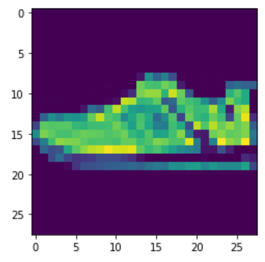
ASSIGNMENT 2 R00122087:

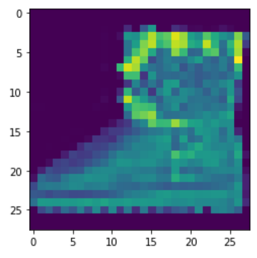
Task 1:

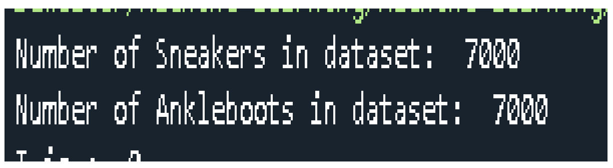
For task 1 I read in the data frame and create 2 subframes of ankle boots and sneakers separating by label. Next, I create 2 sub frames to store the labels before dropping the labels in the data sets. With that done I get the number of samples in each sub frame using the len() function as each row represents a sample in the form of 28x28 pixel values.

Finally, I plot out the first row of each dataset to display the image of both an ankle boot and sneaker and reshaping the image to 28x28 to display each sample correctly.

These steps can be seen in the code from line 21 to 52.





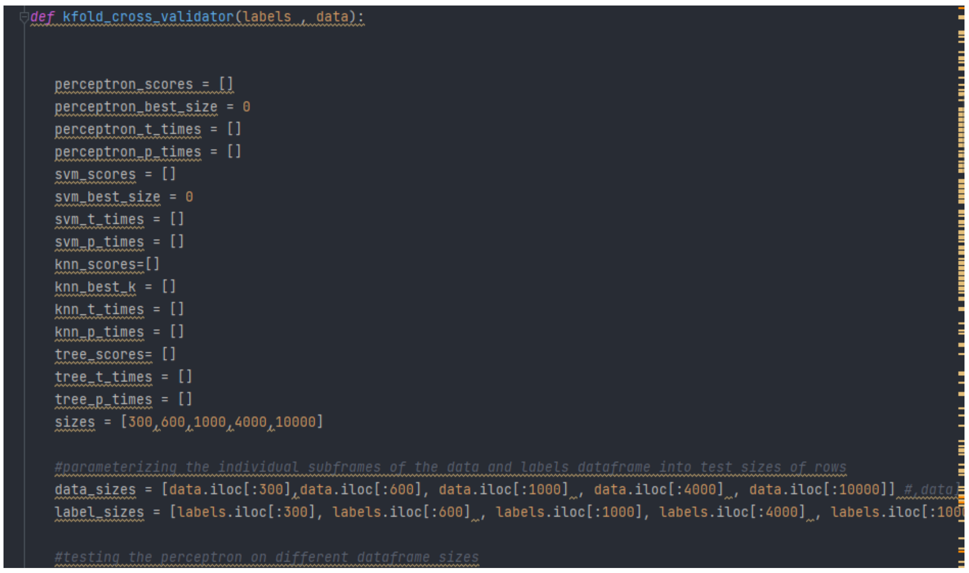


```
Number of Sneakers in dataset:  7000
Number of Ankleboots in dataset:  7000
```

Task 2:

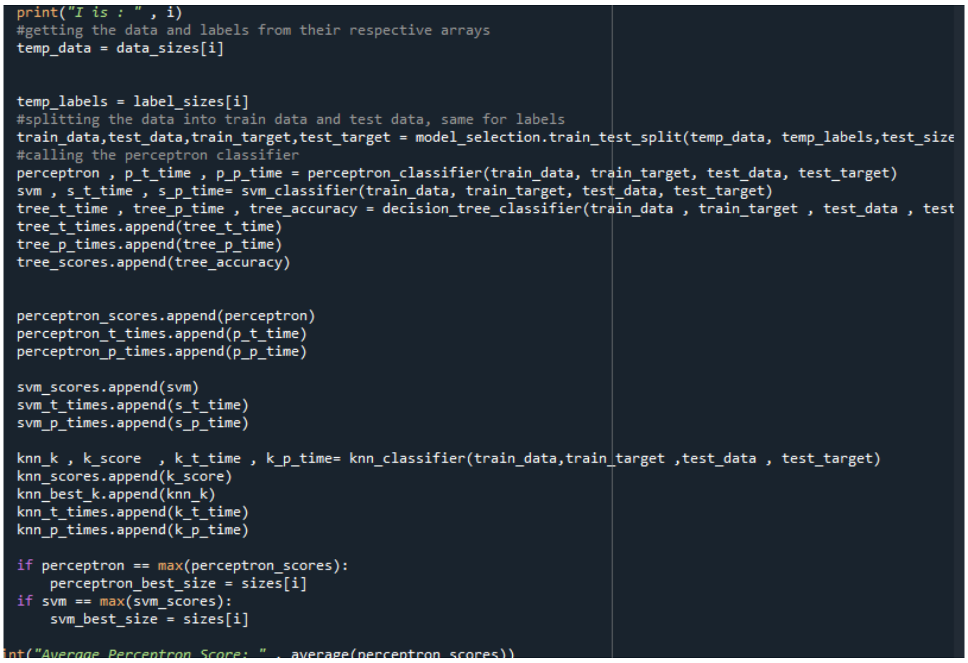
Task 2 is viewable from line 54 to 174.

