A Brief Review of Classifiers used in OCR Applications

Satish Kumar

Panjab University, SSG Regional Centre,
Hoshiarpur, Punjab(India)

Abstract *— The performance of a recognition system depends upon the classifiers used for classification purpose. Powerful is the discrimination ability of a classifier, better is its recognition performance. The generalization ability of a classifier is measured on the basis of its performance in classifying the test patterns. There are various factors which affect generalization. Moreover, the feature extraction method(s) used for training a classifier also affects the performance of a classifier. In this paper, a brief theoretical review of various classifiers is made. The various characters of each are covered. The classifiers covered are Bayes, Parzen, probabilistic, polynomial, discriminant, radial basis networks, multi layer perceptron(MLP), k-NN, SVM and SOM.*

Keywords — Classifiers, Recognition, PNN, SOM, *k*-NN, SVM, MLP.

I. INTRODUCTION

A recognition system must be trained in training or learning stage using a set of training samples. In the recognition process, the features or properties extracted from a character image are compared with the features of the character images (either stored in computer memory or used to train the system) whose classes are known and a class label is assign to it. A system that performs such type of classification task is called a classifier. Actually, a classifier works in two stages: 1) Learning stage and 2) Decision stage. In learning stage, the features extracted from the handwritten samples are used to train the classifier. The designing and training a classification system using a set of training samples, whose classes are known, is termed as supervised learning.

The type of classifier used for classification purpose affects the performance of a recognition system a lot. Powerful is the discrimination ability of a classifier; better is its recognition performance. The final goal of a recognition system is to classify a test sample which is unknown and not used to train the classifier and assign a label to it. The performance of a classifier on a test set (test samples) may not be same as it is for training sample set. One must have to optimize a classifier on given training and test sets, no matter which classifier is used. The generalization ability of a classifier is measured on the basis of its performance in classifying the test patterns. The various factors which affect generalization are as [1-3]:*a*) Size of dataset, *b*) The discrimination ability of the features used to represent patterns, *c*) Physical complexity of the problem, *d*) The optimality observed in training a classifier, *e*) Unknown parameters.

There is a lot of impact of the size data set used for training a classifier on the generalization. In addition to this, the available samples should be well representatives of the various classes under consideration. In real-world applications, it is better if the samples are taken from real situations rather than artificial ones. Curse of dimensionality is concerned with the size of training data set and the size of feature vector used to represent a sample. The size of the feature vector should be reasonably small with limited size of training data. The size of feature vector not only effects the generalization but also contributes to the classification time a lot. Moreover, the feature extraction method(s) used for training a classifier also affects the performance of a classifier a lot. The classifier should be reasonably trained on training samples. In learning or in training phase as a classifier is trained, the error on training dataset as well as on test dataset decreases. After certain epochs or cycles the error on training set decreases or remains constant but error on validation or test set further starts increasing. In such situation the classifier is said to be over-trained and it affects the generalization and degrades the performance of a classifier.

One most important factor related to the architecture of a classifier that affects generalization is the number of unknown parameters required to be optimized in a classifier. If these parameters are very large, the classifier will be too complex. Some of these parameters may not be sufficiently updated or molded due to large structure of a classifier and limited size of training data set. Consequently, it results in poor generalization. On the other hand, fewer numbers of unknown parameters may give large errors even on training data due to improper mapping of input data to desired output data. It also contributes to poor generalization. In case of neural network the unknown parameters are weights and biases. The complexity of the problem is natural factor and it depends upon the complexity of the patterns under consideration.

II. BRIEF REVIEW OF CLASSIFIERS

The various authors have suggested different classifiers to solve different character recognition problems and some such classifiers are: Bayes Classifier, Parzen window classifier, linear discrimination function, quadratic discrimination function, *k*-NN (*k*-nearest neighbors), polynomial classifier(PC),), learning vector quantization (LVQ), radial basis function(RBF), self organization map (SOM), probabilistic neural network(PNN multilayer perceptron(MLP), hidden Markov model (HMM). and support vector machine (SVM). A brief review of various classifiers such as Bayes classifier, linear discrimination function, quadratic discrimination function, Parzen window classifier, radial basis function (RBF), self organization map (SOM), probabilistic neural network, polynomial classifier (PC) is made in this Section. In addition to this, three more classifiers, *i.e.*, MLP, SVM and *k*-NN are also discussed in this Section.

.

A. Bayes Classifiers

In Bayes classifier, an unknown pattern u is assigned a class ω_i by minimizing conditional average risk *i.e.*

If $\Re_i(u) < \Re_j(u)$ for

$$j = 1,2,3,\ldots q;\ j \neq i. \quad (1)$$

Where $\Re_i(u)$ and $\Re_j(u)$ are the conditional average risk or loss incurred in assigning u to a class ω_i and $\omega_{j,}$ respectively. The number q represents total number of classes under consideration.

