Assignment 2 (35%) - Supervised Learning

COMP8043 Machine Learning – Munster Technological University

Lochlann O Neill – BSc (Honours) in Software Development

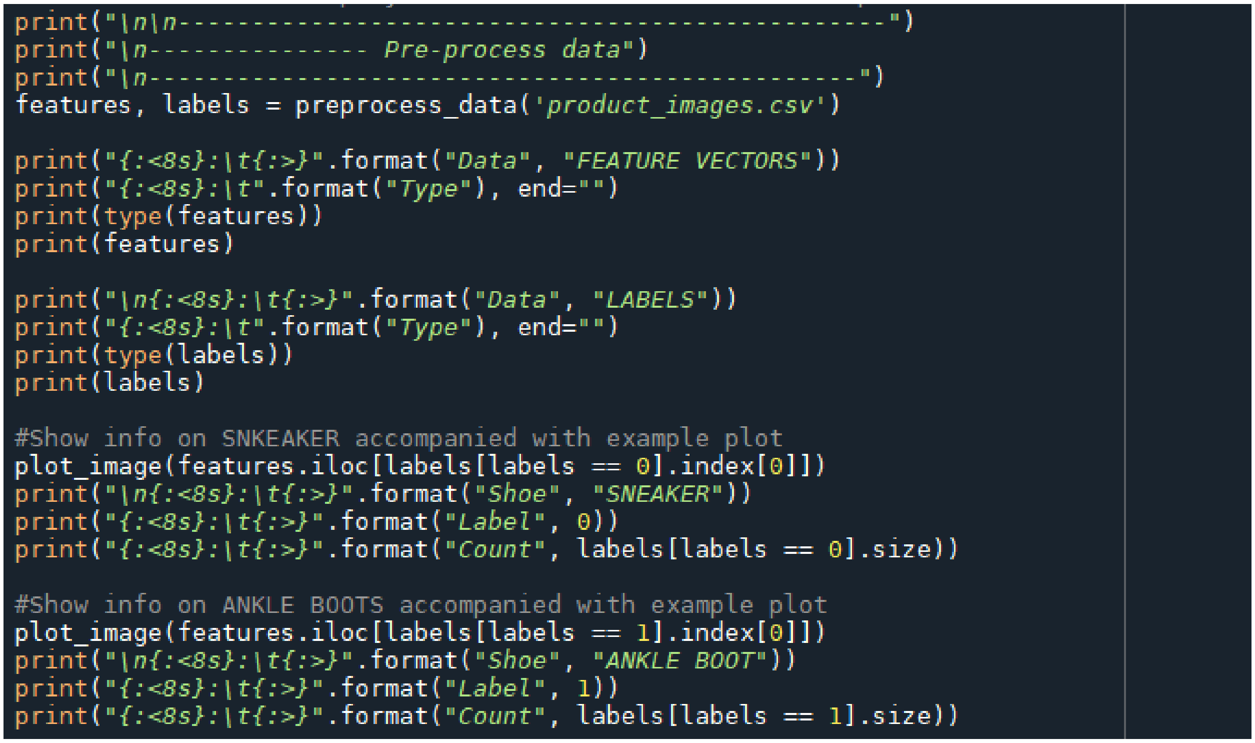
https://www.lochlannoneill.com

TASK 1 (pre-processing and visualisation, 5 points)

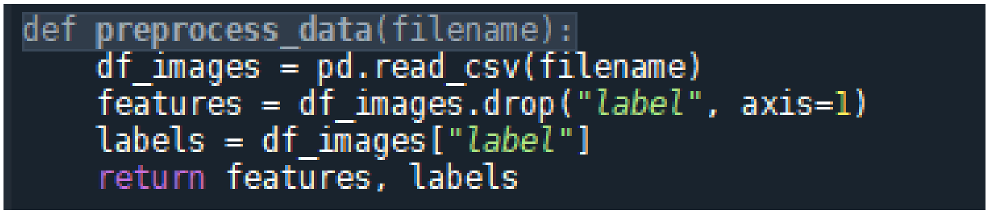
Load the product image dataset and separate the labels [1 point] from the feature vectors [1 point]. How many samples are images of sneakers, how many samples are images of ankle boots [1 point]? Display at least one image for each class [2 point].

Taking the .csv filename as an input parameter, the processed dataframes 'features' and 'labels' are returned.

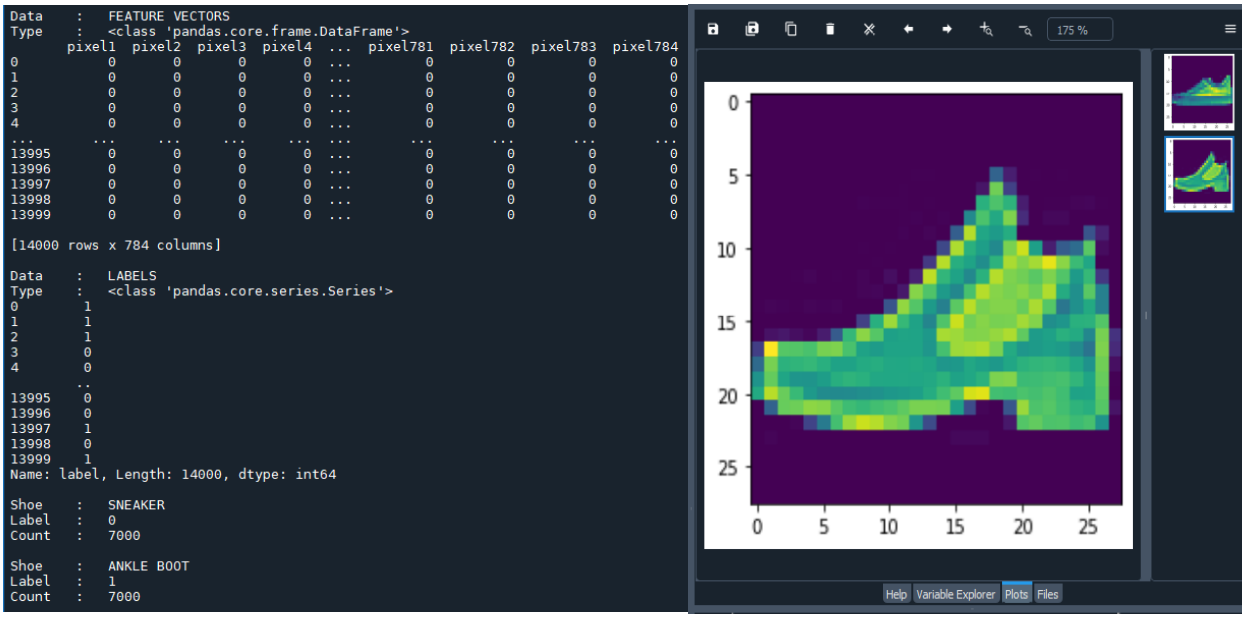
The first column entry denotes the 'label'; either 0 (sneaker) or 1 (ankle boot). There exists 7000 of each shoes type, totalling to 14000. The next 784 column entries denote the color value for the respective pixel. A 28x28 grid can be used to display these pixels.



```
print("\n\n-------------------------------------------------")
print("\n--------------- Pre-process data")
print("\n------------------------------------------------")
features, labels = preprocess_data('product_images.csv')

print("{:<8s}:\t{:>}".format("Data", "FEATURE VECTORS"))
print("{:<8s}:\t".format("Type"), end="")
print(type(features))
print(features)

print("\n{:<8s}:\t{:>}".format("Data", "LABELS"))
print("{:<8s}:\t".format("Type"), end="")
print(type(labels))
print(labels)

#Show info on SNKEAKER accompanied with example plot
plot_image(features.iloc[labels[labels == 0].index[0]])
print("\n{:<8s}:\t{:>}".format("Shoe", "SNEAKER"))
print("{:<8s}:\t{:>}".format("Label", 0))
print("{:<8s}:\t{:>}".format("Count", labels[labels == 0].size))

#Show info on ANKLE BOOTS accompanied with example plot
plot_image(features.iloc[labels[labels == 1].index[0]])
print("\n{:<8s}:\t{:>}".format("Shoe", "ANKLE BOOT"))
print("{:<8s}:\t{:>}".format("Label", 1))
print("{:<8s}:\t{:>}".format("Count", labels[labels == 1].size))
```



```
def preprocess_data(filename):
    df_images = pd.read_csv(filename)
    features = df_images.drop("label", axis=1)
    labels = df_images["label"]
    return features, labels
```



```
Data    :   FEATURE VECTORS
Type    :   <class 'pandas.core.frame.DataFrame'>
       pixel1  pixel2  pixel3  pixel4  ...  pixel781  pixel782  pixel783  pixel784
0           0       0       0       0  ...         0         0         0         0
1           0       0       0       0  ...         0         0         0         0
2           0       0       0       0  ...         0         0         0         0
3           0       0       0       0  ...         0         0         0         0
4           0       0       0       0  ...         0         0         0         0
...       ...     ...     ...     ...  ...       ...       ...       ...       ...
13995       0       0       0       0  ...         0         0         0         0
13996       0       0       0       0  ...         0         0         0         0
13997       0       0       0       0  ...         0         0         0         0
13998       0       0       0       0  ...         0         0         0         0
13999       0       0       0       0  ...         0         0         0         0

[14000 rows x 784 columns]

Data    :   LABELS
Type    :   <class 'pandas.core.series.Series'>
0        1
1        1
2        1
3        0
4        0
        ..
13995    0
13996    0
13997    1
13998    0
13999    1
Name: label, Length: 14000, dtype: int64

Shoe    :   SNEAKER
Label   :   0
Count   :   7000

Shoe    :   ANKLE BOOT
Label   :   1
Count   :   7000
```

