ECE5470 Computer Vision

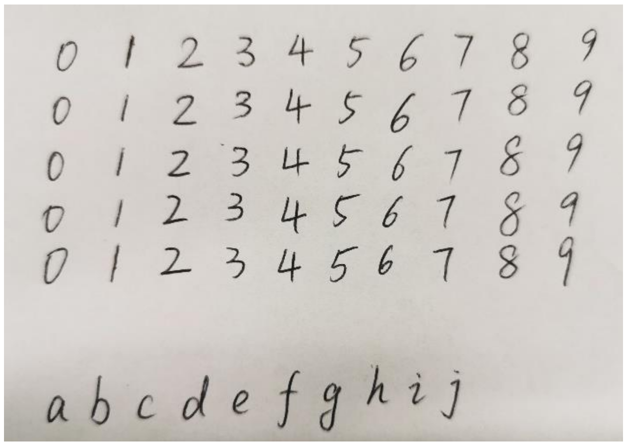
Lab 7: Machine Learning

Jingwen Ye (jy879)

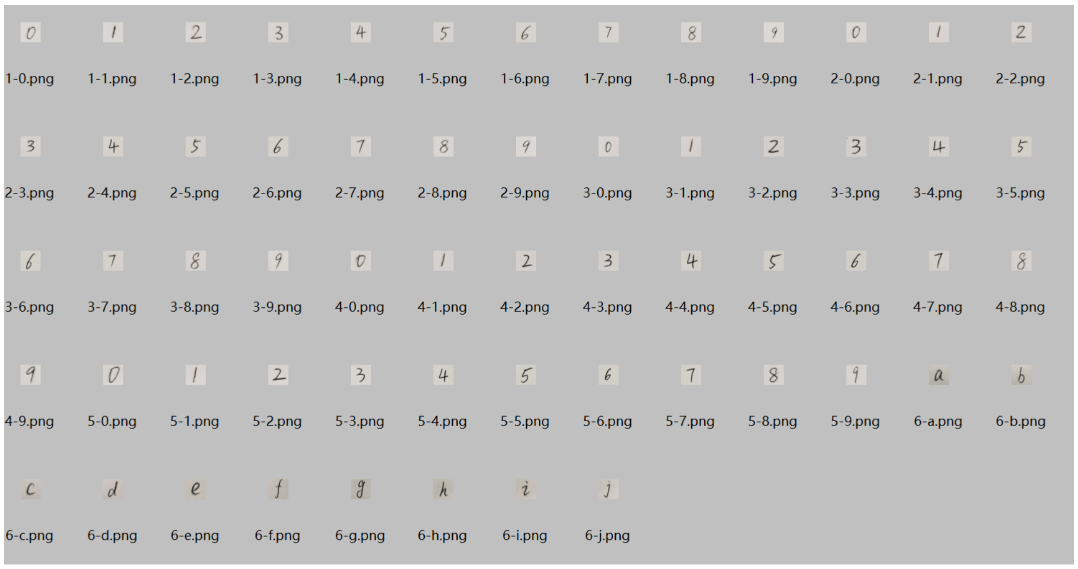
Part A

1. Data set creation.

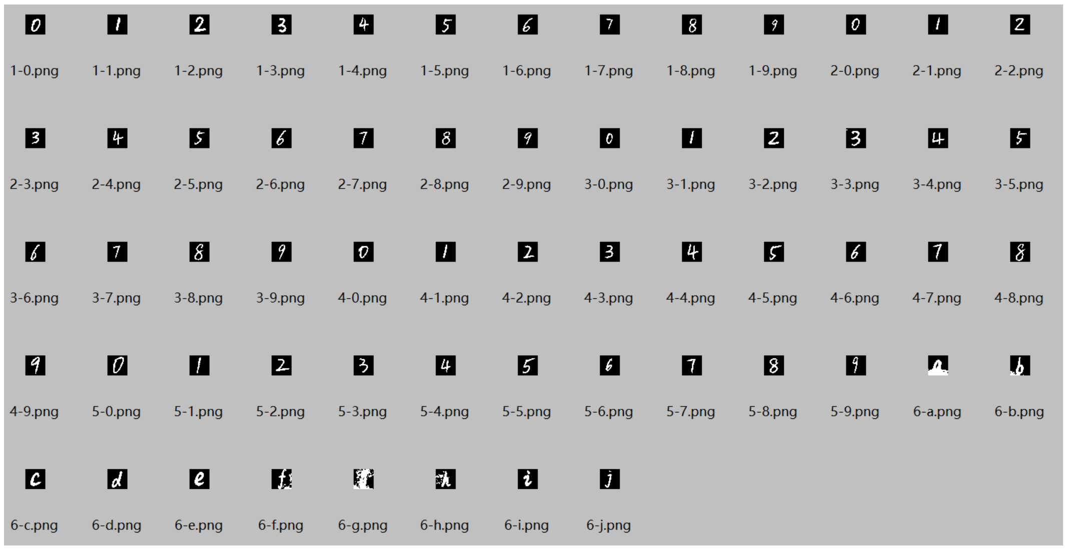
1. ***Create 60 handwritten characters (50 of these are the digits 0-9 handwritten (5 of each) number; 10 of these to be symbols that are not the characters 0-9.***