Expression (1) can be equivalently written as

if $p(u/\omega_i)P(\omega_i) > p(u/\omega_j).P(\omega_j)$ for

$$j = 1,2,3,4,5,\ldots,q;\ j \neq i, \quad (2)$$

Where $P(\omega_i)$ and $P(\omega_j)$ are the probabilities of occurrence of class ω_i and $\omega_{j,}$ respectively and $p(u/\omega_i)$ and $p(u/\omega_j)$ are probability density functions of patterns from class ω_i and ω_j , respectively.

In Bayes classifier, a sample u is assigned to a class having decision function $\Psi_j(u)$ [4].

$\Psi_j(u) = p(u/\omega_j).P(\omega_j)$ for

$$j = 1,2,3,4,5,\ldots,q; \quad (3)$$

In Bayes classifiers, the probability density functions used to express patterns in each class and the probability of each class must be available.

B. Discriminant Classifiers (Linear & Quadratic)

The probability density $p(u/\omega_j)$ of the Bayes classifier can be estimated using the following multivariate Gaussian function

$$p(u/\omega_j) = \frac{1}{(2\pi)^{n/2}|\sum_j|^{1/2}} e^{-\frac{1}{2}(u-\mu_j)^T \sum_j^{-1}(u-\mu_j)} \quad (4)$$

Here $\sum_j$ is a covariance matrix , μ_j is mean vector and $j = 1,2,3,\ldots q$; n is the dimension of pattern vector u.

From (3) and (4), we have quadratic discriminant (QD) classifier whose decision function is given by (5).

$$\Psi_j(u) = \ln P(\omega_j) - \frac{1}{2}\ln|\sum_j| - \frac{1}{2}[(u-\mu_j)^T \sum_j^{-1}(u-\mu_j)] \quad (5)$$

The decision boundaries of (5) are quadratic equations in u. If covariance matrix is the same for all the classes *i.e.* $\sum_j = \sum$ then (5) can be written as:

$$\Psi_j(u) = \ln P(\omega_j) + u^T \sum^{-1} \mu_j - \frac{1}{2}\mu_j^T \sum^{-1} \mu_j \quad (6)$$

Since decision boundaries of (6) are linear equations in u, it is called as linear classifier. Furthermore, if covariance matrix is identity matrix and the probability of each class is same, *i.e.*, $\frac{1}{q}$; then (6) can be written as:

$$\Psi_j(u) = u^T \mu_j - \frac{1}{2}\mu_j^T \mu_j \ ; \text{ for}$$

$$j = 1,2,3,\ldots q; \quad (7)$$

Quadratic discriminant classifier has following problems [5].

1) Poor estimation in parameter degrades performance.
2) The processing time and memory requirements are large.
3) Performance degrades if distribution is away from the normal distribution.

C. Parzen Classifiers

It is a non-parametric classifier. In Bayesian classifier the probability distribution $p(u/\omega_i)$ is usually unknown. One way to estimate the probability distribution is to use the Parzen's probability distribution function. In Parzen classifier,

the probability densities are estimated locally using Gaussian kernel function which is as follow:

$$p(u/\omega_i) = \frac{1}{V_i}\sum_{l=1}^{V_i}\varphi(u-u_{il}) \quad (8)$$

Where kernel φ is given as:

$$\varphi_i(u-u_{il}) = \frac{e^{-\frac{1}{2.D_i^2}(u-u_{il})^T.\sum_i^{-1}.(u-u_{il})}}{(2\pi)^{n/2}.D_i^n.|\sum_i|^{1/2}}$$

Where n is dimension of feature space, u_{il} is l^{th} pattern from class ω_i, V_i is number of training patterns from class ω_i, D_i is window width of class ω_i and $\sum_i$ is sample covariance matrix of ω_i. The performance of Parzen classifier depends upon the kernel function that depends upon the window width D_i. The selection of D_i is critical in design of Parzen classifiers [6-7].

D. Radial Basis Function Network

Radial basis function (RBF) network is a special class of multilayer neural network with a single hidden layer having non-linear functional nodes and an output layer with linear functional nodes. The output of a hidden layer unit is determined by the distance (generally the Euclidean distance) between the input vector u and the prototype vector μ_i. The activation function is Gaussian kernel. The output of the hidden layer node in terms of Gaussian function is given as:

$$\varphi_i(u) = \exp\left(-\frac{\|u-\mu_i\|^2}{2\sigma_i^2}\right) \quad (9)$$

Where u is input sample having n-dimensional feature vector and μ_i is mean vector and σ_i is standard deviation. The vector μ_i is also called as centre of the RBF unit. The output at the j^{th} functional node of the output layer is given as:

$$v_j(u) = \sum_{i=1}^{l} w_{ij}\varphi_i(u) \quad (10)$$

Where w_{ij} is weight between the i^{th} node of the hidden layer and j^{th} node of the output layer and l is the number of hidden layer nodes. Its some pros and cons are as [8,9]:

1). In training phase it converges fast as compared to back-propagation algorithm.

