Digit Recognition Using KNN
CSE 537

Nahid Hossain(109722087) and Ibrahim Hammoud(109953544)

December 6, 2014

Instructor: Professor I.V Ramakrishnan

Abstract

In this report we show our work on the digit detection project. We start by introducing the problem we are trying to solve, then we discuss the PCA technique we used. After that we show our implementation of our solution before we move into giving our simulation results. We conclude on the difference between K-Nearest Neighbor and Neural Networks algorithms and which one would be a prefereble approach.

1 Introduction

Optical character recognition (OCR) refers to the branch of computer science that involves converting images and handwritten text into date that can be used and manipulated by a computer.
One of the active topics in OCR is the specialized field of digit recognition. In digit recognition, we have the same objective as in optical character recognition but with limiting ourselves to the domain of digits in our classification and detection.
Digit recoginition has many important applications like data entry of business documents(checks, postal service...etc), car plate detection, computer pen input and many more. It is clear from the type of applications that the accuracy and performance of such tasks is very crucial and essential. Also it is clear that a slight improvment in accuracy can change the reliability and usability of the applications drastically (like the difference between a 98.7% and 99.7% accuracy) Such important applications and challenging accuracy requirments motivated the further research and development on this topic. One of the major developments that appeared was the feature extraction and classification algorithms. The accuracy of detecting text emerged to have strong ties with the feature extraction and classification method used. Thus the challeging battleground moved into finding good feature extraction and classification algorithms. Various algorithms are currently used in digit recognition such as K-Nearest neighbor, Support Vector Machine, Artificial Neural Network and Convolutional Neural Network.

2 Different types of Classifiers

In this section we discuss about couple of known good classifiers which are used for handwritten digit recognition. We found some of their accuracy on the internet and for some of them we calculated and evaluated ourselves.

2.1 Support Vector Machines

For a binary classifier SVM tries to find the best possible hyperplane to to separate both the classes. For multiple classes it tries to find a non linear hyperplane to separate all the classes. There are different types of kernels for SVM such as: linear, polynomial, radial basis etc. It is seen that using SVM 98

2.2 Convolutional Neural Network

Convolutional neural network (CNN) is a kind of locally connected neural network inspired by biological systems, where a cell is only sensitive to a small subregion of the input space, called a receptive field. At each stage, we have a convolutional layer, local contrast normalization and a pooling layer. Convolutional layer increases the number of feature maps. Pooling layer decreases

the spatial resolution. After multiple stages, the output is connected to a fully connected neural network. It is seen that CNN can achieve 98.85

2.3 K Nearest Neighbors

KNN is a really simple non parametric classifier. It assigns weight to its k neighbors and decide which digit it belongs to. The main advantage of using KNN is that it requires no training time. We'll briefly on KNN later in the documentation.

2.4 Fully Connected Multi-Layer Neural Network

Fully connected multi-layer neural network is a typical neural network model, which consists of an input layer, one or several hidden layers and an output layer. For the handwritten digit recognition problem, we have 10 classes from digit 0 to digit 9. Therefore, we have 10 output units in the output layer. Each output unit represents a class label. The choice of initialization weights, activation function, learning rate and the number of iterations can affect the performance of neural networks. We have done some brief analysis of these parameters on how to they affect the hand written digits.

3 Our Implementaion

We have implemented the KNN method for recognizing hand written digits. We've also used PCA for reducing the size of the features. The model of our implementation is given in Figure.1. An higher level of the interaction between PCA and KNN is given below:

