OPTIMAL ATTRIBUTE WEIGHTING IN A NEAREST NEIGHBOR CLASSIFIER

by

JUGAL RAJU SHETH

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2016

Acknowledgements

I would like to express my sincere gratitude to my advising professor Dr. Michael T Manry for his continuous support throughout my master study and related research, for his patience, guidance and immense knowledge. His guidance helped me in all the time of research and writing of this thesis.

I would also like to thank Dr. Ionnis D. Schizas and Dr. W. Alan Davis for their time, valuable comments and being a member of my thesis defense committee.

Finally, I must express my sincere gratitude to my family for their love and support. I dedicate this thesis to my parents, Mr. Raju Sheth and Mrs. Asmita R. Sheth and my sister, Arpita R. Sheth.

November 21, 2016

Abstract

OPTIMAL ATTRIBUTE WEIGHTING IN A

NEAREST NEIGHBOR

CLASSIFIER

JUGAL RAJU SHETH, MS

The University of Texas at Arlington, 2016

Supervising Professor: Dr. Michael T. Manry

Nearest Neighbor algorithms are non-parametric algorithms that use distance measure techniques for classification and regressions. This thesis, first improves the traditional nearest neighbor classifier by optimizing its distance measure using a second order training algorithm. It then presents a second order method to adjust center vectors. It is shown that the distance measure weight optimization and center vector optimization, individually and together reduce the final classification error. The testing error of our algorithm is shown to be less than that of LVQ V.2.1

List of Illustrations

List of Tables

Chapter 1

INTRODUCTION

This chapter is a review of basic concepts of the classifiers. It also reviews nearest neighbor classifiers and their properties.

1.1 Classifiers

In the fields of statistics and pattern recognition, classification is the task of identifying the class or the category of a new test pattern, on the basis of a training set of data containing similar patterns whose class membership is known [32]. As an example, tasks like assigning an email into spam or non-spam, classifying an image of an animal as a dog or a cat etc. Algorithms that implement classification are known as classifiers. Depending on the characteristics of the data to be classified, different kinds of classifiers are used. Linear and generalized linear classifiers like the perceptron [33], support vector machines [34] etc. are used to classify simple datasets. For complex datasets non-linear classifiers like multi-layer perceptron [35], decision trees [36] etc. are used. All of these classifiers have parameters like weights that need to be adjusted differently depending on the training dataset. Thus these classifiers are called as parametric classifiers.

1.2 Nearest Neighbor Classifier

Classifiers based on the nearest neighbor (NN) rule are traditional, simple and effectively used in pattern recognition [13, 14], text categorization [15-17], ranking models [18], object recognition [20] and event recognition [19]. They achieve consistently high performance without priori assumptions about the distributions from which the training examples are drawn. Nearest neighbor classifiers are a type of instance based learning or lazy learning classifiers, in which the generalization beyond the training data is delayed

until a new query is encountered. NN classifiers are thus consistent non parametric estimators. Here the term non-parametric refers to the fact that there is no prior knowledge of the statistical distribution of the data. During testing, the distance of the input vector (pattern) to each cluster's center vector is calculated. The estimated class of the input vector is that of the nearest center vector. As the number of training patterns tend to infinity, classifiers based on nearest neighbor rule converge to the corresponding Bayes estimate.

1.3 Objective of this thesis

This thesis, proposes an algorithm for optimal attribute weighting in a nearest neighbor classifier (OAWNCC) that optimizes the weights in a nearest neighbor classifier's distance measure using a second order training algorithm. The optimized weights of important input features are greater in magnitude than those of less important input features. This enables us to neglect features by using weights having small absolute magnitude. It also proposes a center vector optimization algorithm that is a technique for optimizing the center vector locations in input dimension space which directly minimizes the input-output mapping error.

Chapter 2, reviews the structure and operations of the traditional nearest neighbor classifiers from which OAWNCC is derived. Chapter 3, explains the motivation behind the designing of OAWNCC and describes training using steepest descent technique with an optimal learning factor and then eventually with Newton's algorithm for faster convergence. Chapter 4, introduces optimization of center vectors in OAWNCC using the steepest descent method and with second order algorithms for calculating multiple optimal learning factors. Chapter 5 present the results on several datasets.

Chapter 2

STRUCTURE AND OPERATION OF NNC

This chapter, reviews the structure and operation of the traditional nearest neighbor classifier. It also reviews problems associated with the nearest neighbor classifiers.

2.1 Structure of Nearest Neighbor Classifier

The training data consists of 'K' N-dimensional cluster center vectors, $\boldsymbol{m}_{ik}$, where $\boldsymbol{m}_{ik}$ is the center vector of the k^{th} cluster of the i^{th} class. Let the total number of classes be N_c. The p^{th} N-dimensional test vector is denoted by $\boldsymbol{x}_p$. The symbol d_i is the distance between $\boldsymbol{x}_p$ and the closest center vector of the i^{th} class.

Since the nearest neighbor classifier is an instance based classifier [42], unlike many other artificial learners, they do not extract any information from the training data during the learning phase. The learning phase is merely a question of encapsulating the training data. During the time of classification, an unlabeled test pattern is classified by assigning the class of the center vector that is the closest to this new test pattern. The distance metric can be empirically chosen among the Euclidean, Minkowski [44], and Mahalanobis [23] methods among others, based on the training data and application. The most commonly used distance metric for continuous variables is the Euclidean distance (L_2)

2.2 Basic Nearest Neighbor Classifier operation

In the N_v, N - dimensional training patterns, each pattern is associated with the class label to which it belongs. There are two stages of operations involved during classification using nearest neighbor classifiers.

1) Clustering: N_v training patterns of N-dimensional input space is divided into K clusters. The value of K is chosen such that patterns in the same clusters are more similar to each other than to those in other clusters. This division of input patterns into K cluster's center vectors can be done by implementing clustering algorithms such as K – Means [3], SOM [2] or EM [42].
2) Classification: Once the training patterns are divided into clusters, their center vectors and membership to a particular class $\boldsymbol{m}_{ik}$ are cached into disk. When a new test input vector needs to be classified, its distance from all the center vectors is computed. The closest center vector from each class is determined as

$$\mathrm{d_i} = \underset{\mathrm{k}}{\mathrm{argmin}}\, \mathrm{d}(\mathbf{x}_\mathrm{p}, \mathbf{m}_\mathrm{ik})$$

where, $\mathrm{d}(\mathbf{x}_\mathrm{p}, \mathbf{m}_\mathrm{ik})$, is the distance of the p^{th} test pattern, $\boldsymbol{x}_p$, from the center vector, $\boldsymbol{m}_{ik.}$ d_i is the distance of the new test pattern from the closest center vector of i^{th} class. The class membership of this new test pattern is then estimated as

$$\mathrm{i_c'} = \underset{\mathrm{i}}{\mathrm{argmin}}\, \mathrm{d_i}$$

where, i_c' is the estimated class of the p^{th} test pattern, $\boldsymbol{x}_p$.

2.3 Generating center vectors

Clustering is the task of grouping a set of objects such that objects in the same group are more similar to each other in some sense than to those in other groups. It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, bioinformatics, data compression, and computer graphics

[36]. Center vectors can be generated using various algorithms like K – Means [3], SOM [2] and EM [42]. The appropriate center vector generating algorithm depends on the individual dataset and the intended use of the results. The training dataset can be used in different ways for classification using nearest neighbor classifiers. For example,

1) Entire training dataset or patterns can be used as center vectors. This avoids the need for using any clustering algorithms. The number of training patterns can be large, even in the order of millions. Thus this method can cause huge memory strains as the entire dataset needs to be cached in memory. Along with this, computing distance of a test pattern from all these training patterns can be computationally very expensive. However, the above method can prove to be useful if the number of training patterns are limited and testing time is not a matter of concern.
2) We can randomly choose few training patterns and use them as center vectors. This method also avoids the need for using any clustering algorithms and does not cause much memory strains as only center vectors need to be cached in memory as compared to the entire training dataset. Number of center vectors is much less than the number of training patterns. Since these randomly picked center vectors are not optimal, they may group dissimilar objects to each other and might even prove to be use less center vectors. This can lead to decreased performance.
3) K - Means clustering [3] is a popular vector quantization method that partitions N_v patterns into K clusters where each pattern belongs to the cluster with the nearest mean. This results in partitioning of the data space into Voronoi cells [4]. K - Means clustering is an iterative refinement, efficient heuristic algorithm that is commonly employed and converge quickly to local optimums [38].

