Handwritten Digit Classification

Abrita Chakravarty William Day

Michigan State University
Email: chakrav7@msu.edu, daywilli@msu.edu

Abstract

The aim of this project was to evaluate the effectiveness of various types of classifiers in recognizing handwritten digits. It has been shown in pattern recognition that no single classifier performs the best for all pattern classification problems consistently. So the goal of the project was to experiment with different classifiers and combination methods and evaluate their performance in this particular problem of handwritten digit recognition. This report discusses the different classifiers used in the project, the rationale for their choices and the results obtained by their individual and combined performances.

I. INTRODUCTION

The task of handwritten digit recognition, using a classifier, finds use in many places such as - postal mail sorting to read zip codes on mail, processing bank checks to read amounts on the checks, numeric entries in forms filled up by hand (for example - tax forms) and so on. The challenges faced in solving this problem are varied. The handwritten digits are not always of the same size, thickness, or orientation and position relative to the margins. However these problems were left out of the scope of this project. Our goal was to study the effectiveness of various statistical pattern classification methods, by experimenting on the MINIST data set of images of hand written digits (0-9). The MNIST data set is composed 60,000 training images and 10,000 testing images. Each image is a 28 x 28 grayscale (0-255) labeled representation of an individual digit. The general problem we predicted we would face in this digit classification problem was the similarity between the digits like 1 and 7, 5 and 6, 3 and 8, 9 and 8 etc. Also people write the same digit in many different ways - the digit '1' is written as '1', '1', '1' or '1'. Similarly 7 may be written as 7, 7, or 7. Finally the uniqueness and variety in the handwriting of different individuals also influences the formation and appearance of the digits.

II. REVIEW OF DIGIT RECOGNITION APPROACHES

From the literature we reviewed we found that the highest accuracies achieved by classifiers on the MNIST database were 99.05%, by individual classifiers and multiple classifier systems, and 99.3% (boosted), by multiple neural networks [1] [2].

III. CLASSIFIERS USED FOR THIS PROJECT

The Linear Classifier was chosen for its ease of implementation and also to give us a baseline against which we could evaluate the other classifiers. The linear classifier also helped to gauge the effectiveness of selecting various numbers of principal components to be used as features, after carrying out PCA on the raw data. The Quadratic Classifier was an improvement on the linear classifier and proved to be one of the simplest yet best performing classifiers in our project. The K-Nearest Neighbor classifier was chosen to experiment with a non-parametric method in the absence of a known distribution for the data. The Neural Network was chosen as the fourth classifier, as its performance had been reported as among the best of all the classifiers used on the MINST dataset. Finally we used an ensemble method, consisting of a simple voting scheme, to combine the results of the component classifiers in an effort to improve the performance.

IV. TOOLS USED FOR THIS PROJECT

MATLAB was extensively used for all coding purposes in this project. The rich library of math functions available in MATLAB, however came at a high cost in terms of computing resources required. Each 8-bit pixel and each training label was represented as a double precision floating point number. So with 20 bytes per pixel, 784 pixels per image and 60000 images we required 60,000 x 784 x 20 = 940,800,000 bytes of memory. With another 1,200,000 bytes for the training labels, a total of 942,000,000 bytes of memory was required just to load the training data. The test set required another 157,000,000 bytes giving us over 1 gigabyte of just data. Our programs in MATLAB required working with the data in the memory instead of accessing it from data files. To encounter the memory constraint problem, we first implemented our algorithms using smaller data sets of 15000 images and once the implementations were successful, the classifiers were trained on the entire dataset of 60000.

