Machine Learning
Assignment 2: Supervised Learning

R00159222 - Zachary Dair
zachary.dair@mycit.ie

Extra Information:

The assignment was done in python 3.8, uses the following libraries:

Pandas, matplotlib, time and sklearn.

I reference the driving for loop multiple times, this is loop found at line 132, this loops through our samples sizes, and for each sample sizes loops through our models and for each model processes them accordingly, the perceptron runs fully only once per sample size, but actually runs 3 times internally due to the k-fold's number of splits.

The SVM with a linear kernel is similar to the perceptron as it only runs once per sample size.

There is then a further nested loop, allowing us to cycle through possible gamma values.

For the sake of speed, the sample sizes used in development are 10, 50, 100, 500, 1000, 2000, an additional list can be found on line 129, that will use the full dataset as well (0 uses the full dataset).

If anything is unclear, please let me know.

Table of contents:

Task 1: Pre-Processing and Visualisation

Loading the Dataset: *(Line 24-49)*

Using try and except to prevent an error if the file is not found, we read in the dataset contents using the pd.read_csv function.

Separating the labels and features: *(Line 44-47)*

Stores the contents of the 'label' column from our dataframe, we then copy the original dataframe, minus the 'label' column as this gives us just the feature vectors, we also convert these features to a numpy array

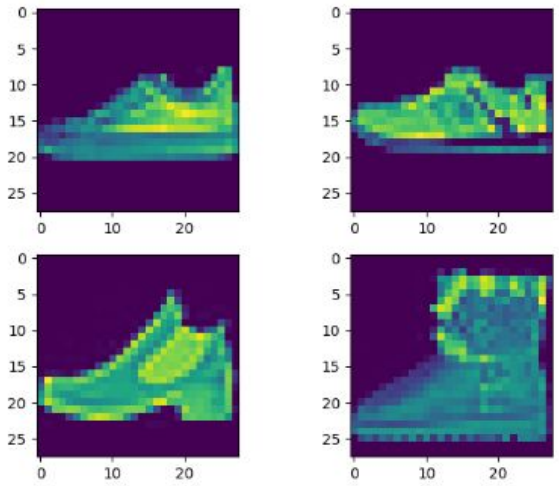
Counting the occurrences of each label: *(Line 10-11)*

Using our previously extracted data in the forms of two variables, the features and the labels, we can count the length of the labels variable where it's equal to 0 (the sneaker label) and again for 1 (the ankle boot label), these are then printed.

```
The dataset consists of 7000 sneakers and 7000 ankle boots.
```

Displaying the images of each label: *(Line 15-20)*

Using the matplotlib library we create a figure, and inside this figure we add two subplots for each classification type, using the imshow function, and the pixel data reshaped into 28x28, we can then see two images of each class.



Parameterise the dataset size: *(Line 31-34)*

We have encapsulated the loading of the dataset into its own function, allowing a parameter to be passed in this function, which corresponds to the number of rows of data to be read in, using an if statement in the function we check if the sample size entered was equal or less than 0, if that's the case, we read the full dataset instead. This allows us to specify an amount of data to be used in our evaluations.

```
# Try to load the csv data into a dataframe (use num rows to limit the size)
if sampleSize <= 0:
    data = pd.read_csv(filename)
else:
    data = pd.read_csv(filename, nrows=sampleSize)
```

```
# Define a list of sample sizes (these represent the amount of entries we take from the dataset)
# sampleSizes = [10, 50, 100, 500, 1000, 5000, 10000, 0]
sampleSizes = [10, 50, 100, 500, 1000, 2000]
```

Task 2: Perceptron

K-Fold Cross Validation: *(Line 112 & 65)*

We first define the k-fold with a number of splits, and then using the kfold.split function, we can create the train and test index to be used inside a for loop, this can be found inside the trainAndRunModel function as K-fold cross validation is needed for every model.

