithm (kNN) is one of the well studied non-parametric learning algorithms. kNN assumes that the underlying joint probability density function of the training set is unknown and it estimates the underlying joint probability density functions using the labeled data set (training set). Although this is a realistic assumption in terms of the real world problems, it introduces some limitations on the predictive accuracy, the storage complexity and computational compexity of the kNN.

The goal of this thesis is to understand kNN and techniques that are used to increase the predictive accuracy of kNN. This thesis mainly focuses on the effect of the irrelevant features on the predictive accuracy of the kNN and introduces the Stretch method, a new preprocessing method to increase the predictive accuracy of kNN by doing linear transformation on the training data matrix. The method incrementally constructs a linear transformation that maximizes the nearest neighbor classification accuracy on the training set. At each iteration the method picks an instance from the data set, and computes a transformation that moves the instance closer to the instances with the same category and/or away from the instances in other categories. The composition of these iterative linear transformations can lead to statistically significant improvements in kNN learning algorithms.

# ÖZETÇE

En yakın k komşu algoritması (EKK) uzun süreli çalışılmış parametresiz sınıflandırma algoritmalarındandır. EKK sınıflandırılmış örnek verilerin dağılımının altında yatan birleşik olasılık yoğunluk fonksiyonunun bilinmediğini kabul eder ve bu fonksiyonu sınıflandırılmış örnek verileri kullanarak ölçümler. Her ne kadar bu varsayım pratikte karşılaşılan problemler açısından gerçekçi bir yaklaşım olsa da EKK'nin sınıflandırma doğruluğu, veri depolama miktarı ve hesaplama zamanı üzerinde olumsuz etkilere sebep olur.

Bu tezin amacı EKK algoritmasının anlaşılması ve EKK'nin sınıflandırma doğruluğunun arttırılması için kullanılan yöntemlerin incelenmesidir. Bu tez esas olarak verilerin sahip olduğu ilgisiz özelliklerin EKK algoritmasının sınıflandırma doğruluğuna olan etkisi üzerine yoğunlaşmıs ve bu sorunu çözmek amaçlı Stretch adında yeni bir yöntem önermiştir. Bu yöntem sınıflandırma öncesinde örnek veriler üzerinde dogrusal dönüşumler uygulayarak EKK'nin sınıflandırma doğruluğunu arttırmayı amaçlar. Başka bir değişle, Stretch örnek verileri kullanarak EKK algoritmasının sınıflandırma doğruluğunu en yuksek büyüklüğe çıkartacak doğrusal dönüşümleri adım adım hesaplar. Bu yöntem her adımda örnek sınıflandırılmış veriler arasından bir veri seçer ve bu veriyi kendi ile aynı sınıftaki verilerle yakınlaştıracak ve/veya kendisi ile farklı sınıflardaki verilerle uzaklaştıracak olan doğrusal dönüşümü hesaplar. Farklı adımlarda oluşturulmuş bu doğrusal dönuşümlerin bileşimi olan sonuç doğrusal dönüşümü EKK algoritmasının sınıflandırma doğruluğu üzerinde istatiksel olarak kayda değer bir artış gösterir.

# ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor Assist. Prof.Deniz Yuret for his valuable guidance, understanding and patience throughout my graduate study. Also, I would like to thank Assist. Prof. Alper Tunga Erdoğan and Assist. Prof. Metin Türkay for their valuable comments as members of my thesis committee.

I am grateful to my parents, Leyla and Naci Emin for their endless support, patience and love during my entire life. I would like to thank my brothers Hakan İşbilir, Mehmet Özgüçlü and Mustafa Karaman who have shared everything with.

I was fortunate to have such great friends close to me during my graduate education. Ergun Biçici, Ahmet Engin Ural and Mehmet Akgul were great office mates. It was a pleasure to share the same house with Muhittin Emre Özdemir, Mehmet Ali Dündar and Kadir Onur Unutulmaz. And also I would like to thank my friends and colleagues; Emra Atsan, Burak Görkemli (Müdür) ,Erkan Keremoğlu, Emre Güney, Tuğba Özbilgin, Çağdaş Atici, Volkan Dedeoğlu and Bengi Mizrahi.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# Chapter 1

# INTRODUCTION

The importance of Machine Learning (MacLer) is increasing day by day as the information on science, engineering disciplines and business domains becomes larger and larger. Therefore the real world applications of MacLer have started to emerge in industrial, educational and business areas.

The algorithms of the MacLer can be grouped into two categories, which are

- Supervised Learning
- Unsupervised Learning

In supervised learning, the data consists of the features that represent the properties of the instances and a desired output which is also called label. On the other hand, in unsupervised learning the data only consists of the features.

The aim of the supervised learning is to classify the new unlabeled data by learning from the labeled training data. There are plenty of learning algorithms that are using different techniques to estimate the underlying pattern of the training data.

The k-nearest neighbor(kNN) classifier [2] is a well studied example of the MacLer algorithms that construct local models using the labeled data. kNN postpones the model building task until a new instance is introduced. When a new instance is introduced, kNN finds the k nearest neighbors of this new instance and determines the label of the new instance by using these k instances. Therefore, the family of kNN algorithms are also referred as lazy learning algorithms. There are also other learning algorithms that build the learning model during the training phase. These algorithms have a longer training time compared to lazy learning algorithms and these algorithms are refered as eager learning algorithms.

## *1.1 Motivation*

Since a new instance is classified based on the k nearest instances, the quality of the kNN generalization depends on the distances between the instances. The existence of irrelevant features decrease the generalization accuracy of kNN since they affect the distances between the instances.

Feature weighting methods construct a diagonal feature weighting matrix, therefore they are unable to identify the corellation of the features.

The motivation of this thesis is to develop a preprocessing method, *Stretch*, that constructs a linear transformation to increase the generalization accuracy of kNN. This linear transformation is nothing but a full weighting matrix that decreases the importance of the irrelevant features while increasing the importance of the relevant features.

## *1.2 Contributions*

During the course of the studies leading up to this thesis, the main research work has been on the development of a preprocessing method that increases the classification accuracy of kNN. Furthermore, other preprocessing methods that construct diagonal weighting matrices together with kNN algorithms have been studied.

The contributions can be listed as:

- Construction of a full weight matrix, therefore theoretically our approach can construct any scale matrix while other methods can only construct diagonal matrix,
- Improving the leave-one-out cross validation(LOOCV) error of kNN on the training set,
- Improving the accuracy of 1NN on the test sets,
- Improving the accuracy of kNN on the noise-free(less noisy) data sets.

## *1.3 Related Work*

The focus of the thesis has been on feature weighting methods, where each instance of the data set is multiplied by a diagonal matrix. Various approaches have been introduced to calculate feature weights: Relief[3] and Relief-F[4] uses incrementaly updates the weights similar to *Stretch* algorithm. The main difference between *Stretch* and Relief-F is, *Stretch* can construct full matrix while Relief-F construcs a diagonal weight matrix. Therefore, *Stretch* can build any arbitrary matrices including the diagonal matrices. VSM[5] uses conjugate gradient methods, PCF and CCF[6] uses a probabilistic model. VDM[7] uses different weight vectors for different parts of the instance space.

## *1.4 Outline*

The thesis is organized as follows: Chapter 2 surveys the literature and defines some basic concepts of the feature weighting/selection methods. Chapter 3 forms the basis of the thesis. It describes the details of the *Stretch* method and presents motivating examples to clarify the idea of *Stretch* in detail. In Chapter 4, the results of the experiments are presented and the performance of *Stretch* on nineteen data set is discussed by comparing them with the results of the Relief-F [8] method. Finally, Chapter 7 concludes the thesis pointing out the achievements obtained in the thesis and discusses possible future work.

Chapter 2

# NEAREST NEIGHBOR ALGORITHMS AND FEATURE WEIGHTING

## *2.1 Nearest Neighbor Algorithms*

k Nearest Neighbor (kNN) algorithm is one of the well known non-parametric methods. Many supervised learning algorithms that are parametric (like maximum likelihood) assume that the underlying density functions of the data set are known, which is not a realistic assumption in real world problems. In other words, most of the time real world problems can not be well represented by known parametric forms. On the other hand, non-parametric methods (like kNN, Parzen Window) relax the learnining by assuming the underlying density functions are unknown and estimate the underlying densities from the data set. However, this relaxation, may increase the error of the learning task as explained in the section 2.2.1.

The main idea of the kNN is very similar to the Parzen window technique [9]. The Parzen window technique defines a fixed size window and estimates the local density function by using the instances that fall into this window. The main difference of kNN with this method is, instead of keeping the size of the window fixed, the size is determined by $k^{th}$ nearest neighbor. Therefore, the window size is nothing but a function of sample (training) data set.

kNN uses the estimated local density function during the calculation of the posterior probabilities that are used for classification.

## 2.2 Theoretical Framework

The theoretical framework of the kNN requires special attention, since the goal of the *Stretch* algorithm is to minimize the classification error of the kNN that is represented by $E_{NN}$. $E_{NN}$ is equal to the number of misclssified instances in a set divided by the number of instances in the set. The set of misclassified instances that is constructed by kNN has a crucial role on the performance of *Stretch*. The kNN algorithm is divided into components and each component explained in detail in the rest of the chapter.

### *2.2.1 Error Bounds of NN*

The Bayes method classifies the unlabeled data using the probability density function multiplied by a priori probability of each label. Therefore, the minimum error can only be achieved when the underlying koint density functions of labels are known. Parametric methods, like Bayes analysis, assume priori knowledge of the underlying joint distribution, so the error $E^*$ of these methods is optimal. However, as a member of non-parametric statistics family, kNN does not have a priori knowledge of the underlying joint distribution of the sample points. Therefore, the error of kNN, $E_{NN}$, must be at least as great as $E^*$ which is a tight lower bound for any learning algorithm.

Calculation of the upper bound of $E_{NN}$ is not as trivial as the lower bound case and was not clear until [10]. A tight upper bound can be defined as the sample set goes to infinity. Although it is not possible to have infinite amount of sample data in practice, it gives an intuition about the upper bound of the error as the sample data size becomes larger.

As a result, [10] proved that the bounds of error $E_{NN}$ on a data set with $|C|$ categories as the data set size goes to infinity can be shown to be

$$E^* \leq E_{NN} \leq E^* \left(2 - \frac{|C|E^*}{|C|-1}\right) \tag{2.1}$$

### 2.2.2 Convergence of kNN

The work of Wagner [11] enhances the contribution of [10] by defining the convergence of the nearest neighbor rule. It is shown that for an $n$ dimensional data set the error of the nearest neighbor rule converges to $E_{NN}$ that is defined in equation 2.1 with probability of 1 for mild continuity and moment assumptions.

$$E_{NN} \rightarrow E \text{ with probability 1.} \tag{2.2}$$

### 2.2.3 Decision Function of kNN

Most of the non-parametric methods assume that the probability density function under consideration is a locally constant function. An example that uses this assumption is kernel estimation where each local instance has a contribution on the local estimate, $\hat{p}(x)$. The posterior probabilities of the kNN can be defined by using the kernel function. The kernel is bounded function that integrates to one. Since the kernel is a weighted function, the contribution of each instance highly depends on the kernel function, $K(x)$.

A kernel estimator for $n$ sample instances can be defined as

$$\hat{p}(x) = \frac{1}{n} \sum_{i}^{n} K(x - x_i) \tag{2.3}$$

Kernel estimation can be defined locally for the category $i$ in $C$,

$$\hat{p_i}(x) = \frac{1}{n_i} \sum_{j=1}^{n_i} K(x - x_{f(j)}) \tag{2.4}$$

where $n_i$ is number of sample instances from the category $i$, $f(j)$ is a function of $j$ and $p_i$ is the estimated density for category $i$. Using the $\hat{p}(x)$, an estimated posterior probability can be found using bayes rule

$$\hat{p}(c_i|x) = \frac{\pi_i \hat{p_i}(x)}{\sum_i^{|C|} \pi_i \hat{p_i}(x)} \tag{2.5}$$

where $\pi_i$ is the prior probability of the $i^{th}$ category. If $\hat{p_i}(x)$ is substituted with the right hand side of equation 2.4 then equation 2.5 becomes

$$\hat{p}(c_i|x) = \frac{\frac{\pi_i}{n_i} \sum_i^k K(x - x_i)}{\sum_j^k \frac{\pi_i}{n_i} K(x - x_j)} \tag{2.6}$$

As it is showed in [12] equation 2.6 simplifies to

$$\hat{p}(c|x) = \frac{\sum_n I(c, c_i) K(x - c_i)}{\sum_i^k K(x - x_i)} \tag{2.7}$$

where $c_i$ stands for the category of the $i^{th}$ instance and $I$ is an indicator function that is defined as

$$I(a, b) = \begin{cases} 0 \, if \; a \neq b, \\ 1 \, if \; a = b. \end{cases} \tag{2.8}$$

The accuracy of the estimation is highly correlated with the number of sample instances and the definition of the kernel function. The posterior probabilities of kNN can be defined in terms of the density estimations that are calculated using the kernel with $k$ local sample instances and a constant kernel function. As a result, the probability of $x$ being from the category $c_i$ is

$$\hat{p}(c_i|x) = \frac{k_i}{k} \tag{2.9}$$

where $k_i$ is the number of instances that are members of the categor $c_i$. Different types of kernel functions for kNN are discussed in the subsection 2.2.5.

### *2.2.4 Distance Function of kNN*

Distance function defines the relationship between two instances in terms of the feature set $F$. A generalized version of distance function can be defined as

$$d(x,y) = \sqrt[r]{\sum_{i=1}^{|F|} w(f_i)(\delta(x_i,y_i))^r} \tag{2.10}$$

where $\delta(x,y)$ and $w(f)$ is a proximity and feature weighting function defined on $f_i \epsilon F$, respectively. For a basic kNN algorithm,

$$\begin{aligned} r &= 2, \\ \delta(x_i,y_i) &= x_i - y_i, \\ w(f) &= 1. \end{aligned}$$

### *2.2.5 Distance Weighting Function*

Distance weighting function determines the effect of each nearest neighbor during the estimation of posterior probabilities when $k > 1$. This function defined in equation 2.7 as $K(x - x_i)$, is called the kernel function of kNN.

- Majority voting

  If $K(x - x_i) = c$ where $c$ is a constant, then each nearest instance has the same effect on the decision function. As a result, the posterior probability is given by equation 2.9. It is clear that, in majority voting there are more ties compared to the non-constant kernels and these ties are sometimes responsible for the high value of error when they are broken randomly.

- Inverse Distance Weighting

  If $K(d(x,x_i)) = \frac{1}{d(x,x_i)}$, then the evidence of the closer neighbor will be weighted more heavily compared to further neighbor. However, this kernel function produces unexpected results when the nearest instance are sparse. Moreover when

the data set is not normalized the $K(d(x, x_i))$ approaches to zero for the further nearest instances.

- A Generalization of Dudani's Weighting Function

  This kernel is a generalized version of the kernel proposed in [13] and published in [14]. It defines the voting weight of $i^{th}$ nearest neighbor, $v_i$, as

$$v_i = \begin{cases} \frac{(d_s - d_j) + \alpha(d_s - d_1)}{(1+\alpha)(d_s - d_1)} \text{ if } d_s \neq d_1 \\ 1 \text{ if } d_s = d_1 \end{cases} \tag{2.11}$$

  where $d_s$ is the distance between the query instance and its $s^{th}$ nearest neighbor (s = k,k+1,...) and $\alpha > 0$. This generalized version of [13] takes $k^{th}$ nearest neighbor into account on decision process unlike the original work proposed by [13] which is a special case of this generalized version where $\alpha = 0$ and $d_s = d_k$.

  Since the kernel scales the distances between query point and its nearest neighbors, each instance has a comparable effect on the decision procedure.

One can define non-linear distance weighting functions like Radial Based Kernels [15] and Probabilistic Neural Networks [16] that fits a Gaussian distribution that assigns higher weights to closer instances and lower to further ones.

