Enron Submission Free-Response Questions

---

1. Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those? [relevant rubric items: “data exploration”, “outlier investigation”]

The goal of this project is to explore a dataset of publicly available email and financial information available from a failed company called Enron, which went bankrupt due to rampant fraud, and identify potential persons of interest based on this dataset. A person of interest in this case is someone who was possibly involved in the fraud and may be defined as someone who has an unusual amount of interaction with known person of interest, someone who was fined, jailed, or settled in court, or someone who was making significantly more money than others in the company.

Machine learning is useful in identifying these people as we can use patterns in the email and finance data to try and predict if someone will be a person of interest such as a great deal of contact with known persons of interest or if their financial data falls outside the scope of what we would expect, or a combination of these.

Data Analysis

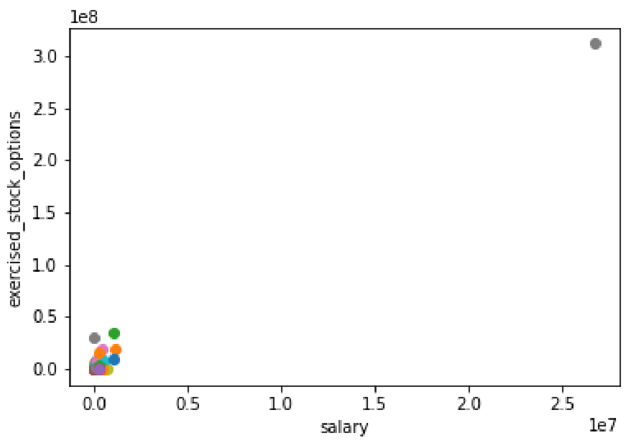
- Total Number of people in dataset: 146
- Total Number of POI in dataset: 18
- Total Number of non-POI in dataset: 128
- There are 21 total features in total, 14 financial features, 6 email features and one feature to identify subject as a poi
- Financial Features: `salary, bonus, long_term_incentive, deferred_income, deferral_payments, loan_advances, other, expenses, director_fees, total_payments, exercised_stock_options, restricted_stock, restricted_stock_deferred, total_stock_value`
- Email Features: `email_address, to_messages, from_messages, from_poi_to_this_person, from_this_person_to_poi, shared_receipt_with_poi`
- POI (true or false) Feature: `poi`

For how much data we have in each feature:

```
salary feature: 95 people or 65.07% of people
bonus feature: 82 people or 56.16% of people
loan_advances feature: 4 people or 2.74% of people
deferral_payments feature: 39 people or 26.71% of people
deferred_income feature: 49 people or 33.56% of people
expenses feature: 95 people or 65.07% of people
director_fees feature: 17 people or 11.64% of people
other feature: 93 people or 63.7% of people
total_payments feature: 125 people or 85.62% of people
long_term_incentive feature: 66 people or 45.21% of people
restricted_stock_deferred feature: 18 people or 12.33% of people
exercised_stock_options feature: 102 people or 69.86% of people
restricted_stock feature: 110 people or 75.34% of people
total_stock_value feature: 126 people or 86.3% of people
email feature: 111 people or 76.03% of people
to_messages feature: 86 people or 58.9% of people
from_messages feature: 86 people or 58.9% of people
from_poi_to_this_person feature: 86 people or 58.9% of people
from_this_person_to_poi feature: 86 people or 58.9% of people
shared_receipt_with_poi feature: 86 people or 58.9% of people
poi feature: 146 people or 100.0% of people
```

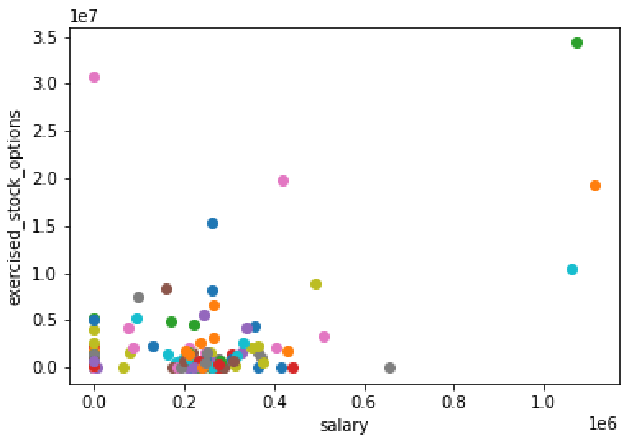
Outliers

In our financial data we see there is a massive outlier as can be seen in the graphic below:



This is due to the fact that our spreadsheet has a section totaling all of the financial data and this was included as a data point named TOTAL

After removing this outlier using the python dictionary pop method we get a much more reasonable plot as seen below:



Another outlier seems to be THE TRAVEL AGENCY IN THE PARK. This only contains "other" financial data and would not be an Enron employee, and so I removed this data point as well.