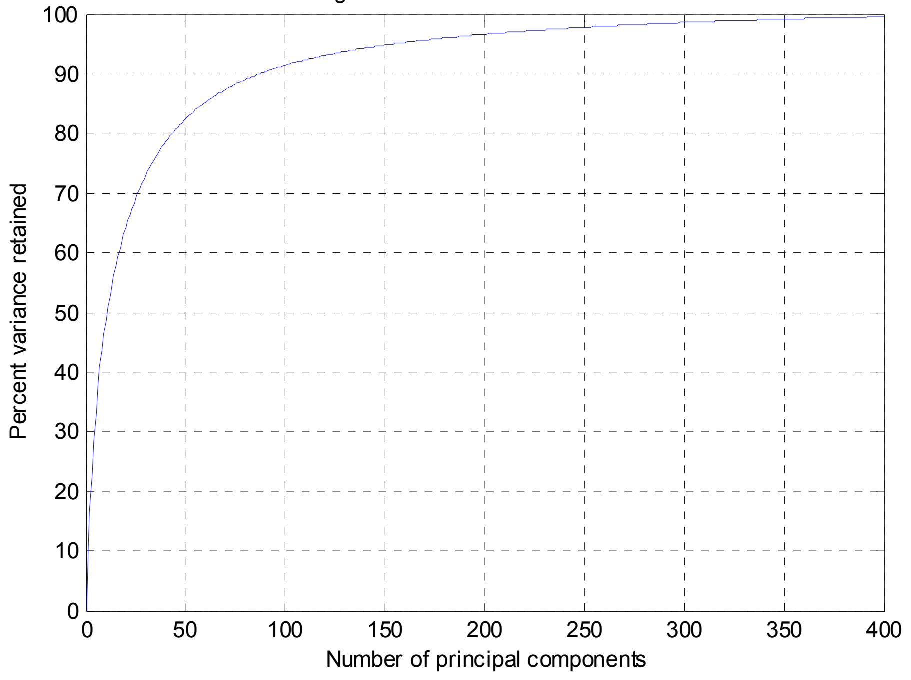
We implemented Principal Component Analysis and both the Linear and Quadratic classifiers in MATLAB. For the K-NN classifier we used PRTools toolbox for MATLAB, version 4.0 [9]. The Neural Network was again implemented using the Neural Network toolbox in MATLAB. Finally the Ensemble Classifier was a simple MATLAB program to carry out a voting between the results of the component classifiers.

V. RESULTS

V.A) PRINCIPAL COMPONENT ANALYSIS

PCA was used to reduce the dimensions of the dataset from 784 to a lower value – for ease of computation as well as to avoid the curse of dimensionality. Determining the best size for the feature set proved to be an important decision. We found 67 features have the same value for each sample, so an absolute maximum of 717 features would be useful. After performing PCA on the data, we found that 331 features retain over 99% of the variance. 154 features retain 95% of the variance, which was our benchmark from [5]. Other values we examined were 196 features (1/4th of 784), and 49 (1/16th of 784). The following figure (Fig 1) shows the amount of variance retained by the different numbers of principal components.

