Comparison Study of Different Pattern Classifiers

Ameet Joshi♣,
Shweta Bapna §,
Sravanya Chunduri§

Abstract

This paper presents a comparison study of the different parametric and non-parametric pattern classifiers that are commonly used for pattern recognition. An evaluation of combining the different classifiers is also presented. Three different data sets are employed for the comparison of the performance of the classifiers under study. The classification study is based on supervised classification. Results of individual pattern classifiers, classifier combinations as well as error estimates for the various data sets used are presented, along with the CPU-time consumption[*]. The evaluation results reported here might be useful when designing practical pattern classification systems for the type of applications (data sets) we considered in our study.

Index Terms

Combination of classifiers, dynamic classifier selection, local area accuracy, curse of dimensionality, jackknife, confidence intervals.

I. INTRODUCTION

The field of pattern recognition (or pattern classification) has a wide variety of commercial, medical and industrial applications. Some of the many applications include handwriting recognition, noise classification, face recognition, fingerprint recognition, biomedical image processing applications, mammography, speech recognizers, etc. With such a wide variety of applications it is not possible to come up with a single classifier that can give good results in all the cases. Hence, the classifier(s) adopted are highly dependent on the problem domain.

The objective of a classification system is to assign a pattern presented to it to a class using the feature vector (list of attribute values). There are a variety of classifiers used for pattern classification. The complexity of the classification problem is dependent on the variability of the feature values for patterns in the same class relative to the difference between feature values for patterns in different classes. Consequently the optimality of a classifier is dataset dependent. Therefore achieving optimal performance for a pattern recognition system is not necessarily consistent with obtaining the best performance for a single classifier. In practice, one might come across a case where no single classifier can make a classification with an acceptable level of accuracy. In such cases it might be better to pool the results of different classifiers to achieve the optimal decision accuracy. Each classifier can operate well on different aspects of the input feature vector. Under appropriate assumptions, combining multiple classifiers may lead to improved generalization performance when compared with any single constituent classifier.

We restricted the scope of our project only to compare some commonly used non-parametric pattern classifiers and a single parametric pattern classifier in view of the error estimates and time of execution. We evaluate the classification accuracy of four classifiers for three different datasets, which represent a wide range of possible raw data. We use both statistical and neural network based classifiers. The statistical techniques in our study include *k-nearest neighbor (k-NN)*, p*arzen window* and *Bayesian classifier*. Our neural classifier is a *simple multi-layer perceptron (MLP)*. The classifiers chosen are some of the common classifiers used in most practical applications. *k-NN* is widely used in pattern recognition due to its conceptual simplicity, general applicability and efficiency. We also explored the p*arzen window* method, which is though not commonly used because of its high computational requirements, but we believed and also proved that this method can be of substantial use in some kind of problems. The *Bayesian classifier* was chosen, as it is one of the simplest non-parametric classifiers that is said to give high classification accuracy.

♣ All the authors are currently graduate students in the Department of Electrical & Computer Engineering, Michigan State University.
[*] All the code was written in MATLAB^{TM} 6.0 and was executed on Mulder server at MSU.

In pattern classification, the most popular measure of performance is the misclassification rate, which is simply the percentage of cases misclassified by the model. We have used the same misclassification or error rate for our comparison of the four classifiers.

II. BAG OF CLASSIFIERS

The first step for us was to implement the classifiers we had chosen independently. Here is our bag of classifiers:

***k*-Nearest Neighbor**:
This classifier classifies a pattern *x* by assigning it to the class label that is most frequently represented among it's *k* nearest patterns. In the case of a tie, the test pattern is assigned the class with minimum average distance to it. Hence, this method is sensitive to the distance function. For the minimum average distance, the metric employed is the Euclidean distance. This metric requires normalization of all features into the same range. The *k-nearest neighbor* classifier is a conventional nonparametric classifier that is said to yield good performance for optimal values of *k*.

