Project 2 – USPS Handwritten Digits Dataset Analysis

I. Principal Component Analysis

Dataset used throughout this project is usps_all.mat, which is easily available online. The dataset once loaded, gives us a matrix, 'data' of size of 256x1100x10. In order for us to perform PCA on the whole dataset, we must reshape our dataset in such a way that each row of the dataset corresponds to an observations/examples, and each column corresponds to variables.

Hence, I transposed my 'data' matrix to create 'digits' matrix of size 1100x256x10, where each row in digits is one of the 1100 examples and each column is a variable/feature or in our case, a pixel value. PCA was performed on this dataset using the inbuilt PCA function in MATLAB. By default, PCA centers the data and uses SVD algorithm.

For each digit (0..9) in the 'digits' matrix, we have a matrix of the form 1100x256. I used for-loop to iterate through each digit and pass their corresponding 1100x256 matrix into the PCA function. The function returned a set of four matrices for each digit (0..9): *ceoff*, *score*, *latent*, and *mu*. The *coeff* matrix is of the size 256x256 where each column represents the coefficients for a principal component, and the columns are then sorted in a descending order of component variance. The *score* (representation of our dataset in the principal component space) is of the size 1100x256, *latent* (principal component variances) is of the size 256x1, and *mu* (estimated mean of each variable) of size 1x256.

Below, in figure-1, are the plots of the first and the last principal components as functions on [1,256], for each digit.

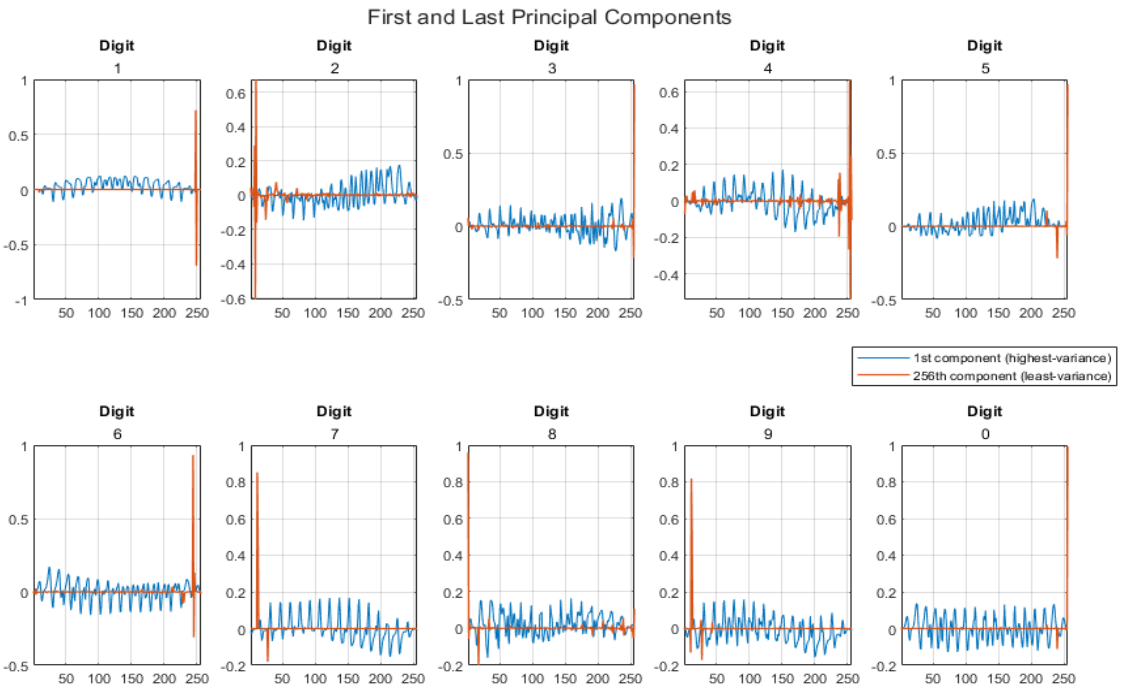

fig-1

II. Data Representation [Post-PCA]

In the previous section, using PCA, we found the coeff and score matrices of the 'digits' dataset (1100x256x10). While coeff matrix (256x256x10) contains just the coefficients of the principal components, the score matrix is basically the representation of X in the principal component space. Rows of score (1100x256x10) correspond to observations, and each column correspond to components.

