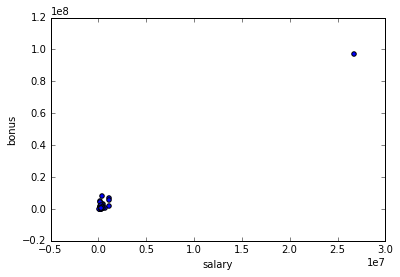
**Dataset**

The dataset contains emails of employees of Enron and a set of 21 features of a total of 146 data points, of which 18 are labeled as persons of interest(POIs). The goal is to use a number of the 21 features to create a predictive model that can identify the persons of interest. The 21 features can be seen below:

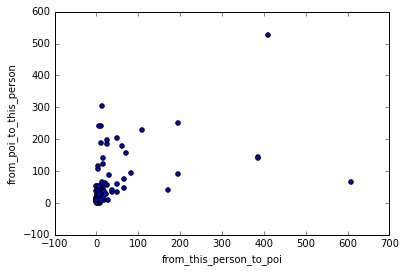
['salary', 'to\_messages', 'deferral\_payments', 'total\_payments', 'exercised\_stock\_options', 'bonus', 'restricted\_stock', 'shared\_receipt\_with\_poi', 'restricted\_stock\_deferred', 'total\_stock\_value', 'expenses', 'loan\_advances', 'from\_messages', 'other', 'from\_this\_person\_to\_poi', 'poi', 'director\_fees', 'deferred\_income', 'long\_term\_incentive', 'email\_address', 'from\_poi\_to\_this\_person']

Plotting two of the features ‘salary’ and ‘bonus’ we see that there is an apparent outlier, which from further investigation is the ‘Total’ data point. As a result I had to delete it.



On a secondary level I looked at the ‘NaN’ entries in the features of the dataset. I discovered that one of the entries that has everything as ‘NaN’ is the ‘THE TRAVEL AGENCY IN THE PARK’ entry.

On a tertiary level we need to check whether any specific data entries have an abnormal amount of emails they received from POIs which may reveal that they are POIs themselves and their features adversely affect the robustness of the analysis. The graphical representation shows 3 outliers, namely ‘LAVORATO JOHN J’, ‘DELAINEY DAVID W' and ‘KEAN STEVEN J’. ‘DELAINEY DAVID W' is included as POI and convicted, whereas ‘LAVORATO JOHN J’ is convicted but not included as POI. ‘KEAN STEVEN J’ was the Chief of Staff, which is why he had such an active email exchange with everyone.



**Features**

Firstly I added ‘LAVORATO JOHN J’ as a POI.

Following I created 3 new aggregate features:

fraction\_from\_poi: Fraction of emails received from POIs.

fraction\_to\_poi: Fraction of emails sent to POIs.

wealth: Salary, total stock value, exercised stock options and bonuses

Then I scaled I scaled all features using the scikit-learn MinMaxScaler to avoid problems caused by different units in the dataset. The algorithm chosen in the end however did not require feature scaling.

Following I added all 21 original and 3 created features to test the ones that were more relevant using the Forrest of Trees and SelectKBest approaches. Two methods yielded equivalent results, but not exactly the same, so let’s see their differences.

| **Feature** | **Score** |
| --- | --- |
| bonus | 0.0937 |
| exercised\_stock\_options | 0.0813 |
| fraction\_to\_poi | 0.0786 |
| total\_stock\_value | 0.0717 |
| deferred\_income | 0.0607 |
| expenses | 0.05766 |
| other | 0.05413 |

Filtering using scikit ExtraTreesClassifier(n\_estimators=250,random\_state=0)

Filtering using scikitlearn SelectKBest

| **Feature** | **Score** |
| --- | --- |
| bonus | 40.58 |
| exercised\_stock\_options | 24.83 |
| total\_stock\_value | 24.00 |
| salary | 9.32 |
| wealth | 17.52 |
| fraction\_to\_poi | 16.18 |
| from\_poi\_to\_this\_person | 16.17 |

**Algorithm selection & Parameter Tuning**

I tested three different classifiers GaussianNB, DecisionTree, AdaBoost. I fine-tuned these parameters performing a scikit-learn GridSearchCV parameter optimization on each of them. Tuning a machine learning algorithm is crucial because different functions and initial settings can have a profound effect on its performance. For example using a higher number of features especially the ones generated from K-Best filtering which seem to be correlated such as bonus, exercised\_stock\_options, total\_stock\_value, wealth, salary, reduces the precision for GaussianNB and especially DecisionTree methodologies as we are over-fitting the data. In some cases, such as selecting a wrong minimum number of samples per leaf in a DecisionTree algorithm, the algorithm can over fit.

In the end I chose the features generated from the Forrest of Trees selection methodology which improve the predictive accuracy and control over-fitting and the GaussianNB as the best evaluation performer because it exhibited Precision(0.51) and Recall(0.36). Precision for this exercise is more important as false positives do not carry major consequences.

**GaussianNB**

<bound method GaussianNB.score of GaussianNB()> GaussianNB()

..........

Precision: 0.399226190476

Recall: 0.352924603175

**DecisionTree**

..........

Precision: 0.274823593074

Recall: 0.288861111111

criterion='entropy',

max\_depth=2,

max\_leaf\_nodes=None,

min\_samples\_leaf=5,

min\_samples\_split=2,

**AdaBoost**

..........

Precision: 0.387952380952

Recall: 0.205948412698

algorithm='SAMME',

learning\_rate=0.5,

n\_estimators=40,

**GaussianNB**

<bound method GaussianNB.score of GaussianNB()> GaussianNB()

..........

Precision: 0.513444444444

Recall: 0.376357142857

**DecisionTree**

..........

Precision: 0.359850649351

Recall: 0.323702380952

criterion='entropy',

max\_depth=None,

max\_leaf\_nodes=None,

min\_samples\_leaf=5,

min\_samples\_split=2,

**AdaBoost**

..........

Precision: 0.49625

Recall: 0.362476190476

algorithm='SAMME.R',

learning\_rate=1.2,

n\_estimators=30,

## Evaluation Metrics and Validation

Validation allows us to assess how well the chosen algorithm generalizes beyond the dataset used to train it. One of the biggest mistakes would have been the usage of the same data for training and testing. To validate the algorithms I iterated 50 randomized trials and assessed mean evaluation metrics.

Two evaluation metrics are used precision and recall, as accuracy would not have been appropriate given the imbalance between POIs and non-POIs. Precision is expressed as the true samples of positive predictions over the sum of all the predictions done (the proportion of individuals identified as POIs, who actually are POIs). Recall on the other hand is defined as the true positives over the sum of all the true samples (proportion of POIs who have successfully been identified). The mean results using GaussianNB:

Precision: 0.5134 Recall: 0.376357

Using the tester.py file cross validation using the StratifiedShuffleSplit method on GaussianNB algorithm returns the following evaluation metrics:

GaussianNB() Accuracy: 0.86100 Precision: 0.47157 Recall: 0.35250 F1: 0.40343 F2: 0.37125 Total predictions: 15000 True positives: 705 False positives: 790 False negatives: 1295 True negatives: 12210