Parzen Window:
In this approach of classification a *d*-dimensional window is created around all the training samples and depending upon the number of patterns that belong to those windows the probability estimates of the different classes is made. Formally this can be stated as,

$$p_n(x) = \frac{1}{n}\sum_{i=0}^{n}\frac{1}{v_n}j\left(\frac{x - x_i}{h_n}\right)$$

Where $\boldsymbol{v_n}$ is a d-dimensional hypercube in feature space. Here $\mathbf{j}$ is a general probability distribution function. And p_n(X) is the probability that the pattern belongs to the given class. It remains to designer to choose the form of $\mathbf{j}$ and for all practical purposes Gaussian distribution is chosen. The windows are centered on the training points, hence, the mean is known, but there is no predefined method for the determination of the variance. Hence depending upon the problem under consideration the variance is chosen to maximize the classifier performance, or in other words minimize the error rate. One can think of basing the selection of variance on the range of values of features in various dimensions as the most intuitive approach, but again the exact relation needs to be determined by trial and error. Thus from these theoretical considerations we have come up with an algorithm, which is giving an optimum performance in our datasets.

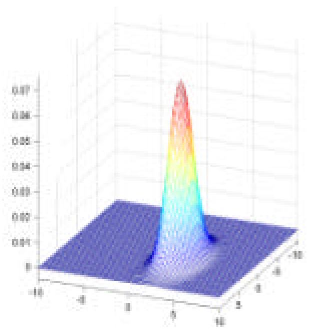

Figure 1 A sample of a parzen window generated in 2 dimensions emphasizing the fact that its variance in both the dimensions is different.

<u>*Algorithm*</u>

1. *Divide the given dataset into test data and training data using bagging methods. (The methods used for this will be discussed later in this report).*
2. *Generate the covariance matrix for the Gaussian distribution as a diagonal matrix (assuming there is no correlation in between the classes) with only diagonal elements. Each element will be calculated as ratio of the range of values of the feature in that dimension and the total number of samples. This value is representative of the variance in that dimension.*
3. *Take a test sample.*
4. *Develop parzen windows around all the training samples.*
5. *Find the combined probability at the test pattern for all the classes as d1, d2...*

6. *The class, which has maximum combined density at the test pattern, will be assigned to the test pattern.*
7. *Select next test sample and repeat the steps from 3 through 5, to classify it.*
8. *Stop the classification after the test samples are over.*

Bayesian Classifier:
The quadratic discriminant function using the Bayesian approach is the most general approach in the theory of supervised parametric classifiers. We included it in our bag of classifiers in order to compare its performance with the parametric, supervised classifiers. The decision boundaries obtained by these classifiers can be extremely complicated when dealing in *d*-dimensions. Though this method is highly computation intensive, most of the computation of generating the discriminant function is done off-line. As the quadratic discriminant function contains a large number of parameters, this approach is prone to the "*curse of dimensionality*", and the performance can be drastically affected if the number of training samples is small. We will be considering all these issues in the discussion when we show the results of the classification on the datasets.

Decision Tree:
The decision tree classifier uses a layered, or hierarchical approach to classification. At each level of the tree the attributes of a measurement are matched to a number of mutually exclusive nodes. The leaf nodes assign classes to the measurement. The classification of a measurement therefore involves a sequence of tests, with each successive test narrowing the interpretation. The sequence of tests for the classifier is determined during a training period. Given some training data *T*, the ideal solution would test all possible sequences of actions on the attributes of *T* in order to find the sequence resulting in the minimum number of misclassifications. The software we used for the decision tree classifier is C5.0. It needs four types of files for generating the decision tree for a given data set, out of which two files are optional. The first file is the *.names* file. It describes the attributes and classes. The first line of the *.names* file gives the classes, either by naming a discrete attribute (the *target* attribute) that contains the class value (as in this example), or by listing them explicitly. The attributes are then defined in the order that they will be given for each case. The attributes can be either explicitly or implicitly defined. The value of an explicitly defined attribute is given directly in the data. The value of an implicitly defined attribute is specified by a formula. In our case all our data attributes are explicitly defined. The second file is the *.data* file. It provides information on the *training* cases from which C5.0 will extract patterns. The entry for each case consists of one or more lines that give the values for all explicitly defined attributes. The `?' mark is used to denote a value that is missing or unknown. We chose datasets that had no missing features. Also, `N/A' denotes a value that is not applicable for a particular case. The third file used by C5.0 consists of new test cases on which the classifier can be evaluated and is the *.test* file. This file is optional and, if used, has exactly the same format as the *.data* file. We gave a *.test* file for all our data sets. The last file is the *.costs* file. This file is also optional and sets out differential misclassification costs. In some applications there is a much higher penalty for certain types of mistakes. In our usage we haven't included this file simply because it wasn't specified for any of the data sets we used. After the program is executed for a dataset we get the output results for both the training and testing data. A confusion matrix is generated showing the misclassifications. Another function implemented by C5.0 is *boosting* which we will describe in a later section.

