Identify Fraud From Enron Email

September 15, 2017

0.0.1 Sirron Melville

1 Identify Fraud From Enron Email

1.1 Introduction

Enron was a U.S. energy-trading and utilities company that was part of one of the biggest accounting frauds in history. Enron's executives utilized accounting practices that falsely inflated the company's revenues, making the firm the seventh largest corporation in the United States at the height of the scandal. Once it was established that fraud occured, the company quickly unraveled and filed for Chapter 11 bankruptcy on Dec. 2, 2001. In this project, I will investigate the Enron Corpus which is a large database of over 600,000 emails generated by 158 employees of the Enron Corporation and acquired by the Federal Energy Regulatory Commission during its investigation after the company's collapse.

1.2 The Goal of the Project

Machine learning is a method of data analysis that automates analytical model building. I will be using supervised classification algorithms which build a prediction model by training it on a training dataset and the make predictions on a testing dataset. The goal of this project is to use these predictive models on the Enron dataset which contains email and financial data to identify Persons of Interest(POI) that may be involved in the Enron fraud.

In [1]:
```
import sys
import pickle
import random
import matplotlib.pyplot as plt
from time import time
import numpy as np
from numpy import mean
sys.path.append("../tools/")

from sklearn.feature_selection import SelectKBest
from sklearn.preprocessing import MinMaxScaler
from sklearn import cross_validation
from sklearn.cross_validation import train_test_split, StratifiedShuffleSpl
from sklearn.metrics import *
from sklearn.grid_search import GridSearchCV
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn.pipeline import Pipeline


from feature_format import featureFormat, targetFeatureSplit
from tester import dump_classifier_and_data
```

1.3 Overview of Dataset

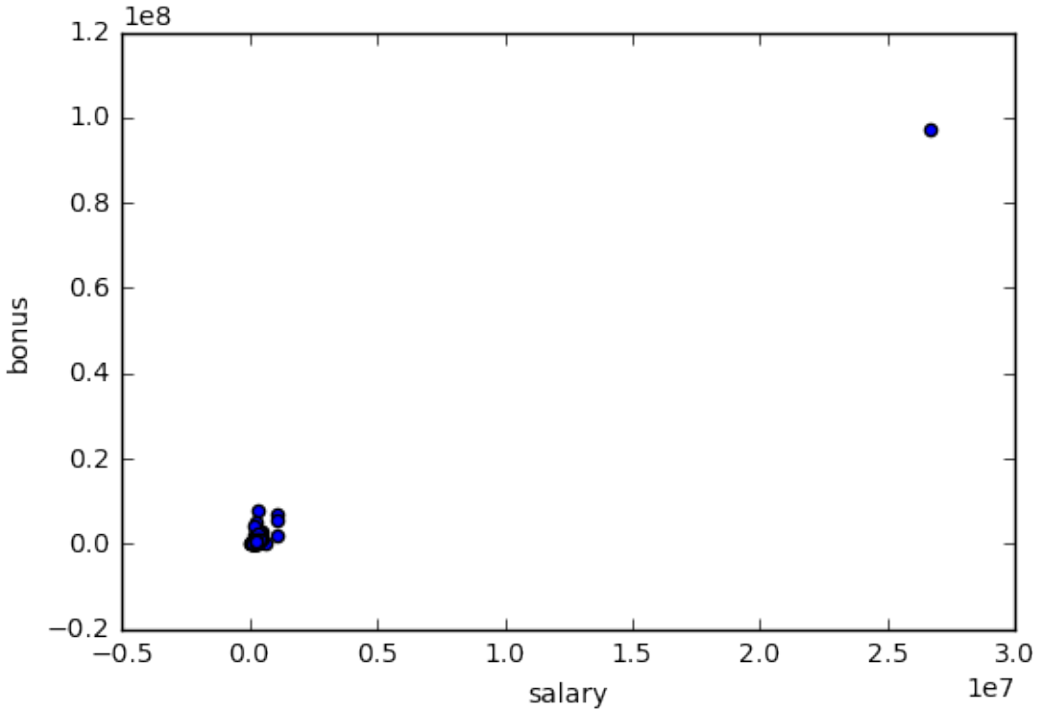
```
In [2]: with open("final_project_dataset.pkl", "r") as data_file:
            data_dict = pickle.load(data_file)

        # Number of people in the dataset
        total_number_people = len(data_dict)
        print 'Number of people in the dataset:', total_number_people

        # Number of POI in the dataset
        num_poi = 0
        for i in data_dict:
            if data_dict[i]['poi']==True:
                num_poi=num_poi+1
        print 'Number of POI in the dataset:', num_poi
        print 'Number of people who are not POI:', len(data_dict)-num_poi


        # features_list is a list of strings, each of which is a feature name.
        # The first feature must be "poi". Excluded are "others" and "email_address
        features_list = ['poi', 'salary', 'deferral_payments', 'total_payments', 'l
                         'bonus', 'restricted_stock_deferred', 'deferred_income', '
                         'expenses', 'exercised_stock_options', 'long_term_incentiv
                         'restricted_stock', 'director_fees', 'to_messages', 'from_
                         'from_messages', 'from_this_person_to_poi', 'shared_receip

        print 'Number of features: ', len(features_list)
```

```
Number of people in the dataset: 146
Number of POI in the dataset: 18
Number of people who are not POI: 128
Number of features:  19
```

