Identifying Fraud from Enron data

-By Kiran Hegde

The goal of this project is to basically classify Enron employees as either Persons of Interest (POI) or Non-Persons of Interest (Non-POI) by looking at the financial data as well as the email data available to us in order to identify fraud. The data set mainly contains 145 data points or data of 145 employees among which 18 of them are identified as POIs. Each of these datapoins have 21 features ranging from salary details, bonus details, amount of stock owned to number of emails sent, number of emails recieved, number of emails shared with POIs and many more such features either related to financial details or email details of the employees.

Many features have missing values for various datapoints, which might affect of model's performance. The number of missing values for each of the features available are listed below:

1. salary : 51
2. deferral_payments : 107
3. total_payments : 21
4. loan_advances : 142
5. bonus : 64
6. restricted_stock_deferred : 128
7. deferred_income : 97
8. total_stock_value : 20
9. expenses : 51
10. exercised_stock_options : 44
11. other : 53
12. long_term_incentive : 80
13. restricted_stock : 36
14. director_fees : 129
15. to_messages : 59
16. from_poi_to_this_person : 59
17. from_messages : 59
18. from_this_person_to_poi : 59
19. shared_receipt_with_poi : 59

Machine learning can be seen as an apt technique for achieving this classification because of the characteristic of the it which involves exploiting information in available data by identifying various patterns and relating new datapoints with the identified patterns, inturn predicting the target label of the new datapoints. And this is exactly what we want for the classification problem at hand.

Since the number of datapoints is very less, we don't pay much attention to identifying and dealing with outliers as we don't want to risk losing information. However, one datapoint named "TOTAL" which is not data related any one particular employee is identified as an outlier and removed.

The initially available set of features seem to contain different kinds of information in them. Since the number of features we have is quite small, I don't want to miss out on any of those by manually choosing any particular set of features for testing my models. So, except from "poi" which we are using as a target variable and "email_address" which is a string containing the email address of the employees, we use the rest 19 of the available features to create and test our basic models.

Along with these features, 2 more features which might help our classification problem are manually created and added to the dataset. These two features are “fraction_from_poi” and “fraction_to_poi” which are fractions of total emails from POIs to the employee and fractions of total emails sent to POIs from the employee respectively. These features are created with hypotheses that the total fraction of emails POIs get from other POIs might be higher than the total fraction of emails Non-POIs get from POIs. This is similar with the fraction_to_poi feature, that it might have higher values for POIs than for Non-POIs. The effects of the new features on our model’s predictions are discussed later.

We intend to test GaussianNB and DecisionTreeClassifier for the purpose of selecting an algorithm for our classification model.

Most machine learning algorithms contain a large number of parameter settings available for each of them, hence it becomes important to tune the model for a particular required performance which is usually case dependent. In our case of identifying POIs, we intend to get a high “F1” score since our data is skewed in terms of number of POIs and Non-POIs in the dataset. Because GaussianNB has a very limited number of parameter settings available when compared to DecisionTreeClassifier, I expect the metrics from DT to beat those from GNB after tuning DT for the best parameter setting.

With the above aim in mond, we create a train/test split of the data to test the set of algorithms (GaussianNB and DecisionTreeClassifier) hoping to find one suitable algorithm which can further be tuned to get better results. The training and testing sets contain all of the initial 19 features plus 2 new features. The models were initially trained with almost default parameter settings to pick the best behaving one. The metrics used to gauge this were accuracy, precision and recall scores from each of the models. GaussianNB beat the DecisionTreeClassifier, however, not by much difference. Since we know that DecisionTreeClassifier has a far more number of parameter combinations when compared to GaussianNB, we believe that an optimized DecisionTreeClassifier would do much better. This is analyzed by tuning the DT model with a combination of parameters passed on to GridSearchCV while optimizing the “F1” score. The parameter options used for optimizing are as follows:

```
[{'criterion': ('gini', 'entropy'),
'min_samples_split':[2, 10, 20],
'max_depth':[10,15,20,25,30],
'max_leaf_nodes':[5,10,30]}]
```

This gives us the following parameter setting:

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=10,
                       max_features=None, max_leaf_nodes=10, min_impurity_split=1e-07,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False, random_state=24,
                       splitter='best')
```

The metrics from the above DecisionTreeClassifier model are also exactly equal to the metrics of our previously tested GaussianNB model which are as follows:

accuracy_score: 0.886363636364
recall_score: 0.4
precision_score: 0.5

But, we select DecisionTreeClassifier as the algorithm for creation of our final model since it is more flexible in terms of parameter settings when compared to GaussianNB which we hope will be helpful in getting better results after further model tuning.

All of the processes discussed so far trained and tested the models using all of the 21 features available. Since sometimes, irrelevant or very slightly relevant features do more harm than good while creating a prediction model, we now try to select the most relevant features that influence the label variable the most. This is done using SelectKBest function that scores the features of the dataset depending on how much they affect the outcome variable or label variable and removes all but 'k' highest scoring features (where k is a parameter passed to SelectKBest). However, since we don't know what value of 'k' would give the highest "F1" score, we use GridSearchCV and pass on all possible values that 'k' can take. This SelectKBest step was pipelined to the previously optimized DecisionTreeClassifier. The best_estimator_ we get out of this step along with the obtained metrics is as follows:

```
Pipeline(steps=[('skb', SelectKBest(k=10, score_func=<function f_classif at 0x7f57a91ab668>)),
        ('dt', DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=10,
        max_features=None, max_leaf_nodes=10, min_impurity_split=1e-07,
        min_samples_leaf=1, min_samples_split=2,
        min_weight_fraction_leaf=0.0, presort=False, random_state=24,
        splitter='best'))])
```

accuracy_score: 0.886363636364
recall_score: 0.8
precision_score: 0.5

From the optimized model, we can see that 'k' takes a value of 10. We now see these 10 features and their scores obtained by SelectKBest below:

salary : 15.8060900874
total_payments : 8.962715501
bonus : 30.6522823057
deferred_income : 8.49349703055
total_stock_value : 10.814634863
exercised_stock_options : 9.95616758208
long_term_incentive : 7.53452224003
restricted_stock : 8.051101897
shared_receipt_with_poi : 10.6697373596
fraction_to_poi : 13.7914132368

We see that one of our created feature "fraction_to_poi" having a score of 13.7914132368 has been selected during univariate feature selection while the other created feature "fraction_from_poi" has a score of 0.490609903605 is insignificant and has not been used in our model.

Since as of now, we have only trained and tested our models using one particular train/test split we made using train_test_split function, we further tune the model by cross validating it against our data using StratifiedShuffleSplit method with 100 folds. Also while cross validating, GridSearchCV is used with an aim of getting a parameter setting which produces the highest "F1" score across all 100 folds of splitting the data. The cross validation step is vital in the sense that our dataset is very small which would poorly simulate the real world scenario where the randomness is at a larger scale.

Also, StratifiedShuffleSplit specifically helps us include the facts of skewness of our dataset while validating our classification model by maintaining the ratio between the share of datapoints across the two classes (POI and Non-POI) constant during each train/test split.

Our final DecisionTreeModel obtained by the cross validation process is tested using test_classifier function from the tester module which outputs an average value of important metrics across 1000 folds of train/test splits obtained through StratifiedShuffleSplit. The parameter settings of our final model is as follows:

```
Pipeline(steps=[('skb', SelectKBest(k=10, score_func=<function f_classif at 0x7fb5911748c0>)),
        ('dt', DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=10,
        max_features=None, max_leaf_nodes=5, min_impurity_split=1e-07,
        min_samples_leaf=1, min_samples_split=20,
        min_weight_fraction_leaf=0.0, presort=False, random_state=24,
        splitter='best'))])
```

Also, the average metrics obtained from using test_classifier function on our final model is as follows:

Accuracy: 0.83280
Precision: 0.37996
Recall: 0.40200
F1: 0.39067
F2: 0.39739
Total predictions: 15000
True positives: 804
False positives: 1312
False negatives: 1196
True negatives: 11688

The interpretation of some important metrics obtained by testing our final optimized model is as follows:

- Accuracy of 0.8328 or 83.280% suggests that the model is classifying the POIs and Non-POIs correctly 85.8% of times.
- Precision of 0.37996 tells us that 37.996% of the times our model classified someone as a POI, it was correct. Or we can say that it precisely identified POIs 37.996% of the times.
- Recall of 0.402 tells us that our model classified 40.2% of the total number of POIs correctly. We can also say that the model recalled 40.2% of all POIs correctly.