### *2.3 Computational Complexity of kNN*

As stated in section 2.4.2, the computational complexity is one of the biggest drawbacks of kNN algorithms compared to the eager learning algorithms. The complexity of the basic kNN on a data set with $m$ instances is $O(m^2)$ since it is not using any data structures. However, if the kNN algorithm uses a data structure such as vp-tree [17] then finding $k$ nearest neighbors of all the instances in the data set on average case can be accomplished in $O(m \log m)$ time.

## *2.4 Liminations of the kNN*

As a member of lazy learning algorithm family, kNN has some limitations in real life problems [1]. These limitations are,

1. High storage complexity,

2. High computational complexity,

3. Sensitivity to selection of similarity function and value of $k$,

4. Sensitivity to noisy and irrelevant attributes,

5. Provides little information regarding the structure of categories.

Each of these problems is a research area by itself and therefore needs special attention that is beyond of the scope of this thesis. This thesis focuses on the solutions that reduce effect of the irrelevant features on the classification accuracy of kNN and suggest a method that significantly increases the classification accuracy of kNN on the data sets that have irrelevant and correlated features. All of the above limitations are explained briefly in the rest of this chapter.

### *2.4.1 Storage Complexity*

As it is mentioned in Chapter 1, standard kNN keeps the whole dataset in memory and does not preprocess it prior to any classification of new instances. As a result, the storage complexity of the algorithm increases as the dataset size increases. An increase in the dataset size causes an increase in the memory requirement of the learning algorithm.

The complexity of storage is directly related to the number of instances and attributes so any reduction attempt on these decreases the memory requirement. In order to reduce the memory requirement different reduction techniques are proposed [18]. These reduction techniques select a smaller subset of the dataset and use this

subset during the classification of new instances. There are also some techniques that create new instances by using weighted averages of similar instances. As a result the initial size of the dataset becomes smaller.

Selecting some subset of the dataset may decrease the generalization accuracy of the classification. A successful storage reduction algorithm does not decrease the classification accuracy significantly while reducing the storage size significantly [18].

The storage requirement can also be reduced by selecting a smaller subset of the feature set which is discussed in section 2.5.

Storage reduction techniques have two positive side effects [18]:

1. Run time of the algorithm decreases,

2. The number of noisy instances decreases.

### *2.4.2 Computational Complexity*

kNN does not pre-process the dataset therefore compared to eager algorithms it has less computational cost during training phase. However, during the classification of a new instance kNN has a higher computational requirement. That is mainly because it calculates the similarity between the new instance and each instance in the dataset. Therefore, it can be concluded that as the number of instances increases computational complexity also increases.

Although computational complexity is mainly due to the number of instances, computational complexity can be reduced without reducing the number of instances by using indexed tree search algorithms such as k-d tree [19], projection [20], ANN [21], vp-tree [17] and r-tree [22]. However, as the number of attributes increases instances become very close to each other. Therefore, the number of branches in the indexed tree algorithm increases which increases the number of comparisons. As a result the tree loses its efficiency [19].

### *2.4.3 Sensitivity to selection of similarity function and value of k*

The similarity function calculates the distance between instances therefore it has a crucial role in determining the nearest neighbor set of an instance. Similarity functions are discussed in detail in section 2.2.

The variable $k$ can get different values from 1 to dataset size, $n$. The value of $k$ has an important role on the performance of kNN. Selecting larger $k$ values not only yields smoother decision function but also provides more probabilistic information about the nearest neighbors of the instances. Moreover, larger $k$ values increase the robustness of the algorithm to noise. On the other hand, as $k$ goes to $n$, the locality of the classification is damaged because farther instances are taken into account. There is also instance weighting by distance which is discussed in section 2.2. In addition, the computational complexity of the kNN increases as $k$ becomes larger.

### *2.4.4 Noisy Instances*

Noisy instances are the instances that are randomly assigned into some category so their information content is misleading. In order to reduce the effect of the noisy instances during the decision process, some extra work must be done. There are two methods to cope with noisy instances:

1. Using (k>1)NN algorithms.

2. Selecting non-noisy training instances and using them during the decision process.

The first method assumes that if $x_{noise}$ is a noisy instance then the probability that its nearest neighbor is from some other category will be high. As a result, any instance $x \in X$ that is nearest neighbor with $x_{noise}$ is misclassified by 1NN. While $k > 1$, $k$ neighbors are taken into account during the classification therefore the effect of the noisy instance are reduced.

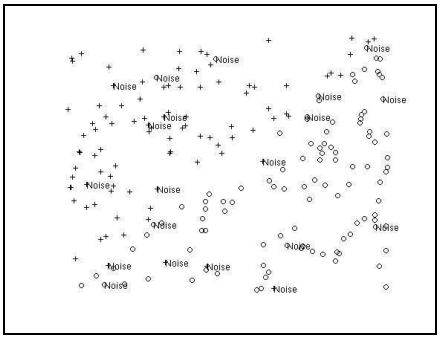


Figure 2.1: The figure is 2D data set with 200 instances and 2 categories showed with plus and star. 20 of the instances are randomly generated and assigned to a category. These instances are labeled as 'Noise' on the figure.

The second method is to select none-noisy instances by storage reduction techniques which are discussed in the section 2.4.1. These techniques do not only reduce the storage required but also remove the noisy instances by selecting the instances that are better in classification.

### *2.4.5 Irrelevant Attributes*

Irrelevant attributes are the ones that have misleading or little information about the categories. According to [23], features can be grouped into three disjoint categories. The relevance of a feature $F_i \epsilon F$ where $F$ is the set of all features, and $C$ is the set of all categories, can be defined as

- Strongly relevant

$$P(C|F) \neq P(C|F - F_i) \tag{2.12}$$

- Weakly relevant Let $F' \subset F - F_i$ then $F_i$ is weakly relevant if and only if

$$\exists F', \text{ such that } P(C|F_i, F') \neq P(C|F'). \tag{2.13}$$

- Irrelevant

$$\forall F' P(C|F_i, F') = P(C|F') \tag{2.14}$$

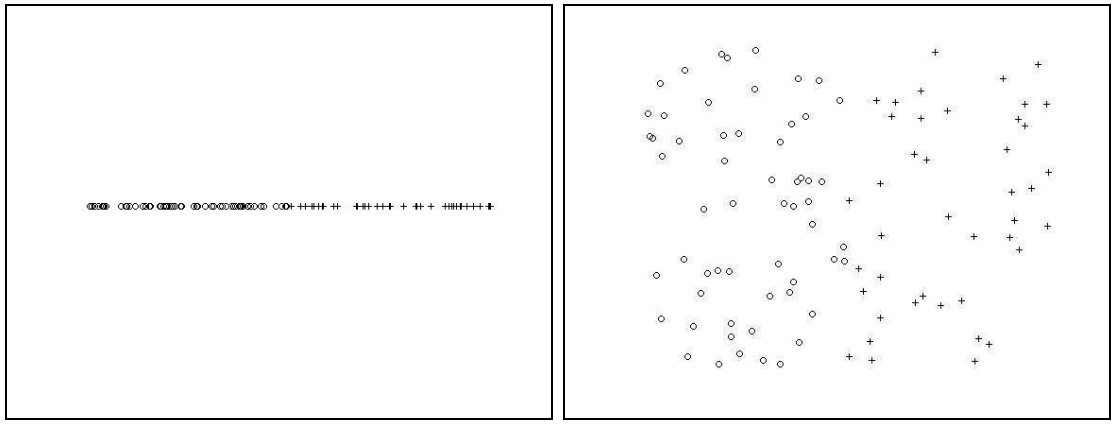

Figure 2.2: The figure on the left is 1D data set with 100 instances and 2 categories showed with plus and star. The one on the right is after adding irrelevant dimension that is generated randomly. The original dimension is enough to determine the categories of the instances.

Irrelevant attributes need special attention during the decision process. Normalizing all the features to [0,1] at the beginning of the learning task equalizes the effect interval of each feature on the decision function. However, as the number of irrelevant attributes increases, they become dominant on the decision functions. This situation is known as the curse of dimensionality [24]. Since kNN is using local estimates as a decision function, irrelevant attributes reduce the performance of the kNN. One solution to this problem is to construct a weight function, $w(f)$, such that it assigns low weights to irrelevant and weakly relevant features and high weights to the strongly relevant features. Another solution is to select the relevant features while discarding the irrelevant ones. This method is a special case of the previous solution, since it uses a weight function that assigns zero weight to irrelevant features.

Another solution to this problem is to stretch(transform) the data set in order to decrease the effect of the irrelevant features. A method that is using this approach is introduced in 3.1.

### *2.4.6 kNN Provides little information regarding the structure of categories*

Eager algorithms like decision trees [25], neural networks[26] or support vector machines [27] pre-process the dataset and gather information about the dataset prior to classification of the new instances. Therefore, by using the information gathered one can classify a new instance and one also can talk about the structure of the dataset. As it is mentioned in the computational complexity section 2.4.2, basic kNN algorithms do not pre-process the input data so they do not gather information about structures of categories at the beginning of learning task. However, there are some non-trivial kNN algorithms like RISE [28] that generalize instances by constructing rules in order to reduce the storage and thereby provide information about the structure.

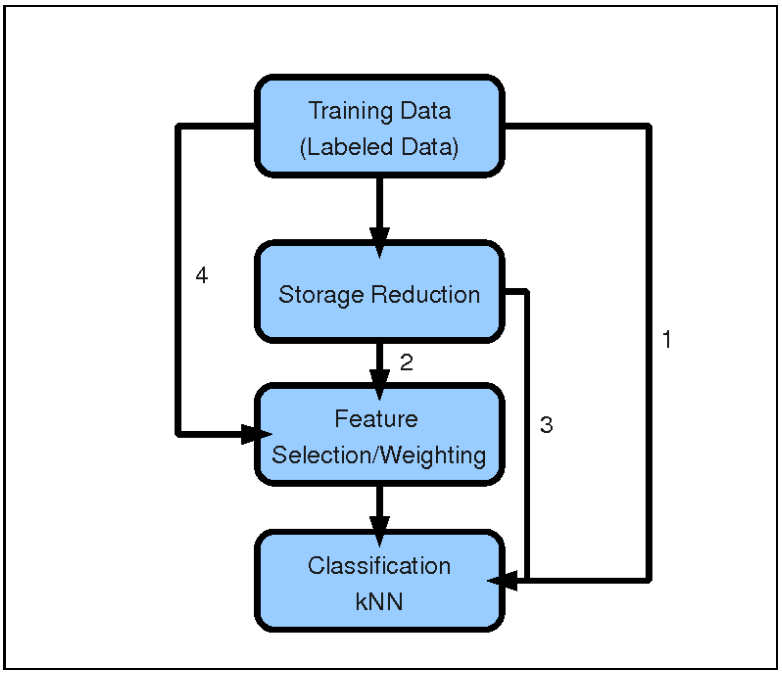


Figure 2.3: The figure is the flow chart of a classification procedure of kNN . The first path (labeled with 1) is an ordinary kNN classification procedure without using pre-processing methods. The second path is a kNN algorithm that uses storage reduction and feature selection/weighting methods before the classification. The third path is a kNN algorithm that only uses storage reduction whereas the fourth path only uses the feature selection/weighting before the classification.

## *2.5 Feature Weighting/Selection*

The quality of a feature is determined by how well it increases the prediction accuracy of the estimated function. A good feature should have information about the categories of the data set by itself or together with other features [29]. As stated at section 2.4.5, the feature weighting/selecting algorithms has a crucial role on the classification accuracy of the learning algorithm. There are various approaches to the feature weighting problem. Feature weighting methods can be categorized in several different manners with respect to their properties [30]. In this chapter, feature weighting methods are organized according to

- The type of weight space

  The type of weight space determines whether the method is a feature selection or a feature weighting method and both of these methods are discussed in section 2.6.

- The Feature Selection/Weighting Model

  Feature weighting methods can be divided into two categories based on usage of a learning algorithm as a feedback mechanism during the calculation of feature weights or selection of features [23]. These two categories are the filter model and the wrapper model which are discussed in section 2.7.1 and 2.7.2, respectively.

## *2.6 Weight Space*

Weight space determines the range of the feature weighting function which is defined in equation 2.10. As the range set size gets larger, the possible values that a weight can be assigned increase. In another words, if the range set size equals to one then all the feature weights are equal thus; the irrelevant features can not be distinguished from the relevant ones. There are three types of feature space and each of them are described in detail.

### *2.6.1 Binary Weight Space*

Feature weighting methods that use binary weight space are called feature selection methods and they set the weight to zero if the feature is irrelevant or to one if the feature is relevant. These methods select the relevant features and discard the irrelevant ones. As a result, most of the time, they increase both the classification accuracy and the computational performance of the learning algorithms. However, since the feature selection methods select the relevant features, they are unable to differentiate the relevant and weakly relevant features from each other [23]. According to [31] a feature selection method has four design issues.

- The starting point in the feature set space

  The starting point of the search determines the direction of the search. The method may start with an empty feature set then it adds each relevant feature one by one or it may start with the set of all features then it removes irrelevant features one by one. The first approach is called forward selection and the second one is called backward selection. There are also some hybrid methods which start with a randomly selected subset of feature space and perform a backward or forward search [32].

- The organization of the search

  Since the selection of a minimal subset of the feature set is NP-Complete [23], it is not possible to find the optimum subset of the feature space in a reasonable time frame (i.e., $2^f$ subsets where f is the dimension size of the feature space). Other search strategies such as sequential or randomized search are computationally more appropriate for the feature selection task [33]. Sequential algorithms use stepwise selection or elimination of the features from the feature set whereas the randomized algorithms use hill climbing or simulated annealing to construct their feature sets.

- The evaluation of the feature subsets

These strategies are discussed in section 2.7.1 and 2.7.2.

- The halting criteria

  This criteria determines the stopping point of the method.

Each of these design issues needs special attention since each has an effect on the construction of the feature subset. The feature selection algorithms are beyond the scope of this thesis; therefore, the rest of the subject is not discussed in detail. More detailed work can be found in [29, 34, 31, 32]

### *2.6.2 Nominal Weight Space*

The methods that use this weighting space assign weights from a pre-defined finite set of discrete values. The advantage of this method over the continuous weight space is the chance of over-fitting and the variance of weights is reduced. Since the values of weights are assigned from a pre-defined finite set, the number of different valued weights is less compared to the continuous case. On the other hand, since it uses finite set of weight values, the representative power of the weights is reduced [35].

### *2.6.3 Continuous Weight Space*

Feature weighting methods use a continuous weight space therefore they assign higher weights to relevant features while assigning lower weights to weakly relevant and irrelevant features. Therefore, they are more sensitive to weakly relevant features. On the other hand, since they do not eliminate any features, they are unable to reduce the storage complexity of the data set. However, they can be converted to a feature selection method by setting a threshold value for weights. If the computed weight is smaller than threshold value then the corresponding feature is discarded otherwise it is accepted.

## *2.7 The Feature Selection/Weighting Model*

The feature weighting/selection algorithms are categorized into two, based on the usage of a learning algorithm during the feature selection or weighting. Each model has its own advantages and disadvantages in terms of the computational cost and the effect on the classification accuracy of the learning algorithm.

### *2.7.1 The Filter Model*

The weighting methods that use filter model does not get any feedback from the learning algorithm during the calculation of weights or selection of features. In the filter model, the bias of the learning algorithm does not affect the selection/weighting of the features or vice versa. As a result the performance gain or loss of any selection/weighting action is not known during the weigh calculation or feature selection [23]. On the other hand, the advantage of not using a learning algorithm as a feedback mechanism is, the reduction of the computational cost compared to the wrapper methods. Instead of using the feedback from a learning algorithm these methods use other methods such as conditional probabilities, class projection, and information theory as a model [30]. There are also some methods that use a learning algorithm only once to get the necessary information about the instances and features. However, these algorithms do not get feedback as they update the weights.