Post-PCA, to represent the dataset, I found the product of the original dataset 'digits' (1100x256x10) and the coefficient matrix (256x256x10) and used the resulting matrix, Q, of size 1100x256x10 for kNN classification. Below, figure-2 is a visual representation of the data in the new basis of principal components. Each picture from left to right represents the first example/instance of each digit post-PCA, with the leftmost in the first row being digit 1, and the right most in the second row being digit 0.

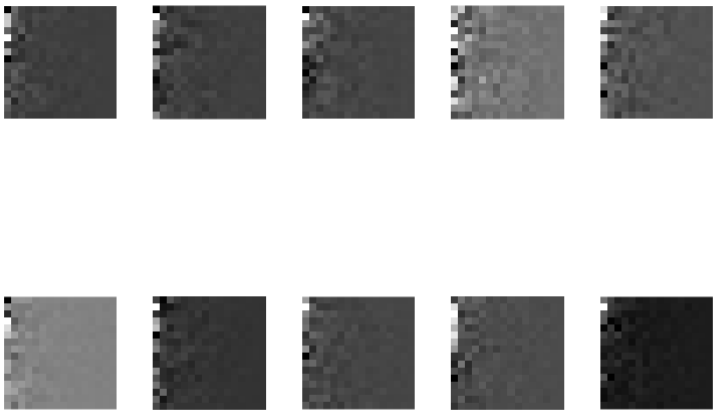

fig-2 - Matrix Q

Further tests were performed to confirm if the PCA did indeed process the data correctly. I multiplied the score matrix with the transpose of coeff matrix. In the form of an equation, it looks like this: {X = score * coeff ’}. Here, matrix X is a centered version of our original dataset. I went ahead displayed the first examples of each digit of the resulting matrix X. As expected, following was the output:

fig-3

Hence, the coeff and the score matrices were computed correctly, and our score matrix was the right representation of our dataset in the new basis of principal components.

III. kNN Classification

Using this PCA represented dataset, Q, we obtained in the previous section, as the new dataset, kNN classification was performed using the optimal parameters obtained from the previous project. The optimal train/test size I obtained in the previous project was 1050/50, where the first 1050 instances of each digit are put into the training set, and the rest 50 are tossed into the testing set. Optimal metric used was Euclidean. The test and the training set of size 256x1050x10 and 256x50x10, respectively, were reshaped into 10500x256 and 500x256, respectively. They were reshaped in such a way that each row in the training set is an observation/example and each row in the testing set is a query point that needs to be classified.

1. Validation

Before classifying, I performed validation on my data to find an optimal 'k' for the dataset. For validation, my training set consisted of first 900 examples of each digit while the validation set consisted of the next 150 examples of each digit, and the test set was left untouched (last 50 examples of each digit). Following figure-4 represents the findings.