2. ***Photograph these characters into images.***


1-0.png 1-1.png 1-2.png 1-3.png 1-4.png 1-5.png 1-6.png 1-7.png 1-8.png 1-9.png 2-0.png 2-1.png 2-2.png
2-3.png 2-4.png 2-5.png 2-6.png 2-7.png 2-8.png 2-9.png 3-0.png 3-1.png 3-2.png 3-3.png 3-4.png 3-5.png
3-6.png 3-7.png 3-8.png 3-9.png 4-0.png 4-1.png 4-2.png 4-3.png 4-4.png 4-5.png 4-6.png 4-7.png 4-8.png
4-9.png 5-0.png 5-1.png 5-2.png 5-3.png 5-4.png 5-5.png 5-6.png 5-7.png 5-8.png 5-9.png 6-a.png 6-b.png
6-c.png 6-d.png 6-e.png 6-f.png 6-g.png 6-h.png 6-i.png 6-j.png

3. ***Preprocess these images as outlined above. First, threshold to make binary (bilevel) images and then resample to the required size and reposition according to the center of mass (COM). Your final images should be stored in .png format.***

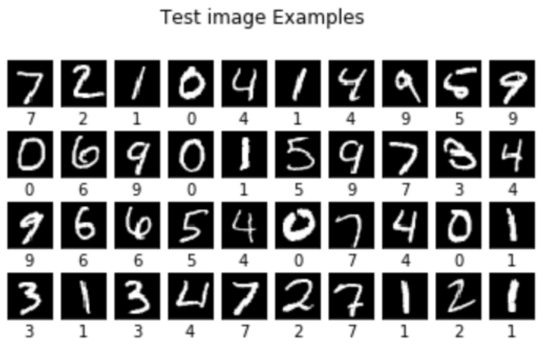

1-0.png 1-1.png 1-2.png 1-3.png 1-4.png 1-5.png 1-6.png 1-7.png 1-8.png 1-9.png 2-0.png 2-1.png 2-2.png
2-3.png 2-4.png 2-5.png 2-6.png 2-7.png 2-8.png 2-9.png 3-0.png 3-1.png 3-2.png 3-3.png 3-4.png 3-5.png
3-6.png 3-7.png 3-8.png 3-9.png 4-0.png 4-1.png 4-2.png 4-3.png 4-4.png 4-5.png 4-6.png 4-7.png 4-8.png
4-9.png 5-0.png 5-1.png 5-2.png 5-3.png 5-4.png 5-5.png 5-6.png 5-7.png 5-8.png 5-9.png 6-a.png 6-b.png
6-c.png 6-d.png 6-e.png 6-f.png 6-g.png 6-h.png 6-i.png 6-j.png

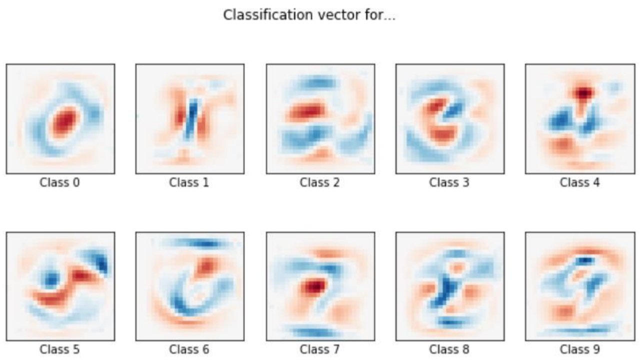
4. ***Invert images if necessary (vpix) to have light foreground and dark background.***

2. Experiments with Logistic Regression classifier

1. ***Starting with the classifier described in the Lab tutorial evaluate the performance of the classifier for both the standard test dataset and also your own dataset.***

2. ***Plot the images from the standard MNIST test set of the first 40 images with errors together with the correct and incorrect labels. Discuss these results.***





Sparsity with L1 penalty: 16.80%

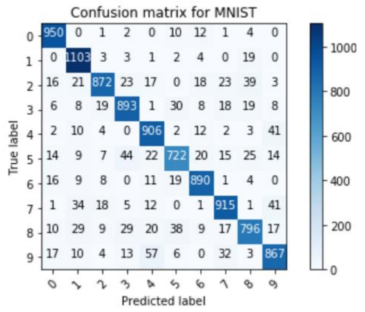
Test score with L1 penalty: 0.8914

y_predicted_values [b'7' b'2' b'1' b'0' b'4' b'1' b'4' b'9' **b'4'** b'9' b'0' b'6' b'9' b'0' b'1' b'5' b'9' b'7' b'3' b'4']

y_predicted_values [b'9' b'6' b'6' b'5' b'4' b'0' b'7' b'4' b'0' b'1' b'3' b'1' b'3' **b'0'** b'7' b'2' b'7' b'1' **b'3'** b'1']

The total score of Logistic Regression classifier is around 90%, which means a good result. Since the MNIST has 60000 images in its training set and the training set provide enough sample for test image, the final result of test shows that this classifier has a good performance on the detect of handwritten numbers.

