Project 4: Enron Corporate Fraud Analysis

1. Introduction

Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it.

1.2 Overview

The aims of this project are to investigate the Enron scandal using the publicly available data and machine learning to determine a valid algorithm to identify POI - People of Interest, with respect to the scandal. The Enron scandal was one of the largest corporate scandals of the time after shorting its own share price from $90.75 to $1. Large amounts of data from this scandal was released to the public making this a prime example of a case where machine learning can be put to good use.

1.2 Machine Learning

Machine learning is useful in identifying persons of interest in this case study because it is powerful at finding trends, identifying and categorising that data, and use that data to improve to identify new datasets trends.

Given the size of the dataset it was possible to do an initial manual investigation into the data to remove extreme outliers. While looking through the data two key outliers were found. Firstly a value of “TOTAL” was identified, this represented total values and was not a person and as a result was dropped from the dataset. Similarly, another outlier was found “THE TRAVEL AGENCY IN THE PARK”. This appears to be a company account and not that of an employee and was also dropped from the dataset. These were removed with a .pop function.

1.3 Overview of the Data

- Number of People in the dataset: 146 prior to omitting TOTAL and TRAVEL AGENCY accounts
- Each person has 21 features
- The dataset contained 18 out of 34 previously identified POIs
- The accounts of :
  ['KOPPER MICHAEL J', 'FASTOW ANDREW S', 'YEAGER F SCOTT', 'HIRKO JOSEPH'] have missing or NaN data in their email correspondences

2. Feature Selection and Creation

2.1 Feature Creation

When I considered what parameters might be indicative of a person committing fraud my initial thoughts were through excessive bonuses in comparison to the person's salary. As a result i produced a new feature called 'bonus_salary_ratio' that compared the bonuses and the respective salaries. This new feature was then added to the feature list.

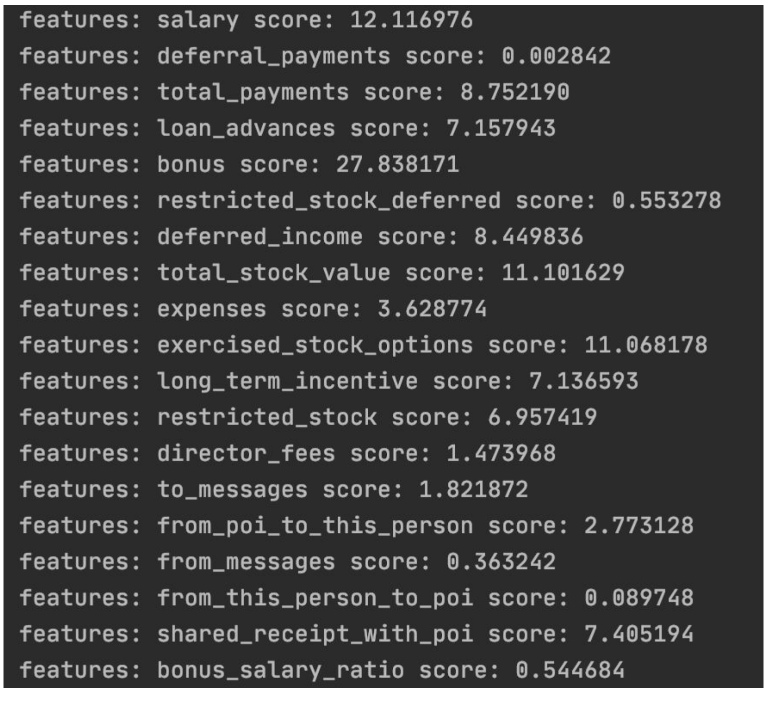
2.2 Feature Scaling

Given that the data in this dataset is financial data and email addresses etc, there is a large disparity between the magnitudes of values in the data. As a result without scaling the data the information can not be accurately compared. As a result I implemented a MaxMinScaling to correct this and allow for proper evaluation.

