CS401 HW2 – BONE [SUPERVISED ML]

Student Name: David Ryan

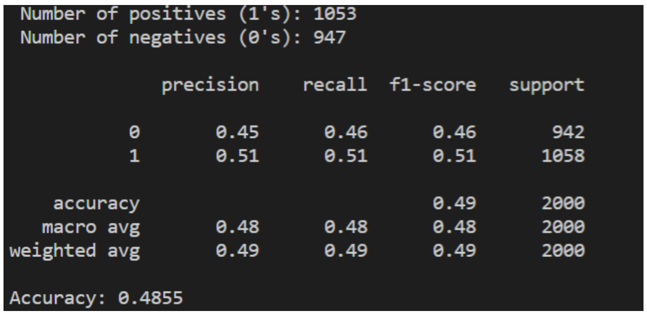
Student Number: 17417364

I started this assignment by looking into the tools available for carrying out a supervised Machine Learning task, specifically those that are digestible for a beginner in the field. I settled on using Python along with VSCode as an editor due to my familiarity with them. Regarding libraries, I made use of pandas for reading in data and Sk.learn for access to classifiers and metrics to gauge the success of my approaches.

Using pandas read_csv method, to take in the assigned data and specified that the files contained no headers and were separated by spaces rather than commas. I set up the script to train based on all the training set and predict based on the test set using an arbitrary model for the time being, and following this, print the out to a new text file in the desired format.

Once this was set up, I went about trying out different classifiers and playing around, tuning their parameters to produce the best result. I read into each of the available models available through Sk.learn and figured the best way to see what fit best would be to try each of them out.

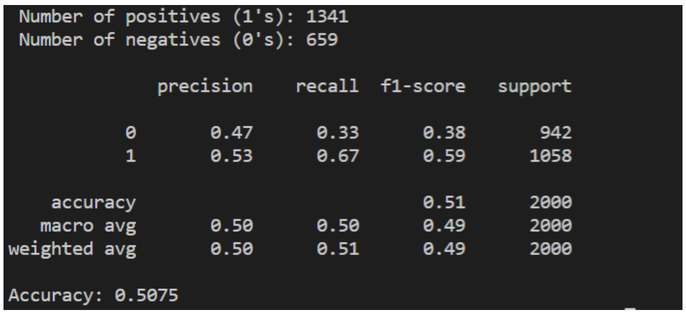
To be able to gauge each model's performance, a validation set had to be created. I split a subsection of the training set by 80% and 20% into a training and test set respectively, allowing the script to grade itself.



```
 Number of positives (1's): 1053
 Number of negatives (0's): 947

              precision    recall  f1-score   support

           0       0.45      0.46      0.46       942
           1       0.51      0.51      0.51      1058

    accuracy                           0.49      2000
   macro avg       0.48      0.48      0.48      2000
weighted avg       0.49      0.49      0.49      2000

Accuracy: 0.4855
```

1 Decision Tree

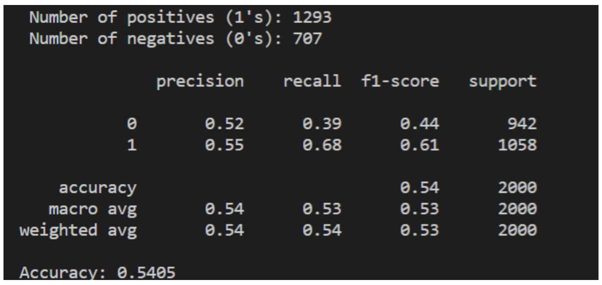
The decision tree performed the worst out of all the models with an accuracy of 49%. Being below the 0.5 line of an ROC curve, this model was incorrect more than it was correct.



```
 Number of positives (1's): 1341
 Number of negatives (0's): 659

              precision    recall  f1-score   support

           0       0.47      0.33      0.38       942
           1       0.53      0.67      0.59      1058

    accuracy                           0.51      2000
   macro avg       0.50      0.50      0.49      2000
weighted avg       0.50      0.51      0.49      2000

Accuracy: 0.5075
```