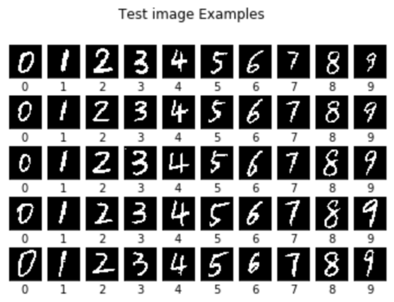
From 40 images shows above, we can find that only 3 of them are wrongly labeled. Although the accuracy is enough high, we can find some points to improve this classifier. For example, row 4 column 4, this "4" number is not written in normal way and it seems like letter "U" or "u". Hence, it is hard for our model to classify. Row 1 column 9, this number-"5", its bottom cannot be written in a "correct" way, the bottom of 5 is mixed. In my first time to look at this image, I made a mistake and thought it was number 4. For previous analyzation, writing the number in "correct" way is very important, in other words, the aim of machine learning is trying to find common features for regular things. If the handwritten number is not normal, it will be hard for model to make a correct prediction.



	0	1	2	3	4	5	6	7	8	9
0	950	0	1	2	0	10	12	1	4	0
1	0	1103	3	3	1	2	4	0	19	0
2	16	21	872	23	17	0	18	23	39	3
3	6	8	19	893	1	30	8	18	19	8
4	2	10	4	0	906	2	12	2	3	41
5	14	9	7	44	22	722	20	15	25	14
6	16	9	8	0	11	19	890	1	4	0
7	1	34	18	5	12	0	1	915	1	41
8	10	29	9	29	20	38	9	17	796	17
9	17	10	4	13	57	6	0	32	3	867

From the confusion matrix, we can find that this model actually has a high accuracy of prediction.

3. ***Plot all your test images with truth and predicted labels. Discuss these results.***

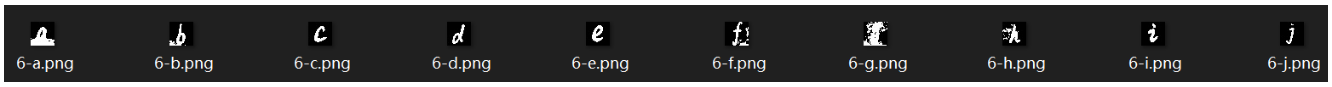


Test score with L1 penalty: 0.6400

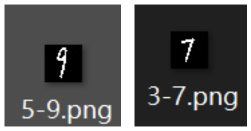
```
y_predicted_values    [0 1 2 3 4 5 0 1 8 7 9 1 3 3 4 5 6 1 3 9 9 1 2 3 4 5 5 1 8 1 9 1 2 3 4 5 6 1 3 7 6 1 2 3 4 5 6 1 8 1]
y_labels              [0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9]
```

The test score is 0.64 and compared with the MNIST, it is really relatively lower. The reason is similar with the previous part: not normal handwritten number, but in this question, I will show other problems.

- The illumination of photograph, we can see from the first picture, the bottom has a weaker light and letters from "a" to "j" have dark background, thus, after preprocess of images, image with bad illumination shows some problem as shown below:



- Some handwritten numbers are not continuous, the model will make a mistake when it classifies this number.
- Some numbers are similar: 0, 6, 9, since all of them has a "circle" in writing, I guess the model easily makes mistakes when they classify these numbers.
- The upper half part of number 9 or 7 is too thin and it looks like a line i.e. looks like "1":

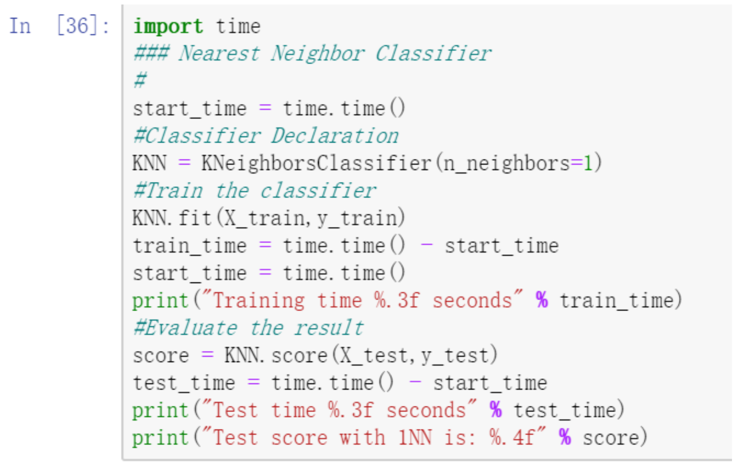


In my data set, numbers from 1 to 5 and 8 are almost classified correctly and other numbers make several mistakes. Hence, the result is level of pass. It needs to be improved.

