Enron Submission

The task of this project is to identify individuals who were involved in fraud while working at ENRON. This is to be done using nothing but financial data and information from emails.

In order to go though both the financial and email data and develop a relation between this data and individuals suspected of fraud we are going to use machine learning.

The data provided also includes list of individual who were one way or the other determined by the government as person of interest (POI).

Using all this information, we are going to determine if a give individual is a person of interest (POI) or not. We shall use some part of the data for training our model and ultimately we shall test the accuracy of our model with the rest of the data.

Data Exploration

For this project we have 145 data points. Each point has 21 features including an email address and POI. Email address is not going to be used in our machine learning program and POI is going to be the label, the feature we are planning to determine. Of the rest of the features, 6 are from emails while the rest are financial features.

From the 145 points, there are 18 persons of interest.

Each feature has multiple missing information designated as NaN. The two features with most missing information are 'loan_advances' and 'director_fees' with 142 and 128 missing information respectively.

Outlier Investigation

In order to investigate outliers, we developed a formula. There are different ways to identify an outlier including through visualization. In our case, we have decided to use a formula as that automates the process. Our definition of an outlier is anything that is two standard deviation away from the mean of the data.

$$2 \times \boldsymbol{SD} + \boldsymbol{mean} \text{ >= Outlier >= } 2 \times \boldsymbol{SD} + \boldsymbol{mean}$$

The first outlier detected is the key 'TOTAL'. This is a key used to sum the data and gives no relevant information to our analysis. As such, it is removed from the data.

Another outlier that was also removed is BHATNAGAR SANJAY. This person has a *restricted_stock_deferred* of approximately 15.5 million. This is by far larger than any restricted stock deferred attributed to anyone. This is probably a typo error and so, it is removed from the final data set.

While there were other outliers, none of them were removed as they seem to be relevant to the process. Also, these outliers though detected as outlier by the equation, they weren't far away from the general population. For example, there were outliers in *from_ messages* and *from _this_person_to poi* features. But these are informations you want to keep as the expectation is that people in the same circle will communicate more often with each other than with the rest of the employees.

Feature Selection

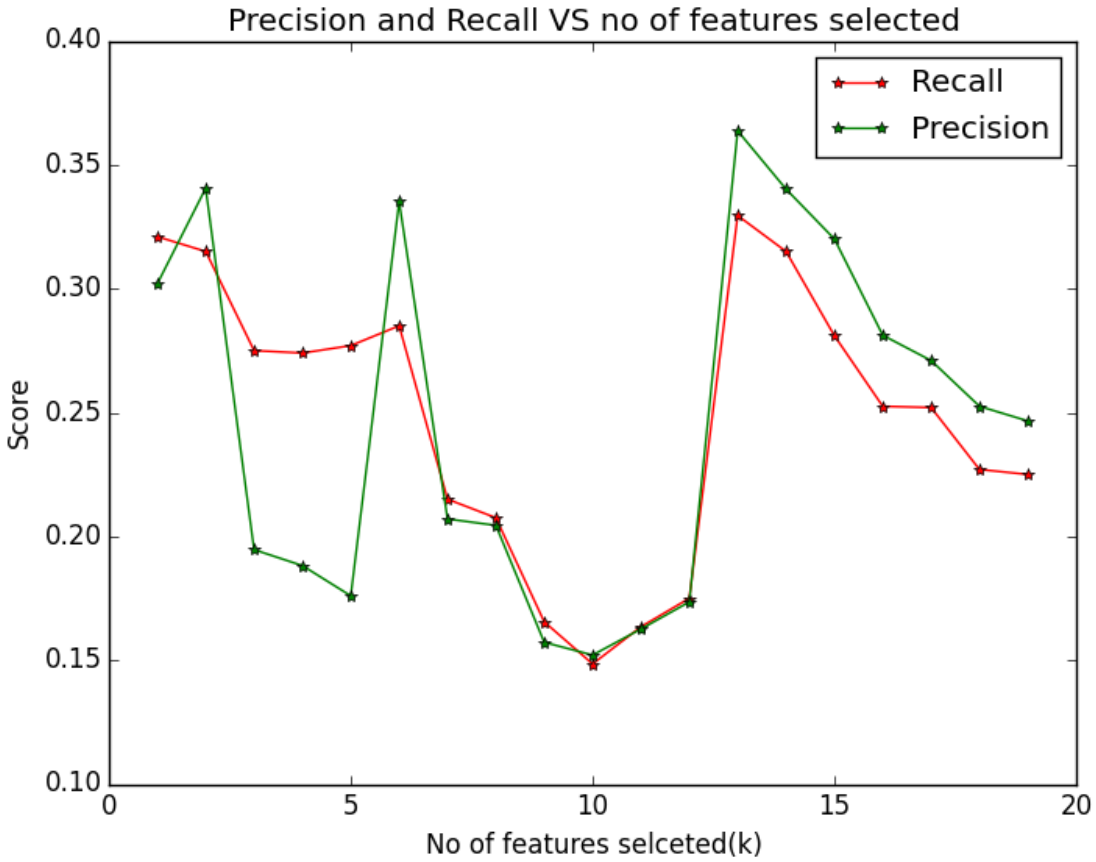
Of the twenty features, we needed to identify which features are relevant and most optimal for our classification. To do this we used the ***SelectKBest*** feature selection from sklearn. This built in algorithm tries to identify the optimal features that give you the highest possible score. To do that you will have to specify the number of features you want.