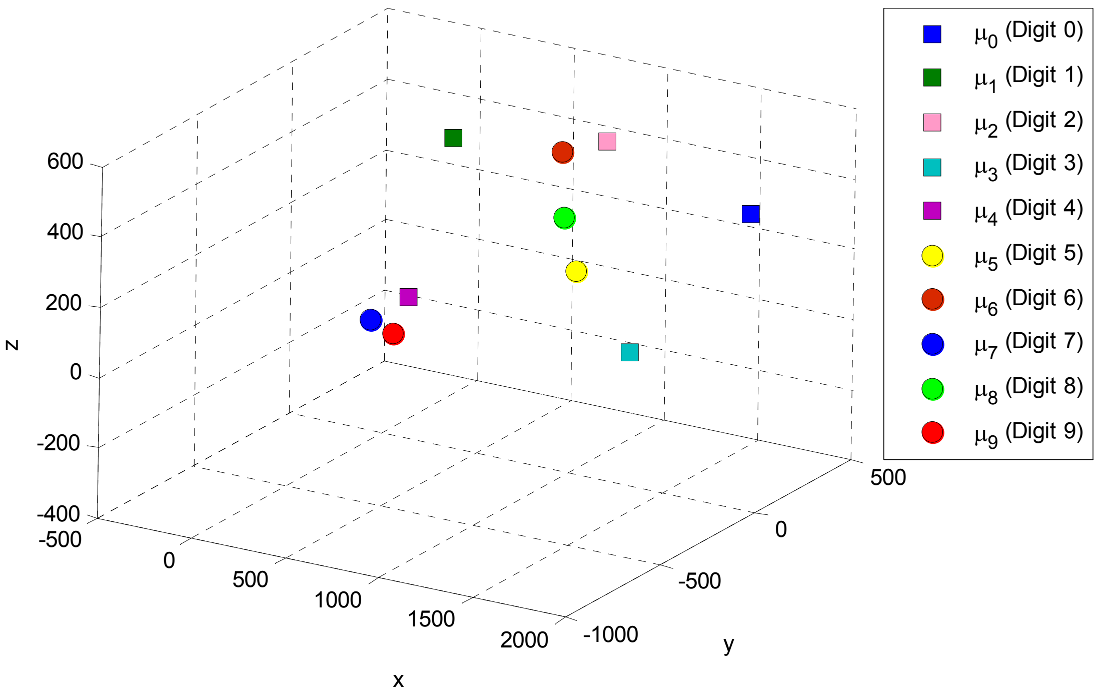
Fig 1: Variance retained after PCA



V.B) VISUALIZING THE DATA

Visualizing the data was important in giving us a perception of how the different classes were really distributed in problem space. Projecting the data in 2 and 3 dimensions, we found that the first 2 principal components retain 16.80% of the original variance and the first 3 retain 22.97%. Plotting all of the data points did not give us a clear indication of separation between the digits, so we plotted the class-means to obtain a more legible graphical representation of the data (Fig 2).

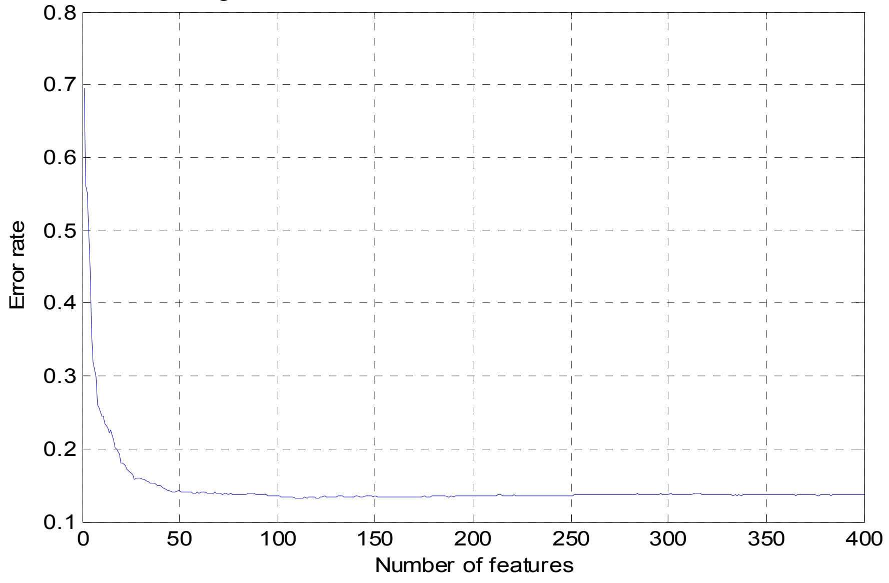
Fig2: Means considering 22.9697% of the principal components



V.C) LINEAR CLASSIFIER

The Linear Classifier was the fastest to implement. Initially we used 154 features because that retained 95% of the original variance. However using 154 features, gave us an error rate of 13.40%. We subsequently found that both 111 and 121 features (corresponding to 92.44% and 93.18% of the original variance) give better error rates of 13.23%. We generated the following graph (Fig 3) to look for a possible curse of dimensionality. However as seen in the graph, increasing the number of features does not increase the error rate of the linear classifier beyond a point, the error rate stabilizes at a plateau.