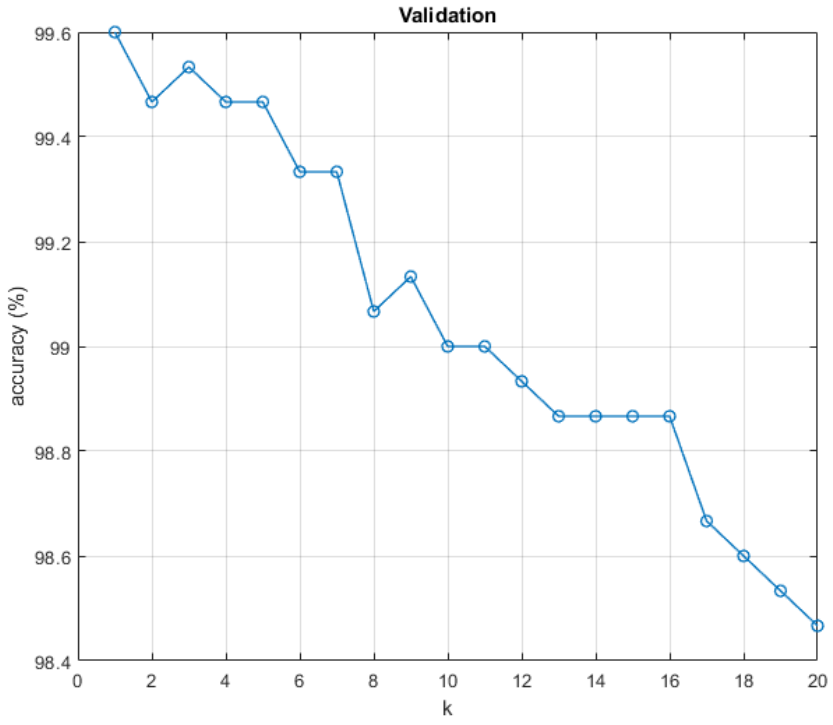

fig-4

I then computed the mean of accuracies for each ‘k’ from 1 to 20 and found the optimal ‘k’ with the accuracy closes to the mean of accuracies using the formula:

```
[val,pos] = min(abs(accuracy-mean(accuracy)))
```

From validation I obtained k=8 as the optimal value.

2. Testing

Using the optimal value of k=8, the optimal train/test:1050/50 split, and using Euclidean metric, the test dataset was classified. Individual success rates were obtained for each digit.

a. *Test Case 1*: k=8 (optimal)

At k=8, the global accuracy reached **99%**, and following bar-chart in figure-5 represents the individual success rates for each digit.

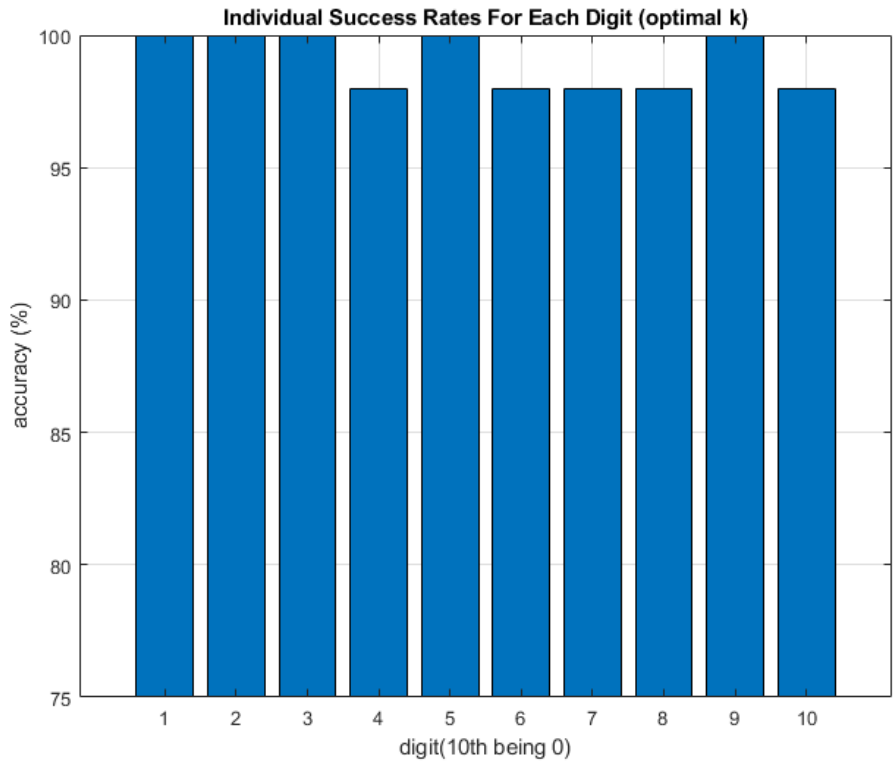

fig-5

b. *Test Case 2*: k=20

At k=20, the global accuracy score reached **98.6%**, and following bar-chart in figure-6 represents the individual success rates for each digit.

