CS 391L HW1: EigenDigits

Mit Shah (UT EID: mks3226)
University of Texas at Austin
Austin, TX 78712
mkshah@utexas.edu

Abstract

This assignment is aimed at learning Principal Component Analysis (PCA) for the task of classification of digits. It mainly involves calculation of Eigenspace decomposition of covariant matrix of input features. Then on reduced feature space, K Nearest Neighbor (KNN) method is used for classification. The dataset used contains 60K training images and 10K testing images.

1. Introduction

In general, for the task of classification we use some features from input data (Xi's) and their label (Yi) to train a model. Once the training is completed, we can use the model to predict the label of a new instance, given its features. When there are large number of features, it might be difficult to train the model. And even if it is easy, it might not give the best possible results. Not all features will be equally important. So, it is necessary to reduce the number of features by eliminating some of them or projecting them to a lower dimensional space.

Principal Component Analysis is one such method. It transforms the data to a new coordinate system such that first axis has greatest variance, second axis has second greatest variance and so on. We will explore this method here by using it in the task of digit classification.

2. Dataset

The dataset contains images of digits from 0 to 9. All the images are gray-scale and pixels have values between 0 to 255. Their resolution is 28 x 28. So, have 784 pixels, which we will be using as features for the classification task. It is divided into training set and testing set, which contain 60K and 10K images respectively.

3. Method

3.1. Finding Eigenvectors

First we reshape both the training data from 28 x 28 x 60K to 784 x 60K. So, now we have each training instance as 784 x 1 column vector and 60K such vectors. Let's call this 784 x 60K matrix 'A'.

Now, we will construct Eigenspace decomposition from some of the training instances. Let's denote this number of instances by 'k'. Let's crop the matrix A to get the first k columns of it.

B = A(:,1:k);

Now, we first need to subtract mean column vector of B from all the column vectors. That is, we are finding an average image for all the k digits and subtracting them from all the k images.

m = mean(B,2);

B(:,i) = B(:,i) - m; for i=1,..,k

Covariance Matrix of B is calculated with little modification such that it results in k x k system as follows:

C = (1/k)x((transpose(B))x(B));

Now, Eigenvectors of this covariance matrix are calculated, which will be of length k. Premultiplying them with B will result in vectors of length 784, which will be same as the eigenvectors of x x x dimensional covariance matrix of B. Let's say the matrix having these eigenvectors as their columns is V.

3.2. Visualizing Eigenvectors

Let's visualize how these eigenvectors look like if reshaped to original image dimensions. In Figure 1, eigenvectors at intervals of 50 indexes are displayed till 700. 700 training samples were used in order to construct this particular eigenvalue decomposition.

3.3. Projecting Training Instances and Modeling KNN Classifier

Now that we have eigenvector decomposition, we can project training instances on it. To project, premultiply each

Figure 1. Visualizing reshaped first 700 Eigenvectors at intervals of 50

Figure 2. Visualizing Reconstructed Test Instances. First row corresponds to original test cases. Second row corresponds to their reconstruction after projecting them onto 700 Eigenvectors.

instance with V'. Suppose, we project such n training instances on Eigenspace.

From these projected training instances and their true labels we can model a KNN Classifier, for different values of K.

3.4. Predicting Test Instances

To predict, the class of a test digit image, we first reshape it to a 784 x 1 column vector and then project it to Eigenspace. Then we can use the KNN model developed above to predict its class.

3.5. Reconstructing Test Instances from Eigenspace

Now, let's reconstruct the test instances after projecting it onto Eigenspace. To do so, premultiply the projected test instances with V. (Remember that while projecting, we premultiplied test instances with V'. We are essentially reversing that.)

Figure 2 visualizes original test instances (row 1) and their corresponding reconstructions (row 2), after projecting them on to Eigenspace with **700** eigenvectors.

4. Experiments and Results

Different models were constructed by varying mainly three parameters:

a) Number of training samples used to construct Eigenvectors (k)- Possible values: 1 to 784

b) Number of training samples used to project on EigenSpace (n)- Possible values: 1 to 60K

c) Value of K in KNN classification (K)- Tried with values: 1,2,4,8,16

While testing, three subsets of the testing set were used: 1) With easy 5000 images 2) With difficult 5000 images and 3) Complete training set.