<u>Part B</u>

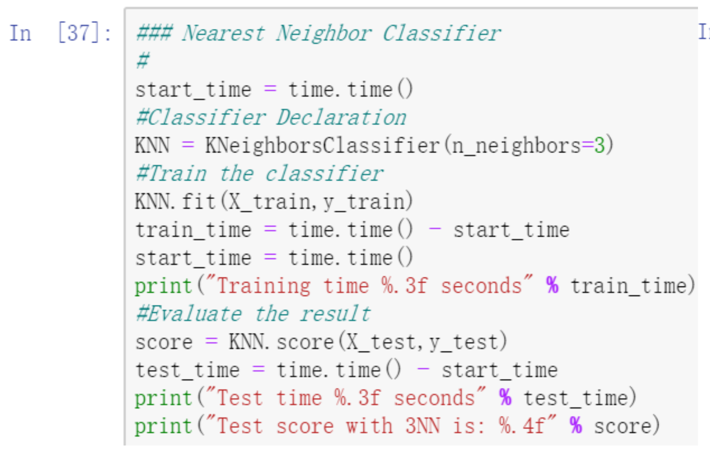
<u>2. Experiments with classifiers</u>

1. ***Evaluate a K-nearest-neighbor (K-NN) classifier for 1 and 3 near neighbors. Also, select a number of near neighbors for an additional test.*** The following using **<u>full</u>** data set (tested in Philip 314)

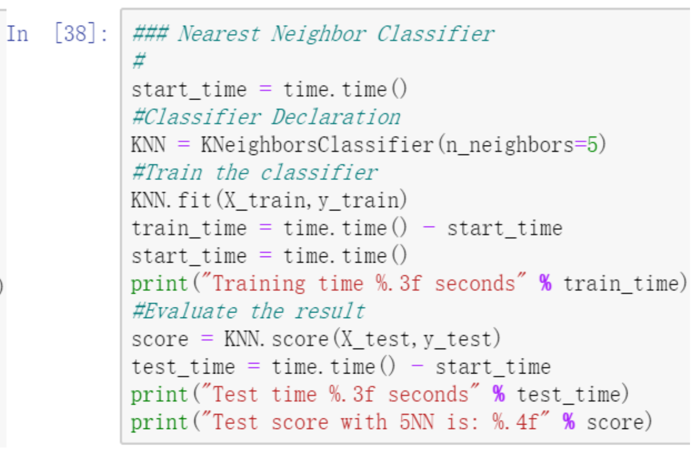


```
In  [36]: import time
          ### Nearest Neighbor Classifier
          #
          start_time = time.time()
          #Classifier Declaration
          KNN = KNeighborsClassifier(n_neighbors=1)
          #Train the classifier
          KNN.fit(X_train,y_train)
          train_time = time.time() - start_time
          start_time = time.time()
          print("Training time %.3f seconds" % train_time)
          #Evaluate the result
          score = KNN.score(X_test,y_test)
          test_time = time.time() - start_time
          print("Test time %.3f seconds" % test_time)
          print("Test score with 1NN is: %.4f" % score)
```

Training time 13.346 seconds; Test time 557.796 seconds; Test score with 1NN is: 0.9434



```
In [37]: ### Nearest Neighbor Classifier
         #
         start_time = time.time()
         #Classifier Declaration
         KNN = KNeighborsClassifier(n_neighbors=3)
         #Train the classifier
         KNN.fit(X_train, y_train)
         train_time = time.time() - start_time
         start_time = time.time()
         print("Training time %.3f seconds" % train_time)
         #Evaluate the result
         score = KNN.score(X_test, y_test)
         test_time = time.time() - start_time
         print("Test time %.3f seconds" % test_time)
         print("Test score with 3NN is: %.4f" % score)
```



```
In [38]: ### Nearest Neighbor Classifier
         #
         start_time = time.time()
         #Classifier Declaration
         KNN = KNeighborsClassifier(n_neighbors=5)
         #Train the classifier
         KNN.fit(X_train, y_train)
         train_time = time.time() - start_time
         start_time = time.time()
         print("Training time %.3f seconds" % train_time)
         #Evaluate the result
         score = KNN.score(X_test, y_test)
         test_time = time.time() - start_time
         print("Test time %.3f seconds" % test_time)
         print("Test score with 5NN is: %.4f" % score)
```

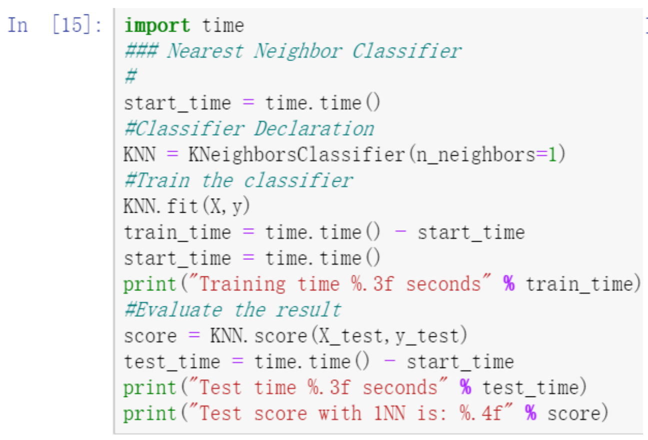
k=3: Training time 14.638 seconds; Test time 565.762 seconds; Test score with 3NN is: 0.9452

k=5: Training time 13.451 seconds; Test time 571.444 seconds; Test score with 5NN is: 0.9443