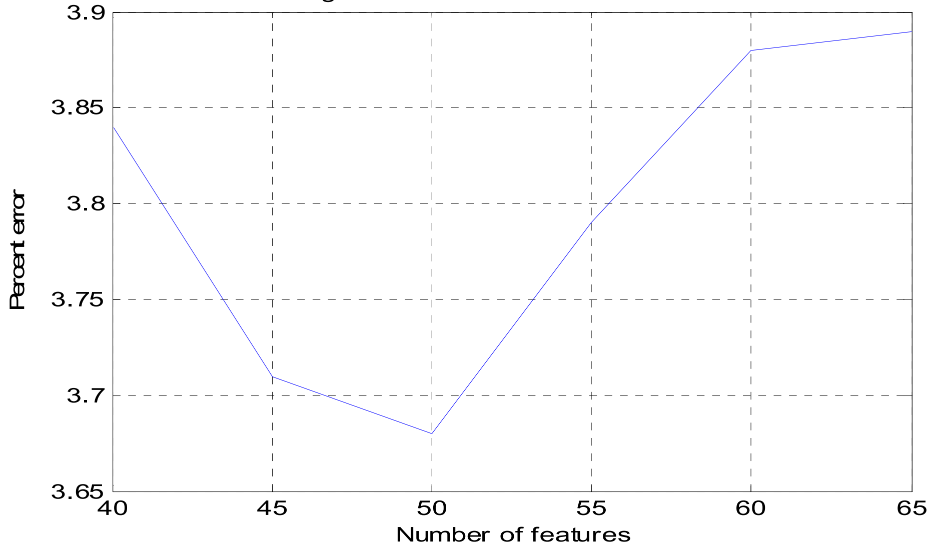
Fig 3: Number features vs. Error rate - Linear classifier



V.D) QUADRATIC CLASSIFIER

The Quadratic Classifier was also fairly easy to implement in MATLAB. We found that by using the first 50 principal components after performing PCA on the dataset, we were able to achieve a 3.68% error rate. We tried to use MDA on the 50 feature data to reduce the between-class-scatter, in hope of improving the performance of the quadratic classifier. However, the error rose to above 5%, so we returned to using only PCA. The following figure (Fig 4) shows the changing error rate for different number of principal components used in the quadratic classifier.