TASK 2 (evaluation procedure, 9 points)

Create a k-fold cross-validation procedure to split the data into training [1 point] and evaluation subsets [1 point]. Parameterize the number of samples to use from the dataset to be able to control the runtime of the algorithm evaluation [1 point]. Start developing using a small number of samples and increase for the final evaluation.

Make the function flexible to accommodate different types of classifiers as required in tasks 3-6. Measure for each split of the cross-validation procedure the processing time required for training [1 point], the processing time required for prediction [1 point] and determine the confusion matrix [1 point] and accuracy score of the classification.

Calculate the minimum, the maximum, and the average of

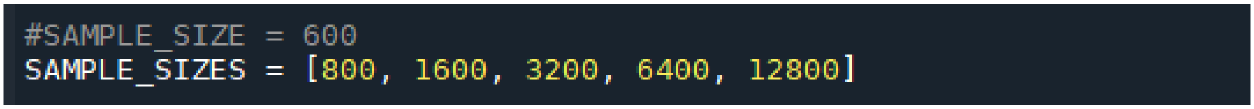
- ***the training time per training sample [1 point]***
- ***the prediction time per evaluation sample [1 point]***
- ***and the prediction accuracy [1 point].***

The evaluate_classifier() function act as a sort of framework for the other tasks. With no repetition of code, every other task may be run through it, where only the new respective classifier parameter is required for each task. A dictionary is returned as the summary of the given classifier, containing all of the important values for plotting.

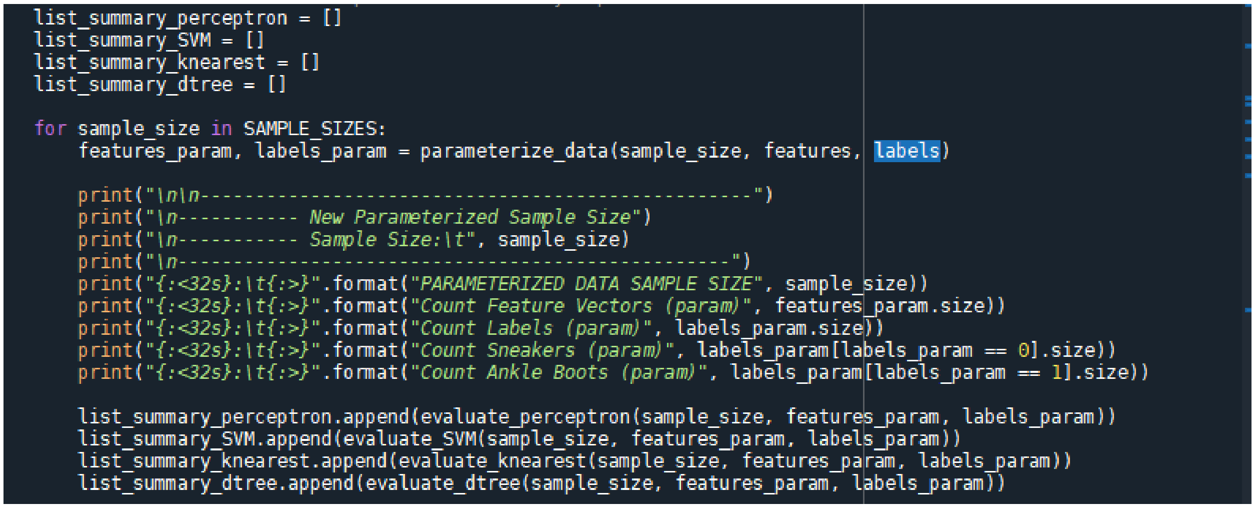
Using the full size of the original dataframe (14000) may result in slow processing speeds due to hardware limitations. Thus, a parameterized sample size is preferred. Utilizing an easily changeable constant variable located at the top of the source code, one can quickly alter the sample size. With a sample size of 800, the total processing time is not drastically decreased, while still providing a sizable sample size for evaluation. However, I will use an array of exponentially increasing sample sizes, for classifier evaluation comparison between the different processing times.

At each kfold, the training time, testing time, prediction accuracy and confusion matrix are displayed to the terminal in a formatted output.

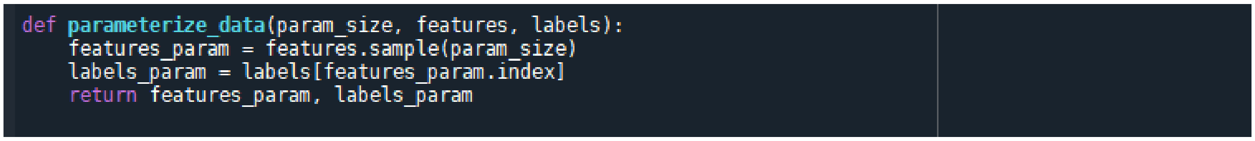
The information summary dictionary is outputted to the terminal. This dictionary is also returned non-formatted.



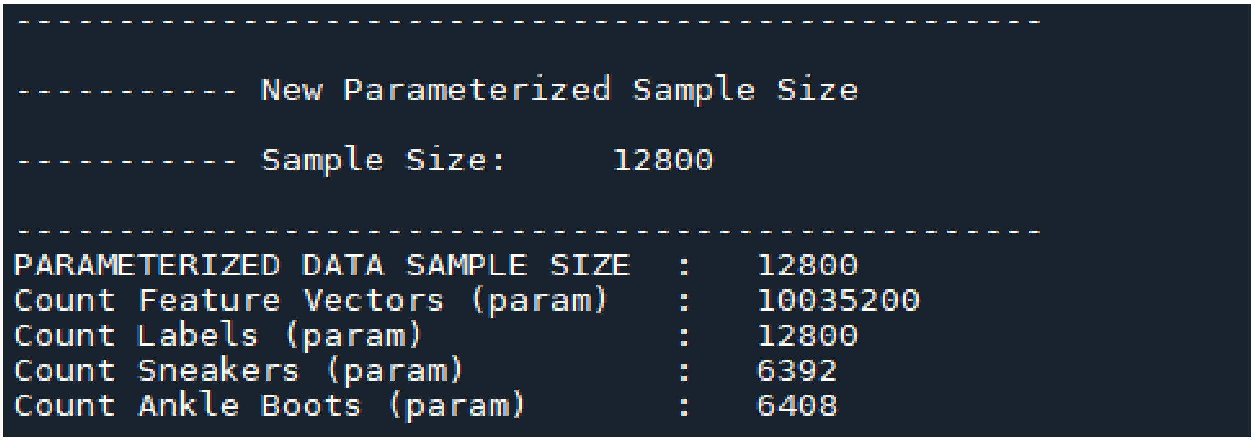
```
#SAMPLE_SIZE = 600
SAMPLE_SIZES = [800, 1600, 3200, 6400, 12800]
```



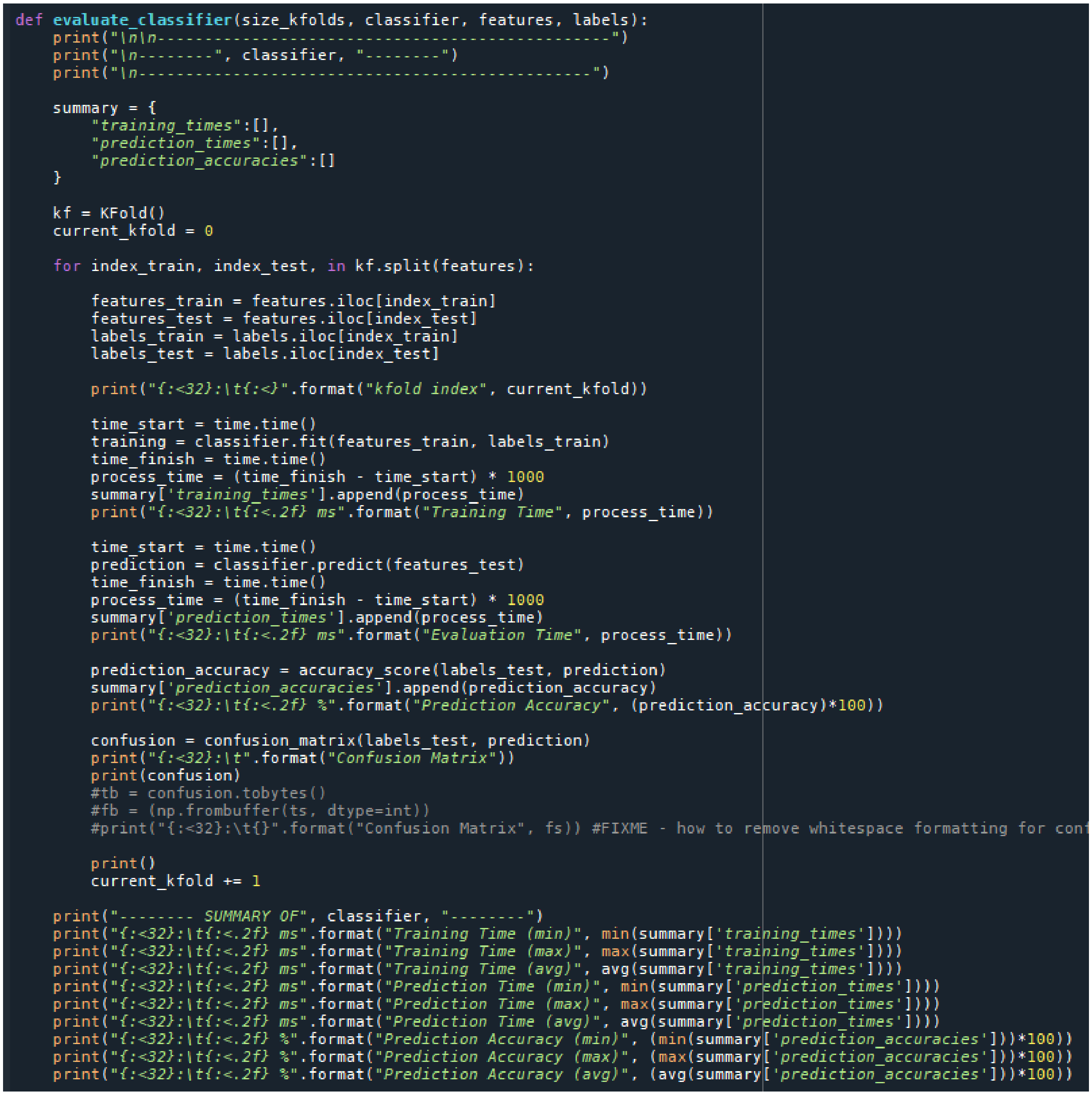
```
list_summary_perceptron = []
list_summary_SVM = []
list_summary_knearest = []
list_summary_dtree = []

for sample_size in SAMPLE_SIZES:
    features_param, labels_param = parameterize_data(sample_size, features, labels)

    print("\n\n--------------------------------------------------")
    print("\n----------- New Parameterized Sample Size")
    print("\n----------- Sample Size:\t", sample_size)
    print("\n--------------------------------------------------")
    print("{:<32s}:\t{:>}".format("PARAMETERIZED DATA SAMPLE SIZE", sample_size))
    print("{:<32s}:\t{:>}".format("Count Feature Vectors (param)", features_param.size))
    print("{:<32s}:\t{:>}".format("Count Labels (param)", labels_param.size))
    print("{:<32s}:\t{:>}".format("Count Sneakers (param)", labels_param[labels_param == 0].size))
    print("{:<32s}:\t{:>}".format("Count Ankle Boots (param)", labels_param[labels_param == 1].size))

    list_summary_perceptron.append(evaluate_perceptron(sample_size, features_param, labels_param))
    list_summary_SVM.append(evaluate_SVM(sample_size, features_param, labels_param))
    list_summary_knearest.append(evaluate_knearest(sample_size, features_param, labels_param))
    list_summary_dtree.append(evaluate_dtree(sample_size, features_param, labels_param))
```



```
def parameterize_data(param_size, features, labels):
    features_param = features.sample(param_size)
    labels_param = labels[features_param.index]
    return features_param, labels_param
```



```
--------------------------------------------------

----------- New Parameterized Sample Size

----------- Sample Size:     12800

--------------------------------------------------
PARAMETERIZED DATA SAMPLE SIZE  :    12800
Count Feature Vectors (param)   :    10035200
Count Labels (param)            :    12800
Count Sneakers (param)          :    6392
Count Ankle Boots (param)       :    6408
```

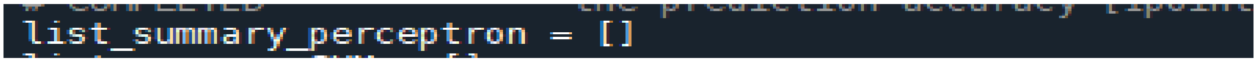


```
def evaluate_classifier(size_kfolds, classifier, features, labels):
    print("\n\n------------------------------------------------")
    print("\n--------", classifier, "--------")
    print("\n------------------------------------------------")

    summary = {
        "training_times":[],
        "prediction_times":[],
        "prediction_accuracies":[]
    }

    kf = KFold()
    current_kfold = 0

    for index_train, index_test, in kf.split(features):

        features_train = features.iloc[index_train]
        features_test = features.iloc[index_test]
        labels_train = labels.iloc[index_train]
        labels_test = labels.iloc[index_test]

        print("{:<32}:\t{:<}".format("kfold index", current_kfold))

        time_start = time.time()
        training = classifier.fit(features_train, labels_train)
        time_finish = time.time()
        process_time = (time_finish - time_start) * 1000
        summary['training_times'].append(process_time)
        print("{:<32}:\t{:<.2f} ms".format("Training Time", process_time))

        time_start = time.time()
        prediction = classifier.predict(features_test)
        time_finish = time.time()
        process_time = (time_finish - time_start) * 1000
        summary['prediction_times'].append(process_time)
        print("{:<32}:\t{:<.2f} ms".format("Evaluation Time", process_time))

        prediction_accuracy = accuracy_score(labels_test, prediction)
        summary['prediction_accuracies'].append(prediction_accuracy)
        print("{:<32}:\t{:<.2f} %".format("Prediction Accuracy", (prediction_accuracy)*100))

        confusion = confusion_matrix(labels_test, prediction)
        print("{:<32}:\t".format("Confusion Matrix"))
        print(confusion)
        #tb = confusion.tobytes()
        #fb = (np.frombuffer(ts, dtype=int))
        #print("{:<32}:\t{}".format("Confusion Matrix", fs)) #FIXME - how to remove whitespace formatting for conf

        print()
        current_kfold += 1

    print("-------- SUMMARY OF", classifier, "--------")
    print("{:<32}:\t{:<.2f} ms".format("Training Time (min)", min(summary['training_times'])))
    print("{:<32}:\t{:<.2f} ms".format("Training Time (max)", max(summary['training_times'])))
    print("{:<32}:\t{:<.2f} ms".format("Training Time (avg)", avg(summary['training_times'])))
    print("{:<32}:\t{:<.2f} ms".format("Prediction Time (min)", min(summary['prediction_times'])))
    print("{:<32}:\t{:<.2f} ms".format("Prediction Time (max)", max(summary['prediction_times'])))
    print("{:<32}:\t{:<.2f} ms".format("Prediction Time (avg)", avg(summary['prediction_times'])))
    print("{:<32}:\t{:<.2f} %".format("Prediction Accuracy (min)", (min(summary['prediction_accuracies']))*100))
    print("{:<32}:\t{:<.2f} %".format("Prediction Accuracy (max)", (max(summary['prediction_accuracies']))*100))
    print("{:<32}:\t{:<.2f} %".format("Prediction Accuracy (avg)", (avg(summary['prediction_accuracies']))*100))
```

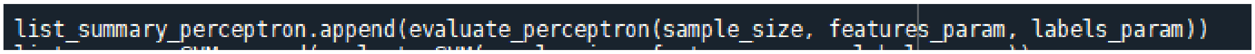
TASK 3 (Perceptron, 3 points)

Use the procedure developed in task 2 to train and evaluate the Perceptron classifier [1 point]. What is the mean prediction accuracy of this classifier [1 point]? Vary the number of samples and plot the relationship between input data size and runtimes for the classifier [1 point].

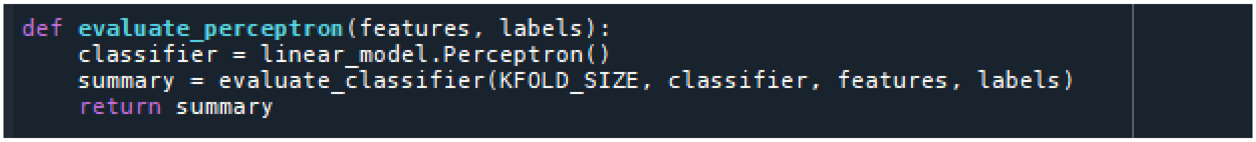
The evaluate_perceptron() function is passed into the evaluate_classifier() function, providing its respective classifier as a parameter. Different sample sizes are used for process time comparison. The summary dictionary received from the evaluate_classifier() is further returned to the Main, appended to the list containing similar classifier summaries.



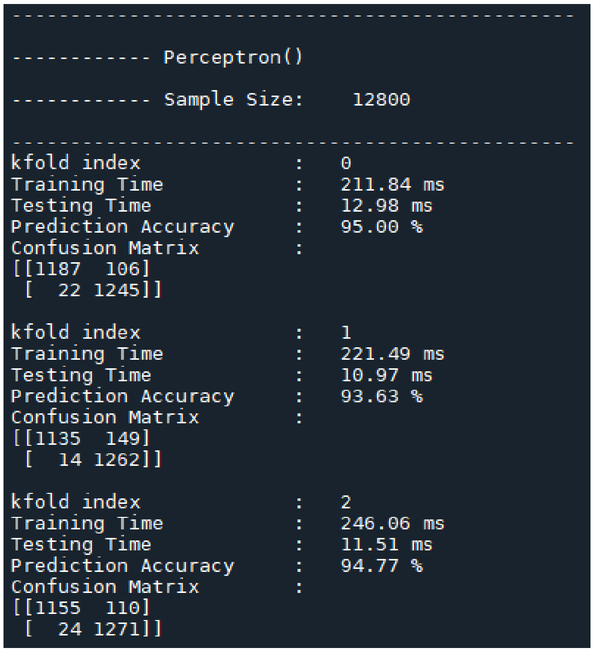
```
list_summary_perceptron = []
```



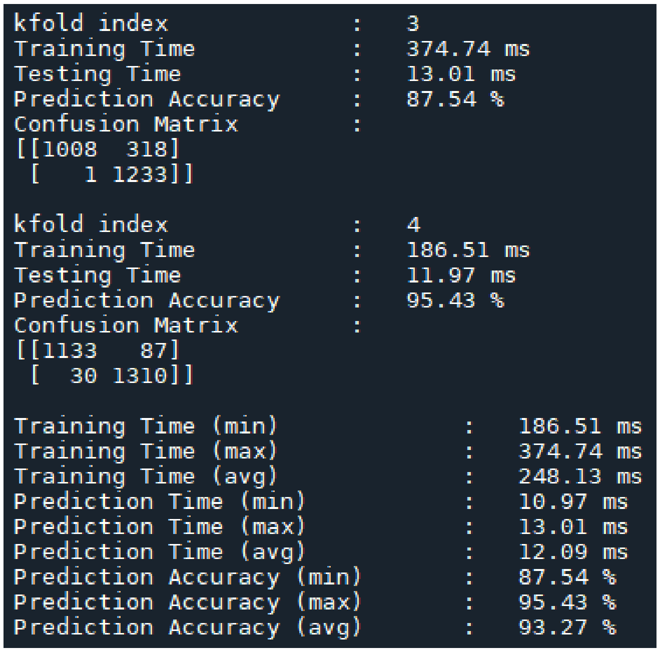
```
list_summary_perceptron.append(evaluate_perceptron(sample_size, features_param, labels_param))
```



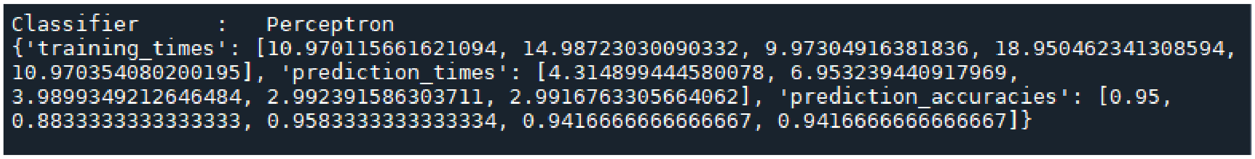
```
def evaluate_perceptron(features, labels):
    classifier = linear_model.Perceptron()
    summary = evaluate_classifier(KFOLD_SIZE, classifier, features, labels)
    return summary
```



```
-----------------------------------------------

------------ Perceptron()

------------ Sample Size:    12800

-----------------------------------------------
kfold index              :   0
Training Time            :   211.84 ms
Testing Time             :   12.98 ms
Prediction Accuracy      :   95.00 %
Confusion Matrix         :
[[1187  106]
 [  22 1245]]

kfold index              :   1
Training Time            :   221.49 ms
Testing Time             :   10.97 ms
Prediction Accuracy      :   93.63 %
Confusion Matrix         :
[[1135  149]
 [  14 1262]]

kfold index              :   2
Training Time            :   246.06 ms
Testing Time             :   11.51 ms
Prediction Accuracy      :   94.77 %
Confusion Matrix         :
[[1155  110]
 [  24 1271]]
```



```
kfold index              :   3
Training Time            :   374.74 ms
Testing Time             :   13.01 ms
Prediction Accuracy      :   87.54 %
Confusion Matrix         :
[[1008  318]
 [   1 1233]]

kfold index              :   4
Training Time            :   186.51 ms
Testing Time             :   11.97 ms
Prediction Accuracy      :   95.43 %
Confusion Matrix         :
[[1133   87]
 [  30 1310]]

Training Time (min)             :   186.51 ms
Training Time (max)             :   374.74 ms
Training Time (avg)             :   248.13 ms
Prediction Time (min)           :   10.97 ms
Prediction Time (max)           :   13.01 ms
Prediction Time (avg)           :   12.09 ms
Prediction Accuracy (min)       :   87.54 %
Prediction Accuracy (max)       :   95.43 %
Prediction Accuracy (avg)       :   93.27 %
```



```
Classifier      :   Perceptron
{'training_times': [10.970115661621094, 14.98723030090332, 9.97304916381836, 18.950462341308594,
10.970354080200195], 'prediction_times': [4.314899444580078, 6.953239440917969,
3.9899349212646484, 2.992391586303711, 2.9916763305664062], 'prediction_accuracies': [0.95,
0.8833333333333333, 0.9583333333333334, 0.9416666666666667, 0.9416666666666667]}
```

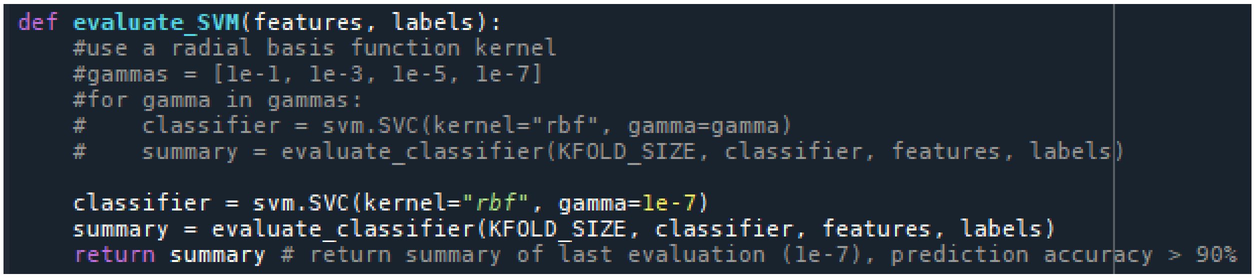
TASK 4 (Support Vector Machine, 5 points)

Use the procedure developed in task 2 to train and evaluate the Support Vector Machine classifier [1 point]. Use a radial basis function kernel and try different choices for the parameter γ [1 point]. Determine a good value for γ based on mean prediction accuracy [1 point]. What is the best achievable mean prediction accuracy of this classifier [1 point]? Vary the number of samples and plot the relationship between input data size and runtimes for the optimal classifier [1 point].

