Enron Submission Free-Response Questions
Link to the rubric

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”]

   In 2000, Enron was flying high as one of the largest companies in the US of A. By 2002 though it had abruptly fallen into bankruptcy. During the ensuing Federal witch-hunt, a large amount of confidential information was dispersed into public record (i.e. the internet). This data included tens of thousands of emails and detailed financial data for top executives throughout the company. Udacity has been so kind as to find a dataset which organizes this info into an easy to understand form. This data set includes the following features:

   financial features: ['salary', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options', 'other', 'long_term_incentive', 'restricted_stock', 'director_fees'] (all units are in US dollars)
   email features: ['to_messages', 'email_address', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi']

   This dataset also includes a hand-generated list of 18 persons of interest in the fraud case out of a total of 146 persons in the data set. These persons of interest were selected based upon being indicted, reaching a settlement or plea deal with the government, or testifying in exchange for prosecution immunity.

   The goal of the project was to create a way to accurately predict if an Enron employee is a person of interest and should be investigated for fraud in the scandal. Since the dataset provides a large amount of features about each person and we have a list of persons of interest we can use supervised machine learning to create a discrete output of whether a person should be labeled as a person of interest and potentially burned at the stake.

   I found three distinct outliers in the dataset. A so called person, “TOTAL”, had a ‘salary’ of $26,704,229. This was an extremely high salary, even for the witches practicing dark moneymaking magic at Enron. I found this data point to be an outlier based upon a spreadsheet quirk and removed it. Another outlier I removed based upon being a spreadsheet quirk was “THE TRAVEL AGENCY IN THE PARK” since this is obviously not a person. I also removed the person "LOCKHART EUGENE E" since he only had “NaN” values and added only noise to the dataset.

There was a lot of missing data within this dataset. Table 1 below are the total percent of missing values {i.e “NaN”) for each feature.

Table 1: Percent Total Missing Values for each Feature and Percent of those Missing Features for POI

Name	Percent Missing Value	POI Percent Missing
poi	0%	0%
total_stock_value	13%	0%
total_payments	14%	0%
email_address	22%	0%
restricted_stock	24%	1%
exercised_stock_options	29%	4%
salary	34%	1%
expenses	34%	0%
other	36%	0%
to_messages	40%	3%
from_this_person_to_poi	40%	3%
shared_receipt_with_poi	40%	3%
from_messages	40%	3%
from_poi_to_this_person	40%	3%
bonus	43%	1%
long_term_incentive	55%	4%
deferred_income	66%	5%
deferral_payments	73%	9%
restricted_stock_deferred	88%	13%
director_fees	89%	13%
loan_advances	98%	12%