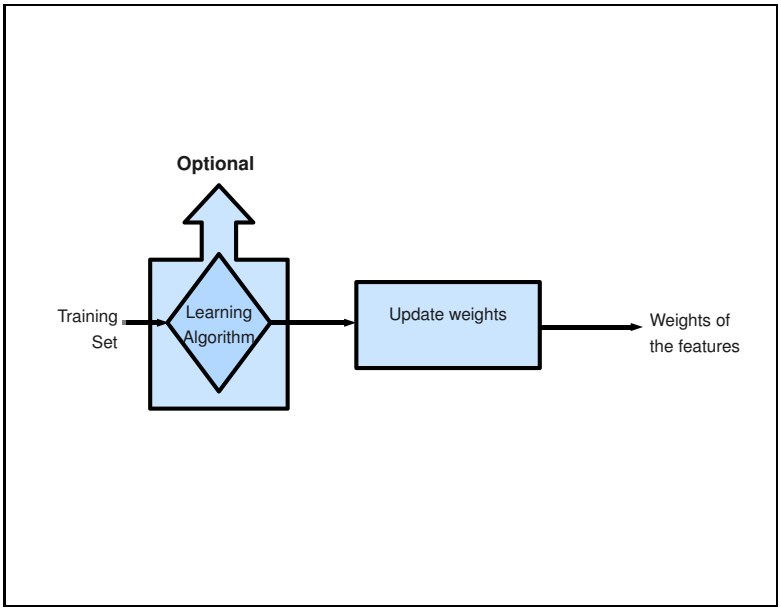


Figure 2.4: The figure is the flow chart of the filter model. The methods that use the filter model do not get any feedback from the learning algorithm however they may use a learning algorithm before the feature selection or weighting task.

Creecy et al. [6] used a conditional probability model to assign weights to features according to their ability to classify instances. They developed two methods both of which binarize the features before calculating the weights. The first method they proposed, cross-category feature importance (CCF), assigns higher weights to the features that are observed in fewer different categories. For example, if the occurrence of a feature is equally likely for each category then the ability of the feature to distinguish the categories is low, therefore CCF assigns a low weight to that feature. A feature gets the maximum weight (i.e., 1) if and only if it is observed only in one of the categories. The weights are calculated by using

$$w(f) = \sum_{i=1}^{|C|} P(c_i|f)^2 \tag{2.15}$$

where $f$ is the feature that is weighted and $C$ is the set of all categories (i.e., $C$ is the set of $c_i$'s where $c_i$ represents the $i^{th}$ category). However, this method assumes that the weights of features are independent of the categories. As a result, although

CCF assigns higher weights to the features that are observed in fewer categories, it provides no information about the relation of the feature with these few categories that the feature is observed.

The second method removes the independence assumption and calculates the weights for every feature in every category. Therefore, the weight of every feature is different for the given category. This method is named per category feature importance (PCF), and it calculates the distance between a new instance $q$ and the instance from the training set $x$ by using

$$distance(x, q) = \sum_{i=1}^{|F|} \delta(i, x_i, q_i) \tag{2.16}$$

$$\text{where } \delta(f, x_f, q_f) = \sqrt{w(f, c_x)(x_f - q_f)} \tag{2.17}$$

$$w(f, c_i) = P(c_i|f) \tag{2.18}$$

where $F$ is the set of all features, $c_x$ and $c_q$ represents the category of $x$ and the category of $q$. The above conditional probability calculates the weight of a feature, $f$, for a given category $c_i$. Since the feature values are binary, there is no need to consider the other values of $f$. The weakness of PCF is, if one of the categories is significantly larger than the other categories then the feature weights are highly determined by the majority class therefore PCF is highly sensitive to the distribution of the categories [36].

They perform better than the non-weighted algorithms. However, [6] did not compare these two methods with each other. Mohri and Tanaka [36] showed that CCF outperforms PCF on six out of eight classification tasks.

Stanfill and Waltz [7] defined a more complex similarity function based on the class projection model. Traditional similarity functions of nominal values calculates the difference by binarizing them or calculating the number of mismatching features. However, these similarity functions assume all of these nominal values have equal importance and the difference between every pair of nominal values is same. Stanfill and Waltz [7] described the value-difference metric (VDM) and used the distribution

of the nominal feature values while calculating the difference between two instances. This difference metric is defined between two instances(i.e. $x$ and $q$) of the training set and formalized as,

$$distance(x, q) = \sum_{i=1}^{|F|} w(i, x_i)\delta(i, x_i, q_i) \tag{2.19}$$

$$\text{where } w(i, v) = \sqrt{\sum_{i=1}^{|C|} p(c_i|x_i = v)^2} \tag{2.20}$$

$$\delta(i, v, u) = \sum_{j=1}^{|C|} (p(c_j|x_i = v) - p(c_j|x_i = u))^2 \tag{2.21}$$

where $F$ is the set of all features, $u$ and $v$ are the possible values of the given discrete feature. The feature weights are defined based on how well the feature value of some particular instance predicts the instance category. Therefore, the weights are averaged over the all categories based on the conditional probability defined in equation 2.20. VDM calculates distances for every pair of values of a nominal feature. As a result, two instances are close to each other for some particular feature if their values for that feature is distributed similarly over the categories. If the distribution of the feature value over the categories is uniform then VDM assigns a relatively low weigh to that feature. Stanfill and Waltz [7] did not compare VDM with other methods, however [37] and [38] outperforms the VDM in several classification tasks.

Daelemans and Van den Bosch [38] extends the idea of [39] and introduces a new method that runs on nominal valued data sets. Like [39], [38] uses information gain[25] to define the weights of each feature. They calculate the initial entropy of the data set X, by using

$$Entropy(X) = -\sum_{i=1}^{|C|} p(c_i) log_2 p(c_i) \tag{2.22}$$

where $p(c_i)$ is the prior probability of each category. The entropy of $X$ given the value of that feature is defined as,

$$Entropy(X|f) = -\sum_{v_i \epsilon V_f} \sum_{i=1}^{|C|} -p(c_i|f = v_i) log_2 p(c_i|f = v_i) \tag{2.23}$$

where $V_f$ is the set of possible values of feature $f$. The information gain is calculated by subtracting the equation 2.23 from 2.22.

$$Gain(f, X) = Entropy(X) - Entropy(X|f) \tag{2.24}$$

This method reduces the classification error of kNN algorithms of on a word hyphenation task [38] and a grapheme-to-phoneme conversion task [40]. One drawback of this method is, it does not provide a solution for continuous features. However, [30] discritizes the continuous features by dividing the continuous space into predefined buckets and assigning same discrete value to the continuous values that fall into same buckets.

Kira and Rendell [3] introduced a new algorithm, *Relief*, that calculates the weight based on the k nearest neighbors of each instance and selects the features that have high weights. Although this method uses a kNN algorithm to calculate weights it does not use it as a feedback mechanism, thus we present it in this section. kNN is run once to construct the k nearest neighbor set of each instance. *Relief* updates the weights of the features by using

$$w(f) = w(f) - \delta(x_f, H_f)/m + \delta(x_f, M_f)/m \tag{2.25}$$

where $x_f$, $H_f$ and $M_f$ are the feature value of f for the instance $x$, the nearest neighbor of $x$ that is from the same category as $x$, and the nearest neighbor of $x$ that is from a different category, respectively. The difference function, $\delta(x, y)$, that calculates the distance between $x$ and $y$. The contribution of distances to weights are normalized by the size of data set such that the weights are between [-1,1]. *Relief* assumes that the value of a feature is closer for two instances from the same category and is different for

two instances from different categories therefore features that vary across categories are assigned higher weights. Although *Relief* is a feature selection method it can be converted to a feature weighting method by removing the selection of high weighted feature part of the algorithm.

Kononenko [4] modified *Relief* to cope with noisy, incomplete and multi-category data sets. [4] defined a new weight formula as,

$$w(f) = w(f) - \delta(x_f, H_f)/m + \sum_{i=1}^{|C|} p(c_i)\delta(x_f, M_f(c_i))/m \tag{2.26}$$

where $M_f(c_i)$ is the value of the feature that is nearest neighbor of $x$ from the the category $c_i$ and $m$ is the number of the instances in the data set. This new modified *Relief*, *Relief-F*, performs well on selecting/weighting irrelevant features however *Relief* and its extension *Relief-F* can not recognize redundant or highly interacting features [1]. *Relief-F* is one of the best methods that increase the classification accuracy of kNN [1].

### *2.7.2 The Wrapper Model*

The methods that are designed based on the wrapper model get feedback from a learning algorithm during the calculation of the feature weights or the selection of features. The wrapper methods run the learning algorithm to learn the effect of each change on the feature weights or in the selected features set. These methods may perform a better heuristic search since they are informed by a learning algorithm during the feature weighting/selection. These methods halt when any change on the feature weights or selected feature set does not increase the classification performance of the learning algorithm.

On the other hand the usage of a learning algorithm increases the computational cost of these methods and also increases the risk of over-fitting on small training data sets [41]. The importance of a feature is different across the different learning algorithms. Therefore, the learning algorithm that is used in the wrapper method

decides the importance of the each feature [29] since the bias of the learning algorithm affects the direction of the search.

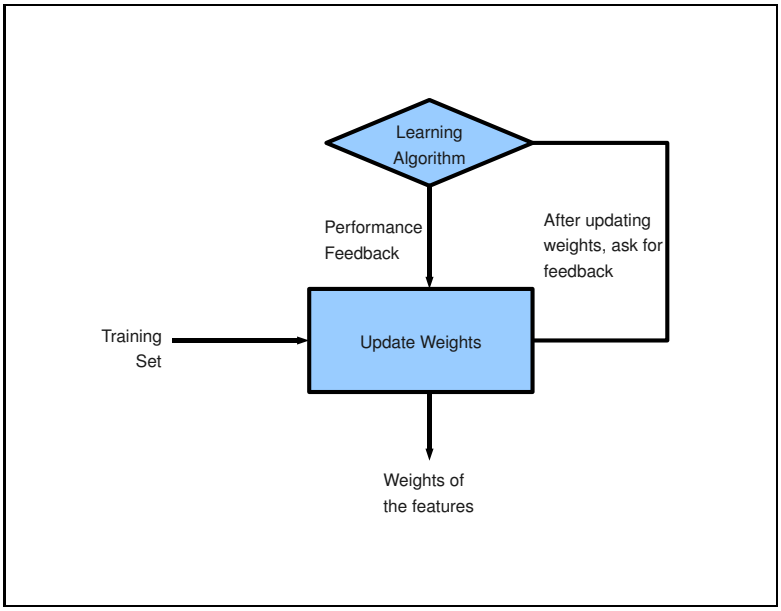


Figure 2.5: The figure is the flow chart of the wrapper model. The methods that use the wrapper model get feedback from the learning algorithm after each update of the feature weights or selected features set.

Salzberg [42] introduced a new algorithm, EACH, that generalizes the data set by defining hyper-rectangles. The algorithm also handles nested hyper-rectangles where the inner hyper-rectangle is an exception of the outer one. EACH randomly selects $n$ elements of the data set to construct these rectangles and uses the rest of the data set to calculate the weights. EACH gets feedback during the classification of the these instances. For each of these instances, the distance between hyper-rectangles and the new instance is calculated using the current weights and they are classified according to the closest hyper-rectangle. If the classification of the instance that is calculated based on this distance is incorrect then the weight of the matching features of the instance and the hyper-rectangle is increased by a constant to make them farther away. If the values of the features do not match than the weight of that feature is decremented by the same constant. One drawback of EACH is, the constant that is

used to adjust feature weights is set by the user. Salzberg [42] outperformed basic kNN algorithms by selecting good values for the feature adjustment constant.

Aha [43] introduced IB4 which is similar to EACH except that it does not update the feature weights by a constant. While updating the feature weights IB4 uses the category distributions of an instance and its nearest neighbor. Therefore, the update of feature weights is category dependent and the features are updated by using

$$\delta(x_f, y_f)(1 - max(p(c_x), p(c_y))) \tag{2.27}$$

where $\delta(x_f, y_f)$ calculates the distance between the feature $f$ of $x$ and $y$, $c_x$ and $c_y$ stands for the category of $x$ and $y$.

Lowe [5] designed a new method, variable-kernel similarity metric (VSM), which optimizes the feature weights and the similarity metric such that the leave-one-out cross validation (LOOCV) error of kNN on the data set is minimized. VSM uses the conjugate gradient technique to optimize the error therefore the optimization converges rapidly without any convergence parameter. VSM also uses an optimized Gaussian kernel to weight the importance of each nearest neighbor of any instance. One advantage of VSM is, it does not need any configuration parameters that is set by user. Lowe [5] compared VSM with other methods and reported good results.

Skalak [44] defines a new method that assigns binary weights to the features. The method he proposes is a genetic algorithm that uses random mutation hill climbing to weight the features. The method starts with a binary weight string and mutates each bit of this string to obtain smaller classification error. The algorithms halt when it reaches the maximum number of iterations. This method outperformed 1NN on four classification tasks.

Basic feature weighting corresponds to the specific subset of linear transformations that have diagonal matrices. In constrast, *Stretch* can generate arbitrary linear transformations, thus includes features weighting as a special case. Moreover, since *Stretch* uses the wrapper model, it gets feedback from kNN and optimizes feature

weights accordingly. The methods like VSM and IB4 uses the wrapper method while *Relief*, *Relief-F*, VDM ,CCF and PCF are using the filter method. The later methods are computationally faster than the wrapper methods.

# Chapter 3

# THE STRETCH METHOD

## *3.1 Idea*

The kNN algorithm has some limitations especially when it deals with a data set that has irrelevant features. One solution to the irrelevant feature problem is explained in section 2.2.4 which introduces attribute weighting into the decision function.

Attribute weighting is a well-studied subject and there are a variety of methods that can assign weights to attributes. Some of the well-known methods in the field are discussed in 2.5

As it is also stated in section 2, most of the methods construct a weight vector, $w$, to represent the relevance of attributes for the learning task and use this vector during the classification of the instances. If each diagonal entry of a matrix is set with corresponding $w$ value, then the resulting matrix will be a equivalent feature weighting matrix $W$. However, since all the entries of $W$ except the diagonals are equal to zero, this matrix has no information concerning the relation of the attributes with each other. Therefore, instead of constructing a diagonal matrix, a more generalized full weighting matrix can be constructed. As a result, $W_{general}$ has information about the relations of attributes with each other.

The *Stretch* method iteratively constructs linear transformations, $A_i$, that minimizes the error of the nearest neighbor classification accuracy in the data set, $X$. The method generates arbitrary linear transformations, thus includes feature weighting as a special case. The final linear transformation is constructed to satisfy two basic goals:

1. To bring the instances in the same category closer,

2. To push the instances from different categories apart.

*Stretch* uses the wrapper model that is discussed in section 2.7.2 and it gets feedback from kNN about the iteratively constructed linear transformations, $A_i$ where $i$ represents the iteration number. The final linear transformation, $A_f$, is a combination of $A_i$ of selected iterations.

### *3.2 Motivating Example*

The effect of the method can be seen on a 2D artificially generated 2 category data set. On figure 3.1 the effect of the method on the boundary and the data set can be clearly seen. The category boundary of the original data set becomes smaller as the data set is stretched, meanwhile the instances from the same category come closer.

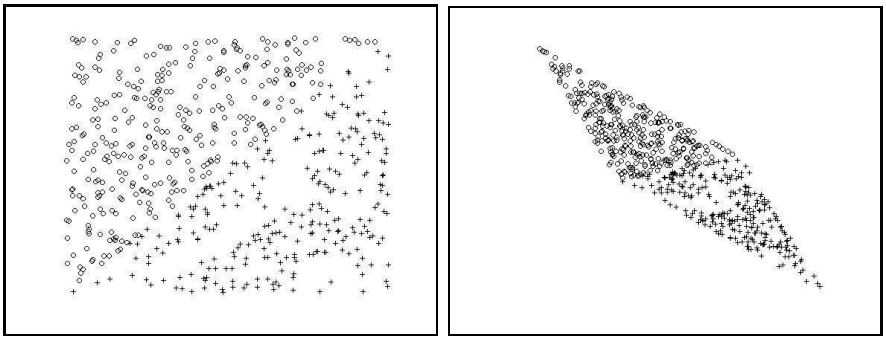

Figure 3.1: The figure on the left is the initial data set with 400 instances and 2 categories showed with plus and star. The one on the right is after stretching the original data set.

