# Identifying Fraud from Enron Email Project

**Min Lai**

**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?**

The goal of this project is to build a machine learning algorithm which use features extracted from Enron emails dataset to identify if an Enron employee is the person who was involved in Enron fraud or not. Person evolved in Enron fraud is referred as Person of Interest(POI). Here are some background about target dataset:

| | |
|---|---|
| Number of Eron Employees | 146 |
| Number of POI | 18 |
| Nubmer of Features | 21 |
| Feature Category | Financial data, email data, POI indicator |

In the dataset, an employee is POI or not is known. So how to identify POI is a typical supervised classification problem where machine learning provides full set of tools and methodology to to find an optimized classifier to predict whether an Enron employee is POI or not based on selected features.

Outlier could greatly deteriorate the accuracy of classifier, so finding and removing outliers is a very critical step before classifier training. One way to identify is to visualize the distribution of data points. The data points which distances away from majority data points are likely to be outlier. Statistically, top 5% or 10% percentile of contain feature can be treated as outlier. I used ggplot package to created scatter plots for 4 pairs features: salary vs bonus, from_this_person_to_poi vs from_poi_to_this_person, exercised_stock_options vs total_stock_value and shared_receipt_with_poi vs from_message. I compared the scatter plots with all data points and 98% data points. The scatter plot looks much better with 98% data points(Scatter plots given in Appendix section 1). However, when I looked into top 2% data points removed, some of them are POI data point. Since there are only 18 POI data points in the data, no POI data point should be removed. Then the data points close to outlier like POI point should not be removed either. Percentile may not be a propitiate approach considering the small size and special distribution of the target dataset. In the process, I detected one error in the data set one is the 'TOTAL' which looks like total number for all employees, which is an outlier. In data cleaning process, I found 'THE TRAVEL AGNECY IN THE PARK' which is not an employee and has lots of NaN value. After removing these two outliers, 144 data points will be used in classifier training and testing.

**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 doesn't 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.) If you used an algorithm like a decision tree, please also give the feature importances of the features that you use.**

This is final set of features that I picked: **from_poi_ratio, shared_receipt_with_poi, to_poi_ratio** which includes two features that I created. I creates four new features using ratio instead of independent values:

***salary_total_pay_ratio*** = salary/total_payments

***total_stock_exstock_ratio*** = exercised_stock_options/total_stock_value

***from_poi_ratio*** = from_poi_to_this_person/from_messages

***to_poi_ratio*** = from_this_person_to_poi/to_messages

the ratios can reveal the relationship better, which gives comparative value. For example, if a person's salary is only small portion of total payment, it is more likely this person is high rank executive who may be a POI. If larger portion of a person's emails are to POIs or from POIs, it tells this person closely tied to POIs who is very likelyh a POI as well.

My selection process had multiple iterations. Initially, I picked up most important representatives from both category: financial features and email features 7 features based my knowledge of the fraud, understanding of the dataset: salary, bonus, total_payments, total_stock_value, shared_receipt_with_poi, from_this_person_to_poi, from_poi_to_this_person to test the water. I tried those features with 3 types of classifier(Naive Bayer, Decision Tree, SVM), the highest accuracy is around 0.79, highest precision is only 0.24, which is not very good. Then, I used KBest to help me to select features since total number of features is 20. I tried K value from 2 to 14, and I found when K = 3, the validation metrics looks the best for Decision Tree, SVM (the validation metrics for each K value and classifier listed in Appendix section II). To my surprise, three features selected by KBest is from_poi_ratio', 'shared_receipt_with_poi', 'to_poi_ratio' which are all email relation features. Using Decision Tree classifier, without tuning, I saw following cross validation metric:

| accuracy | precision | recall | f1 | f2 |
|---|---|---|---|---|
| 0.839333 | 0.658286 | 0.576 | 0.6144 | 0.590769 |

Which is not bad. However, I suspected that it is too good to be true. Finally, I evaluated 3 set of features with K = 3-5 and played with 5 different classifiers(in poi_classifier_selection.py) and chose the set which give me the best overall validation metric, which is still **'from_poi_ratio', 'shared_receipt_with_poi'**, **'to_poi_ratio'**. So it looks like financial features are not good predictors which actually be noisy.

| K | K Best Features |
|---|---|
| 3 | 'from_poi_ratio', 'shared_receipt_with_poi', 'to_poi_ratio' |
| 4 | 'from_poi_ratio', 'salary_total_pay_ratio', 'shared_receipt_with_poi', 'to_poi_ratio' |
| 5 | from_poi_ratio', 'salary_total_pay_ratio', 'shared_receipt_with_poi', 'loan_advances', 'to_poi_ratio' |

I used scaling in the training process since the algorithm based on distance are largely impact by scale like SVM, K Nearest Neighbor. Without scaling, the feature with larger scale can dominate. I used MinMaxScaler to transform all values between 0 and 1 so that SVM and similar classifier will have more accurate prediction.

