Enron Submission Free-Response Questions

A critical part of machine learning is making sense of your analysis process and communicating it to others. The questions below will help us understand your decision-making process and allow us to give feedback on your project. Please answer each question; your answers should be about 1-2 paragraphs per question. If you find yourself writing much more than that, take a step back and see if you can simplify your response!

When your evaluator looks at your responses, he or she will use a specific list of rubric items to assess your answers. Here is the link to that rubric: [Link] Each question has one or more specific rubric items associated with it, so before you submit an answer, take a look at that part of the rubric. If your response does not meet expectations for all rubric points, you will be asked to revise and resubmit your project. Make sure that your responses are detailed enough that the evaluator will be able to understand the steps you took and your thought processes as you went through the data analysis.

Once you've submitted your responses, your coach will take a look and may ask a few more focused follow-up questions on one or more of your answers.

We can't wait to see what you've put together for this project!

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"]

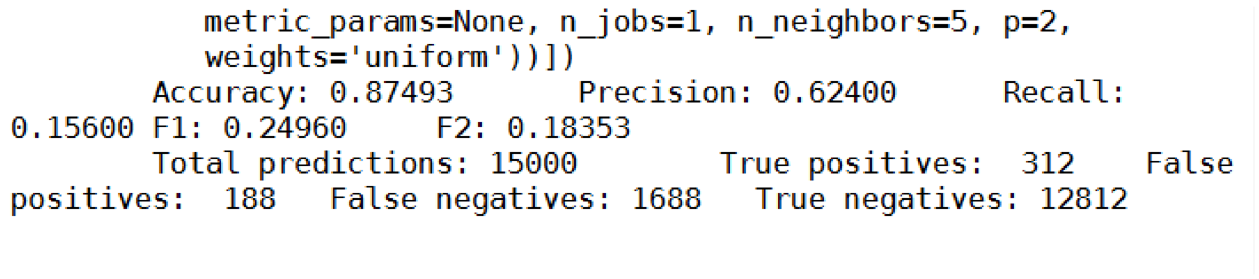
   The goal of this project was to demonstrate our knowledge of machine learning by developing and tuning a classifier that can take in known people of interest who may have committed fraud at Enron and applying that classifier to a larger dataset of people to flag anyone else who might have characteristics (features) that might indicate they are also a person of interest for fraud. Machine learning is useful in accomplishing this because there are so many features for each person and so much data that trying to analyze this data and draw conclusions from it manually would be an absolute nightmare if not impossible. I don't even know how this problem could be approached without using machine learning to generalize to new data using known features and labels from existing data.

   There were 146 people in the dataset, and 18 of them were persons of interest. These persons of interest were also the test labels that the algorithms were trained on to see if the features of these 18 were similar to the features of any of the other 128 people. There were 20 features in the dataset and 1 label used to train the algorithms, and during my analysis I used some approaches to find which of these features had the most predictive power for finding a POI. In the end, I ended up using 6 of these features that I found to have strong predicative power. There were outliers in the data, but I tried to keep as many of the 146 data points as I could since that was such a small sample size. I only ended up removing two data points: one named "TOTAL" and one named "THE TRAVEL AGENCY IN THE PARK" since the first was an aggregate of all other data points and I have no idea what the other was, but it was not a persons name. When I thought about cleaning 5% or 10% of the other outliers, my Udacity mentor suggested against it since the data set was so small already. I used data_dict.pop('key') to clean these up.

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 ended up using the following six features: 'salary', 'total_payments', 'bonus', 'total_stock_value', 'exercised_stock_options', and 'shared_receipt_with_poi'. In order to pick these, I took the original features list that I thought would be intuitively interesting to look at and excluded those features which were missing over 55% of their data (i.e., there were NaNs for more than 55% of the data).

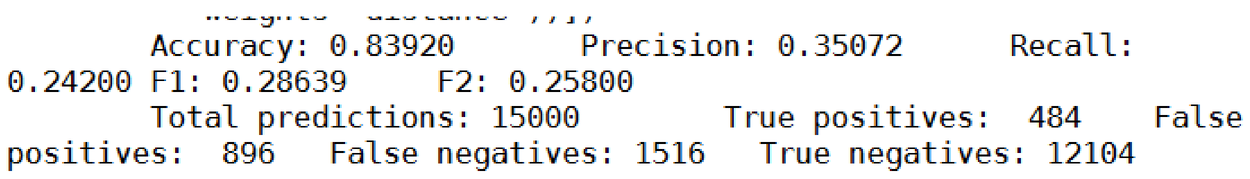
The Udacity reviewer asked me to create a more exhaustive feature selection process. The reviewer asked me to test other combinations of missing data values to see what gives the best performance. Beyond excluding the features which were missing over 55% of their data, I tested my algorithm on those features that were missing greater than or equal to 40% of their features. This can be seen in features_list_v3 in my code in poi_id.py. When I reran the algorithm this was my result:



```
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'))])
        Accuracy: 0.87493       Precision: 0.62400      Recall: 
0.15600 F1: 0.24960     F2: 0.18353
        Total predictions: 15000        True positives:  312    False 
positives:  188   False negatives: 1688   True negatives: 12812
```

