C753-Machine Learning - Final Project

Enron Fraud - Finding POI (Person(s) Of Interest)

Maya Mathews
STUDENT ID 001075274
Program: BSDMDA (January 1, 2019)
My Mentor: Cora Peterson
Email: mmat100@wgu.edu

Summary

The goal of this project is to identify the POI (persons of interest) from the Enron data provided for analysis. The data used is public Enron financial and email dataset.
Starter code: `https://github.com/udacity/ud120-projects.git`

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. I have attempted to play detective in this project to build a person of interest identifier based on financial and email data made public as a result of the enron scandal. The data has already been combined with a hand-generated list of persons of interest in the fraud case, which means individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

I have used part of the data to train the classifier algorithm and test it with the few select data points. I used a few different algorithms to evaluate the best choice as a regular decision tree.

Machine Learning is best suited to build a model to identlfy data points that can be classified as person of interest or not. With more data points the training fit would be better tuned to predict with higher accuracy levels. In the data we were given there are only 18 persons of interest included even though there are over 35 persons of interest according to the financial data available.

The only outlier that was discarded was the `TOTAL`. Since our focus is the outliers in the data, we decided to keep the rest of the data points.

Python

I used Python 3.8.5 (default, Sep 3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)] for my project.
I had to change a few of the starter scripts to work with this version.
Here is the link to my github repository for this project.
https://github.com/4makrim/ud120-projects.git

Explore the data

Some key features we learn on a quick explore of the dataset are the number of data points, number of features on each data point, correlation between a few key variables. Listed below are some findings.

- There are 146 individuals whose email data we are looking through. We have some missing information like salary which we have only for 95 of the 146 individuals.
- We have only 18 POIs to train on with the enron data. The financial information shows there are 35 POIs.
- There are 21 features identified with the enron data. Of these we will use the salary, bonus, total_payments, for our analysis.

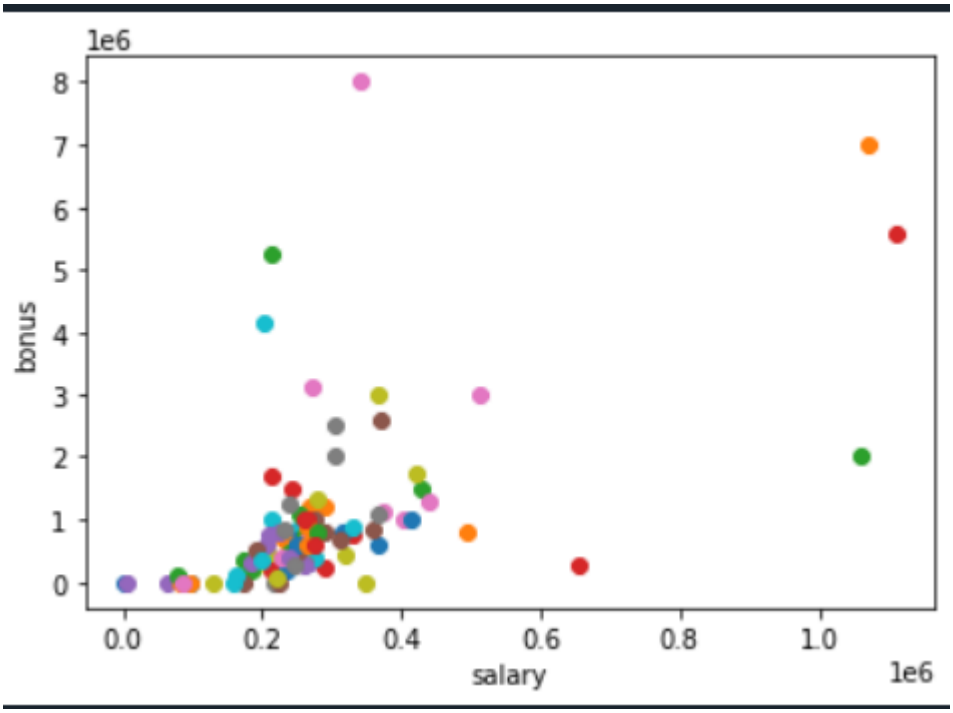
```
Total no. of individual accounts:  146
No. of valid salary entries:  95
No. of valid email IDs we have data for:  111
How many people received a total payout? :  21
Total no. of POI:  18
Percentage of people who received payout:  14.383561643835616
Percentage of POIs that have received total payments:  100.0
Percentage of POIs that have no total payments:  0.0
```