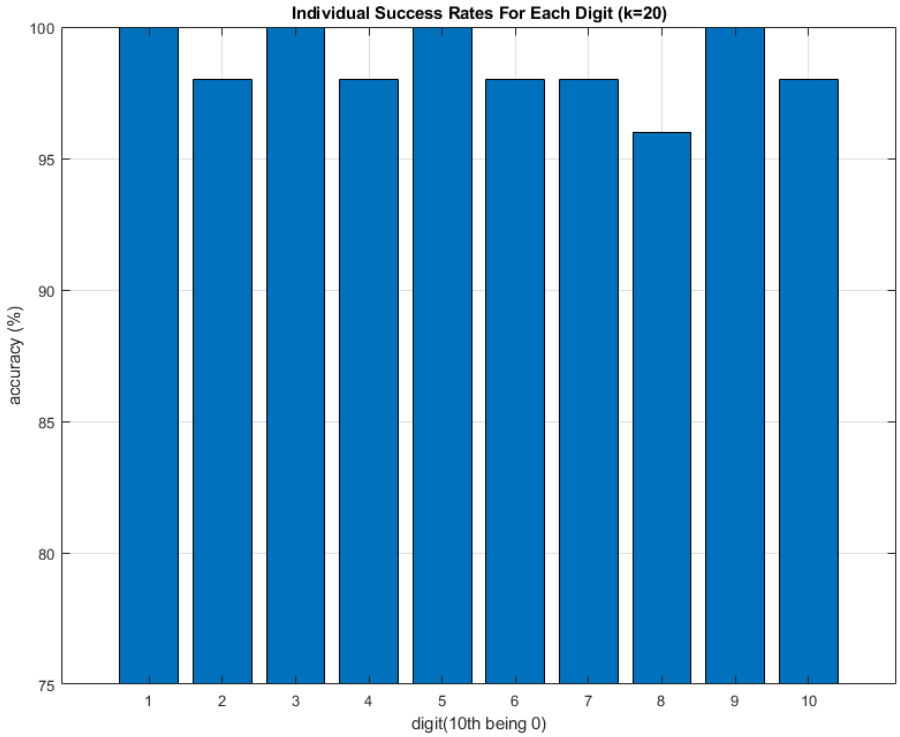

fig-6

c. *Test Case 3*: k=1:20

The model was finally ran for all 'k' from 1 to 20 to check how well the validation set represented the test set. Following figure-7 plots the accuracy scores from each k. At k=4, the accuracy was the highest at 99.4%.