After removing these outliers our dataset has changed to:

- Total Number of people in dataset: 144
- Total Number of POI in dataset: 18
- Total Number of non-POI in dataset: 126

---

2. What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that does not come ready-made in the dataset -- explain what feature you tried to make, and the rationale behind it. (You do not necessarily have to use it in the final analysis, only engineer and test it.) In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importances of the features that you use, and if you used an automated feature selection function like SelectKBest, please report the feature scores and reasons for your choice of parameter values. [relevant rubric items: “create new features”, “intelligently select features”, “properly scale features”]

I used GridSearchCV to get the SelectKBest k features to select the features to use for my POI identifier. GridSearchCV came up with only 4 features being the best score:

Here are the scores of the kbest features according to the result of the algorithm:

1. Feature: restricted_stock_deferred Score: 24.815079733218194
2. Feature: restricted_stock Score: 24.18289867856688
3. Feature: salary Score: 20.792252047181535
4. Feature: poi Score: 18.289684043404513
5. Feature: shared_receipt_with_poi Score: 16.579618415784097
6. Feature: deferral_payments Score: 11.458476579280369
7. Feature: total_payments Score: 9.922186013189823
8. Feature: exercised_stock_options Score: 9.2128106219771
9. Feature: other Score: 8.772777730091676
10. Feature: from_this_person_to_poi Score: 8.589420731682381
11. Feature: bonus Score: 7.184055658288725
12. Feature: deferred_income Score: 6.094173310638945
13. Feature: from_messages Score: 5.243449713374958
14. Feature: director_fees Score: 4.187477506995375
15. Feature: emails_to_poi_vs_total Score: 3.356757331172283
16. Feature: from_poi_to_this_person Score: 2.382612108227674
17. Feature: expenses Score: 2.1263278020077054
18. Feature: total_stock_value Score: 1.6463411294420076
19. Feature: loan_advances Score: 0.2246112747360099
20. Feature: to_messages Score: 0.16970094762175533
21. Feature: long_term_incentive Score: 0.06549965290994214

Using GridSearchCV I found the optimal k value for kbest the scores of which are listed below:

For k = 1 the score is: 0.8466666666666665 For k = 2 the score is: 0.8399999999999999 For k = 3 the score is: 0.8257142857142856 For k = 4 the score is: 0.8533333333333332 For k = 5 the score is: 0.8466666666666665 For k = 6 the score is: 0.8466666666666665 For k = 7 the score is: 0.8395238095238093 For k = 8 the score is: 0.8395238095238093 For k = 9 the score is: 0.8257142857142856 For k = 10 the score is: 0.8466666666666667 For k = 11 the score is: 0.8395238095238093 For k = 12 the score is: 0.8395238095238093 For k = 13 the score is: 0.8466666666666667 For k = 14 the score is: 0.8252380952380951 For k = 15 the score is: 0.8109523809523809 For k = 16 the score is: 0.8109523809523809 For k = 17 the score is: 0.8109523809523809 For k = 18 the score is: 0.768095238095238 For k = 19 the score is: 0.7485714285714284 For k = 20 the score is: 0.6519047619047619 For k = 21 the score is: 0.511904761904762

k=4 gives the highest value and so is the optimal k value, and so I then reduced the features to the 4.

I did scale my features as we are dealing with some very large numbers in our financial data and it might not scale well with other numerical data. I used MinMaxScaler in sklearn to do this.

I added two new features; emails_to_poi_vs_total which calculates to two decimal points the amount of emails to a person of interest from each each person as a fraction of their total sent emails and emails_from_poi_vs_total which calculates to two decimal points the amount of emails from a person of interest to each person as a fraction of their total received emails. A trend in these new features would tell us if someone had a lot of communication with a person of interest making them a potential person of interest themselves. These new features did not quite make the cut for the kbest.

3. What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms? [relevant rubric item: “pick an algorithm”]

The algorithms I tried and their scoring are listed below:

---

1. **Naive Bayes Scores**

```
GaussianNB()
    Accuracy: 0.73900   Precision: 0.22604  Recall: 0.39500 F1: 0.28753 F2: 0.34363
    Total predictions: 15000    True positives:  790    False positives: 2705   False negatives: 1210   True
negatives: 10295
```

Naive Bayes has good scores here, the recall and accuracy are fairly high but since there are no tuning parameters for this algorithm this is as good as it gets. We need at least a .3 score in precision so our search continues.

---

1. **Decision Tree Scores**

```
DecisionTreeClassifier()
    Accuracy: 0.79633   Precision: 0.22879  Recall: 0.22250 F1: 0.22560 F2: 0.22373
    Total predictions: 15000    True positives:  445    False positives: 1500   False negatives: 1555   True
negatives: 11500
```

The Decision Tree is fairly promising with a balanced precision and recall that we might be able to bring up together. Other algorithms performed better as far as their F1 Scores and accuracy, but this is still a contender.

---

1. **SVM/SVC Scores**

```
SVC()
    Accuracy: 0.86267   Precision: 0.30000  Recall: 0.02250 F1: 0.04186 F2: 0.02761
    Total predictions: 15000    True positives:   45    False positives:  105   False negatives: 1955   True
negatives: 12895
```

SVC is looking good as far as precision and accuracy go but the recall might be too far gone to save at .02. This was confirmed with my GridSearchCV which only managed to bring the recall up to 0.05800.