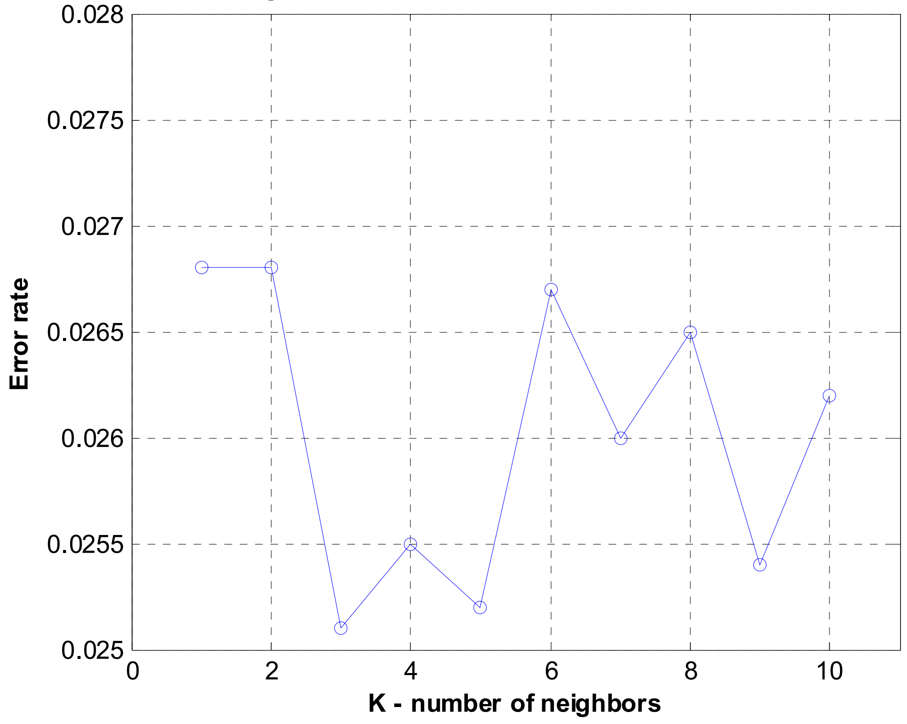
Fig 4: Quadratic classifier with PCA



V.E) K-NN CLASSIFIER

We initially tried to implement the 'Branch and Bound Algorithm' [10] in MATLAB to build our own K-NN classifier. We were able to cluster the data, by implementing the K-Means clustering algorithm into a 3-level, 3-node per level tree with average cluster sizes at levels 1, 2, and 3 being 4,998, 1,666, and 555. However this classifier gave a high error rate of over 80%. So the KNN classifier from PRTools [9] was used instead. We experimented with different values of K and found the 3 Nearest Neighbors rule worked best, giving an error rate of 2.51%. The following figure (Fig 5) shows the error rate versus the number of neighbors used for the KNN classifier.

Fig 5: Error rates for different K-NN classifiers



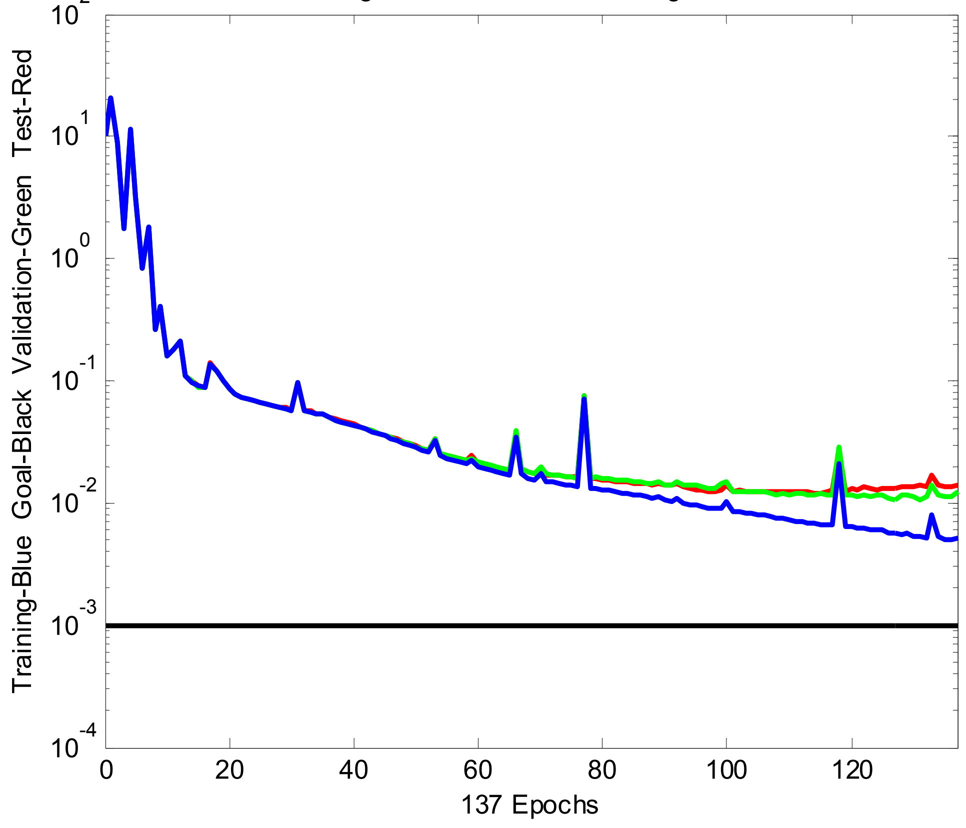
V.F) NEURAL NETWORK CLASSIFIER

We used the MATLAB neural network toolkit to implement a 300-100-10 neural network that was able to achieve a 5.32% error rate after training for 87 epochs, LeCun reports 3.05 for this same configuration [3]. We trained MATLAB on the same network without normalizing the data and received a 16.15% error after training for 96 epochs, so normalizing the data is important for MATLAB's neural network toolkit. With a little experimenting we were able to achieve a 4.47% error rate with a 300-120-67-10 network.

