Project: Identify Fraud from Enron Dataset

Project Overview:

In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives.

Project Objective:

Leverage Machine learning techniques to build a person of interest (POI) identifier based on financial and email data made public as a result of the Enron scandal utilizing Udacity's hand-generated list of persons of interest in the fraud case.

Project Execution:

Following are the steps, I have taken to build my predictive model of POI identifier:

1) Enron Data exploration
2) Feature processing
3) Applying machine learning algorithms
4) Validation

Enron Data exploration

The Enron email and financial data are formatted into a dictionary, where each key-value pair in the dictionary corresponds to one person. The dictionary key is the person's name, and the value is another dictionary, which contains the names of all the features and their values for that person. The features in the data fall into three major types, namely financial features, email features and POI labels.

financial features: ['salary', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options', 'other', 'long_term_incentive', 'restricted_stock', 'director_fees'] (all units are in US dollars)

email features: ['to_messages', 'email_address', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi'] (units are generally number of emails messages; notable exception is 'email_address', which is a text string)

POI label: [‘poi’] (boolean, represented as integer)

Following is the summary of the data set:

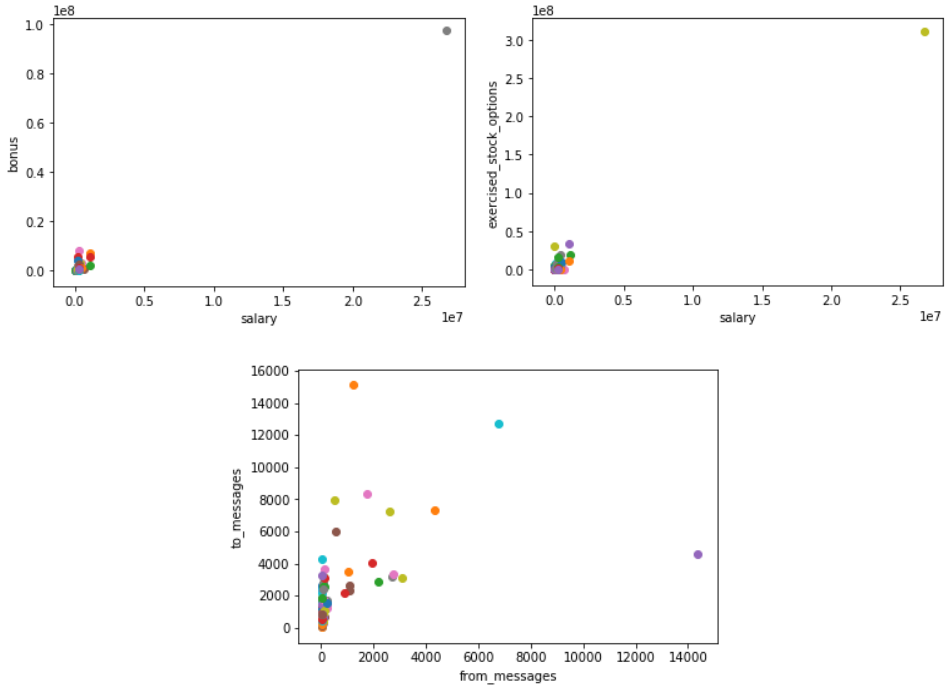
- Number of people (data points) in the dataset: **146**
- Number of features available for each person: **21 (14 financial, 6 email and 1 label)**
- Number of POIs in the dataset: **18**
- When a feature doesn’t have a well-defined value, it is represented as ‘NaN’
- Through research of Enron data set, following people held key position in Enron: **Jeff Skilling, Ken Lay** and **Andy Fastow**

The next step is to identify and clean away outliers.

Identifying and Eliminating Outliers

To identify outliers, I have chosen five key features (from both financial and email features) and plotted the following scatter plots:

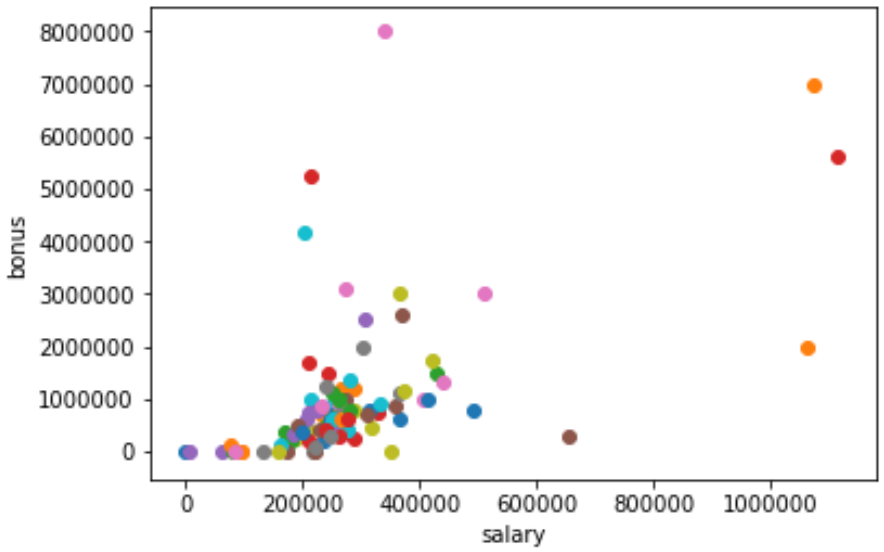
- Salary vs Bonus
- Salary vs Exercised Stock options
- from_messages vs to_messages