In our case, we looped through 1-19, to find the best combination of features. So, for each number (n) we try to find the best **n** features. To measure the score for each combination, we used a slightly modified version of tester.py.

Precision and Recall were used for scoring. These scoring parameters are chooses over others including accuracy basically because of the data we have. We have an asymmetric data where only 18 of the 145 are persons of interest.

We didn't need to use any scaling as the classifier used was decision tree classifier.

Here is the score of the feature selection test(Figure 1):



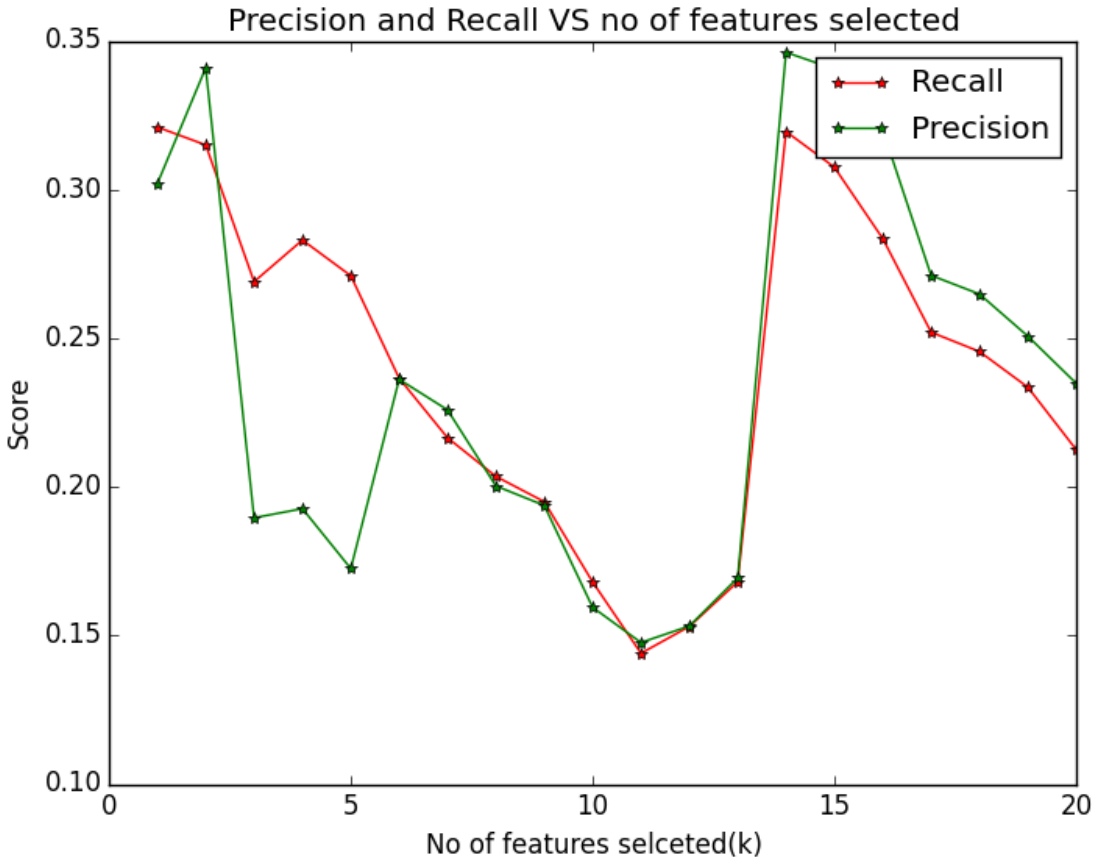
As can be inferred from the plot, a combination of 13 features gave us the highest precision and recall scores. The 13 features selected by ***SelectKbest*** were,

