Data Analyst Nanodegree

Project 5

Identify Fraud from Enron Email

Section 1: Data Exploration and Outlier Investigation

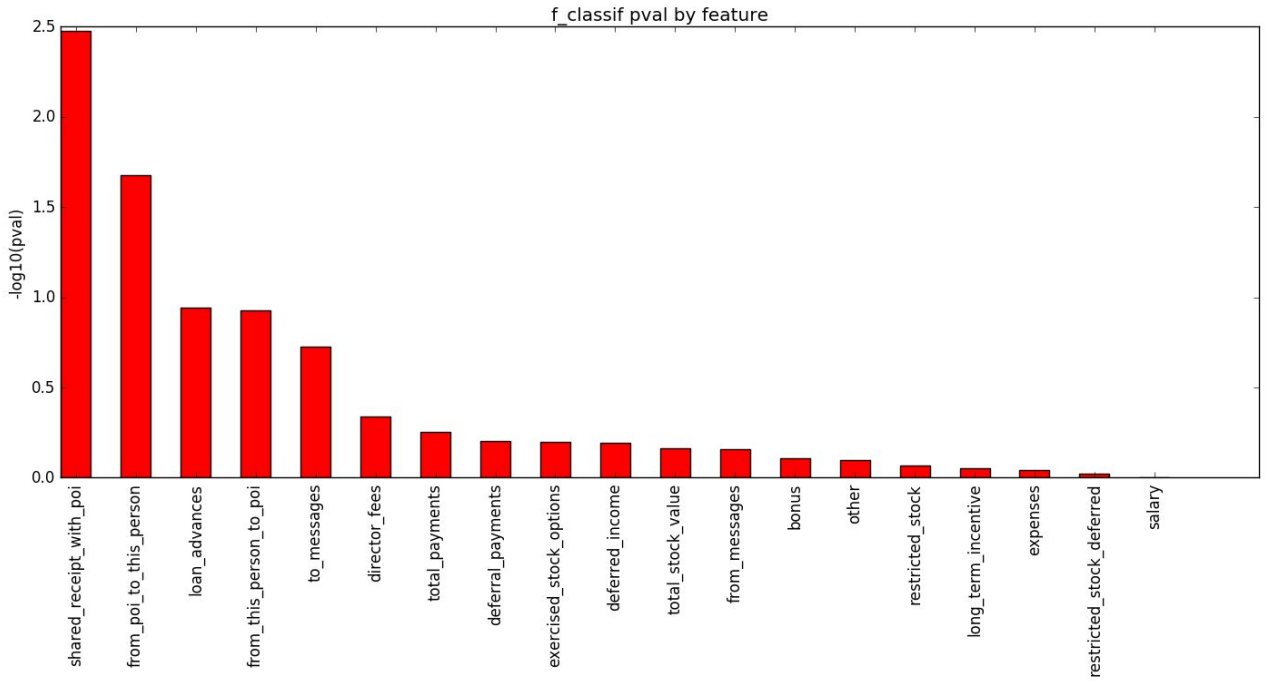
Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those?

The dataset consisted of 144 points, where each point corresponded to one Enron employee and had a poi label as well as 20 features. There were 126 non-POIs and 18 POIs. There were several outliers in the dataset. Namely, each row in the dataset was supposed to contain financial and email information for one Enron employee, but I found rows for "TOTAL" and "THE TRAVEL AGENCY IN THE PARK". These were not Enron employees so I removed these rows from my dataset. Additionally, I found that some financial values in the dataset did not match the values listed in enron61702insiderpay.pdf . Therefore, I modified the values in the dataset to match the values in the pdf file. I couldn't find a pattern as to why these values were mis-extracted from the pdf so I couldn't use that pattern to verify that other rows which matched that pattern were not mis-extracted as well. However, I did spot-check some of the other rows to make sure that they matched the pdf file. Ideally, I would re-extract the data from the pdf file, but initial attempts at doing so revealed that it would not be straightforward. Therefore, I decided to move forward with the data in the current state and see if I could meet the accuracy requirements as is.