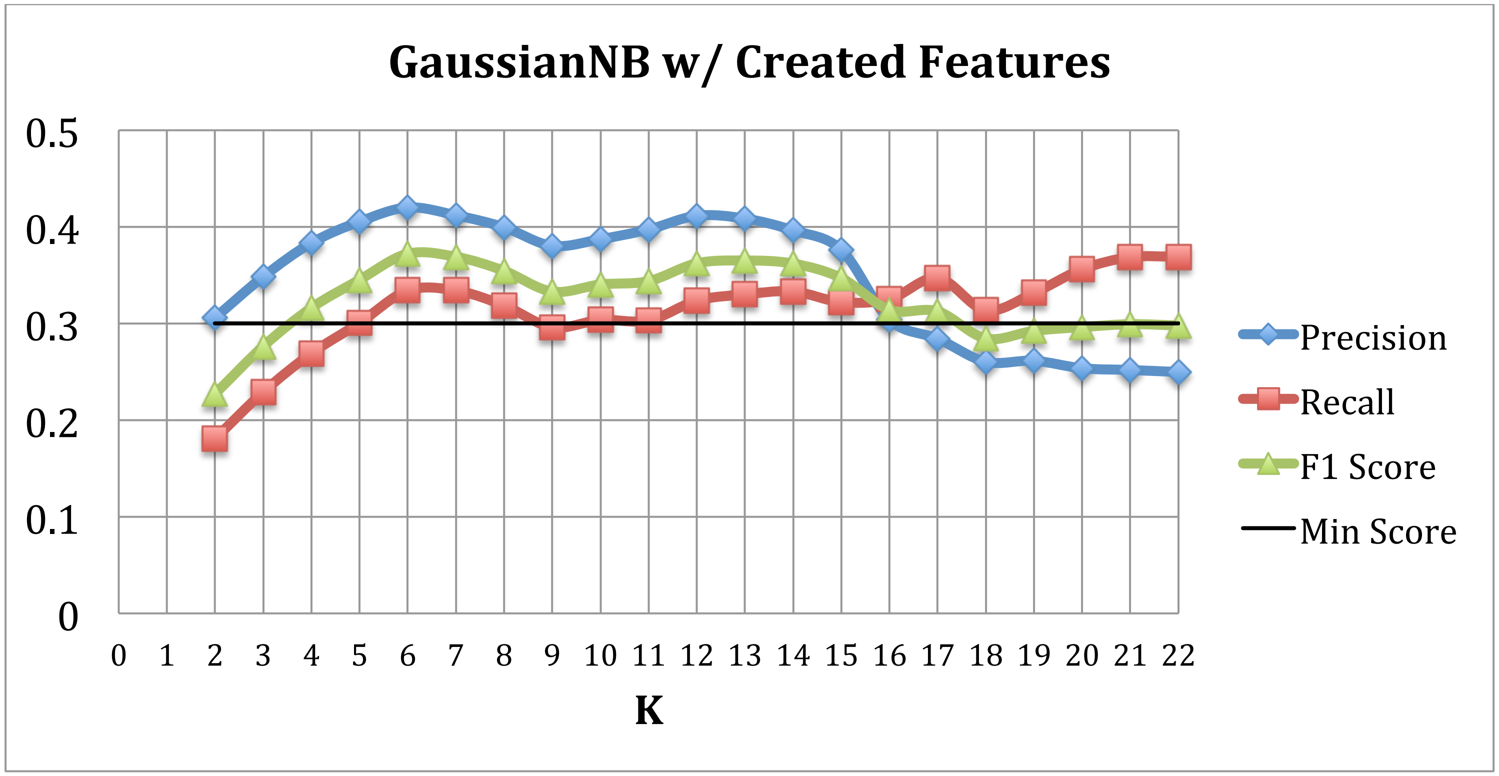
These missing values could lead to problems when applying an algorithm to find persons of interest in the Enron case. For example, about 14% of the 143 persons in the dataset have “NaN” written for their total payments. None of those with “NaN” written for their total payments were POIs. For this reason, a machine-learning algorithm would most likely associate a “NaN” value for total payments with non-POIs. This holds true for the total_stock_value, email_address, expense and other features aswell. I did not remove the loan_advances, director_fees and restricted_stock_deferred from my dataset even though there was a large amount of data missing from these features since I believe only those higher in the organization would be given these benefits. If a large amount of those people were persons of interest than associating having any loan advances with POI would be useful.

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 importance 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”, “properly scale features”, “intelligently select feature”]

   I ended up using the following features in my final POI identifier: ['poi', 'loan_advances', 'deferred_income', 'expenses', 'director_fees', 'fraction_from_poi']. I did not use feature scaling since Naïve Bayes performs feature scaling by design since it is based purely upon the probabilities of each feature. However, I did use Min Max feature scaling for my support vector classifier since this algorithm is based upon creating the smallest distance to a specific classification line and therefore is highly reliant on each features range. The Min Max scalar allow me to scale each variable to a value between 0 and 1. This was needed since some variables had huge ranges (total_stock_value) as compared to other variables (fraction_from_poi).

   I used the select k best algorithm to choose these features. Select k best selects the k best features based on an univariate statistical tests. I applied the select k best for k = 2 through k=22 to find the least amount of features that would produce the best validation results as shown in figure 1. I chose a final k value of 6 since it produced the best validation results with a precision of .41986 and a recall of .334.