2). It is unaffected against garbage patterns due to a set of local radial basis functions.

3). RBF is slower in test phase as it consists of large number of functional units in hidden layer.

E. Probabilistic Neural Network (PNN)

The basis of PNN is the Bayesian classification theory. It uses Parzen windows to approximate the probability distribution of the input pattern which is given by (8).

The PNN design is four layered having 02 hidden, 01 input and 01 output layer. The input pattern is passed through input layer and it is fully linked with hidden layer called as pattern layer. There is one summation node to compute (8) for each class in second hidden layer. Final layer is decisive. PNN gives decision as per Bayes rule expressed in (3)

Some points against and in favor of PNN are as follows [10-11]:

1) Its training time is even small in comparison to other neural network classifiers.
2) It is slow during test phase since it required combining various training samples to get Parzen estimate.
3) Difficult to implement in applications that require large size of training data samples. So it is suited in applications that need learning rather generalization.

F. Polynomial Classifier

Polynomial Classifier consists of 3-layered structure having 01 hidden layer; 01 input and 01output layer. The input pattern is passed through input layer as in case of PNN. The elements of the hidden layer are corresponding to the enhanced features obtained by polynomial combination of original features (corresponding to the first layer). The enhanced features are represented by a set of basis functions, $Q(u)$. For example, for two dimensional feature vector, $u = [u_1, u_2]^T$ and with polynomial of degree two, the vector is given by

$$Q(u) = [1, u_1, u_2, u_1^2, u_1u_2, u_2^2]^T \quad (11)$$

The nodes of the output layer are corresponding to the decision functions, where j^{th} node represents the decision function of class j. The output of the final layer is linear combination of basis functions. The output of the j^{th} node is given by

$$v_j(u) = \sum_{i=1}^{l} w_{ij}Q_i(u) \quad (12)$$

Where, l is the size of hidden layer. The above equation can be written as

$$v(u) = w^T Q(u) \quad (13)$$

The coefficients of w are computed without iterative procedure (learning) from cross correlation matrix $\Lambda\{Q(u)\,d^T\}$ and moment matrix of enhanced features $\Lambda\{Q(u)Q(u)^T\}$ acquired from training set using equation no. (14)

$$\Lambda\{Q(u)Q(u)^T\}.\mathrm{w} = \Lambda\{Q(u)\,d^T\} \qquad (14)$$

Here *d* represents the desire response of the input pattern *u* and Λ (.) represents the expectation operator. The size of the polynomial grows if higher degree polynomial is considered; this enhances the size of feature vector. However, some measures have been suggested in [8] to shrink feature size.

G. Self Organizing Map

In Kohonen **s**elf organizing map(SOM), all the input nodes of input layer are connected to all the output nodes of the output layer. The nodes of output layer are generally arranged in two dimensional arrays. The connection weights are initialized using a random number generator. The Euclidean distance on each output node is computed. The output unit having minimum distance is selected and this node is called as winner node. All the connection weights to this node and some of its neighboring nodes(selected using some criteria) are updated. The neighborhood identifying function is generally a Gaussian function. SOM learning algorithm in brief is summarized as [2,12-13]:

1). the connection weights between all input and output nodes are initialized to a small random value. The neighbourhood function and learning parameters are also initialized.

2). Input a sample *u*. Compute Euclidean distance $DE_j(u)$ between the input node and the weight on each output node j and find a node (also called winning node) having this distance minimum using below given expression.

$$DE\ (u - w_{ij}) = \| u - w_{ij} \| ; \quad j = 1,2,3, \ldots, q; \quad i = 1,2,3, \ldots\ldots, n;$$
$$l = \text{Index of } [\min_j (\ DE\ (u - w_{ij}))] \qquad (15)$$

3). Update the weights of the winning node as well as some of its neighboring nodes in $(t+1)^{th}$ iteration using

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t)\, s_{lj}(u)[u(t) - w_{ij}(t)] \qquad (16)$$

Where $s_{lj}(u)$ represents the neighbourhood of l^{th} winner node on output layer to j^{th} node on output layer estimated using Gaussian function and $\eta(t)$ is learning parameter.

