Machine Learning Project: The Enron Case

1. Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it.

The Enron dataset contains emails and financial data from Enron employees. Enron Corporation, was an energy, commodities, and services company that went down in December 2001 as a result of massive accounting fraud. In the years from 2000–2002 more than 1.6 million emails sent and received by Enron executives were released.

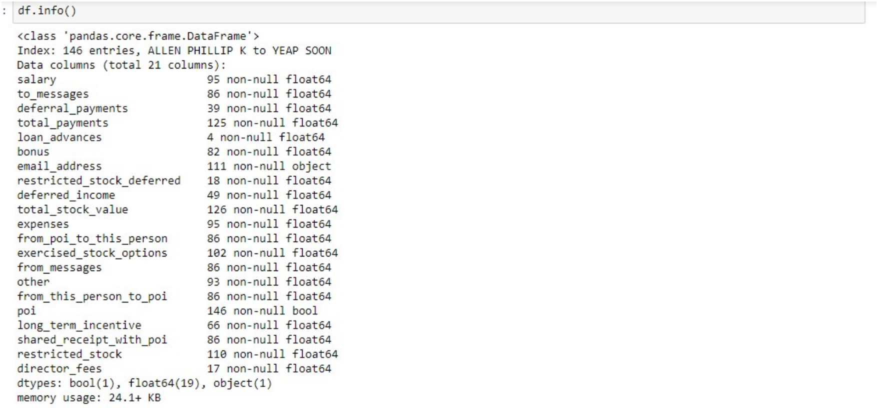
The email + financial data contains the emails themselves, metadata about the emails such as number received by and sent from each individual, and financial information including salary and stock options.

Now machine learning models can be developed that are able to identify persons of interests (POIs), individuals who were involved in fraud or criminal activities during the Enron bankruptcy, based on features on the data.

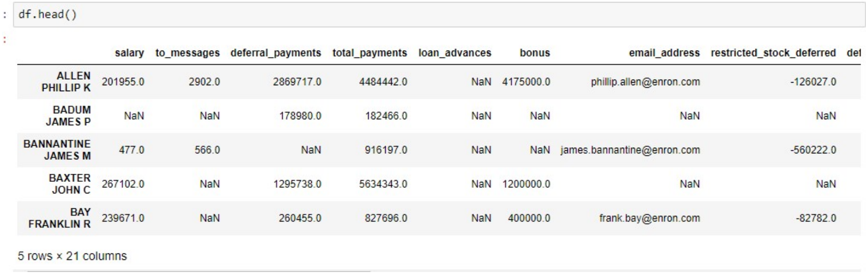
The objective of this project is to train a machine learning model that can find out what features identify a POI.

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?

To get a feel for the data I printed out some data.



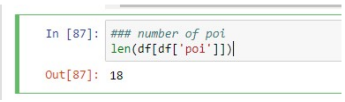
```
: df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 146 entries, ALLEN PHILLIP K to YEAP SOON
Data columns (total 21 columns):
salary                       95 non-null float64
to_messages                  86 non-null float64
deferral_payments            39 non-null float64
total_payments               125 non-null float64
loan_advances                4 non-null float64
bonus                        82 non-null float64
email_address                111 non-null object
restricted_stock_deferred    18 non-null float64
deferred_income              49 non-null float64
total_stock_value            126 non-null float64
expenses                     95 non-null float64
from_poi_to_this_person      86 non-null float64
exercised_stock_options      102 non-null float64
from_messages                86 non-null float64
other                        93 non-null float64
from_this_person_to_poi      86 non-null float64
poi                          146 non-null bool
long_term_incentive          66 non-null float64
shared_receipt_with_poi      86 non-null float64
restricted_stock             110 non-null float64
director_fees                17 non-null float64
dtypes: bool(1), float64(19), object(1)
memory usage: 24.1+ KB
```



```
: df.head()
```

	salary	to_messages	deferral_payments	total_payments	loan_advances	bonus	email_address	restricted_stock_deferred	det
ALLEN PHILLIP K	201955.0	2902.0	2869717.0	4484442.0	NaN	4175000.0	phillip.allen@enron.com	-126027.0	
BADUM JAMES P	NaN	NaN	178980.0	182466.0	NaN	NaN	NaN	NaN	
BANNANTINE JAMES M	477.0	566.0	NaN	916197.0	NaN	NaN	james.bannantine@enron.com	-560222.0	
BAXTER JOHN C	267102.0	NaN	1295738.0	5634343.0	NaN	1200000.0	NaN	NaN	
BAY FRANKLIN R	239671.0	NaN	260455.0	827696.0	NaN	400000.0	frank.bay@enron.com	-82782.0	