2.3 Feature Selection

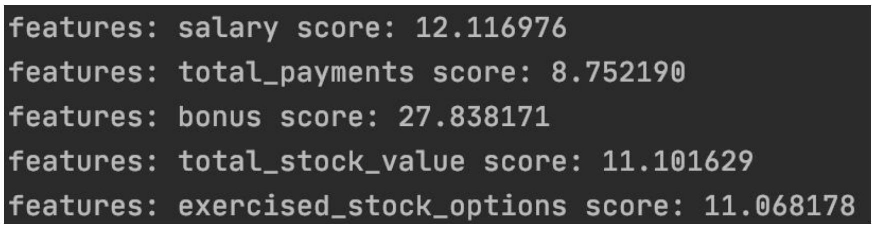
Feature selection is a key stage of ensuring a good algorithm output. In order to identify my key features to use within the algorithm I performed a SelectKBest function across the entire list of features with respect to POI in order to identify which the most heavily weighted features were.

From this, I initially took the top 10 k-score features and printed them to a new list which redefined the features_list variable.



```
features: salary score: 12.116976
features: deferral_payments score: 0.002842
features: total_payments score: 8.752190
features: loan_advances score: 7.157943
features: bonus score: 27.838171
features: restricted_stock_deferred score: 0.553278
features: deferred_income score: 8.449836
features: total_stock_value score: 11.101629
features: expenses score: 3.628774
features: exercised_stock_options score: 11.068178
features: long_term_incentive score: 7.136593
features: restricted_stock score: 6.957419
features: director_fees score: 1.473968
features: to_messages score: 1.821872
features: from_poi_to_this_person score: 2.773128
features: from_messages score: 0.363242
features: from_this_person_to_poi score: 0.089748
features: shared_receipt_with_poi score: 7.405194
features: bonus_salary_ratio score: 0.544684
```

However during the tuning phase of the project it was identified that a better result was derived using the same algorithm parameters but a feature list limited to the top 5 most powerful features.


```
features: salary score: 12.116976
features: total_payments score: 8.752190
features: bonus score: 27.838171
features: total_stock_value score: 11.101629
features: exercised_stock_options score: 11.068178
```

When selecting my features it was also very interesting to notice how the feature I had generated actually scored very low on the kbest values indicating that my thoughts regarding what could identify a POI were completely wrong.

3. Using Algorithms

3.1 Initial GaussianNB experimentation.

In order to run the file firstly I had to amend a few errors in the provided tester.py file as a result of an update to the sklearn libraries. The first algorithm that I looked at was a simple Gaussian Naive Bayes. This was the first algorithm I looked at as it didn't have any parameters that could be tuned.

Using a complete feature_list the results of the GaussianNB were:
Accuracy: 0.71687 Precision: 0.23571 Recall: 0.50100 F1: 0.32059 F2: 0.40895

Using k=10
Accuracy: 0.82213 Precision: 0.32550 Recall: 0.31150 F1: 0.31834 F2: 0.31420

And k=5
Accuracy: 0.85387 Precision: 0.41892 Recall: 0.24800 F1: 0.31156 F2: 0.27003

This showed that my feature selection was accurate and was enabling the algorithm to more accurately identify new potential POI but up to a certain point it was starting to reduce the algorithm's ability to recall POIs. This interested me to see how my feature list impacted the results of another algorithm type which had more variable parameters to tune. Especially given that the GaussianNB algorithm at k=10 was able to pass the thresholds of the project.

3.2 Decision Tree

I decided to use DecisionTree as my second algorithm after having investigated Adaboost and SVM gridsearchcv and struggled to get an accurate response from either of those two algorithms when it came to tuning.