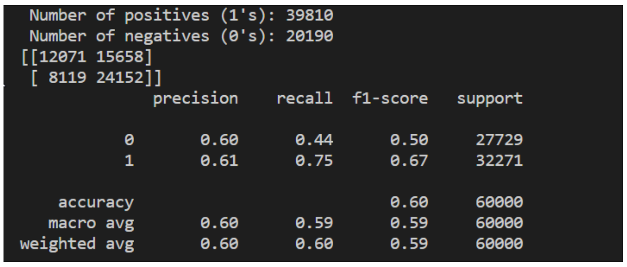
2 Support Vector Machine - Polynomial Kernel

This was followed in performance by a Support Vector Classifier using the polynomial kernel, with an accuracy of 51%. Then by K Nearest Neighbour and Support Vector Machine using a Radial Based Function in the kernel, pictured below along with their respective scores.



```
  Number of positives (1's): 1293
  Number of negatives (0's): 707

              precision    recall  f1-score   support

           0       0.52      0.39      0.44       942
           1       0.55      0.68      0.61      1058

    accuracy                           0.54      2000
   macro avg       0.54      0.53      0.53      2000
weighted avg       0.54      0.54      0.53      2000

Accuracy: 0.5405
```

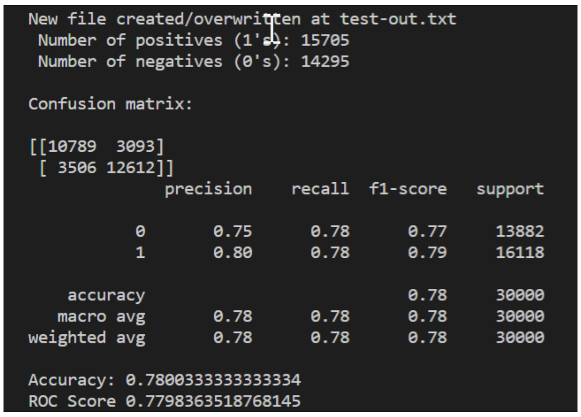
3 KNN



```
  Number of positives (1's): 39810
  Number of negatives (0's): 20190
 [[12071 15658]
  [ 8119 24152]]
              precision    recall  f1-score   support

           0       0.60      0.44      0.50     27729
           1       0.61      0.75      0.67     32271

    accuracy                           0.60     60000
   macro avg       0.60      0.59      0.59     60000
weighted avg       0.60      0.60      0.59     60000
```

4 SVM - RBF Kernel

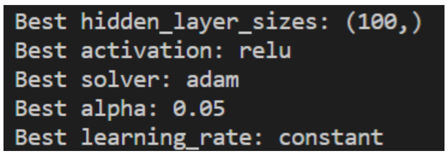
Finally, the classifier that performed best by far for me was the Multi-Layer Perceptron. This had the highest accuracy and ROC score by a significant amount in comparison to the other classifiers I tried. With an accuracy and ROC Score of 78%, the MLP was much more accurate in its predictions. The confusion matrix for this classifier is also included in the below figure. I used Sk.learn's metrics library to complete these computations.



```
New file created/overwritten at test-out.txt
 Number of positives (1's): 15705
 Number of negatives (0's): 14295

Confusion matrix:

[[10789  3093]
 [ 3506 12612]]
              precision    recall  f1-score   support

           0       0.75      0.78      0.77     13882
           1       0.80      0.78      0.79     16118

    accuracy                           0.78     30000
   macro avg       0.78      0.78      0.78     30000
weighted avg       0.78      0.78      0.78     30000

Accuracy: 0.7800333333333334
ROC Score 0.7798363518768145
```

5 Multi-Layer Perceptron

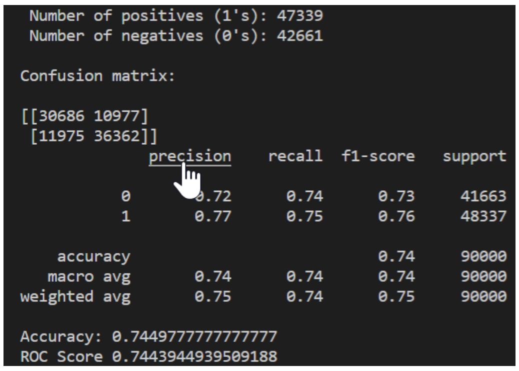
This model performed very well for me on a subsection of the training data set (split 80/20 for validation), however, did not perform well on even smaller subsections of the set. I tried to tune the parameters to boost performance by making use of Sk.learn's Gridsearch library. This allows you to set a dictionary of the parameters you want to try and after testing on each combination, decides upon the best set of parameters. These were the parameters it decided on after being ran on our data.



```
Best hidden_layer_sizes: (100,)
Best activation: relu
Best solver: adam
Best alpha: 0.05
Best learning_rate: constant
```