**3. What algorithm did you end up using? What other one(s) did you try?**

I ended up pick decision tree algorithm. Picking up algorithm is a long and painful process for me. I went through several rounds. I found different feature selections works differently in different algorithm. For example, three email features works best with decision tree algorithm, if adding fanatical features, accuracy, precision and recall all goes down. While if I used SVM, more features including both email features and financial features works better. That puzzled me which direction I should go. So I experimented different combination of feature selections and algorithm , collected accuracy, precision and recall to find the combination which gives me the best result. Decision Tree is winner through this process. so I finally picked it.

I selected 5 algorithms: Decision Tree, SVM, K Nearest Neighbors, Random Forest, Adaboost. Decision tree, random Forest, adaboost are all tree based algorithms. Random Forest, Adaboost are enhanced version to deal with over plotting and boost week points. However, maybe this dataset is small and with relatively low number of dimensions, the simple Decision Tree algorithm works better than Random Forest and Adaboost. K Neares Neighbor is based on distances to nearest neighbors. SVM segregates classes in high dimensional space by defined Kernel. So I scaled the dataset, before I trained SVM and K Nearest Neighbor classifier. K Nearest Neighbor seems doesn't work well with my feature selections. SVM performed better but not as good as Decision Tree. It may be related to parameter tuning. So I actually tuned both algorithms: Decision Tree and SVM, which I will document in next section. After tuning, the decision tree still outperforms SVM classifier, so I finally choose Decision Tree. For features that I selected, their importance is

| shared_receipt_with_poi | to_poi_ratio | from_poi_ratio |
|---|---|---|
| 0.446789186 | 0.311064347 | 0.242146468 |

Which is reasonable, no signature feature dominates.

**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?**

Algorithm model has set of parameters associated it which controls behavior of the model. Depends on nature of dataset, the parameters need to be tune to make algorithm fit the data set optimally. Some algorithm like SVM, it performance can varies a lot using different parameter. If parameters are not well tuned, the model may not have good accuracy, or may overfit.

In this project, I found Decision Tree Algorithm works unexpectedly well even without much tuning, which actually makes me suspect if I did something wrong. So I tuned SVM classifier with different set of feature selection as a comparison. Decision tree doesn't have lots of parameter to tune, I just tried different min_samples_split value which only makes minor differences.

To tune SVM algorithm, I used Grid Search. I actually pipelined with MinMaxScaler, PCA and SVM classifier. with following parameter grid:

***PAC:***

| | |
|---|---|
| n_components | 2, 3, 4 |

***SVM***

| | |
|---|---|
| C | 1, 5, 10, 20, 50, 100, 500, 1000, 1e4, 1e5, 1e6 |
| gamma | 0.01, 0.1, 1, 5 |
| class_weight | {1:2}, {1:5}, {1:8}, {1:10}, {1:20} |
| ~~kernel~~ | ~~linear, kbf~~ |

The linear kernel took lots of time to run. And from two dimensional scatter plot, the data points spread out a lot, so linear kernel may not be a good fit here. I removed it from the grid. I added class_weight parameter to the grid because the dataset is not balanced in two classes. There are only 18 POIs out of 144 data points(after removing outliers). When I ran the tester program by passing the best estimator given by GridSearchCV, I got divided by zero exception. I found out that the default scoring function is based on accuracy score. Since precision and recall are more important for this project. I set the scoring parameter to 'precision' in GridSearchCV, But the result is not very satisfactory:

| accuracy | precision | recall | f1 | f2 |
|---|---|---|---|---|
| 0.81633 | 0.25943 | 0.05500 | 0.09076 | 0.06529 |

Then, I set the scoring to 'recall'. In this case, recall is almost 1, precision is only around 0.15. This tells me that parameter selection by GridSearchCV depends on scoring function. In this project, balance between precision and recall is needed. So I tried f1 score, this is the best estimator given by Grid Search:

Pipeline(steps=[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('reduce_dim', PCA(copy=True, n_components=2, whiten=False)), ('svm', SVC(C=100, cache_size=200, class_weight={1: 5}, coef0=0.0, degree=3, gamma=1, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))])

which gives acceptable result but not as good as DecisionTreeClassifier

| accuracy | precision | recall | f1 | f2 |
|---|---|---|---|---|
| 69992 | 0.30518 | 0.62700 | 0.41054 | 0.51780 |

However, for DecisionTreeClassifier, when min_samples_split=10, I got following result

| accuracy | precision | recall | f1 | f2 |
|---|---|---|---|---|
| 0.85567 | 0.71230 | 0. 58800 | 0.64421 | 0.60926 |