Train the perceptron classifier: *(Line 115 & 69)*

We define our perceptron classifier on line 115, using the linear_model part of the Sklearn library, this model is then appended to our model list.

A for loop is used to cycle through all of our models, each model calls the trainAndRunModel function, and on line 69 inside that function the perceptron classifier is trained using the feature data and labels.

Also this can be seen on line 140-146 where the model is run inside our driving for loop.

Predict using the perceptron classifier : *(Line 74)*

Using the .predict function on our model variable with at this time is the perceptron classifier we can run predictions on our test data from inside the k-fold loop.

Measure the training time: *(Line 68-70)*

Calling the time.time() function and storing the result before the fit function is called allows us to get the starting time, we then call the time.time function again directly after the fit function we can then minus the start time from the end time to give us the amount of time it took to train the model, this is done inside the k-fold loop.

Measure the prediction time: *(Line 73-75)*

Similarly to above, except we get the time directly before the model.predict function is called, and directly after, giving us the prediction time. This is also done inside the k-fold loop.

Accuracy Score: *(Line 78)*

Calling the metrics.accuracy_score function using our test labels and actual predictions we can get the accuracy of our given model. This is also done inside the k-fold loop.

Confusion Matrix: *(Line 79)*

Calling the metrics.confusion_matrix function using our test labels and actual predictions we can get the confusion matrix of our given model. This is also done inside the k-fold loop.

Displaying model results inside the k-folds loop: *(Line 80-85)*

Our trainAndRunModel function takes an additional parameter, a boolean value representing verbosity, using this we have an if statement on line 80, that allows the model accuracy, training and prediction times and confusion matrix to be printed to the user, if the verbosity is set to true, this results in the previously stated information being displayed, however as this is done inside the k-fold loop there is quite a lot of output, hence the decision to make the output toggleable.

```
Model: Perceptron Running with a Dataset of 500 entries.

Model Accuracy: 0.9461077844311377
Training Time: 0.011971235275268555
Prediction Time: 0.0019958019256591797
Confusion Matrix:
[[76  5]
 [ 4 82]]
```

(Individual results each time the model runs in the k-fold loop)

Above is the expected output when verbosity is true.

Min Max Average: *(Line 91-93 & 53-54)*

We are asked for the min, max and average of the training times, predictions times and accuracies, due to this frequency this needs to be calculated it's encapsulated in a function at line 53-54, where we get the minimum, maximum of an iterable, we also get the mean by getting the sum and dividing it by the length of the iterable.

This function is then called with the corresponding lists of values and the results outputted to the user, we also store some of these results in a dict that is returned for future evaluation.

Run on a range of sample sizes: *(Line 129-177)*

A list is initialized, containing a range of size values (a commented line at 130 contains a higher range of values including 0, which represents the whole dataset) but for testing and development purposes we use the smaller list at line 130.

We then use a for loop to cycle through each sample size, and for each sample size extract the contents from our dataset, and the loop through our models, allowing each model to process the data at that sample size.

```
Model: Perceptron Running with a Dataset of 2000 entries.
Training Time:
Min: 0.05285763740539551
Max: 0.07532525062561035
Avg: 0.06254808108011882

Prediction Time:
Min: 0.004987001419067383
Max: 0.009953022003173828
Avg: 0.006966114044189453

Accuracy:
Min: 0.8920539730134932
Max: 0.9594594594594594
Avg: 0.931513972743358
```

```
Model: Perceptron Running with a Dataset of 1000 entries.
Training Time:
Min: 0.0219419002532959
Max: 0.030916690826416016
Avg: 0.02593231201171875

Prediction Time:
Min: 0.0029883384704589844
Max: 0.0029931068420410156
Avg: 0.002990881601969401

Accuracy:
Min: 0.9431137724550899
Max: 0.9459459459459459
Avg: 0.9450018881156605
```

The above is the average training, prediction and accuracy for the Perceptron on different sample sizes.

Task 3: Support Vector Machine

K-Fold Cross Validation: *(Line 112 & 65)*

As previously mentioned, our cross-validation is encapsulated in the trainAndRunModel function.

Train the SVM (linear) classifier: *(Line 115 & 69)*

We define our SVM classifier on line 118, using the svm.SVC part of the Sklearn library, and the kernel parameter set to 'linear', this model is then appended to our model list.

A for loop is used to cycle through all of our models, each model calls the trainAndRunModel function, and on line 69 inside that function the linear SVM classifier is trained using the feature data and labels.

Also this can be seen on line 172-177 where the model is run inside our driving for loop.

Train the SVM (rbf) classifier: *(Line 115 & 69)*

We define our SVM classifier on line 119, using the svm.SVC part of the Sklearn library, and the kernel parameter set to 'rbf', we also omit the gamma value as this will be added in a nested loop (line 157), this model is then appended to our model list.

As with the previous two models, the rbf svm is also processing inside the main driving for loop.

We use conditional checking to identify when the model is the SVM using an rbf kernel, once that is the case, we define a list of gamma values from 1e-5 to 1e-10, for each of these gamma values, we assign them to the classifier and then we run our model in our trainAndRunModel function.

Common Classifier Functions : *(Line 58-102)*

1. Training the classifier (line 69)
2. Prediction with the classifier (line 74)
3. Measuring training time (line 68-70)
4. Measuring predicting time (line 73-75)
5. Retrieving the accuracy score (line 78)
6. Retrieving the confusion matrix (line 79)
7. Min max and average times and scores (line 91-93 & 53-54)

As in Task 2, these are all found in the trainAndRunModel function

Optimal SVM RBF Gamma Value: *(Line 192-205)*

Inside the driving for loop, when the model is an SVM with an RBF kernel we loop multiple times, using different gamma values, for each gamma value we store the result in a dict.

As our driving for loop, loops through sample sizes, then models, and finally in the case the model is an rbf it loops through gamma values, we have a wide range of results, and in order to get the mean accuracy for each gamma value, we need to store a running total, we do this using the dict.

Once our driving for loop has ended, we can then calculate the mean of the gamma accuracies, this is the running total of a certain gamma value, divided by the amount of sample sizes.

We can then store the best gamma accuracy and the value that gave us this result.

Run on a range of sample sizes: *(Line 129-177)*

As with the two prior models, our driving for loop allows us to cycle through various sample sizes.

Task 4: Comparison

Runtime and Accuracy (size 100):

Using the average training, prediction time and score (this is the average of the values from inside the k-folds) I observed the following outputs.

These results were found using dataset length 100:

Model	Avg Training Time	Avg Prediction Time	Avg Score
Perceptron	0.00265	0.00033	0.919
SVM Linear	0.00299	0.00116	0.949
SVM RBF Gamma: 1e-05	0.00664	0.00299	0.479
SVM RBF Gamma: 1e-06	0.00631	0.00299	0.910
SVM RBF Gamma: 1e-07	0.00299	0.00166	0.960
SVM RBF Gamma: 1e-08	0.00498	0.00199	0.939
SVM RBF Gamma: 1e-09	0.00631	0.00299	0.479
SVM RBF Gamma: 1e-10	0.00631	0.00265	0.480

This is the python output:

```
100, 'Model': 'Perceptron', 'Train Time': 0.0026587645212809243, 'Predict Time': 0.0003333886464436849, 'Score': 0.9197860962566845}
100, 'Model': 'SVM [Linear]', 'Train Time': 0.002991914749145508, 'Predict Time': 0.0011674563090006511, 'Score': 0.949494949494495}
100, 'Model': 'SVM[RBF, Gamma:1e-05]', 'Train Time': 0.006648461023966472, 'Predict Time': 0.002992550532023112, 'Score': 0.47979797979797983}
100, 'Model': 'SVM[RBF, Gamma:1e-06]', 'Train Time': 0.006316026051839192, 'Predict Time': 0.002992073694864908 7, 'Score': 0.9102792632204397}
100, 'Model': 'SVM[RBF, Gamma:1e-07]', 'Train Time': 0.0029919942220052085, 'Predict Time': 0.0016624132792154949, 'Score': 0.960190136660725}
100, 'Model': 'SVM[RBF, Gamma:1e-08]', 'Train Time': 0.004984617233276367, 'Predict Time': 0.001994450887044271, 'Score': 0.9396910279263221}
100, 'Model': 'SVM[RBF, Gamma:1e-09]', 'Train Time': 0.006316184997558594, 'Predict Time': 0.002991676330566406 2, 'Score': 0.47979797979797983}
100, 'Model': 'SVM[RBF, Gamma:1e-10]', 'Train Time': 0.006315867106119792, 'Predict Time': 0.002659400304158529, 'Score': 0.4800950683303624}
```

(The initial 100, is the dataset size)

Runtime and Accuracy (size 2000):

Using the average training, prediction time and score (this is the average of the values from inside the k-folds) I observed the following outputs.

These results were found using dataset length 2000:

Model	Avg Training Time	Avg Prediction Time	Avg Score
Perceptron	0.08277	0.00864	0.925
SVM Linear	0.37444	0.13382	0.942
SVM RBF Gamma: 1e-05	2.20124	1.02820	0.480
SVM RBF Gamma: 1e-06	1.10474	0.49701	0.952
SVM RBF Gamma: 1e-07	0.49171	0.21380	0.949
SVM RBF Gamma: 1e-08	0.64141	0.29627	0.934
SVM RBF Gamma: 1e-09	1.25414	0.59778	0.920
SVM RBF Gamma: 1e-10	2.05656	1.01561	0.489

This is the python output:

```
2000, 'Model': 'Perceptron', 'Train Time': 0.0827784538269043, 'Predict Time': 0.008646408716837565, 'Score': 0.9250194722458591}
2000, 'Model': 'SVM [Linear]', 'Train Time': 0.37444130579630536, 'Predict Time': 0.13382649421691895, 'Score': 0.9429999714857286}
2000, 'Model': 'SVM[RBF, Gamma:1e-05]', 'Train Time': 2.2021440664927163, 'Predict Time': 1.028203010559082, 'Score': 0.48000249124686906}
2000, 'Model': 'SVM[RBF, Gamma:1e-06]', 'Train Time': 1.1047401428222656, 'Predict Time': 0.49701984723409015, 'Score': 0.9525027276151713}
2000, 'Model': 'SVM[RBF, Gamma:1e-07]', 'Train Time': 0.4917104244232178, 'Predict Time': 0.21380090713500977, 'Score': 0.9490029759894827}
2000, 'Model': 'SVM[RBF, Gamma:1e-08]', 'Train Time': 0.6414181391398112, 'Predict Time': 0.2962762514750163, 'Score': 0.9349987168577872}
2000, 'Model': 'SVM[RBF, Gamma:1e-09]', 'Train Time': 1.2541414101918538, 'Predict Time': 0.5977816581726074, 'Score': 0.9205022113567841}
2000, 'Model': 'SVM[RBF, Gamma:1e-10]', 'Train Time': 2.0565617084503174, 'Predict Time': 1.0156184037526448, 'Score': 0.48999599299449376}
```

(The initial 2000, is the dataset size)

My Model Choice:

From analysing the above data, the perceptron classifier seems to outperform the rest in terms of speed, with training and prediction time much faster than the other models.

However despite being close to the top in accuracy, it is still outperformed by the Linear, and several RBF SVM models.

The SVM with a linear kernel is faster than the SVM with a RBF kernel using gamma 1e-07, however SVM RBF Gamma 1e-07 outperforms it in terms of accuracy, and is only narrowly slower, but an important notice is that SVM RBF Gamma 1e-07 performed consistently across the board, performing close to top in speed and accuracy each time.