MATLAB offers several back propagation training functions. The MATLAB documentation states the default *trainlm* is the fastest but has high memory requirements. The *trainbfg* function is slower but requires less memory than *trainlm*, and *trainrp* is even slower but requires even less memory than *trainbfg*. Even on the more powerful servers at our disposal we were memory constrained requiring the slowest training function *trainbfg*.

The following graph (Fig 6) shows the learning rate for the network. The y-axis is the Mean Square Error (MSE), the 3 plots show the MSE for the training data, validation data, and the test data. The training for the network was stopped when the MSE for the validation data did not show improvement over 5 epochs.

Fig 6: Neural Network training rate



V. G) ENSEMBLE CLASSIFIER

Our ensemble method was a simple voting algorithm. We aggregated the predicted labels from each classifier, and used MATLAB's *mode* function to determine a vote. In case of a tie, the predicted label from the best classifier with the lowest error rate was chosen. This allowed us to break ties in favor of the superior classifiers. This ensemble method gave us an error rate of 2.57%. However, removing the worst performing Linear Classifier from the Ensemble, we were able to improve the error rate to 2.34%. The ensemble classifier thus gave the best results in our project.

V. H) SUMMARY OF RESULTS

CLASSIFIER	PARAMETERS	ERROR RATE
Linear	PCA reduction to 111 components	13.23%
Quadratic	PCA reduction to 50 components	3.68%
K-NN	K=3	2.51%
Neural Network	[300 120 67 10]	4.46%
Ensemble	Voting of 4 classifiers Voting of 3 classifiers	2.57% with Linear 2.34% without Linear

VI. CONCLUSION

This project helped us to learn and appreciate the effectiveness of 4 different classification methods, as well as their combination, in solving a difficult problem of pattern recognition. Dimensionality reduction was an important step in classifying these patterns of high dimensions. Visualizing the data, using the first two or three principal components obtained after PCA, gave us a better perception about the distribution of the samples. We experimented with both simple and complex classifier designs, implemented them in MATLAB and compared their performances. Even with classifiers of very simple design (e.g. Quadratic classifier); it is possible to obtain a fairly good performance. A combination of classifiers seems to improve the performance over that of a single component classifier. This project allowed us to gain confidence in our knowledge of well-known classifiers and allowed us to put theory into practice.

REFERENCES

[1] Y. Lecun et al., “Comparison of learning algorithms for handwritten digit recognition,” in *International Conference on Artificial Neural Networks, France*.

[2] C. Liu, K. Nakashima, H. Sako, and H. Fujisawa, “Handwriting recognition using state-of-the-art techniques,” in *Proceedings eigths International Workshop on Frontiers in Handwriting Recognition*, 2002.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *IEEE*, vol. 86, no. 11, pp.2278–2324, November 1998.

[4] R.Duda, P.Hart, and D.Stork, “Course text,” in *Pattern Classification*, 2001.

[5] W. Zhaoand A. Krishnaswamy, R. Chellappa, D.L.Swets, and J.Weng, “Discriminant analysis of principal components for face recognition,” in *International Conference on Automatic Face and Gesture Recognition*, 1998.

[6] V. Athitsos, J. Alon, and S. Sclaroff, “Efficient nearest neighbor classification using a cascade of approximate similarity measures,” Tech. Rep., Boston University Computer Science, April 2005.

[7] A.K.Jain, J.Mao, and K.Mohiuddin, “Articial neural networks: A tutorial,” *IEEE Computer*, vol. 29, no. 3, pp. 31–44, March 1996.

[8] S.Behnke, M.Pfister, and R.Rojas, “A study on the combination of classiers for handwritten digit recognition,” in *In Proceedings of Neural Networks in Applications, Third International Workshop - Magdeburg, Germany*, 1998.

[9] R.P.W. Duin, P. Juszczak, P. Paclik, E. Pekalska, D. de Ridder, D.M.J. Tax, PRTools4, A Matlab Toolbox for Pattern Recognition*,* Delft University of Technology, 2004.

[10] Keinosuke Fukunaga and Patrenahalli M. Narendra, “A Branch and Bound Algorithm for Computing k-Nearest Neighbours”, *IEEE Transactions on Computers,* July 1995