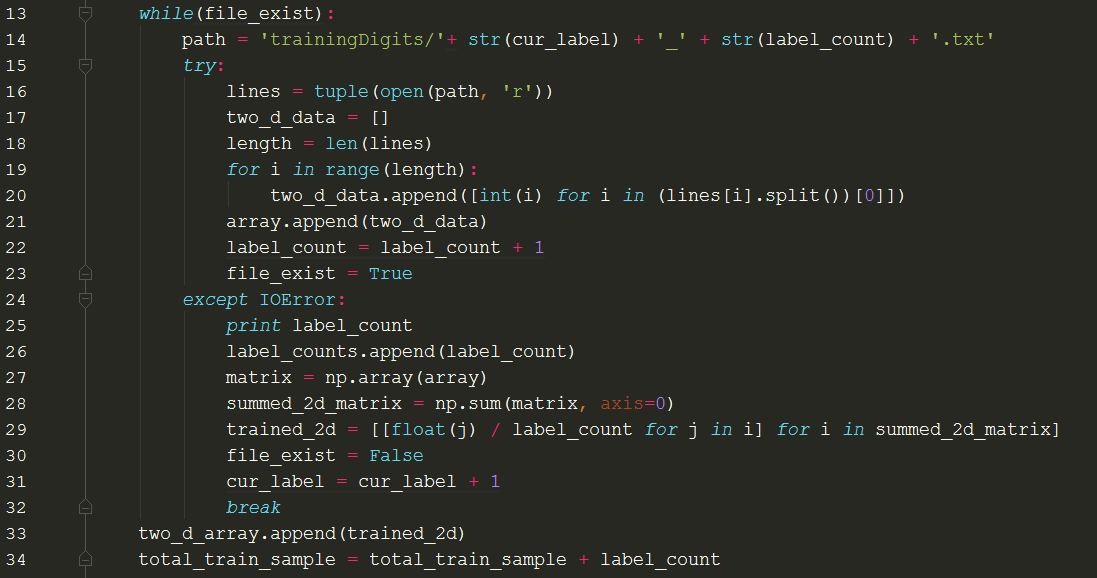
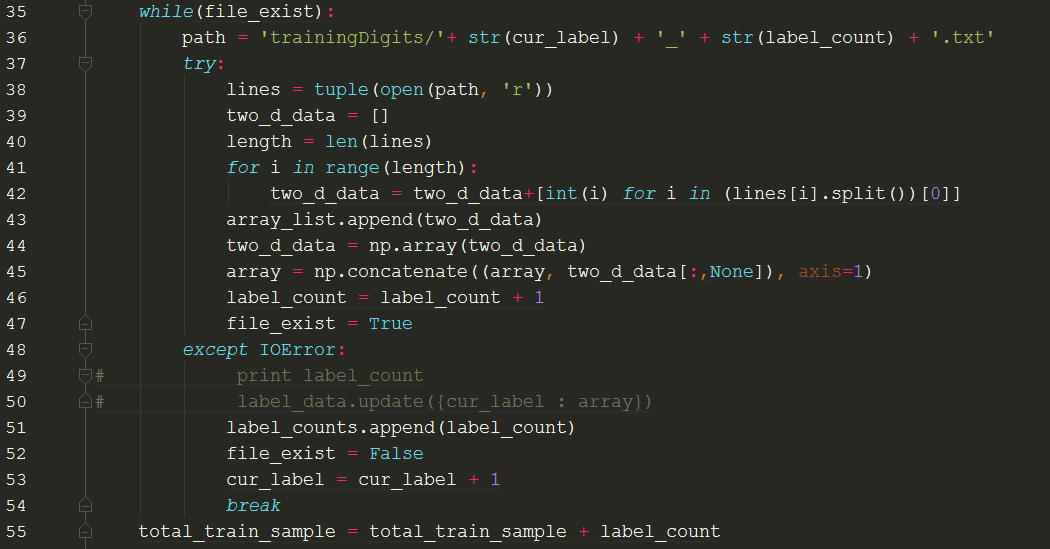
Programming Assignment 1