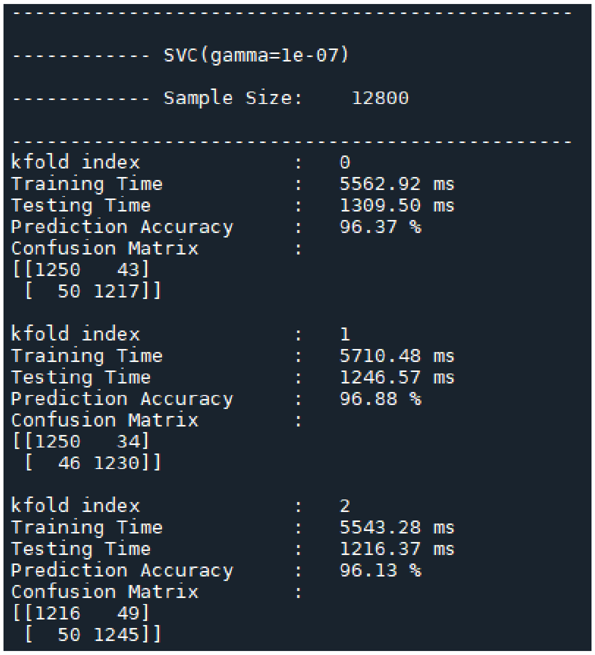
As seen in the commented-out code, a gamma list was utilized to find the highest prediction accuracy at an iterative gamma value. This valued was found to be '1e-7', thus a single classifier replaced the loop.

```
list_summary_SVM = []
```

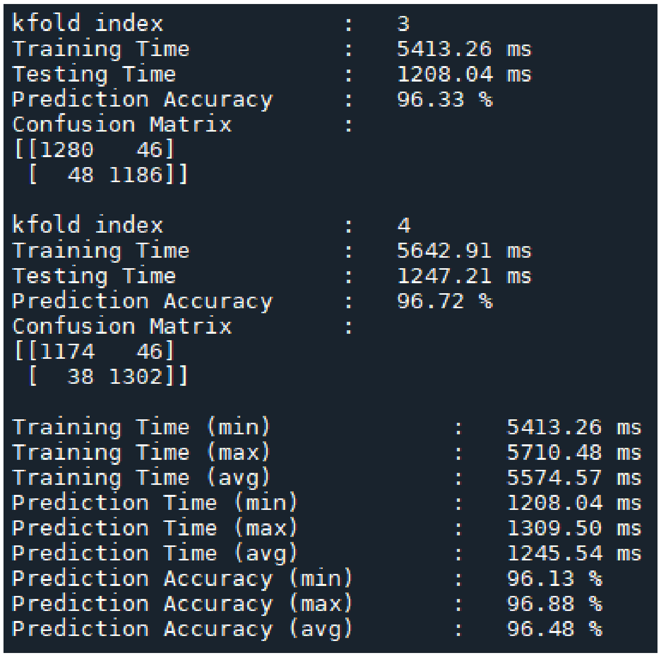
```
list_summary_SVM.append(evaluate_SVM(sample_size, features_param, labels_param))
```



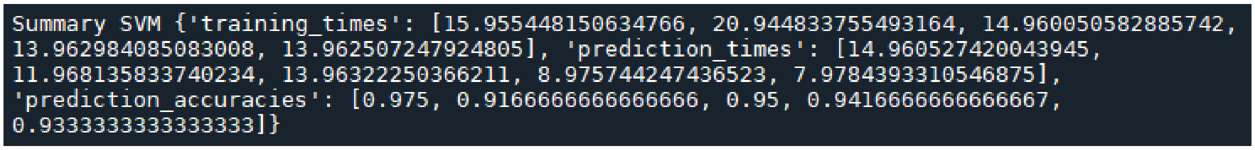
```
def evaluate_SVM(features, labels):
    #use a radial basis function kernel
    #gammas = [1e-1, 1e-3, 1e-5, 1e-7]
    #for gamma in gammas:
    #    classifier = svm.SVC(kernel="rbf", gamma=gamma)
    #    summary = evaluate_classifier(KFOLD_SIZE, classifier, features, labels)

    classifier = svm.SVC(kernel="rbf", gamma=1e-7)
    summary = evaluate_classifier(KFOLD_SIZE, classifier, features, labels)
    return summary # return summary of last evaluation (1e-7), prediction accuracy > 90%
```



```
-------------------------------------------------

------------ SVC(gamma=1e-07)

------------ Sample Size:    12800

-------------------------------------------------
kfold index              :   0
Training Time            :   5562.92 ms
Testing Time             :   1309.50 ms
Prediction Accuracy      :   96.37 %
Confusion Matrix         :
[[1250   43]
 [  50 1217]]

kfold index              :   1
Training Time            :   5710.48 ms
Testing Time             :   1246.57 ms
Prediction Accuracy      :   96.88 %
Confusion Matrix         :
[[1250   34]
 [  46 1230]]

kfold index              :   2
Training Time            :   5543.28 ms
Testing Time             :   1216.37 ms
Prediction Accuracy      :   96.13 %
Confusion Matrix         :
[[1216   49]
 [  50 1245]]
```



```
kfold index              :   3
Training Time            :   5413.26 ms
Testing Time             :   1208.04 ms
Prediction Accuracy      :   96.33 %
Confusion Matrix         :
[[1280   46]
 [  48 1186]]

kfold index              :   4
Training Time            :   5642.91 ms
Testing Time             :   1247.21 ms
Prediction Accuracy      :   96.72 %
Confusion Matrix         :
[[1174   46]
 [  38 1302]]

Training Time (min)              :   5413.26 ms
Training Time (max)              :   5710.48 ms
Training Time (avg)              :   5574.57 ms
Prediction Time (min)            :   1208.04 ms
Prediction Time (max)            :   1309.50 ms
Prediction Time (avg)            :   1245.54 ms
Prediction Accuracy (min)        :   96.13 %
Prediction Accuracy (max)        :   96.88 %
Prediction Accuracy (avg)        :   96.48 %
```

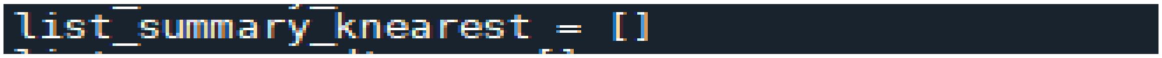


```
Summary SVM {'training_times': [15.955448150634766, 20.944833755493164, 14.960050582885742,
13.962984085083008, 13.962507247924805], 'prediction_times': [14.960527420043945,
11.968135833740234, 13.96322250366211, 8.975744247436523, 7.9784393310546875],
'prediction_accuracies': [0.975, 0.9166666666666666, 0.95, 0.9416666666666667,
0.9333333333333333]}
```

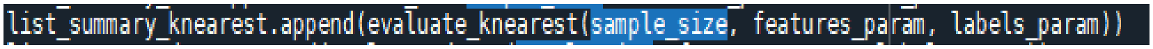
TASK 5 (k-nearest Neighbours, 5 points)

Use the procedure developed in task 2 to train and evaluate the k-nearest neighbour classifier [1 point]. Try different choices for the parameter k [1 point] and determine a good value based on mean prediction accuracy [1 point]. What is the best achievable mean prediction accuracy of this classifier [1 point]? Vary the number of samples and plot the relationship between input data size and runtimes for the optimal classifier [1 point].