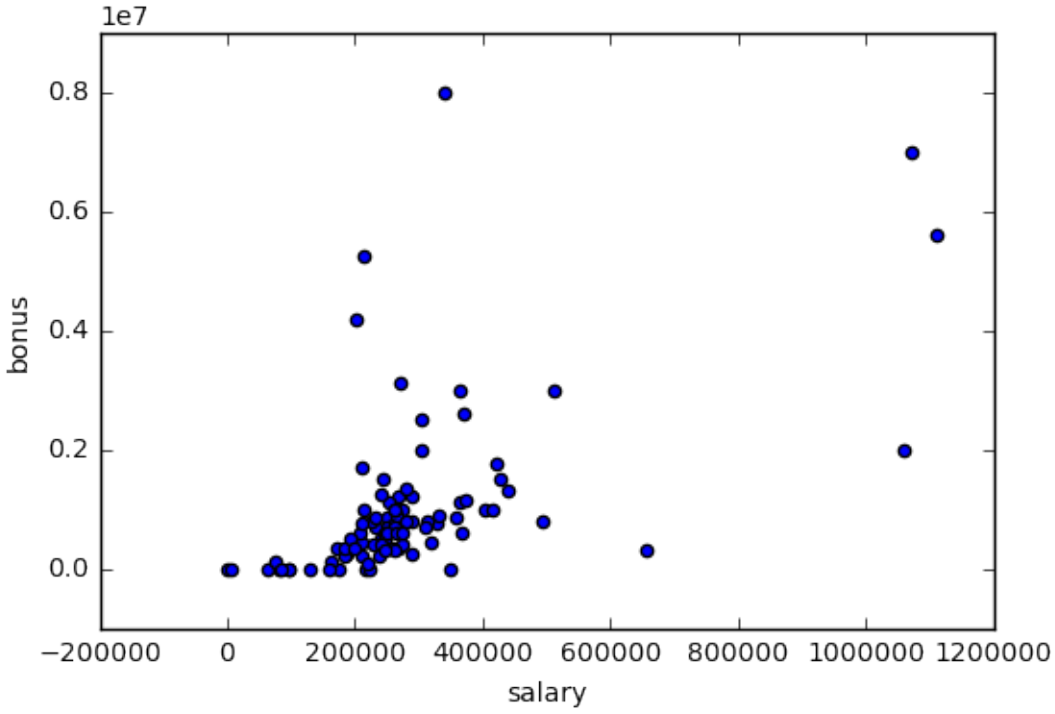
The original dataset contains 146 data points and 21 features. Of these 146 data points, 18 of them are identified as POI. The features are made up of three major categories: POI labels, Email features like “email address” and “to_messages” and financial features like “salary” and “bonus”.

1.4 Outliers

In [3]:
```
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()

for i, j in data_dict.items():
    if j['salary'] != 'NaN' and j['salary'] > 10000000:
        print i
data_dict.pop('TOTAL', 0)
data_dict.pop('THE TRAVEL AGENCY IN THE PARK', 0 )
data_dict.pop('LOCKHART EUGENE E', 0)
print "Number of datapoints after removing outliers is :", len(data_dict)

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()
```



TOTAL
Number of datapoints after removing outliers is : 143



Upon using a scatter plot to visualize the data, I noticed that there was an outlier named "TOTAL" which turned out to be an artifact. There was also name called "THE TRAVEL AGENCY IN THE PARK" which was probably an error. Lastly, "LOCKHART EUGENE E" was removed as this data point only contained NaN values. The dataset was reduced to 143 data points after removing the three outliers.