Multi-layer Perceptron (MLP):
The multi-layer perceptron classifier is a basic feedforward artificial neural network. We have used a single hidden layer initially for simplicity (simplifies choosing the number of neurons) and then went for two hidden layers for better classification performance. The hidden units were chosen differently for each data set. The number of hidden neurons was found out experimentally over a number of trials. A rule of thumb [1] is to choose the number of hidden neurons such that the total number of weights in the net is roughly *n*/10, *n* being the total number of training points. The neural network was trained using the back-propagation algorithm, According to [3], the *multi-layer perceptron* trained using the back-propagation learning algorithm approximates the optimal discriminant function defined by Bayesian theory. The outputs of the *MLP* approximate posterior probability functions of the classes being trained. We used MATLAB 6.0 Neural Network Toolbox to implement our *MLP* classifier. The activation function we used was the hard-limit transfer function used in Matlab.

III. DATA SETS

We have chosen three different generic data sets for our study. They were acquired from UCI Machine Learning Databases Repository. In Table 1 we give some information about the samples, features, and classes for the three datasets. The following briefly describes the data sets chosen.

Balance Scale Weight and Distance Dataset: This data set was generated to model psychological experimental results. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The attributes are the left weight, the left distance, the right weight, and the right distance. The correct way to find the class is the greater of (left-distance * left-weight) and (right-distance * right-weight). If they are equal, it is balanced. There are a total of 625 samples and each sample consists of 4 numeric features.

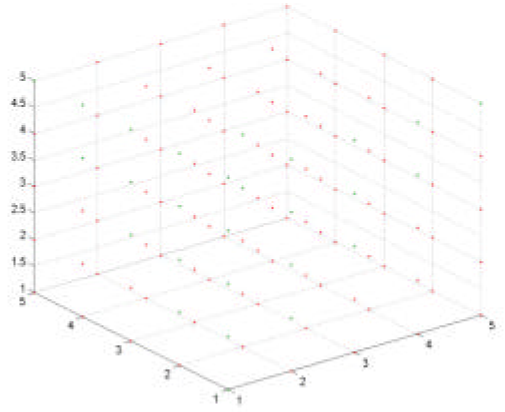

Figure 2 ***Using Principal Component Analysis (PCA), the initial four-dimensional Balance Scale data is represented in three-dimensions.***

Image Segmentation Dataset: The samples for this data set were drawn randomly from a database of seven outdoor images. The images were hand segmented to create a classification for every pixel. There are a total of 2310 samples (210 training samples and 2100 test samples provided separately) and each sample consists of 19 numeric features.

Wine Recognition Dataset: This data set contains the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of thirteen constituents found in each of the three types of wines. These constituents make up the 13 features for each of the total 178 samples.

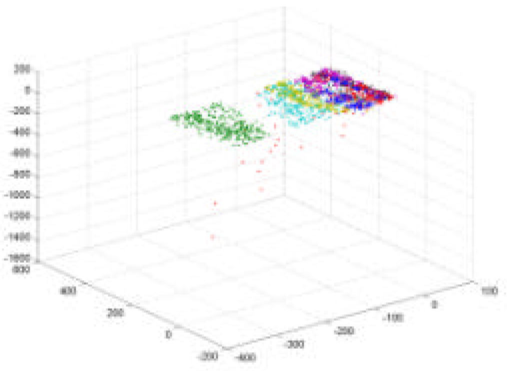

Figure 4 Using Principal Component Analysis (PCA), the initial nineteen-dimensional Image Segmentation data is represented in three-dimensions.