This assignment aims to classify digits, I implemented classifiers using Python without using build-in packets except ‘numpy’ computation module

1. **Read the digits data and convert them into vectors**

In this assignment, I implemented this part in two ways. For Naïve Bayes Classifier and KNN without PCA, I read digits into two-dimensional matrix; for classifiers with PCA, I read digits into vectors. The implement snapshot is shown below, respectively. They are extracted from the file ‘nbc.py’ and ‘pca\_nbc.py’, respectively.

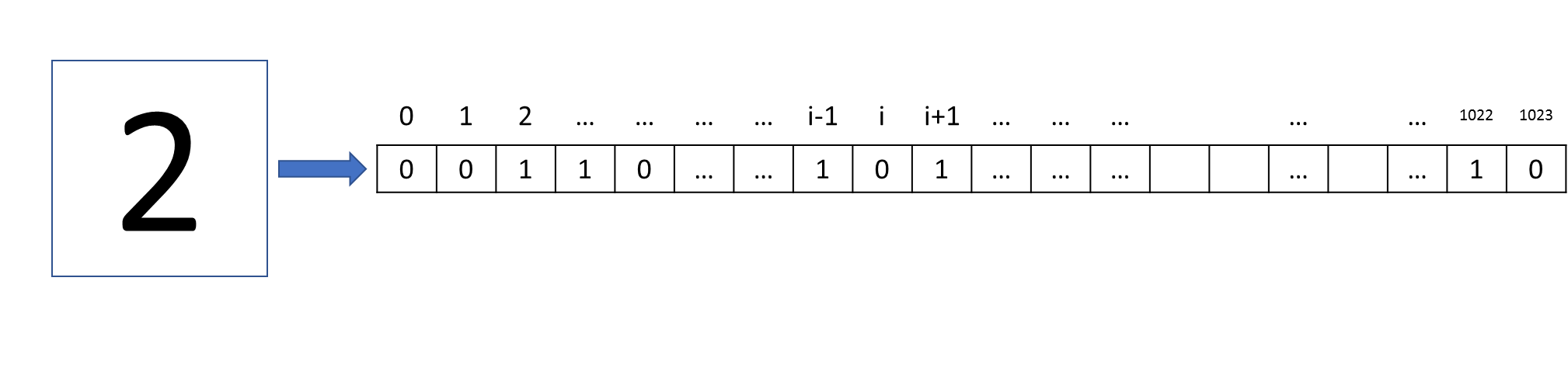




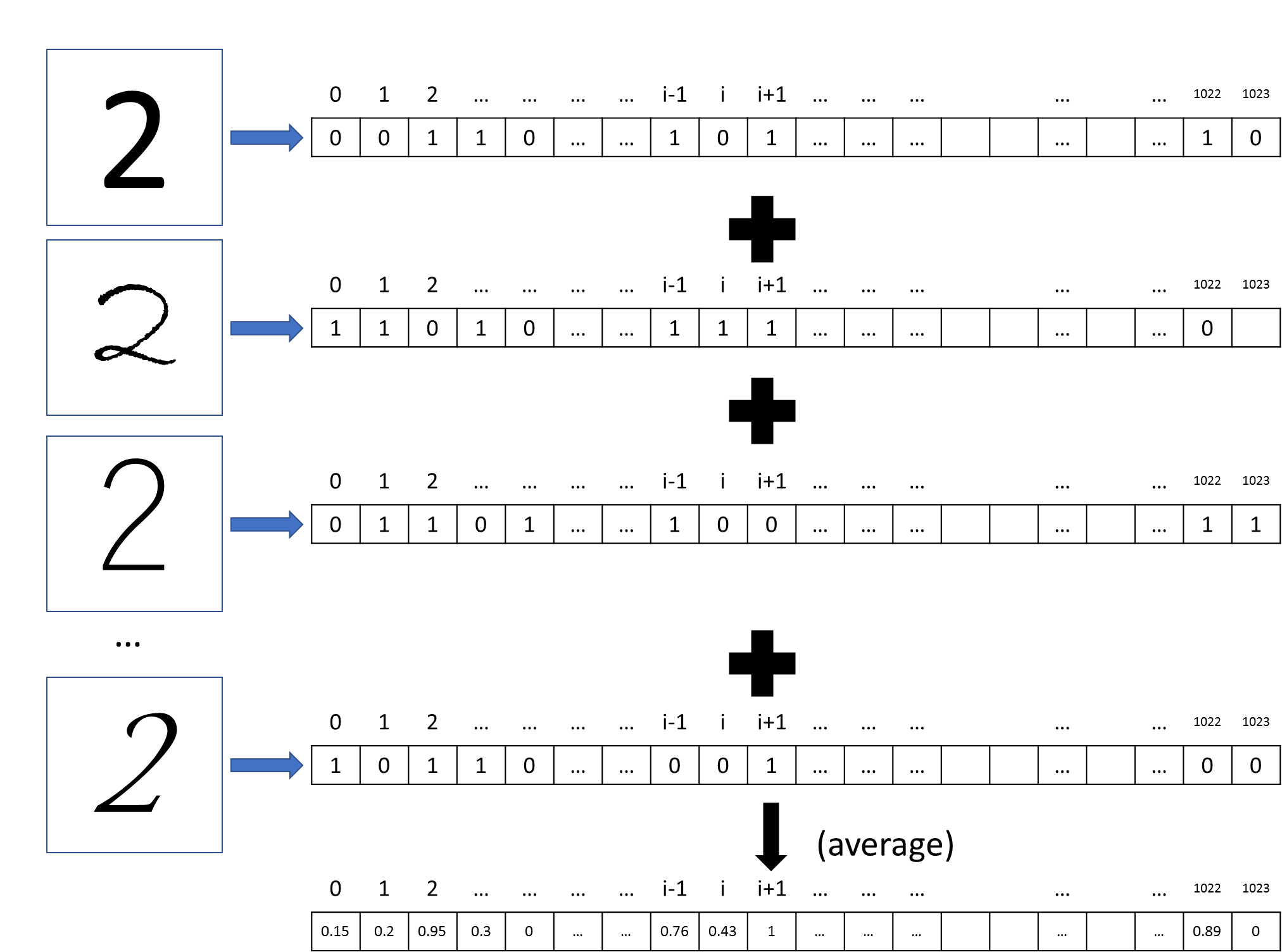
This implementation is because I was seeking efficiency to compute for specific cases. For cases with PCA de-dimensionalization, we need vectors to get the less dimensions. While in cases without PCA de-dimensionalization, we don’t need to do that. I can just read the whole matrix and compute based on the raw arrangement of the 0/1 values in the 32\*32 matrices.

1. **Implement naïve Bayes classifier, compute both training error and testing error rates**

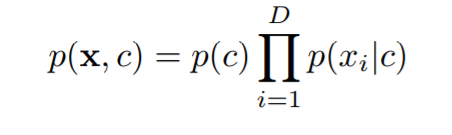
The main implementation flow is as follows: First, we convert each digit sample in the training set into vectors/matrix (refer to the 1st part) containing 0/1 values. For example, when we reach the place reading a training sample labelled as ‘2’.



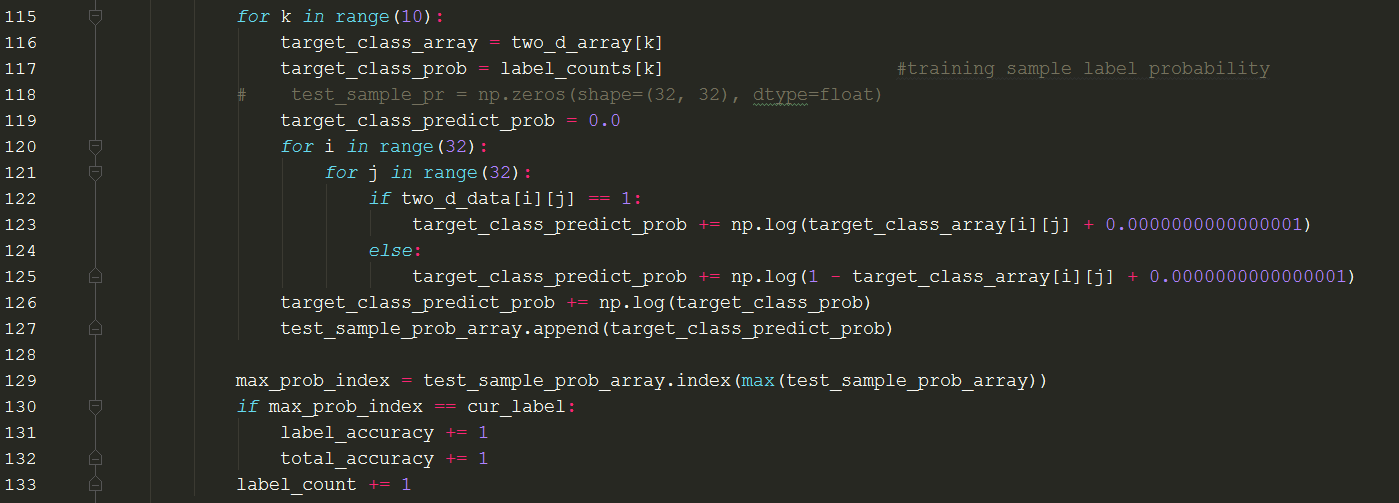
Second, when we have read all the labelled sample in the training set, I computed the average of all the train samples with same label. For example, after I have got all the 2s in the training data, I just compute the average of each pixel.



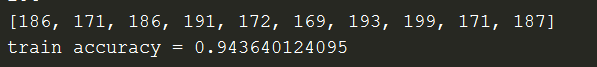
Third, I’ll compute the training error. The process is just following the Naïve Bayes computation rule. For each picked training sample, I check each pixel. If the element in the pixel is 1, I’ll see the likelihood probability in this pixel as the computed average number shown above; if the element is 0, I’ll see the likelihood as (1-average). Here, I’d rather say that I was doing frequency learning here. The theoretical equation is ()



Where p(c) is represented as the frequency of labels. The code implementation is shown below (extracted from ‘nbc.py’ file). Here I used the sum of log in place of the multiplication for computation and memory efficiency.

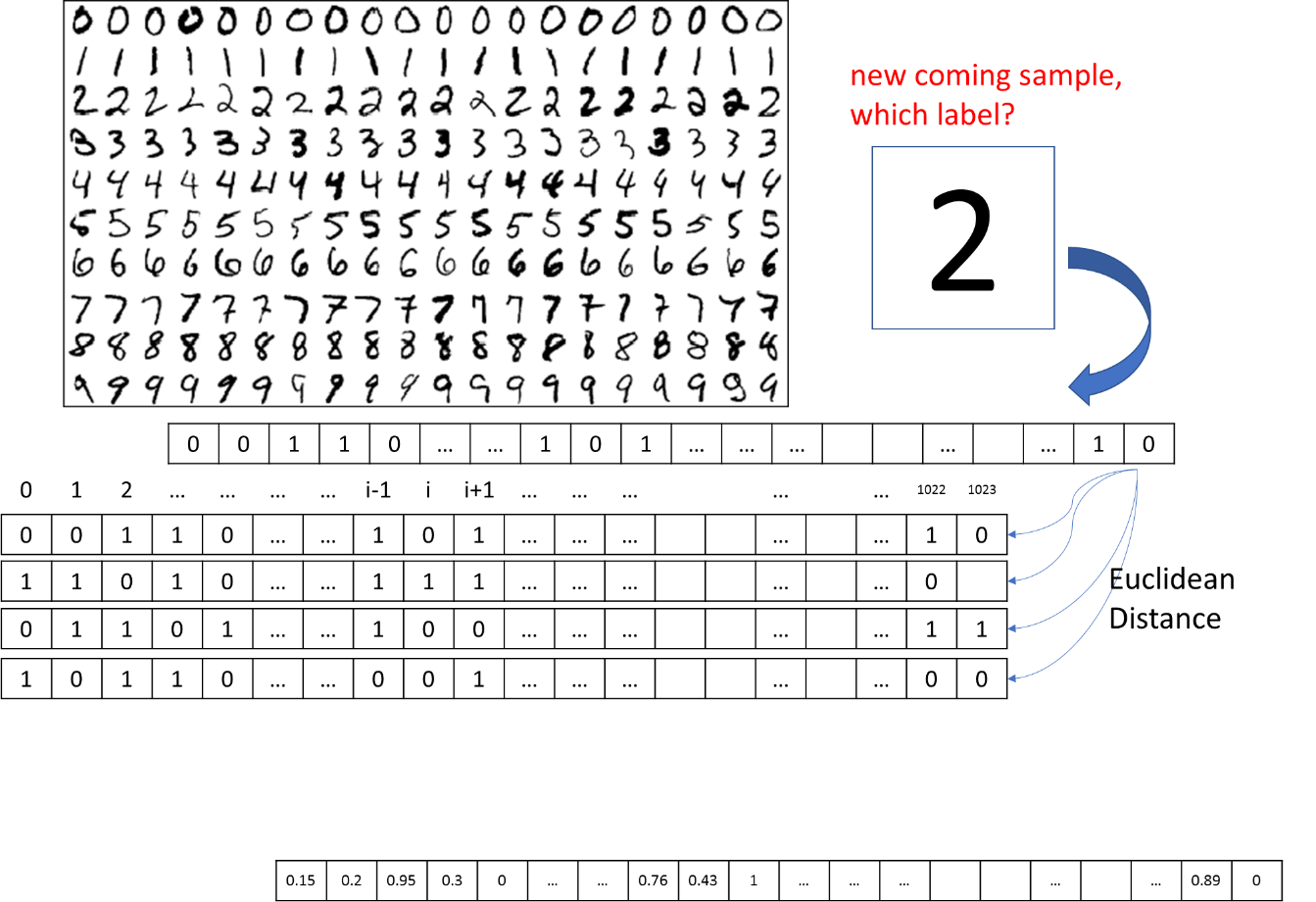


Similarly, we compute the accuracy for test samples using the same logic. The accuracy for both training samples and testing samples are shown below. So the error rates for training set is 1-0.9436 = 0.0564, error rates for testing set is 1-0.92494 = 0.07506.