5 rows × 21 columns

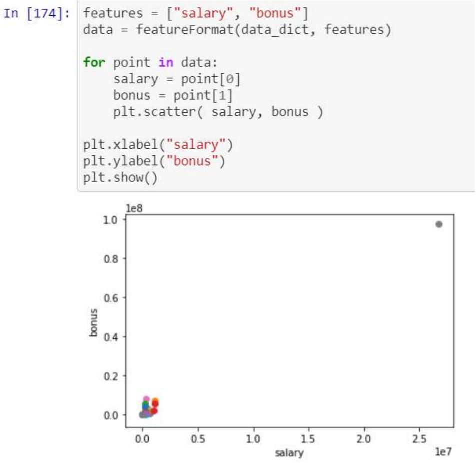


```
In [87]: ### number of poi
         len(df[df['poi']])

Out[87]: 18
```

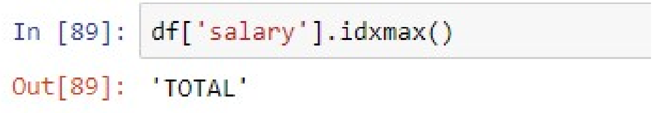
We have in total 146 observations and 21 variables in our dataset - 6 email features, 14 financial features and 1 POI label - and they are divided between 18 POI's and 128 non-POI's.

Now that we got a feel for the dataset, lets visualize the data and check for outliers, especially when plotting salary and bonus.

Now that we got a feel for the dataset, lets visualize the data and check for outliers, especially when plotting salary and bonus.

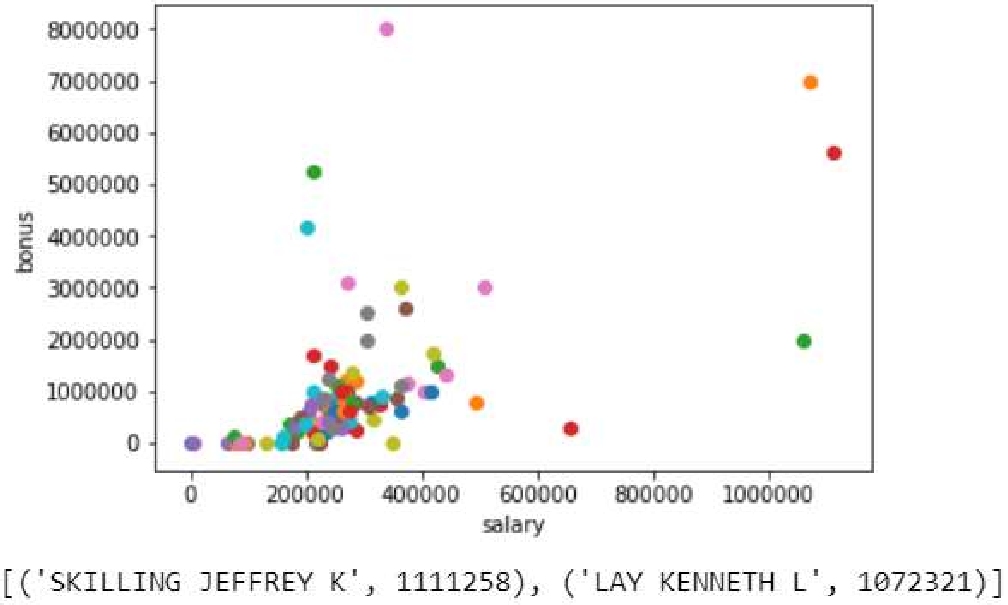


```
In [174]: features = ["salary", "bonus"]
          data = featureFormat(data_dict, features)

          for point in data:
              salary = point[0]
              bonus = point[1]
              plt.scatter( salary, bonus )

          plt.xlabel("salary")
          plt.ylabel("bonus")
          plt.show()
```



```
In [89]: df['salary'].idxmax()

Out[89]: 'TOTAL'
```

We clearly see an outlier, which seems to be the "total".



```
[('SKILLING JEFFREY K', 1111258), ('LAY KENNETH L', 1072321)]
```

After removing the outlier, the plot shows a better picture. We still have some data points with very high bonus and salary values, but this seems ok for C-level positions and possibly POIs.