Clearly, this does not satisfy the project requirements.

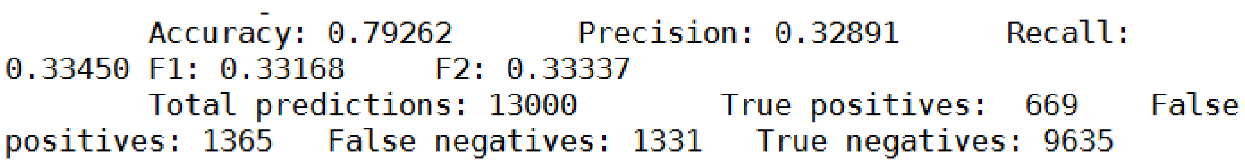
I also reran the algorithm by excluding features that were missing more than 20% of their data. This only left me with two features: “total_payments” and “total_stock_value”. Please see the results of this below:



```
        Accuracy: 0.83920       Precision: 0.35072      Recall: 
0.24200 F1: 0.28639     F2: 0.25800
        Total predictions: 15000        True positives:  484    False 
positives:  896   False negatives: 1516   True negatives: 12104
```

This does not satisfy the project requirements either, so I am sticking with my original exclusion of those missing over 55% of their data.

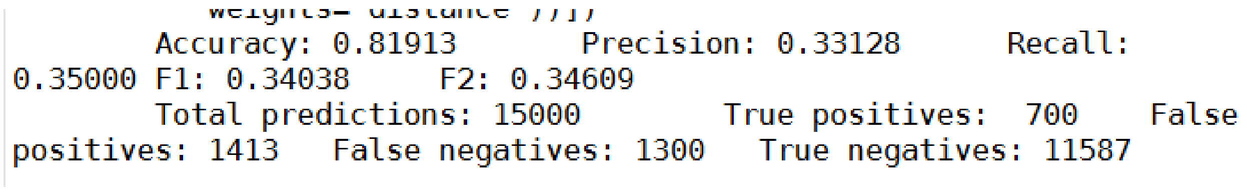
Then I originally used SelectKBest to find those with feature scores of over 8 and focused on those for the rest of my analysis by removing everything else. However, the Udacity reviewer suggested that I make my feature scoring more exhaustive by trying different thresholds. Looking at my feature score, I additionally tried the thresholds of 18 and 2 as those looked to be interesting breaking points. When I ran my algorithm with a threshold of 18, this was my result:



```
        Accuracy: 0.79262       Precision: 0.32891      Recall: 
0.33450 F1: 0.33168     F2: 0.33337
        Total predictions: 13000        True positives:  669    False 
positives: 1365   False negatives: 1331   True negatives: 9635
```

This is actually a really good result and while it does meet the project requirements, it has a slightly lower precision and recall then my original.

Then I looked at these score using a threshold of 2 and this was my result:



```
            weights='distance')))
        Accuracy: 0.81913       Precision: 0.33128      Recall:
0.35000 F1: 0.34038     F2: 0.34609
        Total predictions: 15000        True positives:  700    False
positives: 1413   False negatives: 1300   True negatives: 11587
```

This precision and recall is very good as well. If we round to two decimal places, the precision and recall is exactly the same as what is was when I originally used a threshold of 8. I'm glad the Udacity reviewer pointed out this approach to me as it gives me some insight into tuning my algorithm. I would argue, however, that the minute increase in precision and recall is not worth the extra time that GridSearchSV and SelectKBest has to spend trying to find the optimal features from a larger set (2 additional features). So I have left the final classifier as it was.

Since I used the KNearestNeighbors algorithm for my classifier, I had to use scaling, which I built into my pipline when I was tuning the algorithm. Without normalizing, the KNearestNeighbors does not correctly classify the data since it classifies data points based on their distance to a test point on an XY plane. Without this normalization, mismatches in the values of the scales on the x or y axis can really cause this classifier to perform poorly.