Given a set of N_v training patterns, $\boldsymbol{x}_p$, where each pattern has N-dimensional inputs. K - Means clustering aims to partition N_v patterns into K clusters, so as to minimize the within-cluster sum of squares distances.

Algorithm Summary [31]

1. i_t = 0, where i_t = iteration number and N_{it} = total number of iterations.
2. $i_t = i_t + 1$
3. Calculate center vector, $\boldsymbol{m}_{k,}$ as
4. $\mathbf{m}_{\mathrm{k}} = \frac{1}{\mathrm{N_v(k)}} \sum_{\mathrm{p:m(p)=k}} \mathbf{x}_{\mathrm{p}}$,

   where $m(p)$ equals the cluster number of the p^{th} pattern and

   $N_v(k)$ equals the number of patterns in the k^{th} .

   Reclassify $\boldsymbol{x}_p$s, in one data pass. If $\boldsymbol{x}_p$ belongs to the k^{th} cluster, $m(p)$ equals k. $m(p)$ therefore specifies the cluster membership of the p^{th} pattern. If any clusters change and iteration number < N_{it}, go to step 2.

The K – Means clustering error, $E_{k-means}$, is

$$\mathrm{E_{k-means}} = \frac{1}{\mathrm{N_v}} \sum_{\mathrm{p=1}}^{\mathrm{N_v}} \mathrm{d}\left(\mathbf{x}_{\mathrm{p}}, \mathbf{m}_{\mathrm{m(p)}}\right) = \frac{1}{\mathrm{N_v}} \sum_{\mathrm{k=1}}^{K} \mathrm{E_k}$$

$$\mathrm{E_{ik}} = \sum_{\mathrm{p:m(p)=k}} \mathrm{d}\left(\mathbf{x}_{\mathrm{p}}, \mathbf{m}_{\mathrm{k}}\right)$$

2.4 Example Distance Measure Techniques

Nearest neighbor algorithms calculates the distance between the new test pattern and the center vectors to estimate the class of this new test pattern. There are a variety of distance measures available.

2.4.1 Euclidean Distance

Euclidean distance measure is by far the most common distance measure technique used. The associated norm is called the Euclidean norm [38]. Euclidean distance between the p^{th} test pattern, $\boldsymbol{x}_p$ and $\boldsymbol{m}_{ik}$ center vector is given by

$$\mathrm{d}(\mathbf{x}_\mathrm{p}, \mathbf{m}_\mathrm{ik}) = \| \mathbf{x}_\mathrm{p} - \mathbf{m}_\mathrm{ik} \| = \left(\sum_{\mathrm{n}=1}^{\mathrm{N}} \left(\mathrm{x}_\mathrm{p}(\mathrm{n}) - \mathrm{m}_\mathrm{ik}(\mathrm{n}) \right)^2 \right)^{½} \tag{2.1}$$

The square root is often not computed in practice, because the closest center vector will still be the closest, regardless of whether or not the square root is taken [40].

2.4.2 Mahalanobis Distance Measure [23]

Mahalanobis distance measure computes the distance between p_{th} test pattern, $\boldsymbol{x}_p$ and the distribution, D. It is a multi-dimensional dimensional generalization of the idea of measuring how many standard deviations point $\boldsymbol{x}_p$ is away from the mean. If the point is at the mean of distribution, D, the distance is zero [41]. Mahalanobis distance measure can also be defined as the measure of dissimilarity between the test pattern, $\boldsymbol{x}_p$, and center vector, $\boldsymbol{m}_{ik}$, of the same distribution [31].

$$\mathrm{d}(\mathbf{x}_\mathrm{p}, \mathbf{m}_\mathrm{ik}) = \sum_{\mathrm{n}=1}^{\mathrm{N}} \sum_{m=1}^{N} a_k(n,m)\left[x_p(n) - m_{ik}(n)\right]\left[x_p(m) - m_{ik}(m)\right] \tag{2.2}$$

where,

$$\mathrm{a}_\mathrm{k}(n,m) \in \boldsymbol{C}_k^{-1}, \quad \boldsymbol{C}_k = E[(\boldsymbol{x}_p - \boldsymbol{m}_{ik})(\boldsymbol{x}_p - \boldsymbol{m}_{ik})^T]$$

2.4.3 Minkowski Distance Measure [44]

It is a generalization of Euclidean distance and the Manhattan distance. The distance of order v between the p^{th} test pattern, $\boldsymbol{x}_p$ and $\boldsymbol{m}_{ik}$ center vector is given by

$$\mathrm{d}(\mathbf{x}_\mathrm{p}, \mathbf{m}_\mathrm{ik}) = \left(\sum_{\mathrm{n}=1}^{\mathrm{N}} \left(\mathrm{x_p(n)} - \mathrm{m_{ik}(n)}\right)^{\mathrm{v}}\right)^{\frac{1}{\mathrm{v}}} \tag{2.3}$$

If $v = 2$, the Minkowski distance is equivalent to Euclidean distance (L_2).

2.4.4 Weighted Euclidean Distance

It is a modified Euclidean distance that incorporates weights in the distance calculation such that distance measure for each input element $x_p(n)$ is multiplied with the corresponding input weights element $w(n)$. The weighted Euclidean distance between the p^{th} test pattern, $\boldsymbol{x}_p$ and center vector, $\boldsymbol{m}_{ik}$, is given by

$$\mathrm{d}(\mathbf{x}_\mathrm{p}, \mathbf{m_{ik}}) = \sum_{\mathrm{n}=1}^{\mathrm{N}} \left(\mathrm{w(n)} \cdot \left(\mathrm{x_p(n)} - \mathrm{m_{ik}(n)}\right)\right)^2 \tag{2.4}$$

This thesis uses weighted Euclidean distance measure to compute the distance between the test pattern and the center vectors as the weights are a measure of importance for the corresponding input element. Optimally tuning these weights such that they are small in magnitude for noisy and less discriminative features will reduce the participation of these features in computing the distance for nearest neighbors. Thus this helps in solving the problem of misclassification due to noisy features.

Weight Initialization

Weights are one of the most important parameters that determine the performance of any classifier. The training error convergence, performance and training hyper-parameters like learning rates etc. depend heavily on initial weights. If the weights are too

small the gradients would in turn be very small in magnitude and thus the classifier will take more time to converge or might not converge to a desired error value. On the other hand if weights are very large then their gradients would be very large in magnitude too. So a small change in weight update can lead to large change in the output, thus a very small learning rate would be required to compensate for this problem. Later, a small learning rate will need more iterations to converge weights with small magnitudes, since their gradients are also small. If some inputs have much larger variance than others, they can dominate the training. So to avoid dominance of some high variance inputs over others, inputs are normalized by initializing weights as $w(n) = \frac{1}{Var(x(n)) + \epsilon}$, where ϵ is a small positive constant of order 10^{-3} used to avoid division by zero.

2.5 Theoretical Properties of Nearest Neighbor Classifier

The Bayes error is the minimum achievable error rate by any classifier. Incase if the classes overlap then the error rate will be nonzero. For example, suppose that the training input pattern, with the correct class label of that pattern, follows a Gaussian distribution with mean μ_i and fixed variance. The two Gaussians overlap so no classifier can predict the class label correctly for all training patterns, and the Bayes error rate is nonzero.

The Bayes error rate is the average over the space of all examples of the minimum error probability for each example. The optimal prediction for any test pattern $\boldsymbol{x}$ is the label that has highest probability given $\boldsymbol{x}$. The error probability for this example is then one minus the probability of this label. Formally, the Bayes probability of error rate is

$$\mathrm{P}_{\mathrm{e-Bayes}} = \int_{\boldsymbol{x}} \mathrm{p}(\mathbf{x})[1 - \max_{\mathrm{i}} \mathrm{p}(\mathrm{i}|\,\mathbf{x})] \tag{1.1}$$

where $p(i|\boldsymbol{x})$ is the probability that $\boldsymbol{x}$ has label i and $1 - p(i|\boldsymbol{x})$ is the probability that $\boldsymbol{x}$ has a different label**.** The maximum is taken over the N_c possible labels $i = 1$ to $i = N_c$ [30].