---

1. **KNearestNeighbors Scores**

```
KNeighborsClassifier()
    Accuracy: 0.87920   Precision: 0.65461  Recall: 0.19900 F1: 0.30521 F2: 0.23118
    Total predictions: 15000    True positives:  398    False positives:  210   False negatives: 1602   True
negatives: 12790
```

KNearestNeighbors looks very promising to me with a great accuracy and precision score with potential wiggle room to bring up the recall, we will definitely be trying to tune this one.

---

1. **Random Forest Score**

```
RandomForest Scores
RandomForestClassifier()
    Accuracy: 0.86027   Precision: 0.42027  Recall: 0.12650 F1: 0.19447 F2: 0.14706
    Total predictions: 15000    True positives:  253    False positives:  349   False negatives: 1747   True
negatives: 12651
```

The RandomForestClassifier is looking a little underwhelming as far as our metrics go, and indeed after tuning it with GridSearchCV I could only get the recall to 0.12650, and so I scrapped it.

We see here that KNearestNeighbors and DecisionTreeClassifier were our top performers and so I chose to fine tune them both and see if I could get them to be a bit more balanced. We see here that KNearest Neighbors had a higher precision but a lower recall score than the Decision Tree and so is more likely to be accurate in detecting true positive cases. Our Decision Tree is very balanced looking and might be more likely to make the cut as there is not a large gap.

After fine tuning, the algorithm I chose was the DecisionTreeClassifier because after I optimized KNearestNeighbors I still found that I couldn't bring the precision above .276. The DecisionTreeClassifier though, after fine tuning got up to 0.37416 in precision and 0.33600 in recall.

---

4. What does it mean to tune the parameters of an algorithm, and what can happen if you don't do this well? How did you tune the parameters of your particular algorithm? What parameters did you tune? (Some algorithms do not have parameters that you need to tune -- if this is the case for the one you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier). [relevant rubric items: "discuss parameter tuning", "tune the algorithm"]

Tuning the parameters of the algorithm is finding the ideal values for each parameter in the algorithm to get the best model possible. This is important as what you put into the parameters can greatly affect the result of your algorithm. For example with clustering algorithms, if you know how many groups you are lookng to classify, you can define how many clusters you are specifically looking for in the algorithm parameters which can help the algorithm give you a desirable result. I used GridSearchCV to test all of the algorithms above. The parameters I tested in my final algorithm and altered based on GridSearchCV's results were as follows:

DecisionTreeClassifier GridSearchCV result:

```
{'criterion': 'gini',
 'max_features': 9,
 'min_samples_leaf': 1,
 'min_samples_split': 13,
 'random_state': 8,
 'splitter': 'best'}
```

Which resulted in these scores:

DecisionTreeClassifier(max_features=9, min_samples_split=13, random_state=8) Accuracy: 0.83653 Precision: 0.37416 Recall: 0.33600 F1: 0.35406 F2: 0.34300 Total predictions: 15000 True positives: 672 False positives: 1124 False negatives: 1328 True negatives: 11876

So we see after tuning the algorithm the results of our metric scores of accuracy, precision and recall improved very much.

---

5. What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis? [relevant rubric items: "discuss validation", "validation strategy"]

Validation is when you try your testing new data that you set aside, that your algorithm has not yet seen, on your trained algorithm. You validate your model by using data that you have not trained your model in to see how it does with new data. A classic mistake would be to use the same data to test and train your algorithm. This would likely result in you having much higher scores than might be realistic if your algorithm was dealing with new, never-before-seen data, instead of the data it has already been given the classification for. This gives you a false sense of confidence in the accuracy and overall performance of your model. I validated my model by cross validation with GridSearchCV with a test size of 30% of my data and a training size for 70% of my data. I validated using the provided testing_classifier function which used cross validation with sklearns StratifiedKFold.

---

6. Give at least 2 evaluation metrics and your average performance for each of them. Explain an interpretation of your metrics that says something human-understandable about your algorithm's performance. [relevant rubric item: "usage of evaluation metrics"]

Recall is the fraction of correctly identified actual positive data points and is useful to tell you if your algorithm is good at identifying true positives in your data. High recall does not care if you have some false positives in there, and so it is better to have high recall if you mostly care about finding positives at the risk of having false positives hanging around. High recall would be useful if you were making something that could detect bombs for example. You would rather this technology be skewed toward detecting all positives even if it incorrectly identified something as being a bomb here and there. The Recall score of my model was 0.33600. Precision tells us the fraction of how many classifications were actually correct of all of the points predicted as correct. This is a useful metric top let you know that your algorithm is more certain of it's positive cases. You would want to have high precision in the case of something like a machine that detected whether it was a good day to launch a weather balloon. If the machine falsely predicts it's a bad day here and there, it's no big deal, but it needs to be as close to 100% certain as it can be when it predicts that it's a good day to launch one or it will have been a useless venture. The precision of my final model was 0.37416.

In this case the precision and recall of the model are fairly similar and so this algoritm is just about as good at detecting all positives as it is being certain of it's results. Both metrics are still fairly low though, and probably could be improved by someone with more experience or by gathering more data or manipulating it in a different way.