I engineered a new feature called "stock_salary_ratio" which I though was important to add because it was something that helps us see when total_stock_value and salary are out of alignment. For example, if an employee received a huge amount of stock but a small salary, I postulated that something might be up. If for example, there was some way money could be tax deferred or otherwise hidden by paying employees in stock, this might have been an indicator of fraud when the salaries and stock were outliers. However, when I looked at the predictive power of this feature, I found it wasn't very good. This could have been because POIs often received both high salaries and a lot of stock, so the ratio was no different than someone with a small salary and a proportional about of stock. This would reduce the predictive power of this feature. As a result, I did not use this.

Here are the feature score for my use of the default SelectKBest, which was powerful enough for my purposes here:

[('salary', 18.003739993113935), ('total_payments', 8.671773243105207), ('bonus', 20.524645181851792), ('total_stock_value', 23.898259813869416), ('exercised_stock_options', 24.532722463057976), ('to_messages', 1.5942560277180795), ('from_poi_to_this_person', 5.14221919450697), ('from_messages', 0.1753832041587958), ('from_this_person_to_poi', 2.3388361146462624), ('shared_receipt_with_poi', 8.432635423024681), ('stock_salary_ratio', 0.11905356728510776)]

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"]

I created classifiers for Naïve Bayes, Support Vector Machines, Decision Tree, and K-Nearest Neighbors. Because the accuracy of K-Nearest Neighbors was highest and I was most able to understand how this classifier works behind the scenes (and this course was my first introduction to machine learning), I took K-Nearest Neighbors and tuned it in the final analysis to see how good I could get it to perform.

Here is the accuracy score of the algorithms I used:

Naïve Bayes: 0.8604651162790697

Support Vector Machines: 0.8837209302325582

Decision Tree: 0.813953488372093

K-Nearest Neighbors: 0.9069767441860465

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 an algorithm means you are making small tweaks to how the algorithm classifies the features and labels. The default settings of the algorithm might not train the classifier using the training data in such a way that gives it the most predictive power on test data. In other words, it does not optimize how the classifier generalizes to new data. If you don't tune the parameters well, you are either under fitting or overfitting the data and the algorithm either missing important dynamics of the features in the data (bias) or thinks that random variation in the data is actually important when it's not (variance).

   I tuned the parameters in my data by using GridSearchCV to find the optimal number of features to use, the optimal number of n_neighbors to use in the K-Nearest Neighbors Algorithm, and whether it was optimal to weight the importance of the nearest neighbors based on distance or to keep them uniform.

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"]

   The concept of validation is making sure that the algorithm we have trained and use as a classifier can accurately handle new data it has never seen. The issues that are talked about at length in the Udacity Machine learning course is underfitting (causing bias) and overfitting (causing the algorithm to think random variance is meaningful). In both cases, the model might work well when fit to the training data set, but it does horribly when it tries to label data is has not seen before because it is either not picking up meaningful relationships or seeing meaning when none is there. We have seen this in the course when we split the data and test the accuracy and it comes out lower than .7. However, a common mistaken is we can also get extremely high accuracy of near 1 when training on all data and testing on all data available because the model is predicting things it has already seen before and has not learned how to apply what it has learned to new situations. This is a false positive because if the model is actually confronted with data from the wild, it will not perform anywhere near an accuracy score of 1. Validation helps us determine if we have a model that will actually perform well in the wild and find that bias and variance trade-off.

   When we just split the data into test and training data sets, especially on a data set as small as the one here with only 18 POI, the model becomes unstable. To address this, I validated my analysis by using stratified shuffle split due to the small size of the dataset. This cross validation method was also used in the tester script.

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"]

Here is the performance of my classifier for two evaluation metrics:

Precision: 0.33065

This means that 33% (there were 699 true positives, 1,415 false positives, and so this value comes from 699/2114) of the people identified by the model as people of interest are actually people of interest. Looked at in another way, two thirds of the people the model identified as people of interest are not people of interest. But, it is wholly possible that the original labeling of POIs was not totally accurate. Maybe some people who committed fraud got away with it and we could take a deeper look at these other 66% and do further investigations on their activities.

Recall: 0.34950

This means that 35% (there were 699 true positives, 1,301 false negatives, and so this value comes from 699/2000) of actual POIs were identified as POIs by the algorithm. Looked at in another way, nearly two thirds of the people that are actual POIs were not classified by the model as POIs. This doesn’t seem that fantastic actually.