Theorem: When the number of training examples tend to infinity, the probability of error rate of NNC is at worst twice the Bayes error rate as proven by [1].

Proof: Let $\boldsymbol{x}$ be a test pattern and $\boldsymbol{m}_{ik}$ be its closest neighbor. If the number of training examples N_v is large, then the probability distribution for any test pattern and its nearest neighbor will be essentially the same. In this case, for the p^{th} test pattern, $\boldsymbol{x}$, the expected probability of error rate of NNC is

$$P_{e-NNC} = \sum_{\mathrm{i}=1}^{\mathrm{N_c}} \mathrm{p(i|x)[1 - p(i|x)]} \tag{1.2}$$

To prove the theorem, we need to show that

$$\sum_{\mathrm{i}=1}^{\mathrm{N_c}} \mathrm{p(i|x)[1 - p(i|x)]} \leq 2[1 - \max_{\mathrm{i}} \mathrm{p(i|x)}] \tag{1.3}$$

$$\text{i.e} \qquad \mathrm{P_{e-NNC}} \leq 2 \cdot \mathrm{P_{e-Bayes}}$$

Let $\max_i p(i|\boldsymbol{x}) = r$ and let the maximum be attained with $i = j$. Then the left hand side is

$$\sum_{\mathrm{i}=1}^{\mathrm{N_c}} \mathrm{p(i|x)[1 - p(i|x)]} = r(1 - r) + \sum_{\mathrm{i}\neq\mathrm{j}} \mathrm{p(i|x)[1 - p(i|x)]} \tag{1.4}$$

and the right hand side is $2(1 - r)$. The summation above is maximized when all the values $p(i|\boldsymbol{x})$ are equal for $i \neq j$. The value of left hand side is then

$$A = \mathrm{r(1 - r)} + (\mathrm{N_c} - 1)\frac{1 - \mathrm{r}}{\mathrm{N_c} - 1}\frac{(\mathrm{N_c} - 1) - (1 - \mathrm{r})}{\mathrm{N_c} - 1} \tag{1.5}$$

$$\therefore\ A = \mathrm{r(1 - r)} + (1 - \mathrm{r})\frac{\mathrm{N_c} + \mathrm{r} - 2}{\mathrm{N_c} - 1}$$

Now $r \leq 1$ and $\mathrm{N_c} - 2 + r < \mathrm{N_c} - 1$ so $A < 2(1-r)$.This proves that, with large enough training set, no classifier can do better than half the probability of error rate of a 1-nearest neighbor classifier [30]

2.6 Problems with Nearest Neighbor Classifier

Though nearest neighbor methods are very easy to implement, they have many drawbacks.

1 Computationally expensive – Nearest neighbor classifiers compute distance of the input vector to all the center vectors. This distance measurement is computationally expensive and requires that all the center vectors to be stored in memory. This increases the computational complexity and memory requirements. Due to these computational complexities they cannot be used for real time applications.

2 Curse of dimensionality - The accuracy of the nearest neighbor classifiers tends to decrease as the number of features or inputs increases [46]. The reason is that in a high-dimensional space all points tend to be far away from each other, so nearest neighbors are not meaningfully similar. Practically, if vectors (patterns) are represented using many features, then every pair of examples will likely disagree on many features, so it will be rather arbitrary which vectors are closest to each other [9].

3 Contaminated input features – noise and less discriminative input features can cause problems such as convergence difficulties, poor classification accuracy and contamination of the distance measure which leads to false classification.

4 Rigid Voronoi cells - Clustering algorithms often get stuck in local minima and the result is largely dependent on the choice of initial cluster centers [3] [4]. Generated clusters, $\boldsymbol{m}_{ik}$ are not changed after initialization, and are not chosen to minimize, P_{e-NNC}, the

probability of error of the nearest neighbor classifier so they are not optimal. Clustering methods other than the Learning Vector Quantization (LVQ) method [2] do not adapt the center vectors in a way that minimizes the probability of error. [10].

Chapter 3

STRUCTURE OF OAWNNC

This chapter, introduces Optimal Attribute Weighting in a Nearest Neighbor Classifier (OAWNNC) algorithm that optimizes distance measure weights. This provides a solution to problem solves problem 3 mentioned in section 2.6.

3.1 Motivations behind the structure of OAWNNC

OAWNNC uses weighted Euclidean distance measure instead of regular Euclidean distance measure. Weights for the distance measure are initialized as $w(n) = \frac{1}{Var(x(n))+\epsilon}$ as mentioned in section 2.4.4. This reduces the dominance of inputs with high variance. However these weights are not optimal as they barely contribute in improving the performance of the classifier. Traditional nearest neighbor classifier, has a probability of error, P_{e-NNC}, as a measure of how well the classifier performs. To calculate the optimal weights, there needs to be a way to minimize P_{e-NNC} with respect to the weights, $\boldsymbol{w}$. SinceP_{e-NNC} is a scalar value and its gradient with respect to weights is zero i.e $\frac{\partial P_{e-NNC}}{\partial w(n)} = 0$, there is no direct way to minimize P_e, with respect to the weights to find optimal weights.

To solve this problem, OAWNNC maps traditional nearest neighbor classifier to a neural net. Optimizing the objective function of the neural network helps in calculating optimal weights that improve the classification performance of the neural network. It is really important to derive a mapping function that provides a one to one mapping between the NNC discriminant d_i and the neural network discriminant $y(i)$, such that improving the classification performance of the neural network, improves the probability of error,

P_{e-NNC}, of the NNC. OAWNNC uses a modified softmax discriminant function $y(i)$. This function is defined as

$$\mathrm{y(i)} = \frac{(\mathrm{d_i})^{-1}}{\sum_{\mathrm{j=1}}^{\mathrm{N_c}}(\mathrm{d_j})^{-1}}$$

The softmax discriminant $y(i)$ takes in the inverse of distances of test vector from the closest center vectors of each class and outputs a score for that class. It provides a one to one mapping from d_i when the inverse of distances of the test pattern to the closest center vector of each class adds up to one, i.e $\sum_{\mathrm{j=1}}^{\mathrm{N_c}}(\mathrm{d_j})^{-1} = 1$. Since this function is continuous at all points, gradients for optimization can be easily computed. The class of the test pattern is estimated from the output score vector, $\boldsymbol{y}$, as

$$\mathrm{i_c'} = \underset{\mathrm{i}}{\mathrm{argmin}}\, \mathrm{y(i)}$$

Softmax function outputs a score in the range of 0 to 1. With the score of the correct class being close to 1 and that of the incorrect class close to 0. It makes it easier to comprehend the performance of the classifier if its outputs as the scores are interpretable as posterior probabilities of categorical target output. For this reason, OAWNNC chooses the target output of the p^{th}, $\boldsymbol{t}_p$, of the correct class to be 1 and those of incorrect class to be 0 is chosen. This is called one-hot encoding technique.

$$\mathrm{t_p(i)} = \delta(\mathrm{i} - \mathrm{i_c(p)})$$

where $i_c(p)$ is the correct class of the p^{th}, pattern.

OAWNNC converts the classification problem into regression by using the mean square error function (MSE). Squared error loss is one of the most widely used loss function in statistics. In statistical modelling the MSE, representing the difference between the actual target output and the output values predicted by the neural network, is used to determine the extent to which the network fits the data and whether the removal or some

explanatory variables, simplifying the model, is possible without significantly harming the model's predictive ability [25]. The objective function used in training $y(i)$ is

$$\mathrm{E} = \frac{1}{\mathrm{N_v}} \sum_{\mathrm{p=1}}^{\mathrm{N_v}} \sum_{\mathrm{j=1}}^{\mathrm{Nc}} \left(\mathrm{t_p(j)} - \mathrm{y_p(j)} \right)^2 \tag{3.1}$$

3.2 Modified Weighted Euclidean Distance Measure

During classifier training to calculate optimal weights, $\boldsymbol{w}$, using regular weighted Euclidean distance measure as mentioned in section 2.4.4 few weight elements $w(n)$ would become negative. This made the distance measure value, d, negative and thus led to decreased performance and increased misclassification error. To solve this issue, this thesis proposes two different modified weighted Euclidean distance measure techniques.