4.1. Initial Coarse Grid Search

Initially to find the optimal configuration a grid search was performed over the following values:

k=100,250,400,550,700;

n=20k,30k,40k,50k,60k;

K=1,2,4,8,16.

As the training samples to be projected were in large amount, this grid search took approximately 14 hours on an Intel i5 - 3rd Gen processor.

Results found are shown in Figure 3. There are 5 surface plots, each corresponding to a different K value in KNN. No. of Eigenvectors are on X axis and no. of training samples used to project on Eigenspace are on Y-axis. Z-axis represents the accuracy on full test set.

Exact values for plot 3 (as K=4 achieves best accuracy) in Figure 3 are given in Table 1. We can see that with k=700 and n=60k, it achieves the highest accuracy of 94.55%.

4.1.1 Variation in accuracy with #Eigenvectors and #training samples projected

From all the 5 plots, it can be concluded that, both increase in no.of Eigenvectors (k) and training samples used for projection (n), improves accuracy substantially in long run. Only increasing one of them won't improve accuracy that much.

4.1.2 Variation in accuracy with different K values for KNN

Also, it can be seen that K=4 for KNN (3rd plot) performs better than other K values. To compare the behavior of accuracy with change in K value, refer to Figure 4. From both the plots, we can see that accuracy was highest for K=4, except for the one instance in 2nd plot (lowest one), where it was highest for K=8. So, unlike other two parameters, accuracy is not always increasing with increasing K values.

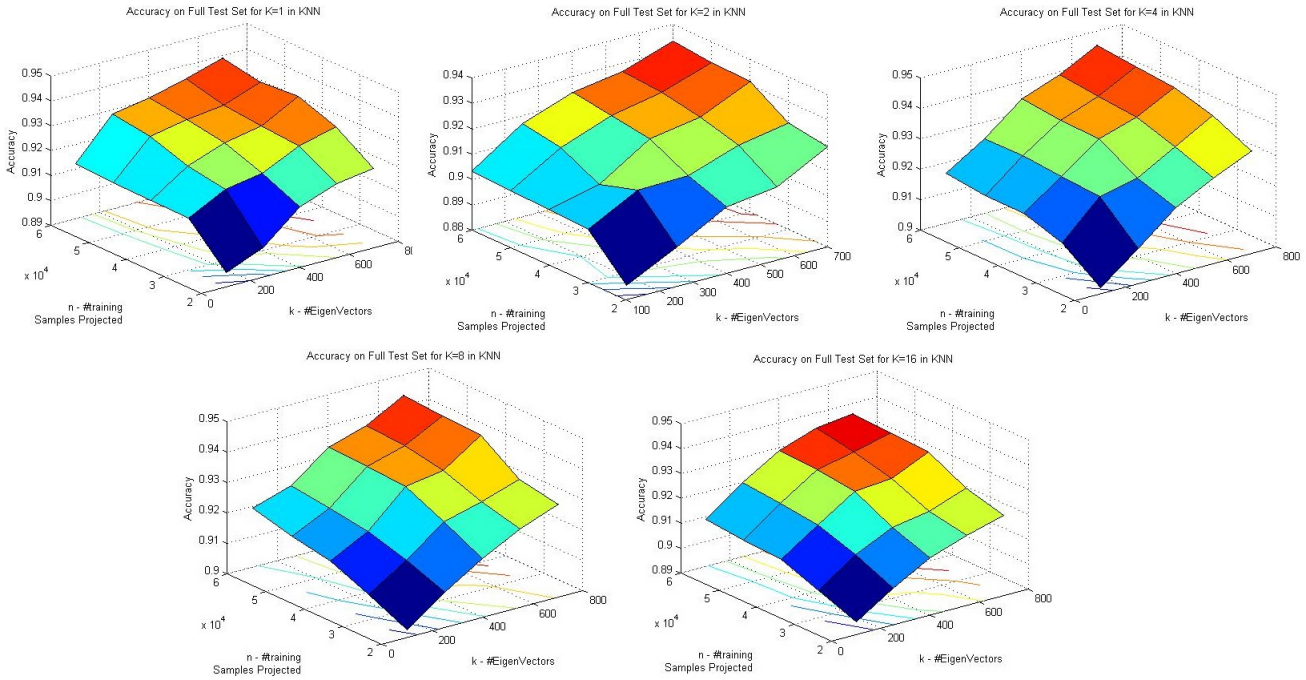

Figure 3. Surface plots for Accuracy with #EigenVectors as X-axis and #training Samples Projected as Y-axis. 5 plots belongs to different K-values: 1,2,4,8,16.