4). Repeat 2 and 3 until no evident changes in the feature map.

The SOM is generally used for clustering as it learns in unsupervised manner.

H. Nearest Neighbors Classifier

Nearest neighbor classifier is based on non-parametric classification method. It is not only simple but also proves to be very successful in many pattern recognition applications. It has its own place in classification methods as it is theoretically sound and practically simple and efficient. Cover et al [15] showed that the probability of error, say R_e, of *k*-NN is bounded below by Bayes probability of errors R_b and bounded above by twice the Bayes probability of errors for any number of classes, *i.e.*, $R_b \le R_e \le 2R_b$

Time complexity in *k*-NN is O($n \times m$). Here *n* is the size of training sample dataset and *m* is the size of features used. As far as the space requirement is concerned, the classifier requires complete training data set in memory, although this requirement can be relaxed by using secondary storage device and using main memory optimally by bringing only that data which is currently active. Recently, various rules such as editing rules, reduced rules, condensed rules and prototype based methods have been proposed by various authors in order to enhance the computational speed, reduce memory requirements and optimize classification accuracy. Some references about such rules are available in [30]. Though, it is not fast still successfully used in many pattern recognition problems. *k*-nearest neighbors classifier has been used in some handwritten recognition problems [17-19]. Some work on Indian languages, where *k*-NN is used, is reported in[20-23]. *k*-nearest neighbors classifier is a simple and flexible method of classification. It predicts unlabeled samples based on their similarity with samples in training data set.

Consider we have labeled training samples (u_i, d_i), $i=1,2,3 \ldots\ldots\ldots m$, $u_i \in R^n$ and $d_i \in (1,2,3\ldots..q)$, where u_i represents training samples and d_i label represents the class of samples from which a test sample belongs to out of *q* classes. The goal is to guess the correct label of a new unlabelled example *u*.

1) k-Nearest Neighbors Rule: *k*-NN assigns, *u* , the name of class that emerge as majority among *k* nearest samples. The rule is known as majority rule. In case the value of *k* is even or greater than two. In addition to this all the *k* samples belong to different categories creates uncertainty. In such cases, a random and nearest tiebreaker is taken to remove the conflict.

In order to implement *k*-NN rule, one requires

1) a data set of labeled training samples.
2) a distance metric, to compute the distance between a training sample and a test sample.
3) the value of *k* *i.e.* the number of nearest neighbors to be considered.

2) k-Factor: The choice of *k* impacts the performance of *k*-NN a lot. The performance of *k*-NN is satisfactory when *k=1*, but with overlapping classes the performance deters too. Similarly, if *k* is too large, the neighborhood may include samples from other classes leading to misclassification. There are various distance metrics used to find the distance between training samples and a test sample. Some popular metrics are Euclidean distance, city-block, and Hamming distance.

3) Euclidean Distance: The Euclidean distance $DE(u, u_i) = ||u - u_i||$ between an unlabeled test sample u and a labeled training sample u_i is given as

$$DE(u, u_i) = \sqrt{\sum_{j=1}^{n} (u(j) - u_i(j))^2} \qquad (17)$$

Where *n* is the size of feature vector of a sample pattern under consideration. *k*-nearest neighbors of an unlabeled test sample mean out of total training samples, *k* samples which have minimum value of Euclidean distance from an unlabeled sample are used for decision making.

The *k*-NN classifier is simple to use and implement. It requires no training before use and thus can easily adapt new training data. But it is lazy algorithm as it learns in test phase only. It requires large storage space since each training sample of data set is compared with the test sample. It is highly vulnerable to curse of dimensionality. With high dimensional feature vector the Euclidean distance calculation becomes quite expensive. High dimensional data not only increases the computational cost but also, some times, degrades the classification efficiency due to the presence of some non-active feature vectors.

I. Feed Forward Network with Back propagation

Among the various ANN based classifiers; a Feed Forward Network with Back propagation is mostly used classifier for handwritten recognition problems. Neural networks are not only used to solve pattern recognition problems. Jain et al [13] give the various other tasks that ANNs can perform and these tasks are categorization, function approximation, forecasting, optimization, associative memory and controlling I/O of various systems. The most commonly used feed forward network for pattern recognition problem is trained with error back-propagation algorithm, which is based on error-correction learning rules.

1) **Error Back-Propagation Algorithm:** Back-propagation algorithm was developed by Werbos in 1974 and it was further rediscovered by Parker and LeCun in 1975 and these developments were reported in 1986[13]. When feed-forward network is trained with error Back-propagation algorithm, the network consists of two kinds of signals [2]:

a) Forward signal (Input Signal), which is originated at input neuron of input layer and transmitted in forward direction through the network and appears at output neuron at output layer as an output. It is used to map given input data to desired output.

b) Backward signal (Error signal), which is originated at output neuron of output layer and propagated in backward direction through the network and used to update network weights.

