Machine Learning Assignment Report

1. Introduction

I am comparing the Random Forest Classifier (RF) and Decision Tree Classifier (DT) models to predict a student's final grade using the OULAD [1]. Using the datasets provided, I will first prepare the data into a suitable form, and then split the data into training and testing subsets. The models are then fitted on the train sets and predictions are made on the test set. These predictions will be used to evaluate the performance of these models. I will be using the scikit-learn implementation of the algorithms.

2. Data Preparation

The OULAD provides the datasets in CSV format. Using the pandas library I can perform data exploration to get insights and quickly find out what data is relevant/irrelevant. I am going to be using the three datasets 'studentInfo.csv', 'assessments.csv' and 'studentAssessment.csv'.

From looking at the merged data, the columns that give unique rows are "id_student", "code_presentation" and "final_result". This gives a unique row for each student receiving a final result in a course. I then one-hot encode all the categorical type data. This is best practice for classifiers - it removes the error where the model assumes $0 < 1 < 2$ etc. if label encoding was used - as it uses binary columns for each option of each original column. I then remove the withdrawn students to make it a 3 class classification problem and group the data. I remove withdrawn to improve accuracy as they could have been on track to get any result. I created the columns "score" and "score_mean". I can look at the correlation matrix of the data to see how much each row correlates to "final_result". This gives a range of results from 0.5 to -0.14. Here is a sample:

Column	Correlation
score_mean	0.528
date_submitted	-0.005
num_of_prev_attempts	-0.138

The last thing is to split it into a test/train set and remove the labels into a separate dataset.

3. Experimental Procedure

Two models thought to be appropriate were RF and DT as they are both models that work well for classification problems. I briefly tested SVM Regression and other regressors, but with the same data it performed worse. I wanted to use the same data on the two models to compare the difference in performance of two classifiers, so this ruled out regressors as they would not be comparable.

After the data was processed properly, I started training the models. Initial training was done on the basic model with no changes to the parameters to get an idea of the performance of each model. For both models, they performed poorly; I need to tune the parameters. The chosen metric to maximise was f1-weighted. After hyper-tuning I want the models to produce the highest f1-weighted score possible. I chose f1-weighted score as this gave a better representation of how the model performed w.r.t. the class imbalance.

After the initial fitting, the rest of the procedure was to use GridSearchCV to iteratively search parameter spaces for the best parameters to use and repeat for different combinations of parameters and ranges.

During the process, slight changes were made to the data to increase performance, e.g.

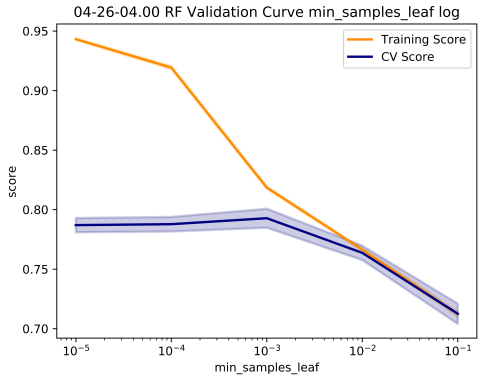
changing to one-hot-encoding from label encoding, and summing/mean score columns. This meant the model fitting process had to start again, but gave better results. Before the new score columns the highest score was 65% but afterwards this rose to nearer 79%.