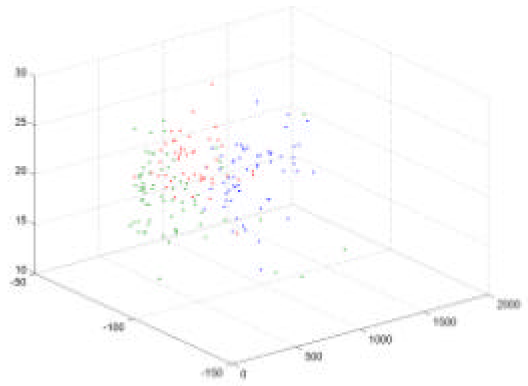

Figure 3 Using Principal Component Analysis (PCA), the initial thirteen-dimensional Wine data is represented in three-dimensions.

The 3-D representation of the data sets using PCA is shown in figures 2, 3, and 4 respectively.

Table 1 A brief summary of the selected datasets.

	Balance Scale	Image Segmentation	Wine
Classes	3	7	3
Features	4	19	13
Total Samples	625	2310	178
Samples/Class	1-49, 2-288, 3-288	330	1-59, 2-71, 3-48
Nature of features	Continuous	continuous	continuous

IV. THE APPLICATION OF INDIVIDUAL CLASSIFIERS ON DATASET

We will discuss each dataset and its classification by all the classifiers individually first and then we will discuss their combination and other methods to improve the overall performance.

Balance-Scale Data:

This dataset is unique in its own way. First form the figure1 it can be seen that the distance between any two samples is same. Second, the samples from one class ‘right’ and class ‘between’ are equal and more in comparison to the class ‘left’ with 288,288and 49 samples respectively. Moreover, the ‘between’ class doesn’t have normal distribution due to sparse and less data. There are total four features in this dataset and they all are continuous ranging from 1 to 5.

We tried both parametric and non-parametric methods. In parametric we used Bayesian Quadratic Classifier. Here the mean and covariances for each class are assumed to be unknown. The data is randomly divided into 50% training and 50% test. This classifier has two phases. In the training phase the means and covariances are computed of the training set. In the test phase, all the samples from the test data is passed to discriminant function of each class and is classified to the class that has maximum discriminant function value. We found the percentage error rate to be 8.11% i.e. an accuracy of 91.89 % .The misclassification is high due to the less reliability of discriminant function for class ‘between’ as it does not have normal distribution. In non-parametric approach, the *k-NN* gave less performance than the quadratic classifier. The reason is again the distribution of the data. The p*arzen window* also couldn’t perform better than the parametric approach with an accuracy of 86.28%. *MLP* also gave an accuracy of around 90.4%. This percentage was in accordance with the result [3,4]. Decision tree C5.0 gave an classification accuracy of 86.38%. So, in this data set parametric approach performed better than the non-parametric ones.

Table 2 The results of the classifiers on Balance Scale Dataset.

	Bayesian Classifier	1-NN Classifier	k-NN Classifier (k = 5)	Parzen window Classifier	Multilayer Perceptron	C5.0
Average Error	8.11%	22.4%	25.3%	13.72%	9.6%	13.62%
Average time of execution (sec)	0.881	15.94	46.74	48.09	2.5	0.0

The Bayesian classifier gave the best results in this case. The *k-NN* classifier was the worst with error rate of above 20%.

Wine data:

This dataset has three classes and 13 features and 178 total samples. The accuracy achieved for this dataset by quadratic classifier is 97.33 % .The optimum accuracy of Knn is 97.84 %. The Parzen classifier gave is 99.10 % accuracy, which is the highest achieved. The accuracy achieved for MLP is 92.45%, which is not more than that of p*arzen* window. Decision tree C5.0 gave an accuracy of 80.7 % on test set. We did the Principal Component Analysis of the dataset but found that all the features were important. We have used Jackknife in *k-NN* and *parzen windows* classifier.

Table 3 The results of the classifiers on Wine Dataset.