6 Gridsearch's parameter suggestions

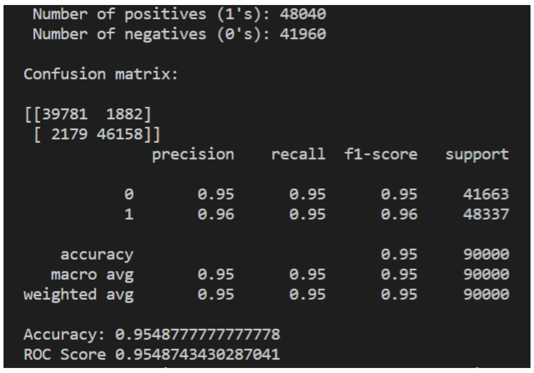
However, funnily enough, after setting the model to train based on these parameters, I found slightly worse performance. This prompted me to go back to my original set of parameters. I feel like I must have made a mistake somewhere in my code for this to be the case.



```
 Number of positives (1's): 47339
 Number of negatives (0's): 42661

Confusion matrix:

[[30686 10977]
 [11975 36362]]
              precision    recall  f1-score   support

           0       0.72      0.74      0.73     41663
           1       0.77      0.75      0.76     48337

    accuracy                           0.74     90000
   macro avg       0.74      0.74      0.74     90000
weighted avg       0.75      0.74      0.75     90000

Accuracy: 0.7449777777777777
ROC Score 0.7443944939509188
```

7 MLP's performance with suggested parameters

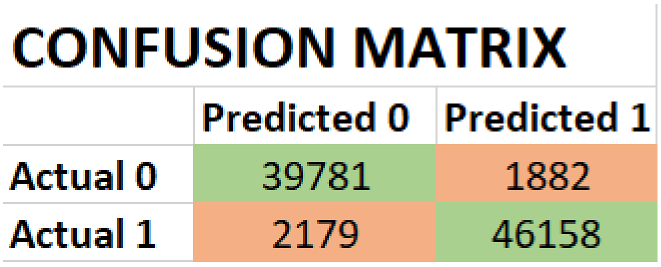
Having decided on MLP as a classifier and choosing my own parameters, I tested against the entirety of the training set with the same 80/20 split and received an accuracy and ROC score of 95%. I am hoping this accuracy carries over to the true test set!



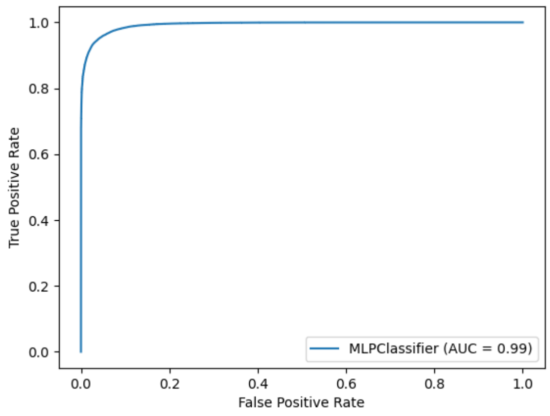
```
  Number of positives (1's): 48040
  Number of negatives (0's): 41960

 Confusion matrix:

 [[39781  1882]
  [ 2179 46158]]
               precision    recall  f1-score   support

           0       0.95      0.95      0.95     41663
           1       0.96      0.95      0.96     48337

    accuracy                           0.95     90000
   macro avg       0.95      0.95      0.95     90000
weighted avg       0.95      0.95      0.95     90000

 Accuracy: 0.9548777777777778
 ROC Score 0.9548743430287041
```

8 Final MLP performance

Here is the corresponding Confusion Matrix and ROC Curve:



CONFUSION MATRIX		
	Predicted 0	Predicted 1
Actual 0	39781	1882
Actual 1	2179	46158


True Positive Rate
False Positive Rate
MLPClassifier (AUC = 0.99)