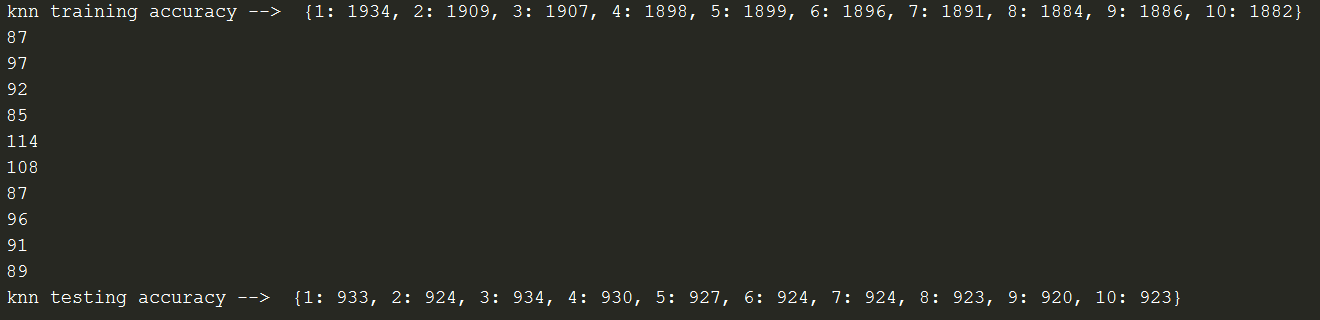


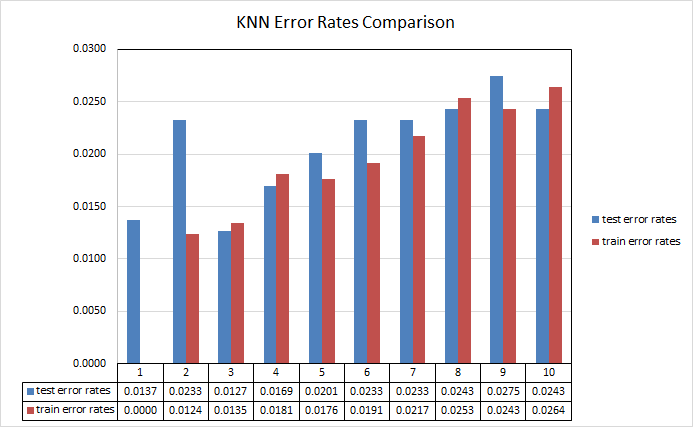
1. **Implement the KNN classifier**

(1) my KNN implementation is based on the computation of the Euclidean Distance between two arrays. Suppose we have already read the training samples into vectors with 0/1 values, for each coming new sample, I compute the Euclidean distance of this sample with all the samples we have got. In this way, we can pick the top k smallest distances with their respective labels. The more frequent label in the k labels would be our final label choice.



(2) Evaluate testing/training error rates, and plot with respect to k: I plotted the KNN error rates below. The k = 3 is the best choice for the test case, while k = 1 is the best for training set. It can be easily seen that the error rates increase as the size of k increases. Also, I give the accuracy printed on the terminal here for references.





(3) Model Averaging: As shown in the figure above, we can simply evaluate the model averaging as selecting the most probable label given a coming new sample. In terms of the correct prediction given a sample point, the least error case is corresponding largest Bayes’ factor, i.e., for training set, we select the KNN with k = 1 as our model since it has least error; for the test set, we select KNN with k = 3 as our model since it has least error for the test samples.

(4) Compare with NBC: The most error for KNN is 0.0275, which much less than NBC errors, either in terms of test error rates and train error rates.

Pros for NBC: for a given novel sample, we directly apply Bayes Rule to it. The likelihood and prior should be given, it is simply to understand and easy to do the computations

Cons for NBC: the result and accuracy is subject to the variations of prior distribution. In this assignment, the prior is taken as the frequency. The underlying assumption is that the data needs to be i.i.d., which cannot hold true in reality.

Pros for KNN: compared to NBC, KNN is a non-parametric learning method, it is simple to implement, also it is more accurate.

Cons for KNN: As shown in this assignment, we rely on the selection of k, to get the optimal k value, we need to test multiple k values, which is very computation expensive

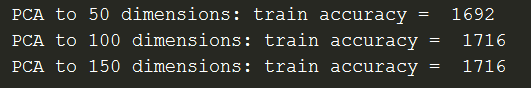
Possible ways to improve NBC performance: In terms of feature selection, we can ignore some part of data, and focus only on those representing best the selected samples. Also, I think we can keep the same size of a particular label, in another, we make our class uniform distributed.

1. **Apply PCA to both training and testing data**
2. PCA application:

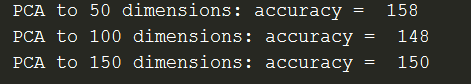
In my application, I did not take the built-in PCA functions or packages. I was following the tutorial which I found very useful (<http://sebastianraschka.com/Articles/2014_pca_step_by_step.html#sc_matrix>). I listed the error rates of both NBC (‘pca\_nbc.py’ file) and KNN (‘pca\_knn.py’ file) after the application of PCA. Note that I was trying to reduce the feature dimension of samples into 50, 100, 150, respectively.

