Identify Fraud from Enron Email

By: Ming Ho

July 8th, 2018

Understanding the Dataset

Dataset and Project Summary

The goal of this project is to build a supervised learning algorithm to identify employees who may have been involved in one of the biggest corporate fraud scandals in recent U.S history, the Enron scandal. Who was involved in causing the downfall of the energy conglomerate that claimed a revenue of $101 billion in 2000?

The dataset used for analysis consists of using financial and email data of employees and top executives that was made public by federal investigations after Enron's collapse. The dataset will be compared against a hand-made list of persons of interest in the scandal in order to build a persons of interest identifier. The list contains individuals who were either indicted, reached a settlement or plea deal with the government, or testified in exchange for immunity from prosecution. Machine learning is great for this project as we already have a accurate list of known people of interest and a good amount of data for the people in question (top executives). I can build a model utilizing supervised learning classification algorithms to analyze the available information for each person and predict if the person should be classified as a person of interest. Then, I can compare it with the list of actual persons of interest and evaluate how accurate the model is.

There were a few outliers in the dataset. There was a record with a business name, a record for a person with no data, and a record that included a grand total of available data for all employees. I decided to remove those records as they were skewing the dataset.

Data Exploration

In [79]:
```
import sys
import pickle
sys.path.append("../tools/")

from feature_format import featureFormat, targetFeatureSplit
from tester import dump_classifier_and_data
import matplotlib.pyplot
import numpy as np
import pandas as pd

### Task 1: Select what features you'll use.
### features_list is a list of strings, each of which is a feature name.
### The first feature must be "poi".
features_list = ['poi', 'salary', 'to_messages', 'deferral_payments', 'total_p
ayments', 'exercised_stock_options', 'bonus',
                 'restricted_stock', 'shared_receipt_with_poi', 'restricted_st
ock_deferred', 'total_stock_value', 'expenses',
                 'loan_advances', 'from_messages', 'other', 'from_this_person_
to_poi', 'director_fees',
                 'deferred_income', 'long_term_incentive', 'from_poi_to_this_p
erson']

### Load the dictionary containing the dataset
with open("final_project_dataset.pkl", "r") as data_file:
    data_dict = pickle.load(data_file)

print "Total number of people in dataset : ", sum([1 for key in data_dict.keys
()])
print "Total number of POI : ", sum([1 for key in data_dict.keys() if data_dic
t[key]['poi']])
print "Total number of non-POI : ", sum([1 for key in data_dict.keys() if not
data_dict[key]['poi']])

def count_NaN_features(feature):
    print feature, sum([1 for key in data_dict.keys() if data_dict[key][featur
e] == 'NaN' or type(data_dict[key][feature]) == None])

#print "Total number of data points with missing values :"
for j in features_list:
    print count_NaN_features(j)
```

```
Total number of people in dataset :  146
Total number of POI :  18
Total number of non-POI :  128
poi 0
None
salary 51
None
to_messages 60
None
deferral_payments 107
None
total_payments 21
None
exercised_stock_options 44
None
bonus 64
None
restricted_stock 36
None
shared_receipt_with_poi 60
None
restricted_stock_deferred 128
None
total_stock_value 20
None
expenses 51
None
loan_advances 142
None
from_messages 60
None
other 53
None
from_this_person_to_poi 60
None
director_fees 129
None
deferred_income 97
None
long_term_incentive 80
None
from_poi_to_this_person 60
None
```

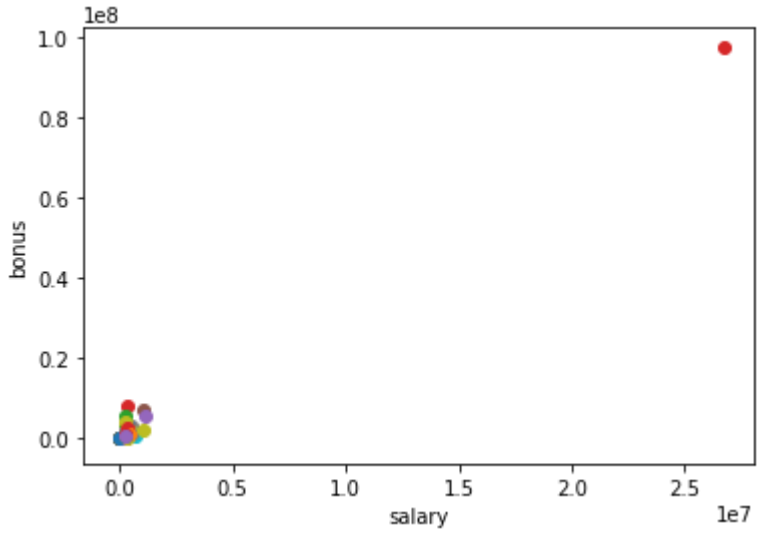
The data is not very clean. There is a significant amount of missing values (NaN) and some features have missing values for more than 50% of the dataset. I will handle missing values later. For now, let's start by looking for outliers.