	Bayesian Classifier	1-NN Classifier	k-NN Classifier (k = 5)	Parzen window Classifier	Multilayer Perceptron	C5.0
Average Error	2.67%	2.16%	2.80%	0.90%	7.55%	19.3%
Average time of execution (sec)	0.51	15.94	46.74	7.3	2.5	1.0

Image Segmentation Data:

The image segmentation was the largest dataset that we considered. It had total of 2310 samples out of which 2100 were test samples and 210 were training ones. The results of applying the classifiers on this dataset are given in Table 4. The classifiers like *parzen* window, which were highly computing intensive were tested for 5 times but due to the fact that the data set was large enough there was little variance in the observed values of errors. The original data set was consisted of 19 features; hence there was a good scope of feature extraction. We applied PCA to reduce the number of features, which was also going to help reduce the computation time for especially parzen window classifier. We tried to reduce the features upto 3, features. The results for the parzen window were especially interesting as it helped to reduce the time of execution drastically. It clearly shows the advantages using PCA.

Table 4 Summary of Parzen window results with the use of feature extraction.

Features Extracted	None[*]	5	7	9	11
% Accuracy	7.10%	13.59%	3.03%	2.68%	2.68%
Time of execution (sec)	3545	178	812	2131	2403

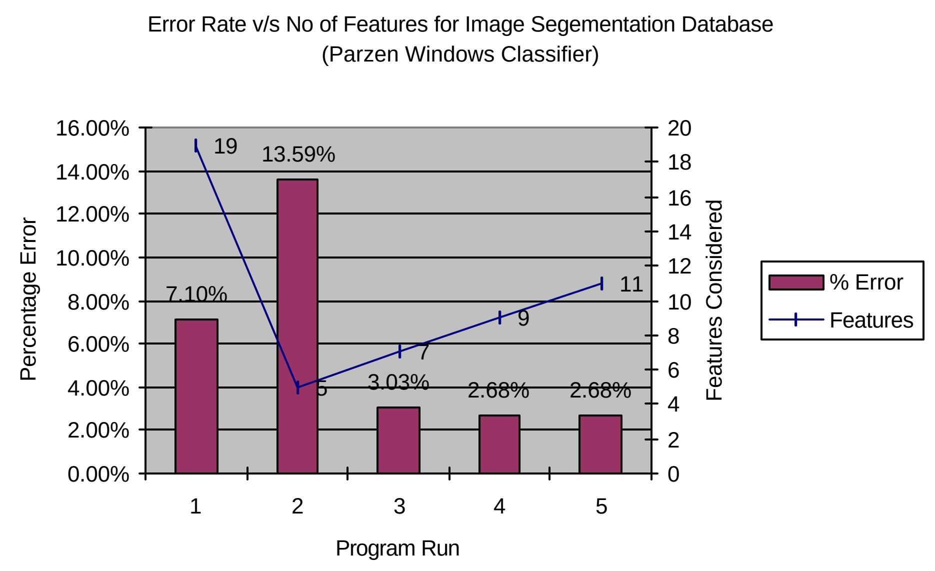

Chart 1 This chart summarizes the results of parzen window classifier with the use of feature extraction.

[*] Here PCA was not used and hence all the 19 initial features were directly used to classify.

Table 5 The results of the classifiers on Image Segmentation Dataset.

	Bayesian Classifier	1-NN Classifier	k-NN Classifier (k = 5)	Parzen window Classifier	Multilayer Perceptron	C5.0
Average Error	10.95%	7.16%	7.22%	2.68%	11.7%	7.6%
Average time of execution (sec)	0.5	240	748.94	2403	101.23	1.1

V. BAGGING & BOOSTING

After evaluating the raw individual performances for optimizing them we used primitive bagging techniques, as jackknife, which is also called as the leave one out method. The boosting was observed in C5.0 only.

Jackknife:
In this method the accuracy of a given algorithm is estimated by training the classifier n separate times, each time using the training set D from which a different single training point has been deleted. Each resulting classifier is tested on the single deleted point, and the jackknife estimate of the accuracy is then simply the mean of these leave-one-out accuracies. Here the computational complexity may be very high, especially for large n. The jackknife, in particular, generally gives good estimates, because each of the n classifiers is quite similar to the classifier being tested (differing solely due to a single training point). A particular benefit of the jackknife approach is that it can provide measures of confidence or statistical significance in the comparison between two classifier designs.