1. Use absolute value of weights, $| w(n) |$, for distance calculation.

$$\mathrm{d}\left(\mathbf{x_p}, \mathbf{m_{ik}}\right) = \sum_{\mathrm{n=1}}^{\mathrm{N}} \left(| \mathrm{w(n)} | \cdot \left(\mathrm{x_p(n)} - \mathrm{m_{ik}(n)} \right) \right)^2 + \in \tag{3.2}$$

2. Use squared values of weights, $\mathrm{w(n)}^2$, for distance calculation.

$$\mathrm{d}\left(\mathbf{x_p}, \mathbf{m_{ik}}\right) = \sum_{\mathrm{n=1}}^{\mathrm{N}} \left(\mathrm{w(n)}^2 \cdot \left(\mathrm{x_p(n)} - \mathrm{m_{ik}(n)} \right) \right)^2 + \in \tag{3.3}$$

$\in$ is a small positive constant of the order 10^{-3} that prevents the distances from being zero and avoids division by zero while mapping the NNC to the neural network. Throughout this thesis squared weights, $w(n)^2$, are used, since during weight optimization training with different datasets, it was found that squared weights, $w(n)^2$, perform better as compared to absolute weights, $| w(n) |$.

3.3 Training OAWNNC for distance measure improvement.

In order to solve the problems of the curse of dimensionality, and misclassifications caused by noise and less discriminative features, it is necessary to optimize the distance measure algorithm. To ensure that our distance measure emphasizes more on highly discriminative features than the less discriminative features, we optimize weights corresponding to each input feature.

Training of a classifier consists of changing the weights in order to make the computed output as close as possible to the desired output, thus reducing the mean square error (MSE). It mainly involves the following two independent steps. First a search direction has to be determined. i.e., in what direction do we want to search in weight space for a new current point? Once the search direction has been found we have to decide how far to go in the specified search direction, i.e., a step size has to be determined. Most of the optimization methods used to minimize error functions are based on the same strategy. The minimization is a local iterative process in which an approximation to the error function in a neighborhood of the current point in weight space is minimized.

3.3.1 First Order Training Algorithms for Weight Optimization

In this section, steepest descent which is the first order optimization algorithm is used. The negative gradient the MSE with respect to input weights are calculated as follows,

$$\mathbf{g} = -\frac{\partial \mathrm{E}}{\partial \mathbf{w}} \tag{3.4}$$

Elements of the negative gradient vector $\boldsymbol{g}$ are calculated from the above equation (3.4) as

$$g(n) = -\frac{\partial E}{\partial w(n)} = \frac{2}{N_v} \cdot \sum_{p=1}^{N_v} \sum_{j=1}^{Nc} \left(t_p(j) - y_p(j) \right) \cdot \frac{\partial y_p(j)}{\partial w(n)} \tag{3.5}$$

where, taking the partial derivative of E in equation (3.1) yields

$$\frac{\partial y_p(j)}{\partial w(n)} = \frac{-\sum_{u=1}^{Nc} d_u^{-1} \cdot \left(d_j\right)^{-2} \cdot 2 \cdot w(n) \cdot \left[x_p(n) - m_{jk}(n)\right]^2 + \left(d_j\right)^{-1} \cdot \sum_{u=1}^{Nc} (d_u)^{-2} \cdot 2 \cdot w(n) \cdot \left[x_p(n) - m_{uk}(n)\right]^2}{\left(\sum_{u=1}^{Nc} (d_u)^{-1} \right)^2} \tag{3.6}$$

Input weight changes are calculated using the negative gradients from equation (3.4), (3.5) and (3.6) and a learning factor z, where, z, is a heuristically chosen scalar value. The weights are updated as follows,

$$\mathbf{w} \leftarrow \mathbf{w} + z \cdot \mathbf{g} \tag{3.7}$$

Optimal Learning Factor

The learning factor, z, decides the rate of convergence of training. Usually a small positive value for z will work, but convergence is likely to be slow. If z is too large the error, E, can increase [25]. In order to avoid this uncertainty a lot of heuristic scaling approaches have been introduced to modify the learning factors between iterations and thus speed up the rate of convergence. However using a Taylor's series for the error E, a non-heuristic Optimal Learning Factor (OLF) can be calculated as**,**

$$z = \frac{-\frac{\partial E}{\partial z}}{\frac{\partial^2 E}{\partial z^2}} \tag{3.8}$$

where the numerator and denominator derivatives are evaluated at $z = 0$. Assume that the learning factor, z, is used to update only the input weights $\boldsymbol{w}$, as given in equation (3.7).

Using the gradient $\boldsymbol{g}$, the optimal learning factor is derived in the following steps,

$$-\frac{\partial \mathrm{E}}{\partial \mathrm{z}} = \frac{2}{\mathrm{N_v}} \cdot \sum_{\mathrm{p=1}}^{\mathrm{N}} \sum_{\mathrm{j=1}}^{\mathrm{Nc}} \left(\mathrm{t_p(j)} - \mathrm{y_p(j)} \right) \cdot \frac{\partial \mathrm{y_p(j)}}{\partial \mathrm{z}} \tag{3.9}$$

where,

$$\frac{\partial \mathrm{y_p(j)}}{\partial \mathrm{w(n)}} = \frac{\mathrm{A} - \mathrm{B}}{\left(\sum_{\mathrm{u=1}}^{\mathrm{N_c}} (\mathrm{d_u})^{-1} \right)^2} \tag{3.10}$$

where,

$$\mathrm{A} = -\sum_{\mathrm{u=1}}^{\mathrm{N_c}} \mathrm{d_u^{-1}} \cdot \left(\mathrm{d_j}\right)^{-2} \cdot \sum_{\mathrm{p=1}}^{\mathrm{N_v}} (2 \cdot \mathrm{g(n)} \cdot \mathrm{w(n)} \cdot \left[\mathrm{x_p(n)} - \mathrm{m_{jk}(n)}\right]^2)$$

and

$$\mathrm{B} = -\left(\mathrm{d_j}\right)^{-1} \cdot \sum_{\mathrm{u=1}}^{\mathrm{Nc}} \mathrm{d_u^{-2}} \cdot \sum_{\mathrm{p=1}}^{\mathrm{N_v}} (2 \cdot \mathrm{g(n)} \cdot \mathrm{w(n)} \cdot \left[\mathrm{x_p(n)} - \mathrm{m_{uk}(n)}\right]^2)$$

Also, Gauss-Newton approximation for second partial is given by,

$$\frac{\partial^2 \mathrm{E}}{\partial \mathrm{z}^2} = \frac{2}{\mathrm{N_v}} \cdot \sum_{\mathrm{p=1}}^{\mathrm{N}} \sum_{\mathrm{j=1}}^{\mathrm{Nc}} \left[\frac{\partial \mathrm{y_p(j)}}{\partial \mathrm{z}}\right]^2 \tag{3.11}$$

Thus the optimal learning factor is calculated using equations (3.8), (3.9), (3.10) and (3.11). After finding the optimal learning factor the input weights are updated as given in equation (3.7)

First Order Training Algorithm Summary for Weight Optimization

1) Cluster N_v training patterns into K clusters using K-Means++ clustering algorithm [45], where $K = \sum_{\mathrm{i=1}}^{\mathrm{Nc}} k_i$. k_i is the number of clusters of the i^{th} class

   After clustering we get K center vectors $\boldsymbol{m}_{ik}$, where $\boldsymbol{m}_{ik}$is the k^{th} center vector of the i^{th} class

2) Initialize $w(n)$
3) For iteration, i_t = 1 to N_{it}, where N_{it} is the total number of iterations,

4) During first data pass calculate $d_i, \boldsymbol{y}, E, \boldsymbol{g}$

5) During second data pass calculate $-\frac{\partial E}{\partial z}, \frac{\partial^2 E}{\partial z^2}, z$

6) Update $\boldsymbol{w} \leftarrow \boldsymbol{w} + z \cdot \boldsymbol{g}$

7) End iterations

* Refer to Appendix A for the pseudocode

3.3.2 Second Order Training algorithm for Weight Optimization

The second order training of a MLP involves quadratic modeling of the error function. Second order training methods are preferred because of their fast convergence. However, they can lead to problems like memory limitation, since the hessian and gradient matrices should be computed and stored. They also are computationally very expensive.

Newton's Method

Newton's method is the basis of number of popular second order optimization algorithms. Newton's algorithm is iterative, where in each iteration, [28]

1) Calculate Newton weight change vector $\boldsymbol{e}$.
2) Update weights with this weight change vector $\boldsymbol{e}$.