An initial run of my code using a random series of parameters of DecisionTree and a complete features_list resulted in the following:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
        max_features=None, max_leaf_nodes=None,
        min_impurity_decrease=0.0, min_impurity_split=None,
        min_samples_leaf=1, min_samples_split=5,
        min_weight_fraction_leaf=0.0, presort=False, random_state=None,
        splitter='random')
Accuracy: 0.81693    Precision: 0.29706    Recall: 0.27300     F1: 0.28452   F2:
0.27750
```

This showed to me that the DecisionTree classifier was a more balanced algorithm with respect to Precision and Recall as the two values were more in line with each other in comparison to GaussianNB.

4. Tuning the algorithm

Now that I had identified how to use the DecisionTree algorithm and seen its untuned results I had to tune the parameters to meet my specification. Tuning is the process of selecting the best parameters in order for an algorithm to optimise its performance. Tuning is important as it can allow different algorithms to run in different environments such as on different hardwares, with specific workloads etc. A tuned algorithm also is more likely to lead to a successful training of an algorithm. If an algorithm is untuned or poorly tuned it can take exceptionally long times to complete while providing you with a poor final outcome.

Following an initial look at DecisionTree, a search was performed using GridSearchCV in order to identify the most appropriate parameters for this algorithm. This was performed at K=10 initially, this produced a set of tuned parameters from the GridSearchCV function:

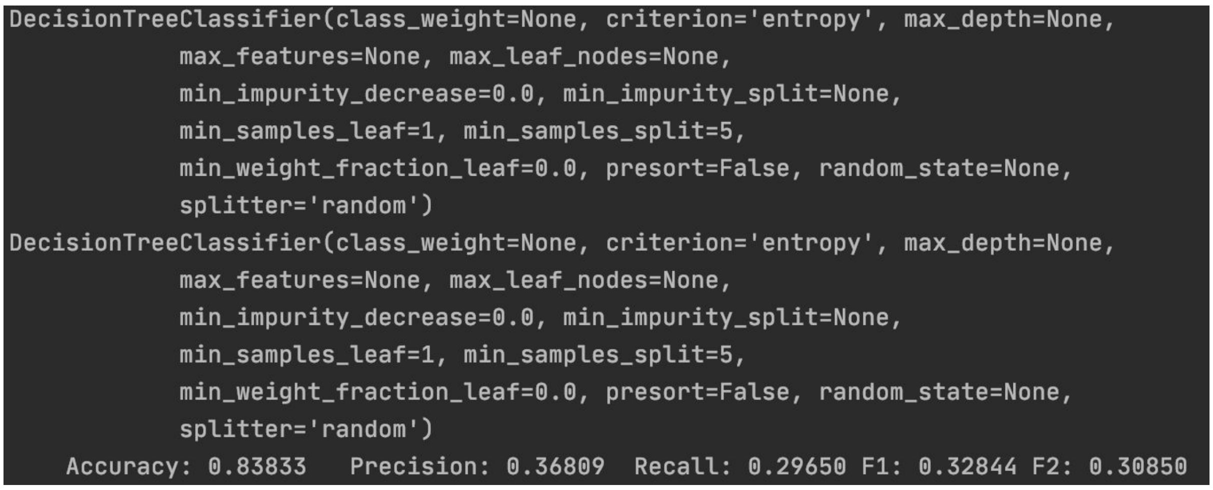
```
done in 0.094s
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
        max_features=None, max_leaf_nodes=None,
        min_impurity_decrease=0.0, min_impurity_split=None,
        min_samples_leaf=1, min_samples_split=2,
        min_weight_fraction_leaf=0.0, presort=False, random_state=None,
        splitter='random')
```

As a result when putting these parameters into the algorithm the following results were produced:

Accuracy: 0.79820 Precision: 0.25606 Recall: 0.26950 F1: 0.26261 F2: 0.26670

As this did not meet the benchmark of a successful algorithm as stated in the rubric I was curious as to what the issue could be given these were supposed to be the tuned parameters. I tried changing the testing size from the 0.4 I had chosen at the start to 0.2 but this did little to aid the situation. I then recalled the effect that k=5 versus k=10 had on GaussianNB and wondered whether a smaller feature list would aid this case and whether a top 10 was providing too much conflicting information.

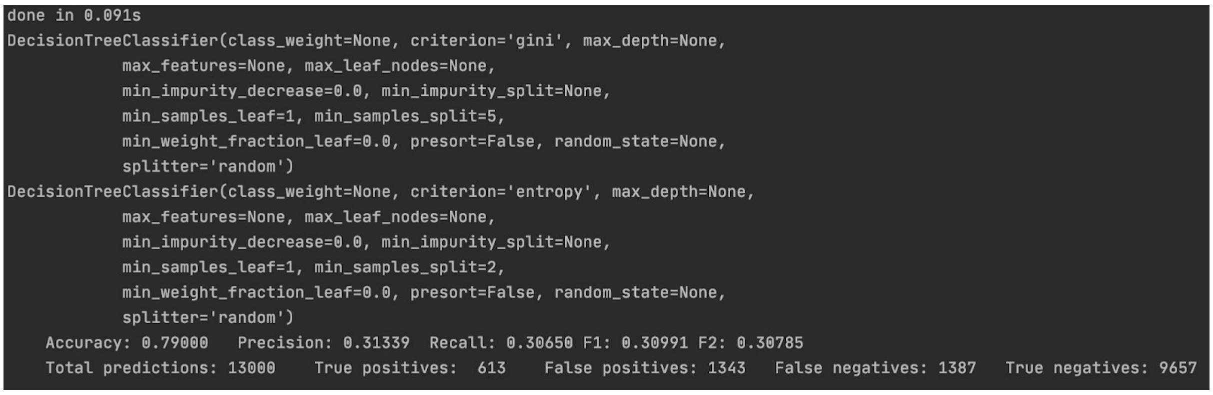
By reverting back to my initial 0.4 test size and switch k=6 the new tuned parameters were as follows according to the GridSearchCV