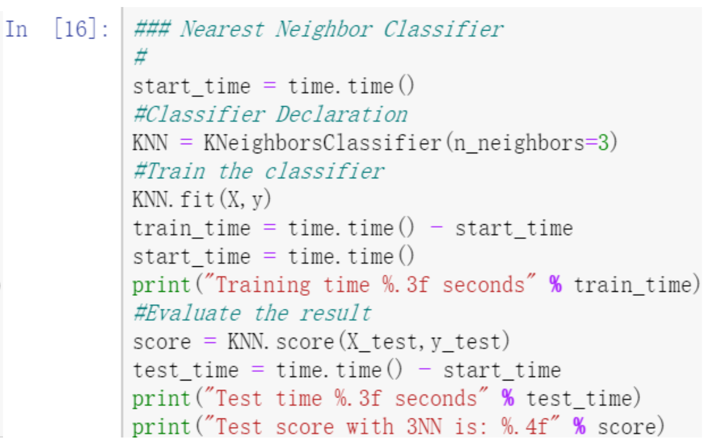
With the change of value of k, the training time does not change obviously. Since the kNN model is an unsupervised learning process, the training time is very small, but test time is very large. In KNN model test stage, one test sample is compared with every sample in the training sample. It really a high cost of computation. The training time becomes larger with the increment of k since when k=1, it only need to find 1 nearest neighbor, when k=3, it need to find 3 nearest neighbor, the same as k=5, the larger k, the more neighbors need to be compared, the higher computation. With the increment of value of k, 3 scores of kNN are similar since we have enough training and test data set.

The K-NN classifier may take a long time to execute on the full MNIST dataset. Intelligently modify the size of your training and testing datasets so that the evaluation time on your computing platform is less than 10 minutes. Show the time in seconds taken for training and testing. Discuss the expected impact of reducing the size of the dataset on your results.

The size of training data set is 60000, the size of test data set is 500 (test on my laptop)



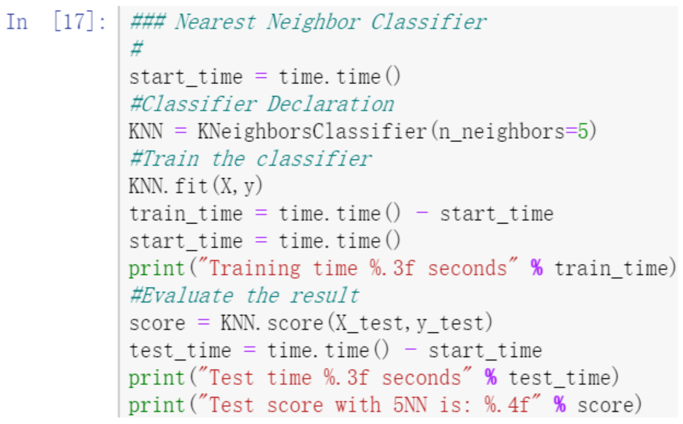
```
In [15]: import time
         ### Nearest Neighbor Classifier
         #
         start_time = time.time()
         #Classifier Declaration
         KNN = KNeighborsClassifier(n_neighbors=1)
         #Train the classifier
         KNN.fit(X, y)
         train_time = time.time() - start_time
         start_time = time.time()
         print("Training time %.3f seconds" % train_time)
         #Evaluate the result
         score = KNN.score(X_test, y_test)
         test_time = time.time() - start_time
         print("Test time %.3f seconds" % test_time)
         print("Test score with 1NN is: %.4f" % score)
```



```
In [16]: ### Nearest Neighbor Classifier
         #
         start_time = time.time()
         #Classifier Declaration
         KNN = KNeighborsClassifier(n_neighbors=3)
         #Train the classifier
         KNN.fit(X, y)
         train_time = time.time() - start_time
         start_time = time.time()
         print("Training time %.3f seconds" % train_time)
         #Evaluate the result
         score = KNN.score(X_test, y_test)
         test_time = time.time() - start_time
         print("Test time %.3f seconds" % test_time)
         print("Test score with 3NN is: %.4f" % score)
```

k=1: Training time 24.942; seconds Test time 73.592 seconds; Test score with 1NN is: 1.0000

k=3: Training time 27.534 seconds; Test time 75.673 seconds; Test score with 3NN is: 0.9620