In task 2 I setup arrays that the relevant factors for each classifier are stored as well as the data samples of individual sizes.



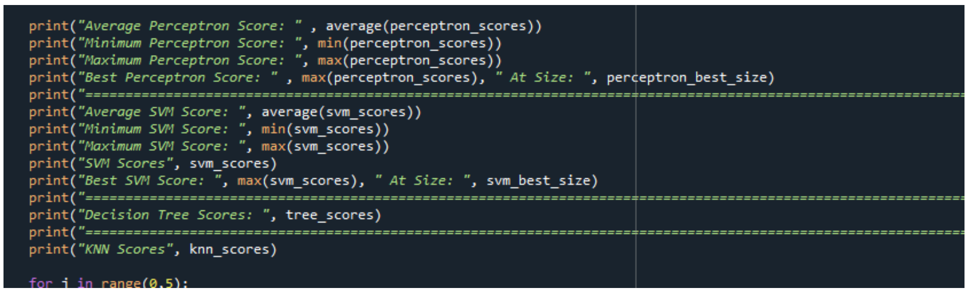
```
def kfold_cross_validator(labels , data):


    perceptron_scores = []
    perceptron_best_size = 0
    perceptron_t_times = []
    perceptron_p_times = []
    svm_scores = []
    svm_best_size = 0
    svm_t_times = []
    svm_p_times = []
    knn_scores=[]
    knn_best_k = []
    knn_t_times = []
    knn_p_times = []
    tree_scores= []
    tree_t_times = []
    tree_p_times = []
    sizes = [300,600,1000,4000,10000]

    #parameterizing the individual subframes of the data and labels dataframe into test sizes of rows
    data_sizes = [data.iloc[:300],data.iloc[:600], data.iloc[:1000] , data.iloc[:4000] , data.iloc[:10000]] #,data
    label_sizes = [labels.iloc[:300], labels.iloc[:600] , labels.iloc[:1000], labels.iloc[:4000] , labels.iloc[:100

    #testing the perceptron on different dataframe sizes
```

After these are initialised I run each classifier for each data frame size ranging from 300 to 10,00.



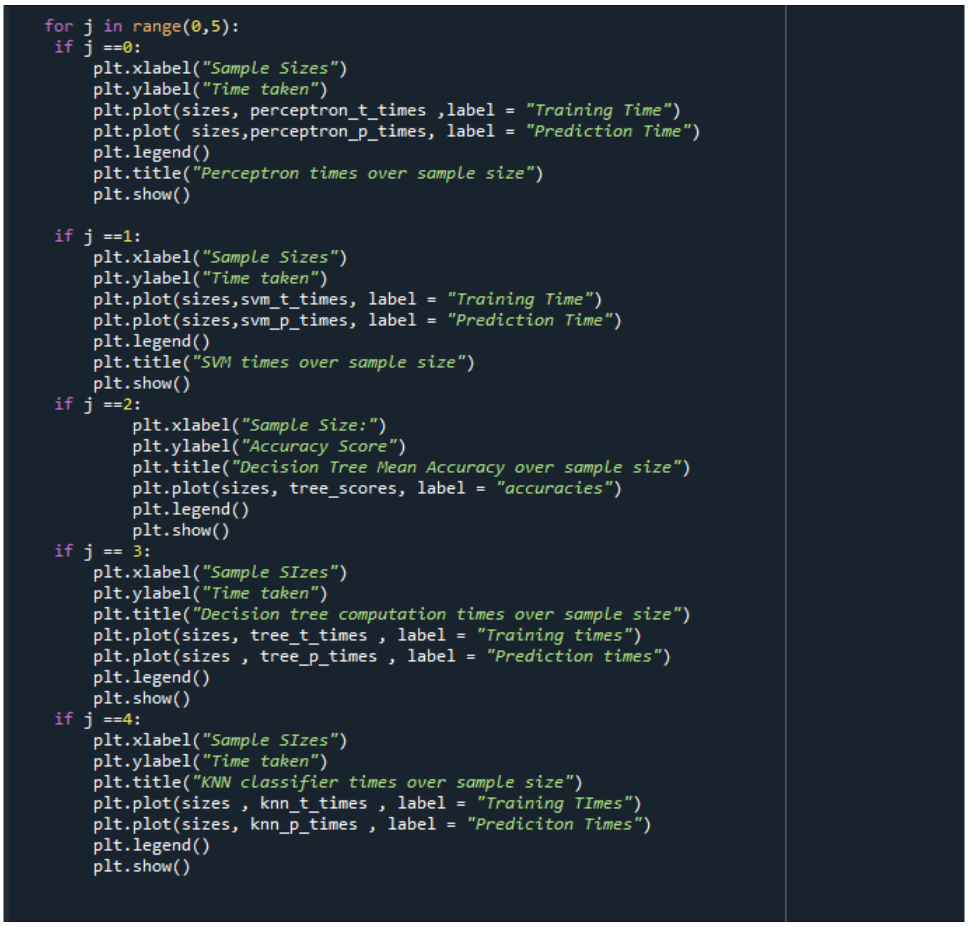
```
print("I is : " , i)
#getting the data and labels from their respective arrays
temp_data = data_sizes[i]


temp_labels = label_sizes[i]
#splitting the data into train data and test data, same for labels
train_data,test_data,train_target,test_target = model_selection.train_test_split(temp_data, temp_labels,test_size
#calling the perceptron classifier
perceptron , p_t_time , p_p_time = perceptron_classifier(train_data, train_target, test_data, test_target)
svm , s_t_time , s_p_time= svm_classifier(train_data, train_target, test_data, test_target)
tree_t_time , tree_p_time , tree_accuracy = decision_tree_classifier(train_data , train_target , test_data , test
tree_t_times.append(tree_t_time)
tree_p_times.append(tree_p_time)
tree_scores.append(tree_accuracy)


perceptron_scores.append(perceptron)
perceptron_t_times.append(p_t_time)
perceptron_p_times.append(p_p_time)

svm_scores.append(svm)
svm_t_times.append(s_t_time)
svm_p_times.append(s_p_time)

knn_k , k_score  , k_t_time , k_p_time= knn_classifier(train_data,train_target ,test_data , test_target)
knn_scores.append(k_score)
knn_best_k.append(knn_k)
knn_t_times.append(k_t_time)
knn_p_times.append(k_p_time)

if perceptron == max(perceptron_scores):
    perceptron_best_size = sizes[i]
if svm == max(svm_scores):
    svm_best_size = sizes[i]

int("Average Perceptron Score: " , average(perceptron_scores))
```