If the above example is rotated 45 degree by a full matrix then the y-axis will be an irrelevant feature. A feature weighting method constructs a diagonal weight matrix. However, this matrix can not both rotate and select the relevant features. On the other hand, *Stretch* constructs a full matrix which can handle these types of problems.

### 3.3 *Definition of* Stretch

At iteration $i$, *Stretch* selects a random misclassified instance, $x$ and its nearest neighbor, $x_{NN}$ to construct $A_i$ based on the distance vector, $d$, where $d = x - x_{NN}$. The $A_i$ matrix stretches or shrinks the data set $X$ along the $d$ vector. If the number of nearest meighbors $k$ is bigger than 1 then $x_{NN}$ can belong to the same category with $x$ or different category from $x$. $A_i$, either pushes $x_{NN}$ further from $x$ or brings it closer to $x$ according to the category of $x_{NN}$. $A_i$ is accepted if it reduces the loss function of the method on the whole training set, $E_{stretch}$, otherwise it will be discarded. $E_{stretch}$ is defined as

$$E_{stretch} = E_{NN} + C_{penalty} \tag{3.1}$$

where $C_{penalty}$ stands for the complexity penalty of the stretch matrix. As a result, instead of restricted to principle components [1], *Stretch* constructs linear transformations along any arbitrary direction. Accepted transformations are multiplied to construct $A_F$ which is a full feature weighing matrix that aims to improve the performance of kNN on the data set.

### 3.4 *Main Loop*

The main loop of *Stretch* is composed of three main steps:

- Stage 1: Determining the misclassified instance set, $M$, by kNN,

  In Stage 1, *Stretch* constructs the misclassified instance set and calculates the error of the kNN, $E_{NN}$, on the given data set.

- Stage 2: Constructing a linear transformation, $A_i$,

  In Stage 2, the algorithm picks a random instance from $M_x$, and constructs a linear transformation $A_i$ based on this instance.

- Stage 3: Calculating $E_{stretch}$

  In Stage 3, $A_i$ is temporarily accepted and multiplied with the former accepted $A_i$'s to construct a temporarily $A'_F$. $X'$ is calculated by multiplying the data set $X$ by $A'_F$ and Stage 1 is re-run for the transformed data set, $X'$. At this stage a move penalty is defined for $A'_F$ for regularization in terms of the singular values of $A'_F$. The $E_{stretch}$ is calculated using the move penalty of $A'_F$ and the $E_{NN}$. If the $E_{stretch}$ has improved the previous loss then $A_i$ is accepted and accepted iteration counter is incremented, otherwise $A_i$ is discarded and a new misclassified instance selected from $M$ set of $X$. The final linear transformation, $A_F$, is constructed by multiplying all the accepted linear transformations.

Table 3.1: Pseudo-code of Main Loop. The data set, the number of nearest neighbor, the penalty coefficient, the $\alpha$ value used for the instances in the same category and from the different categories are represented by $X$, $k$, $\alpha_{same}$ and $\alpha_{diff}$. $A_F$ and $A_V$ represent the final transformation matrix of the data set and validation set.

---

**Algorithm 3.4.1:** MAIN LOOP$(X, k, C, \alpha_{same}, \alpha_{diff})$

---

**while** i< $accept_{max}$ AND j< $jump_{max}$

**comment:** Checks for halting criteria

**do**
- $i, j, c \leftarrow 0$
- $A_F = Identity(dim(X))$
- *Stage*1: $(M, E_{NN}) \leftarrow Construct\_M(k, X)$
- *Stage*2: $A_i \leftarrow Construct\_A(x, x_{NN}, \alpha_{same}, \alpha_{diff})$
- *Stage*3:
  - $E'_{stretch} \leftarrow Calculate\_E_{stretch}(A_i, A_F, C, k, X, E_{stretch})$
  - **if** $E'_{stretch} \leq E_{stretch}$
    - **then**
      - $A_F = A_i A_F$
      - $X' = A_F X$
      - $E_{stretch} \leftarrow E'_{stretch}$
      - $i \leftarrow i + 1$
      - **if** $A_i$ reduces $E_{stretch}$ on validation set
        - **then**
          - $A_V \leftarrow A_i A_V$
          - **go to** *Stage*1
    - **else**
      - **if** cut> $cut_{max}$ OR Can not improve $E_{stretch}$
        - **then**
          - **comment:** The algorithm does a random jump
          - Do Stage2
          - Update $E_{stretch}$, $F$ and $X$
        - **else go to** *Stage*2

---

### *3.5 Convergence*

The convergence of *Stretch* is controlled by three variables.

1. $accept_{max}$

   This variable controls how many linear tranformations are accepted by *Stretch*. If the number of accepted transformations reaches this upper limit than the method stops iterating and returns $A_F$.

2. $jump_{max}$

   This variable is the upper limit of the maximum number of allowed random jumps that the algorithm can do. If all the misclassified instances in the current data set are not able to construct an accepted linear transformation then the algorithm selects the misclassified instance that least distorts the data set and accepts the linear transformation without considering the effect on the $E_{stretch}$. If this variable reaches its upper limit than the algorithm stops iterating and returns the final linear transformation.

3. $cut_{max}$

   This variable is the upper limit of the number of consecutive accepted linear transformations that give the same $E_{stretch}$. Although these consecutive linear transformations do not improve the error, they have side effects on data set that are discussed in next chapter. If this consecutive non-improving linear transformation chain is broken by a linear transformation that changes the $E_{stretch}$ then the value of cut will be set to zero. If the value of the cuts reaches this upper limit then the method resets the number of cuts and jumps.

Higher values of these variables increase the chance of over fitting which is discussed in section 3.11.

### *3.6 Stage1 Definition*

In Stage 1, the algorithm constructs the misclassified instance set, $M$, using leave-one-out cross validation (LOOCV) in kNN. Any instance that is misclassified increases the $E_{NN}$ and gets added to $M$. The size of $M$, depends on $k$ and the decision function that is used for classification. For example, an instance that is classified correctly by $(k \leq c)$NN, might be classified incorrectly by $(k > c)$NN or vice versa, where $c \geq 1$. While constructing $M$, the algorithm also constructs the $k$ nearest neighbor set of each instance and keeps the distances to each neighbor in this set. The $k$ nearest neighbor set of $x_i$ and distances from $x_i$ to each of its $k$ nearest neighbor is represented by $kNN_{x_i}$ and $kD_{x_i}$, respectively.

This stage has a crucial role because all the linear transformations that is constructed on Stage 2 will be based on a misclassified instance that is selected from $M$.

#### *3.6.1 Misclassified Instances*

The misclassified instances can be categorized into two groups:

1. Noisy Instances

   As it is explained in section 2.4.4, these instances do not truly reflect the properties of the categories they belong. Therefore, if *Stretch* selects a noisy instance, $x_{noise}$, and constructs a linear transformation accordingly, this might transform the data set along an undesired direction. Moreover, after this transformation, it is highly probable that $x_{noise}$ is still misclassified by kNN and therefore, will be a member of $M$ on the next iteration of main-loop. As a result, *Stretch* might choose $x_{noise}$ more then once and keep repeating stretches in the wrong direction.

2. Boundary Instances

Figure 3.2 presents the difference between noisy instances and the boundary instances. These instances define the category boundary and therefore the effects of linear transformations based on these instances are more predictable and desirable. The misclassification probability of these instances by kNN is high compared to the non-boundary instances.

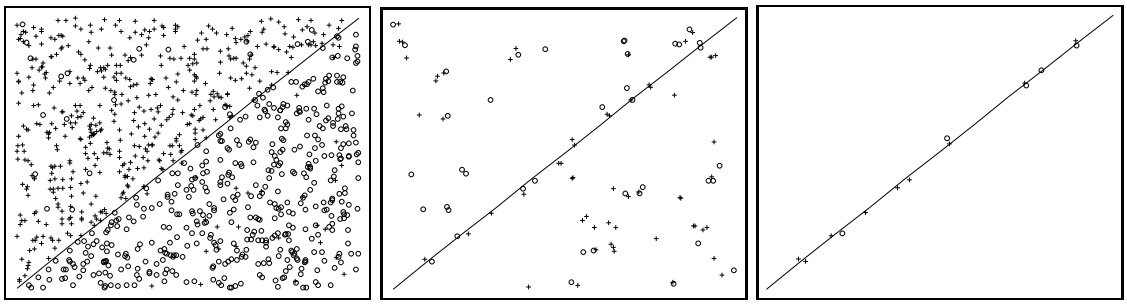

Figure 3.2: An example of instance types. The figure on the left is the original input space in two dimensions with two distinct categories indicated by circles and pluses. The one in the center shows the misclassified instances whereas the one on the right shows only the boundary instances by LOOCV of 1NN.

In order to satisfy the goals of the algorithm, $M$ should contain misclassified instances that define category boundaries. This can be accomplished by selecting smaller $k$ values since as $k$ becomes larger, the ratio of the number of noisy instances to number of boundary instances becomes greater. As a results, as the number of noisy instances in $M$ increases *Stretch* constructs misleading linear transformations.

The impact of the distance weighting function, described in the section 2.2.5, on the success of *Stretch* is more obvious. In some cases, *Stretch* constructs a linear transformation that neither increases nor reduces the $E_{stretch}$. In these circumstances, linear transformations have a side effect. Since any accepted transformation linearly transforms the data set, the distances between instances are changed. Obviously, this side effect is only available when the distance weighting function is not a constant function and has a distance parameter.

### *3.6.2 Pseudo-code of Stage 1*

Table 3.2: Pseudo-code of Stage 1. The data set and the number of nearest neighbor is represented by $X$ and $k$, respectively.

**Algorithm 3.6.1:** CONSTRUCT_$M(k, X)$

**for each** $x \in \mathcal{X}$

**do**
- **comment:** Leave one out kNN(LOOkNN)
- $kNN_{x_i} \leftarrow$ Find k nearest neighbor(s) of x
- $kD_{x_i} \leftarrow$ Distance to each k nearest neighbor
- **if** x is classified incorrectly by its k nearest neighbor(s)
  - **then**
    - $M \leftarrow M + x$
    - $E_{NN} \leftarrow E_{NN} + 1$

**return** $(M, E_{NN})$

### *3.6.3 Design issues of Stage1*

**Constant Distance Weighting Function (i.e.,Majority Voting)** When kNN uses majority voting, distance information between the query instance, $q$ and the its $k$ nearest neighbors is not used. Therefore, after a linear transformation, the nearest neighbor set of the $q$ may not change although the distances between the nearest neighbors are changed. As a result, $E_{NN}$ will be equal to the value before the transformation.

**Distance Weighting Function with Distance Parameter** In these functions, any change in distances between $q$ and its nearest neighbors affects the decision function. Therefore, even if *Stretch* constructs a linear transformation that does not

change the nearest neighbor set of $q$, $E_{NN}$ might be reduced due to the distance changes.

### *3.7 Stage 2 Definition*

In this stage, *Stretch* constructs a linear transformation matrix, $A_i$, at each iteration. The method first picks a random instance $x$ from $M$ then calculates the distance vector $d$ between $x$ and its nearest neighbor, $x_{NN}$. The $A_i$ matrix is defined as

$$A = QVQ^T \tag{3.2}$$

where $Q$ and $V$ are an orthonormal and a diagonal matrices, respectively. The $Q$ matrix is orthonormal version of $S$ where $S$ keeps direction information of the stretch (transformation). $Q$ is constructed by applying the QR decompostion on $S$. Each column of the $Q$ matrix keeps the orthonormal bases of the stretch matrix $S$ whereas each diagonal entry of $V$ stands for the stretching coefficients along the corresponding base. In order to stretch along the desired direction, *Stretch* initializes $V$ and $S$ to identity matrix then replaces the first column of the $S$ matrix with $d$ and first diagonal entry of $V$ with $\alpha_{same}$ or $\alpha_{diff}$. The selection of $\alpha$ is determined by the category of the $x$ and $x_{NN}$. If the categories of these two instances are same then $\alpha_{same} < 1$ is used to bring these two instances together otherwise $\alpha_{diff} > 1$ is used to push these instances apart. When $k$ equals to 1 then the algorithm always uses $\alpha_{diff}$ since all the $x$ from $M$ have nearest neighbors from a different categories. On the other hand when $k > 1$, $x_{NN}$ may belong to the same category with $x$.

For example, if $A_i$ is constructed by setting the first diagonal entry of $V$ to $\alpha$ and $S$ to identity, then $A_i$ will stretch the data set along the $[1, 0, 0, .., 0, 0]$ by $\alpha$ amount.

The method checks the singularity of $S$. If any singularity occurrs after replacing the first column with $d$ then the column that is responsible for singularity is replaced by an appropriate unit vector to remove the singularity. Introducing $d$ into $S$ removes the orthogonal property of the $S$ matrix, therefore the algorithm uses QR decompo-

sition to construct a new orthonormal matrix, $R$. QR decomposition by means of Gram-Schmidt process also solves the singularity problem of $S$ by introducing new orthogonal bases along the singular dimensions. In the final step of stage two, $A_i$ is calculated by using $Q$ and $V$ matrices.

### 3.7.1 Stage 2 Pseudo-code

Table 3.3: Pseudo-code of the Stage 2. The randomly selected misclassified instance, its nearest neighbor and the values of $\alpha$ is represented by $x$, $x_{NN}$, $\alpha_{same}$ and $\alpha_{diff}$. The category of the $x$ and $x_{NN}$ is represented by $c_x$ and $c_{x_{NN}}$.

---

**Algorithm 3.7.1:** CONSTRUCT_$A(x, x_{NN}, \alpha_{same}, \alpha_{diff})$

---

$d = x - x_{NN}$

$n = dim(x)$

$V = Identity(n)$

$S = Identity(n)$

**comment:** Set $V$ and $S$ to n by n identity matrix.

Replace $1^{st}$ column of $S$ with $d$.

**if** $c_x = c_{x_{NN}}$

  **then** $V(1,1) \leftarrow \alpha_{same}$

  **else** $V(1,1) \leftarrow \alpha_{diff}$

**comment:** Set the $1^{st}$ diagonal entry of $V$ and $1^{st}$ column of $S$

$(Q, R) = QR(S)$

**comment:** Construct the orthonormal matrix Q using QR decomposition.

$Volume_A = \prod_{i=1}^{n} v_i$

$V \leftarrow V / \sqrt[n]{Volume_A}$

$A \leftarrow QVQ^T$

**return** $(A)$

---

### 3.8 *The* Stretch *Matrix Composition*

Every row of the data set $X$ is an instance vector form $R^n$. Therefore, multiplying $A_i$ with $X$ is equivalent to multiplying each instance vector with $A_i$. Multiplying an instance vector with a matrix rotates and/or scales this vector.

The $A_i$ matrix is a linear transformation that shrinks or stretches along $d$ therefore any $x$ that has a component parallel to $d$ vector is affected. The $Q$ matrix that is described previously is constructed such that the first column of $Q$ is a normal vector that keeps the stretch direction and the rest of the mutually orthonormal columns are responsible for the remaining $n-1$ dimensions, where $n$ is the number of features of the data set.