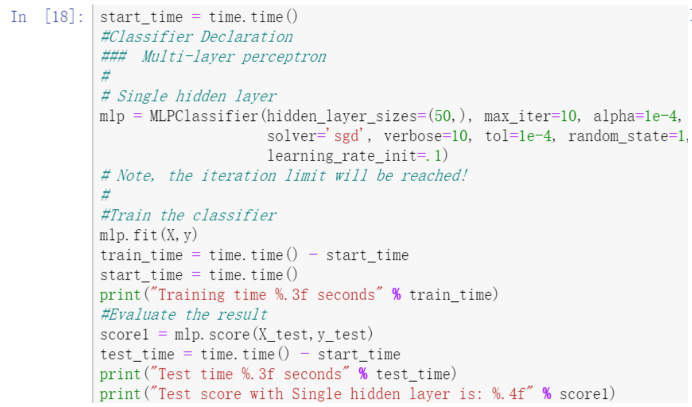
```
In [17]: ### Nearest Neighbor Classifier
         #
         start_time = time.time()
         #Classifier Declaration
         KNN = KNeighborsClassifier(n_neighbors=5)
         #Train the classifier
         KNN.fit(X, y)
         train_time = time.time() - start_time
         start_time = time.time()
         print("Training time %.3f seconds" % train_time)
         #Evaluate the result
         score = KNN.score(X_test, y_test)
         test_time = time.time() - start_time
         print("Test time %.3f seconds" % test_time)
         print("Test score with 5NN is: %.4f" % score)
```

k=5: Training time 26.192 seconds; Test time 80.598 seconds; Test score with 5NN is: 0.9420

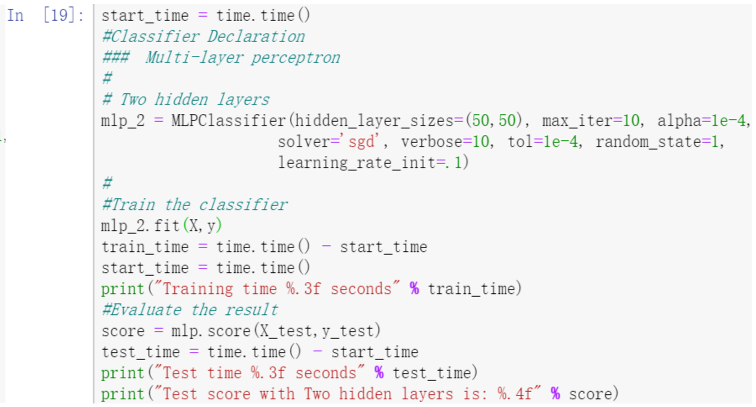
Although the training time increases obviously compared with the previous data set due to the different of computing performance between laptop and PC, the testing time reduce significantly. Reducing the size of testing set improves the test score because not all the sample in testing data set are tested, maybe the sample which will be wrongly classified are ignored. As mentioned above, testing sample have to be compared with all the sample in training set. Hence, reducing size of testing set leads to decrement of time cost of computation. With the increment of value of k, the training time increases and the reason has introduced in previous part.

2. *Evaluate two multi-layer perceptron classifiers one with a single hidden layer and one with two hidden layers. As with the K-NN classifier, intelligently modify the number the dataset to complete each evaluation within 10 minutes.*

The size of training data set is 60000, the size of test data set is 10000 (test on my laptop)



```
In [18]: start_time = time.time()
         #Classifier Declaration
         ###  Multi-layer perceptron
         #
         # Single hidden layer
         mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=10, alpha=1e-4,
                             solver='sgd', verbose=10, tol=1e-4, random_state=1,
                             learning_rate_init=.1)
         # Note, the iteration limit will be reached!
         #
         #Train the classifier
         mlp.fit(X,y)
         train_time = time.time() - start_time
         start_time = time.time()
         print("Training time %.3f seconds" % train_time)
         #Evaluate the result
         score1 = mlp.score(X_test,y_test)
         test_time = time.time() - start_time
         print("Test time %.3f seconds" % test_time)
         print("Test score with Single hidden layer is: %.4f" % score1)
```



```
In [19]: start_time = time.time()
         #Classifier Declaration
         ###  Multi-layer perceptron
         #
         # Two hidden layers
         mlp_2 = MLPClassifier(hidden_layer_sizes=(50,50), max_iter=10, alpha=1e-4,
                               solver='sgd', verbose=10, tol=1e-4, random_state=1,
                               learning_rate_init=.1)
         #
         #Train the classifier
         mlp_2.fit(X,y)
         train_time = time.time() - start_time
         start_time = time.time()
         print("Training time %.3f seconds" % train_time)
         #Evaluate the result
         score = mlp.score(X_test,y_test)
         test_time = time.time() - start_time
         print("Test time %.3f seconds" % test_time)
         print("Test score with Two hidden layers is: %.4f" % score)
```

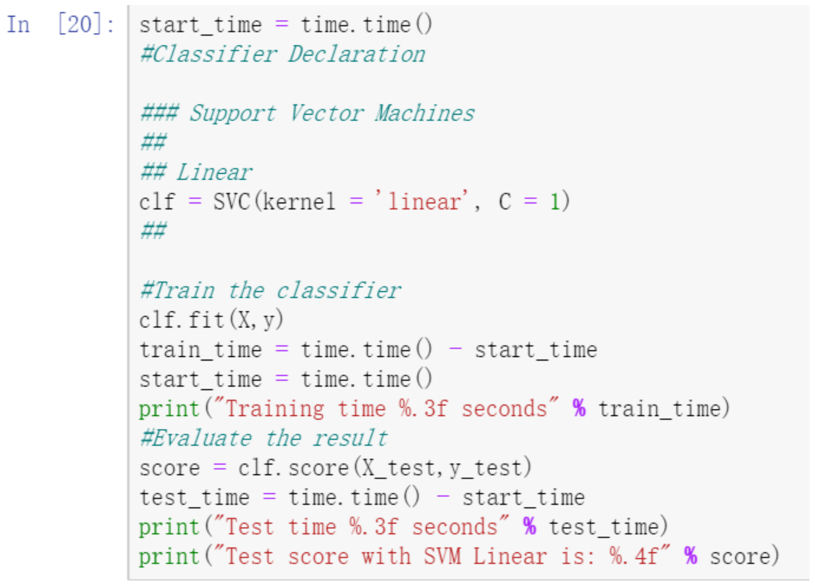
Single hidden layer: Iteration 10, loss = 0.04645355; Training time 8.328 seconds; Test time 0.050 seconds; Test score with Single hidden layer is: 0.9700

Two hidden layers: Iteration 10, loss = 0.04180814; Training time 10.277 seconds; Test time 0.251 seconds; Test score with Two hidden layers is: 0.9868

The perceptron is supervised learning and thus, its training time is much larger than test time. Maybe the data set is good, the test score is very high. The more layers, the more steps to update the parameters and hence, the better test score and the smaller loss.

3. *Evaluate three support vector machine classifiers; one with a linear function, one with a polynomial function and one with radial basis function. As with the K-NN classifier intelligently modify the number the dataset to complete each evaluation within 10 minutes.*

The size of training data set is 60000, the size of test data set is 10000 (test in Philips 314)

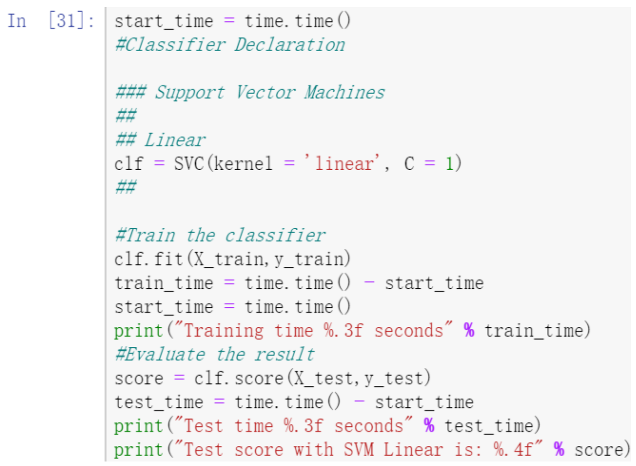


```
In [20]: start_time = time.time()
         #Classifier Declaration

         ### Support Vector Machines
         ##
         ## Linear
         clf = SVC(kernel = 'linear', C = 1)
         ##

         #Train the classifier
         clf.fit(X,y)
         train_time = time.time() - start_time
         start_time = time.time()
         print("Training time %.3f seconds" % train_time)
         #Evaluate the result
         score = clf.score(X_test,y_test)
         test_time = time.time() - start_time
         print("Test time %.3f seconds" % test_time)
         print("Test score with SVM Linear is: %.4f" % score)
```

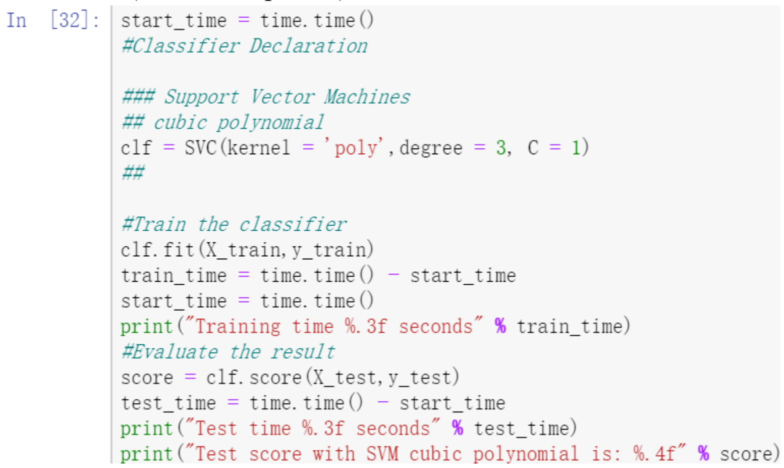
Training time 545.283 seconds; Test time 76.984 seconds; Test score with SVM Linear is: 0.9280

The linear SVM is the simplest one of SVM model and if I use the full data set, the time cost is bigger than 10 mins. Hence, I have to reduce the size of data set. Since the SVM is supervised learning, as mentioned above, its training time is much larger than test time.