Each classifier function returns their score, training time and prediction time. After each classifier is run on all sample sizes, I print out the results which can be seen below in the end data section.



```
print("Average Perceptron Score: " , average(perceptron_scores))
print("Minimum Perceptron Score: ", min(perceptron_scores))
print("Maximum Perceptron Score: ", max(perceptron_scores))
print("Best Perceptron Score: " , max(perceptron_scores), " At Size: ", perceptron_best_size)
print("===============================================================================================================
print("Average SVM Score: ", average(svm_scores))
print("Minimum SVM Score: ", min(svm_scores))
print("Maximum SVM Score: ", max(svm_scores))
print("SVM Scores", svm_scores)
print("Best SVM Score: ", max(svm_scores), " At Size: ", svm_best_size)
print("===============================================================================================================
print("Decision Tree Scores: ", tree_scores)
print("===============================================================================================================
print("KNN Scores", knn_scores)

for i in range(0,5):
```

Finally, I graph out the data required for each classifier,



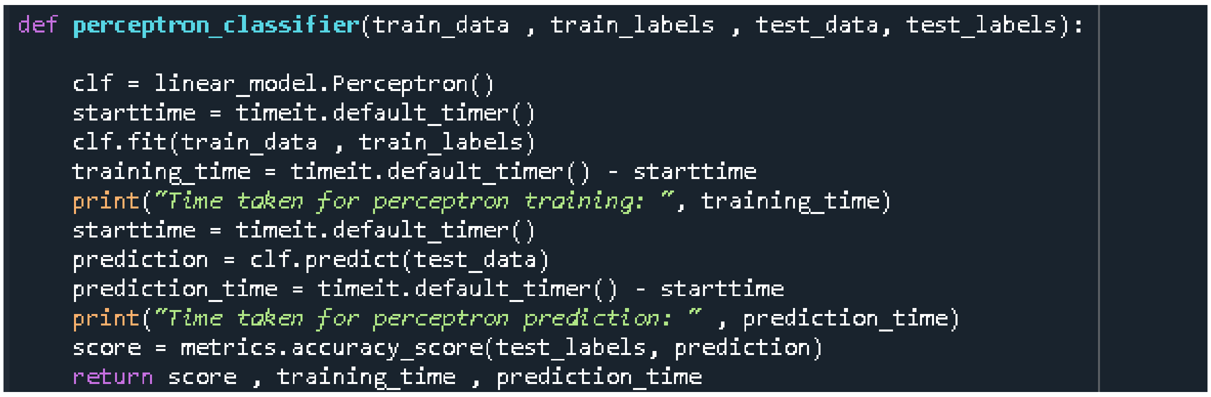
```
for j in range(0,5):
  if j ==0:
      plt.xlabel("Sample Sizes")
      plt.ylabel("Time taken")
      plt.plot(sizes, perceptron_t_times ,label = "Training Time")
      plt.plot( sizes,perceptron_p_times, label = "Prediction Time")
      plt.legend()
      plt.title("Perceptron times over sample size")
      plt.show()

  if j ==1:
      plt.xlabel("Sample Sizes")
      plt.ylabel("Time taken")
      plt.plot(sizes,svm_t_times, label = "Training Time")
      plt.plot(sizes,svm_p_times, label = "Prediction Time")
      plt.legend()
      plt.title("SVM times over sample size")
      plt.show()
  if j ==2:
          plt.xlabel("Sample Size:")
          plt.ylabel("Accuracy Score")
          plt.title("Decision Tree Mean Accuracy over sample size")
          plt.plot(sizes, tree_scores, label = "accuracies")
          plt.legend()
          plt.show()
  if j == 3:
      plt.xlabel("Sample SIzes")
      plt.ylabel("Time taken")
      plt.title("Decision tree computation times over sample size")
      plt.plot(sizes, tree_t_times , label = "Training times")
      plt.plot(sizes , tree_p_times , label = "Prediction times")
      plt.legend()
      plt.show()
  if j ==4:
      plt.xlabel("Sample SIzes")
      plt.ylabel("Time taken")
      plt.title("KNN classifier times over sample size")
      plt.plot(sizes , knn_t_times , label = "Training TImes")
      plt.plot(sizes, knn_p_times , label = "Prediciton Times")
      plt.legend()
      plt.show()
```