I noticed warning message when using f1 score : "UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples." The default F1 score implementation may have some limitation here. sklearn document says that custom scoring function can be implemented, which I can look further out the scope of this project. Because the nature of this small dataset, I implemented multiple layer loops program to tune SVM classifier and DecisionTreeClassifier further using the provided tester(Part of the result listed in Appendix section IV) . Best Overall metrics that I got are:

**Pipeline**(steps=[('scale', MinMaxScaler(copy=True, feature_range=(0, 1))), ('reduce_dim', PCA(copy=True, n_components=2, whiten=False)), ('svm', **SVC**(C=100, cache_size=200, class_weight={1: 5}, coef0=0.0, degree=3, gamma=5, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False))])

```
Accuracy: 0.75275       Precision: 0.34933      Recall: 0.56050  F1: 0.43041      F2: 0.50004
Total predictions: 12000   True positives: 1121    False positives: 2088      False negatives:  879
True negatives: 7912
```

```
DecisionTreeClassifier(compute_importances=None, criterion='gini',      max_depth=None,
max_features=None, max_leaf_nodes=None,   min_density=None, min_samples_leaf=1,
min_samples_split=20, random_state=37, splitter='best')
```

```
Accuracy: 0.84856          Precision: 0.65145          Recall: 0.68500 F1: 0.66780       F2: 0.67802
Total predictions: 9000  True positives: 1370        False positives:  733       False negatives:  630
True negatives: 6267
```

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

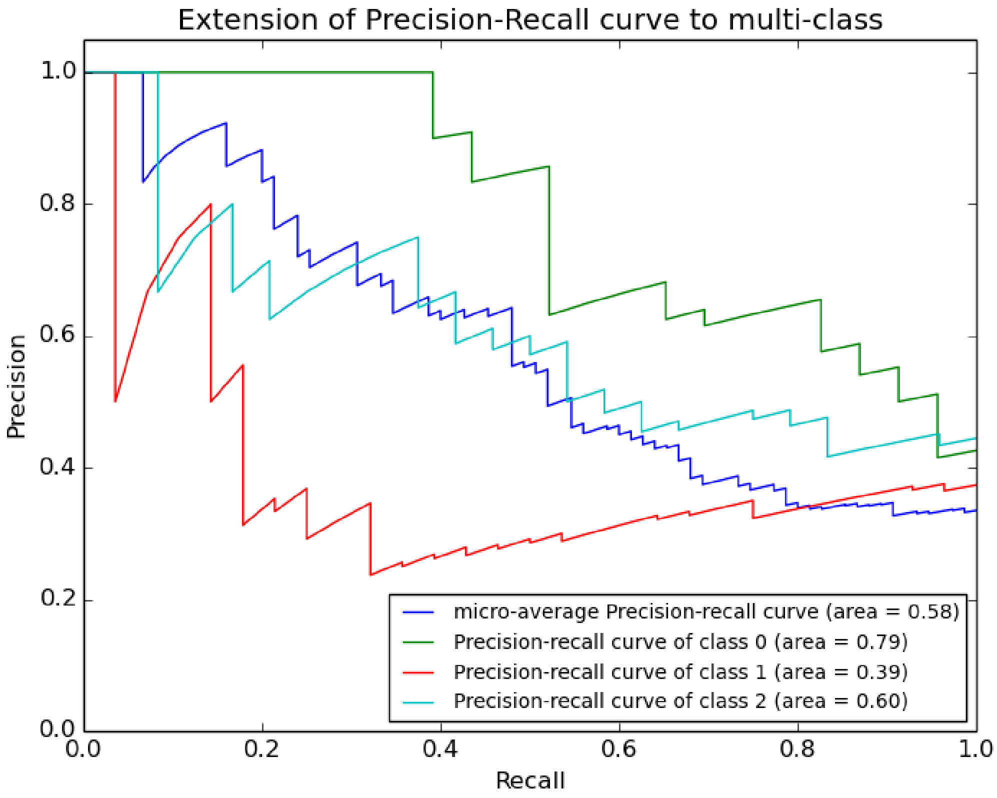
Validation is a test to assess how well a model(classifier/regression) predict results. One classic mistake you can make is to use training data or data depending on training data to  run validation test. Purpose of validation is to detect over fitting problem and performance issue. To do an effective cross validation, an independent dataset other than training set should be used. In general practice, the given dataset need to be randomly split into  two sets: one for training and another for test/validation. In reality, available dataset have limited number of data points. There is a dilemma to maximize both training and test set.  Sklearn provides solution called cross-validation. The basic approach is k-fold:  the training set is randomly split into k equal sized smaller sets. The following procedure is followed for each of the k “folds”: A model is trained using k-1 of the folds as training data; the resulting model is validated on the remaining part of the data.  And the process will repeat K times and each K subsamples used just once as validation set. Finally, cross validation process will give average validation scores like accuracy, precision and recall.