Adding new features

We already have some features. It is quite possible that POI have relatively high bonus and salary numbers. However, lets create some new features. POIs possible wrote a lot of emails back and forth. Moreover, other potential POIs wrote emails to actual POI and also received emails from POIs. Therefore, we define the features fraction_to_poi_email and fraction_from_poi_email. Seeing how frequently a person communicates with a POI compared to non-POIs is potentially more useful than a raw count. Based on these ideas, the following features are created.

Machine Learning starts here

In this section we will apply machine learning. Specifically, I will first split my data into training and testing data. In machine learning this is common practics to avoid overfitting.

When using the k-fold cross validation method, the training set is split into k smaller sets. 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 (i.e., it is used as a test set to compute a performance measure such as accuracy).The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop.

Now that we have split the data, can can apply an algorithmn. I first applied the decision tree classifier. Decision trees ask multiple linear questions, one after another. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Feature Scaling

I have used decision tree as my final algorithm. Algorithms like decision tree and linear regression don't require feature scaling, whereas Support Vector Machines (SVM) and k-means clustering does.

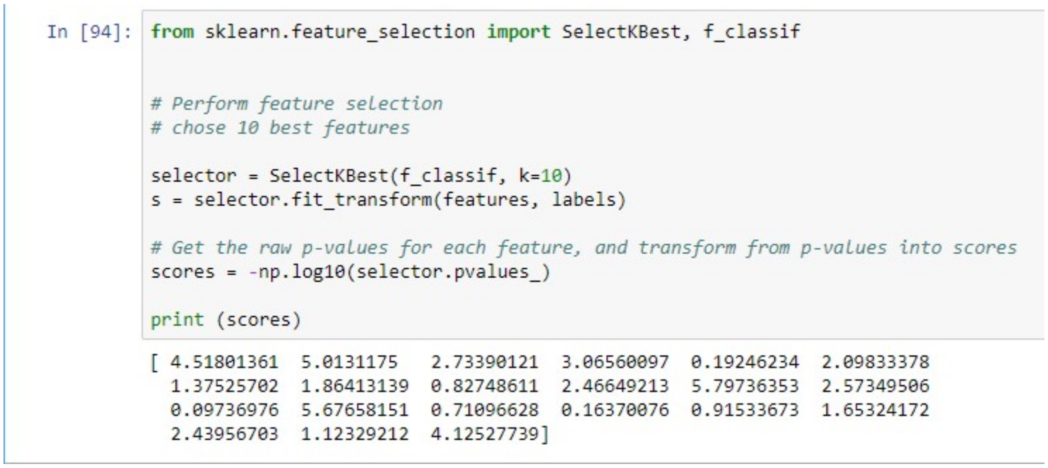
SVM and k-means clustering calculate Euclidean distance between points. If one of the features has a large range, the distance will be governed by this particular feature. These classifiers are affine transformation variant.

In case of linear regression, there is a coefficient for each feature. If a feature has large ranges that do not effect the label, the regression algorithm will make the corresponding coefficients small.

Feature Selection

Having more features does not necessarily mean more Information. We want to have the minimum number of features than can capture the most trends and patterns in our data. We want to get rid of features that do not give us any extra information. The algorithm is going to be as good as the features we put into it.

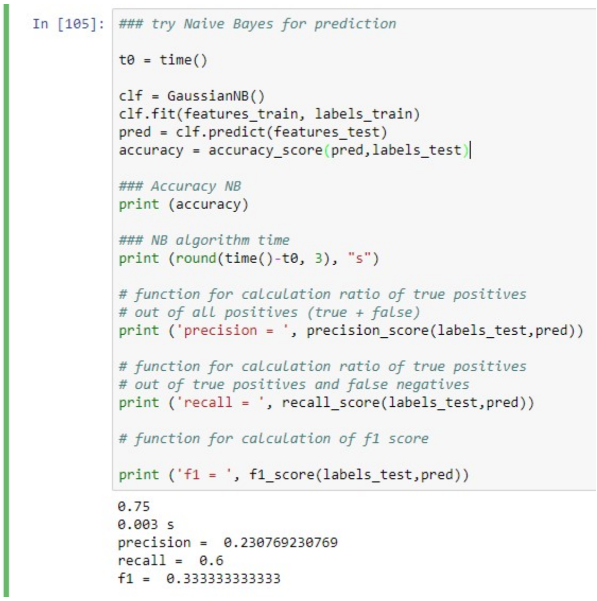
I used SelectKBest on features and selected the 10 best features.



```
In [94]: from sklearn.feature_selection import SelectKBest, f_classif


         # Perform feature selection
         # chose 10 best features

         selector = SelectKBest(f_classif, k=10)
         s = selector.fit_transform(features, labels)

         # Get the raw p-values for each feature, and transform from p-values into scores
         scores = -np.log10(selector.pvalues_)

         print (scores)

         [ 4.51801361  5.0131175   2.73390121  3.06560097  0.19246234  2.09833378
           1.37525702  1.86413139  0.82748611  2.46649213  5.79736353  2.57349506
           0.09736976  5.67658151  0.71096628  0.16370076  0.91533673  1.65324172
           2.43956703  1.12329212  4.12527739]
```

3. What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms?

I first tried a naive bayes classifier. This algorithm did not perform as good as the decision tree (slower. The NB model shows an accuracy score of 0.75, a recall score of 0.6 and a precision score of 0.23. Additionally, I got an f1 score of 0.33.



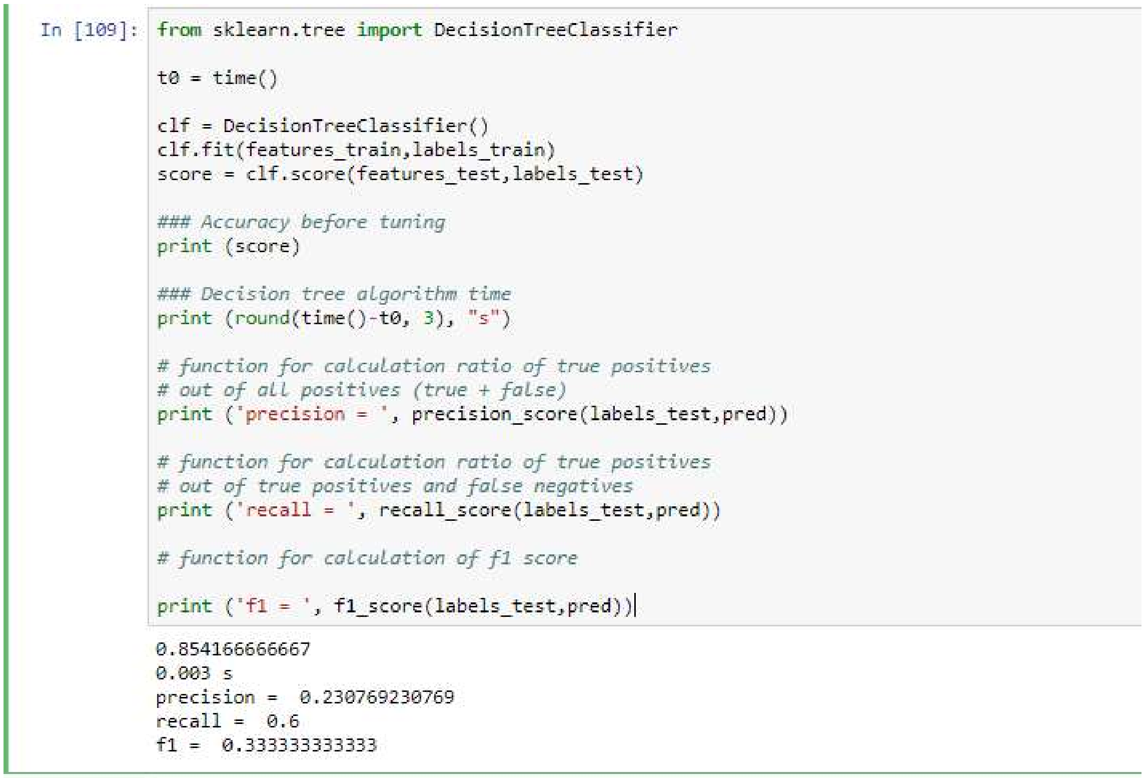
```
In [105]: ### try Naive Bayes for prediction

          t0 = time()

          clf = GaussianNB()
          clf.fit(features_train, labels_train)
          pred = clf.predict(features_test)
          accuracy = accuracy_score(pred,labels_test)

          ### Accuracy NB
          print (accuracy)

          ### NB algorithm time
          print (round(time()-t0, 3), "s")

          # function for calculation ratio of true positives
          # out of all positives (true + false)
          print ('precision = ', precision_score(labels_test,pred))

          # function for calculation ratio of true positives
          # out of true positives and false negatives
          print ('recall = ', recall_score(labels_test,pred))

          # function for calculation of f1 score

          print ('f1 = ', f1_score(labels_test,pred))

          0.75
          0.003 s
          precision =  0.230769230769
          recall =  0.6
          f1 =  0.333333333333
```

In the following I used a decision tree classifier and ranked the 10 features according to their importance.