The weight change vector $\boldsymbol{e}$ is calculated by solving the linear equations using OLS [27]

$$\mathbf{H} \cdot \mathbf{e} = \mathbf{g} \tag{3.12}$$

where, $\boldsymbol{g}$ is the negative gradient of MSE with respect to weights, calculated using equation (3.4), (3.5) and (3.6) and $\boldsymbol{H}$ is Hessian of the objective function calculated with respect to all the weights in the network and has elements defined as,

$$\mathrm{h(i,j)} = \frac{\partial^2 \mathrm{E}}{\partial \mathrm{w(i)}\, \partial \mathrm{w(j)}} \tag{3.13}$$

Equation (3.12) is solved for $\boldsymbol{e}$ using Orthogonal Least Squares (OLS) [27] and $\boldsymbol{w}$ is updated as

$$\mathbf{w} \leftarrow \mathbf{w} + \mathbf{e} \tag{3.14}$$

We continue to update $\boldsymbol{w}$ in this fashion till the change in the training error from the previous iteration is less than10^{-6}.

Second Order Training Algorithm Summary for Weight Optimization

1) Cluster N_v training patterns into K clusters using K-Means++ clustering algorithm [45].

   After clustering we get K center vectors $\boldsymbol{m}_{ik}$, where $\boldsymbol{m}_{ik}$ is the k^{th} center vector of i^{th} class

2) Initialize $w(n)$
3) For iteration, i_t = 1 to N_{it}, where N_{it} is the total number of iterations,
4) Calculate $d_i, \boldsymbol{y}, E, \boldsymbol{g}, \boldsymbol{H}$
5) Calculate $\boldsymbol{e}$
6) Update $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{e}$
7) End iterations

* Refer to Appendix A for the pseudocode

3.4 OAWNNC Results

Table – 1 compares randomized 10-fold testing results from traditional nearest neighbor classifier and OAWNNC with distance measure optimization (DMO).

Data Set	*Number of inputs*	*Number of classes*	*NNC accuracy %*	*DMO accuracy %*
F17C.dat	*17*	*39*	*25.5005*	*68.7039*
SKIN.dat	*2*	*2*	*93.8531*	*94.4196*
GONGTST.tst	*16*	*10*	*66.8333*	*77.8000*
COMF18.TRA	*18*	*4*	*54.2776*	*73.6481*
PHONEME.dat	*5*	*2*	*61.4701*	*75.3252*
MAGIC	*10*	*2*	*71.4346*	*79.3871*

Table 1 - Classification Performance of NNC v/s OAWNNC with Distance Measure Optimization (DMO).

This is a plot of mean square error (MSE) versus iteration number, N_{it}, for the nearest neighbor classifiers mapped neural network with first order and second training for weight optimization on COMF18.tra dataset. From the plot it concludes, that MSE converges much faster using second order training as compared to the first order training.

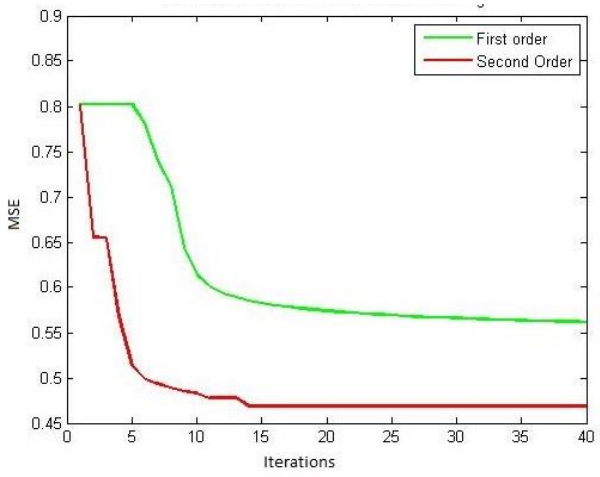

Figure 1 – Weight Optimization Training Iteration Difference for COMF18.tra dataset.

From the results shown in Table – 1, it can be concluded that OAWNNC with distance measure optimization outperforms the traditional NNC on all the datasets. From Figure – 1, second order training error converges much faster than first order training. The trained weights are such that the weights corresponding to noisy features are less in magnitude and vice versa. Thus OAWNNC with distance measure optimization (DMO) solves problem 3 mentioned in section 2. 6.

Chapter 4

CENTER VECTOR OPTIMIZATION (CVO)

This section introduces a method to move or adjust the center vectors $\boldsymbol{m}_{ik}$ in an optimized way using second order training to reduce the probability of error and improve classification accuracy.

Cluster center vectors formed by using clustering algorithms are rigid so they do not change after initialization. Thus our OAWNNC algorithm is dependent on these initial clusters. Since these clusters do not contribute to minimizing the probability of error, they are not optimal. In addition to this, clustering is sensitive to the choice of initial clusters, clustering parameters etc. Thus a few center vectors generated may not be unique. To alleviate these problems, we need to remove the sensitivity of the classifier to initial clustering and the rigid nature of clusters. For this it is necessary to find a way to move the center vectors in a way that minimizes the probability of error. The LVQ method is a technique that does something similar by adapting the center vectors that minimizes input-output mapping [10].

4.1 Structure of Center Vector Optimization (CVO)

Here, CVO minimizes the MSE error from equation (3.1) with respect to the i^{th} class, k^{th}center vector $\boldsymbol{m}_{ik}$**.** First, it calculates the negative gradient of the MSE with respect to $\boldsymbol{m}_{ik}$ and then moves the center vector in the direction of the negative gradient using a second order algorithm. The negative gradient, $\boldsymbol{g}_{ik}$, of the MSE with respect to $\boldsymbol{m}_{ik}$ is calculated as follows,

$$\boldsymbol{g}_{ik} = -\frac{\partial \mathrm{E}}{\mathbf{m}_{\mathrm{ik}}} \tag{4.1}$$

$$-\mathbf{g}_{\mathrm{ik}} = -\frac{\partial \mathrm{E}}{\partial \mathrm{m}_{\mathrm{ik}}(\mathrm{n})} = \frac{2}{\mathrm{N_v}} \cdot \sum_{\mathrm{p=1}}^{\mathrm{N_v}} \sum_{\mathrm{j=1}}^{\mathrm{Nc}} \left(\mathrm{t_p(j)} - \mathrm{y_p(j)} \right) \cdot \frac{\partial \mathrm{y_p(j)}}{\partial \mathrm{m_{ik}(n)}} \tag{4.2}$$

where, $y_p(j)$ is calculated as,

$$\mathrm{y_p(j)} = \frac{\left(\mathrm{d_j}\right)^{-1}}{\sum_{\mathrm{u=1}}^{\mathrm{Nc}} (\mathrm{d_u})^{-1}} \tag{4.3}$$

and $\frac{\partial \mathrm{y_p(j)}}{\partial \mathrm{m_{ik}(n)}}$ is computed as,

$$\frac{\partial \mathrm{y_p(j)}}{\partial \mathrm{m_{ik}(n)}} = 0$$

if $\boldsymbol{m}_{ik}$ is not participating in computing $y_p(j)$

else,

$$-\frac{\partial \mathrm{y_p(j)}}{\partial \mathrm{m_{ik}(n)}} = 2 \cdot \mathrm{d_j^{-2}} \cdot \mathrm{w(n)^2} \cdot \left(\mathrm{x_p(n)} - \mathrm{m_{ik}(n)} \right) . \left(\sum_{\mathrm{u=1}}^{\mathrm{Nc}} \mathrm{d_u^{-1}} - \mathrm{d_j^{-1}} \right) / \sum_{\mathrm{u=1}}^{\mathrm{Nc}} \mathrm{d_u^{-1}} \tag{4.4}$$

when $\mathrm{i} = \mathrm{j}$, and

$$-\frac{\partial \mathrm{y_p(j)}}{\partial \mathrm{m_{ik}(n)}} = -2 \cdot \mathrm{d_j^{-1}} \cdot \mathrm{d_i^{-2}} \cdot \mathrm{w(n)^2} \cdot \left(\mathrm{x_p(n)} - \mathrm{m_{ik}(n)} \right) / \sum_{\mathrm{u=1}}^{\mathrm{Nc}} \mathrm{d_u^{-1}}$$

when $\mathrm{i} \neq \mathrm{j}$

After calculating the negative gradient, $\boldsymbol{g}_{ik,}$ of the MSE with respect to $\boldsymbol{m}_{ik}$, CVO moves the center vectors in the direction of these negative gradients in order to minimize the error. Center vector are updated as follows

$$\boldsymbol{m}_{ik} \leftarrow \boldsymbol{m}_{ik} + z_{ik} \cdot \boldsymbol{g}_{ik} \tag{4.5}$$