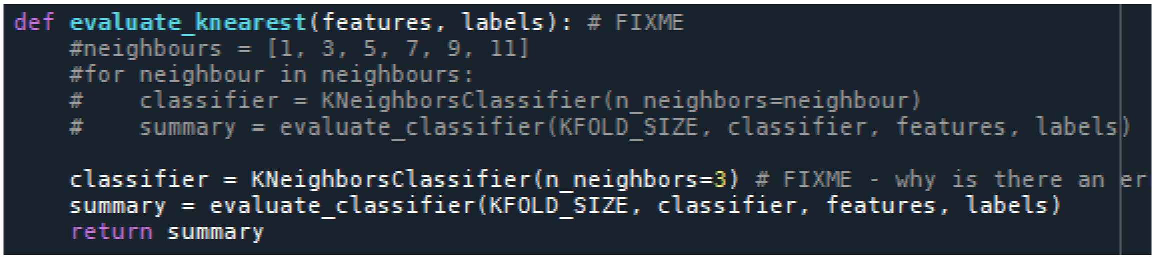
As seen in the commented-out code, a neightbours list was utilized to find the value for the k-nearest neighbour with the highest prediction accuracy; discovered to be 3.



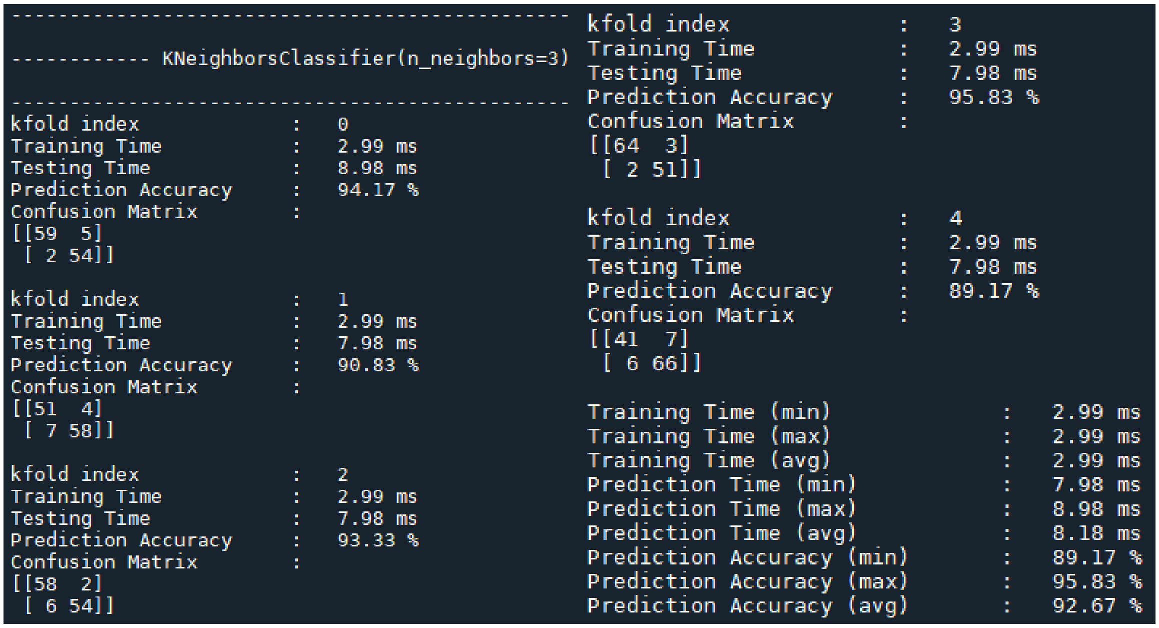
```
list_summary_knearest = []
```



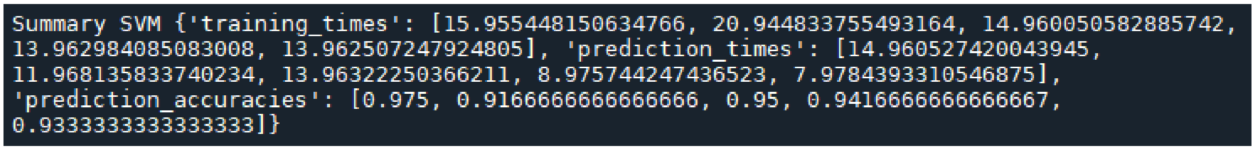
```
list_summary_knearest.append(evaluate_knearest(sample_size, features_param, labels_param))
```



```
def evaluate_knearest(features, labels): # FIXME
    #neighbours = [1, 3, 5, 7, 9, 11]
    #for neighbour in neighbours:
    #    classifier = KNeighborsClassifier(n_neighbors=neighbour)
    #    summary = evaluate_classifier(KFOLD_SIZE, classifier, features, labels)

    classifier = KNeighborsClassifier(n_neighbors=3) # FIXME - why is there an er
    summary = evaluate_classifier(KFOLD_SIZE, classifier, features, labels)
    return summary
```



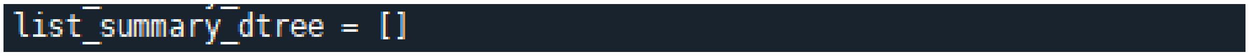
```
-----------------------------------------------

------------ KNeighborsClassifier(n_neighbors=3)

-----------------------------------------------
kfold index               :   0
Training Time             :   2.99 ms
Testing Time              :   8.98 ms
Prediction Accuracy       :   94.17 %
Confusion Matrix          :
[[59  5]
 [ 2 54]]

kfold index               :   1
Training Time             :   2.99 ms
Testing Time              :   7.98 ms
Prediction Accuracy       :   90.83 %
Confusion Matrix          :
[[51  4]
 [ 7 58]]

kfold index               :   2
Training Time             :   2.99 ms
Testing Time              :   7.98 ms
Prediction Accuracy       :   93.33 %
Confusion Matrix          :
[[58  2]
 [ 6 54]]
kfold index               :   3
Training Time             :   2.99 ms
Testing Time              :   7.98 ms
Prediction Accuracy       :   95.83 %
Confusion Matrix          :
[[64  3]
 [ 2 51]]

kfold index               :   4
Training Time             :   2.99 ms
Testing Time              :   7.98 ms
Prediction Accuracy       :   89.17 %
Confusion Matrix          :
[[41  7]
 [ 6 66]]

Training Time (min)                :   2.99 ms
Training Time (max)                :   2.99 ms
Training Time (avg)                :   2.99 ms
Prediction Time (min)              :   7.98 ms
Prediction Time (max)              :   8.98 ms
Prediction Time (avg)              :   8.18 ms
Prediction Accuracy (min)          :   89.17 %
Prediction Accuracy (max)          :   95.83 %
Prediction Accuracy (avg)          :   92.67 %
```



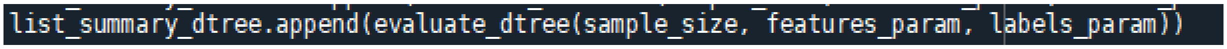
```
Summary SVM {'training_times': [15.955448150634766, 20.944833755493164, 14.960050582885742,
13.962984085083008, 13.962507247924805], 'prediction_times': [14.960527420043945,
11.968135833740234, 13.96322250366211, 8.975744247436523, 7.9784393310546875],
'prediction_accuracies': [0.975, 0.9166666666666666, 0.95, 0.9416666666666667,
0.9333333333333333]}
```

TASK 6 (Decision trees, 3 points)

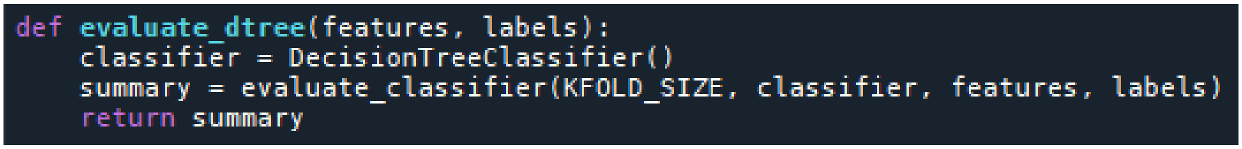
Use the procedure developed in task 2 to train and evaluate the Decision tree classifier [1 point]. What is the mean prediction accuracy of this classifier [1 point]? Vary the number of samples and plot the relationship between input data size and runtimes for the classifier [1 point].