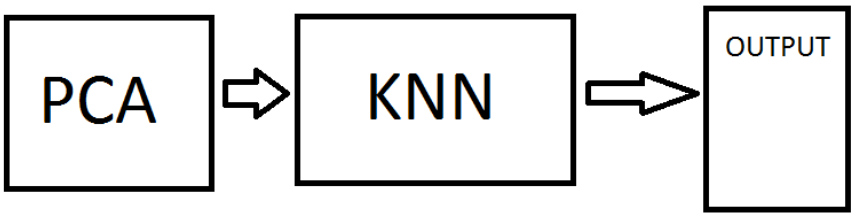

Figure 1: **A flow diagram showing the different steps in our implementation**

3.1 Principal Component Analysis

PCA finds the principal components of data. It is often useful to measure data in terms of its principal components rather than on a normal x-y axis. Principal

components are the underlying structure in the data. They are the directions where there is the most variance, the directions where the data is most spread out. We can find the principal components of a given dataset with eigen-values and eigen-vectors. Eigenvectors and values exist in pairs: every eigenvector has a corresponding eigenvalue. An eigenvector is a direction. An eigenvalue is a number, telling how much variance there is in the data in that direction. The eigenvector with the highest eigenvalue is therefore the principal component. In the given test data the images has dimension of 28x28 so the total number of pixels for each data is 784. So the images has 784 features and as a result it has 784 eigen vectors. PCA can be used to reduce the dimensions of a data set. It reduces the data down into its basic components, stripping away any unnecessary parts. So using PCA we can reduce the number of features. In our implementation we have reduced the dataset features from 784 to 10 features as a result we have reduced the execution time. Also even though we have reduced the number of features it doesn't decrease the performance.
The steps for performing PCA on a 42000 data of 784 features:

- At first we subtract the mean from the data so that every data has a zero mean. The mean removed matrix still has a dimension of 42000x784
- We find the covariance matrix Σ for the data. Which will have a dimension of 784x784.
- We calculate the eigen value and eigen vector of this covariance matrix and sort them in order of their eigen value from high to low. The dimension of the eigen vector matrix will be 784x784. So there are 784 eigen vectors on each colum with 784 features.
- We select the top 10 eigen vectors and this top ten eigen vectors are the principal components of the data. So now the matrix is 784x10
- Now we multiply our mean removed data matrix with the eigen vector which is result gives a matrix of dimension 42000x10. So this is the featured reduced data.

3.2 K-Nearest Neighbor (KNN)

The K-Nearest Neighbhor is one of the famous classification techniques used in digit recoginition. The algorithm is composed of two phases:

1- Training Phase: Training Phase consists of getting the image data and storing them in the appropriate data structure for proper use later.

2- Classification Phase: This phase consists of finding the correct class for the newly introduced image. This is done by finding the K nearest neighbors of the object and then decide using a deciding criteria to choose the correct class for this image.
The classification phase can differ according to the distance function used. Here we display two important distance functions we used in our work:

Euclidean Distance

The Euclidean distance between two vectors X and Y in $\mathbb{R}^n$ is defined to be:

$$\sqrt{\sum_{i=1}^{n}(X[i]-Y[i])^2}$$

Manhattan Distance

The Manhattan distance between two vectors X and Y in $\mathbb{R}^n$ is defined to be:

$$\sum_{i=1}^{n}|X[i]-Y[i]|$$

Also the classification results may vary according to the classification criteria used. Here we display different criteria and discuss the one we chose in the end:

Majority Voting: In this classification critirion, we assign our input image X to the most repeated class in our K nearset neighbors. This method works well most of the times and is good on general datasets.

Average Distance: This method assigns X to the class in the K nearest neighbors that has minimum average distance from X. This method can be useful in some cases but it has some issues when we have small set of noisy values that can have small average distance and thus fool the classification. (for example if K=10, and we got 9 valuess of the correct class and one noisy value of an incorrect class, we might classify our image to belong to the wrong class if this noisy point has distance less than the average of the other 9 points).