4.2 Multiple Optimal Learning Factor (MOLF)

Since learning factor, z, decides the rate of convergence of classifier training, multiple learning factor, $\boldsymbol{z}$, one for each center vector proves to be very efficient during training. Using a Taylor's series for the error E, and Newton's algorithm [26] a non-heuristic multiple optimal learning factor (MOLF) can be calculated as,

$$\boldsymbol{H} \cdot \boldsymbol{z} = \boldsymbol{g_z} \tag{4.6}$$

where, $\boldsymbol{g}_z$ is the negative gradient of error E with respect to the learning factors $\boldsymbol{z}$, after replacing center vector $\boldsymbol{m}_{ik}$ as given in equation (4.5) and evaluated at $z_{ik} = 0$. Matrix $\boldsymbol{H}$, is the Hessian matrix of the objective function. Assume that the learning factor, z_{ik}, is used to update only the center vector, $\boldsymbol{m}_{ik}$, as given in equation (4.5). Using $\boldsymbol{g}_{ik}$ from equation (4.1), negative gradient $\boldsymbol{g}_{zik}$ is calculated as follows,

$$\mathbf{g}_{\mathrm{zik}} = -\frac{\partial \mathrm{E}}{\partial \mathrm{z}_{ik}} = \frac{2}{\mathrm{N_v}} \cdot \sum_{\mathrm{p=1}}^{\mathrm{N_v}} \sum_{\mathrm{j=1}}^{\mathrm{Nc}} \left(\mathrm{t_p(j)} - \mathrm{y_p(j)} \right) \cdot \frac{\partial \mathrm{y_p(j)}}{\partial \mathrm{z}_{ik}} \tag{4.7}$$

as mentioned in section 4.1, if center vector, $\boldsymbol{m}_{ik,}$ is not participating in computing $y_p(j)$ using equation (4.3) then,

$$\frac{\partial \mathrm{y_p(j)}}{\partial \mathrm{z_{ik}}} = 0$$

else,

$$-\frac{\partial \mathrm{y_p(j)}}{\partial \mathrm{z_{ik}}} = 2 \cdot \mathrm{d_j^{-2}} \cdot \Big(\sum_{\mathrm{n=1}}^{\mathrm{N}} \mathrm{g_{ik}(n)} \cdot \mathrm{w(n)^2} \cdot \left(\mathrm{x_p(n)} - \mathrm{m_{ik}(n)} \right) \Big) . \Big(\sum_{\mathrm{u=1}}^{\mathrm{Nc}} \mathrm{d_u^{-1}} - \mathrm{d_j^{-1}} \Big) / \sum_{\mathrm{u=1}}^{\mathrm{Nc}} \mathrm{d_u^{-1}} \tag{4.8}$$

when i = j, and

$$-\frac{\partial \mathrm{y_p(j)}}{\partial \mathrm{z_{ik}}} = -2 \cdot \mathrm{d_j^{-1}} \cdot \mathrm{d_i^{-2}} \cdot \Big(\sum_{\mathrm{n=1}}^{\mathrm{N}} \mathrm{g_{ik}(n)} \cdot \mathrm{w(n)^2} \cdot \left(\mathrm{x_p(n)} - \mathrm{m_{ik}(n)} \right) \Big) / \sum_{\mathrm{u=1}}^{\mathrm{Nc}} \mathrm{d_u^{-1}}$$

when i ≠ j .The Hessian matrix, $\boldsymbol{H}$, of the objective function calculated with respect to all the center vectors is computed as,

$$h(i,j) = \frac{\partial^2 E}{\partial z_{ik} \partial z_{jk}} \tag{4.9}$$

Equation (4.6) is solved for $\boldsymbol{z}_{,}$ using OLS [27] and $\boldsymbol{m}_{ik}$ is updated according to equation (4.5).

CVO Training Algorithm Summary

1) For iteration, i_t = 1 to N_{it}, where N_{it} is the total number of iterations.
2) During 1st data pass calculate $d_i, \boldsymbol{y}, E, \boldsymbol{g}_{ik}$
3) During second data pass calculate $-\frac{\partial E}{\partial z_{ik}}, \frac{\partial^2 E}{\partial z_{ik}^2}, \boldsymbol{z}$
4) Update $\boldsymbol{m}_{ik} \leftarrow \boldsymbol{m}_{ik} + z_{ik} \cdot \boldsymbol{g}_{zik}$
5) End iterations

4.3 CVO Results

Table – 2 compares randomized 10-fold testing results from traditional nearest neighbor classifier, OAWNNC and OAWNNC with center vector optimization.

Data Set	*Number of inputs*	*Number of classes*	*NNC accuracy %*	*OAWNNC with DMO accuracy %*	*OAWNNC with DMO and CVO accuracy %*
F17C.dat	*17*	*39*	*25.5005*	*68.7039*	*69.0200*
SKIN.dat	*2*	*2*	*93.8531*	*94.4196*	*95.4206*
GONGTST.tst	*16*	*10*	*66.8333*	*77.8000*	*80.4667*
COMF18.TRA	*18*	*4*	*54.2776*	*73.6481*	*77.2800*
PHONEME.dat	*5*	*2*	*61.4701*	*75.3252*	*76.3194*
MAGIC	*10*	*2*	*71.4346*	*79.3871*	*81.3103*

Table 2 - Classification Performance of NNC v/s OAWNNC with DMO v/s OAWNNC with DMO and CVO

From the results shown in the Table – 2, it can be concluded that center vector optimization makes OAWNNC insensitive to initial cluster center vectors. It optimally moves the cluster center vectors that further improves the performance of OAWNNC as compared to traditional nearest neighbor classifier. Thus CVO solves problem 4 mentioned in section 2.6.

Chapter 5

RESULTS AND CONCLUSION

This chapter presents results on several data sets to show that the new methods are successful in improving traditional NNC. These datasets are described in details in Appendix B. The training iterations were stopped when the mean square error difference in consecutive iterations was less than 10^{-5}, for five consecutive iterations or if number of iterations reached 100. Center vectors were generated from N_v training patterns using K – Means clustering algorithm [45]. The location of these cluster center vectors were intentionally perturbed in a random manner by adding Gaussian noise. This was done to validate the dependency of the traditional nearest neighbor classifier on initial center vectors and also validate the performance of the center vector optimization algorithm.

5.1 Improved Distance Measure Results

Random noise features drawn from standard normal distribution were added to the data files and the larger datasets were put through the distance measure optimization. To measure how distance measure optimization improves the classification accuracy of the NNC. Table – 3 compares randomized 10 - fold testing results from traditional nearest neighbor classifier and OAWNNC with improved distance measure after augmenting the training data with noise features.

Data Set	*Number of inputs*	*Number of noise features*	*Number of classes*	*NNC accuracy %*	*OAWNNC with DMO accuracy %*
F17C.dat	*17*	*5*	*39*	*19.6985*	*60.5496*
SKIN.dat	*2*	*2*	*1*	*71.3204*	*79.8257*
GONGTST.tst	*16*	*4*	*10*	*59.9184*	*73.1963*
COMF18.TRA	*18*	*5*	*4*	*52.0759*	*71.8160*
PHONEME.dat	*5*	*5*	*2*	*56.9096*	*70.3860*
MAGIC	*10*	*5*	*2*	*69.3054*	*78.6154*

Table 3 – Effect of Weight Optimization on Random Noise Features.

5.2 Center Vector Optimization Results

This sub-section shows the improvements in probability of error of the traditional nearest neighbor classifier after center vector optimization only. Training data is divided into cluster center vectors using K-Means++ [45]. Classification is performed using these center vectors. Center vector optimization moves these cluster center vectors optimally to improve the performance. Table – 4 compares randomized 10-fold testing results from traditional nearest neighbor classifier and OAWNNC with center vector optimization only.