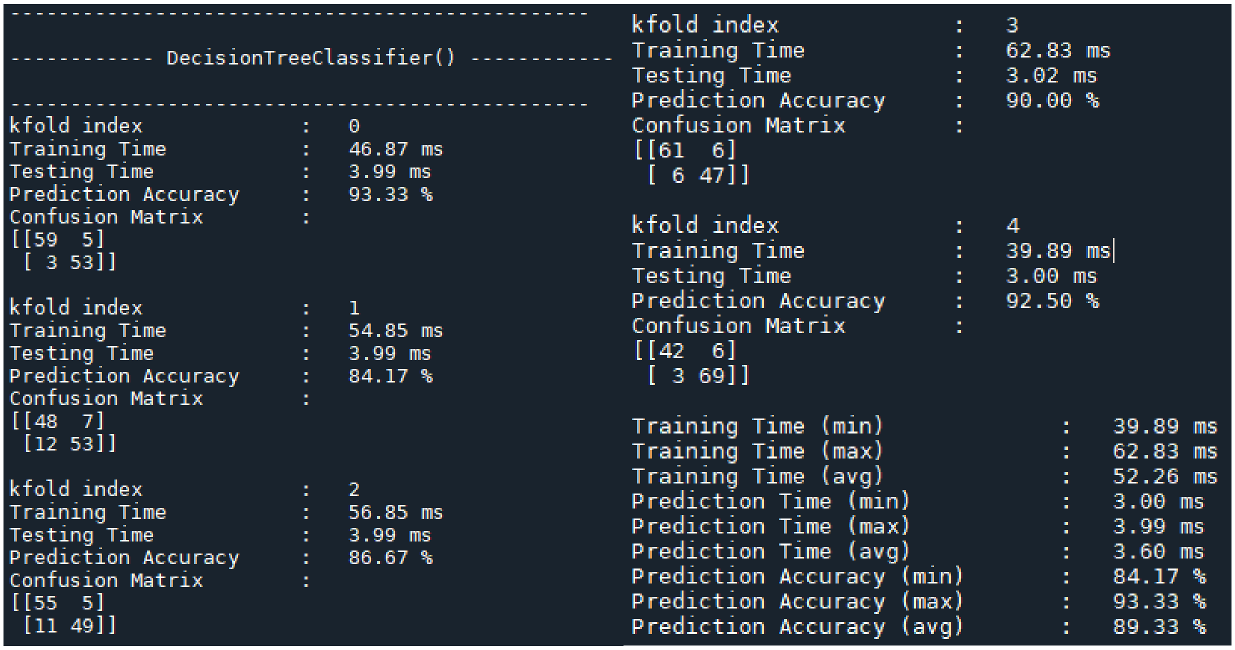
```
list_summary_dtree = []
```



```
list_summary_dtree.append(evaluate_dtree(sample_size, features_param, labels_param))
```

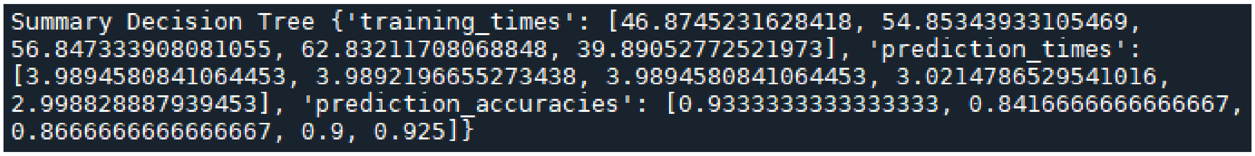


```
def evaluate_dtree(features, labels):
    classifier = DecisionTreeClassifier()
    summary = evaluate_classifier(KFOLD_SIZE, classifier, features, labels)
    return summary
```



```
-----------------------------------------------

------------ DecisionTreeClassifier() ------------

-----------------------------------------------
kfold index              :   0
Training Time            :   46.87 ms
Testing Time             :   3.99 ms
Prediction Accuracy      :   93.33 %
Confusion Matrix         :
[[59  5]
 [ 3 53]]

kfold index              :   1
Training Time            :   54.85 ms
Testing Time             :   3.99 ms
Prediction Accuracy      :   84.17 %
Confusion Matrix         :
[[48  7]
 [12 53]]

kfold index              :   2
Training Time            :   56.85 ms
Testing Time             :   3.99 ms
Prediction Accuracy      :   86.67 %
Confusion Matrix         :
[[55  5]
 [11 49]]
```

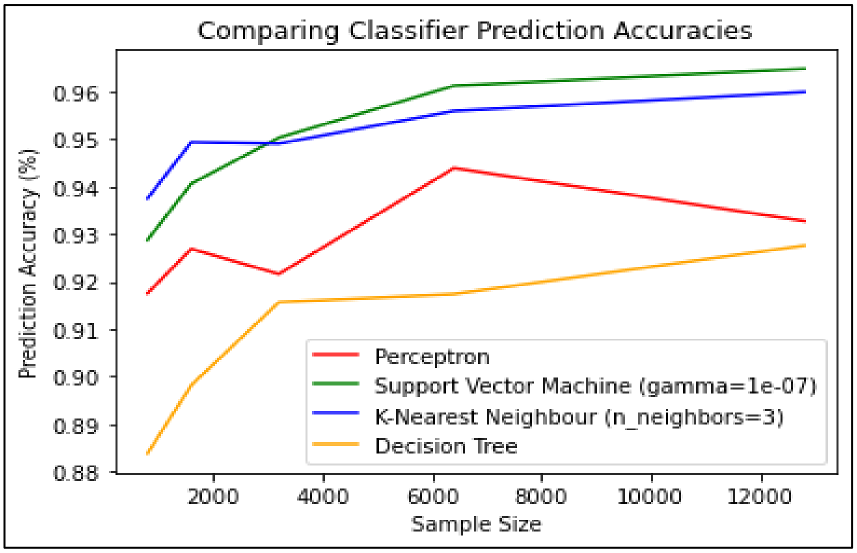
```
kfold index              :   3
Training Time            :   62.83 ms
Testing Time             :   3.02 ms
Prediction Accuracy      :   90.00 %
Confusion Matrix         :
[[61  6]
 [ 6 47]]

kfold index              :   4
Training Time            :   39.89 ms
Testing Time             :   3.00 ms
Prediction Accuracy      :   92.50 %
Confusion Matrix         :
[[42  6]
 [ 3 69]]

Training Time (min)              :   39.89 ms
Training Time (max)              :   62.83 ms
Training Time (avg)              :   52.26 ms
Prediction Time (min)            :   3.00 ms
Prediction Time (max)            :   3.99 ms
Prediction Time (avg)            :   3.60 ms
Prediction Accuracy (min)        :   84.17 %
Prediction Accuracy (max)        :   93.33 %
Prediction Accuracy (avg)        :   89.33 %
```



```
Summary Decision Tree {'training_times': [46.8745231628418, 54.85343933105469,
56.847333908081055, 62.83211708068848, 39.89052772521973], 'prediction_times':
[3.9894580841064453, 3.9892196655273438, 3.9894580841064453, 3.0214786529541016,
2.998828887939453], 'prediction_accuracies': [0.9333333333333333, 0.8416666666666667,
0.8666666666666667, 0.9, 0.925]}
```

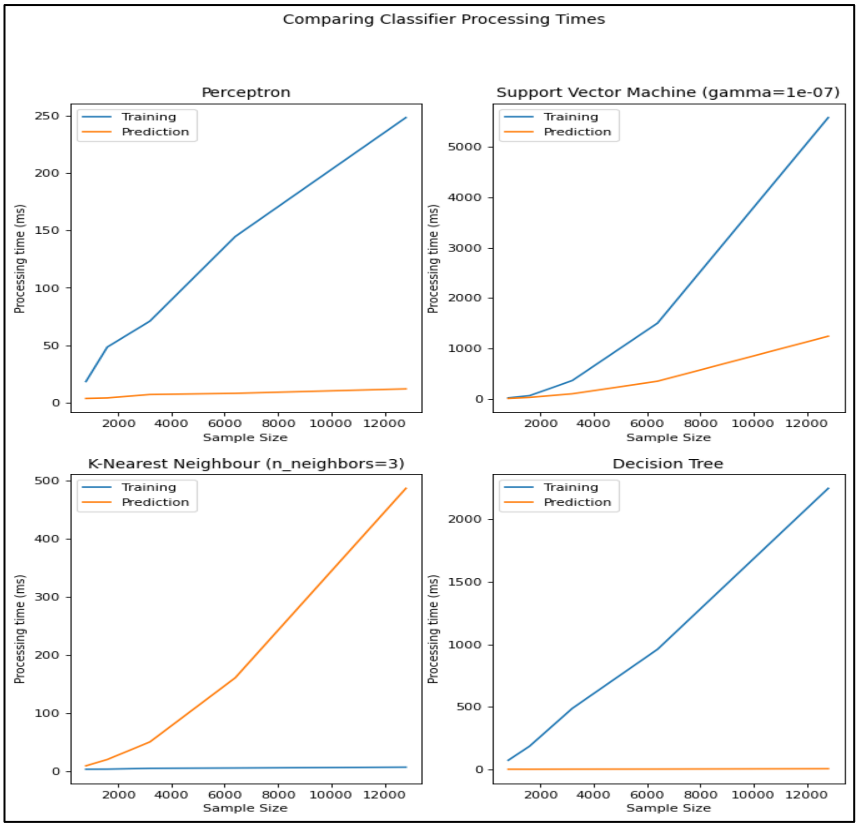
TASK 7 (comparison, 5 points)

Compare the training and prediction times of the four classifiers. What trend do you observe for each of the classifiers and why [4 points]? Also taking the accuracy into consideration, how would you rank the four classifiers and why [1 point].



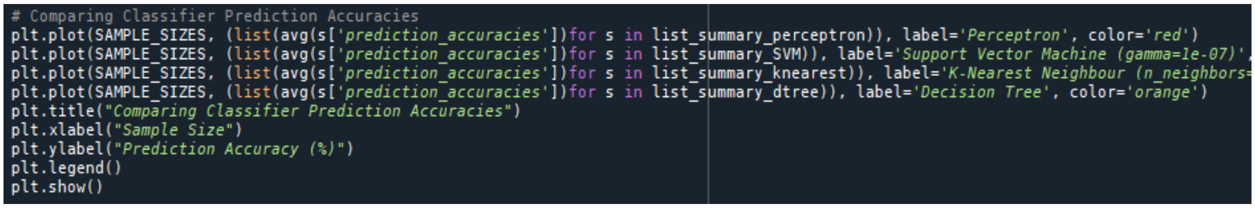
The prediction accuracies for each classifier are relatively consistent, except for the decision tree when given a low sample size.

Using this information, it is observed that a greater prediction accuracy is achieved alongside a larger sample size. With a higher sample size, the lower the effect of an outlier on the sample size prediction.

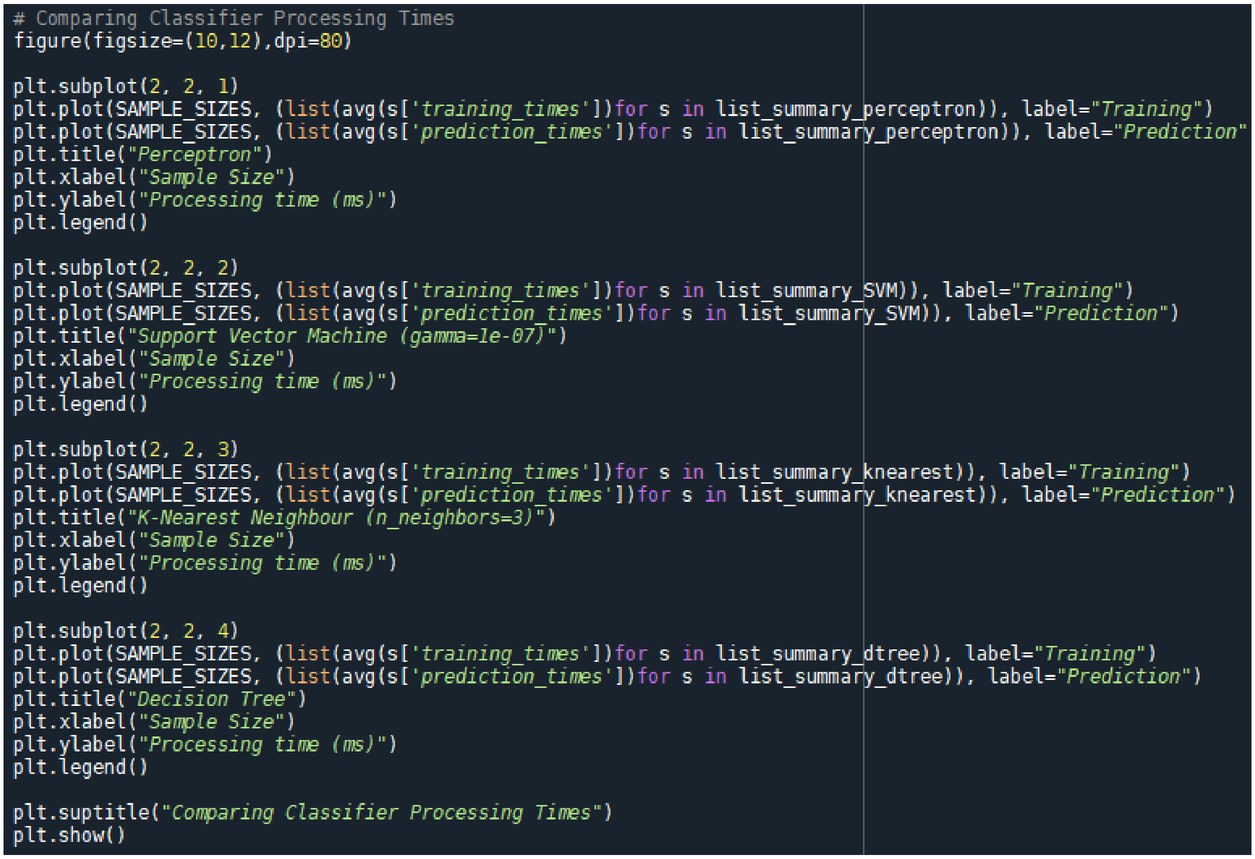


Regarding the k-NN algorithm, the classifier processing times for training and prediction are surprisingly opposite to what one might expect. However, this is an example of lazy learning, where the distances to each neighbour must be calculated upon prediction. At training time, it doesn't need to do very expensive distance calculation.

Perceptron must be the best fit in this circumstance, considering its high prediction accuracy accompanied by low prediction times.



```
# Comparing Classifier Prediction Accuracies
plt.plot(SAMPLE_SIZES, (list(avg(s['prediction_accuracies'])for s in list_summary_perceptron)), label='Perceptron', color='red')
plt.plot(SAMPLE_SIZES, (list(avg(s['prediction_accuracies'])for s in list_summary_SVM)), label='Support Vector Machine (gamma=1e-07)',
plt.plot(SAMPLE_SIZES, (list(avg(s['prediction_accuracies'])for s in list_summary_knearest)), label='K-Nearest Neighbour (n_neighbors=
plt.plot(SAMPLE_SIZES, (list(avg(s['prediction_accuracies'])for s in list_summary_dtree)), label='Decision Tree', color='orange')
plt.title("Comparing Classifier Prediction Accuracies")
plt.xlabel("Sample Size")
plt.ylabel("Prediction Accuracy (%)")
plt.legend()
plt.show()
```



```
# Comparing Classifier Processing Times
figure(figsize=(10,12),dpi=80)

plt.subplot(2, 2, 1)
plt.plot(SAMPLE_SIZES, (list(avg(s['training_times'])for s in list_summary_perceptron)), label="Training")
plt.plot(SAMPLE_SIZES, (list(avg(s['prediction_times'])for s in list_summary_perceptron)), label="Prediction"
plt.title("Perceptron")
plt.xlabel("Sample Size")
plt.ylabel("Processing time (ms)")
plt.legend()

plt.subplot(2, 2, 2)
plt.plot(SAMPLE_SIZES, (list(avg(s['training_times'])for s in list_summary_SVM)), label="Training")
plt.plot(SAMPLE_SIZES, (list(avg(s['prediction_times'])for s in list_summary_SVM)), label="Prediction")
plt.title("Support Vector Machine (gamma=1e-07)")
plt.xlabel("Sample Size")
plt.ylabel("Processing time (ms)")
plt.legend()

plt.subplot(2, 2, 3)
plt.plot(SAMPLE_SIZES, (list(avg(s['training_times'])for s in list_summary_knearest)), label="Training")
plt.plot(SAMPLE_SIZES, (list(avg(s['prediction_times'])for s in list_summary_knearest)), label="Prediction")
plt.title("K-Nearest Neighbour (n_neighbors=3)")
plt.xlabel("Sample Size")
plt.ylabel("Processing time (ms)")
plt.legend()

plt.subplot(2, 2, 4)
plt.plot(SAMPLE_SIZES, (list(avg(s['training_times'])for s in list_summary_dtree)), label="Training")
plt.plot(SAMPLE_SIZES, (list(avg(s['prediction_times'])for s in list_summary_dtree)), label="Prediction")
plt.title("Decision Tree")
plt.xlabel("Sample Size")
plt.ylabel("Processing time (ms)")
plt.legend()

plt.suptitle("Comparing Classifier Processing Times")
plt.show()
```