Machine Learning Assignment 2

rESULTS AND report

Emmett Fitzharris R00222357

2024

# Introduction

My main function is used to call the functions for each task. It also calls the read\_data function to load information from the CSV file, and caches the file once read.

A screen shot of a computer program

Description automatically generated

# Task 1

In task 1, I start by slicing the data frame to only include our target labels, 5,7 and 9.

I then split the labels from the features using the separate\_labels\_and\_features function.

|  |  |
| --- | --- |
| A screen shot of a computer program  Description automatically generated | A computer screen shot of a code  Description automatically generated |

A screen shot of a computer

Description automatically generated

The data in this file consists of 28x28 pixels, represented as greyscale brightness 0-255.

Using matplotlib, I create a plot the first of each unique class of labels, “Sandle”, “Ankle\_boot” and “Sneaker”.

A screen shot of a computer code

Description automatically generated

|  |  |  |
| --- | --- | --- |
| A close-up of a shoe  Description automatically generated |  |  |

# Task 2

Task 2 is completed in the “run\_classifier” function.

Firstly if I pass a value of number of samples, I take a sample of the feature set based on the parameter. If no parameter is passed, I default to using the full data set.

I create a kfold model, using the Sci-Kit Learn package, using 5 splits.

As training the models will be an expensive task in terms of processing power and time, next I implement multi-core processing using pools from the multiprocessing library. Rather than using a single core, I use all but two of the available cores on the system, so I can exploit more of the processing power available on my computer without overloading it.

The pools will call the “train\_and\_evaluate\_fold” function, passing the “args” variable as parameters.

I collate the results for each fold, then print the accuracy, training time and prediction time, and print the confusion matric of each split.

I return the results back to the main function.

A computer screen shot of text

Description automatically generated

The “train\_and\_evaluate\_fold” function prepares the data, implementing feature scaling to ensure uniformity.

I measure the time for training, and the time for prediction, then test for the accuracy of predictions against the test data.

A computer screen shot of a program

Description automatically generated

I created a function to print the confusion matrix, so I can call it for each split.

I also made one to extract a summary of the results data frame, which will be used later to find required information easily.

A screen shot of a computer

Description automatically generated

I set the global variable “num\_samples” to 10% of the total set data for development. The classifiers will be evaluated at a number of sample sizes.



Task 2 does not produce any output itself, but will do so when called in future tasks

# Task 3

In task 3 I implement the Perceptron Classifier. This is available from SciKit Learn, so the implementation is quite simple. I create an instance of the classifier, then pass it to the “run\_classifier” function from task 2.

A screen shot of a computer

Description automatically generated

For each fold we see an output like the following:

A screenshot of a computer

Description automatically generated

Once the classifier is trained and evaluated, we can see an accuracy score of 0.89 on average.

A black background with white text

Description automatically generated

Finally, I create a function which can be used to plot the sample size against the runtime of a classifier. This not only handles the diagram via matplotlib, but also the collection of the runtime information by looping through a set of sample sizes and calling the “run\_classifier” function for each. We will use this function going forward both for this task and subsequent classifiers.

A computer screen shot of a program code

Description automatically generated

Based on the graph below, we can see that the run time increases slowly at first, then speeds up somewhat after a sample size of 10000. Oddly it seems that the run time dips after 15000, which may indicate that a decision boundary was hit earlier.

A graph with a line

Description automatically generated

# Task 4

The implementation of task 4 is almost identical to the implementation of task 3, this time using a Decision Tree Classifier model, also offered by Sci-Kit Learn. Once again I find the average accuracy for the default sample size, then plot the run time at varying sample sizes.

A screen shot of a computer screen

Description automatically generated

A graph with a line

Description automatically generated

# Task 5

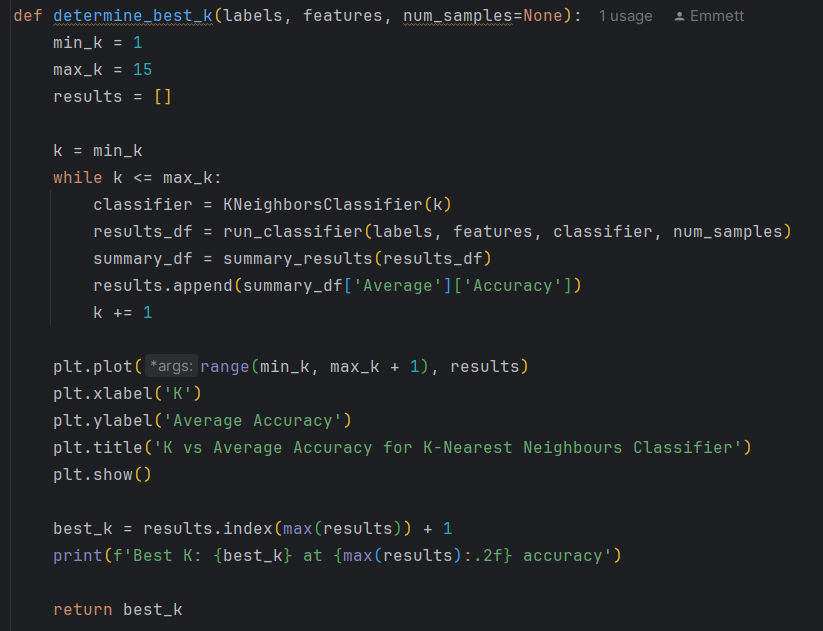
In task 5 we test the K-Nearest Neighbours Classifier. This time the implementation does change quite significantly from previous tasks.