By exploring Salary, Bonus and Exercised stock options, it is clear that there is one outlier which is “TOTAL”.

By exploring from_messages and to_messages, there are 3 outliers. But, further exploring the data, I have decided not to exclude them since they are actual persons and removing them could affect the outcome of our predictive model.

After removing TOTAL key from the dictionary set, I further plotted the data for Salary vs Bonus. Following was the plot.



Even though it looks there might be some outliers, further exploring the data, I have decided not to remove them because those outliers include key POIs like Ken Lay and Jeff Schilling.

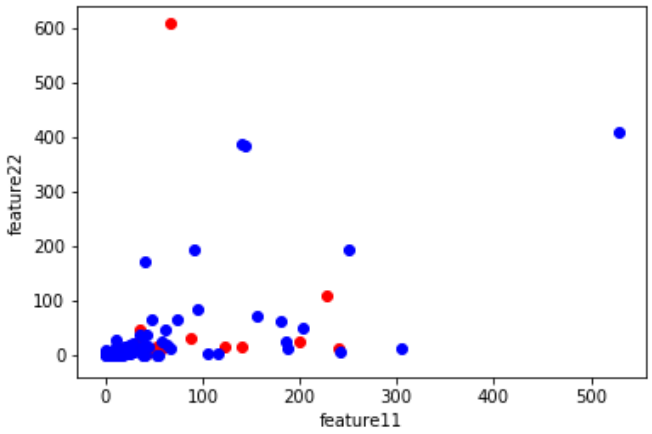
Feature Processing

After cleaning the data, the next step involves feature processing wherein we try to add new relevant features in to the data set and remove any unwanted features (like email address and poi label) from the feature list where we are sure they won’t have any influence. Finally, we find k best features that are most powerful.

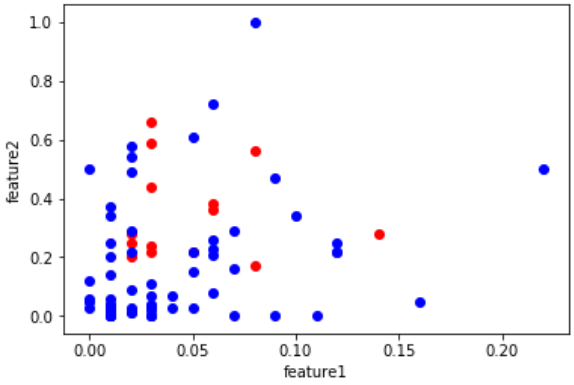
New features

fraction_from_poi_feature and **fraction_to_poi_feature**

I was trying to understand whether a pattern exists between **from_poi_to_this_person** and **from_this_person_to_poi.** Plotting these 2 features shows the below graph, but it doesn’t bring any clear pattern to illustrate POI and Non-POI differences.



So, the next step taken was to extract features **fraction_from_poi_feature** (from_messages/from_this_person_to_poi) and **fraction_to_poi_feature** (to_messages/from_poi_to_this_person) which might give a better picture to see if a trend exists between POI and Non-POI. Interestingly, it was found that parties with approximately less than 0.2 fraction_to_poi_feature were all Non-POIs. So, these features were added in to the feature list.



I also tried to investigate few other relations from financial_features but couldn't find a good pattern to be included as a new feature.

Feature Scaling and Reduction

For feature reduction, I tried my hands on two techniques: SelectKBest and PCA. Prior to Feature reduction, I also applied the Scaling technique using MinMax Scaler to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges.

Comparing PCA and Select KBest, I got better results for Select KBest and finalized on SelectKBest as my feature reduction technique. I also tried various values for k and finally decided that k=15 is the best value that will suit my algorithm for the best fit. The top15 features along with their scores are listed below:

Feature	**Score**
exercised_stock_options	25.097
total_stock_value	24.467
bonus	21.06

salary	18.575
fraction_to_poi	16.813
deferred_income	11.595
long_term_incentive	10.072
restricted_stock	9.346
total_payments	8.866
shared_receipt_with_poi	8.746
loan_advances	7.242
expenses	6.234
from_poi_to_this_person	5.344
other	4.204
fraction_from_poi	3.442

From the above list of top 15 features, it can be seen that the newly created features - *fraction_to_poi* and *fraction_from_poi* also take a spot. *fraction_to_poi* comes in the 5th position and *fraction_from_poi* takes the 15th position.