Weighted Average Distance: This method is similar to the average distance method, however it emphasis more weight to the closer neighbors. One possible implementation of this method is to assign a weight function $f(d)$, where d is the distance between our image and the training any image inthe training data, that makes the contribution of the nearer neighbors larger than the contribution of the distant neighbors in the mean of the same class, after that we choose the class in our K nearest neighbors set that has the smallest average. This method can be useful in cases where we have the correct class having few images very near the actual distance, but at the same time we have a lot of noisy data. The close neighbors will contribute more and thus we will get the correct value. An example of this is when we

have k=7 and 3 of the nearest neighbors are the correct class while 4 others are the wrong class. Among the 3, two of them are very close and one is relativly far. This resulted in having the average of the 3 neighbors more than the average of the 4 neighbors of the wrong class. Here both of majority voting and average distance will fail, but weighted average distance might have a chance of success with a propper choice of $f(d)$ that can make the contribution of the 2 closer neighbors overshadow the bad contribution of the far neighbor and thus result in a correct classification.

In our work we used the majority voting criteria because it is one of the most standard techniques used while the other techniques are application specific. We tried our data on different methods and we didn't see major improvment over majority voting which means that the training data had some degree of balance that was suitable for majority voting over the other criteria mentioned.

4 Results

4.1 KNN Simulations

In our simulations we used KNN on a sample of 42000 images representing the training data. Then we two implementation of the KNN one using PCA and other without using PCA. We note here that we used only 100 images for the testing. The main reason behind that is that our test data was not labeled and we needed to label them manually.
In our first simulation we compared the KNN and PCA, using the Euclidean distance, over different values of K.
We note here that as shown in figure 2, in both cases our algorithm gave a high acurracy initially (97% with PCA and 98% without PCA) for K=5. And as K increased we can from the graph that on both simulations our accuracy started decreasing until reaching a value of 67 % with PCA and 59% without PCA.
In our second simulation, as in the first simulation we compared the KNN and PCA, over different values of K but this time using Manhattan distance instead of Euclidean distance. The results are shown in the graph below.
We note here that as shown in figure 3, in both cases our algorithm gave a high acurracy initially (97% with PCA and 97% without PCA) for K=5. And as K increased we can from the graph that on both simulations our accuracy started decreasing until reaching a value of 63 % with PCA and 51% without PCA.
According to these simulations we can conclude that the accuracy value is inversly propotional to the value of K. This is expected because as K increases, the ammount of noise will increase and there will be more chance for other incorrect classes to have the majority instead of the correct one. As an example take the case when K is equal to the training data, then all the values are equiprobable and all of the queries will give the same decisions which will result in a very high error rate. Other example is when we have a hard query that has a certain

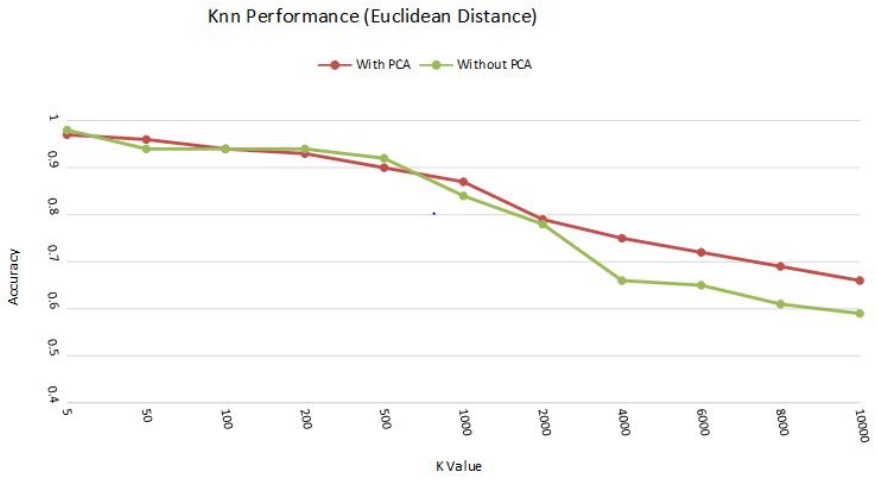

Figure 2: **Accuracy of KNN using Euclidean distance over different values of K**

similarity with an incorrect class, then increasing k will increase the competition of the incorrect class and thus increase it probability of being the majority and thus decreasing the accuracy.
As we saw the accuracy rate of using PCA is as good as (if not better as K increases) without using PCA. But the more important factor for using PCA is that it was able to reduce the time required for digit recognition. In the case of PCA it was **3.4 seconds** per digit while it was **4.25 seconds** per digit without using PCA. This shows that the PCA improved our running time by **20%**.

