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 primary goal of the project is to determine if there is a pattern to the people of interest (POI) in the enron case, so that you could see if any new POIs exist that were not specifically named in the case based on the email & financial data involved.

The data set was a combination of those factors, which was organized into a dictionary with the person as the primary key, and a set of attributes for them. Included in the data was a ‘Total’ line from the base spreadsheet of financial data that had to be removed as it was skewing the data. The data is also limited due the the small number of employees and POIs.

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”]

The dataset had 146 entries (one of which was a total row that was later removed as an outlier), 18 of those being POIs. Each of these had 21 features initially so there was a lot of width to work with, but not a lot of examples as enron likely had many thousands of employees.

To start, I created two ratios of emails, one of emails to the POI from the person and one from the POI to the person in question, which were normalized against the total to and from emails, as there was a strong correlation that had to be normalized (ie the more you make the more email you send, as more highly placed members of organizations tend to get more email). This was done so that they values could be used.

I did end up using one of the two email ratios (the ratio of the number of emails from other poi’s to the individual as a proportion of their total emails). Including it increased the precision and recall (from about .2 to .4).

The ‘email_to_poi’ helped to balance the precision and recall. With it included:
Accuracy: 0.81269 Precision: 0.38654 Recall: 0.37050

Without it:
Accuracy: 0.80131	Precision: 0.36410	Recall: 0.39050

The ‘email_from_poi’l feature did not seem to improve the stats, so it was removed. With it included:
Accuracy: 0.80538	Precision: 0.36038	Recall: 0.34200

With it removed:
Accuracy: 0.81269	Precision: 0.38654	Recall: 0.37050

I ultimately left it out of the final version as it was only a 1% increase which didn’t seem to justify the additional complexity.

Next, I had to try a large number of feature combinations, initially starting with just POI and salary and bonus, and adding and removing features to tune the algorithm. I also used the GGPairs module in R to do a correlation matrix between all features, which served as a starting point, however since many of the features tended to contribute to underlying ideas like compensation, I manually tried to add and remove features to get the best result.

I used SelectKBest as a starting point, which pointed to bonus, exercized_stock_value and salary (which scored much higher than the other features.

```
features_list = ['poi','bonus','total_stock_value','exercised_stock_options',
                 'pct_to_poi','salary','to_messages','deferral_payments', 'total_payments',
                 'restricted_stock','shared_receipt_with_poi', 'restricted_stock_deferred',
                 'expenses','from_messages','other','from_this_person_to_poi',
                 'director_fees','deferred_income','long_term_incentive','from_poi_to_this_person']
```

```
[21.06000171 24.46765405 25.09754153  3.21076192 18.57570327  1.69882435
  0.21705893  8.86672154  9.34670079  8.74648553  0.06498431  6.23420114
  0.1641645   4.20497086  2.42650813  2.10765594 11.59554766 10.07245453
  5.34494152]
```

The salary feature produced pretty low evaluation metrics:
Precision: 0.22803	Recall: 0.19850

So salary was not as a good an indicator as the stock options and bonus were. Long term incentives seemed promising, but in the final algorithm it didn’t make a huge difference on the results.

Including 'deferred_income' and 'long_term_incentive' seemed to drive down the scores of the model:

Accuracy: 0.80836   Precision: 0.30890   Recall: 0.27600

Without them:
Accuracy: 0.81138   Precision: 0.38398   Recall: 0.37400

I did try feature scaling while testing, but the result didn't seem to increase the performance of the algorithm. In the end, a smaller feature set worked much better than a larger one. The final features after manual and software based tuning were:
['poi','bonus','total_stock_value','exercised_stock_options','pct_to_poi']

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"]

This was the most annoying part of the process. I tried about 5-10 different options, including SVC, 2 types of naive bayes (Gaussian and Multivariable), Random Forrest & KMeans, and a few other. After much testing, changing of included features etc I got the best results from using the Tree Classifier. Gaussian NB also worked quite well, but and no options for further tuning and the tuned Tree Classifier out performed it.

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"]

If an algorithm was a physical machine, they would be the knobs and dials you would use to change its behavior. It's critical to tune the classifiers (and machine learning in general) to get the best output. If you don't they can be less efficient (a poorly tuned SVC cause my pc to run for hours with no extra value) and they can be less accurate.

I needed to tune the max depth of the tree, the criterion (aka the method being used to determine entropy) and minimum number of samples to split on. I used the GridSerachCV module to search through a number of different combinations to find the best one.

I tuned the algorithm:

```
tree = DecisionTreeClassifier()
parameters =
{'criterion':['gini','entropy'],'max_depth':[4,5,6,7,8,9,10,20,50,100,150],'min_samples_split':[2,3,4,
5,6]}
```

```
clf = GridSearchCV(tree, parameters)
```

Which provided the results:

```
{'min_samples_split': 2, 'criterion': 'entropy', 'max_depth': 10}
```

I incorporated this into the final code, since running the tuning logic repeatedly w/ the test program caused overly long run times due to the number of iterations in the validator. You would likely not have this issue when running the code in a production environment and could replace the code with the GridSerachCV version.

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 determining if your software is producing the output you expect. The mistake mentioned in the course a number of times is using your training data for validation, rather than using a test set. Small test sets often have this problem, which is why randoming the data (by bucketizing it and then randomly picking a test set) can be a good option with a very small data set like the one that we have.

In the context of this particular evaluation, a positive outcome meant that someone was a POI, a negative that they were not. Precision is the likelihood that a predicted POI is a POI (true positives / true positives and false positives) and Recall that a person predicted to be a POI was a POI.

My strategy was to balance precision and recall, since both are problematic in a case where you could potentially implicate an innocent person or ignore a guilty one. The overall numbers seem pretty low compared to the test exercises at 36%.

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"]

As mentioned above, precision and recall were the primary metrics I considered. Accuracy is unhelpful in this case as most people are not POIs, so a high accuracy could be obtained by considering everyone innocent in the fraud.

Final metrics of my testing:

```
Accuracy: 0.80285      Precision: 0.36044      Recall: 0.36350
```

A multifold validation was used (essentially spilling all data into a number of subsets and then treating one subset as the test data and re-running the testing for all possible training and test sets). This was done because the number of people is small, and this method is excellent in this case as it maximizes the value that can be obtained from the data. To accomplish this the StratifiedShuffleSplit module of sklearn was used, which is a k-fold validation & randomizer.

Resources Used

https://stackoverflow.com/ - Various technical question on python and sklearn
http://scikit-learn.org/ - Documentation of SKlearn