Boosting:
The goal of boosting is to improve the accuracy of any given learning algorithm. In principle, boosting can be defined as a procedure of adding new component classifiers to form an ensemble whose joint decision accuracy is arbitrarily high. A feature available in C5.0 is *adaptive boosting*. The idea is to generate several classifiers (either decision trees or rulesets) rather than just one. When a new case is to be classified, each classifier votes for its predicted class and the votes are counted to determine the final class. First, a single decision tree or ruleset is constructed from the training data. This classifier will usually make mistakes on some cases in the data. When the second classifier is constructed, more attention is paid to these cases in an attempt to get them right. As a consequence, the second classifier will generally be different from the first. It also will make errors on some cases, and these become the focus of attention during construction of the third classifier. This process continues for a pre-determined number of iterations or *trials*. The number of trials is figured out through trial and error.

VI. COMBINATION OF MULTIPLE CLASSIFIERS (CMC)

We concentrated more on the combination of the multiple classifiers (CMCs), and evaluated the results obtained.

Table 6 The general summary of all the classifiers applied on all the datasets

Datasets / *Classifiers*	Balance-Scale	Wine	Image Segmentation
Bayesian	91.89%	97.33%	89.05%
1-NN classifier	87.60%	97.84%	92.84%
k-NN classifier	84.70%	97.20%	92.78%
Parzen Window	84.49%	99.10%	97.32%
MLP	90.4%	92.45%	88.30%
C5.0 (Decision Tree)	86.38%	80.70%	92.40%

Table 1 indicates that the performances of the individual classifiers on the different datasets indicate that there is no single classifier which is performing best, even the performance of the professional package C5.0 is not overall best.

Thus the combination of multiple Classifiers (CMC) is called for. There are various ways one can think of combining the classifiers.

The most simple way is to find the overall performance of the classifiers and choose the one, which performs best on the given dataset. We call it a naïve CMC. Though this may not really seem to be a CMC, for a general case it is indeed a better performer than individual classifiers. The output of this combination will simply be the best performance in each column in Table 6.

The second method, also called as parallel combination, for each dataset all the classifiers are used and voting is taken. The class getting maximum votes from the individual classifiers will be assigned to the test sample. This method intuitively appears to be better than the previous one. When we actually tried this on all the three datasets, the results were quite anticipatory.

Table 7 shows the actual performance of this combination.

The third method, also called as DSC-LA (dynamic selection of classifiers based on the local accuracy estimates) from Kevin Woods et. al. [6]. This method further investigates the use of classifiers on the particular test sample. In the local neighborhood of the test sample the performance of all the individual classifiers on training samples is evaluated, and the one, which performs best, is chosen for the decision-making. We used the following variant of this concept,

Algorithm

1. *Take a training sample.*
2. *Find the k nearest neighbors (we chose 5) to that sample in training set.*
3. *Apply all the classifiers on those points and get the accuracy of component classifiers.*
4. *Choose the one, which performs best, and apply that one to classify the test sample*
5. *Stop after the whole of test samples are exhausted.*

As can be anticipated this algorithm is highly computation intensive and recorded maximum computation time. But also the results obtained were slightly better that the parallel paradigm of CMC. The

Table **7** shows the results.