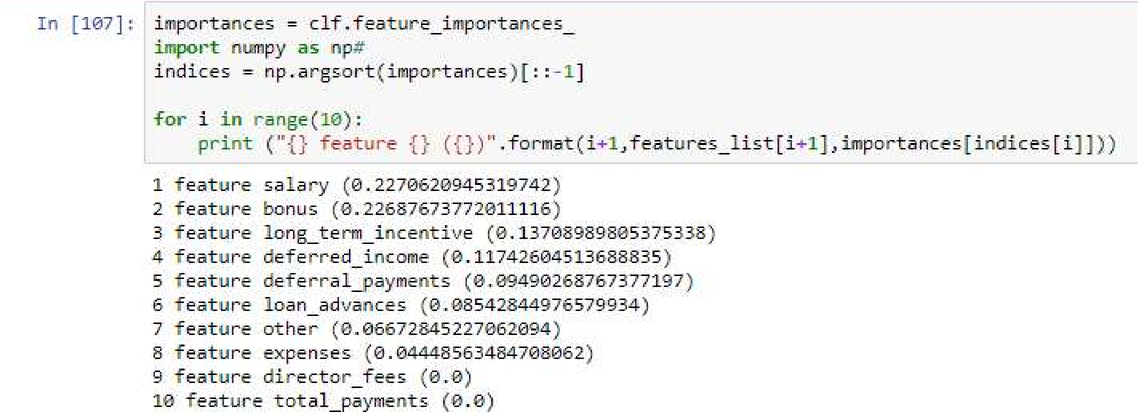
I first started with the default parameters and got an accuracy score of 85.4% after 0.003s. I got a recall score of 0.6 and a precision score of 0.23. Additionally, I got an f1 score of 0.33.

I ranked the feature importance. Salary seems to be the feature of highest importance when identifying POIs (followed by bonus and long term incentive).

Since my decision tree classifier performed a little better, I continued with this one and tuned it.



```
In [109]: from sklearn.tree import DecisionTreeClassifier

          t0 = time()

          clf = DecisionTreeClassifier()
          clf.fit(features_train,labels_train)
          score = clf.score(features_test,labels_test)

          ### Accuracy before tuning
          print (score)

          ### Decision tree algorithm time
          print (round(time()-t0, 3), "s")

          # function for calculation ratio of true positives
          # out of all positives (true + false)
          print ('precision = ', precision_score(labels_test,pred))

          # function for calculation ratio of true positives
          # out of true positives and false negatives
          print ('recall = ', recall_score(labels_test,pred))

          # function for calculation of f1 score

          print ('f1 = ', f1_score(labels_test,pred))

          0.854166666667
          0.003 s
          precision =  0.230769230769
          recall =  0.6
          f1 =  0.333333333333
```

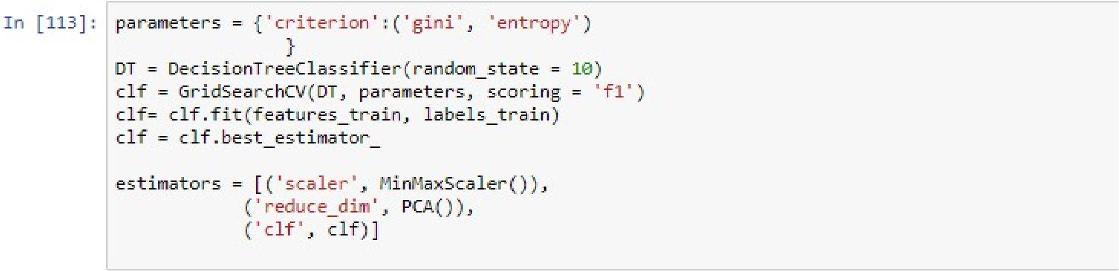


```
In [107]: importances = clf.feature_importances_
          import numpy as np#
          indices = np.argsort(importances)[::-1]

          for i in range(10):
              print ("{} feature {} ({})".format(i+1,features_list[i+1],importances[indices[i]]))

          1 feature salary (0.2270620945319742)
          2 feature bonus (0.22687673772011116)
          3 feature long_term_incentive (0.13708989805375338)
          4 feature deferred_income (0.11742604513688835)
          5 feature deferral_payments (0.09490268767377197)
          6 feature loan_advances (0.08542844976579934)
          7 feature other (0.06672845227062094)
          8 feature expenses (0.04448563484708062)
          9 feature director_fees (0.0)
          10 feature total_payments (0.0)
```