Figure 1: Gaussian Naive Bayes Select K Best Validation Outputs w/ New Features Inputted



I created 4 new features, which I added to the dataset. The fraction to poi feature is the fraction of emails a person sent to a person of interest divided by the total amount of emails they sent. The fraction from poi feature is the fraction of emails from poi that were sent to a person divided by the total amount of emails a person received. I created the fraction to poi and fraction from poi since I wanted to get a better understanding of the scale of email contact a person had with a poi. It would be a lot more incriminating if a person sent 30 emails out of 40 emails to poi rather than 30 emails out of 100 emails. I also created the fraction of bonus to salary and fraction of total stock value to salary based upon an assumption that those who helped orchestrate the Enron fraud would likely have lined there pockets with large bonuses and stock options much greater than any normal salary would allow.

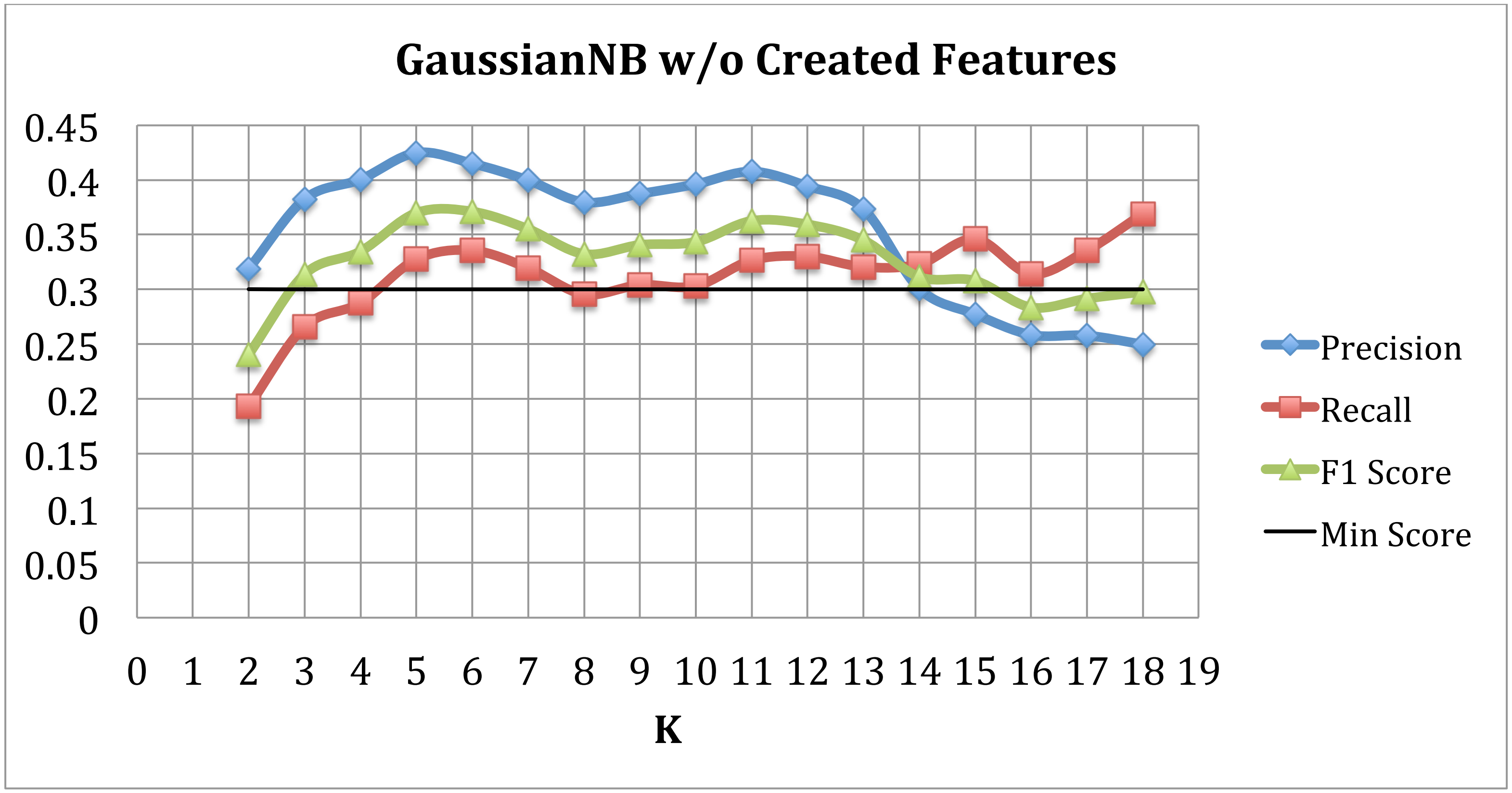
As seen in Table 2, the overall f1 score was higher when deploying the new features in the GaussianNB algorithm with SelectKBest. The f1 score is the great indication of the overall performance of the algorithm since it takes into account not only the precision of the algorithm but also the recall. This means, an algorithm with a high f1 score will most like have a small amount of false person of interest predictions and a high amount true person of interest predictions. For this reason, I chose to use these new features in my final algorithm. As you can see in Figure 2, when k=5 while deploying the GaussianNB without my created features the highest f1 score is observed is still less than when I deploy my new feature.

Table 2: Validation performance comparison with and without newly created features

Algorithm	**Input Features**	**Tuning Parameter Input Ranges in GridSearchCV using scoring=f1**	**Non-Default Tuning Parameter Chosen By GridSearchCV using scoring=f1**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
SelectKBest + GaussianNB without new created features	['poi','salary','deferral_payments','total_payments','loan_advances','bonus','restricted_stock_deferred','deferred_income','total_stock_value','expenses','exercised_stock_options','other','long_term_incentive','restricted_stock','director_fees','total_stock_value','fraction_bonus_	SelectKBest: k= [2-22]	SelectKBest: k=6 Features = ['poi', 'loan_advances', 'deferred_income', 'expenses', 'director_fees', 'fraction_from_poi']	0.84967	0.41986	0.33400	0.37204

	salary','fraction_from_poi','fraction_to_poi','fraction_total_stock_value_salary','from_poi_to_this_person', 'from_messages','from_this_person_to_poi']						
SelectKBest + GaussianNB without new created features	['poi','salary','deferral_payments','total_payments','loan_advances','bonus','restricted_stock_deferred','deferred_income','total_stock_value','expenses','exercised_stock_options','other','long_term_incentive','restricted_stock','director_fees','total_stock_value','from_poi_to_this_person', 'from_messages','from_this_person_to_poi']	SelectKBest: k= [2-18]	SelectKBest: k=5 Features = ['poi', 'loan_advances', 'deferred_income', 'expenses', 'director_fees']	0.85120	0.42477	0.32750	0.36985

Figure 2: Gaussian Naive Bayes Select K Best Validation Outputs w/o New Features Inputted



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 tried the Gaussian Naive Bayes, Decision tree and Support Vector Machine classifier algorithms. To make sure I was using the best features, I applied the SelectKBest feature selection algorithm with each of these algorithms. I also tried these algorithms with and without my newly created features to find out whether these new features increased or decreased the performance for each algorithm. The final algorithm I chose was the GaussianNB since it achieved the highest f1 score and precision score out of all three algorithms as shown in Table 3. Although the precision was slightly better w/o my newly created features, the overall drop in recall was larger then the increase in precision. For this reason, I chose to use the GaussianNB algorithm with my newly created features deployed.

One performance difference that I noted was the support vector machine classifier while underperforming with precision over-performed the GaussianNB algorithm with regards to a higher recall score.

Table 3: Comparison of Performance Between Algorithms Chosen:

Algorithm	**Input Features**	**Tuning Parameter Input Ranges in GridSearchCV using scoring=f1**	**Non-Default Tuning Parameter Chosen By GridSearchCV using scoring=f1**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
SelectKBest + GaussianNB with new created features	['poi','salary','deferral_payments','total_payments','loan_advances','bonus','restricted_stock_deferred','deferred_income','total_stock_value','expenses','exercised_stock_options','other','long_term_incentive','restricted_stock','director_fees','total_stock_value','fraction_bonus_salary','fraction_from_poi','fraction	SelectKBest: k= [2-22]	SelectKBest: k=6 Features = ['poi', 'loan_advances', 'deferred_income', 'expenses', 'director_fees', 'fraction_from_poi']	0.84967	0.41986	0.33400	0.37204

	_to_poi','fraction_total_stock_value_salary','from_poi_to_this_person', 'from_messages','from_this_person_to_poi']						
SelectKBest + GaussianNB without new created features	['poi','salary','deferral_payments','total_payments','loan_advances','bonus','restricted_stock_deferred','deferred_income','total_stock_value','expenses','exercised_stock_options','other','long_term_incentive','restricted_stock','director_fees','total_stock_value','from_poi_to_this_person', 'from_messages','from_this_person_to_poi']	SelectKBest: k= [2-18]	SelectKBest: k=5 Features = ['poi', 'loan_advances', 'deferred_income', 'expenses', 'director_fees']	0.85120	0.42477	0.32750	0.36985
SVC+SelectKBest with new created features	['poi','salary','deferral_payments','total_payments','loan_advances','bonus','restricted_	SVM: C = np.logspace(-2,	SVM: C = 158489.3192461110	0.82733	0.35735	0.3695	0.36332

	stock_deferred','deferred_income','total_stock_value','expenses','exercised_stock_options','other','long_term_incentive','restricted_stock','director_fees','total_stock_value','fraction_bonus_salary','fraction_from_poi','fraction_to_poi','fraction_total_stock_value_salary','from_poi_to_this_person','from_messages','from_this_person_to_poi']	10, 6) Tol = [1.0e-6, 1.0e-3] Gamma = np.logspace(-9, 3, 6) K=[2-22]	9 Tol = 1.0e-6 Gamma = 0.01584893192461111 K= 22				
SVC+Select KBest without new created features	['poi','salary','deferral_payments','total_payments','loan_advances','bonus','restricted_stock_deferred','deferred_income','total_stock_value','expenses','exercised_stock_options','other','long_term_incentive','restricted_stock	SVM: C = np.logspace(-2, 10, 6) Tol = [1.0e-6, 1.0e-3] Gamma = np.logspace(-9,	SVM: C = 158489.31924611109 Tol = 1e-06 Gamma = 0.01584893192461111	0.84773 .	.41003 0.35735	.328 0.3695	.36485 0.36332

	','director_fees','total_stock_value','from_poi_to_this_person', 'from_messages','from_this_person_to_poi']	3, 6) K=range(2,19)	K= 18				
Decision tree + SelectKBest with new created features	['poi','salary','deferral_payments','total_payments','loan_advances','bonus','restricted_stock_deferred','deferred_income','total_stock_value','expenses','exercised_stock_options','other','long_term_incentive','restricted_stock','director_fees','total_stock_value','fraction_bonus_salary','fraction_from_poi','fraction_to_poi','fraction_total_stock_value_salary','from_poi_to_this_person','from_messages','from_this_person_to_poi']	DTC: Criterion :['gini', 'entropy'], Splitter: 'best', 'random'] min_sample_split = range(2,40) SelectKbest: k : [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22]	DTC: Criterion = 'entropy' Splitter = 'random' min_sample_split = 5 SelectKBest: K = 16 K Features = ['poi', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_op	0.83033	0.35262	0.32600	0.33879

			tions', 'other', 'long_term_incentive', 'restricted_stock', 'director_fees', 'fraction_from_poi', 'fraction_total_stock_value_salary']				
Decision tree + SelectKBest without new created features	['poi','salary','deferral_payments','total_payments','loan_advances','bonus','restricted_stock_deferred','deferred_income','total_stock_value','expenses','exercised_stock_options','other','long_term_incentive','restricted_stock','director_fees','total_stock_value','from_poi_to_this_person', 'from_messages','from_this_person_to_poi']	DTC: Criterion :['gini', 'entropy'], Splitter: 'best', 'random'] min_sample_split = range(2,40) SelectKbest: K=range(2,19)	DTC: Criterion = entropy Splitter = random min_sample_split = 9 SelectKBest: K = 17 K Features = ['poi', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_op	0.8162	0.30317	0.2915	0.29722

			tions', 'other', 'long_term_incentive', 'restricted_stock', 'director_fees', 'total_stock_value', 'from_poi_to_this_person', 'from_messages']				

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? (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 item: "tune the algorithm"]

   Tuning an algorithm means changing the parameters to create the most optimized prediction output for your particular set of data. If you do not tune the parameters of an algorithm well you could significantly decrease predictive power of your algorithm. For example, if you under fit your training data with algorithm tuned to be highly biased the algorithms prediction will be to generalized and have a hard time differentiating between different classes in the training set. One can also create algorithms that are tuned with extremely high variance which completely over-fit the training data and then fail to accurately predict in the test set or real world situations.

   My final algorithm was a GaussianNB algorithm, which did not have parameters to tune. On the other hand, I also deployed a support vector machine algorithm classifier. I chose to tune the tolerance, gamma, C parameters. The tol is the tolerance for stopping the criterion. I found that decreasing the tolerance value increased my overall performance. The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. When gamma is to small, the algorithm will be to constrained and will not capture the complexity or "shape" of the data. In other words, the algorithm will be to biased. On the other hand, if gamma is too large, the radius of the area of influence of the support vectors will only include the support vector itself and over fit the data. I used a wide variety of gamma values on a log scale to allow the Gridsearch cross validating algorithm find the best possible gamma fit. I also tuned the C parameter on a log scale using Gridsearch cross validation. The C parameter changes the smoothness of the decision classification surface.

The lower the C value the smoother the decision surface, increasing the bias or generalization of the algorithm. Entering in a large variety of C values allow the Grid Search cross validation function to find the best fitting C value.

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 item: "validation strategy"]

   Validation allows a data analyst to test whether or not a particular algorithm is performing well or is failing to produce useful result. Many people make the classic mistake of running validation testing on the exact same data that they trained their algorithm on. This can lead to much higher performance results during validation testing than what the actual algorithm should produce.

   I used GridSearchCV to find parameter recipe, which produced the highest f1 score. The f1 score is a mixture of the precision and recall and a good overall validation of the performance of ones algorithm.

   I then validated my results by sending my final algorithm through the tester.py file checking the accuracy, recall, precision and f1 score. The tester.py file employs the StratifiedShuffleSplit, which merges the StratifiedKFold and ShuffleSplit. The Shufflesplit function randomly assigns persons from our dataset into a test and training set. The Stratified K fold creates k number of folds which house the same percentage of samples for each class (POI and non-POI). This is to make sure that you cannot accidently have a training set with only one type of class which would greatly reduce the accuracy of your algorithm. Then these training and test folds are fit to the algorithm and averaged to give the overall validation performance of the algorithm.

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"]

   I achieved a precision of about .42 and a recall of about .33 with an overall f1 score of .37 for my final GaussianNB algorithm. I preferred this algorithms performance to all the other algorithms I tested since it had the highest precision while also having a decently high recall score. Precision is equal to the total number of true positives divided by the sum of the true positives and false positives. A true positive is when my algorithm correctly predicts that a person is a POI. On the other hand, a false positive is when my incorrectly algorithm predicts a person to be a POI. I wanted to make sure that my algorithm had the least amount of false positives as possible since this algorithm would potentially be used to find persons of interest for detectives to investigate. In other words, I tuned the algorithm to not incriminate a large amount of people who were not involved in the fraud at all. At the same time this algorithm had a decently high recall score of .33. Recall is the calculated by dividing the total amount of true positives by the sum of the true positives and false negatives. A high recall score mean that the algorithm does not fail to incriminate those who are actually persons of interest. I wanted to have a decently high recall score to make sure that my algorithm was not failing to find a large amount of persons of interest who would then potentially not be investigated for there crimes in the Enron fraud.