As a result, multiplying $Q^T$ with $x$ is equivalent to rotating $x$ on to a unit circle that is defined by $Q^T$.

$$Q^T x = x' \tag{3.3}$$

where $x'$ is a vector that is described by the rows of $Q^T$. By using this fact $x'$ can be scaled along the desired direction (i.e., along $d$) by setting the corresponding diagonal entry of the $V$ matrix. For example, if we want to scale $x$ along $d$, it is enough to set the first diagonal entry of the $V$ matrix to $\alpha$ while setting the rest of diagonals to 1.

$$VQ^T x = x'' \tag{3.4}$$

where $x''$ is a vector that is scaled version of $x$ and defined by the bases of $Q^T$. Therefore in order to get a stretched version of $x$, $x''$ is rotated back to the original domain of $x$ by multiplying $Q$.

$$QVQ^T x = x_{stretched},$$
$$A_i x = x_{stretched},$$
$$\forall x \epsilon X \ , \ A_i X = X_{stretched}$$

where $Q$ is a set of orthonormal basis vectors (i.e., singular vectors) of $A_i$, $Q^T$ is the transpose of $Q$ (since $Q$ is an orthonormal matrix $Q^T = S^{-1}$) and the diagonal entries of $V$ are the singular values of $A_i$ matrix.

This is a special case of SVD of $A_I$ that is also called Eigenvalue Decomposition(ED). This is because, $A_i = QVQ^{-1}$ and the diagonal entries of $V$ are non-negative, as a result $A_i$ is a positive semidefinite Hermitian matrix.

## *3.9 Normalization*

At the beginning of the main loop $A_F$ is equal to $I$ therefore the length of all singular vectors of $A_F$ are equal to 1. At each iteration, $A_i$ either stretchs or shrinks $A_F$ which changes the singular vectors of $A_F$. If we think of a prism that is defined by the mutually orthanormal singular vectors of $A_F$ with a unit volume before the transform, then multiplying it with $A_i$ changes its volume by the amount of $\alpha_{same}$ or $\alpha_{diff}$. As a result, as the number of iterations increases the entries of $A_F$ will become smaller or larger which may cause an arithmetic overflow during the execution.

### *3.9.1 Volume of $A_i$*

The $A_i$ matrix is defined as

$$A = QVQ^T \tag{3.5}$$

where $Q$ and $V$ are an orthonormal and a diagonal matrices, respectively. The matrix $Q$ can be defined in terms of its column vectors, $Q = [q_1, q_2, .., q_n]$. Because of the orthonormal property of $Q$

$$q_1 \perp q_2 \perp \cdots \perp q_n,$$
$$|q_i| = 1, \text{for i=1,2,}\cdots\text{,n.}$$

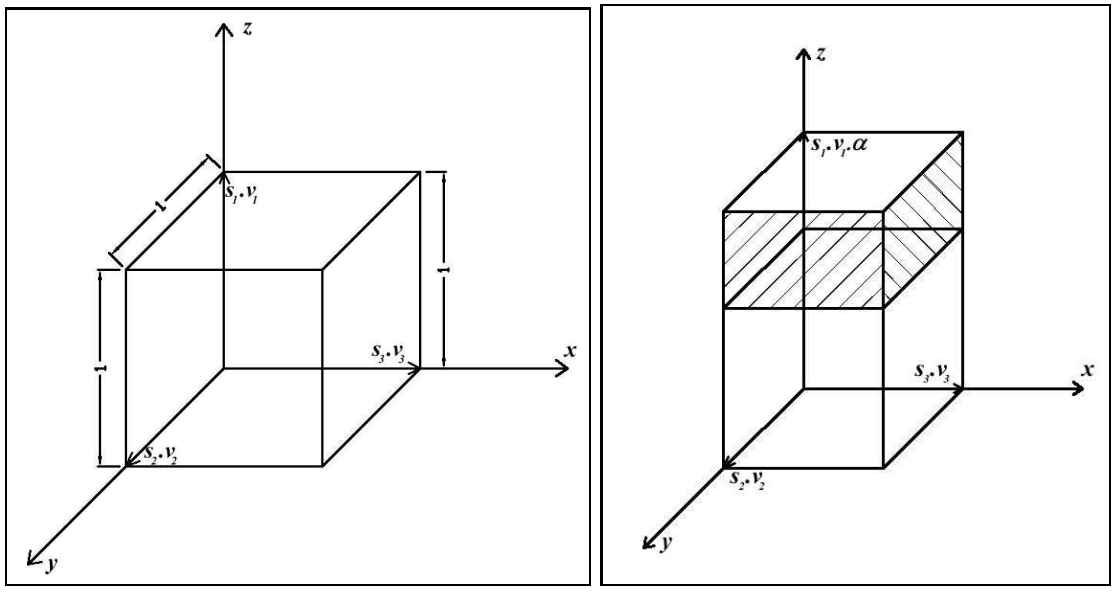


Figure 3.3: The figure on the left is the initial stretch matrix, $A$, that is equal to $I$ therefore $Volume_A$ is equal to 1. The one on the right is after stretching $A$ along $d = z$ direction by $\alpha$ times. As a result the new $Volume_A$ is larger than 1.

The $Volume_{A_i}$ is defined in terms of the singular vectors of the $A_i$ matrix. Since the singular vectors are orthonormal to each other we think of these vectors as the vertices of a prism. The length of each $q_i$ equals to 1, therefore singular vectors only have direction information. The length of the each column vector is kept in the corresponding diagonal entry of $V$. If each diagonal entry of $V$ is defined as $v_i$ where $i = 1, 2 \cdots, n$ then $Volume_{A_i}$ is equal to

$$Volume_{A_i} = \prod_{i=1}^{n} v_i \tag{3.6}$$

In order to keep the $Volume_{A_i}$ equals to 1, each $v_i$ of $A$ is divided by $\sqrt[n]{Volume_{A_i}}$. As a result, each entry of $A_i$ is divided by a constant term determined by $\alpha$ that is used to construct that particular $A_i$. The normalization affects only the values of each entry of $A_i$, it does not change the relative ratios between the values since each entry is divided by the same constant term.

Otherwise the $Volume_{A_F}$ will be different from unity.

This problem is solved by normalizing the entries of $A_i$ in order to keep the volume of $A_F$ equals to unity.

### *3.10 Stage 3 Definition*

In this stage, *Stretch* calculates $E_{stretch}$ by adding the complexity penalty of $A'_F$ to $E_{NN}$ where $A'_F$ is the temporarily final matrix equals to $A_F A_i$. This stage prevents *Stretch* from overfitting the training data. Moreover, this stage defines a move penalty for the each misclassfied instance in order to select a missclassfied instance such that the linear transformation constructed by using it does not deform the data set significantly. As a result, Stage 3 is responsible for the regularization of the *Stretch* method.

### *3.10.1 Pseudo-code of Stage3*

Table 3.4: Pseudo-code of Stage 3. Current stretch matrix, the final stretch matrix, the penalty coefficient, the number of nearest neighbor , the data set and the loss function of $A_F$ is represented by $A_i$, $A_F$, $c$, $k$, $X$ and $E_{stretch}$, respectively. The diagonal entries of $V$ matrix are represented by $v_i$ where $i = 1..n$.

---

**Algorithm 3.10.1:** CALCULATE_$E_{stretch}(A_i, A_F, c, k, X, E_{stretch})$

---

$[S, V, D] \leftarrow SVD(A_i A_F)$

$\delta = \sum_{i=1}^{n} (\log v_i)^2$

$X' \leftarrow A'_F X$

$(M, E_{NN}) \leftarrow Construct\_M(k, X')$

**comment:** Run stage1 on $X'$

$E'_{stretch} \leftarrow E_{NN} + c\delta$

**return** $(E'_{stretch})$

---

## *3.11 When does* Stretch *may not improve kNN accuracy?*

*Stretch* minimizes $E_{NN}$ by constructing linear transformations ,however not all the transformations lead to a reduction in $E_{NN}$. This is mainly due to:

- Structure of the data set

  A linear transformations that minimizes the kNN error can be constructed for any data set that is separable by linear hyper-planes or at least the categories of the data set can be separated by linear transformations while reducing $E_{NN}$ significantly. However, for some data sets it is not possible to construct a linear transformation that decreases the $E_{NN}$ significantly.

- The input space subset that is used to construct linear transformation

  *Stretch* constructs the linear transformations using some subset (i.e., training set) of the input space. Therefore the training set partially reflects the structural properties of the input space. However unclassified instancesi (i.e., test set) can belong to any part of the input space. As a result, although a linear transformation reduces $E_{NN}$ of the training set significantly, similar effect might not be observed on $E_{NN}$ of the test set.

### *3.11.1 Overfitting*

For some particular data sets the significant improvement on the training set is not observed on test set. This is because the algorithm memorizes the training set instead of learning it. The problem of memorizing instead of generalizing on training set is called over-fitting.

### *3.11.2 Solutions to Overfitting*

There are common methods and some algorithm specific methods to cope with overfitting.

- Validation Set
- Regularization
- Validation Set with Regularization
- Algorithm Specific Methods

### *3.11.3 Validation set*

In this method, some portion of the training set is used as a fake test set (i.e., validation set) and the performance of the algorithm is evaluated both over the $E_{NN}$ of the training and the validation data sets.

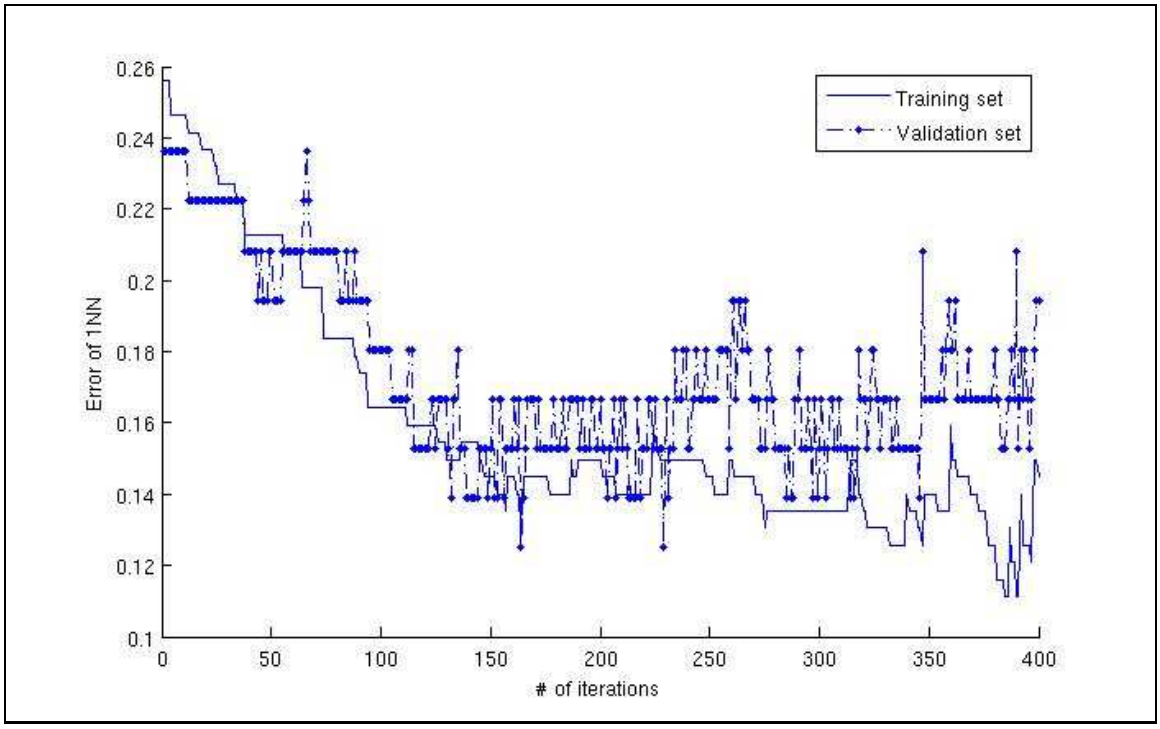


Figure 3.4: The figure shows the error of 1NN on the training set and the validation set of heart-h after iteratively applying the transformation matrix that is constructed to optimize LOOCV error of the training set.

Figure 3.4 displays the convergence of the 1NN error on the training and the test set of heart-h data set [45]. The errors of both data sets decrease as the number of iterations increase. However at some point the error of the validation set increases even though the error of the training data keeps decreasing which means, at some point *Stretch* overfits the training set. Although the validation data set solves the over-fitting problem for most of the data sets, it does not give any information about which stretch direction is better to minimize the $E_{NN}$. Since validation set is used as a test set, any instance that is member of validation set can not be in training set therefore the information content of the data set decreases.

Regularization solves these problems of the validation set, meanwhile it does not reduce the size of training data.

### *3.11.4 Regularization*

The Regularization method introduces a complexity penalty into *Stretch*. As a result, as the model becomes more complex (i.e., the ratio of the largest singular value to the smallest singular value of $A_F$ increases) *Stretch* is punished. The move penalty, $\delta$, is calculated for every linear transformation. Therefore instead of using only $E_{NN}$, it uses

$$E_{stretch} = E_{NN} + c\delta \tag{3.7}$$

where $c$ is a penalty coefficient that balances the effect of $E_{NN}$ and $\delta$. The value of $c$ is determined empirically and varies with data set. The move penalty $\delta$ has two important roles:

1. Penalize the move

   In Stage 2 of *Stretch*, a random instance selected then $A_i$ is constructed accordingly for this particular iteration. $A_i$ stretches or shrinks the data set, as a result although the volume of the data set does not change, the principle components of the data set are affected by this linear transformation. For example, a 2D data set initially has $v_1 = 1$ and $v_2 = 1$ as singular values and $Volume_{2D} = v_1 v_2 = 1$. Assume that this 2D set is stretched along $v_1$ by 1.05. As a result, $v_1 = 1.054$ and $v_2 = 1$. After normalizing the 2D data on stage2 $v_1 = 1.029$ while $v_2 = 0.98$ and $Volume_{2D} = 1$.

   Every transformation stretches or shrinks the current data set along some direction while keeping the volume equal to 1 thus at each iteration the difference between the initial data set and the current modified data set becomes larger. The amount of stretch is determined by $\alpha$, therefore every selected instance during the construction of linear transformation has same impact on the the singular values of $A_F$. By using this fact the difference between the initial data set and the current modified data set can be defined in terms of singular values.

As a result each selected move has a penalty, $\delta$, according to the distortion it causes on data set.

**Definition of move penalty, $\delta$:** The data set volume does not depend on the random instance selection in Stage 2, therefore the move penalty is defined by the singular values of the data set. Initially each singular value is equal to 1 and any linear transformation updates the singular values. For example , if the algorithm always stretches along the direction that $v_1$ corresponds than $v_1$ becomes larger compared to the rest of the singular values as the number of iterations increases. The move penalty, $\delta$, is defined as

$$\delta = \sum_{i=1}^{n} (\log v_i)^2. \tag{3.8}$$

In order to *Stretch* to accept $A_i$, $A_i$ has to decrease the $E_{NN}$ while it does not distort the data set significantly compared to the other possible linear transformations.

2. Penalize the complexity

   As the number of iterations increases $\delta$ becomes larger or smaller. This is due to the differences between the singular values becoming larger. At some point, the $\delta$ term dominates over the $E_{NN}$ term and prevents the algorithm from accepting the linear transformations. The $E_{NN}$ of the data set can be reduced by *Stretch*, $E_{stretch}$ does not let *Stretch* to do any further reduction on the error of $E_{NN}$ of the data set with careful selection of the coefficients this may prevent overfitting.

### *3.11.5 Validation set with Regularization*

This method can be called as guided validation set which uses the advantages of both of the methods described in previous sections. In this method, regularization together

with validation set decreases the chance of overfitting meanwhile regularization increase the quality of the linear transformations.

### *3.11.6 Algorithm Specific Methods*

If $iterate_{max}$ is increased then the algorithm overfits the training set. However the value of $iterate_{max}$ has to be determined by the experimenter and this method also does not penalize the linear transformations.

## *3.12 Computational Complexity*

Each iteration of the algorithm requires the computation of the k nearest neighbors of all points, QR decomposition of $A_i$, and singular value decomposition to calculate the move penalty of that iteration. For an $m$x$n$ data set, the kNN task can be accomplished in $O(m \log m)$ average time using a data structure such as vp-trees[17]. QR decomposition of $A_i$ can be calculated $O(n^2)$ time and the calculation of the move penalty by using SVD can be completed in $O(n^3)$ time.Empirically the algorithm converges after trying each point in the boundary a small number of times, i.e. the number of iterations is $O(m)$. The overall time complexity of the algorithm is thus $O(m(m \log m + n^3 + n^2))$ which can be rewritten as $O(m^2 \log m + mn^3)$.