Data Set	*Number of inputs*	*Number of clusters*	*Number of classes*	*NNC accuracy %*	*CVO only accuracy %*
F17C.dat	*17*	*78*	*39*	*25.5005*	*43.6954*
SKIN.dat	*2*	*6*	*2*	*93.8531*	*94.1651*
GONGTST.tst	*16*	*30*	*10*	*66.8333*	*73.3521*
COMF18.TRA	*18*	*12*	*4*	*54.2776*	*69.4915*
PHONEME.dat	*5*	*6*	*2*	*61.4701*	*72.1839*
MAGIC	*10*	*6*	*2*	*71.4346*	*75.9686*

Table 4 – Effect of Center Vector Optimization on Traditional Nearest Neighbor Classifier without Noise Features

5.3 Comparison with LVQ [2]

Learning Vector Quantization [2] is a supervised learning algorithm that applies a winner-take-it-all based approach. . Winner-take-all training algorithm determines, for each input pattern, the center vector that is closest to the input according to a given distance measure. The position of this so-called winner center vector is then adapted, i.e. the winner is moved closer if it correctly classifies the data point or moved away if it classifies the data point incorrectly. The performance of OWNNC with CVO was compared to LVQ version 2.1 from MATLAB to show its practicality. To have a fair comparison, same number of center vectors were used for both LVQ and OAWNNC with CVO training. Table 5 shows that the OAWNNC with CVO algorithm performs better than LVQ on all five datasets.

Data Set	*Number of inputs*	*Number of classes*	*NNC accuracy %*	*LVQ V 2.1 accuracy %*	*OAWNNC + CVO accuracy %*
F17C.dat	*17*	*39*	*25.5005*	*68.7039*	*69.0200*
SKIN.dat	*2*	*2*	*93.8531*	*94.4*	*95.4206*
GONGTST.tst	*16*	*10*	*66.8333*	*67.3*	*80.4667*
COMF18.TRA	*18*	*4*	*54.2776*	*58.33*	*77.2800*
PHONEME.dat	*5*	*2*	*61.4701*	*74.99*	*76.3194*
MAGIC	*10*	*2*	*71.4346*	*71.9*	*81.3103*

Table 5 – Classification Results from NNC v/s LVQ v2.1 v/s OAWNNC with DMO and CVO

5.4 Conclusion

Form the results in Table – 3, it is evident that nearest neighbor classifier suffers greatly by noisy features. However, OAWNNC with DMO makes the classifier robust to these noisy features. Thus the performance of the traditional nearest neighbor classifier is improved. OAWNNC solves problem 3 mentioned in section 2.6. Results in Table – 4 show that the performance of the traditional nearest neighbor classifier is greatly dependent on the initial choice of center vectors. Non-optimal center vectors increases the probability of error nearest neighbor classifier. However, center vector optimization solves this dependency problem by optimally moving the center vector locations in input space. Center vector optimization alone reduces the probability of error. CVO solves problem 4 mentioned in section 2.6. OAWNNC with DMO and center vector optimization improves the performance of the nearest neighbor classifier and out performs similar algorithms like LVQ.

Appendix A

Pseudocode

- **Optimizing distance measure weights with optimal learning factor.**

1) Cluster N_v patterns into K clusters, where $K = \sum_{i=1}^{Nc} k_i$

   k_i is the number of clusters of i^{th} class

   After clustering we get K center vectors $\boldsymbol{m}_{ik}$, where $\boldsymbol{m}_{ik}$ is the k^{th} center vector of i_{th} class

1) Initialize $w(n) = \frac{1}{var(x(n))}$ for $1 \leq n \leq N$

2) Initialize $\boldsymbol{w_{old}} = \boldsymbol{w}$, $MSE_{old} = +inf$, $z_{old} = 0$

3) For $iter$ = 1 to $MaxIter$

   a) Initialize $z = 0, D1 = 0, D2 = 0, G = 0$

   b) For $p = 1\ to\ N_v$

      i. For $i = 1\ to\ N_c$

         - Compute $\mathrm{d_i} = \min\left(\mathrm{d}(\mathrm{x_p}, \mathrm{m_{ik}})\right)$ using equation (3.3)
         - Cache values of $\mathrm{argmin_{ik}}(\mathrm{d_p}(\mathrm{x_p}, \mathrm{m_{ik}}))$ in memory, required while computing OLF

      ii. Compute $\boldsymbol{y}$

      iii. End i

      iv. Compute MSE using equation (3.1) with one-hot encoding target output

      v. $\mathrm{MSE_{old}} = \mathrm{MSE}$

      vi. Compute negative gradient vector $\boldsymbol{g}$ using equation (3.4), (3.5) and (3.6)

      vii. Cache values of $\boldsymbol{y}$ and d_i for each pattern N_v in order to avoid re-computing these values during OLF calculation

      End p

   c) Using the cached values for each training pattern during the 1st pass through the data, and $\boldsymbol{g}$ perform second pass through the data to calculate OLF

      i. For p = 1 to N_v

ii. For $i = 1\ to\ N_c$

- Replace $w(n) \leftarrow w(n) + z \cdot g(n)$ in equation (3. 3) where $z = 0$. Compute $D1$ i.e negative 1_{st} order derivative of MSE w.r.t z using equation (3.9) and (3.10)
- Compute $D2$ i.e second derivative of MSE w.r.t z using equation (3.11)

END i

END p

iii. Using equation (3.8) compute OLF z

d) Update the weights as $w(n) \leftarrow w(n) + z \cdot g(n)$

e) Calculate MSE using updated weights

i. If $MSE < MSE_{old}$

iv. $\boldsymbol{w_{old}} = \boldsymbol{w}$, $z_{old} = z, MSE_{old} = MSE$

Else, perform back-tracking

v. $z = \frac{z}{2},\ w(n) \leftarrow w(n) + z \cdot g(n)$

End $iter$

4) Save Weights to disk

- **Optimizing distance measure weights using Newton's algorithm.**

1) Cluster N_v patterns into K clusters, where $K = \sum_{i=1}^{Nc} k_i$.

   k_i is the number of clusters of i^{th} class

   After clustering we get K center vectors $\boldsymbol{m_{ik}}$, where $\boldsymbol{m}_{ik}$ is the k^{th} center vector of i_{th} class

2) Initialize $w(n) = \frac{1}{var(x(n))}$ for $1 \leq n \leq N$

3) Initialize $\boldsymbol{w_{old}} = \boldsymbol{w}$, $z_{old} = 0$

4) For $iter$ = 1 to $MaxIter$

   f) Initialize $\boldsymbol{g} = 0$, $\boldsymbol{H} = 0$

   a) For $p = 1\ to\ N_v$

      i. For $i = 1\ to\ N_c$

      vi. Compute $d_i = \min\left(d(\boldsymbol{x}_p, \boldsymbol{m}_{i,k})\right)$ using equation (3.3)

      ii. Compute $\boldsymbol{y}$

      End i

      iii. Compute MSE using equation (3.1) with one-hot encoding target output

      iv. Compute negative gradient vector $\boldsymbol{g}$ using equation (3.4), (3.5) and (3.6)

      v. Compute Hessian matrix $\boldsymbol{H}$ using equation (3.13)

      End p

      vi. Solve equation (3.14) to compute update vector $\boldsymbol{e}$ using OLS [27]

      vii. Update weights as $w(n) \leftarrow w(n) + e(n)$

   End $iter$

5) Save Weights to disk

Appendix B

Description of datasets

1 GONGTST.TST

The raw data consists of images from hand printed numerals collected by the Internal Revenue Service. Images are 32 by 24 binary matrices. An image scaling algorithm is used to remove size variation in characters. This dataset contains 16 input features. The 10 classes correspond to 10 Arabic numerals.

2 COMF18.TRA

This dataset comes from [47] and has 18 input features to classify patterns into 4 distinct classes. These features are extracted from images as per Level 1 of the US Geological Survey Land Use/Land Cover Classification System to categorize into four regions of land use: urban areas, fields or open grass lands, trees (forest land) and water (lakes or river).

3 *F17C*

This dataset is used for the application of prognostics or flight condition recognition. It consists of parameters that are available in the basic health usage monitoring systems (HUMS), plus some others. The data was collected from M430 flight load level survey conducted in Mirabel Canada in early 1995. It has 17 input features and 39 classes.

4 Skin Segmentation Data Set

The skin dataset is collected by randomly sampling B,G,R values from face images of various age groups (young, middle, and old), race groups (white, black, and asian), and genders obtained from FERET database and PAL database [48]. Total number of training

patterns is 245057; out of which 50859 is the skin samples and 194198 is non-skin samples. It has 3 input features and 2 classes.