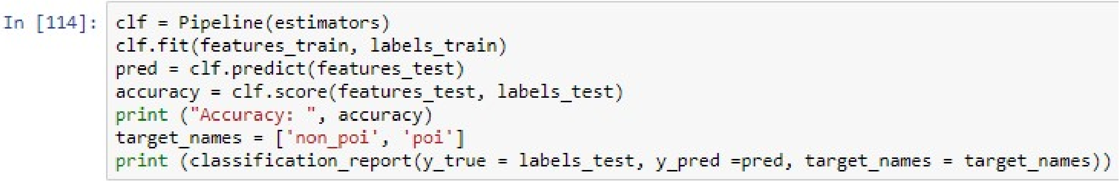
4. What does it mean to tune the parameters of an algorithm, and what can happen if you don’t do this well? How did you tune the parameters of your particular algorithm? What parameters did you tune?

A very important part of machine learning is to adjust the parameters of an algorithm in order to maximize the evaluation metrics and subsequently optimize its performance. If the parameters are not properly tuned, the algorithm can underfit or overfit the data, and thus producing suboptimal results.

In order to tune my decision tree algorithm, I used the GridSearchCV tool provided in scikit learn. It searches for the best parameters between the ones specified in an array of possibilities. The parameters are chosen in order to optimize a chosen scoring function, in my case the f1 score.



```
In [113]: parameters = {'criterion':('gini', 'entropy')
                        }
          DT = DecisionTreeClassifier(random_state = 10)
          clf = GridSearchCV(DT, parameters, scoring = 'f1')
          clf= clf.fit(features_train, labels_train)
          clf = clf.best_estimator_

          estimators = [('scaler', MinMaxScaler()),
                      ('reduce_dim', PCA()),
                      ('clf', clf)]
```



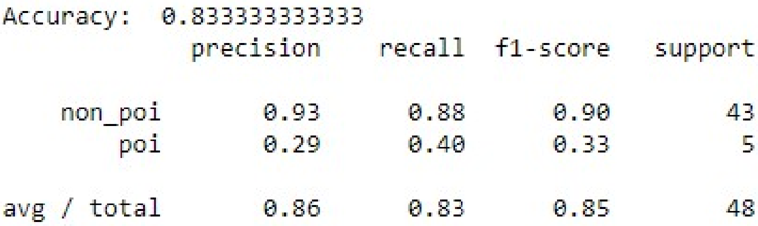
```
In [114]: clf = Pipeline(estimators)
          clf.fit(features_train, labels_train)
          pred = clf.predict(features_test)
          accuracy = clf.score(features_test, labels_test)
          print ("Accuracy: ", accuracy)
          target_names = ['non_poi', 'poi']
          print (classification_report(y_true = labels_test, y_pred =pred, target_names = target_names))
```

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 basically testing your algorithm and its performance on an independent dataset that has not been used for training of the algorithm. A common mistake is to overfit your algorithm. If using PCA transformation only fit the PCA to the training features. PCA.transform however, has to be applied to training AND testing features. DO NOT refit the PCA to the test features.

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.**

I decided to do some tuning to my classifier to achieve better precision, recall and f1 score.



```
Accuracy:  0.833333333333
             precision    recall  f1-score   support

    non_poi       0.93      0.88      0.90        43
        poi       0.29      0.40      0.33         5

avg / total       0.86      0.83      0.85        48
```

Results

My model achieved an overall accuracy score of 83.3%, a precision score of 86% and a recall score of 83%. Additionally, the model has an f1 score of 85%. The classifier beats the .3 both times.

Interpretation

Precision measures the ratio of true positives (meaning a real POI is also predicted to be a POI) out of true positives plus false positives. A high precision states that nearly every time a POI shows up in my test set, I am able to identify him or her.

And recall measures the ratio of true positives in relation to true positives and false negatives.A high recall rate states that I am good at NOT falsely predicting POIs.

F1 is a way of balance precision and recall, and is given by the following formula:

$$F1 = 2 * (precision * recall) / (precision + recall)$$

A good F1 score means both my false positives and false negatives are low, I can identify my POI's reliably and accurately. If my classifier flags a POI then the person is almost certainly a POI, and if the classifier does not flag someone, then they are almost certainly not a POI.