Machine Learning Report – Max Woolterton

For my 2 models I chose to use a Random Forest Classifier (sklearn.ensemble.RandomForestClassifier) and a C-Support Vector Classifier (sklearn.svm.SVC). I opted to use two classifiers instead of regressors as the task is to predict discrete labels. I selected the Random Forest because I believed that, with its use of combining many simple and fast-to-train Random Trees, this method would achieve a good balance between training time and accuracy for the given problem, whilst also requiring less data preparation than many. I subsequently decided to pick a Support-Vector-Classifier, as I believed this classifier was suited to being able to construct a complex fitting model whilst not overfitting the data (by tuning of its parameters).

Data Analysis and Preparation:

My first step was to investigate trends between the data from the OULAD and the students' results (our label), in order to decide what data could be helpful to our models, and how best to encode this data. I did this via plotting my data fields against the label, using matplotlib, to observe their distributions.

One of the most promising data correlations was one I made by summing all the VLE clicks per student of any resource, which I named "sum_click". I originally did the data manipulation without any additional libraries, but due to the size of the dataset, I decided to use 'pandas' to vastly increase efficiency. **Figure.1** plots all of the data, however I realised that, due to the vast disproportion of labels, it would be better to plot my features using 1000 randomly sampled data pairs from each label type, as displayed in **Figure.2** where the correlation between "sum_clicks" and "final_grade" can be seen.

Figure.1

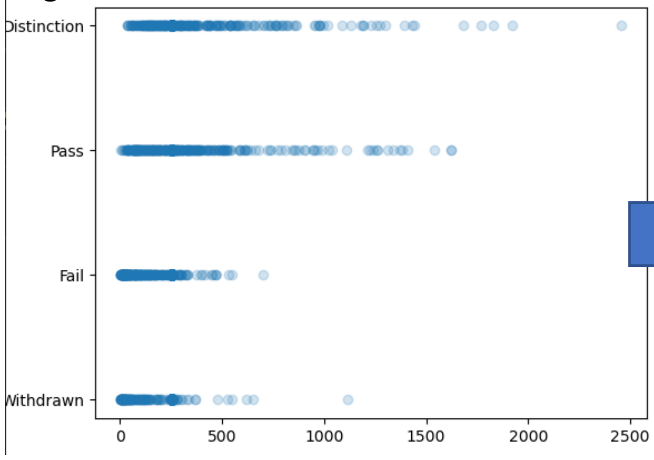
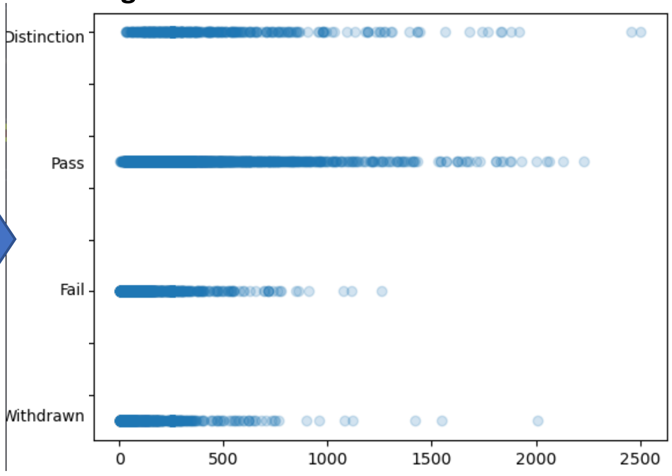


Figure.2

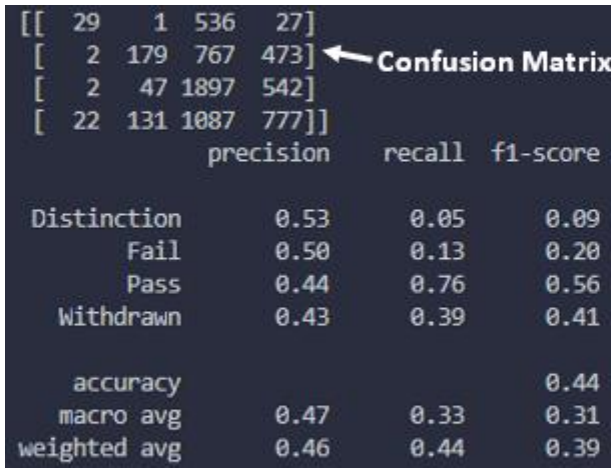


After inspecting all correlations I decided the following features would be relevant to the model: "imd_band", "highest_education","imd_band","num_of_prev_attempts","final_result", "studied_credits","age_band","disability","gender", "sum_click" and "average_score". The feature "average_score" was calculated from the average of all assessments for that student in the records. I then decided to integer encode all my features bar "disability","gender" and "age_band," which I used One-Hot Encoding for. I came to these decisions by comparing the performance of the models when using the two different encoding methods on each of the individual features. All missing or corrupt data entries were replaced by the mean in order to not affect the model's performance.

For the SVC I also normalized all my data using sklearn.preprocessing.StandardScaler in order to obtain optimal results. This resulted in a 5% increase in overall accuracy, but it is of note that this also caused an increase of .39 in F1 score for the “Distinction” label (**Figure.3**).