Section 2: Creating new features, Properly scaling features, and Intelligently selecting features

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.

To determine which features to use in my POI identifier, I used the p-values of the "sklearn.feature_selection.f_classif" function to rank the features, then I started developing a POI identifier which used all the features, with the intention of removing features in reverse rank order if the POI identifier was taking too long to run or consuming too much memory.



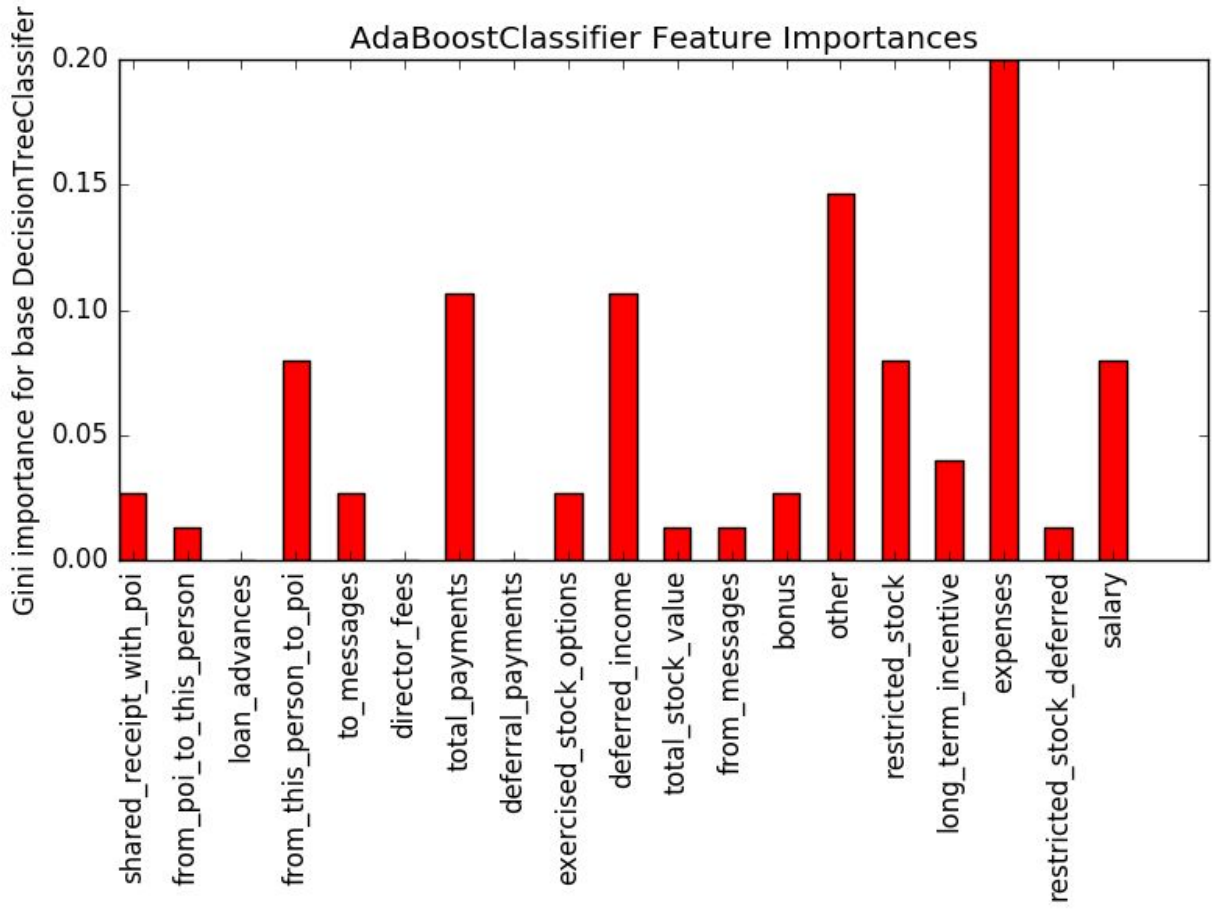
In the end, I used all the features as the POI identifier only took 10s of seconds to fit and validate on my laptop. Since I used AdaBoost with Decision Trees as the base estimators, feature scaling was not necessary.