'from_this_person_to_poi','shared_receipt_with_poi','loan_advances','deferred_income', 'expenses','other','director_fees','to_messages',deferral_payments','total_payments','restricted_stock_deferred','from_poi_to_this_person','from_messages' .

New Feature Creation

In addition to those features, we tried to come up with a new feature i.e. the sum of all non-salary based payments. In other words, the difference between total payments and salary. Our expectation is that people in the inner circle are going to be paid a hefty amount of money but not in salary format. We think they will be reward with stock options and bonus and so on. Thus, finding the total amount of money paid to individuals above their base salary can signal how the company looked at those individuals.

Here is the earlier plot with the new feature included(Figure 2):



The best result is obtained when we have 14 features. Those features are the same as the earlier features plus the new feature *'Non_salary_payments'.* Looking at figure 1 and 2, we see that the new

feature didn't improve the precision and recall, in fact it might have lowered it a bit. So, we won't be using this new feature in the final analysis.

Picking a Classifier

To find out which classifier does better with our data set, we have tested multiple classifiers (Decision tree, Random forest, KNeighbors, Naïve Bayesian and SVM). Since, the performance of each classifier depended on how you set their parameters, GridSearch was used. GridSearch will give us the best parameters for the given classifier so that ultimately we are comparing the classifiers at their best performance.

Setting the parameters right is very important. For example, in the case of the decision tree, if we sent minimum sample split to 2, the tree will go on splitting until we are left with just 2. This in turn could create an over fitting of the classifier. Similarly, depending on the criterion chosen, we can have different techniques of splitting the tree.

We used the F1 score to determine the performance of each classifier. F1 is used as it incorporates both precision and recall it is calculation.

Decision Tree:

Best Parameters : *'min_samples_split' =6,*
'criterion'=entropy]
Score: 0.55

Random Forest:

Best Parameters: *'n_estimators'=10,*
'criterion'=entropy,
'min_samples_split'=4
Score: 0.38

KNeighbors:

Best Parameters: *'n_neighbors'= 3*
Score: 0.13

Naivebayesian:

Best Parameters: None
Score: 0.21

As with the SVM, a linear VS rbf was tested. Unfortunately, the linear method took too much time to execute that I had to halt it.

Decision tree with Entropy and min_sample_split of 6 had the best score i.e. 055. This is the classifier chosen for the final analysis.

Validation

Now that we have our classifier and data set we need to validate our algorithm. In fact, we have been doing that in each of our test cases above. Validation is important because you need to confirm or test your algorithm. You need to make sure that your algorithm is working as expected. To do this you need to use cross validation or as what we have done before, you need to have a separate training and testing data. The training data is used to training and build your classifier. The test data should be totally different data from the one used to train your algorithm. The idea is to test the performance of your algorithm with a new data set. If you just test your classifier with the same data set you used to train it, then you will end up with a biased result, which does not reflect the true performance of your algorithm.

Evaluation Metrics

As a simple test for our algorithm we used the precision and recall scores. In our case, on average we had a precision of 0.33 and a recall of 0.4. What this means is that, only 33% of those determined as POI by this algorithm are actually true POI. On the other hand, the algorithm is able to identify 40% of the actual POI.

In other words, using this algorithm we are able to identify 40% of the POI in ERON Fraud case while at the same time, 67% of those identified as POI by our algorithm are actually not person of interest.

To have a more robust result, the algorithm was tested using the test.py. The result meets the minimum requirement of 0.3. It has a Precision score of 0.36 and a Recall score of 0.31.