I start by finding the optimal value for k . This is calculated in the “determine\_best\_k” function, which iterates through a range of values from 1 to 15, training a classifier model for each using the “run\_classifier” command from task 2.

The iteration with the highest mean accuracy is taken to be the optimal value of k. A classifier with this value is passed to the “plot\_sample\_size\_vs\_runtime” function to gather data on the runtime at differing sample sizes as with the previous tasks.

A screen shot of a computer program

Description automatically generated

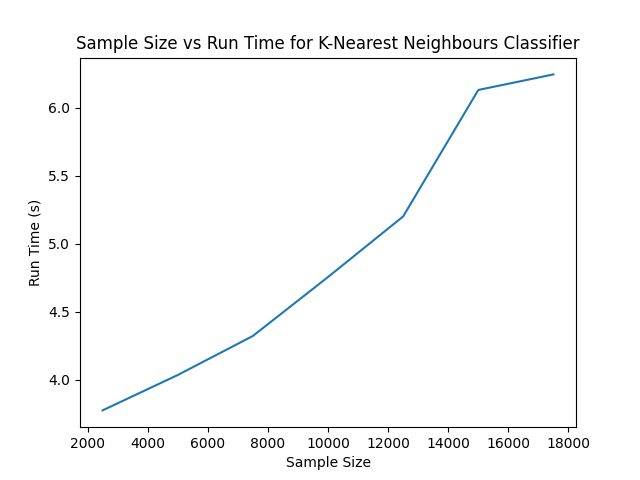


A graph with a line

Description automatically generated

Based on the information calculated above, we can see the ideal value for k is 2 at an accuracy rating of of 0.9





# Task 6

In task 6 I implemented the Support Vector Machine Classifier, using the radial basis function kernel.

This kernal has two parameters, gamma and C, making the optimization a 2D problem. In the function “determine\_best\_y\_value\_for\_rbf” I define a list of values for each parameter, in a dictionary. I will then implement a grid search to find the optimal values.

A computer screen shot of text

Description automatically generated

A computer screen with many colorful text

Description automatically generated

To visually confirm the results are sensible, I use a heatmap, plotting gamma against C, with the heat representing the accuracy.

|  |  |
| --- | --- |
| A screen shot of a computer code  Description automatically generated | A red and yellow squares  Description automatically generated |

Using the above method, It is determined that the best parameters are C of 10 and a Gamma of 0.001.



As with the tasks before, I then plot varying sample sizes against the run time for this model:  
A graph with a blue line

Description automatically generated

# Task 7

In task 7, I compare each of the above four classifiers, Perceptron, Decision Tree, K-Nearest Neighbour, and Support Vector Machine.

I start by printing their summaries, and follow up by creating three basic bar charts, the first comparing the accuracy of each classifier, the second comparing the training time, and the third comparing the prediction time of each classifier.

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

In terms of accuracy, we can see that all models performed within a close margin. The lowest accuracy being Decision tree, and the highest being Support Vector Machine.   
This intuitively makes sense, due to the high dimensionality of the data.

A graph of blue rectangular bars

Description automatically generated

For training time we see quite a bit of variance, with high training times for Decision tree and SVM. Decision trees are recursive, which causes a rapid rise in training time as depth increases. SVM models are computationally complex as they handle high degrees of complexity.

The Nearest Neighbour model has virtually no training time, which makes sense, as this model is just based on distance in prediction. It doesn’t require training.

The perceptron model is less computationally expensive, and as such the training time is much lower.

A graph of a number of blue bars

Description automatically generated

Finally, in terms of prediction time, we see that the nearest neighbour model takes the longest by far, which is expected as the distance between nodes is calculated at prediction time.

The decision tree and perceptron models are almost instantaneous, which is expected as their prediction methods are very simple.

A graph of a number of blue bars

Description automatically generated

Overall, I would recommend the Support vector machine model if training time is not an obstacle, as it offers the highest accuracy, with a middling prediction time.

# Aside

Building on my formatting functions from the last assignment, this time I implemented a class with static methods. This was used to add dividers and headers to the output, as well as provide a progress bar to show visual progress on intensive tasks.

A screen shot of a computer program

Description automatically generated

The total runtime of the program is approximately 9 minutes.

A screen shot of a computer

Description automatically generated