- Two main features we will investigate further are salary and exercised_stock_options

```
Minimum exercised stock options:  3,285.00
Maximum exercised stock options:  34,348,384.00
Minimum salary:  477.00
Maximum salary:  1,111,258.00
```

Features

Some of the features that we started with were identified in the mini projects. We identified the outliers by plotting `salary` against `bonus`.



The outliers were
LAVORATO JOHN J Bonus: $8,000,000.00
LAY KENNETH L Salary: $1,072,321.00
LAY KENNETH L Bonus: $7,000,000.00
BELDEN TIMOTHY N Bonus: $5,249,999.00
SKILLING JEFFREY K Salary: $1,111,258.00
SKILLING JEFFREY K Bonus: $5,600,000.00
FREVERT MARK A Salary: $1,060,932.00
as outlined by the script `final_project/enron_outliers.py`

The upper limit for `salary` was set at one million and `bonus` at five million in order to classify the outliers.
Calculating the regression of `bonus` and `long_term_incentive` showed more correlation

```
slope: [1.19214699]
intercept:  554478.7562150091
##### stats on test dataset for bonus vs long_term_incentive #######
r-squared score: -0.5927128999498643
##### stats on training dataset for bonus vs long_term_incentive  #######
r-squared score: 0.21708597125777662
```

than `salary` and `bonus`

```
slope: [0.00835316]
intercept:  246183.2051581233
##### stats on test dataset #######
```

```
r-squared score: -0.148291297918127
##### stats on training dataset #######
r-squared score: 0.04550919269952447
```

Thus from the initial exploration and the results from the mini projects, I chose to use the `salary`, `exercised_stock_options`, `bonus`, `total_payments` as the features to associate with POI.
Using the feature selection mini project, we identified salary, exercised_stock_options, bonus as the most significant features.
Here are the importances we calculated.

```
importance: 0.37446157808832065
importance: 0.20297965116279068
importance: 0.2297535381907492
```

From our initial exploration it must be noted there is a huge variance between the minimum and maximum salary as well as exercised stock options. In one of the mini projects I had used the bonus to be graphed against the salary and noticed there were a few points that looked like outliers. Mapping the data points to the list of POIs confirmed that POIs not only drew a high salary, they also received abnormal bonuses.
Total payments was added after a few trial and errors.