1.5 Creation of New Features and the Number of Missing Values in Each Feature

```
In [4]: # Creation of new features.
        # Store to my_dataset for easy export below.
        my_dataset = data_dict

        for key, value in my_dataset.iteritems():
            value['from_poi_to_this_person_ratio']=0
            if value['to_messages'] and value['from_poi_to_this_person']!='NaN':
                value['from_poi_to_this_person']=float(value['from_poi_to_this_pers
                value['to_messaages']=float(value['to_messages'])
                if value['from_poi_to_this_person'] > 0:
                    value['from_poi_to_this_person_ratio']=value['from_poi_to_this_

        for key, value in my_dataset.iteritems():
            value['from_this_person_to_poi_ratio']=0
            if value['from_messages'] and value['from_this_person_to_poi']!='NaN':
                value['from_this_person_to_poi']=float(value['from_this_person_to_p
                value['from_messaages']=float(value['from_messages'])
                if value['from_this_person_to_poi'] > 0:
                    value['from_this_person_to_poi_ratio']=value['from_this_person_

        features_list.extend(['from_poi_to_this_person_ratio', 'from_this_person_to

        print "I created two features: 'from_poi_to_this_person_ratio', 'from_this_

        # Number of missing values in each feature.
        nan = [0 for i in range(len(features_list))]
        for i, person in my_dataset.iteritems():
            for j, feature in enumerate(features_list):
                if person[feature] == 'NaN':
                    nan[j] += 1
        for i, feature in enumerate(features_list):
            print feature, nan[i]
```

```
I created two features: 'from_poi_to_this_person_ratio', 'from_this_person_to_poi_r
poi 0
salary 49
deferral_payments 105
total_payments 20
```

```
loan_advances 140
bonus 62
restricted_stock_deferred 126
deferred_income 95
total_stock_value 18
expenses 49
exercised_stock_options 42
long_term_incentive 78
restricted_stock 34
director_fees 127
to_messages 57
from_poi_to_this_person 57
from_messages 57
from_this_person_to_poi 57
shared_receipt_with_poi 57
from_poi_to_this_person_ratio 0
from_this_person_to_poi_ratio 0
```

1.6 Feature Engineering

Two features that were removed are: "email_address" and "other". "email_address" was a string of text corresponding to a person's name and "other" was not pertinent to the analysis and provided no value to the prediction model. However, I created two new features: "from_poi_to_this_person_ratio" which is a measure how frequently a POI sends an email to "this_person" and "from_this_person_to_poi_ratio" which is a measure of how frequently "this_person" sends an email to POI. Later, you will see that after I performed feature selection, "from_this_person_to_poi_ratio" was the only created feature that remained in the analysis.