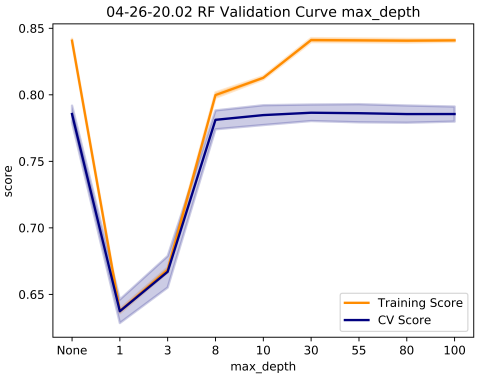
4. Parameter Search and Selection

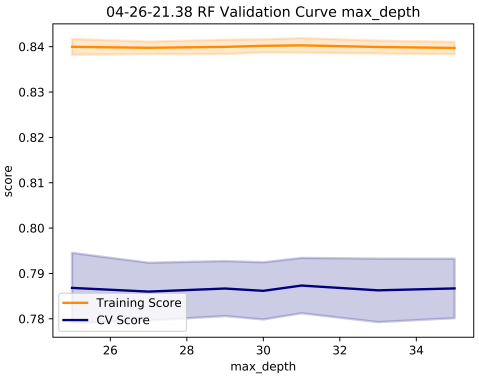
To select parameters for the two models I used GridSearchCV from sklearn to hypertune the parameters. Once the GridSearch completed, I plotted the f1-weighted score vs parameter value plus standard deviation to visualize CV results. There were multiple iterations of CV run for both the DT and the RF. The initial searches were broad sweeps trying to find global maxima for values and then subsequent searches were done in these regions to tune the parameters to find the best results. Here is an example parameter grid used in GridSearchCV on the RF:

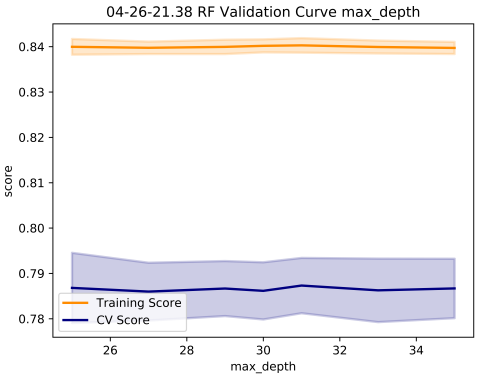
```
params_grid = [
    {
        'criterion': ['gini', 'entropy'],
        'bootstrap': [True],
        'class_weight': [None],
        'max_depth': [40, 45, 50, 55, 60],
        'max_features': [1, 150, len(data.dataset.columns)],
        'min_samples_leaf': [0.00001, 0.0001, 0.001, 0.01, 0.1],
        'min_samples_split': [0.00001, 0.0001, 0.0005, 0.001, 0.01, 0.1],
        'n_estimators': [500],
        'n_jobs': [-1]
    }
]
```

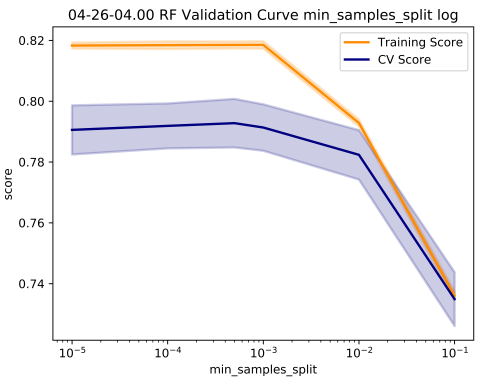
This grid shows how it is narrowing down on ("max_depth"), ("min_samples_split/_leaf") and doing a broad sweep on ("max_features") Here are a few graphs to demonstrate the GridSearchCV results.