Actually this network does not have backward feedback rather errors are back-propagated during network training and weights are adjusted dynamically. As already mentioned, an error back-propagation algorithm is based on error-correction learning rule. The objective is to bring the actual output v_k closer to desired output d_k equivalent to minimizing squared-error cost function (20). The delta rule (21) is used to update the weights of output layer neuron. But multilayer feed-forward network with back-propagation algorithm consists of number of hidden layers in addition to output layer. The delta rule (21) is extended to change the weights of hidden layer and so this rule is also called as generalized delta rule. Weight adjustment between any neurons *j* and *k* is proportional to the negative gradient of the error, generated at k^{th} neuron, with respect to weight, *i.e.*,

$$\Delta w_{jk}(t) = -\eta \frac{\partial E(t)}{\partial w_{jk}} \qquad (18)$$

2) Error-Correction Learning Rules: As already mentioned, in case of supervised learning the network is trained with an exact output for every input pattern. This learning is achieved in various iterations. In a given iteration the output generated by a network is not equal to exact/desired output. If v_k is the actual output generated and d_k is the desired output at k^{th} neuron in output layer in t^{th} iteration, then output error is

$$e_k(t) = d_k(t) - v_k(t) \qquad (19)$$

The error signal $e_k(t)$ is used to adjust all weights of k^{th} neuron. The adjustment in weights are made to bring output $v_k(t)$ of the k^{th} neuron closer to the desired output equivalent to minimizing squared-error cost function expressed in error signal terms as

$$E(t) = \frac{1}{2} e_k^2(t) \qquad (20)$$

This adjustment in weights is done in step-wise manner in number of iterations and when this error is at minimum, the learning process is terminated. The learning rule used to minimize error in this way is called error-correction learning rule.

Considered an input vector $u = \{u_1, u_2, u_3, \ldots\ldots u_i, \ldots u_n\}$ and weights due to this input vector to k^{th} neuron are w_{1k}, w_{2k}, w_{3k},……, w_{ik},…w_{nk} , then small change in i^{th} connection weight between i^{th} input node to k^{th} neuron in t^{th} iteration is given by

$$\Delta w_{ik}(t) = \eta e_k(t) u_i(t) \qquad (21)$$

where, η is learning rate parameter and have positive constant value.

This is delta rule due to Widrow and Hoff (1960) and is so called Widrow–Hoff rule. It is stated as adjustments made in weights of a neuron which is proportional to the product of error signal and input signal to that neuron.

3) **Resilient Propagation:** The Resilient propagation algorithm [16] has been designed to remove the shortcomings of gradient descent method where the change in weights Δw_{jk} is based on the learning rate η and the derivatives $\frac{\partial E}{\partial w_{jk}}$ of the error surface as expressed in (18). In resilient propagation, the weight update Δw_{jk} is done directly without taking into account the partial derivative. In this, each weight (either due to hidden layer or output layer) has its own individual update value Δ_{jk} which only determines the size of the weight update. The update value is designed according to the below given learning rule based on error *E(t)*

$$\Delta_{jk}(t) = \begin{cases} \eta^+ * \Delta_{jk}(t-1), & \text{if } \frac{\partial E(t-1)}{\partial w_{jk}} * \frac{\partial E(t)}{\partial w_{jk}} > 0 \\ \eta^- * \Delta_{jk}(t-1), & \text{if } \frac{\partial E(t-1)}{\partial w_{jk}} * \frac{\partial E(t)}{\partial w_{jk}} < 0 \\ \Delta_{jk}(t-1), & \text{else} \end{cases} \qquad (22)$$

Where $0 < \eta^- < 1 < \eta^+$

Once weight-value is adapted, the weight-update follows a very simple rule which is given as:

$$\Delta w_{jk}(t) = \begin{cases} -\Delta_{jk}(t), & \text{if } \frac{\partial E(t)}{\partial w_{jk}} > 0 \\ +\Delta_{jk}(t), & \text{if } \frac{\partial E(t)}{\partial w_{jk}} < 0 \\ 0, & \text{else} \end{cases} \qquad (23)$$

The value of weights in $(t+1)^{th}$ iteration is

$$w_{jk}(t+1) = w_{jk}(t) + \Delta w_{jk}(t)$$

a) Initializing Update Values: The initial update value $\Delta_{jk} = \Delta_0 = 0.1$ is a good choice. In order to avoid overflow and underflow in update value minimum and maximum values have been fixed at $\Delta_{min} = 1e^{-16}$ and $\Delta_{max} = 50.0$, respectively. The optimal choice for increase factor η^+ and decrease factor η^-, are 1.2 and 0.5, respectively.

b) Learning Mode: Resilient propagation algorithm works in batch mode. The update values and weights are changed after the presentation of entire training examples that constitute an epoch. In pattern recognition problems an MLP (back-propagation) is mostly trained with gradient descent method. Resilient propagation method has some advantages over gradient descent methods as:

1) It converges very fast on pattern recognition problems.
2) Its performance is not very sensitive to the settings of the training parameters which is very much dependent on the learning parameters and momentum in case of gradient descent.