I tried creating my own feature which I called "poi_mail_ratio". This feature was calculated in the following way:

$$poi\_mail\_ratio = \frac{from\_this\_person\_to\_poi + from\_poi\_to\_this\_person + shared\_receipt\_with\_poi}{from\_messages + to\_messages}$$

The rationale behind this feature was that it was an extension of the idea behind the other "email to/from poi" features. I believe that the idea behind those features was that if someone was frequently communicating with a poi, then perhaps they were communicating about "activities of interest" with that poi and were a poi themselves. However, some people may simply communicate a lot in general, so this feature normalizes for that. Unfortunately, adding this feature to my dataset did not seem to increase the performance of my algorithm.

For my final algorithm I used an AdaBoostClassifier with a DecisionTreeClassifier as my base estimator. The feature importances were as follows:



I found it interesting that the feature ranking which was determined using the f_classif p-values differed so greatly from the ordering by Gini importances.

Section 3: Picking an algorithm

What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?

The first algorithm that i tried was a DecisionTreeClassifier. However, after trying various parameter values and features, the best precision and recall scores that I was able to achieve were 0.23893 and 0.23750, respectively. Therefore, next I tried using an AdaBoostClassifier with a DecisionTreeClassifier base estimator and after some tuning, I was able to achieve precision and recall scores of 0.43395 and 0.30550 .

Section 4: Tuning the algorithm

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? (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).

Some algorithms have the ability to perform well for a wide variety of datasets, however, they need to be configured differently for the various datasets to get the best performance on those datasets. In general, "performance" can be defined differently in different applications and for different people, and there are tradeoffs between some commonly used performance metrics such as speed and accuracy. Parameter tuning is the process of configuring an algorithm to achieve the best performance for your definition of performance. If you do not do this well, the algorithm may not give you the results that you want when it otherwise could have. For this particular application, I was primarily concerned with the precision and recall scores that the algorithm was able to achieve and less concerned about the time in which it took to run. I started by identifying the parameters which I needed to tune, then determining if I could easily determine optimal choices for any of the parameters based on the properties of my problem (For example, if one parameter should take a certain value when all the feature values are positive). Next, I looked at parameters which let me trade speed for accuracy and tried a few values to find an acceptable tradeoff for my definition of performance. For the AdaBoostClassifier, the "n_esimators" parameter seemed to do this. I tried using the default value of 50, as well as 75, 100, 200, and 400. I found that 75 gave a good performance boost without taking too much longer than 75 while higher values gave modest to negligible performance gains. Last, I read about the remaining parameters and explored different values for those.

Section 5: Validation strategy

What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis?

Validation is the process of determining how good the algorithm is (or will be) at solving the problem that it was created to solve. In this case, determining how good it is at identifying Persons of Interest from financial and email data. A classic mistake is to validate an algorithm using data which was used to train it. In some cases, the algorithm performs much better

on the training data than on novel or "real world" data that was not used during training. To validate my algorithm, I used the provided "tester.py" script which utilizes the StratifiedShuffleSplit function in sklearn.

Section 6: Usage of evaluation metrics

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.

Two evaluation metrics that are commonly used are "precision" and "recall". The precision metric is the likelihood that a sample which the algorithm identifies as of a particular class is actually of that class. The recall metric is the likelihood that the algorithm will identify a sample as from a particular class when it is that class. My algorithm had a precision score of 0.4339 and a recall score of 0.3055. This means that when my algorithm identifies someone as a POI, they are a POI 43.4% of the time. Furthermore, when my algorithm is asked to classify someone who is a POI, it says they are a POI 30.6% of the time.