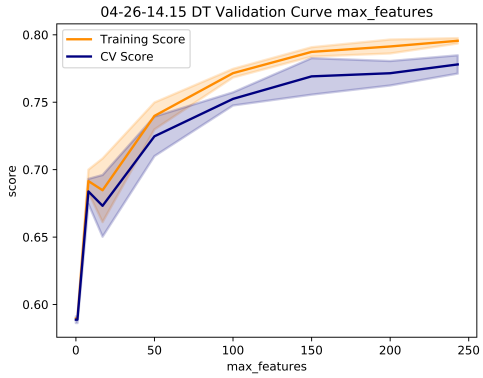


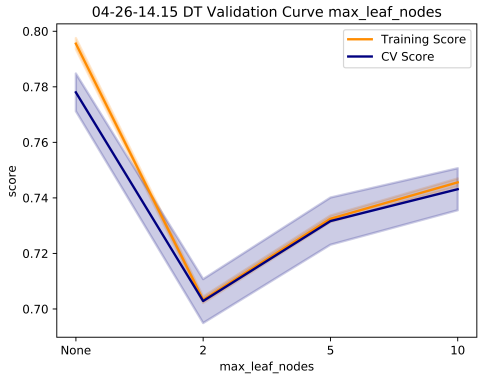


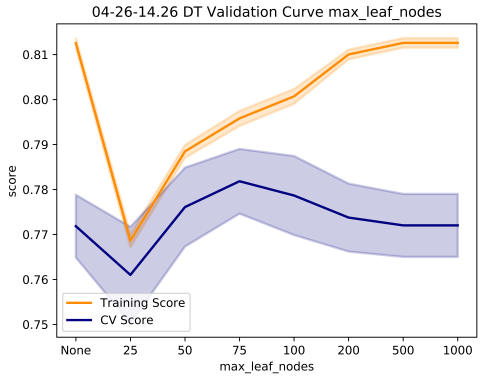












These graphs show the importance of hyper-parameter-tuning as we can see that changing the parameters slightly causes large changes in accuracy of the model.

5. Performance of the Models

The metrics being used to evaluate the performance of the models are root mean squared error (RMSE) - even though this is a regressor metric, it gives an interesting insight into the performance of the models - accuracy, balanced accuracy and f1-score. RMSE is a measure of the standard deviation of the prediction errors, the lower this value the better as there is less error in the predictions. Accuracy is what percentage of predictions are correct, balanced accuracy is the average or recall obtained on each class. It helps with imbalanced datasets. F1-score is another accuracy measure based on the precision and recall of each class. It penalizes more for false positives/negatives than the other measures do as it takes the harmonic mean of the recall and precision accuracies.

The tuned RF managed to produce these results:

```
Test Accuracy:   78.89%
Test Balanced Accuracy: 69.47 %
Test MSE:        0.2161
Test RMSE:       0.4649
Test report:
              precision    recall  f1-score   support

           1       0.88      0.73      0.80      1171
           2       0.77      0.90      0.83      2410
           3       0.66      0.45      0.54       597

    accuracy                           0.79      4178
   macro avg       0.77      0.69      0.72      4178
weighted avg       0.79      0.79      0.78      4178
```

As we can see, for fail and pass it was very good at predicting, however for distinction (3) the f1-score was relatively low due to the recall. Low recall here means that it has a high number of false negatives. This is due to the lack of data for this class.

The tuned DT managed to produce:

```
Test Accuracy:   76.59%
Test Balanced Accuracy: 67.25 %
Test MSE:        0.2427
Test RMSE:       0.4926
Test report:
              precision    recall  f1-score   support

           1       0.84      0.74      0.79      1171
           2       0.76      0.87      0.81      2410
           3       0.60      0.41      0.49       597

    accuracy                           0.77      4178
   macro avg       0.73      0.67      0.69      4178
weighted avg       0.76      0.77      0.76      4178
```

These two models turned out quite similar performance wise, with the RF coming out on top.

6. Conclusion

From this experiment I was able to see the importance of processing data correctly for the models. The models fitted quite well, however due to the lack of instances in certain classes, this created class imbalance which impacted the performance of the models. Without training, the models exhibited strange behaviour where they would overpredict the train set and under predict the test set. This shows the importance of tuning the models parame-

ters.
I have also shown that the RF outperformed the DT which is to be expected, however not by as much as I expected. This could be due to the parameter tuning creating a better DT model than RF.

References

[1] Kuzilek, Jakub, Martin Hlosta, and Zdenek Zdrahal. *Open university learning analytics dataset.* Scientific data 4 (2017): 170171.