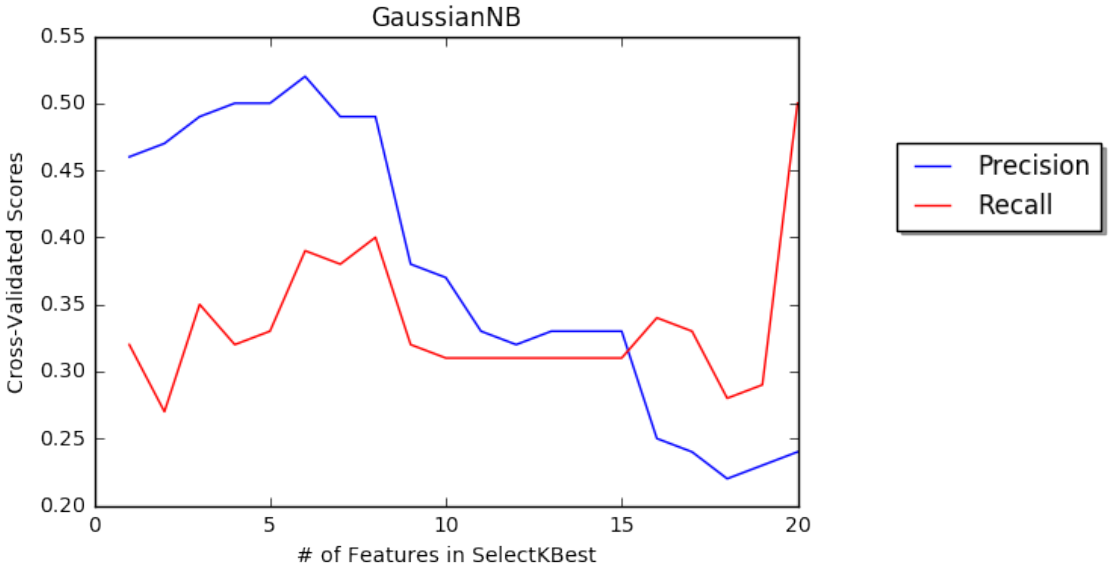
1.7 Feature Selection

I used the manual selection and SelectKBest module from scikit-learn to select the most influential and important features. I then used the MinMaxScaler model to perform feature scaling which ensures that the features are equally weighted before applying them to the supervised classification algorithm classifier. After the creation of two new features, I started with the following features: ['poi', 'salary', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options', 'long_term_incentive', 'restricted_stock', 'director_fees', 'to_messages', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi', 'from_poi_to_this_person_ratio', 'from_this_person_to_poi_ratio'].

I ran SelectKBest for all values from K1 to K20 manually, for GaussianNB, Random Forest and KNN, and recorded the value of KBest, precision and recall. The following plots are visualizations of what I described above.

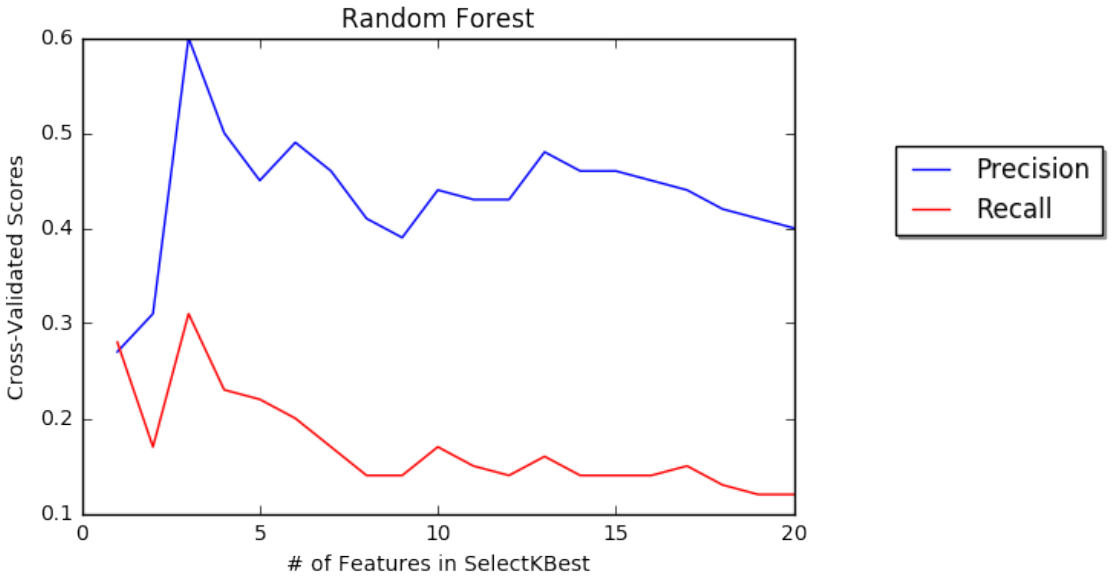
```
In [5]: # Extract the features and labels from the dataset for local testing.
        data = featureFormat(my_dataset, features_list, sort_keys = True)
        labels, features = targetFeatureSplit(data)
```

```
# Scale features via min-max.
scaler = MinMaxScaler()
features = scaler.fit_transform(features)
# Feature selection for GaussianNB.
selection=SelectKBest(k=6)
selection.fit(features, labels)
kbest_gnb = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
precision_gnb =[0.46, 0.47, 0.49, 0.5, 0.5, 0.52, 0.49, 0.49, 0.38, 0.37, 0
                0.33, 0.33, 0.33, 0.25, 0.24, 0.22, 0.23, 0.24]
recall_gnb = [0.32, 0.27, 0.35, 0.32, 0.33, 0.39, 0.38, 0.40, 0.32, 0.31, 0
              0.31, 0.31, 0.31, 0.34, 0.33, 0.28, 0.29, 0.50]

plt.plot(kbest_gnb, precision_gnb, 'b', label='Precision')
plt.plot(kbest_gnb, recall_gnb, 'r', label='Recall')
legend = plt.legend(loc='best', bbox_to_anchor=(1.45, 0.8), shadow=True)
plt.xlabel('# of Features in SelectKBest')
plt.ylabel('Cross-Validated Scores')
plt.title('GaussianNB')
plt.show()
```