4.2 Neural Networks Simulations

For the neural network simulation we used the Weka Simulator. As the data size was really huge (42000×784) which gave us out of memory error, so we used PCA to reduce the data features size to 42000×15.
The neural network gives different results for different types of parameters. In our simulations we tried changing the values of hidden layers, number of iterations and learning rate. Over the different values we tried, we obtained the following graph for a learning rate of 0.1.
For a learning rate of 0.1 the accuracy also depends on the number of hidden layers. As we can see from the graph, increasing the number of hidden layer increases the accuracy. Also the number of iteration during training affects the accuracy. We got the lowest accuracy of 78.8% for the 30 layers and 77% for the 15 layers for 1 iteration. As we increased the number of iterations the accuracy increases until it converges on both curves after 40 iterations into 93 % on 30 layers and 87.2% on 15 layers. We have included some of the images of the

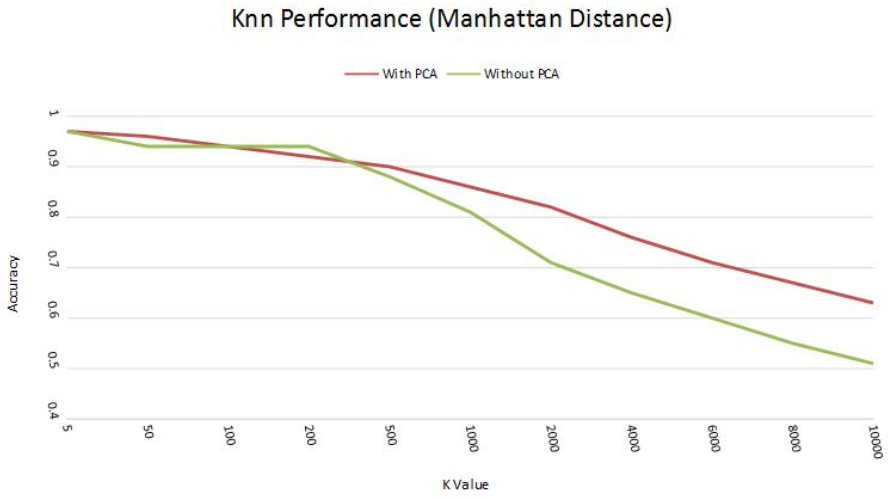

Figure 3: **Accuracy of KNN using Manhattan distance over different values of K**

training in the folder.
Over the different values we tried, we obtained the following graph for a learning rate of 0.01.
As we can see from the graph the accuracy of learning rate of 0.01 gives a very bad accuracy than learning rate 0.1. For a learning rate of 0.01 the accuracy also depends on the number of hidden layers. Increasing the number of hidden layer increases the accuracy. Also the number of iteration during training affects the accuracy. We got the lowest accuracy of 13% for the 30 layers and 12% for the 15 layers for 1 iteration. As we increased the number of iterations the accuracy increases until it converges on both curves after 40 iterations into 90 % on 30 layers and 86% on 15 layers.
It is seen that neural networks gives a 98% accuracy for digit recognition but in our case we only got 92%, which is due to the reason of decreasing the features using PCA. A higher number of features would have given a higher accuracy.

5 Conclusion

As we saw, our simulation results showed a better simulation for KNN over neural networks. However that was because of the reduction of the amount of features on the neural network. According to the literature, the neural networks algorithm should be able to detect the image with accuracy 98% which is very close to our KNN results. So we conclude that the two approaches are comparable in terms of accuracy.