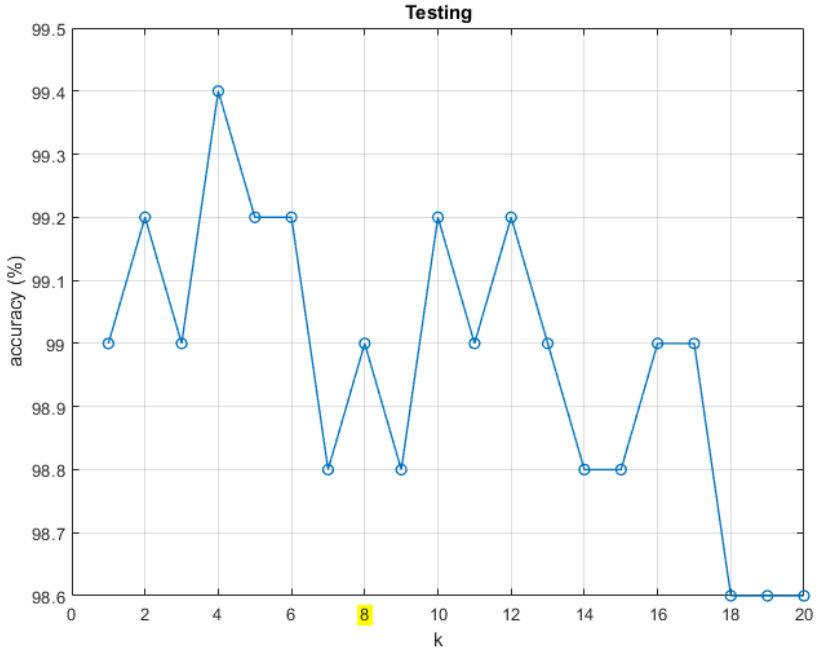

fig-7

IV. Takeaways

In problem 2 of project 1, the model without PCA got a global accuracy score of **90.8%**, and figure-8 has the bar-chart of the individual success rates for each digit where the chosen k was 20. Comparing these numbers to what we have obtained so far, it is safe to say that PCA has improved the classification by a great deal.

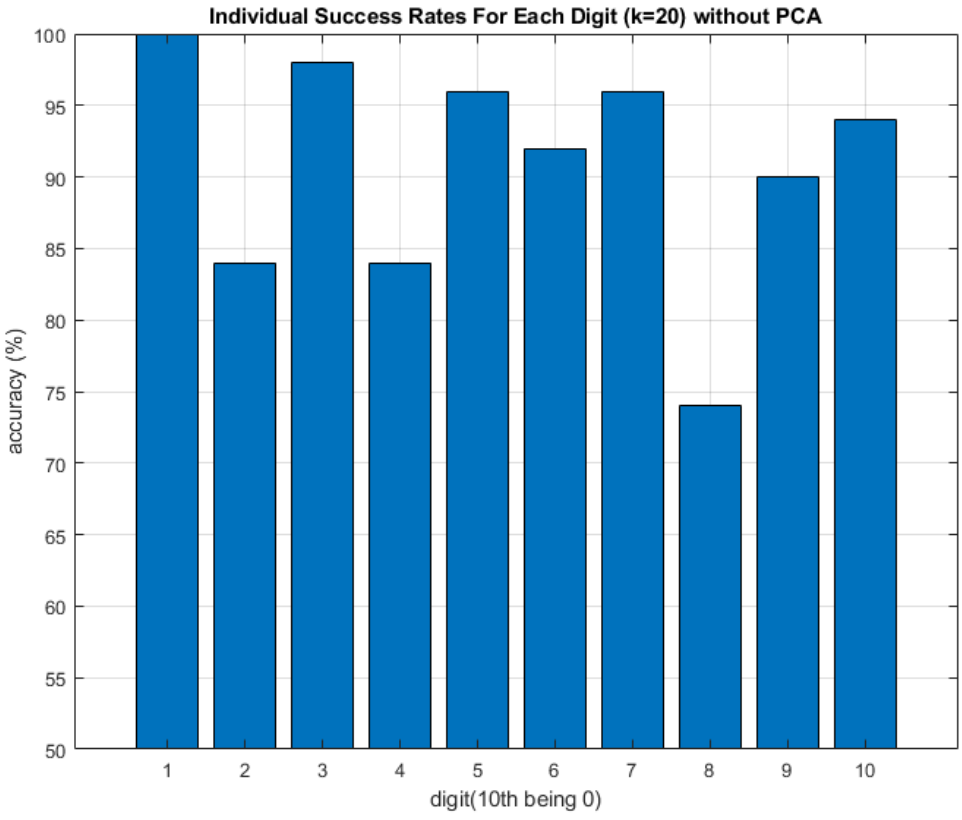

fig-8

Comparing the individual scores in figure-8 with the scores we found in test case 2 of last section (figure-6), we notice that the accuracy score for each digit have risen by at least 10%, while the digits, 3,5 and 9 reached 100% with PCA. Although, digit-1 and digit-8 accuracies did not change by much when compared to other digits, where digit-1 stayed at 100%, making it the most uniformly written digit, and the digit-8's accuracy increased by 15% but still the least accurately classified digit among the set of all digits.