ENGR421 – HW2 report

First I used *pd.read\_csv* to read the datasets provided to us. Right after this I sued *np.squeeze* function on labels array because each individual label integer was represented in a single array, giving the whole labels array a dimension of size 1. This function eliminates these 1-sized dimensions from arrays. Then, for easy-use I stored the number of classes by using *np.max* on the labels array (giving the max value 5), and the data size which is number of pizels in a single image (784) by extracting it from the image set’s shape.

Then, I split the train and test sets for images and their corresponding labels simply by specifying array indices. Here, thus the train and test sets aren’t quite assigned randomly but by a deterministic manner depending on their sequence in the given array. I also created arrays for storing the class sizes in both training and test set, indicating the number of entries in each class using the *np.bincount* function.

In order to estimate the means I wrote a *calculate\_means* it sums all the X (image values) and divides them with the classes size. The resulting values are appended to a list which I then again used the *np.squeeze* function to reduce the 1-sized dimension. I did this for both train and test set since both will be used for generating predictions.

Then, to estimate the deviations I used the sqrt((sum of (x-mu)^2) / class size) formula then I appended the results to a list which I then turned into an appropriate size np.array. The deviations are of dimensions (class no, data size). There was no zero-valued entry in the train set’s deviations but there were some in the test set’s. Since in the bayes calculation we divide the (x-mu)^2 by deviation I turned them into a very small incremental value.

After this, I calculated the prior probabilities for test and train sets by simply dividing the number of data points in a particular class by the total number of data points in the dataset. I repeated this calculation for each class.

Then to define a score for each image by each class, I used the Bayes theorem. For calculation I used the maximum log likelihood we derived in class (where we omit P(data) since all are identical for each class) which gives out a value proportional to Bayes’ theorem that gives the posterior probability of a data belonging to a class given the data. Since the deviations and means were calculated by pixel I also then summed these values for them to cumulatively represent the likelihood of the image itself. These score values wer stored in an matrix of dimensions (number of classes, number of images) since I did the score calculations class after class and appended them. But I found accessing the images themselves first more convenient for this I transposed these score matrices.

I wrote other two methods called *get\_max\_class* which returns the class of the maximum score value of an image. The other method called *generate\_predictions* calls ths method for each data and appends the class result from *get\_max\_class*  in a list which it returns after turning it to a *np.array*. After having the estimations using the *pd.crosstab* function I generated the confusion matrices for the predictions on both the training and test datasets.