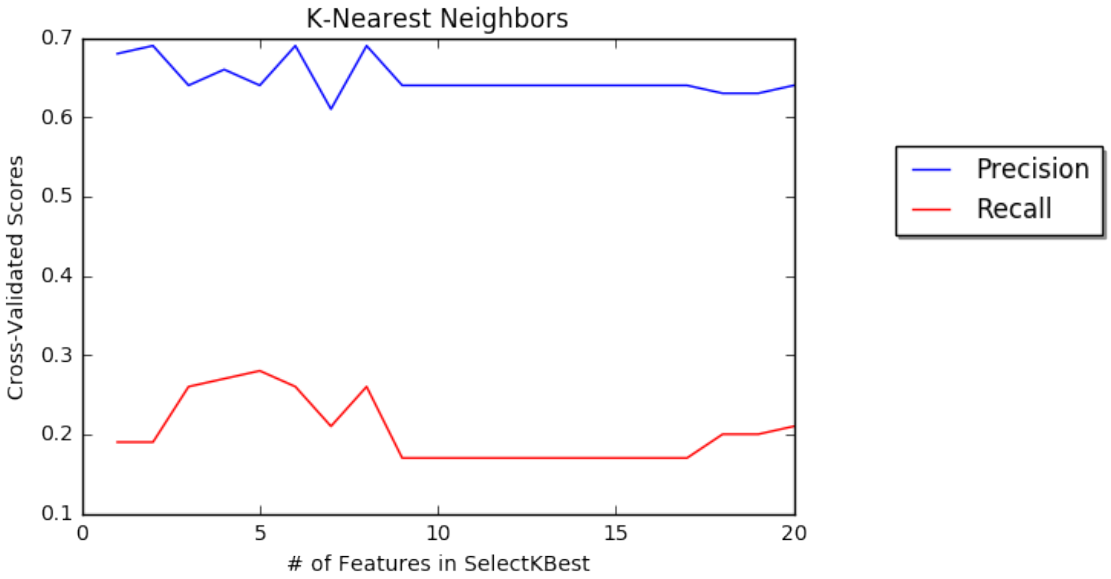


In [6]:
```
# Random Forest feature selection.
kbest_rf=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
precision_rf = [0.27, 0.31, 0.60, 0.50, 0.45, 0.49, 0.46, 0.41, 0.39, 0.44,
                0.48, 0.46, 0.46, 0.45, 0.44, 0.42, 0.41, 0.40]
recall_rf = [0.28, 0.17, 0.31, 0.23, 0.22, 0.20, 0.17, 0.14, 0.14, 0.17, 0.
             0.16, 0.14, 0.14, 0.14, 0.15, 0.13, 0.12, 0.12]
plt.plot(kbest_rf, precision_rf, 'b', label='Precision')
plt.plot(kbest_rf, recall_rf, 'r', label='Recall')
```

```
legend = plt.legend(loc='best', bbox_to_anchor=(1.45, 0.8), shadow=True)
plt.xlabel('# of Features in SelectKBest')
plt.ylabel('Cross-Validated Scores')
plt.title('Random Forest')
plt.show()
```



In [7]:
```
# K Nearest Neighbors feature selection.
kbest_knn=[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
precision_knn = [0.68, 0.69, 0.64, 0.66, 0.64, 0.69, 0.61, 0.69, 0.64, 0.64
                 0.64, 0.64, 0.64, 0.64, 0.64, 0.63, 0.63, 0.64]
recall_knn = [0.19, 0.19, 0.26, 0.27, 0.28, 0.26, 0.21, 0.26, 0.17, 0.17, 0
              0.17, 0.17, 0.17, 0.17, 0.17, 0.2, 0.2, 0.21]
plt.plot(kbest_knn, precision_knn, 'b', label='Precision')
plt.plot(kbest_knn, recall_knn, 'r', label='Recall')
legend = plt.legend(loc='best', bbox_to_anchor=(1.45, 0.8), shadow=True)
plt.xlabel('# of Features in SelectKBest')
plt.ylabel('Cross-Validated Scores')
plt.title('K-Nearest Neighbors')
plt.show()
```