PCA\_NBC:

The accuracy for train samples:

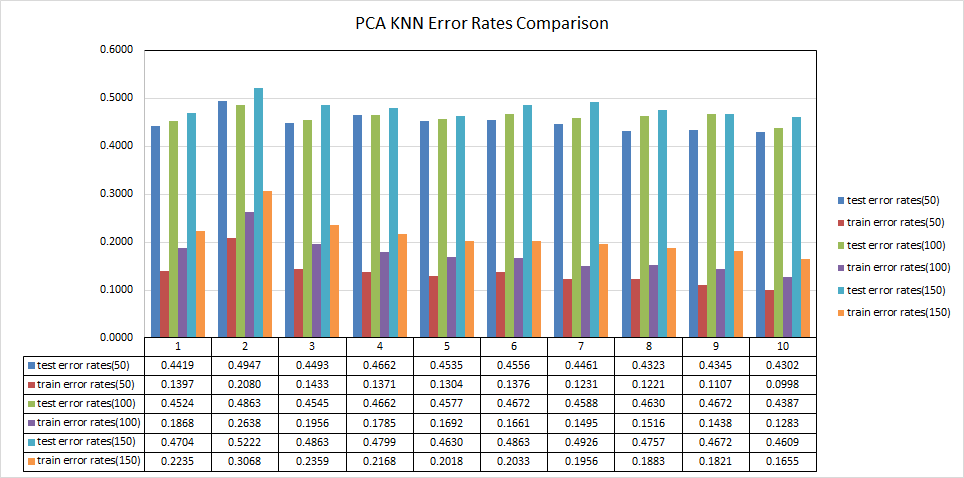


The accuracy for testing samples:

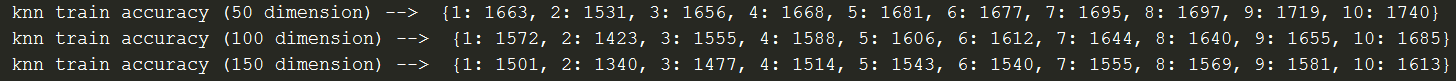


We can see that the accuracy for the test data is very low. I think this might be due to the problem that PCA application eliminate some important features that can represent original data very well, such that a novel data cannot be classified. While in the train set, the underlying model pattern was preserved well after the dimension reduction through PCA, this was reflected by higher accuracies in the training sets.

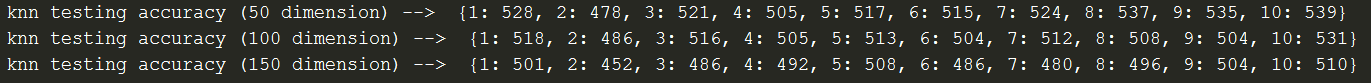
PCA\_KNN: It can be easily seen that for the training set, whether dimension is reduced into 50, 100 or 150 does not make much differences. However, as for the test set, 50 dimensions choice is seemingly more accurate. It can also be drawn from the graph below that information of underlying pattern is lost due to the reduction of dimensions, which makes the prediction model give appreciable higher error rates in terms of novel test sample.



The accuracy for train samples:



The accuracy for test samples:



1. Pros: PCA saves much time in terms of computation, it removes some redundant features of samples. More importantly, when we have a novel input x\* that is very far from all input samples, then it is not wise to purely classify the new input into any existing classes according to the nearest neighbor selection criterion. In this case, PCA solves this problem by convert the data into low dimensions with each dimension containing more information. In this way, it is more convincible to compare distances based on the lower dimension space.

Cons: Ultimately, some information in the original data was lost during the construction of dimension reduction process. This is indicated in the increase of error rates after PCA applications. In addition, although PCA method reduces the dimensions, it still requires us to compute the eigenvalues and eigenvectors, as well as correlation matrix, etc., which are still computation expensive.