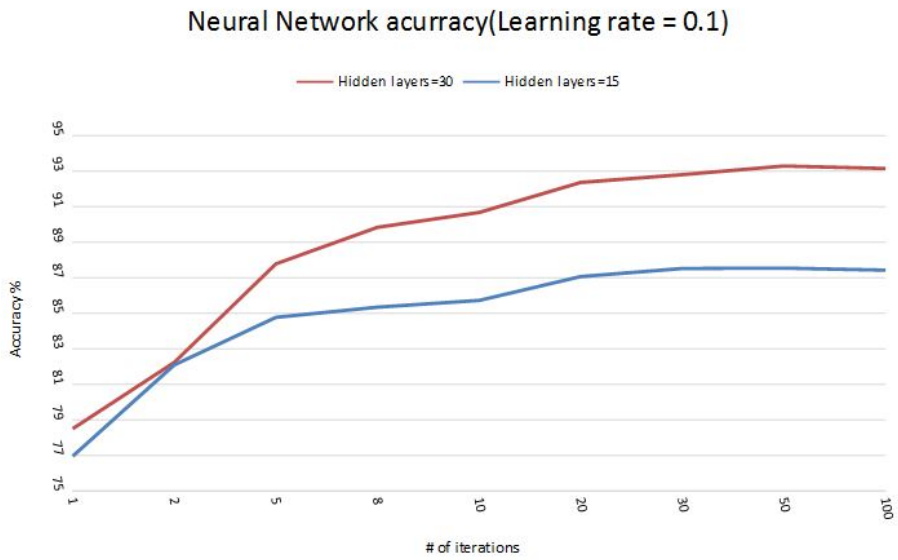

Figure 4: **Neural network simulation with learning rate 0.1**

As in terms of speed and performance, we can see a major difference in the two approaches. The KNN training phase is just storing the data which doesn't require much time, however the neural networks training phase takes a lot of time and memory. On the other hand, the classification of an image on KNN takes some time as it needs to process the whole training set, whereas the classifcation on the neural networks works considerably faster because the training data had already been processed.
So we conclude that both algorithms are comparable with advantages and disadvantages for each. Thus the choice of the better one will depend on the given application, data sizes and computational resources given.

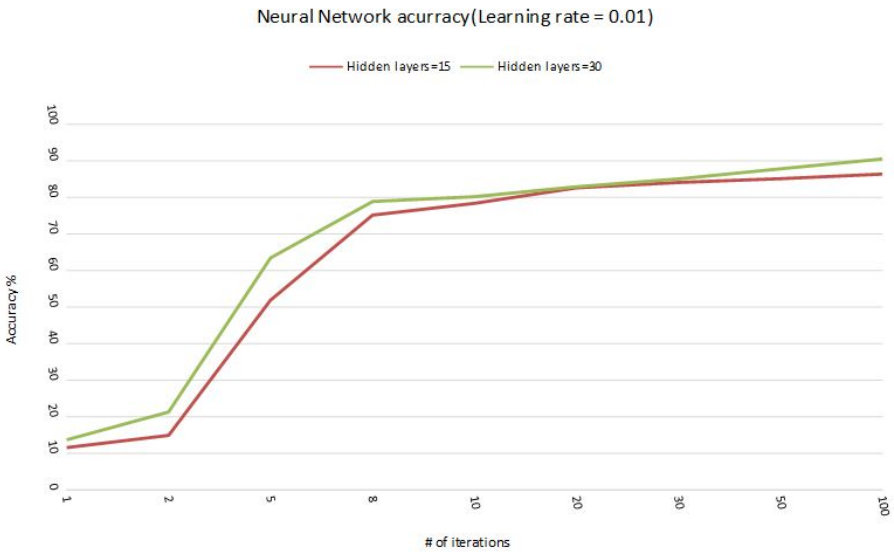

Figure 5: **Neural network simulation with learning rate 0.01**