1.8 Best Features

```
In [8]: # Scores of the features with the highest value of k.
        results = zip(selection.get_support(), features_list[1:], selection.scores_
        results = sorted(results, key=lambda x: x[2], reverse=True)
        for element in results:
            print "K-best features:", element
```

```
K-best features: (True, 'exercised_stock_options', 24.815079733218194)
K-best features: (True, 'total_stock_value', 24.182898678566872)
K-best features: (True, 'bonus', 20.792252047181538)
K-best features: (True, 'salary', 18.289684043404513)
K-best features: (True, 'from_this_person_to_poi_ratio', 16.409712548035792)
K-best features: (True, 'deferred_income', 11.458476579280697)
K-best features: (False, 'long_term_incentive', 9.9221860131898385)
K-best features: (False, 'restricted_stock', 9.212810621977086)
K-best features: (False, 'total_payments', 8.7727777300916809)
K-best features: (False, 'shared_receipt_with_poi', 8.5894207316823774)
K-best features: (False, 'loan_advances', 7.1840556582887247)
K-best features: (False, 'expenses', 6.0941733106389666)
K-best features: (False, 'from_poi_to_this_person', 5.2434497133749574)
K-best features: (False, 'from_poi_to_this_person_ratio', 3.128091748156737)
K-best features: (False, 'from_this_person_to_poi', 2.3826121082276743)
K-best features: (False, 'director_fees', 2.126327802007705)
K-best features: (False, 'to_messages', 1.6463411294420094)
K-best features: (False, 'deferral_payments', 0.22461127473600509)
K-best features: (False, 'from_messages', 0.16970094762175436)
```

```
K-best features: (False, 'restricted_stock_deferred', 0.065499652909891237)
```

1.9 Update Features List That Was Chosen Manually and by SelectKBest.

```
In [9]: # Update features list that was chosen manually and by SelectKBest.
        features_list = ['poi', 'exercised_stock_options', 'total_stock_value', 'bo
                         'salary', 'from_this_person_to_poi_ratio', 'deferred_incom
        data = featureFormat(my_dataset, features_list, sort_keys = True)
        labels, features = targetFeatureSplit(data)

        features_train, features_test, labels_train, labels_test = train_test_split
```

1.10 Tuning of Classifiers to Confirm Best K

```
In [10]: # Potential pipeline steps.
         scaler = MinMaxScaler()
         select = SelectKBest()
         gnb = GaussianNB()
         rf = RandomForestClassifier()
         knc = KNeighborsClassifier()

         # Load pipeline steps into list.
         steps = [
                  # Preprocessing
                  # ('min_max_scaler', scaler),

                  # Feature selection
                  ('feature_selection', select),

                  # Classifier
                  ('gnb', gnb)
                  #('rf', rf)
                  # ('knc', knc)
                  ]

         # Create pipeline.
         pipeline = Pipeline(steps)

         # Parameters to try in grid search.
         parameters = dict(
                           feature_selection__k=[2, 3, 5, 6],
                          # rf__criterion=['gini', 'entropy'],
                          # rf__max_depth=[None, 1, 2, 3, 4],
                          # rf__min_samples_split=[1, 2, 3, 4, 25],
                          # rf__min_samples_leaf=[1, 2, 3, 4],
                          # rf__min_weight_fraction_leaf=[0, 0.25, 0.5],
                          # rf__class_weight=[None, 'balanced'],
```

```
                        # rf__random_state=[42]
                        # knc__n_neighbors=[1, 2, 3, 4, 5],
                        # knc__leaf_size=[1, 10, 30, 60],
                        # knc__algorithm=['auto', 'ball_tree', 'kd_tree', 'brute
                        )
        # Create training sets and test sets.
        features_train, features_test, labels_train, labels_test = \
            train_test_split(features, labels, test_size=0.3, random_state=42)

        # Cross-validation for parameter tuning in grid search.
        sss = StratifiedShuffleSplit(
            labels_train,
            n_iter = 20,
            test_size = 0.5,
            random_state = 0
            )

        # Create, fit, and make predictions with grid search.
        gs = GridSearchCV(pipeline,
                          param_grid=parameters,
                          scoring="f1",
                          cv=sss,
                          error_score=0)
        gs.fit(features_train, labels_train)
        labels_predictions = gs.predict(features_test)

        # Pick the classifier with the best tuned parameters.
        clf = gs.best_estimator_
        print "\n", "Best parameters are: ", gs.best_params_, "\n"


Best parameters are:  {'feature_selection__k': 6}



/Users/Sirron/anaconda/envs/DAND/lib/python2.7/site-packages/sklearn/metrics/classi
  'precision', 'predicted', average, warn_for)
```