Finally, I tried both under-sampling and over-sampling data labels from over and under-populated labels respectively, but this had no overall performance gain.

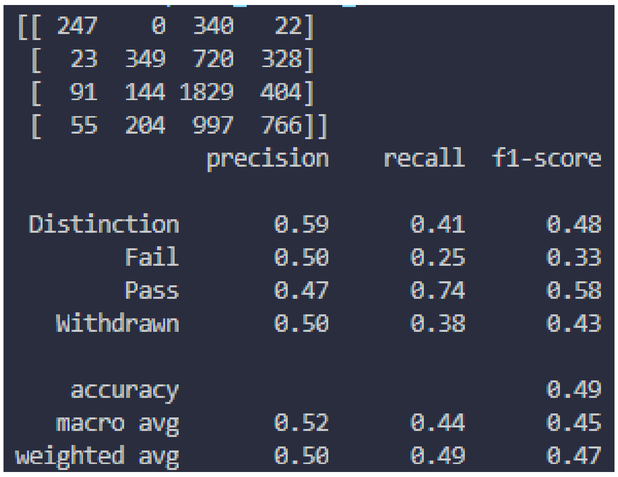
Figure.3



```
[[  29    1  536   27]
 [   2  179  767  473]
 [   2   47 1897  542]
 [  22  131 1087  777]]
              precision    recall  f1-score

 Distinction       0.53      0.05      0.09
        Fail       0.50      0.13      0.20
        Pass       0.44      0.76      0.56
   Withdrawn       0.43      0.39      0.41

    accuracy                           0.44
   macro avg       0.47      0.33      0.31
weighted avg       0.46      0.44      0.39
```



```
[[ 247    0  340   22]
 [  23  349  720  328]
 [  91  144 1829  404]
 [  55  204  997  766]]
              precision    recall  f1-score

 Distinction       0.59      0.41      0.48
        Fail       0.50      0.25      0.33
        Pass       0.47      0.74      0.58
   Withdrawn       0.50      0.38      0.43

    accuracy                           0.49
   macro avg       0.52      0.44      0.45
weighted avg       0.50      0.49      0.47
```

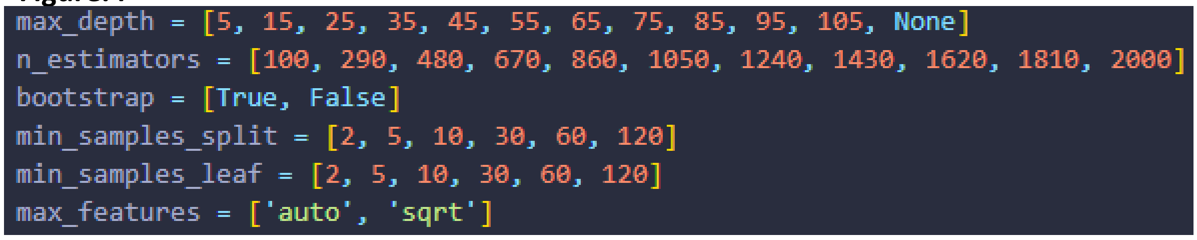
Parameter Search:

In order to measure performance, I decided to compare the confusion_matrix, and the F1 scores from the classification_report, both found in the sklearn.metrics module, on identical unseen data samples where possible, by setting the random seed parameter in many of the methods described. I decided to use the confusion_matrix as it shows a detailed summary of in which cases false-positives and false-negatives negatives occurred, allowing me to guide the model in the right direction with the choosing of which data fields to use as features and how, for example, exactly to integer encode them. I decided to use F1 score, as it encompasses both precision and recall and therefore is a much better indicator than accuracy when there is a large imbalance in label proportions. However, for the purpose of this report, overall accuracy shall also be stated as a concise summary of the performance of a model on all labels. The other performance indicator I used was training time, as I felt it important to achieve a balance between good results and time taken to achieve them.

Initially in order to train my models I used sklearn.model_selection.train_test_split(test_size=0.20), however once I was happy with my features and moved on to hyperparameter tuning, I began to use sklearn.model_selection.RandomizedSearchCV() in order to perform a 3-fold randomized cross-validation search.

To tune my Random Forest I ran sklearn.model_selection.RandomizedSearchCV() over 100 iterations on following grid of parameters to narrow down the range in which I was searching for optimal parameters in:

Figure.4

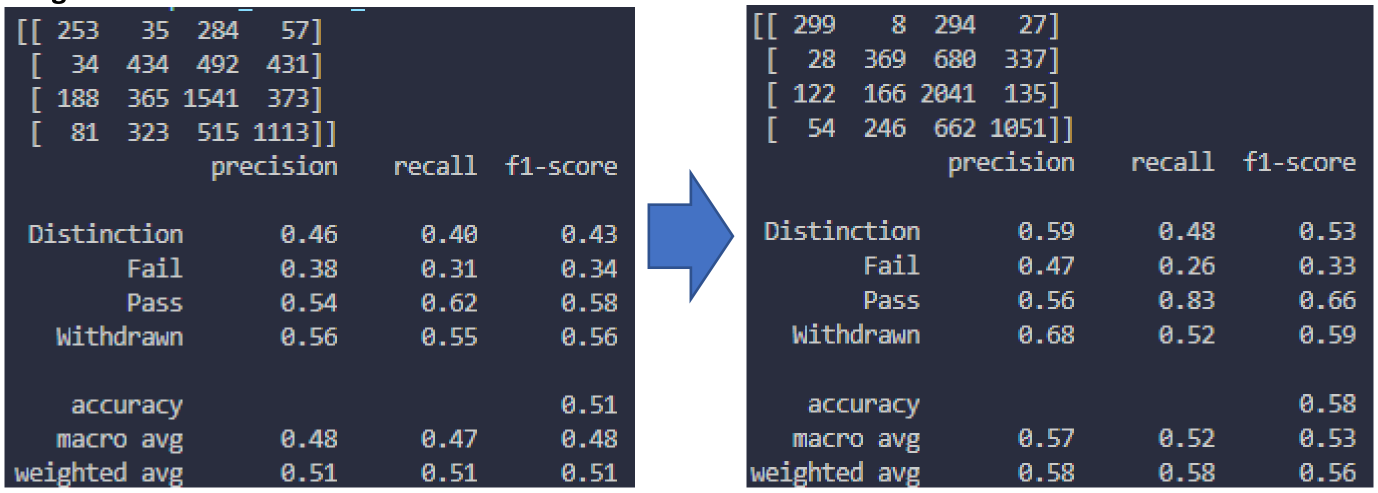


```
max_depth = [5, 15, 25, 35, 45, 55, 65, 75, 85, 95, 105, None]
n_estimators = [100, 290, 480, 670, 860, 1050, 1240, 1430, 1620, 1810, 2000]
bootstrap = [True, False]
min_samples_split = [2, 5, 10, 30, 60, 120]
min_samples_leaf = [2, 5, 10, 30, 60, 120]
max_features = ['auto', 'sqrt']
```

I then used sklearn.model_selection.GridSearchCV() three successive times in order to search all combinations of parameters within the narrowed down grids I created. In the case of my Random Forest, I concluded that none of my parameter changes affectedtraining time enough for it to be considered as a factor.

Overall, hyperparameter tuning resulted in a 7% increase in accuracy, as can be seen in **Figure.5**, after returning the following hyperparameters: 'bootstrap'=True,'max_depth' =42,'max_features'='auto', 'min_samples_leaf'=2,'min_samples_split'=85,'n_estimators'=410.

Figure.5



```
[[ 253   35  284   57]
 [  34  434  492  431]
 [ 188  365 1541  373]
 [  81  323  515 1113]]
              precision    recall  f1-score

 Distinction       0.46      0.40      0.43
        Fail       0.38      0.31      0.34
        Pass       0.54      0.62      0.58
   Withdrawn       0.56      0.55      0.56

    accuracy                           0.51
   macro avg       0.48      0.47      0.48
weighted avg       0.51      0.51      0.51
```

```
[[ 299    8  294   27]
 [  28  369  680  337]
 [ 122  166 2041  135]
 [  54  246  662 1051]]
              precision    recall  f1-score

 Distinction       0.59      0.48      0.53
        Fail       0.47      0.26      0.33
        Pass       0.56      0.83      0.66
   Withdrawn       0.68      0.52      0.59

    accuracy                           0.58
   macro avg       0.57      0.52      0.53
weighted avg       0.58      0.58      0.56
```

However, my SVC hyperparameter tuning proved far more challenging. I attempted to adopt the same approach, but after leaving sklearn.model_selection.RandomizedSearchCV() running for 2 hours, it had not managed to check a single combination of parameters out of my 100 combinations. I discovered that this was because kernel being set to "poly" did not return a result after being run for over an hour, while "sigmoid" took 4x longer than "rbf", achieving 0.12 less accuracy. Therefore due to training time I decided to set kernel="rbf". I then ran sklearn.model_selection.RandomizedSearchCV(), followed by sklearn.model_selection.GridSearchCV(), in order to finely tune "gamma" and "C", only to discover that the optimal parameters were the default ones, "auto" and 1.0, leaving the final accuracy of my SVC model at 0.49 (**Figure.2**).

Conclusions:

Overall, I concluded that with this data set, the Random Forest model performed much better at predicting students' grades due to its higher performance and accuracy (0.58 vs. 0.49) at predicting the given labels along with its vastly shorter training times (2sec vs 3min). I also found that hyperparameter tuning for the Random Forest was far easier, due to training times per SVC being over 90 times longer than that of a forest. This is without considering that many of the parameter options for SVC took considerably more than this base statistic. Although, it is of note that for most data sets, the optimal gamma parameter does not require any fine tuning, as the algorithm used by "auto" is near optimal, and in our case, the default values for C and gamma were also optimal. Therefore, it could be argued that SVC does not benefit from, nor require, hyperparameter tuning and therefore takes less time to tune than the Random Forest model, which required the tuning of 6 hyperparameters and vastly improved performance. Additionally, a Random Forest requires less data preparation as data does not have to be normalised.