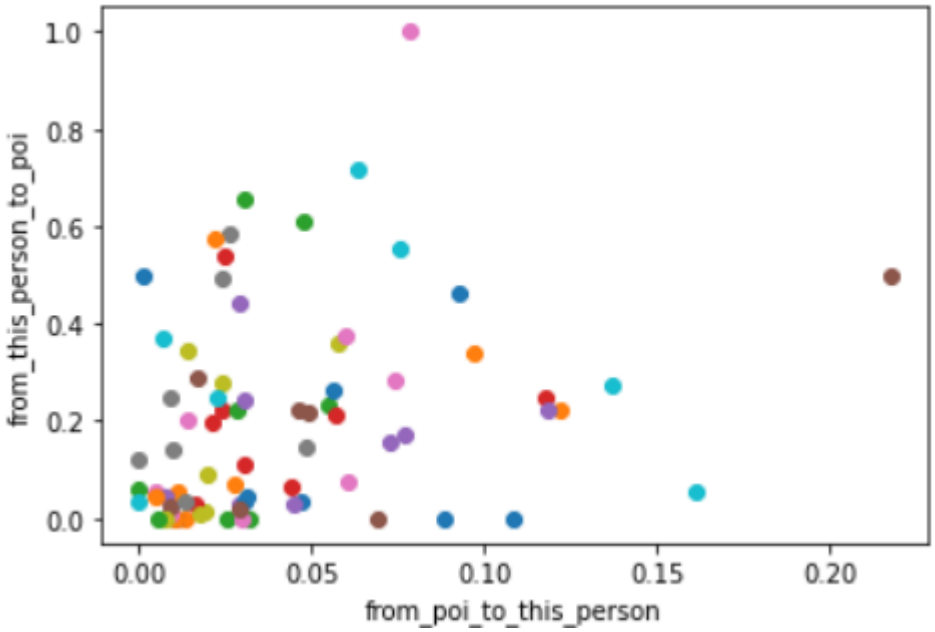
The last feature was the exrecised stock options. The minimum and maximum values suggested this could be another predictor to identify POI. This was the last feature I added to my evaluation script.

I did not do any feature scaling. The features used were very much in the same range of values.

Feature Creation

Salary and Bonus were identified as the most significant features that can predict a person of interest.

We created 2 new features in a mini project - Fraction of messages received by POI and Fraction of messages sent by POI. This was graphed. (code newDataPoint.py)



However, the information did not offer anything conclusive to help us. Thus, we did not pursue with these features further.

Algorithm Choice

I used Gaussian Naive-Bayes, SVM Decision Tree, and AdaBoost algorithms to check which one would give me the best accuracy with the data. The SVM decision tree with an accuracy of 0.689 seemed to be the best solution.

With all four features my precision and recall are pretty high using the decision tree algorithm.

```
Precision score:  1.0
Recall score:  0.3333333333333333
Accuracy:  0.9047619047619048
#Features in data:  4
```

The accuracy of the prediction using the SVM algorithm was even higher, almost 95.2%

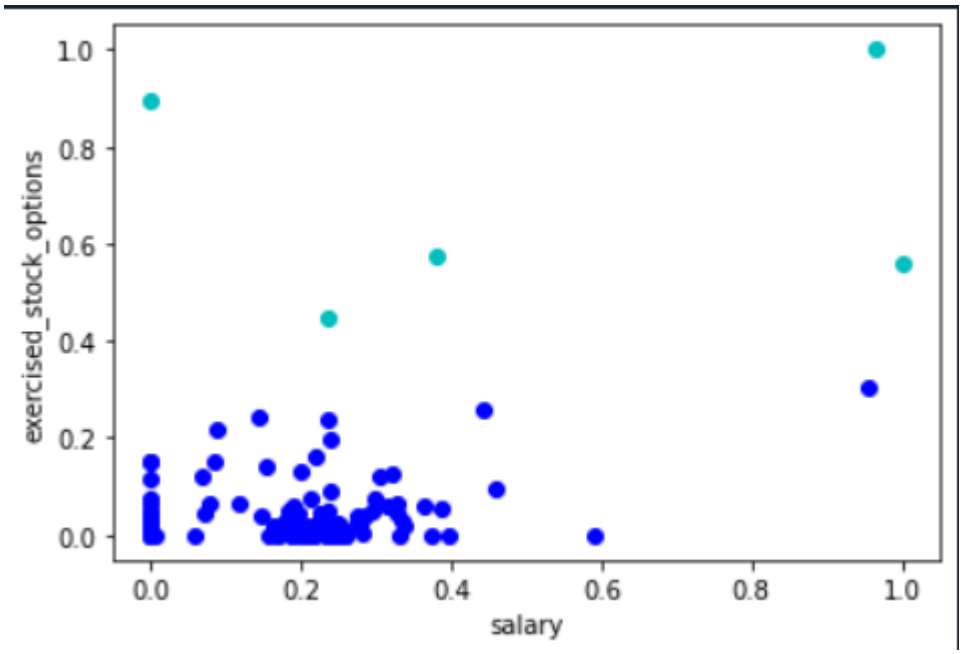
```
*** Classifier SVM ***
training time: 0.0 s
prediction time: 0.0 s
SVM Accuracy:  0.9523809523809523
```

However, the precision and recall scores could not be calculated as there were no postive predictions. This could be because we have highly skewed training data.