Task 3 - Perceptron:

The perceptron is classifier is created in the perceptron_classifier function which can be viewed in line 159 to 171.

Passing in the train and test data and label variables created in task 2 and is rather straightforward and I keep track of the time it takes for both the training and predicting of the perceptron using the timeit module.

The training and prediction times are passed back to task2 for later use as well as the accuracy score of the perceptron prediction.

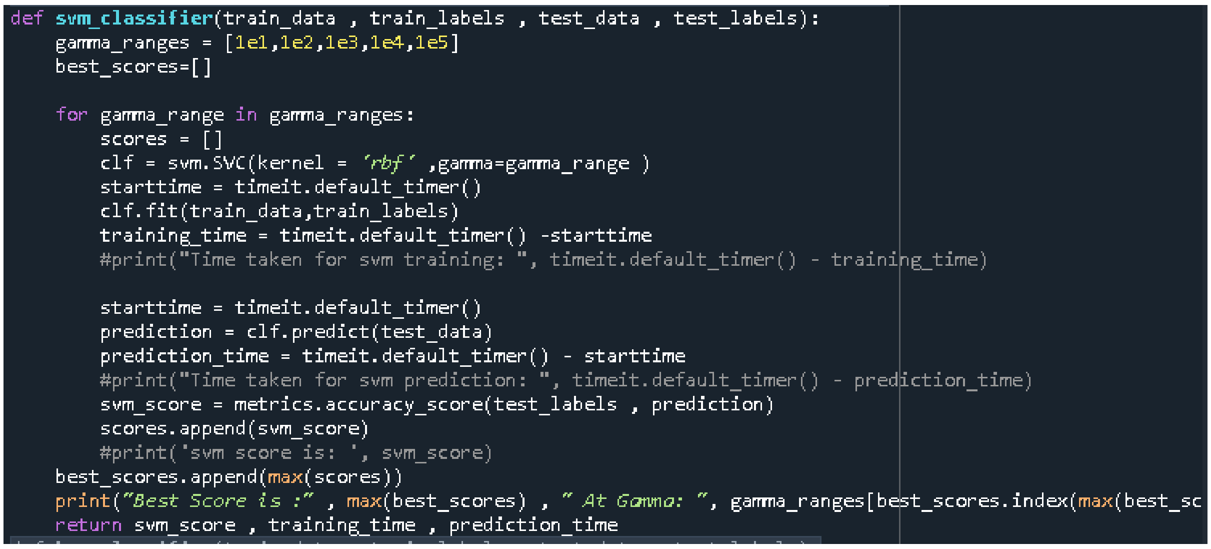


```
def perceptron_classifier(train_data , train_labels , test_data, test_labels):

    clf = linear_model.Perceptron()
    starttime = timeit.default_timer()
    clf.fit(train_data , train_labels)
    training_time = timeit.default_timer() - starttime
    print("Time taken for perceptron training: ", training_time)
    starttime = timeit.default_timer()
    prediction = clf.predict(test_data)
    prediction_time = timeit.default_timer() - starttime
    print("Time taken for perceptron prediction: " , prediction_time)
    score = metrics.accuracy_score(test_labels, prediction)
    return score , training_time , prediction_time
```

Task 4 Support Vector Machine:

The code for the svm classifier is viewable from line 173 to 193.I begin my declaring an array to hold the gamma values which a for loop will iterate through and produce 5 different sets of results representing a svm classifier using each gamma value from 1e1 to 1e5.