## *3.13 Can* Stretch *construct any transformation?*

Most feature weighting methods in the literature correspond to diagonal stretch matrices in our setting. One interesting question is whether the *Stretch* algorithm can potentially construct any arbitrary transformation matrix. The answer turns out to be yes:

**Theorem:** Any square matrix A can be constructed by the product of a number of $A_i$.

**Proof:** Any matrix can be written as a product of three normal matrices due to the SVD decomposition. A normal matrix has orthogonal eigenvectors and can be

written as a sum $\sum \lambda_i u_i u_i^*$ where $\lambda_i$ are eigenvalues and $u_i$ are eigenvectors. This sum can be expressed as a product of stretch matrices:

$$\sum \lambda_i u_i u_i^* = (\lambda_1 u_1 u_1^* + u_2 u_2^* + \cdots + u_n u_n^*)(u_1 u_1^* + \lambda_2 u_2 u_2^* + u_3 u_3^* + \cdots)$$

$$\ldots (u_1 u_1^* + u_2 u_2^* + \cdots + \lambda_n u_n u_n^*)$$

Thus any matrix can be decomposed into a product of stretch matrices. In order to construct rotation matrices one have to construct the stretch matrix using the complex direction vectors.

# Chapter 4

# EXPERIMENTAL RESULTS

Many of the feature weighting methods were surveyed in [1] and all of these methods were compared according to their kNN classification performances on fourteen different data sets. In order to compare the results of *Stretch* with other feature weighting methods, the experiments are conducted on nineteen data sets that include twelve of the data sets that were used in [1]. Isolet and NETtalk data sets are in [1], however they are excluded because of the computational restrictions. The experiments of *Stretch* are conducted with the same methodology that is used in [1]. In order to reduce the bias of the experiments a control feature weighting method is implemented. In [1], it can be observed that, *Relief-F* [4] and $kNN_{VSM}$ [30] are the best feature weighting methods based on the number of significant positive or negative effects on the classification accuracy of kNN. Therefore, *Relief-F* is used as a control method while comparing the results of the experiments. Moreover, it is applied to the new data sets that are not included in [1]. *Relief-F* is implemented as a feature weighting method by ignoring the threshold part of the original method or setting the threshold value to a value lower than the minimum weight.

## *4.1 Compared Method:* Relief-F

In section 2.7.1, *Relief-F* and its predecessor, *Relief*, are discussed in detail. The main reason of selecting *Relief-F* as a control method is the way *Relief-F* calculates the feature weights. In chapter 3.6.3, the calculation of the weight matrix is discussed in detail. The construction of the weight matrix is based on the distance vector, $d$, between the misclassified instance and its nearest neighbor from a different category. *Relief* and *Relief-F* use a similar weight calculation approach with *Stretch*. In

equation 2.25, the weights are calculated based on the distance between the query instance and its nearest neighbors with the same and a different category. Therefore the equation 2.25 can be rewritten as

$$W = W - \delta(x, H)/m + \delta(x, M)/m \tag{4.1}$$

The above equation is a generalization of the equation 2.25 and instead of calculating the feature weights one by one as it is in equation 2.25, it calculates the feature weight vector, $W$. In equation 4.1, $\delta(x, y)$ calculates the distance vector between x and y. The right hand side of the equation can be rewritten as

$$W = W - d' \tag{4.2}$$

where the entries of the $d$ vector is the sum of two $\delta$ function for the corresponding features in the equation 4.1. The last equation shows that *Relief* also updates the weights based on a distance vector that is similar to the idea of *Stretch*. *Relief-F* calculates $d'$ using all of the categories in the data set and it weights each particular distance vector by multiplying them with the prior probabilities of the categories. As a result, *Relief* and *Relief-F* are similar to *Stretch* in terms of the way they calculate the feature weights. In [8] the weight update procedure of *Relief-F* is formulated as

$$w(f) = w(f) - \delta(x_f, H_f)/m + \sum_{i=1}^{|C|} \frac{p(c_i)}{1 - p(c_x)} \delta(x_f, M_f(c_i))/m \tag{4.3}$$

where $p(c_x)$ is the prior probability of the category of $x$. On the other hand there are some differences between these two algorithms and these are

- *Stretch* is using the wrapper method while *Relief-F* is using the filter method,
- *Stretch* computes the distance vector by only using the misclassified instances while *Relief-F* uses all the instances of the data set,

- *Stretch* constructs a full weight matrix while *Relief-F* constructs a diagonal weight matrix,

- *Relief-F* uses the distribution of the categories while calculating the distance vector.

These differences can be eliminated by modifying the methods and these modifications are discussed in detail in chapter 5.

## *4.2 Selected Datasets*

Twenty data sets have selected to run the experiments. Twelve of these data sets are also used in [1]and marked with "*", four of these data sets are artificially created, and eight of them are from the UCI Repository [45]. In order to be compatible with [1], artificial data sets are generated based on the definitions that are presented in [1].

### *4.2.1 Definition of The Data Sets*

The four artificial data sets are

- Banded Data set*

  This data set has two features and ten categories. Each of these categories are defined by a line that is parallel to x-axis therefore the x value of an instance is irrelevant.

- Sinusoidal Data set*

  This data set has two features and two categories. Each of these categories is defined by the border of the sine function.

- Gauss Data set*

  This data set has five features and fourteen categories. The data set is a combination of the banded data set and a four category 2D gauss distribution. Each

category of the gauss data set is defined by a 2D gauss distribution with a variance of 0.025. The fifth feature is a binary feature that decides whether the instance is classified according to the bands or the 2D gauss distributions. As a result, this data set has interacting features.

- Parity Data set*

  This data set has eleven features and two categories. The sum of the first four features decides the category of the instance (i.e., class one if the sum is even, class two otherwise). The remaining seven features are randomly generated irrelevant features.

The rest of the the data sets are

- LED Display Data set*

  This data set is a script generated data set and it is also used in [1] and it has seven features, ten categories and ten percent of noise. As in the gauss data set, some of the relevant features of LED Display are interacting.

- LED17B Display and LED17C Display Data set*

  LED17B Display and LED17C Display data sets are generated by adding seventeen random binary and continuous features to the LED Display data set, respectively. Therefore it has twenty four features, ten categories and ten percent noise.

- Waveform21 Data set*

  This data set is created using the data generator used in [1] and it has twenty one features, three categories and ten percent noise.

- Waveform40 Data set*

This data set is generated by adding nineteen irrelevant features to the Waveform21 data set therefore it has forty features and three categories and ten percent noise.

- Cleveland, Hungarian and Vote Data sets*

  [1] cited [46] for the observations and explanations about these data sets. Some of the features of these data sets have no significant effect on the classification accuracy of kNN, therefore they can be removed. Hungarian and Cleveland data sets have thirteen features and two categories while vote data set has sixteen features and two categories. These data sets have missing features in some of the instances.

- Hayes Data set

  This data set has four features and three categories. Two features are randomly generated and they are not used during the classification task.

- Glass Data set

  This data set has nine features and seven categories.

- Cars Data set

  This data set has six features and four categories and it is mainly used to test structure discovery and constructive induction methods [47].

- Tic-Tac-Toe Data set

  This data set has nine features and two categories and it is used to test constructive induction methods.

- Monk's problem 1, Monk's problem 2 and Monk's problem 3 Data sets

These data sets have six features and two categories. These data sets were used in the first international comparison of learning algorithms [48]. Each data set has irrelevant or interracting features. Monk's problem 3 has five percent noise.

- Nursery Data set

  This data set has eight features and five categories and it is mainly used to test structure discovery and constructive induction methods [47].

### *4.2.2 Missing values*

The missing values of the continuous (nominal) features are replaced with the meansi (modes) that are calculated using the data set.

### *4.2.3 Binarization*

Nominal features are converted to binary features. The conversion procedure defines a new binary feature for every value of the given nominal feature and sets this binary feature to 1 if the instance has the corresponding nominal value or 0 otherwise.

### *4.2.4 Dataset Normalization*

Each value of a feature is normalized using

$$x_f = \frac{x_f - f_{min}}{f_{max} - f_{min}} \tag{4.4}$$

where $x_f$ is the value of the feature $f$ and $f_{min}$ and $f_{max}$ are the minimum and maximum value of the feature $f$ in the training set, respectively.

Table 4.1: Properties of the selected data sets. Binary, numeric and continuous features are represented by b, n and c, respectively. Unknown properties are represented by '?'. The horizontal line separates the data sets that are used in [1] from others.

| | Set Size | | Number and Types of | Number of Irrelevant | | |
|---|---|---|---|---|---|---|
| Dataset | Training | Test | Features | Features | Categories | Noise |
| Banded | 350 | 150 | 2c | 1 | 10 | no |
| Sinusoidal | 350 | 150 | 2c | 1 | 2 | no |
| Gauss-band | 350 | 150 | 4c,1b | 2 | 14 | no |
| Parity | 350 | 150 | 11b | 7 | 2 | no |
| LED-7 Display | 200 | 1000 | 7b | 0 | 10 | yes |
| LED-7+17B | 200 | 1000 | 24b | 17 | 10 | yes |
| LED-7+17C | 200 | 1000 | 7b,17c | 17 | 10 | yes |
| Cleveland | 212 | 91 | 5c,3b,5n | 0 | 2 | ? |
| Hungarian | 206 | 88 | 5c,3b,5h | 1 | 2 | ? |
| Voting | 305 | 130 | 16b | 0 | 2 | ? |
| Waveform-21 | 700 | 300 | 21c | 0 | 3 | yes |
| Waveform-40 | 700 | 300 | 40c | 19 | 3 | yes |
| Cars | 279 | 1449 | 6n | 0 | 4 | no |
| TicTacToe | 670 | 258 | 9n | 0 | 2 | no |
| Glass | 149 | 64 | 9c | ? | 7 | ? |
| Hayes | 132 | 28 | 4n | 2 | 3 | ? |
| Nursery | 648 | 194 | 8n | 0 | 5 | no |
| Monk's Problem 1 | 389 | 167 | 2b,4n | 2 | 2 | no |
| Monk's Problem 2 | 421 | 180 | 2b,4n | 2 | 2 | no |
| Monk's Problem 3 | 388 | 166 | 2b,4n | 2 | 2 | yes |

### *4.3 Methodology*

Every data set is divided into two subsets one of which is the training and the other is the test set based on the sizes in Table 4.1. Methods calculate the feature weights using the training set and then apply these weights to the training set and the test set. *Stretch* constructs a validation set using thirty-five percent of the training set (the validation set size of the LED data set family and the car data set is twenty-five percent of the training set) in order to reduce the effect of overfitting which is described in section 3.11.3. The sizes of the training and test sets for each data set are shown in table 4.1.

*Stretch* keeps the final linear transformation that minimizes the leave-one-out cross validation (LOOCV) error of the training set and the validation set, and these linear transformations are represented by $A_T$ and $A_V$, respectively. The results are presented for both $A_T$ and $A_V$ in order to show which one is a better transformation based on 1NN and BestNN performances on the test set after applying this transformations.

Experiments on each data set are repeated twenty-five times in order to reduce the statistical variance. Six different odd k values (i.e., 1, 3, 5, 7, 9 and 11) are chosen to determine the optimum k value for kNN. One benefit of odd k values is, the chance of ties during the decision process is reduced. The classification results of 1NN and the k value with best classification accuracy, BestNN, is presented for each data set.

When $k > 1$, *Relief-F* also averages the k nearest neighbors from each category. However, during the experiments of BestNN the weights are calculated using one nearest neighbor from each category. BestNN experiments present the effect of weights, calculated using one nearest neighbor, on the classification accuracy of BestNN.

The regularization constant, defined in 3.11.4, is tuned emprically for each data set. In order to determine the effect of the convergence time on the performance of *Stretch*, the $iterate_{max}$ is set to 400 and 800, while the $jump_{max}$ is set to 100 and 200. For all of the experiments the $cut_{max}$ is set to 20. For the rest of the chapter the experiments are labeled based on the number of iterations and the jumps(ex:

100j/400i). Since these variables determine the running time of the algorithm, they are referred to as the convergence configurations for the rest of this chapter.

All the missing values of the data sets are replaced by using the method described in section 4.2.2 then they are normalized and binarized by using the formula in section 4.2.4 and the method that is described in section 4.2.3, respectively.

$\alpha_{same}$ and $\alpha_{diff}$ that are defined in section 3.7 are set to 1,05 and 0.95, respectively. To evaluate the experiment results a paired t-test is performed and a 0.05 level of significance is used. Significant results are written in bold.

The results are scientificly rounded for better visualization, however the paired t-test was applied to the original results.

***Stretch* on the training set:** Table 4.2 presents the LOOCV error of 1NN on the training set after the application of *Stretch*. *Stretch* is run for two different convergence configurations which are represented in columns that are labeled as 100j/400i and 200j/800i, respectively. Since the latter one has longer convergence time, it iterates more and therefore it is expected to construct a better linear transformation in terms of LOOCV error of 1NN on the training set. The results in Table 4.2 satisfies this claim since the LOOCV error of 1NN for the 200j/800i case is better than the 100j/400i case. For some data sets, increasing the convergence time reduces the error to zero as in the parity, waveform40 and tic-tac-toe data sets.

It can be concluded that as the convergence time increase the LOOCV error of 1NN on a training data set approaches zero. However, for some data sets, such as Hayes, increasing the convergence time can lead to an increase in the LOOCV error of the 1NN on training set.

The real problem is, as the convergence time increases the constructed linear transformation that minimizes the LOOCV error of training set will be specific to the training set thus, triggers the overfitting.

One good solution to this problem is to use the $A_V$ instead of using $A_T$. $A_T$ can overfit the training data set therefore it increases the 1NN error on the test set.