The various issues concerned with training an MLP using back-propagation or resilient propagation are:

a) The size of the network *i.e.* the number of layers and number of neurons in each layer.

b) The optimal value of learning parameter η (particularly in case of back-propagation trained with gradient descent method).

J. Support Vector Machine(SVM)

SVMs are being extensively used for classification. The decision boundaries are defined purely on the basis of decision planes such as a hyper plane having a line like structure. As in case of neural network, learning in this case is also on the basis of a number of examples.

The foundation of SVM is due to Vapnik [26]. It has been successfully used for handwritten

recognition by Dong et al [27] and Oliveira et al [28]. A comprehensive tutorial on SVM for pattern recognition is given by Burges[24]. SVM is originally a two-class binary classifier used to deal with linear classes. But this behavior of SVM has been extended to multi-class classification problems too. In addition to this, its ability only to deal with linear classes has been extended to non-linear classes.

Support vector machines are derived from a class of hyper planes $z.u+c = 0;\ z \in R^n$ represents the normal to hyper plane and $c \in R$ represents the offset; corresponding to the decision function $g(u)= sign(z.u+c)$. An optimal hyper plane used for separating the two classes is one, which does not commence misclassification errors on data samples having small perturbations. The purpose is to find the optimal solution for all kinds of two class classification problems. The main cases of our interest are: 1) Linearly separable 2) Linearly non-separable and 3) Non-linearly separable.

Actually, the training examples appear in linear classifier, either for separable or non-separable cases, in form of dot product *i.e.* $u_i.u_j$. In case of non-linear support vector machine, the non-linear training data is mapped onto a higher dimensional feature space say, F_s, using a function Φ in which training data also appears in the form of dot product $\Phi(u_i).\ \Phi(u_j)$ such as

$$\Phi: R^n \rightarrow F_s \tag{24}$$

and the data in new space is in linear form which can be easily classified using a linear algorithm. If a kernel function K , given in (25), performs the above said task then only we need K in training algorithm

$$K(u_i, u_j) = \Phi(u_i).\Phi(u_j) \tag{25}$$

It classify a new pattern say, u, as

$$g(u)= sign\left(\sum_{i=1}^{U_S} \lambda_i d_i K(u,u_i) + c\right) \tag{26}$$

Where λ_i and c are evaluated by maximizing (27) and d_i is tag of u_i

$$L_D = \sum_{i=1}^{m} \lambda_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \lambda_i \lambda_j d_i d_j K(u_i, u_j)$$

$$\text{s.t.} \quad 0 \le \lambda_i \le C \tag{27}$$

Kernel is a non-linear function used for mapping a non-linear input data to a high dimensional data. Data in high dimensional feature space then can be classified by constructing a hyper plane using a linear function. In this way a kernel plays an important role in changing a linear support vector classifier to a non-linear support vector classifier and without this kernel function a support vector classifier can only be used to solve linear classification problems. The most common kernel is linear kernel and it is dot product between the input data to classify and a support vector member set *i.e.*

$$K(u, u_i) = u.u_i \tag{28}$$

This kernel contributes a linear classifier. A radial basis function support vector kernel is expressed as

$$K(u, u_i) = exp\left(-\frac{||u - u_i||^2}{2\sigma^2}\right) \tag{29}$$

where σ^2 is Gaussian width and must be chosen during training.

Polynomial kernel is a p^{th} order polynomial of the form given below

$$K(u, u_i) = (u.u_i + 1)^p \tag{30}$$

Where p is degree of polynomial and it must be chosen during training. Kernels (29) and (30) are non-linear and used to construct non-linear support vector classifier from linear support vector classifier. As discussed that the support vector classifier can be generalized to solve non-linear classification problems by substituting the dot product $u_i.u_j$ with a suitable symmetric kernel function $K(u_i, u_j) = \Phi(u_i).\Phi.(u_j)$.