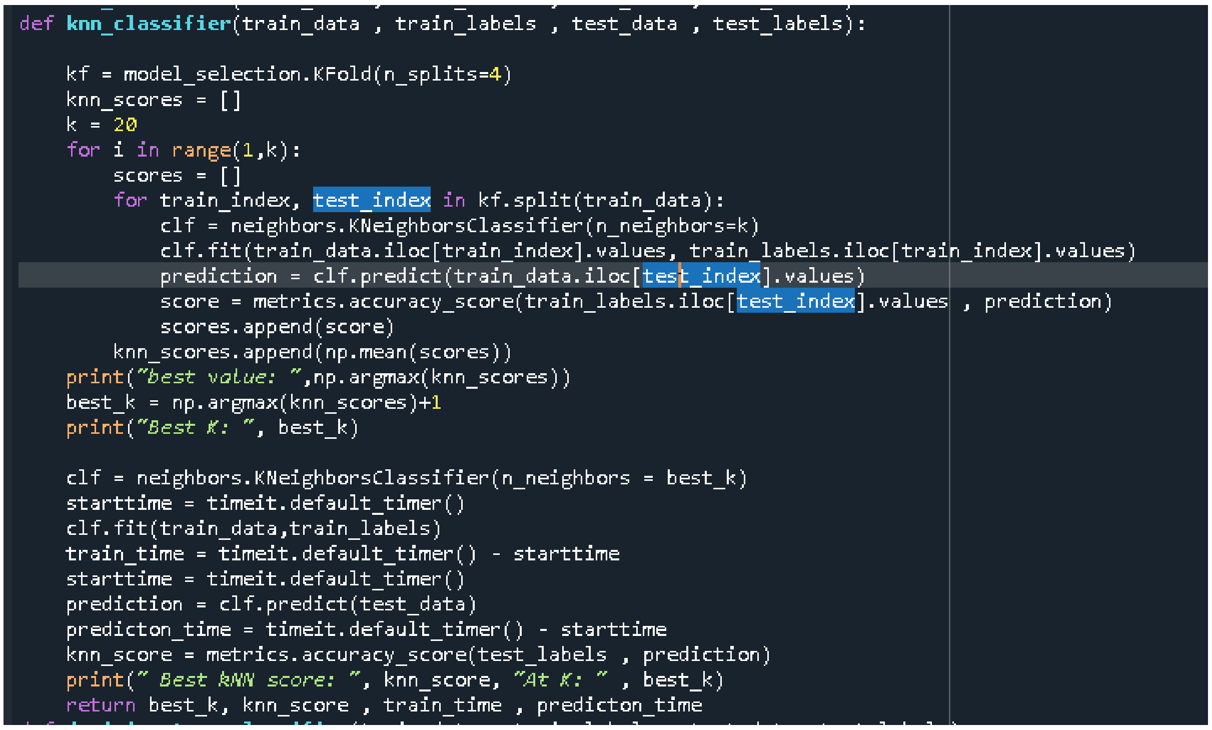
I track the scored of these individual prediction values for each gamma and also track the training and prediction times once again for the svm which are passed back to the task 2 function for future use. The accuracy score is also passed back.



```
def svm_classifier(train_data , train_labels , test_data , test_labels):
    gamma_ranges = [1e1,1e2,1e3,1e4,1e5]
    best_scores=[]

    for gamma_range in gamma_ranges:
        scores = []
        clf = svm.SVC(kernel = 'rbf' ,gamma=gamma_range )
        starttime = timeit.default_timer()
        clf.fit(train_data,train_labels)
        training_time = timeit.default_timer() -starttime
        #print("Time taken for svm training: ", timeit.default_timer() - training_time)

        starttime = timeit.default_timer()
        prediction = clf.predict(test_data)
        prediction_time = timeit.default_timer() - starttime
        #print("Time taken for svm prediction: ", timeit.default_timer() - prediction_time)
        svm_score = metrics.accuracy_score(test_labels , prediction)
        scores.append(svm_score)
        #print('svm score is: ', svm_score)
    best_scores.append(max(scores))
    print("Best Score is :" , max(best_scores) , " At Gamma: ", gamma_ranges[best_scores.index(max(best_sc
    return svm_score , training_time , prediction_time
```

Task 5 KNN:

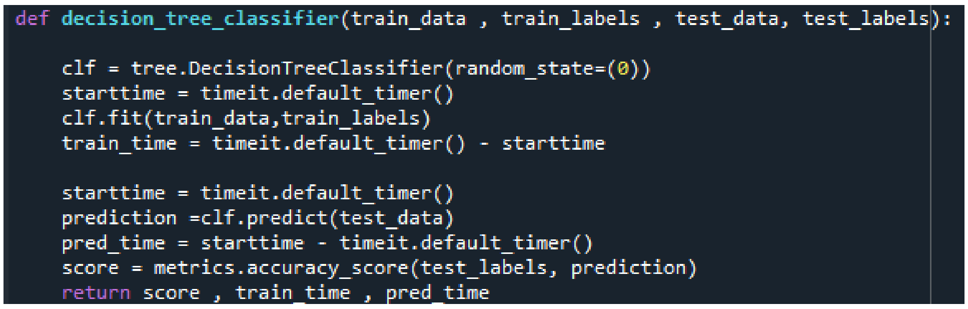
The knn_classifier function is viewable between lines 194 and221. I create a Kfold with 4 splits and use the variable k value to vary the amount of k's I use. The best k is determined by assessing the accuracy scores created by the loop and selecting the best one. I then create a new classifier to train and predict using that k value. I then record that accuracy and return it as well as the best k value. Once again, the time for training the model and predicting are recorded, however in this case I record the times for the best k only.



```
def knn_classifier(train_data , train_labels , test_data , test_labels):

    kf = model_selection.KFold(n_splits=4)
    knn_scores = []
    k = 20
    for i in range(1,k):
        scores = []
        for train_index, test_index in kf.split(train_data):
            clf = neighbors.KNeighborsClassifier(n_neighbors=k)
            clf.fit(train_data.iloc[train_index].values, train_labels.iloc[train_index].values)
            prediction = clf.predict(train_data.iloc[test_index].values)
            score = metrics.accuracy_score(train_labels.iloc[test_index].values , prediction)
            scores.append(score)
        knn_scores.append(np.mean(scores))
    print("best value: ",np.argmax(knn_scores))
    best_k = np.argmax(knn_scores)+1
    print("Best K: ", best_k)

    clf = neighbors.KNeighborsClassifier(n_neighbors = best_k)
    starttime = timeit.default_timer()
    clf.fit(train_data,train_labels)
    train_time = timeit.default_timer() - starttime
    starttime = timeit.default_timer()
    prediction = clf.predict(test_data)
    predicton_time = timeit.default_timer() - starttime
    knn_score = metrics.accuracy_score(test_labels , prediction)
    print(" Best kNN score: ", knn_score, "At K: " , best_k)
    return best_k, knn_score , train_time , predicton_time
```

Task 6 Decision Tree's:

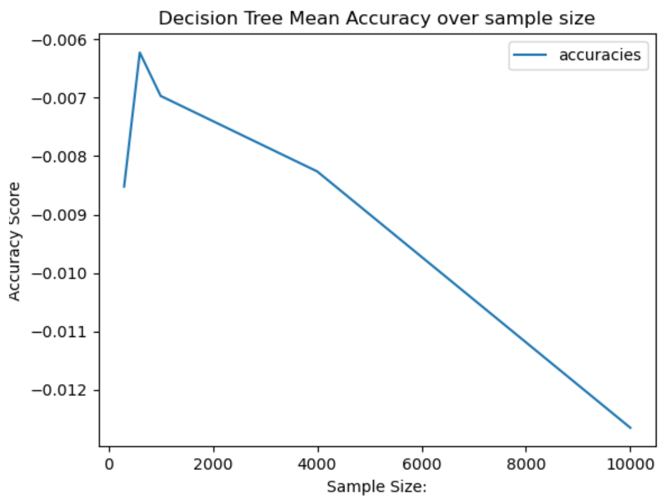
The code for the decision tree can be found at Lines 234 to 245.



```
def decision_tree_classifier(train_data , train_labels , test_data, test_labels):

    clf = tree.DecisionTreeClassifier(random_state=(0))
    starttime = timeit.default_timer()
    clf.fit(train_data,train_labels)
    train_time = timeit.default_timer() - starttime

    starttime = timeit.default_timer()
    prediction =clf.predict(test_data)
    pred_time = starttime - timeit.default_timer()
    score = metrics.accuracy_score(test_labels, prediction)
    return score , train_time , pred_time
```

I know that there is an issue with how I implemented the decision tree as can be seen in the below image.

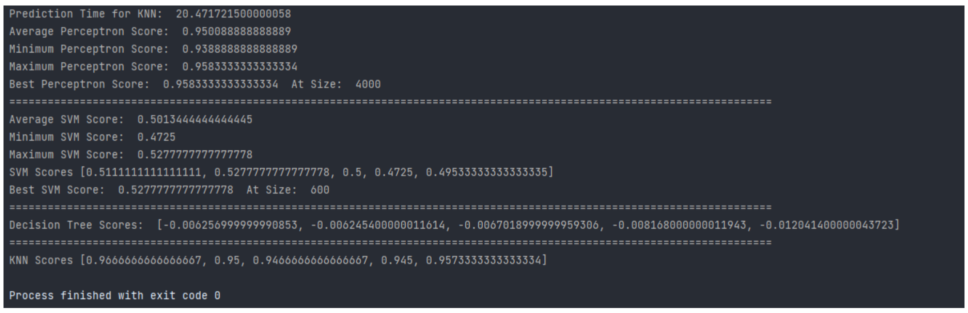
With more samples the accuracy gets progressively worse, and this is due to the lack of a max depth parameter. While not specified in the assignment specification I believe adding one would benefit the decision tree greatly but due to a lack of time this could not be done. With a max depth would also come increased training time and finding an effective depth for a classifier that needs to read a noticeable amount of 784 data points would be very costly.



End Data:

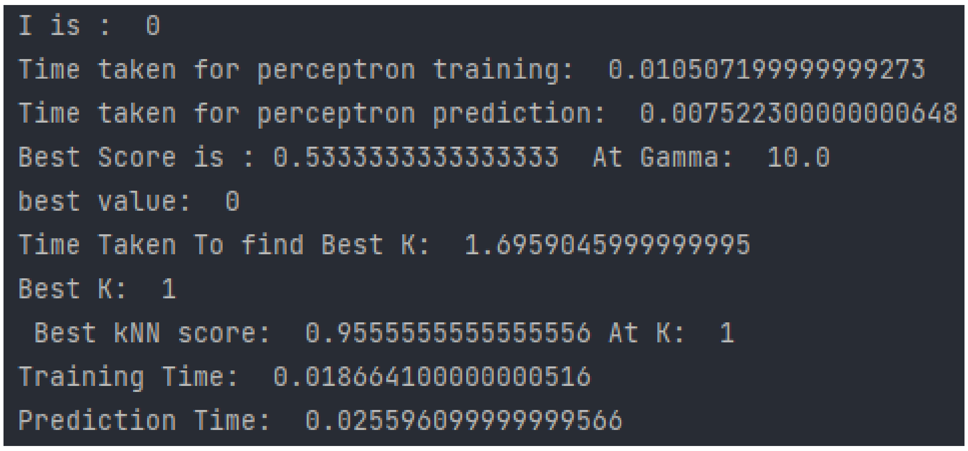
This is an example of the data printed at the end of task 2.

It shows the scores of each classifier



```
Prediction Time for KNN:  20.471721500000058
Average Perceptron Score:  0.950088888888889
Minimum Perceptron Score:  0.9388888888888889
Maximum Perceptron Score:  0.9583333333333334
Best Perceptron Score:  0.9583333333333334  At Size:  4000
=====================================================================================================================
Average SVM Score:  0.5013444444444445
Minimum SVM Score:  0.4725
Maximum SVM Score:  0.5277777777777778
SVM Scores [0.5111111111111111, 0.5277777777777778, 0.5, 0.4725, 0.49533333333333335]
Best SVM Score:  0.5277777777777778  At Size:  600
=====================================================================================================================
Decision Tree Scores:  [-0.006256999999990853, -0.006245400000011614, -0.0067018999999959306, -0.008168000000011943, -0.012041400000043723]
=====================================================================================================================
KNN Scores [0.9666666666666667, 0.95, 0.9466666666666667, 0.945, 0.9573333333333334]

Process finished with exit code 0
```

Here I have included an example of what task 2 prints from each classifier at each sample size.

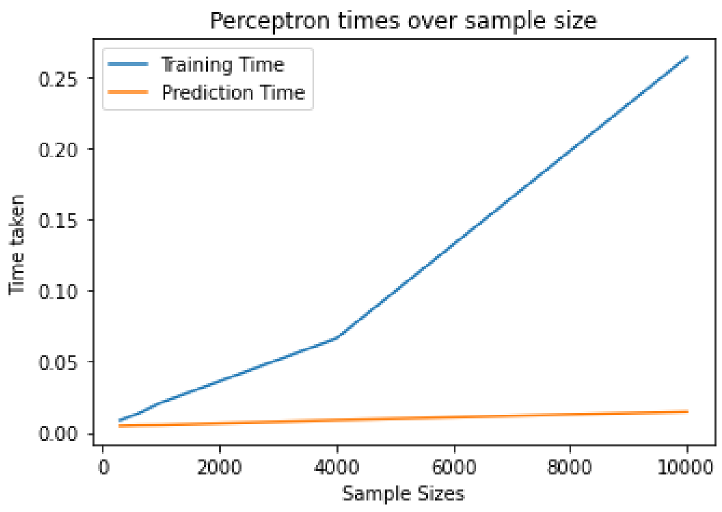


```
I is :  0
Time taken for perceptron training:  0.010507199999999273
Time taken for perceptron prediction:  0.007522300000000648
Best Score is : 0.5333333333333333  At Gamma:  10.0
best value:  0
Time Taken To find Best K:  1.6959045999999995
Best K:  1
 Best kNN score:  0.9555555555555556 At K:  1
Training Time:  0.018664100000000516
Prediction Time:  0.025596099999999566
```

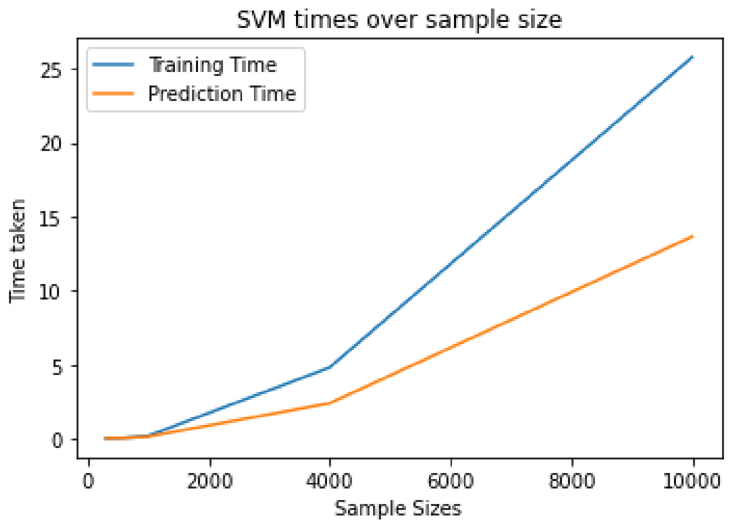
Task 7- Comparison:

The trends of each classifier we can extrapolate from the training and prediction times are:

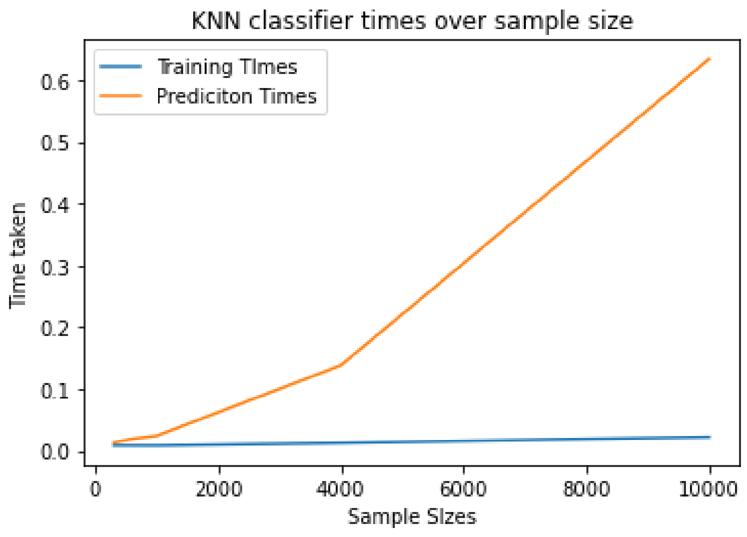
1. Perceptron:
   As the sample size increases the prediction time also increases at a noticeable level, while prediction time is only impacted at an arguably insignificant level.



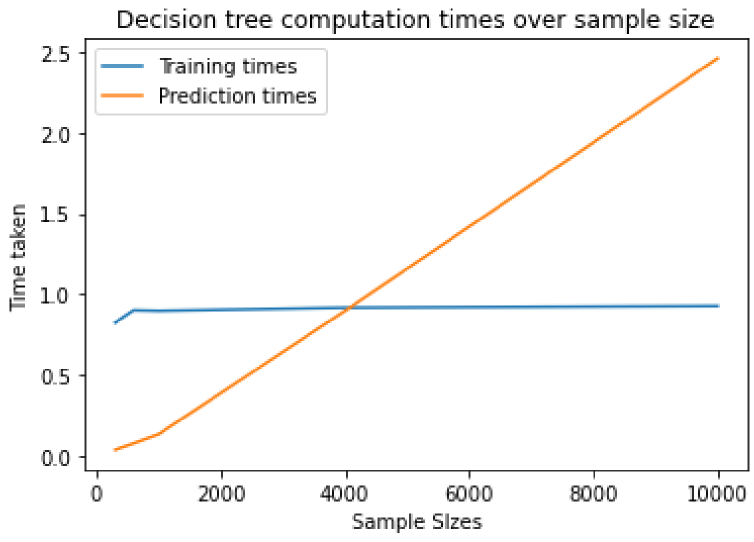
2. Support Vector Machine:
   As the sample size increases in the support vector machine the training time also increases similarily to the perceptron, the prediction time also increases. Also in my runnings the best gamma was consistently 1e1 or 10.



3. K-Nearest Neighbours:
   In the k-nearest Neighbours data we can see that the training time remains the same as the optimal classifier has already been found before this operation. Prediction time increases which we can logically find reasonable due to the fact that as the sample size increases the prediction time also increases.



4. Decision Tree's:
   I believe that there is an error in my code for the decision tree's however I will press on in my conclusion with them. In this case the training times for the decision tree's stayed almost stagnant while the prediction time saw a large increase in the amount of time taken to predict. I am not sure why this is happening but I believe it is due to the increased amount of sample sizes but that leads to the question why the training time is not also effected.



5. Conclusion:
   In my opinion for the sample sizes and the nature of the samples I would recommend the use of the perceptron in this case. As we can see from the charts the training and prediction time is much smaller than other classifiers and therefore would scale well with a growing dataset.