Table 4.2: The LOOCV error of the 1NN on the training set before and after applying *Stretch* with two different convergence configurations are presented. Statistically significant results are marked with bold.

| | | Convergence Configuration | |
|---|---|---|---|
| Dataset | Before | 100j/400i | 200j/800i |
| Banded | 0.82±0.01 | **0.97±0.03** | **0.98±0.01** |
| Sinusoidal | 0.86±0.01 | **0.95±0.02** | **0.96±0.03** |
| Gauss-band | 0.63±0.02 | **0.84±0.03** | **0.84±0.02** |
| Parity | 0.66±0.02 | **1±0.00** | **1±0.00** |
| LED-7 Display | 0.51±0.05 | **0.53±0.05** | **0.54±0.05** |
| LED-7+17B | 0.38±0.03 | **0.73±0.04** | **0.76±0.04** |
| LED-7+17C | 0.60±0.04 | **0.81±0.03** | **0.81±0.02** |
| Waveform-21 | 0.78±0.02 | **0.93±0.01** | **0.94±0.03** |
| Waveform-40 | 0.67±0.03 | **0.96±0.02** | **1±0.00** |
| Cleveland | 0.76±0.03 | **0.84±0.02** | **0.84±0.02** |
| Hungarian | 0.77±0.02 | **0.83±0.01** | **0.85±0.02** |
| Voting | 0.92±0.01 | **0.96±0.01** | **0.97±0.01** |
| Cars | 0.76±0.02 | **0.96±0.02** | **0.97±0.02** |
| TicTacToe | 0.56±0.01 | **1±0.00** | **1±0.00** |
| Glass | 0.68±0.03 | **0.77±0.02** | **0.78±0.03** |
| Hayes | 0.63±0.04 | **0.74±0.05** | **0.72±0.05** |
| Nursery | 0.78±0.02 | **0.96±0.01** | **0.97±0.01** |
| Monk's Problem1 | 0.26±0.02 | **0.27±0.02** | **0.29±0.03** |
| Monk's Problem2 | 0.41±0.03 | **0.76±0.02** | **0.76±0.02** |
| Monk's Problem3 | 0.26±0.02 | **0.27±0.02** | **0.28±0.02** |

Table 4.3: The classification results of 1NN on the test sets after running *Stretch* on the training data sets with two different configurations are presented. On the left column, *Stretch* uses the final transformation matrix, $A_T$ that minimizes the LOOCV error of the training set and on the right, it uses the one, $A_V$ that minimizes the LOOCV error of the validation set. $A_T$ and $A_V$ results are compared and statistically significant results marked with bold for each row of both columns.

| | 100j/400i | | 200j/800i | |
|---|---|---|---|---|
| Dataset | $A_T$ | $A_V$ | $A_T$ | $A_V$ |
| Banded | 0.90±0.07 | **0.95±0.02** | 0.90±0.07 | **0.96±0.02** |
| Sinusoidal | 0.90±0.04 | **0.93±0.02** | 0.91±0.04 | **0.93±0.03** |
| Gauss-band | 0.73±0.06 | **0.75±0.04** | 0.73±0.06 | **0.75±0.04** |
| Parity | 0.85±0.15 | **0.98±0.02** | 0.86±0.15 | **0.98±0.01** |
| LED-7 Display | 0.51±0.05 | 0.51±0.05 | 0.51±0.06 | 0.52±0.05 |
| LED-7+17B | 0.43±0.03 | 0.43±0.02 | 0.42±0.02 | 0.42±0.03 |
| LED-7+17C | 0.59±0.03 | 0.58±0.03 | **0.59±0.03** | 0.57±0.03 |
| Waveform-21 | 0.77±0.03 | 0.76±0.03 | 0.77±0.03 | 0.77±0.03 |
| Waveform-40 | 0.71±0.04 | 0.73±0.04 | 0.72±0.04 | 0.74±0.04 |
| Cleveland | 0.77±0.04 | 0.77±0.05 | 0.78±0.05 | 0.76±0.03 |
| Hungarian | 0.78±0.04 | **0.79±0.03** | 0.77±0.03 | **0.79±0.04** |
| Voting | 0.93±0.02 | 0.94±0.02 | 0.93±0.02 | 0.93±0.02 |
| Cars | 0.83±0.05 | **0.87±0.02** | 0.84±0.06 | 0.87±0.02 |
| TicTacToe | 0.87±0.02 | **0.99±0.01** | 0.84±0.20 | **0.99±0.01** |
| Glass | 0.68±0.05 | 0.66±0.06 | 0.70±0.05 | 0.66±0.05 |
| Hayes | **0.70±0.06** | 0.66±0.05 | **0.70±0.06** | 0.63±0.07 |
| Nursery | 0.86±0.06 | 0.89±0.03 | 0.86±0.07 | 0.89±0.02 |
| Monk's Problem1 | 0.27±0.03 | **0.32±0.03** | 0.27±0.02 | **0.31±0.03** |
| Monk's Problem2 | 0.60±0.06 | **0.73±0.03** | 0.62±0.17 | **0.74±0.02** |
| Monk's Problem3 | 0.27±0.03 | **0.33±0.02** | 0.27±0.02 | **0.32±0.03** |

## 4.4 *Overfitting($A_T$ vs $A_V$)*

As it is described previously, *Stretch* constructs the linear transformations that minimize the LOOCV error of kNN on the training set. However, minimizing the LOOCV error of kNN on the training set might not minimize the error of the kNN on the test set. On the other hand, instead of using $A_T$, one can apply $A_V$ to the test set to minimize the effect of the overfitting while minimizing the error of kNN on the test set.

**$A_T$ vs $A_V$:** Table 4.3 presents the error of 1NN on the test sets that is stretched either with $A_T$ or $A_V$. The results under the label $A_T$ and $A_V$ are 1NN classification accuracies that are calculated by applying $A_T$ and $A_V$ to the test set, respectively. The accuracy improvements of 1NN in the training set can not be consistently observed in Table 4.3. This indicates that the problem is not with *Stretch* being able to find a good linear transformation but that the resulting transformation overfits the training data. The results that are calculated using $A_V$ also supports the overfitting, since the results of $A_V$ outperforms the results of $A_T$(except in the Hayes data set).

For the rest of this chapter, *Stretch* is compared with *Relief-F* based on the $A_V$, since the performance of $A_V$ is obviously better than $A_T$.

## 4.5 Stretch *Experiments*

Table 4.4 represents the accuracy of 1NN on the test set, after applying $A_V$ to the test set. The results are grouped under three labels which are Before, 100j/400i and 200j/800i. The results of the unweighted 1NN is presented under the label "Before" whereas the results for the corresponding configurations are represented in the other columns.

The accuracy of 1NN significantly improved on sixteen *Stretch* applied data sets. *Stretch* significantly reduces the original accuracy of 1NN only on the LED-7+17C data set among these sixteen data sets.

Table 4.4: The classification performance of the 1NN on the test sets before and after applying *Stretch* with two different convergence configurations. Statistically significant results are marked with bold.

| | | Convergence Configuration | |
|---|---|---|---|
| Dataset | Before | 100j/400i | 200j/800i |
| Banded | 0.82±0.03 | **0.95±0.03** | **0.96±0.02** |
| Sinusoidal | 0.87±0.02 | **0.93±0.02** | **0.93±0.03** |
| Gauss-band | 0.67±0.04 | **0.75±0.04** | **0.75±0.04** |
| Parity | 0.66±0.03 | **0.98±0.02** | **0.98±0.01** |
| LED-7 Display | 0.51±0.05 | 0.51±0.05 | 0.52±0.05 |
| LED-7+17B | 0.41±0.01 | **0.43±0.02** | 0.42±0.03 |
| LED-7+17C | 0.60±0.01 | **0.58±0.03** | **0.53±0.03** |
| Waveform-21 | 0.76±0.02 | 0.77±0.03 | 0.77±0.03 |
| Waveform-40 | 0.70±0.03 | **0.73±0.04** | **0.74±0.04** |
| Cleveland | 0.77±0.04 | 0.77±0.05 | 0.76±0.03 |
| Hungarian | 0.77±0.03 | **0.79±0.03** | **0.79±0.04** |
| Voting | 0.92±0.01 | **0.94±0.02** | **0.93±0.02** |
| Cars | 0.77±0.01 | **0.87±0.03** | **0.87±0.02** |
| TicTacToe | 0.56±0.01 | **0.99±0.01** | **0.99±0.01** |
| Glass | 0.68±0.03 | 0.66±0.06 | 0.66±0.05 |
| Hayes | 0.65±0.05 | **0.66±0.05** | 0.63±0.07 |
| Nursery | 0.77±0.03 | **0.89±0.03** | 0.89±0.02 |
| Monk's Problem 1 | 0.27±0.03 | **0.32±0.03** | **0.32±0.03** |
| Monk's Problem 2 | 0.40±0.03 | **0.73±0.03** | **0.74±0.02** |
| Monk's Problem 3 | 0.27±0.04 | **0.33±0.02** | **0.32±0.03** |

The effect of increasing the convergence time is not significant for most of the data sets, nevertheless for some data sets, increasing the convergence time reduces

the 1NN accuracy on the test set even if $A_V$ is used.

## *4.6* Relief-F *Experiments*

These experiments are conducted by running the *Relief-F* method on the training set, then the weights calculated are applied to the test set to improve the classification accuracy of 1NN and BestNN. The experimental results of the method presented in Table 4.5. *Relief-F* reports statistically significant results for 1NN in twelve data sets and for BestNN in seven data sets.

Table 4.5: The classification performance of the 1NN and BestNN before and after applying *Relief-F* are presented. Statistically significant results are marked with bold.

| | 1NN | | BestNN | | |
|---|---|---|---|---|---|
| Dataset | Before | After | Best-k | Before | After |
| Banded | 0.82±0.02 | **0.97±0.01** | 1 | | |
| Sinusoidal | 0.87±0.01 | **0.92±0.02** | 1 | | |
| Gauss-band | 0.67±0.04 | **0.81±0.03** | 1 | | |
| Parity | 0.66±0.03 | **1±0.00** | 1 | | |
| LED-7 Display | 0.51±0.05 | 0.50±0.05 | 11 | 0.72±0.01 | **0.70±0.01** |
| LED-7+17B | 0.41±0.01 | **0.63±0.02** | 11 | 0.50±0.01 | **0.72±0.01** |
| LED-7+17C | 0.60±0.01 | **0.63±0.01** | 11 | 0.70±0.01 | 0.71±0.02 |
| Waveform-21 | 0.76±0.02 | 0.77±0.03 | 11 | 0.81±0.03 | 0.82±0.03 |
| Waveform-40 | 0.70±0.03 | **0.77±0.03** | 11 | 0.75±0.03 | **0.81±0.04** |
| Cleveland | 0.77±0.04 | 0.78±0.03 | 11 | 0.82±0.03 | **0.78±0.05** |
| Hungarian | 0.77±0.03 | 0.78±0.03 | 11 | 0.83±0.03 | **0.81±0.03** |
| Voting | 0.92±0.01 | **0.95±0.02** | 11 | 0.93±0.02 | **0.96±0.01** |
| Cars | 0.77±0.01 | **0.84±0.02** | 1 | | |
| TicTacToe | 0.56±0.01 | **0.76±0.03** | 11 | 0.97±0.03 | **0.80±0.02** |
| Hayes | 0.65±0.05 | **0.77±0.06** | 1 | | |
| Nursery | 0.77±0.03 | **0.90±0.03** | 11 | 0.77±0.03 | **0.87±0.02** |
| Monk's Problem 1 | 0.27±0.03 | 0.27±0.03 | 11 | 0.44±0.02 | **0.40±0.03** |
| Monk's Problem 2 | 0.40±0.03 | **0.44±0.04** | 11 | 0.51±0.03 | **0.53±0.04** |
| Monk's Problem 3 | 0.27±0.04 | 0.27±0.04 | 11 | 0.46±0.02 | **0.39±0.03** |

## *4.7* Relief-F *vs* Stretch

Table 4.7 summarizes the results of Table 4.4 and Table 4.5. This table also provides a statistical comparison of the two methods. Statistically significant results with higher accuracy is marked with bold. In section 4.1, the similarity between *Relief-F* and *Stretch* was discussed. These two methods can not significantly outperformed each other for eight out nineteen data sets. On the eleven data sets, one of these methods significantly outperformed the other one. *Relief-F* outperforms *Stretch* on six out of eleven data sets.

On the parity, gauss-band, waveform40, and LED-7+17B data sets, *Relief-F* outperforms *Stretch* however both algorithms significantly reduce the original error of 1NN on the test sets.

One unexpected result is, *Relief-F* improves the 1NN accuracy of the LED-7+17C and hayes data set significantly while *Stretch* reduces the accuracy of 1NN significantly.

*Stretch* significantly outperforms Relief-f on tic-tac-toe, cars and monk's problem data sets. The tic-tac-toe data set is a relatively interesting case, since *Stretch* can decreases the 1NN error on the test set of tic-tac-toe almost to zero while *Relief-F* decrease the accuracy of 1NN from .80 to .76. All of these data sets are used to test structure discovery and constructive induction methods.

Table 4.6: The classification performance of the 1NN on test sets after applying *Relief-F* and *Stretch*. *Stretch* is run with two different convergence configurations. The results of *Relief-F* and *Stretch* is compared, and statistically significance is tested on the difference between the result of *Relief-F* and *Stretch*. The statistically significant ones are marked with bold. The second column shows which method significantly improves the original 1NN accuracy.

| | | | | Convergence Configuration | |
|---|---|---|---|---|---|
| Dataset | Before | Significant | *Relief-F* | 100j/400i | 200j/800i |
| Banded | 0.82±0.02 | All | 0.97±0.01 | 0.95±0.03 | 0.96±0.02 |
| Sinusoidal | 0.87±0.01 | All | 0.92±0.02 | 0.93±0.02 | 0.93±0.03 |
| Gauss-band | 0.67±0.04 | All | **0.81±0.03** | 0.75±0.04 | 0.75±0.04 |
| Parity | 0.66±0.03 | All | **1±0.00** | 0.98±0.02 | 0.98±0.01 |
| LED-7 Display | 0.51±0.05 | none | 0.50±0.05 | 0.51±0.05 | 0.52±0.05 |
| LED-7+17B | 0.41±0.01 | $R_F$,$S_{100j/400i}$ | **0.63±0.02** | 0.43±0.02 | 0.42±0.03 |
| LED-7+17C | 0.60±0.01 | All | **0.63±0.01** | 0.58±0.03 | 0.53±0.03 |
| Waveform-21 | 0.76±0.02 | None | 0.77±0.03 | 0.77±0.03 | 0.77±0.03 |
| Waveform-40 | 0.70±0.03 | All | **0.77±0.03** | 0.73±0.04 | 0.74±0.04 |
| Cleveland | 0.77±0.04 | None | 0.78±0.03 | 0.77±0.05 | 0.76±0.03 |
| Hungarian | 0.77±0.03 | $S_{100/400}$, $S_{200/800}$ | 0.78±0.03 | 0.79±0.03 | 0.79±0.04 |
| Voting | 0.92±0.01 | All | 0.95±0.02 | 0.94±0.02 | 0.93±0.02 |
| Cars | 0.77±0.01 | All | 0.84±0.02 | **0.87±0.03** | **0.87±0.02** |
| TicTacToe | 0.80±0.03 | All | 0.76±0.03 | **0.99±0.01** | **0.99±0.01** |
| Hayes | 0.65±0.05 | $R_F$,$S_{200/800}$ | **0.77±0.07** | 0.66±0.05 | 0.63±0.07 |
| Nursery | 0.77±0.03 | All | 0.90±0.03 | 0.89±0.03 | 0.89±0.02 |
| Monk's Problem1 | 0.27±0.03 | $S_{100/400}$, $S_{200/800}$ | 0.27±0.03 | **0.32±0.03** | **0.32±0.03** |
| Monk's Problem2 | 0.40±0.03 | All | 0.44±0.04 | **0.73±0.03** | **0.74±0.02** |
| Monk's Problem3 | 0.27±0.04 | $S_{100/400}$, $S_{200/800}$ | 0.27±0.04 | **0.33±0.02** | **0.32±0.03** |

### *4.8 BestNN Performance of Feature Weighting Methods*

***Stretch* on the training set:** Table 4.7 presents the LOOCV accuracy of BestNN on the *Stretch* applied training set. The LOOCV error of BestNN on training data set also decreases as the convergence time increases. The error is reduced significantly for all of the data sets.

Table 4.7: The LOOCV error the BestNN on the training set before and after applying *Stretch* with two different convergence configurations are presented. Statistically significant results are marked with bold.

| | | Convergence Configuration | |
|---|---|---|---|
| Dataset | Before | 100j/400i | 200j/800i |
| LED-7 Display | 0.72±0.04 | **0.75±0.03** | **0.76±0.03** |
| LED-7+17B | 0.50±0.03 | **0.65±0.03** | **0.69±0.03** |
| LED-7+17C | 0.68±0.03 | **0.77±0.03** | **0.76±0.04** |
| Waveform-21 | 0.82±0.02 | **0.87±0.01** | **0.90±0.01** |
| Waveform-40 | 0.76±0.02 | **0.85±0.01** | **0.85±0.02** |
| Cleveland | 0.81±0.02 | **0.85±0.01** | **0.85±0.01** |
| Hungarian | 0.82±0.02 | **0.85±0.01** | **0.84±0.01** |
| Voting | 0.93±0.01 | **0.95±0.01** | **0.95±0.01** |
| TicTacToe | 0.97±0.05 | **0.99±0.00** | **0.99±0.00** |

**$A_T$ vs $A_V$:** Table 4.8 presents the error of BestNN on the data sets that is stretched either with $A_T$ or $A_V$. The results of both linear transformations are not significantly different for all of the data sets, except tic-tac-toe data set. As a result, the results of BestNN do not distinguish one of the transformation matrix from the other. Although there is no difference between the matrices, $A_V$ is used in the rest of the experiments.

