INTRODUCTION

The project attempts to detect Frauds from the Enron corporate scandal. The data contains information about each employee along with the following fields : bonus, deferral_payments, deferred_income, director_fees, email_address, exercised_stock_options, expenses, from_messages, from_poi_by_non_poi, from_poi_to_this_person, from_this_person_to_poi, loan_advances, long_term_incentive, other, poi, restricted_stock, restricted_stock_deferred, salary, shared_receipt_with_poi, to_messages, to_poi_by_non_poi, total_payments, total_stock_value.

Dataset: 145
total poi: 18
non poi: 127
total features used: 3

missing values in to messages: 59
missing values in from poi to this person: 59 missing values in from messages: 59 missing values in from this person to poi: 59 missing values in salary: 64

missing values in bonus: 51
3 features are created and used: “bonus_by_salary", "to_poi_by_non_poi", “from_poi_by_non_poi”

I looked for the more obvious outlier, the row TOTAL, which was the total of all salaries, etc. So I removed it immediately, by checking which point had an absurdly high salary. I also checked a plot between salary and bonus. There were outliers who were potentially frauds, so I held it as a feature for my model.

FEATURE CREATION & RATIONAL BEHIND USING ALL FEATURES
I looked into finding other outliers with really unusual ratio of salary to bonus, by plotting a graph with salary and bonus, and checking for outliers. Similarly I checked for from_message & from_this_person_to_poi. And same with to_message. The assumption was that the fraud would send more messages to POI rather than non poi. The from_this_person_to_poi and from_message didn’t show much promise in the plot. So I tried:

I used these features to build my model : "bonus_by_salary", "from_this_person_to_poi", "from_poi_to_this_person", “shared_receipt_with_poi”.
With all these features, I got a performance metrics of: Accuracy: 0.78355 Precision: 0.29180 Recall: 0.13350
I tried removing shared_receipt_with_poi, and got a performance of

Accuracy: 0.77527 Precision: 0.23543 Recall: 0.10500

Then I tried turning the from_this_person_to_poi to a ratio upon from_messages. I got a performance of Accuracy: 0.84780 Precision: 0.68328 Recall: 0.44550 Which was really good, and then tried removing even bonus_by_salary

Accuracy: 0.84725 Precision: 0.03361 Recall: 0.00800

The results significantly dropped. So this bought me to the conclusion that the 3 features.
"bonus_by_salary","to_poi_by_non_poi", “from_poi_by_non_poi"

These features were created using the base features.

“Bonus_by_salary”: I wanted to consider people who were given high bonus. But bonus alone wouldn’t work, because it would be possible that people at higher position, with high salary, would be given high bonus. So I considered taking the ratio of bonus by salary.

Performance using just bonus, as opposed to bonus/salary: Accuracy: 0.80260 Precision: 0.50868 Recall: 0.38100
“to_poi_by_non_poi & from_poi_by_non_poi”: The number of messages that was sent from the person to poi might might give an indication that the person is a poi. But a person who generally sends a large number of emails might get considered as poi. So i took it as a ratio of from_this_person_to_poi by from from_messages. And did the from_poi_to_this_person and to_messages to create variable from_poi_by_non_poi. Performance of from_poi_to_this_person (along with all other features in the final feature list):

Accuracy: 0.76190 Precision: 0.35513 Recall: 0.23350 Performance of from_this_person_to_poi (along with all other features in the final feature list):
Accuracy: 0.76270 Precision: 0.35946 Recall: 0.23850

FEATURE SCALING

Scaling is used to make every feature equally important. Lets say you have two input vectors: X1 and X2. and lets say X1 has range(0.1 to 0.8) and X2 has range(3000 to 50000).. Now for SVM classifier will be a linear boundary lying in X1-X2 plane. The slope of linear decision boundary will not depend on the range of X1 and X2, instead it should depend upon distribution of points. Now let make a prediction on the point (0.1, 4000) and (0.8, 4000). There will be hardly any difference in the value of function, thus making SVM less accurate since it will have less sensitivity to points in X1 direction. But decision tree doesn’t have a diagonal line, both features aren’t affected at once. It division happens vertically and then horizontally. Therefore you don’t have to worry about what’s happening in the other dimension when your working with one

ALGORITHM

GaussianNB, and Decision Tree was used to do this. GaussianNB score was subpar. So I went ahead with decision tree. GaussianNB score -
Accuracy: 0.74250 Precision: 0.05148 Recall: 0.01650 DecisionTree-

Accuracy: 0.81430 Precision: 0.54341 Recall: 0.44750

Based on the performance, it was an easy selection. I had to select DecisionTree.

Accuracy isn’t the best selection of performance in our case, because even if it doesn’t predict POI for all the data, the accuracy will turn out to be pretty high. So we select Precision and Recall

PARAMETER TUNING

One of the difficulties is that learning algorithms (eg. decision trees, random forests, clustering techniques, etc.) require you to set parameters before you use the models (or at least to set constraints on those parameters). How you set those parameters can depend on a whole host of factors. That said, your goal, is usually to set those parameters to so optimal values that enable you to complete a learning task in the best way possible. Thus, tuning an algorithm or machine learning technique, can be simply thought of as process which one goes through in which they optimize the parameters that impact the model in order to enable the algorithm to perform the best (once, of course you have defined what "best" actual is). This is why tuning the algorithm is so important.

The max_depth parameter denotes maximum depth of the tree. It can take any integer value or None. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

By default, it takes “None” value. And the min_sample_split tells max_depth the minimum no. of samples required to split an internal node. If an integer value is taken then consider min_samples_split as the minimum no. If float, then it shows percentage. By default, it takes “2” value. Without tuning my algorithm, I got a result of

Precision: 0.54425 Recall: 0.45200

On setting max_depth to 2 and min_samples_leaf to 5, (These numbers were selected from the article from which I read, and then tweaked manually to get the best output.) The results were Precision: 0.68328 Recall: 0.44550

VALIDATION

Validation helps us estimate performance on independent dataset and serves as a check on overfitting. The data is split into training and testing data. We train the data with the training data and see if the trained model works against the testing data. This gives us a good estimate of whether the model works.

While performing validation, we need to maximize training set to create the best model possible, but also maximize test set to validate the model, in the best way possible. When the data is limited, this can pose a problem, because if you increase training dataset, the test dates is reduced and vice versa. We use k-fold cross validations, the data is divided into k bins, and you run k separate experiments, using k different bins to test and the rest of the data to train. In the tester.py function, we use StratiedShuffleSplit which is a merge of StratifiedKFold and ShuffleSplit, which returns stratified randomized folds. The folds are made by preserving the percentage of samples for each class. Although it does not guarantee that all folds will be different,

although this is still very likely for sizable datasets.In our
case, the data is shuffled and split 1000 times to get the most accurate performance metrics. The dataset we work on is small and skewed towards non-POI, we need a technique that accounts for that or the risk is that we would not be able to assess, in the validation phase, the real potential of our algorithm in terms of performance metrics. The chance of randomly splitting skewed and non representative validation sub-sets could be high, therefore the need to use stratification (preservation of the percentage of samples for each class) to achieve robustness in a dataset with the aforementioned limitations.

EVALUATION

The two performance metrics we focus on is Precision and Recall. Precision can be interpreted as the likelihood that a person identified as a POI is actually a true POI. While Recall can be interpreted as the fraction of POI the algorithm detected among all the POI.

The complete performance evaluation of the algorithm I selected after tuning was,
Accuracy: 0.84780 Precision: 0.68328 Recall: 0.44550 F1: 0.53935 F2: 0.47883