```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='random')
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='random')
    Accuracy: 0.83833   Precision: 0.36809  Recall: 0.29650 F1: 0.32844 F2: 0.30850
```

As you can see the Precision value shot up and was above the benchmark requirement but the recall was still not enough. By manually tuning from this point a parameter that I identified as affecting recall - min_samples_split to 2 instead of 5 as suggested I was able to bring the recall value up such that both precision and recall were within the required values.

Final result:



```
done in 0.091s
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=5,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='random')
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='random')
    Accuracy: 0.79000   Precision: 0.31339  Recall: 0.30650 F1: 0.30991 F2: 0.30785
    Total predictions: 13000    True positives:  613    False positives: 1343   False negatives: 1387   True negatives: 9657
```

5. **Validation**

It is important when training and testing a classifier that you validate the algorithm on a separate set of data than was used to train it. I experimented and used a series of splits of training/testing splits and they had varying effects on different algorithms. This resulted in me selecting a 40:60 ratio. This provided an adequate amount of training data and allowed for the algorithms to be validated against a larger (40% rather than 20/30%) of data. Given the small amount of data provided and judged that it would be more beneficial to use a smaller split to ensure there was a reasonable amount of data to be validated against. If validation is not performed against data that is separate from the training data then the results of the testing will not be representative of the algorithms' ability which can lead to massively skewed results hence why it is often very useful to employ a feedback loop of multiple testing phases to allow for the most optimal of training.

6. **Evaluation**

When comparing the two algorithms that have been used in this project, primarily Precision and Recall, as mentioned before, were the defining evaluation metrics of the algorithms' performance. Other metrics however were also looked at such as accuracy and the F1 score of the algorithm. F1 is a metric that encompasses both the precision and the recall of the algorithm.

The results of the two algorithms (when tuned if possible) at each feature_list size have been compared below using these metrics.

Algorithm	K	precision	recall	f1	accuracy
GaussianNB	all	0.23	0.501	0.32	0.72
DecisionTree (untuned)	all	0.25	0.27	0.26	0.70
GaussianNB	10	0.32	0.31	0.32	0.82
DecisionTree	10	0.368	0.297	0.309	0.83
GaussianNB	5	0.41	0.25	0.31	0.85
DecisionTree	5	0.31	0.309	0.31	0.79

Given these evaluation metrics it is possible to see that despite the lack of tunable parameters, the GaussianNB algorithm outperforms the DecisionTree algorithm at all stages. I believe that given a larger dataset and a larger training split of that data however, that the DecisionTree algorithm would outperform the GuassianNB, and that it is a more appropriate algorithm as the tunability if the algorithm allows for bespoke usage of it. As it is tunable the DecisionTree algorithm could be used to tune specifically for very high recall rates and low precision or vice versa, whereas with Naive Bayes you are unable to specify how you want it to operate.