The given tester.py uses one of sklearn k fold implementation StratifiedShuffleSplit with folds =1000. This cross validation model is a merger of stratified K fold and shuffle split, which returns stratified randomized folds. The folds are made by preserving the percentage of samples for each class. This model suits for small data set in this project well. So I made use of this model to do across validation for the classifier that I picked up in this project

**6. Give at least 2 evaluation metrics, and your average performance for each of them.**

There are several evaluation metrics: accuracy score, precession score, recall score, F score. Accuracy score is commonly used, which is the ratio of  correct predictions. Precision = number of true positive / (number of true positive +  number of false positive) which measures the classifier's ability not to a negative label as positive label. Recall = number of true positives/( number of true positives + number of false negatives), which measures the classifier's ability to identify all  positive labels. F score is a weighted harmonic mean of the precision and recall.  In tester program the F1 = 2* number of true positives/(= 2* number of true positives + number of false positives +  number of false negatives) and F2 = 5 * precision/ (4*precision + recall) which gives different weight to precision and recall.

Ideally, precision and recall are both high if the classifier really works well. However, from my experience in this project, there should be some trade off.  Flowing plot is form sklearn document, which illustrates the relationship between precision and recall:



In this project, precission and recall are in the stair step area, so I can see precision drops as recall increases. To decide which model is the best, I chose a balanced metrics between precision and recall. The best over average scores I got are

For DecisionTreeClassifier, there is the average metrics that I got using crossing validation

| accuracy | precision | recall | f1 | f2 |
|---|---|---|---|---|
| 0.84856 | 0.687604 | 0.65145 | 0.66780 | 0.67802 |

# Appendix

## I. Outlier detection

## Legends POI : Blue, non - POI: Red

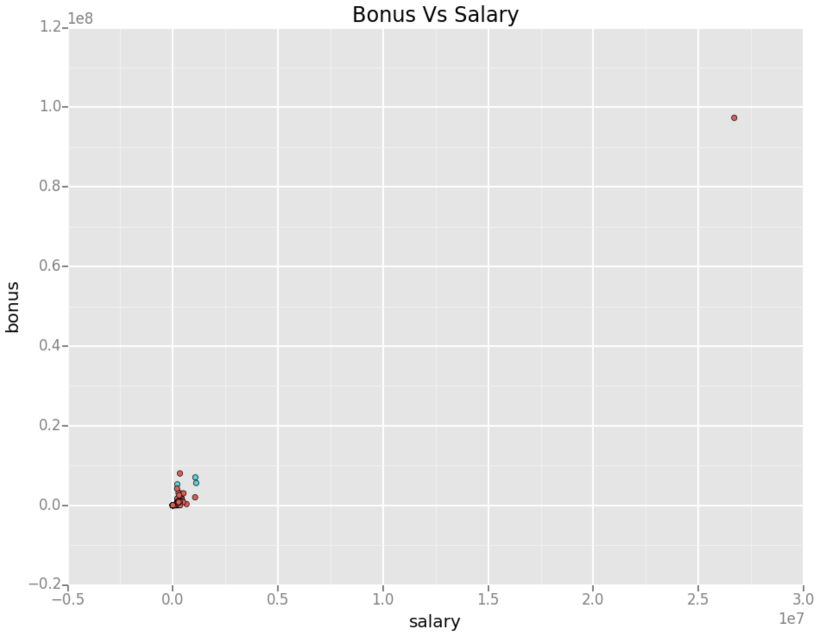


Fig 1 Bonus Vs. Salary all data points

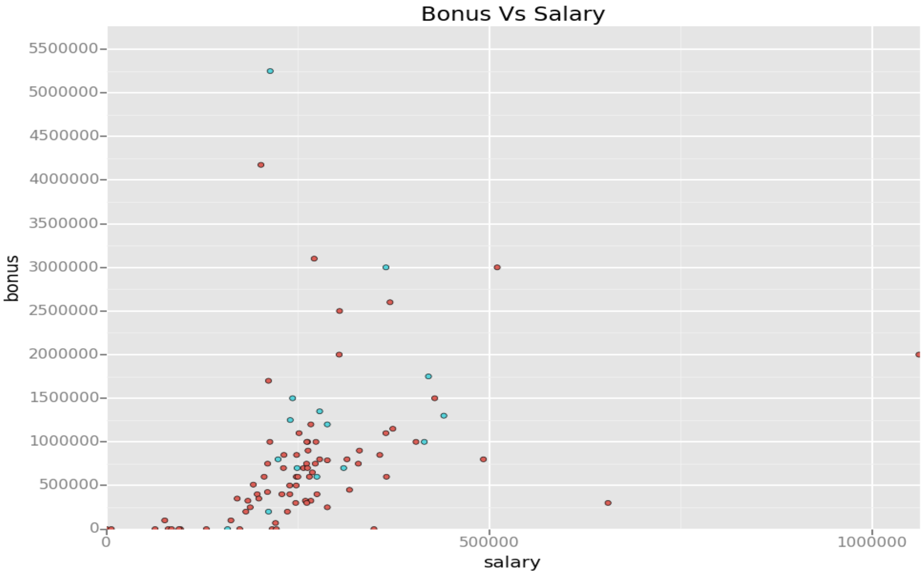


Fig 2 Bonus Vs. Salary excluding top 2% points(poi - blue)