Applying Machine Learning algorithms

I used the following algorithms on the data set: Decision Tree, Random Forest and SVM algorithms. By just using the default values, none of the algorithms gave me a recall value close to 0.3 even when accuracy and precision values were high.

I used the Pipeline technique wherein I transformed the data using MinMax and SelectKBest. Finally, I used my 3 different algorithms to implement fit.

Validation and Tuning

I utilized the tester.py file already provided for the evaluation of my algorithm which splits the entire data in to test and training set. In this project, test_classifier validation method within tester.py is utilized. test_classifier method uses Stratified ShuffleSplit cross-validator. A test set will held out for final evaluation, but the validation set is no longer needed when doing Cross validation (CV). In the basic approach, called k-fold CV, the training set is split into k smaller sets (other approaches are described below, but generally follow the same principles). The following procedure is followed for each of the k “folds”:

- A model is trained using $k - 1$ of the folds as training data;
- the resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).

The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop.

In this project, the k value is set as 1000, i.e. the number of reshuffling and splitting iterations will be done 1000 times.

This approach is especially useful for this project since there is not enough data to split in to the traditional training and testing data set.

It fits the classifier using the training set and tests on test set. Following parameters are calculated: Accuracy, Precision and Recall.

Validation is the process of validating the machine learning algorithm. In this process, the entire data is split in to training and testing features. All the fit, transform and training operation are done on the training features. For the testing feature, only a transformation (similar to the transformation done on training data) is done and then the next step is to predict the labels or values for the test features.

The above feature gives an estimate of performance on an independent dataset. It also serves as a check on overfitting.

Parameter tuning for the various algorithms is the problem of choosing a set of optimal parameters so as to get best desired results (i.e. a desired Accuracy or Precision or Recall). For example, in a SVM machine learning algorithm, the parameters to be tuned are C, gamma and kernel in order to achieve the best desired results.

Precision defines what portion of positive identifications was actually correct, i.e.

Precision = True Positives / (True Positives + False Positives)

In this project, Precision defines what portion of positive POIs or non-POIs was actually correct.

Recall defines what portion of actual positives was identified correctly, i.e.

Recall = True Positives / (True Positives + False Negatives)

In this project, Recall defines what portion of actual POIs or non-POIs was identified correctly.

Accuracy is the fraction of prediction model got right, i.e.

Accuracy = Number of correct predictions / Total number of predictions

For Decision Tree algorithm, following were the best metrics obtained with the tuning parameters listed below:

DecisionTree(max_depth=100, min_samples_split=5, min_samples_leaf=1) SelectKBest(k=15)	Accuracy	Precision	Recall
	0.8295	0.343	0.303

To arrive at the above result, following were the steps taken:

- Start with default value of k
- Try different values for tuning parameters (max_depth, min_samples_split, min_samples_leaf, etc.) and see the best values for Precision and Recall
- Change value of k and repeat the previous step.

For default value of k i.e. k=10, the best result for Precision and Recall obtained after parameter tuning was 0.31 and 0.29 respectively. For another tried value of k=8, best result for Precision and Recall results were 0.27 and 0.23 respectively. I tried various value of k ranging from 4 to 20 and the best value obtained was for k=15.

For Random Forest, following were the best metrics obtained with the tuning parameters listed below:

RandomForest(max_features=0.8, n_estimators=5, min_samples_leaf=1) SelectKBest(k=14)	Accuracy	Precision	Recall
	0.842	0.359	0.234

For SVM, following were the best metrics obtained with the tuning parameters listed below:

SVM(C=1, gamma=10, kernel= 'poly') SelectKBest(k=14)	Accuracy	Precision	Recall
	0.81	0.294	0.299

From the above tables, only Decision Tree has met the desired parameter value of 0.3 for both Precision and Recall. Definitely, adding the new features helped in achieving the desired results for Decision Tree since the top 15 features were considered for algorithm calculation.

Challenges and Conclusion

Following were some of the major challenges I encountered in completing this project:

- Tuning of parameters using GridSearchCV took more time than anticipated. At times, it took more than 3 hours and other times, I had to manually restart the kernel again and provide different parameter values
- I had to try multiple tuning values to get at least one of my 3 algorithms to get the desired value range

I feel the reason for the relative low value of Precision and Recall value is due to sparse data set that was available with very few POIs.

Reference:

https://stats.stackexchange.com/questions/144439/applying-pca-to-test-data-for-classification-purposes

http://scikit-learn.org/stable/tutorial/statistical_inference/putting_together.html

https://www.analyticsvidhya.com/blog/2015/06/tuning-random-forest-model/

https://www.civisanalytics.com/blog/workflows-in-python-using-pipeline-and-gridsearchcv-for-more-compact-and-comprehensive-code/

https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall