1. The goal of this project is to get the best machine learning algorithm to identify ‘persons of interest’ in a fraud investigation. Our dataset consists of 146 data points (representing Enron employees) and their financial and email related activity. Within this dataset there are 18 ‘persons of interest’ and 21 features (plus an ‘email’ address feature which was removed for analysis) and the ‘poi’ identifier LABEL. There were a number of features that had many missing values but I did not pay too much attention to this, figuring that the feature selection algorithm would effectively take this into account. I worked through a series of potential algorithms to test their effectiveness, tweaking features and parameters along the way to get the best result. I gathered together outliers by getting the mean value for each selected feature for POI and non –POI values and then identifying the data points at maximum distance from this mean value at each selected feature. I decided not to even attempt deleting any POI outliers since there were only 18 of them to begin with. But I did delete one max outlier from each of k=8 selected features. After removing these outliers and testing, I noticed that removing the outliers did not improve the best performance of the best classifier, although it did generally improve the classifiers performance – on average. But since the best classifier worked best without deleting these outliers it was decided to keep in the outliers. In gathering together these outliers I did find one value that did not represent an individual employee of Enron. The TOTAL data point – I deleted this data point.
2. But before selecting the best features and deleting the outliers to test performance on all the classifiers (and searching for best hyper parameters in this process as well) I decided it would be best to add two potentially selectable features: ‘ fraction from poi’ and ‘fraction to poi’. These two features basically scaled the emails going to or from a poi by the amount of emails the user sent or received. Which should give us a sense of an Enron employee’s social closeness to a ‘poi’. **I added these newly created features to the original feature set and passed this set** to the SelectKbest feature selection algorithm to find the K best features to use in training and testing all the classifiers I was going to try. Since I did not know the best number of features to use, I looped through to the total number of features to get the very best set. Performance on all the algorithms decreased significantly after about the k=8 feature selection number. I used the test function provided to test the efficacy of the classifiers (paying special attention to the precision and recall rate and the F1 score) to find the best number of features and best features to use. The Decision Tree algorithm worked the best on just one feature selected: ‘exercised\_stock\_options’. So ultimately the added features were not selected to use in the selected classifier. The SelectKBest feature selection algorithm generated the following scores (The highlighted features and scores are the added features): exercised\_stock\_options = 25.09754153

total\_stock\_value = 24.46765405

bonus = 21.0600017

salary = 18.57570327

**fraction\_to\_poi = 16.64170707**

deferred\_income = 11.59554766

long\_term\_incentive = 10.07245453

restricted\_stock = 9.34670079

total\_payments = 8.86672154

shared\_receipt\_with\_poi = 8.74648553

loan\_advances = 7.2427304

expenses = 6.23420114

from\_poi\_to\_this\_person = 5.34494152

other = 4.20497086

**fraction\_from\_poi = 3.21076192**

from\_this\_person\_to\_poi = 2.42650813

director\_fees = 2.10765594

to\_messages = 1.69882435

deferral\_payments = 0.21705893

from\_messages = 0.1641645

restricted\_stock\_deferred = 0.06498431

But before selecting the features and testing , I rescaled all the features. I tried two different scaling algorithms: MinMaxScaler and StandardScaler. Since they performed about the same I wound up just using the StandardScaler.

1. In the end I tested four different machine learning algorithms: Naïve Bayes, K nearest Neighbors, Decision Trees, and Support Vector Machine (SVM). I wound up using the Decision Tree Classifier algorithm since it gave the best recall score and best F1 score. Naïve Bayes worked real well too but I went with the highest scoring classifier.
2. In testing the classifiers, I used GridSearchCV to search through a list of hyper-parameters, where appropriate. For GaussianNB there were no hyper-parameters to search through but for KNN I searched through: {'n\_neighbors': [2, 4, 6, 8, 10], 'weights': ['uniform', 'distance']}. For Decision Trees I searched through {'min\_samples\_split': [ 2, 3, 4, 5, 6, 7, 8]}, for SVM I searched through {'C': [1, 1e3, 1e4], 'gamma': [ 0, 0.1, 0.01, 1, 10]}. As I said above I did a loop to get the K best features for each of the classifiers, using selectKBest algorithm to get the best features in each loop and testing their scores using the test\_classifier function. I did this by using a pipeline to pass the StandardScaler function, the GridSearchCV algorithm for each of the four classifiers and the appropriate hyper-parameters to search through. I changed this a little for Naïve Bayes classifier – since there were no hyper-parameters to search through I just directly passed the classifier into the pipeline. I also changed this a bit for the Decision Tree classifier since it is not necessary to scale the features for this classifier; I passed the GridSearchCV function and the hyper-parameters to search through directly to the test\_classifier function. For K= 1 I got the 'exercised\_stock\_options' features with

NB scores: Precision: 0.60329 Recall: 0.25700 F1: 0.36045

KNN scores: Precision: 0.74836 Recall: 0.45650 F1: 0.56708

DT scores: Precision: 0.59315 Recall: 0.57150 F1: 0.58212

SVM scores: Precision: 0.77851 Recall: 0.17750 F1: 0.28909

For k = 2 I got 'total\_stock\_value', 'exercised\_stock\_options' features

NB scores: Precision: 0.46889 Recall: 0.26750 F1: 0.34066

KNN scores: Precision: 0.46452 Recall: 0.10800 F1: 0.17525

DT scores: Precision: 0.27564 Recall: 0.21100 F1: 0.23903

SVM scores: Precision: 0.67634 Recall: 0.15150 F1: 0.24755

For k = 3 I got 'bonus', 'total\_stock\_value', 'exercised\_stock\_options' features with

NB scores: Precision: 0.48581 Recall: 0.35100 F1: 0.40755

KNN scores: Precision: 0.64312 Recall: 0.25950 F1: 0.36979

DT scores: Precision: 0.36579 Recall: 0.34000 F1: 0.35242

SVM scores: Precision: 0.32266 Recall: 0.06550 F1: 0.10889

For k= 4 I got 'bonus', 'total\_stock\_value', 'salary', 'exercised\_stock\_options'

NB scores: Precision: 0.50312 Recall: 0.32300 F1: 0.39342

KNN scores: Precision: 0.31805 Recall: 0.05550 F1: 0.09451

DT scores: Precision: 0.33244 Recall: 0.31050 F1: 0.32110

SVM scores: Precision: 0.21387 Recall: 0.01850 F1: 0.03405

For k= 5 I got 'bonus', 'total\_stock\_value', 'salary', 'fraction\_to\_poi', 'exercised\_stock\_options'

NB scores: Precision: 0.45558 Recall: 0.30000 F1: 0.36177

KNN scores: Precision: 0.19273 Recall: 0.02650 F1: 0.04659

DT scores: Precision: 0.27501 Recall: 0.27350 F1: 0.27425

SVM scores: Precision: 0.11304 Recall: 0.00650 F1: 0.01229

For k= 6 I got 'deferred\_income', 'bonus', 'total\_stock\_value', 'salary', 'fraction\_to\_poi', 'exercised\_stock\_options'

NB scores: Precision: 0.47723 Recall: 0.35100 F1: 0.40449

KNN scores: Precision: 0.18217 Recall: 0.02350 F1: 0.04163

DT scores: Precision: 0.26474 Recall: 0.22900 F1: 0.24558

SVM scores: Precision: 0.14184 Recall: 0.01000 F1: 0.01868

For k= 7 I got 'deferred\_income', 'long\_term\_incentive', 'bonus', 'total\_stock\_value', 'salary', 'fraction\_to\_poi', 'exercised\_stock\_options'

NB scores: Precision: 0.46376 Recall: 0.36150 F1: 0.40629

KNN scores: Precision: 0.29967 Recall: 0.04600 F1: 0.07976

DT scores: Precision: 0.27751 Recall: 0.23200 F1: 0.25272

SVM scores: Precision: 0.05303 Recall: 0.00350 F1: 0.00657

Etc… The reason we need to tune the hyper-parameters is to optimize the performance of the machine learning algorithm to the complexities of the data. If you do this well then the classifier has the best chance of working well for data points that the classifier was not trained on.

1. Since I used the test\_classifier function provided to test the performance of the classifiers I used the stratifiedShuffleSplit validation strategy. What this does is it splits the indices of the complete data set into training and testing sets randomly. But this isn’t done by taking chunks of the indices of the data set; it’s done by randomly assigning some indices of the data points into a training set and some into a testing set. Where we determine the size of the testing set by passing the argument of the function: test\_size = .3 (that is 30% of the datapoints) and therefore the training set would by .7 or 70% of the data points. Furthermore we do this a number of times where we can specify the number of times where we do this by passing the n\_inter argument in the function. In our case this random splitting, training and testing is done 1,ooo times each time for each classifier and the Accuracy, Precision Recall, FI and F2 are assessed each time. We then get the average of these metrics over the 1000 times we do it. To get our final Metric scores for each classifier. We do this validation so that we don’t train our classifiers on the same datapoints that we test our classifiers on. Since if we did this our classifiers would just need to remember exactly how the datapoint was classified in the training set (this is called overfitting). We want to find out how our classifier is going to behave when it hasn’t seen the data points – will it predict correctly. We specifically want to make sure there is no overlap of these two sets so we know all the data points we encounter when we test were not presented in the training phase.
2. As I said before I used the test\_classifier function which assessed the accuracy, the precision, the recall and F1 and F2 scores. I paid special attention to the precision score which lets us know how often when we say or identify a datapoint is a poi that it actually is: that is, what is the probability that a datapoint is a poi if we predict that it is: true positive /( true positive + false positive). And the recall score which assesses how often when a datapoint is a poi that we identify it as such: that is, what is the probability if a datapoint is a poi that we correctly predict that it is: true positive / (true positive + false negative). Since our main goal was to identify potential POI’s, I thought the recall score was the most important metric. So in choosing a classifier I gave a little more weight to this score than the precision score. I chose a classifier which gave us the highest recall score and F1 score: with the F1 score being the harmonic mean of the precision and recall scores. In our case: Precision: 0.59315 Recall: 0.57150 F1: 0.58212.
3. Of course I used the sklearn website and stack overflow (but this time only sparingly and only for basic python questions) to address any questions I had.