5 Phoneme Data Set

This dataset distinguishes between nasal and oral sounds. It has 3818 patterns with 5 input features and 2 classes [49].

6 Magic Gamma Telescope Data Set

This is a MC generated dataset to simulate registration of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope using imaging technique. The information consists of pluses left by incoming Cherenkov photons on the photomultiplier tubes, arranged on the camera plane. Depending on number of photons collected on the image plane, the patterns are classified into two categories: those caused by primary gamma and those caused hardonic showers initiated by cosmic rays.

References

[1] Cover, Thomas, and Peter Hart, "Nearest neighbor pattern classification." IEEE transactions on information theory 13.1 (1967): 21-27.

[2] Kohonen, Teuvo, Self-organization and associative memory. Vol. 8. Springer Science & Business Media, 2012.

[3] Selim SZ, Ismail MA (1984) K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. IEEE Transactions on Pattern Analysis and Machine Intelligence 6(1):81{87

[4] Du, Qiang, Vance Faber, and Max Gunzburger, "Centroidal Voronoi tessellations: applications and algorithms." SIAM review 41.4 (1999): 637-676.

[5] Daelemans, Walter, Steven Gillis, and Gert Durieux, "The acquisition of stress: A data-oriented approach." Computational Linguistics 20.3 (1994): 421-451.

[6] Wettschereck, Dietrich, A study of distance-based machine learning algorithms. Diss. 1994.

[7] Aha, David W, "Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms." International Journal of Man-Machine Studies 36.2 (1992): 267-287.

[8] Kohavi, Ron, Pat Langley, and Yeogirl Yun, "The utility of feature weighting in nearest-neighbor algorithms." Proceedings of the Ninth European Conference on Machine Learning. 1997.

[9] Syed, Muhammad Ejazuddin, "Attribute weighting in k-nearest neighbor classification." (2014).

[10] Rawat Rohit, Manry Michael T. “Second Order Training of a Smoothed Piecewise Linear Network”. December 2016.

[11] Vapnik, Vladimir. The nature of statistical learning theory. Springer Science & Business Media, 2013.

[12] Bhatia, Nitin. “Survey of nearest neighbor techniques.” arXiv preprint arXiv:1007.0085 (2010).

[13] V.Vaidehi, S. Vasuhi, “Person Authentication using Face Recognition”, Proceedings of the world congress on engg and computer science, 2008.

[14] Shizen, Y. Wu, “An Algorithm for Remote Sensing Image Classification based on Artificial Immune b-cell Network”, Springer Berlin, Vol 40.

[15] G. Toker, O. Kirmemis, “Text Categorization using k Nearest Neighbor Classification”, Survey Paper, Middle East Technical University

[16] Y. Liao, V. R. Vemuri, “Using Text Categorization Technique for Intrusion detection”, Survey Paper, University of California.

[17] E. M. Elnahrawy, “Log Based Chat Room Monitoring Using Text Categorization: A Comparative Study”, University of Maryland.

[18] X. Geng et. al, “Query Dependent Ranking Using k Nearest Neighbor”, SIGIR, 2008.

[19] Y. Yang and T. Ault, “Improving Text Categorization Methods for event tracking”, Carnegie Mellon University.

[20] F. Bajramovie et. al “A Comparison of Nearest Neighbor Search Algorithms for Generic Object Recognition”, ACIVS 2006, LNCS 4179, pp 1186-1197.

[21] Taskar, Ben. “Nearest Neighbor Methods”. Machine learning CIS520. University of Pennsylvania, Philadelphia.

[22] Arthur, David, and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding." Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2007.

[23] De Maesschalck, Roy, Delphine Jouan-Rimbaud, and Désiré L, Massart. "The mahalanobis distance." Chemometrics and intelligent laboratory systems 50.1 (2000): 1-18.

[24] Wikipedia contributors. "Softmax function." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 2 Dec. 2016. Web. 2 Dec. 2016.

[25] Wikipedia contributors. "Mean squared error." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 18 Nov. 2016. Web. 18 Nov. 2016.

[26] S. Haykin, “Neural Networks a Comprehensive Foundation,” 1994

[27] M. D. Robinson, and M. T. Manry, “Two-Stage Second Order Training in Feedforward Neural Networks,” Twenty-Sixth International Florida Artificial Intelligence Research Society Conference, 2013.

[28] Maldonado, F. J., and M. T. Manry. "Optimal pruning of feedforward neural networks based upon the Schmidt procedure." Signals, Systems and Computers, 2002. Conference Record of the Thirty-Sixth Asilomar Conference on. Vol. 2. IEEE, 2002.

[29] Cost, Scott, and Steven Salzberg. "A weighted nearest neighbor algorithm for learning with symbolic features." Machine learning 10.1 (1993): 57-78.

[30] Elkan, Charles. "Nearest Neighbor Classification." Web. <http://cseweb.ucsd.edu/~elkan/250Bwinter2010/nearestn.pdf>.

[31] Manry, Michael T. "Unsupervised Learning and Neural Nets That Use It." Neural networks EE 5353. University of Texas at Arlington. Texas. 16 November, 2016. Lecture.

[32] Wikipedia contributors. "Statistical classification." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 18 Nov. 2016. Web. 18 Nov. 2016.

[33] Bishop, Christopher M. "Pattern recognition." Machine Learning 128 (2006).

[34] Cortes, Corinna, and Vladimir Vapnik. "Support-vector networks." Machine learning 20.3 (1995): 273-297.

[35] Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. No. ICS-8506. CALIFORNIA UNIV SAN DIEGO LA JOLLA INST FOR COGNITIVE SCIENCE, 1985.

[36] Quinlan, J. Ross. "Simplifying decision trees." International journal of man-machine studies 27.3 (1987): 221-234.

[37] Wikipedia contributors. "Cluster analysis." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 13 Nov. 2016. Web. 13 Nov. 2016.

[38] Wikipedia contributors. "K-means clustering." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 22 Nov. 2016. Web. 22 Nov. 2016.

[39] Wikipedia contributors. "Euclidean distance." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 2 Nov. 2016. Web. 2 Nov. 2016.

[40] Wilson, D. Randall, and Tony R. Martinez. "Improved heterogeneous distance functions." Journal of artificial intelligence research 6 (1997): 1-34.

[41] Wikipedia contributors. "Mahalanobis distance." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 27 Sep. 2016. Web. 27 Sep. 2016.

[42] Wikipedia contributors. "K-nearest neighbors algorithm." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 28 Nov. 2016. Web. 28 Nov. 2016.

[43] Dempster, Arthur P., Nan M. Laird, and Donald B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm." Journal of the royal statistical society. Series B (methodological) (1977): 1-38.

[44] Ichino, Manabu, and Hiroyuki Yaguchi. "Generalized Minkowski metrics for mixed feature-type data analysis." IEEE Transactions on Systems, Man, and Cybernetics 24.4 (1994): 698-708.

[45] Arthur, David, and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding." Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 2007.

[46] Marimont, R. B., and M. B. Shapiro. "Nearest neighbour searches and the curse of dimensionality." IMA Journal of Applied Mathematics 24.1 (1979): 59-70.

[47] Bailey, Robert R., et al. "Automatic recognition of usgs land use/cover categories using statistical and neural network classifiers." Optical Engineering and Photonics in Aerospace Sensing. International Society for Optics and Photonics, 1993.

[48] Rajen, Bhatt, and Dhall Abhinav. "Skin segmentation dataset." UCI Machine Learning Repository (2012).

[49] "ELENA Project." ELENA Project. N.p., n.d. Web. 07 Dec. 2016.

Biographical Information

Jugal Raju Sheth was born in India in 1992. He received his Bachelor of Technology in Electronics Engineering from Narsee Monjee Institute of Management Studies, Mumbai, India in May 2014 and Master of Science in Electrical Engineering from the University of Texas at Arlington in December 2016.

He interned at Volvo Construction Equipment in fall 2015 and spring 2016, where he developed applications for real time pedestrian detection using stereo cameras leveraging machine learning and deep neural network algorithms. He has been involved in research activities under the guidance of Dr. Michael T. Manry in Image Processing and Neural Networks Laboratory (IPNNL) since 2015. His main research interests include Neural Networks and Pattern Recognition.