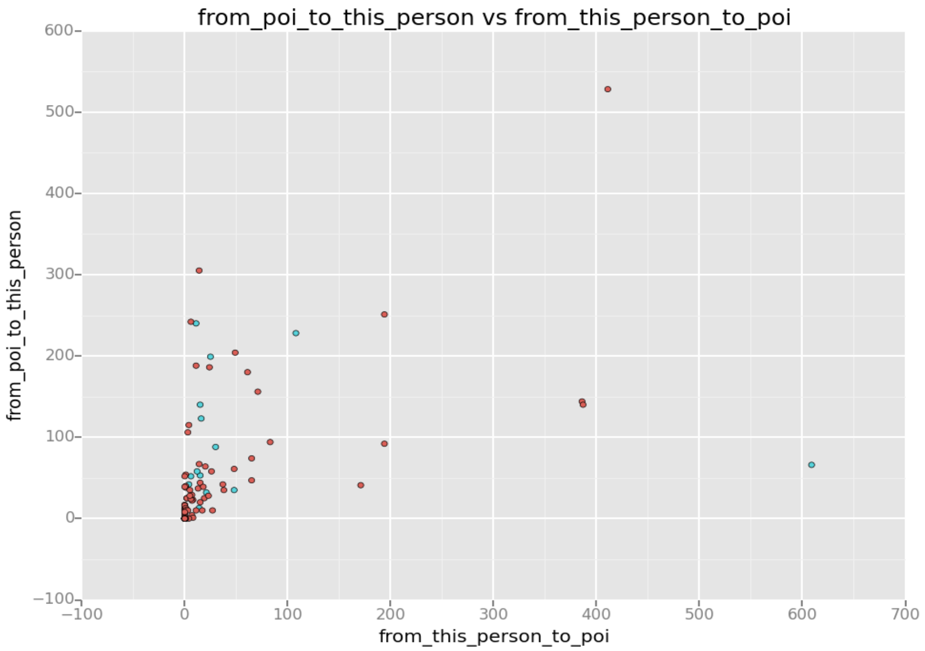


Fig3 From_poi_to_this_person vs from_this_person_to_poi

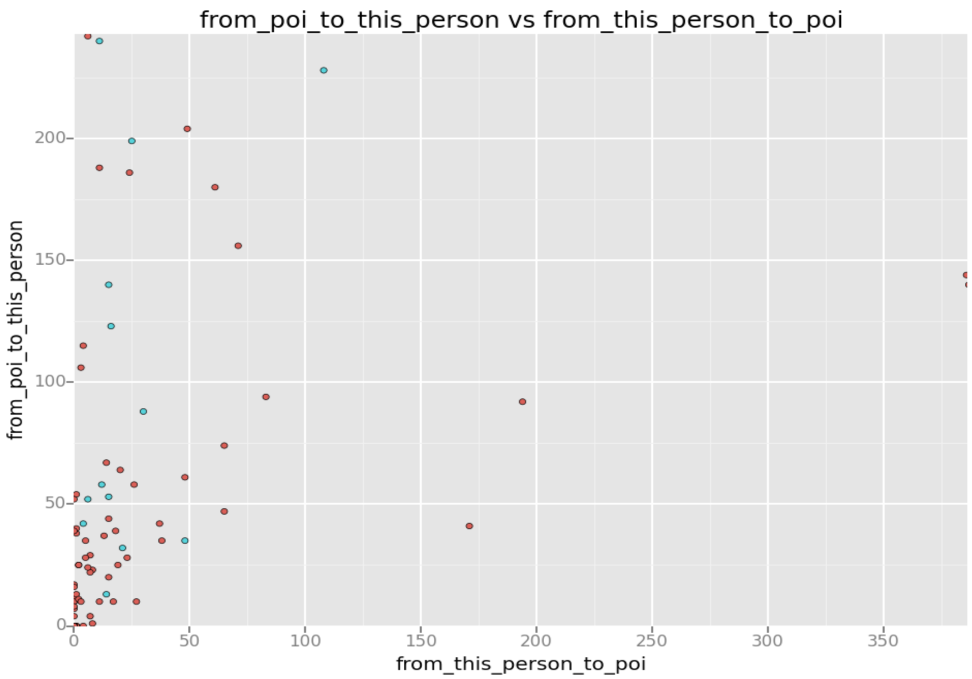


Fig3 From_poi_to_this_person vs from_this_person_to_poi(excluding top 2%)

## II Feature Selection

Cross validation metric for K best features selected by SKlearn KBest algorithm where K from 2- 14.

***Feature list***

| **K** | K Best Features |
|---|---|
| 2 | from_poi_ratio', 'shared_receipt_with_poi' |
| 3 | from_poi_ratio', 'shared_receipt_with_poi', 'to_poi_ratio' |
| 4 | from_poi_ratio', 'salary_total_pay_ratio', 'shared_receipt_with_poi', 'to_poi_ratio' |
| 5 | from_poi_ratio', 'salary_total_pay_ratio', 'shared_receipt_with_poi', 'loan_advances', 'to_poi_ratio' |
| 6 | from_poi_ratio', 'salary_total_pay_ratio', 'shared_receipt_with_poi', 'loan_advances', 'director_fees', 'to_poi_ratio' |
| 7 | deferral_payments', 'from_poi_ratio', 'salary_total_pay_ratio', 'shared_receipt_with_poi', 'loan_advances', 'director_fees', 'to_poi_ratio' |

| | |
|---|---|
| 8 | deferral_payments', 'deferred_income', 'from_poi_ratio', 'salary_total_pay_ratio', 'shared_receipt_with_poi', 'loan_advances', 'director_fees', 'to_poi_ratio' |
| 9 | deferral_payments', 'deferred_income', 'from_poi_ratio', 'salary_total_pay_ratio', 'shared_receipt_with_poi', 'loan_advances', 'director_fees', 'bonus', 'to_poi_ratio' |
| 10 | deferral_payments', 'deferred_income', 'from_poi_ratio', 'salary_total_pay_ratio', 'shared_receipt_with_poi', 'loan_advances', 'other', 'director_fees', 'bonus', 'to_poi_ratio' |
| 11 | 'deferral_payments', 'deferred_income', 'from_poi_ratio', 'salary_total_pay_ratio', 'shared_receipt_with_poi', 'loan_advances', 'other', 'director_fees', 'bonus', 'total_exstock_stock_ratio', 'to_poi_ratio' |
| 12 | 'deferral_payments', 'deferred_income', 'from_poi_ratio', 'salary_total_pay_ratio', 'shared_receipt_with_poi', 'loan_advances', 'other', 'director_fees', 'bonus', 'restricted_stock', 'total_exstock_stock_ratio', 'to_poi_ratio' |

***Naive Bayer:***

| num of features | accuracy | precision | recall | f1 | f2 |
|---|---|---|---|---|---|
| 2 | 0.732111 | 0.002421 | 0.0005 | 0.000829 | 0.000594 |
| 3 | 0.718444 | 0.003717 | 0.001 | 0.001576 | 0.001171 |
| 4 | 0.745917 | 0.017479 | 0.0095 | 0.01231 | 0.010454 |
| 5 | 0.6925 | 0.063533 | 0.0615 | 0.0625 | 0.061896 |
| 6 | 0.299308 | 0.157662 | 0.8185 | 0.264395 | 0.445248 |
| 7 | 0.320857 | 0.145045 | 0.767 | 0.243957 | 0.412898 |
| 8 | 0.320071 | 0.14348 | 0.7565 | 0.241212 | 0.407927 |
| 9 | 0.317643 | 0.142952 | 0.756 | 0.240439 | 0.406955 |
| 10 | 0.314857 | 0.143702 | 0.7655 | 0.241979 | 0.410368 |
| 11 | 0.303667 | 0.133559 | 0.7695 | 0.227612 | 0.39415 |
| 12 | 0.302733 | 0.132249 | 0.7605 | 0.225317 | 0.38998 |
| 13 | 0.304933 | 0.133078 | 0.764 | 0.226673 | 0.392157 |
| 14 | 0.303267 | 0.133235 | 0.7675 | 0.227054 | 0.393166 |

**Decision Tree**

| num of | accuracy | precision | recall | f1 | f2 |
|---|---|---|---|---|---|

| features | | | | | |
|---|---|---|---|---|---|
| 2 | 0.776556 | 0.49644 | 0.3835 | 0.432722 | 0.401781 |
| 3 | 0.839333 | 0.658286 | 0.576 | 0.6144 | 0.590769 |
| 4 | 0.810417 | 0.413901 | 0.3305 | 0.367528 | 0.344378 |
| 5 | 0.81025 | 0.412618 | 0.327 | 0.364854 | 0.341158 |
| 6 | 0.819923 | 0.399291 | 0.338 | 0.366098 | 0.348705 |
| 7 | 0.842929 | 0.435599 | 0.3365 | 0.37969 | 0.352541 |
| 8 | 0.833357 | 0.386348 | 0.283 | 0.326696 | 0.298996 |
| 9 | 0.817143 | 0.344617 | 0.3105 | 0.32667 | 0.316772 |
| 10 | 0.815429 | 0.317955 | 0.255 | 0.283019 | 0.265514 |
| 11 | 0.825533 | 0.322395 | 0.28 | 0.299706 | 0.287563 |
| 12 | 0.818133 | 0.291284 | 0.254 | 0.271368 | 0.260673 |
| 13 | 0.815 | 0.283156 | 0.253 | 0.26723 | 0.258506 |
| 14 | 0.820267 | 0.301822 | 0.265 | 0.282215 | 0.271628 |

**SVM**

| num of features | accuracy | precision | recall | f1 | f2 |
|---|---|---|---|---|---|
| 2 | 0.677667 | 0.308542 | 0.363 | 0.333563 | 0.350623 |
| 3 | 0.716333 | 0.347322 | 0.3145 | 0.330097 | 0.320559 |
| 4 | 0.657917 | 0.248747 | 0.521 | 0.336726 | 0.427435 |
| 5 | 0.654083 | 0.225147 | 0.4405 | 0.297987 | 0.369764 |
| 6 | 0.662846 | 0.231705 | 0.5145 | 0.319516 | 0.413552 |
| 7 | 0.688571 | 0.237428 | 0.5335 | 0.328611 | 0.427005 |
| 8 | 0.703071 | 0.245096 | 0.5185 | 0.332852 | 0.423923 |
| 9 | 0.705357 | 0.245996 | 0.5145 | 0.332848 | 0.42231 |
| 10 | 0.700643 | 0.241322 | 0.511 | 0.327827 | 0.417654 |
| 11 | 0.6826 | 0.18575 | 0.408 | 0.255279 | 0.329218 |
| 12 | 0.6884 | 0.18878 | 0.4055 | 0.257624 | 0.329782 |
| 13 | 0.692333 | 0.1917 | 0.4065 | 0.260535 | 0.332081 |
| 14 | 0.688867 | 0.193378 | 0.4205 | 0.264924 | 0.340513 |

## III Algorithm Selection

**from_poi_ratio, shared_receipt_with_poi, to_poi_ratio**

| | Accuracy | Precision | Recall | F1 | F2 |
|---|---|---|---|---|---|
| SVM | 0.68644 | 0.27879 | 0.259 | 0.26853 | 0.26273 |
| Decision Tree | 0.85611 | 0.71248 | 0.591 | 0.64608 | 0.61186 |
| K Nearest Neighbor | 0.755 | 0.42334 | 0.283 | 0.33923 | 0.3031 |
| Random Forest | 0.82411 | 0.67715 | 0.3985 | 0.50173 | 0.43424 |
| Ada Boost | 0.82856 | 0.64676 | 0.5035 | 0.56621 | 0.52684 |

**from_poi_ratio, salary_total_pay_ratio, shared_receipt_with_poi, to_poi_ratio**

| | Accuracy | Precision | Recall | F1 | F2 |
|---|---|---|---|---|---|
| SVM | 0.69317 | 0.29090 | 0.58500 | 0.38858 | 0.48661 |
| Decision Tree | 0.82267 | 0.45984 | 0.35500 | 0.40068 | 0.37196 |
| K Nearest Neighbor | 0.77675 | 0.21965 | 0.13300 | 0.16568 | 0.14439 |
| Random Forest | 0.80517 | 0.31789 | 0.14500 | 0.19668 | 0.16521 |
| Ada Boost | 0.82342 | 0.40571 | 0.128 | 0.1946 | 0.1483 |

**from_poi_ratio, salary_total_pay_ratio, shared_receipt_with_poi, loan_advances, to_poi_ratio**

| | Accuracy | Precision | Recall | F1 | F2 |
|---|---|---|---|---|---|
| SVM | 0.68733 | 0.25881 | 0.47 | 0.33381 | 0.40406 |
| Decision Tree | 0.82408 | 0.46384 | 0.356 | 0.40283 | 0.37336 |
| K Nearest Neighbor | 0.77675 | 0.21919 | 0.1325 | 0.16516 | 0.14388 |
| Random Forest | 0.80508 | 0.31271 | 0.1415 | 0.19484 | 0.1589 |
| Ada Boost | 0.82333 | 0.40323 | 0.125 | 0.19084 | 0.14501 |