The final algorithm choice was a decision tree as we get the required precision and recall rates.

```
F1 Score:  0.6
Precision score:  0.75
Recall score:  0.5
Decision Tree Classifier Accuracy:  0.9047619047619048
no. features used in classification:  4
```

kmeans



There was only one outlier `TOTAL` that we eliminated from the data. When `salary` and `exercised_stock_options` seems to provide a list of 5 outliers. We decided not to eliminate the outliers in this case as these were the points that we needed to look at closer.

Validation and Evaluation

Why do we need to do validation? Validation offers an estimate on the performance of an independent dataset. We can also verify for overfitting when using training data.

We will use `GridSearchCV` which is a way of systematically working through multiple combinations of parameter tunes, cross-validating as it goes to determine which tune gives the best performance. `GridSearchCV` works through many combinations and can choose the best performance algorithm.

```
Best estimator found by grid search:
SVC(C=300, class_weight='balanced', gamma=10)
```

Decision Tree classifier with new parameters yeilds a better result

```
Precision score:  0.75
Recall score:  0.5
Accuracy:  0.9047619047619048
#Features in data:  4
```

SVM algorithm with `kernel='rbf'` provided the highest accuracy.
SVM algorithm with `kernel = 'linear'` seemed to take very long to train. This is probably what is happening with my GridSearchCV taking too long. Since our goal was to get a precision and recall above 0.3, I stopped when these numbers were reached with a simple **decision tree** with 90.5% accuracy. The precision was at 75% to 100%. Recall was between 33.3% and 50%.

```
Precision score:  1.0
Recall score:  0.3333333333333333
```

After removing the outlier `TOTAL` the metrics are even better

```
Precision score:  0.75
Recall score:  0.5
```

Evaluation Metrics

I have a **precision of 1.0** and a **recall of 0.33**. These meet the rubric requirement that these metrics should be above 0.3. My POI identifier using four features doesn't have great recall, but it does have good precision. That means that whenever a POI gets flagged in my test set, I know with a lot of confidence that it's very likely to be a real POI and not a false alarm. On the other hand, the price I pay for this is that I sometimes miss real POIs, since I'm effectively reluctant to pull the trigger on edge cases. (Credit for this interpretation: one of the lesson 14 exercise responses.)

Precision = TP / (TP + FP)
Recall = TP / (TP + FN)

The results change every time I run the poi_id.py

```
F1 Score:  0.5
Precision score:  1.0
Recall score:  0.3333333333333333
Decision Tree Classifier Accuracy:  0.9047619047619048
no. features used in classification:  4
```

Here is the result from another run even though there are no code changes.

```
F1 Score:  0.6
Precision score:  0.75
Recall score:  0.5
Decision Tree Classifier Accuracy:  0.9047619047619048
no. features used in classification:  4
```

Final Results:

```
**** Final Classifier Algorithm Chosen: Regular Decision Tree *****
training time for all data: 0.001 s
Decision Tree Accuracy on All the data:  1.0
training time: 0.001 s
prediction time: 0.0 s
no. postive predictions:  4
F1 Score:  0.6
Precision score:  0.75
Recall score:  0.5
Decision Tree Classifier Accuracy:  0.9047619047619048
no. features used in classification:  4
```

The above conclusion still holds. I have a better precision than recall. In other words, I might have to deal with a few false negatives that I will have to err on the side of grace.