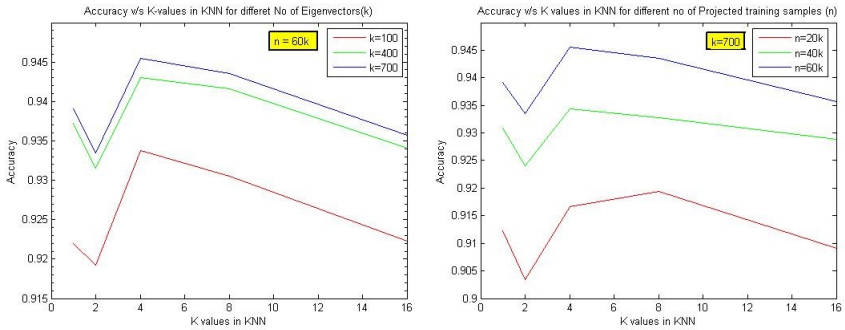

Figure 4. Accuracy for different K values in KNN. (a) First plot contains three lines corresponding to different #Eigenvector(k), while keeping #training Samples Projected(n) constant to 60k. (b) Second plot contains three lines corresponding to different #training Samples Projected(n), while keeping #Eigenvector(k) constant to 700.

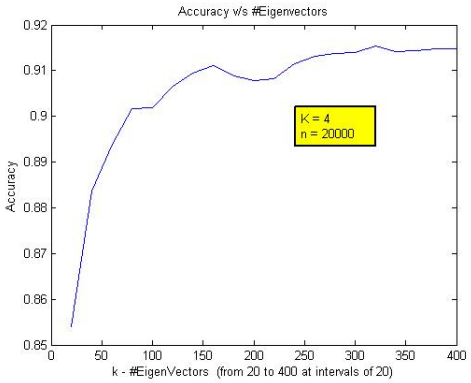

Figure 5. Accuracy v/s #Eigenvectors, with K=4, n=20k

4.2. Fine search on a smaller region

Till now, we have seen the results of coarse Grid search. Now that we know K=4 is best from our choices, we can use this to measure results on a more finer region. Refer to Figure 5, which measures accuracy with respect to no. of Eigenvectors (k) ranging from 20 to 400 at intervals of 20. K value is kept at 4 and due to computational constraints, n value is kept at 20k (though 60k would give much better results). We can see that almost all the times, accuracy is smoothly increasing with increasing k. Initially the increase is steep, but after 300, it is increasing very slowly.

Table 1. Accuracies for Different k and n values, with K=4.

k / n	**20k**	**30k**	**40k**	**50k**	**60k**
100	0.9019	0.9115	0.9216	0.9281	0.9338
250	0.9110	0.9228	0.9256	0.9325	0.9395
400	0.9148	0.9238	0.9328	0.9365	0.9430
550	0.9149	0.9249	0.9330	0.9378	0.9440
700	0.9166	0.9246	0.9344	0.9391	**0.9455**

4.3. Accuracies on subsets of Test Set

For two subsets of the test set, the one with easy images had on an average 1.5% more accuracy, while the one with difficult images had on an average 1% less accuracy then total set. For the best configuration - k=700, n=60k, K=4 - they achieved accuracies of **96.87%** and **93.9%** respectively.

5. Conclusion

From the experiments conducted, it was found that increasing the number of Eigenvectors helped improving accuracy, as it covered more and more data from the inputs. Also, increasing number of training samples used for projecting improved it. While increasing K value in KNN beyond certain limit did not improve the accuracy.

Finally, we can conclude that PCA really achieves a good accuracy in classifying digits and so Eigenvectors constructed from covariance matrix are really representative of input features and thus helps in feature reduction.