Table 4.8: The classification results of BestNN on the test sets after running *Stretch* on the training sets with two different configurations. On the left column, *Stretch* uses the final transformation matrix, $A_T$ that minimizes the LOOCV error of the training set and on the right, it uses, $A_V$ that minimizes the LOOCV error of the validation set. Statistically significant differences are marked with bold for each row of both columns.

| | 100j/400i | | 200j/800i | |
|---|---|---|---|---|
| Dataset | $A_T$ | $A_V$ | $A_T$ | $A_V$ |
| LED-7 Display | 0.73±0.01 | 0.73±0.01 | 0.73±0.01 | 0.73±0.01 |
| LED-7+17B | 0.53±0.03 | 0.52±0.02 | 0.53±0.03 | 0.51±0.03 |
| LED-7+17C | **0.70±0.02** | 0.68±0.02 | **0.70±0.01** | 0.67±0.02 |
| Waveform-21 | 0.82±0.03 | 0.80±0.04 | 0.81±0.03 | 0.81±0.03 |
| Waveform-40 | 0.78±0.03 | 0.77±0.03 | 0.77±0.03 | 0.76±0.04 |
| Cleveland | 0.83±0.03 | 0.83±0.03 | 0.82±0.03 | 0.84±0.03 |
| Hungarian | 0.83±0.03 | 0.82±0.03 | 0.82±0.03 | 0.82±0.03 |
| Voting | 0.94±0.02 | 0.94±0.02 | 0.94±0.02 | 0.94±0.03 |
| TicTacToe | **0.98±0.00** | 0.93±0.01 | **0.98±0.00** | 0.93±0.01 |

***Stretch* Experiments:** Table 4.9 presents the accuracy of BestNN on the test set after applying $A_V$ to the training and the test set. *Stretch* can not improve the error of BestNN on any of the data sets, except the Cleveland data set. It significantly increases the error of tic-tac-toe, waveform21 and LED7-7+17C.

*Stretch* uses the missclassified instances to construct linear transformations. Therefore, linear transformations that are constructed using the boundary instances improves the accuracy of the kNN. On the other hand, if the linear transformations are constructed using noisy instances then the effect of these transformations on the accuracy of kNN is unpredictable.

The misclassified instance set of 1NN consists of the boundary instances and noisy instances. As the k becomes larger the boundary instances are removed from the

misclassified instance set meanwhile noisy ones stays in the set. Therefore, as $k$ becomes larger the probability of a linear transformation is constructed using a noisy instance increases. As a result, it can be concluded that if *Stretch* is unable to improve the accuracy of kNN($k > 1$) for a particular data set, it is mainly due to the noise in the data set. The BestNN experiments also supports this claim, since five out of nine data sets that are presented in Table 4.9 have ten percent of noise.

Table 4.9: The classification performance of the BestNN on test sets before and after applying *Stretch* with two different convergence configurations. Statistically significant results are marked with bold.

| | | Convergence Configuration | |
|---|---|---|---|
| Dataset | Before | 100j/400i | 200j/800i |
| LED-7 Display | 0.73±0.01 | 0.73±0.01 | 0.73±0.01 |
| LED-7+17B | 0.50±0.01 | 0.52±0.02 | 0.51±0.03 |
| LED-7+17C | 0.70±0.01 | **0.68±0.02** | **0.67±0.02** |
| Waveform-21 | 0.81±0.03 | 0.81±0.04 | **0.78±0.04** |
| Waveform-40 | 0.75±0.03 | 0.76±0.03 | 0.76±0.04 |
| Cleveland | 0.82±0.03 | 0.82±0.03 | **0.84±0.03** |
| Hungarian | 0.82±0.02 | 0.82±0.03 | 0.82±0.03 |
| Voting | 0.93±0.02 | 0.94±0.02 | 0.94±0.03 |
| TicTacToe | 0.97±0.03 | **0.93±0.02** | **0.93±0.01** |

***Relief-F* vs *Stretch*:** *Relief-F* is robust to noise [4] therefore it is able to improve the accuracy of BestNN on the five out of nine data sets and outperforms *Stretch*. Table 4.5 represents the results of BestNN after applying *Relief-F*.

### *4.9 Effect of Noise on* Stretch

In order to observe the effect of noise on the accuracy of the *Stretch*, we add ten percent noise to the four artificially created data sets. These data sets are identical with the ones that are used in the previous experiments, except the noise. These four data sets are banded, sinus, gauss and parity. Moreover a noise free LED7+17b is generated. Noise free versions of the other LED data sets have a very high classification accuracy without feature weighting therefore they are not included in this experiment.

On the four new noisy artificial data sets, *Stretch* significantly increases the performance of 1NN, however it can not improve it as much as it improves the none noisy case with the same configurations. Moreover, the improvements in the 1NN accuracy of the noise free versions of these data sets are significantly better than the results presented in Table 4.10. *Stretch* also performs significantly better on noise-free-LED-7+17b data set compared to LED-7+17b.

Table 4.10: The classification performance of the 1NN on test sets before and after applying *Stretch* with two different convergence configurations. Statistically significant results are marked with bold.

| | | Convergence Configuration | |
|---|---|---|---|
| Dataset | Before | 100j/400i | 200j/800i |
| noisy-banded | 0.67±0.01 | **0.71±0.01** | **0.71±0.01** |
| noisy-parity | 0.59±0.01 | **0.80±0.01** | **0.80±0.01** |
| noisy-gauss | 0.52±0.01 | **0.54±0.01** | **0.56±0.01** |
| noisy-sinus | 0.77±0.00 | **0.76±0.00** | **0.76±0.00** |
| noise-free-LED-7+17b | 0.65±0.03 | **0.78±0.02** | **0.78±0.02** |

# Chapter 5

# CONCLUSION AND FUTURE WORK

## *5.1 Conclusion*

In the first part of this thesis an overview of kNN and its limitations are presented. The solutions to these limitations are explained. Since the main focus of this thesis is the feature weighting, we discussed the effects of irrelevant features on kNN in more detail. The work of the past 20 years is surveyed. All of the methods that are surveyed have their own advantages and disadvantages thus none of them is successful on all data sets.

The feature weighting methods construct a diagonal weight matrix which does not have information about the inter-relations of the features. Although this diagonal matrix improves the classification accuracy of kNN on some data sets, it may not be able to improve the performance of the kNN when the features are inter-related.

Considering this, in the second part of the thesis, we introduced a new feature weighting method which can construct an arbitrary feature weighting matrix. This method iteratively constructs linear transformations to reduce the classification error of kNN. As a result, it constructs a full weighting matrix which has the information about the inter-relations of the features. Since this method can construct any weight matrix, any diagonal weight matrix is in the solution subset of this method.

This method compared with Relief-F[8] which is one of the best methods in [1]. Our method *Stretch*, outperforms Relief-F on 5 out of 19 data sets while Relief-F outperforms *Stretch* on 6 out of 19 data sets. Stretch has a dominance on the data sets that are used to test structure discovery and constructive induction methods. *Stretch* outperforms Relief-F on 5 out of 6 of these data sets.

One of the deficiencies of *Stretch* is that noise of the data set reduces the performance of the algorithm. Therefore on noisy data sets especially when $k > 1$NN is used as a classifier *Stretch* can not construct a linear transformation that significantly improves the accuracy of the $k > 1$NN.

Another weakness of *Stretch* compared to Relief-F is, Stretch can not construct a linear transformation based on the $k$ nearest neighbors of a given instance. Although $M$ is constructed by kNN the stretch matrix is constructed only using the one nearest neighbor. As a result, it can not significantly increase the accuracy of $k > 1$NN when $k$ is set the optimum value for the data set.

## *5.2 Future Work*

One important improvement on *Stretch* is, the ability to construct linear transformations based on the $k$ nearest neighborss. Such linear transformations would be less effected by noisy instances. Moreover they would increase the chance of improvement in the accuracy of $k > 1$NN.

The next important topic that needs further observation is the types of data set structures that are more suitable to apply *Stretch*. Therefore, the effect of $k$ value, types of the features, number of categories and the effect of the noise could be examined more clearly.

Another potential improvement is, the wrapper model could be replaced with the filter model to decrease the computational complexity of *Stretch*. In order to increase the robustness of *Stretch* to noise instead of using $M$, *Stretch* can use all the instances in the data set.

Finally, the effect of *Stretch* on other learning algorithms could be examined.

# BIBLIOGRAPHY


[1] D. Wettschereck, D. W. Aha, and T. Mohri, "A review and comparative evaluation of feature weighting methods for lazy learning algorithms.," *Artificial Intelligence Review*, , no. 11, pp. 273–314, 1997.

[2] B. V. Dasarathy, Ed., *Nearest neighbor (NN) norms: NN pattern classification techniques*, IEEE Computer Society Press, Los Alamitos,CA, 1991.

[3] K. Kira and L.A. Rendell, "A practical approach to feature selection.," in *Proceedings of the Ninth International Conference on Machine Learning.*, Aberdeen, Scotland, 1992, pp. 249–256, Morgan Kaufmann.

[4] I. Kononenko, "Estimating attributes: Analysis and extensions of relief.," in *Proceedings of the 1994 European Conference on Machine Learning.*, Catania,Italy, 1994, pp. 171–182, Springer Verlag.

[5] D. Lowe, "Similarity metric learning for a variable-kernel classifier.," *Neural Computation*, , no. 7, pp. 72–85, 1995.

[6] R.H. Creecy, B. M. Masand, S. J. Smith, and D. L. Waltz, "Trading mips and memory for knowledge engineering.," *Communications of the ACM*, , no. 35, pp. 48–64, 1992.

[7] C. Stanfill and D. Waltz, "Toward memory-based reasoning," *Communications of the Association for Computing Machinery*, , no. 29, pp. 1213–1228, 1986.

[8] M. Robnik-Šikonja and I. Kononenko, "Theoretical and Empirical Analysis of ReliefF and RReliefF," *Machine Learning*, vol. 53, no. 1, pp. 23–69, 2003.

[9] E. Parzen, "Nonparametric Statistical Data Modeling," *Journal of the American Statistical Association*, vol. 74, no. 365, pp. 105–121, 1979.

[10] T. Cover and P. Hart, "Nearest neighbor pattern classification," *Information Theory, IEEE Transactions on*, vol. 13, no. 1, pp. 21–27, 1967.

[11] T. Wagner, "Convergence of the nearest neighbor rule," *Information Theory, IEEE Transactions on*, vol. 17, no. 5, pp. 566–571, 1971.

[12] B.D. Ripley, *Pattern Recognition and Neural Networks*, Cambridge University Press, 1996.

[13] S.A. Dudani, "The distance-weighted k-nearest neighbor rule," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 6, no. 4, pp. 325–327, 1976.

[14] JES Macleod, A. Luk, and DM Titterington, "A re-examination of the distance-weighted k-nearest neighbor classification role.," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 17, no. 4, pp. 689–696, 1987.

[15] P.D. Wasserman, *Advanced Methods in Neural Computing*, John Wiley & Sons, Inc. New York, NY, USA, 1993.

[16] DF Spetch, "Probabilistic neural network," *J Neural Networks*, vol. 3, pp. 109–18, 1990.

[17] Peter N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1993.

[18] D.R. Wilson and T.R. Martinez, "Reduction Techniques for Instance-Based Learning Algorithms," *Machine Learning*, vol. 38, no. 3, pp. 257–286, 2000.

[19] R.F. Sproull, "Refinements to nearest-neighbor searching ink-dimensional trees," *Algorithmica*, vol. 6, no. 1, pp. 579–589, 1991.

[20] C.H. Papadimitriou and J.L. Bentley, *A Worst-Case Analysis of Nearest Neighbor Searching by Projection*, Springer-Verlag London, UK, 1980.

[21] K.I. Lin and C. Yang, "The ANN-tree: an index for efficient approximate nearest neighborsearch," *Database Systems for Advanced Applications, 2001. Proceedings. Seventh International Conference on*, pp. 174–181, 2001.

[22] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, pp. 322–331, 1990.

[23] G.H. John, R. Kohavi, and K. Pfleger, "Irrelevant features and the subset selection problem," *Proceedings of the Eleventh International Conference on Machine Learning*, vol. 129, 1994.

[24] R.E. Bellman, *Adaptive Control Processes: A Guided Tour*, Princeton University Press, 1961.

[25] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, pp. 81–106, 1986.

[26] D.E. Rumelhart and J.L. McClelland, *Parallel distributed processing: explorations in the microstructure of cognition, vol. 2: psychological and biological models*, MIT Press Cambridge, MA, USA, 1986.

[27] V.N. Vapnik, "Estimation of Dependences Based on Empirical Data," *New York*, 1982.

[28] P. Domingos, "Rule Induction and Instance-Based Learning: A Unified Approach," *Proc. of IJCAI*, vol. 95, pp. 1226–1232, 1995.

[29] S. Das, "Filters, wrappers and a boosting-based hybrid for feature selection," *Proc. ICML*, 2001.

[30] D. Wettschereck, "A description of the mutual information approach and the variable similarity metric.," Tech. Rep., German National Research Center for Computer Science,Artificial Intelligence Research Division, Sankt Augustin,Germany, 1995a.

[31] P. Langley and S. Sage, *Induction of Selective Bayesian Classifiers*, Morgan Kaufmann, 1994.

[32] D.W. Aha and R.L. Bankert, "Feature selection for case-based classification of cloud types: An empirical comparison," *Proceedings of the 1994 AAAI Workshop on Case-Based Reasoning*, pp. 106–112, 1994.

[33] J. Doak, *An Evaluation of Feature Selection Methods and Their Application to Computer Security*, University of California, 1992.

[34] B. Raman and T.R. Ioerger, "Enhancing learning using feature and example selection," *Journal of Machine Learning Research (submitted for publication)*, 2003.

[35] R. Kohavi, P. Langley, and Y. Yun, "The utility of feature weighting in nearest-neighbor algorithms," *Proceedings of the Ninth European Conference on Machine Learning. Prague: Springer-Verlag*, 1997.

[36] T. Mohri and H. Tanaka, "An optimal weighting criterion of case indexing for both numeric and symbolic attributes," *Case-Based Reasoning: Papers from the 1994 Workshop. AAAI Press, Menlo Park, CA*, 1994.

[37] D.W. Aha and R.L. Goldstone, "Concept learning and flexible weighting.," in *Proceedings of the Fourteenth Annul Conference of the Cognitive Science Society*, Bloomington, IN, 1992, pp. 534–539, Lawrence Erlbaum.

[38] W. Daelemans and A. van den Bosch, "Generalization performance of back-propagation learning on a syllabification task," *Proceedings of the 3rd Twente Workshop on Language Technology*, 1992.

[39] T.M. Cover and J.A. Thomas, "Elements of Information Theory," *New York*, 1991.

[40] W. Daelemans, *Learnability and Markedness in Data-driven Acquisition of Stress*, Tilburg University, Institute for Language Technology and Artificial Intelligence, 1993.

[41] M.A. Hall, *Correlation-based Feature Selection for Machine Learning*, Ph.D. thesis, The University of Waikato, 1999.

[42] S. Salzberg, "A nearest hyperrectangle learning method," *Machine Learning*, vol. 6, no. 3, pp. 251–276, 1991.

[43] D.W. Aha, "Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms," *International Journal of Man-Machine Studies*, vol. 36, no. 2, pp. 267–287, 1992.

[44] D.B. Skalak, "Prototype and feature selection by sampling and random mutation hill climbing algorithms," *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 293–301, 1994.

[45] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz, "UCI repository of machine learning databases," 1998, http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[46] D. Wettschereck, *A study of distance-based machine learning algorithms*, Ph.D. thesis, Oregon State University, 1994.

[47] M. Bohanec and V. Rajkovic, "Knowledge acquisition and explanation for multiattribute decision making," *8th Intl Workshop on Expert Systems and their Applications*, pp. 59–78, 1988.

[48] S. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Dzeroski, SE Fahlman, D. Fisher, et al., "The MONK's Problems: A Performance Comparison of Different Learning Algorithms," 1991.