1) Decomposition Methods: In real world applications, it requires a large training data set. The size of optimization problem is directly related to the size of training data set. The larger is the size of data set, the larger and complex will be the optimization problem. The size of matrix $M_{ij} = d_i d_j K(u_i, u_j)$ in (27) depends on the size of data set of training examples. If we have very large training data set, then it becomes very difficult to fit the matrix M in memory for optimization. The solution to this problem is to decompose the Quadratic Programming (QP) in a series of smaller problems. The commonly used decomposition methods for solving quadratic problem are chunking algorithm, Osuna's algorithm and sequential minimal optimization (SMO) [29].

2) Extension to Multi-Class: So far we have concentrated on the support vector classifier to deal with two classes. Since SVM basically is a binary classifier. Its behavior can be extended to deal with q class ($q > 2$) pattern recognition problems such as text categorization, digit recognition and character recognition, etc.

The various methods to deal with q class recognition problem using binary support vector machines are: one-against-one, one-against-others and directed acyclic graph(DAG) and all-together[25].

3) One-Against-One: In case of one-against-one methods, there is one SVM for each pair of classes. If we have q classes, then there will be $q(q-1)/2$ SVMs. Each SVM is trained independently for two classes, *i.e.*, s^{th} and t^{th} . To find the class of unknown new test example, u , we predict the value of $g_{st}(u)$, using (26) , for all $q(q-1)/2$ SVM.

$$g_{st}(u) = sign(z_{st}.\Phi(u) + c_{st}) \quad (31)$$

where subscripts s and t mean a binary SVM is trained with the examples of s^{th} and t^{th} classes.
Final decision about the class of u will be made on the basis of the voting from all the classifiers. Initially, each class has zero votes. If the value of $g_{st}(u)$ is positive then we increase the vote of class s by one otherwise we increase the vote of class t by one. The class of u is the class having maximum votes. A conflict arises in this case, when two or more patterns have equal votes. In such situations a class is selected having lower index.

Hsu et al[25] studied and compared the performance of some multi-class SVM implementation approaches on 10 practical problems and concluded that one-against-one perform the others. Moreover, with this approach the SVM classifier is easy to train.

III.CONCLUSION

There are various factors which affect generalization and these factors are size of data set, the discrimination ability of the feature type used to represent patterns to train classifier, physical complexity of problem at hand, optimality observed in training a classifier, and unknown parameters present in classifier. The various neural network based classifiers used for handwritten character recognition are radial basis function, self organization map, probabilistic neural network and multilayer perceptron. The RBF is faster in training phase but slower in test phase as it consists of large number of functional units in hidden layer. The PNN is suitable for learning based applications than generalization. The SOM is generally used for clustering as it learns in unsupervised manner. The MLP is most commonly used classifier among the neural network based classifiers. It is fast in test phase. An MLP trained with resilient propagation algorithm converges faster on pattern recognition problems as compared to an MLP trained with gradient decent method. It has been designed to remove the limitations of the gradient descent method. Its performance is also not much sensitive to parameter selection.

Among the other classifiers discussed here, the SVM classifier is robust in dealing with handwritten variations as its classification performance is better as compared to other classifiers. The k-NN based classifier is also mostly used in pattern recognition problems. This classifier is lazy and also called as instance based. This requires large storage space as whole data must be in memory in recognition phase. The performance of Parzen classifier depends upon the kernel function that depends upon the window width and the selection of this window width is very critical. The processing time and memory requirements of QD classifier are large. In addition to this the poor estimation in parameters degrades the performance of QD classifier.

REFERENCES

[1] O. D. Trier, A. K. Jain and T. Taxt, "Feature Extraction Method for Character Recognition – a Survey", Pattern Recognition, Vol. 29, No. 4, pp. 641-662(1996).

[2] S. Haykin, "Neural Networks A Comprehensive Foundation", Second Edition, and Pearson Education, Asia.

[3] A. K. Jain and R. P. W. Duin and J. Mao, "Statistical Pattern Recognition: A Review", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 22, No. 1, pp. 4-37(2000).

[4] R. C. Gonzalez and R. E. Woods, "Digital Image Processing", 2nd Ed., Pearson Education.

[5] T. Kawatani, "Handprinted Numeral Recognition with the Learning Quadratic Discriminant Function", Proceedings of the International Conference on Document Analysis and Recognition, pp. 14-17(1993).

[6] S. J. Raudys and A. K. Jain, "Small Sample Size Effects in Statistical Pattern Recognition: Recommendations for Practitioners", IEEE Transactions on Pattern Analysis and Machine Intelligence", Vol.13, No. 3, pp. 252-264(1991).

[7] Y. Hamamoto, S. Suchimura and S. Tomita, " On the Behavior of Artificial Neural Network Classifiers in High-Dimensional Space", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 18, No. 5, pp. 571-574(1996).