## IV Algorithm Tuning

### SVM:

| Parameters | Accuracy | Precision | Recall | F1 | F2 | Training and Testing Time (Seconds) |
|---|---|---|---|---|---|---|
| C: 100 gamma: 0.1<br>class_weight: {1: 2} | 0.83108 | 0.09091 | 0.0015 | 0.00295 | 0.00187 | 6.665 |
| C: 100 gamma: 0.1<br>class_weight: {1: 5} | 0.687 | 0.29872 | 0.6515 | 0.40962 | 0.52702 | 1.869 |
| C: 100 gamma: 0.1<br>class_weight: {1: 8} | 0.5925 | 0.23574 | 0.6445 | 0.34521 | 0.47854 | 1.976 |
| C: 100 gamma: 0.1<br>class_weight: {1: 10} | 0.33108 | 0.15318 | 0.6655 | 0.24904 | 0.39877 | 2.082 |
| C: 100 gamma: 1<br>class_weight: {1: 2} | 0.81333 | 0.22093 | 0.0475 | 0.07819 | 0.05635 | 4.047 |
| C: 100 gamma: 1<br>class_weight: {1: 5} | 0.69992 | 0.30518 | 0.627 | 0.41054 | 0.5178 | 3.136 |
| C: 100 gamma: 1<br>class_weight: {1: 8} | 0.61025 | 0.23594 | 0.598 | 0.33838 | 0.45757 | 3.286 |
| C: 100 gamma: 1<br>class_weight: {1: 10} | 0.42375 | 0.16349 | 0.597 | 0.25669 | 0.39012 | 3.108 |
| C: 100 gamma: 5<br>class_weight: {1: 5} | 0.75275 | 0.34933 | 0.5605 | 0.43041 | 0.50004 | 3.619 |
| C: 100 gamma: 5<br>class_weight: {1: 8} | 0.63075 | 0.24221 | 0.571 | 0.34013 | 0.44908 | 3.896 |
| C: 100 gamma: 5 | 0.58192 | 0.2183 | 0.5845 | 0.31788 | 0.43766 | 3.882 |

| | | | | | | |
|---|---|---|---|---|---|---|
| class_weight: {1: 10} | | | | | | |
| C: 500 gamma: 0.1 class_weight: {1: 2} | 0.82517 | 0.05455 | 0.003 | 0.00569 | 0.0037 | 15.105 |
| C: 500 gamma: 0.1 class_weight: {1: 5} | 0.66725 | 0.27439 | 0.606 | 0.37775 | 0.48804 | 2.295 |
| C: 500 gamma: 0.1 class_weight: {1: 8} | 0.58975 | 0.22677 | 0.6065 | 0.33011 | 0.45434 | 2.869 |
| C: 500 gamma: 0.1 class_weight: {1: 10} | 0.36908 | 0.15342 | 0.6165 | 0.24569 | 0.38442 | 2.866 |
| C: 500 gamma: 1 class_weight: {1: 2} | 0.8065 | 0.22987 | 0.0685 | 0.10555 | 0.07969 | 8.222 |
| C: 500 gamma: 1 class_weight: {1: 5} | 0.70758 | 0.30952 | 0.613 | 0.41134 | 0.5125 | 7.015 |
| C: 500 gamma: 1 class_weight: {1: 8} | 0.64783 | 0.26034 | 0.6045 | 0.36394 | 0.47809 | 8.578 |
| C: 500 gamma: 1 class_weight: {1: 10} | 0.42317 | 0.16644 | 0.614 | 0.26189 | 0.39927 | 6.838 |
| C: 1000 gamma: 0.1 class_weight: {1: 5} | 0.66967 | 0.27611 | 0.6055 | 0.37927 | 0.48886 | 2.869 |
| C: 1000 gamma: 1 class_weight: {1: 5} | 0.72567 | 0.3269 | 0.61 | 0.42568 | 0.51995 | 12.322 |
| C: 1000 gamma: 5 class_weight: {1: 5} | 0.71983 | 0.30099 | 0.515 | 0.37993 | 0.45088 | 10.925 |
| C: 10000.0 gamma: 0.1 class_weight: {1: 5} | 0.68592 | 0.28915 | 0.6065 | 0.39161 | 0.49733 | 13.286 |
| C: 10000.0 gamma: 1 class_weight: {1: 5} | 0.76875 | 0.37818 | 0.6015 | 0.46439 | 0.53797 | 121.56 |

Green is the best overall, Red is over plotting

**Decision Tree**

| Parameters | Accuracy | Precision | Recall | F1 | F2 | Training and Testing Time (Seconds) |
|---|---|---|---|---|---|---|
| min_samples_split=2 | 0.83133 | 0.6286 | 0.589 | 0.60816 | 0.59652 | 0.613 |
| min_samples_split=5 | 0.83911 | 0.657 | 0.5775 | 0.5775 | 0.61469 | 0.607 |
| min_samples_split=10 | 0.85567 | 0.7123 | 0.588 | 0.64421 | 0.60926 | 0.6 |
| min_samples_split=20 | 0.84856 | 0.65145 | 0.685 | 0.6678 | 0.67802 | 0.585 |
| min_samples_split=50 | 0.80722 | 0.78867 | 0.181 | 0.29443 | 0.21397 | 0.548 |

Green is the best overall