The size of training data set is 20000, the size of test data set is 500 (test in Philips 314)



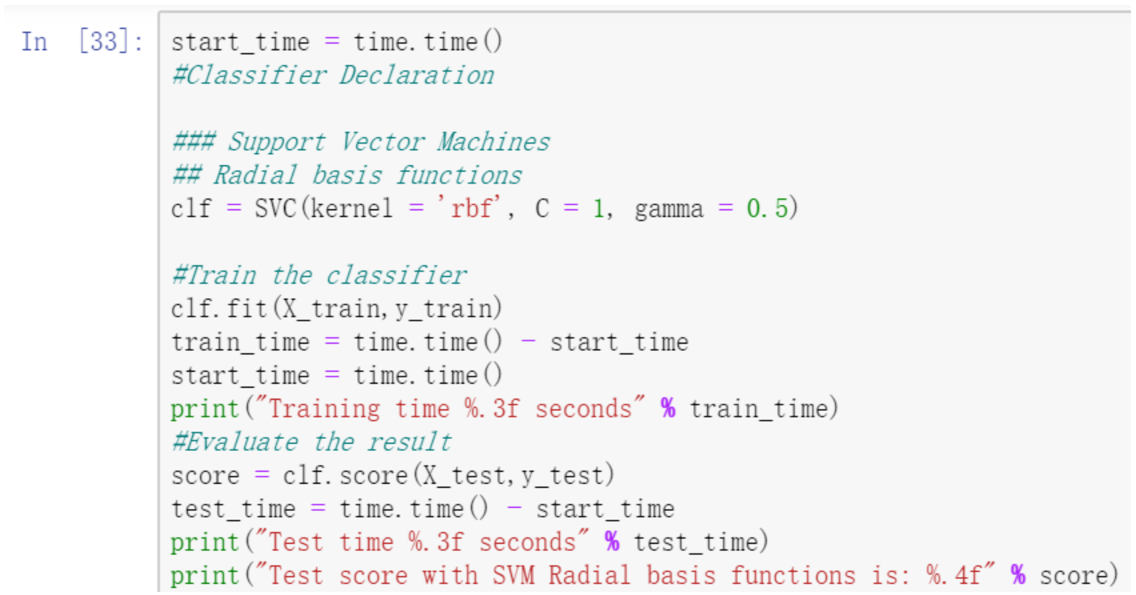
```
In [31]: start_time = time.time()
         #Classifier Declaration

         ### Support Vector Machines
         ##
         ## Linear
         clf = SVC(kernel = 'linear', C = 1)
         ##

         #Train the classifier
         clf.fit(X_train, y_train)
         train_time = time.time() - start_time
         start_time = time.time()
         print("Training time %.3f seconds" % train_time)
         #Evaluate the result
         score = clf.score(X_test, y_test)
         test_time = time.time() - start_time
         print("Test time %.3f seconds" % test_time)
         print("Test score with SVM Linear is: %.4f" % score)
```



```
In [32]: start_time = time.time()
         #Classifier Declaration

         ### Support Vector Machines
         ## cubic polynomial
         clf = SVC(kernel = 'poly', degree = 3, C = 1)
         ##

         #Train the classifier
         clf.fit(X_train, y_train)
         train_time = time.time() - start_time
         start_time = time.time()
         print("Training time %.3f seconds" % train_time)
         #Evaluate the result
         score = clf.score(X_test, y_test)
         test_time = time.time() - start_time
         print("Test time %.3f seconds" % test_time)
         print("Test score with SVM cubic polynomial is: %.4f" % score)
```

Linear: Training time 42.260 seconds; Test time 1.749 seconds; Test score with SVM Linear is: 0.8900

Poly: Training time 236.083 seconds; Test time 4.347 seconds; Test score with SVM cubic polynomial is: 0.9080



```
In [33]: start_time = time.time()
         #Classifier Declaration

         ### Support Vector Machines
         ## Radial basis functions
         clf = SVC(kernel = 'rbf', C = 1, gamma = 0.5)

         #Train the classifier
         clf.fit(X_train, y_train)
         train_time = time.time() - start_time
         start_time = time.time()
         print("Training time %.3f seconds" % train_time)
         #Evaluate the result
         score = clf.score(X_test, y_test)
         test_time = time.time() - start_time
         print("Test time %.3f seconds" % test_time)
         print("Test score with SVM Radial basis functions is: %.4f" % score)
```

rbf: Training time 669.154 seconds; Test time 8.803 seconds; Test score with SVM Radial basis functions is: 0.1200

From above figures and lecture notes, we can find the construction of rbf is more complex than poly, the poly one is more complex than linear one. Hence, the time cost of computation of training and testing increase obviously according to the complexity of model. If we use the full data set, poly model may use more than 1 hour and rbf may use more than 3 hours. After reducing the size of data set, we can find poly one has higher score than linear. According to lecture notes, poly shows a better performance than linear one. But the rbf score is 0.12, which is very low compared with others, the reason maybe we use limited training data set and the trained model is not complete and its test result is not ideal. Since training rbf model with full size has a high time cost, if we have Tensorflow platform and use GPU for computation, may be the result becomes better.