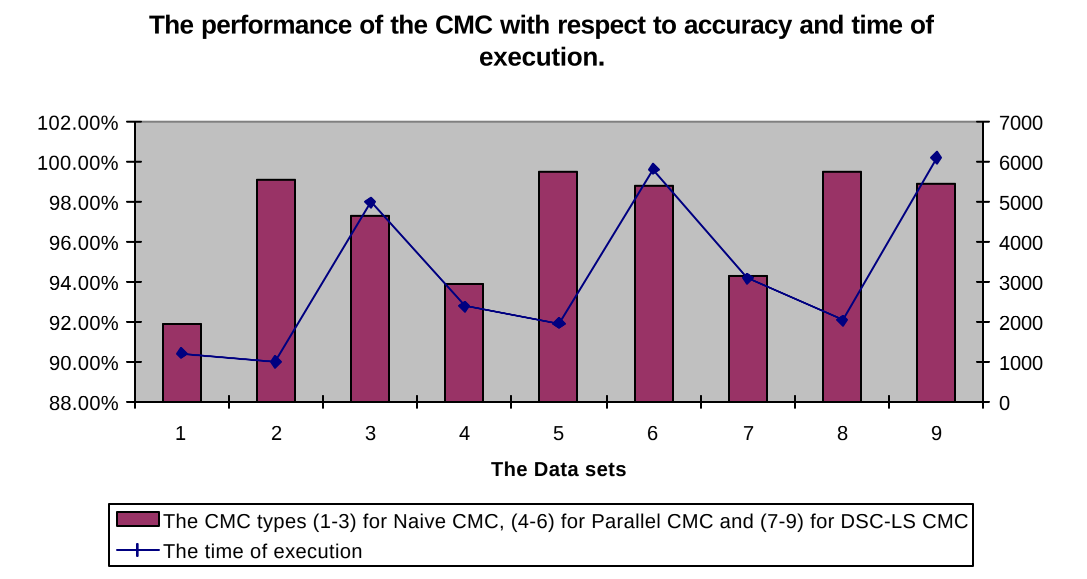

Chart 2 The summary of CMC combinations

Table 7 The summary of the classifier combinations and their performances

Type of CMC	Data Set	Balance Scale	Wine	Image Segmentation
	# Of samples	625	178	2310
Naïve	% Accuracy	91.89%	99.10%	97.32%
	Time of execution (sec)	1234	1020	4981
Parallel	% Accuracy	93.89%	99.49%	98.78%
	Time of execution (sec)	2393	1978	5809
DCS – LA	% Accuracy	94.30%	99.51%	98.88%
	Time of execution (sec)	3093	2039	6092

VII. GENERALIZATION OF THE PERFORMANCES OF THE CMC'S

After evaluating the performances of the individual classifiers followed by the CMCs, it was mandatory to generalize the performances so as to have a general idea about the performance somewhat independent of the given datasets. Table 8 gives the brief summary [5] of the 95% confidence intervals of the CMCs.

Table 8 The generalization of the classifier performance using the 95% confidence interval ranges.

Data Set	Balance Scale		Wine		Image Segmentation	
Type of CMC	Min	Max	Min	Max	Min	Max
Naïve	89.91%	94.4%	98.5%	99.70%	97.02%	98.0%
Parallel	93.0%	94.2%	98.7%	99.8%	98.5%	99.2%
DCS- LA	93.45%	95.1%	99.0%	99.9%	98.57%	99.26%

VIII. SUMMARY AND CONCLUSION

During the overall development of the project the well know fact that there is no one single classifier, which can outperform all the other classifiers in all the datasets was consistently proved. This initiates the unending quest for optimality in the classifier combination. We tried our best to achieve the optimum results on the given datasets with the given classifiers. The bagging and boosting (in C5.0) certainly improved the overall performance, but the real important aspects that we identified from the observations were use of feature extraction using PCA, especially when the source dataset is high dimensional. The combination of multiple classifiers certainly improved the accuracy, but at the cost of heavy increase in computation time. But there can always be a better solution, and again there will be no theoretical proof that that solution will also be optimum.

IX. REFERENCES

1. R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification* 2nd Ed, John Wiley & Sons Inc., 2000
2. R.P.W. Duin, "A Note on Comparing Classifiers", *Pattern Recognition Letters* 17 (1996)
3. D.W. Ruck, S.K. Rogers, M. Kabirsky, M.E. Oxley, and B.W. Suter, " The Multi-Layer Perceptron as an Approximation to a Bayes Optimal Discriminant Function"; *IEEE Transactions on Neural Networks*, vol. 1, no. 4, 1990.
4. E.W. Wan, "Neural Network Classification: A Bayesian Interpretation"; *IEEE Transactions on Neural Networks*, vol. 1, no. 4, 1990.
5. Hyperstat online Confidence Intervals, http://www.davidmlane.com/hyperstat/confidence_intervals.html
6. Kevin Woods, W.Philip Kegelmeyer Jr., Kevin Bowyer, "Combination of Multiple Classifiers Using Local Area Estimates"; *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 4, 1997.
7. C5.0: An Informal Tutorial, http://www.rulequest.com/see5-unix.html