As you can see, this is just a confirmation that the best feature selection for the Gaussian Naive Bayes classifier is k = 6.

1.11 Selected Algorithm

I tested three algorithms manually selecting from k1 to k20. The best value of k for the Gaussian Naive Bayes classifier ended up being k=6. With the GaussianNB, the precision is 0.5157 and the recall is 0.3855, using the following feature set: ['poi', 'exercised_stock_options', 'total_stock_value', 'bonus', 'salary', 'from_this_person_to_poi_ratio', 'deferred_income']. I decided to go with the Gaussian NB as it's the best performer because it has the better recall compared

to the other two tested classifiers. The best result for the Random Forest classifier is a precision of 0.60 and a recall of 0.31 with the following feature set: ['poi', 'exercised_stock_options', 'total_stock_value', 'bonus']. The best result for K-Nearest Neighbors is a precision of 0.64 and a recall of 0.28 with the following feature set: ['poi', 'exercised_stock_options', 'total_stock_value', 'bonus', 'salary', 'from_this_person_to_poi_ratio'].

1.12 Conclusion

The algorithm that I used was the GaussianNB and it did not have parameters that I needed to tune. The process of tuning the parameters of an algorithm permits the configuration of the model and allows the algorithm to perform at its best. E.g. when using K-Means, the number of clusters in a dataset must be specified, else there can be issues of over-fitting or under-fitting. Tuning an algorithm can be a very tedious process. The GaussianNB doesn't require the parameter to be tuned. I compared the parameter tuning of the k-NN and Random Forest algorithms to the GaussianNB. Tuning the parameters for k-Nearest Neighbors using GridSearchCV and the best feature set: ("p" = [2,3], "n_neighbors" = [2, 5]) The precision is 0.67 and the recall is 0.24, with the tune, the precision incresed but the recall decreased. Tuning the parameters of the Random Forest classifier using the best feature set: ("criterion" = ('gini', 'entropy'), "n_estimators" = [2, 3, 5]) The precision is 0.50 and the recall is 0.27. These results still aren't as good as the GaussianNB, I think part of the reason is that the GaussianNB is better for text classification and datasets with big feature spaces.

Validation is the process of deciding whether the numerical results quantifying hypothesized relationships between variables, are acceptable as descriptions of the data. Cross-validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent dataset. The goal of cross-validation is to define a dataset to "test" the model in the training phase (validation dataset). The validation dataset is a sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration. In supervised classification, the goal is to use trained models to make reliable predictions on independent datatsets. A major mistake is overfitting, this is when a machine learning algorithm performs very well on the training dataset but fails to predict accurately on new or unseen data. In order to prevent overfitting, it is very important to split the data into a training set and a validation set.
The model was cross-validated using the Stratified Shuffle Split which randomly creates training and test datasets. This technique is used because the dataset is small and skewed towards non-POIs. If a technique that accounts for the that is not used, the real potential of the algorithm in terms of performance metrics would not be able to be assessed in the validation phase. Stratification (preservation of the percentage of samples for each class) is used to achieve robustness in a dataset with the aforementioned limitations because the chance of randomly splitting skewed and non representative validation subsets could be high.

The main evaluation metrics for the model (GaussianNB) I used were precision and recall. For the GaussianNB classifier, the precision was 0.5157 and the recall was 0.3855. Intuitively, precision is how many of the returned hits were true positive i.e. how many of the returned hits were correct. Recall is how many of the true positives were found i.e. how many correct hits were found. In the case of the Enron corpus, if my model predicts that a person is a POI, roughly 51% of time this person is actually a POI. My model can identify a POI correctly about 38% of the time.

Notebook Created By: Sirron Melville