[8] U. Kressel and J. Schürmann, "Pattern Classification Techniques Based on Function Approximation", Handbook of Character Recognition and Document Analysis, World Scientific , pp. 49-78(1997).

[9] A. K. Jain, J. Mao and K. Mohiuddin, "Artificial Neural Networks: A Tutorial", IEEE Computer Special Issue on Neural Computing, pp. 31-43(1996).

[10] F. Ancona, A M. Colla, S. Rovetta and R. Zunino, "Implementing Probabilistic Neural Networks, Neural Computing & Applications", Vol. 5, pp. 152-159(1997).

[11] N. K. Bose and P. Liang, "Neural Network Fundamentals with Graphs, Algorithms and Applications", Tata McGraw-Hill, New Delhi.

[12] B. Yagnanarayan, "Artificial Neural Networks", Prentice Hall India, New Delhi, (2001).

[13] A. K. Jain, J. Mao and K. Mohiuddin, "Artificial Neural Networks: A Tutorial", IEEE Computer Special Issue on Neural Computing, pp. 31-43(1996).

[14] U. Kressel and J. Schürmann, "Pattern Classification Techniques Based on Function Approximation", Handbook of Character Recognition and Document Analysis, World Scientific , pp. 49-78(1997).

[15] T. M. Cover and P.E. Hart, "Nearest Neighbor Pattern Classification", IEEE Transactions on Information Theory, Vol. 13 , pp. 212-217(1967).

[16] M. Riedmiller and H. Braun, "A Direct Adaptive Method for Faster Back-propagation Learning: The RPROP Algorithm," Proceedings of the IEEE International Conference on Neural Networks, Vol. 1, pp. 586-591 (1993).\[17] S. J. Smith, M. O. Bourgoin, K. Sims and H.L. Voorhees, "Handwritten Character Classification using Nearest Neighbor in Large Database", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.16, No. 9, 915-919(1994).

[18] Zs. M. Kovics and R. Guerrieri, "Massively-Parallel Handwritten Character Recognition Based on the Distance Transform", Pattern Recognition, Vol. 28, No. 3, pp. 293-301(1995).
[19] S. O. Belkasim, M. Shridhar and M. Ahmadi, "Pattern Recognition with Moment Invariants: A Comparative Study and New Results", Pattern Recognition, Vol. 24, No. 12, pp. 1117-1138(1997).
[20] G. S. Lehal and C. Singh, "Feature Extraction and Classification for OCR of Gurmukhi Script", Vivek, Vol. 12, pp. 2–12(1999).
[21] S. Antani and L. Agnihotri, "Gujarati Character Recognition", Proceedings of the Fifth International Conference on Document Analysis and Recognition, Bangalore, India, pp. 418–421(1999).
[22] S. D. Connel, R.M.K. Sinha and A. K. Jain, "Recognition of Unconstrained On- Line Devanagari Characters", Proceedings of the International Conference on Pattern Recognition, Barcelona, Spain, Vol. 2, pp. 368-371(2000).
[23] C. V. Jawahar, M.N.S.S. K. Pavan Kumar and S. S. Ravi Kiran, "A Bilingual OCR for Hindi-Telugu Documents And Its Applications", International conference on Document Analysis and Recognition, Vol. 1, pp. 408-412(2003).
[24] C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition", Data Mining and Knowledge Discovery, Vol. 2, No. 2, pp. 121–167 (1998).
[25] C.-W. Hsu and C.-J. Lin, "A Comparison of Methods for Multi-class Support Vector Machines", IEEE Transactions on Neural Networks, Vol. 13, No. 2, pp. 415–425(2002).
[26] V. Vapnik, "The Nature of Statistical Learning Theory" Springer-Verlag, New Tork (1995).
[27] J.-X. Dong, A. Krzyzak and C. Y. Suen, "An Improved Handwritten Chinese Character Recognition System using Support Vector Machine", Pattern Recogniotion Letters, Vol. 26, No. 12, pp. 1849-1856(2005)
[28] L. S. Oliveira and R. Sabourin, "Support Vector Machines for Handwritten Numerical String Recognition" , Ninth International Workshop on Frontiers in Handwriting Recognition, Kokubunji, Tokyo, Japan, pp. 39-44(2004).
[29] T. Joachims, "Making Large-Scale SVM Learning Practical", In Advances in Kernel Methods- Support Vector Learning, B. Schölkopf, C.J.C. Burges, and A. J. Smola, Eds. Combridge, MA: MIT Press(1998).
[30] C.- L. Liu and M. Nakagawa, "Evaluation of Prototype Learning Algorithms for Nearest- Neighbor Classifier in Application to Handwritten Character Recognition", Pattern Recognition, Vol. 34, pp. 601-615 (2001).