Outliers

I'd imagine that if a person is a POI, they would have higher salaries and/or bonuses than others. One of the easiest ways to find outliers is by visualization.

In [80]:
```
data = featureFormat(data_dict, features_list)
for point in data:
    salary = point[1]
    bonus = point[6]
    matplotlib.pyplot.scatter(salary, bonus)

matplotlib.pyplot.xlabel("salary")
matplotlib.pyplot.ylabel("bonus")
matplotlib.pyplot.show()
```



There seems to be one outlier with a very high salary of 2.5+ million and a bonus of around 1 million. Let's take a closer look.

In [81]:
```
for k,v in data_dict.iteritems():
    if data_dict[k]['bonus'] > 700000 and data_dict[k]['salary'] > 2000000 and
 data_dict[k]['salary'] != 'NaN' \
    and data_dict[k]['bonus'] != 'NaN':
        print k, 'salary :', data_dict[k]['salary'] , 'bonus :',data_dict[k][
'bonus']
```

```
TOTAL salary : 26704229 bonus : 97343619
```

The name of the person is TOTAL, which is actually the total sum of all data values for every person. Therefore, it shouldn't be included in the data set.

In [82]:
```
# for each employee, print the number of features with the value of NaN if it
 is unusually high. Total number of features is 20.
for k,v in data_dict.iteritems():
    count = 0
    for i,j in v.iteritems():
        # change NaN to 0
        if j == 'NaN' or type(j) == None:
            count +=1
            data_dict[k][i] = 0
    if count > 17:
        print k, count
```

```
WHALEY DAVID A 18
WROBEL BRUCE 18
LOCKHART EUGENE E 20
THE TRAVEL AGENCY IN THE PARK 18
GRAMM WENDY L 18
```

In [83]:
```
print 'WHALEY DAVID A', data_dict['WHALEY DAVID A']
print 'WROBEL BRUCE', data_dict['WROBEL BRUCE']
print 'LOCKHART EUGENE E', data_dict['LOCKHART EUGENE E']
print 'THE TRAVEL AGENCY IN THE PARK', data_dict['THE TRAVEL AGENCY IN THE PAR
K']
print 'GRAMM WENDY L', data_dict['GRAMM WENDY L']
```

```
WHALEY DAVID A {'salary': 0, 'to_messages': 0, 'deferral_payments': 0, 'total
_payments': 0, 'exercised_stock_options': 98718, 'bonus': 0, 'restricted_stoc
k': 0, 'shared_receipt_with_poi': 0, 'restricted_stock_deferred': 0, 'total_s
tock_value': 98718, 'expenses': 0, 'loan_advances': 0, 'from_messages': 0, 'o
ther': 0, 'from_this_person_to_poi': 0, 'poi': False, 'director_fees': 0, 'de
ferred_income': 0, 'long_term_incentive': 0, 'email_address': 0, 'from_poi_to
_this_person': 0}
WROBEL BRUCE {'salary': 0, 'to_messages': 0, 'deferral_payments': 0, 'total_p
ayments': 0, 'exercised_stock_options': 139130, 'bonus': 0, 'restricted_stoc
k': 0, 'shared_receipt_with_poi': 0, 'restricted_stock_deferred': 0, 'total_s
tock_value': 139130, 'expenses': 0, 'loan_advances': 0, 'from_messages': 0,
'other': 0, 'from_this_person_to_poi': 0, 'poi': False, 'director_fees': 0,
'deferred_income': 0, 'long_term_incentive': 0, 'email_address': 0, 'from_poi
_to_this_person': 0}
LOCKHART EUGENE E {'salary': 0, 'to_messages': 0, 'deferral_payments': 0, 'to
tal_payments': 0, 'exercised_stock_options': 0, 'bonus': 0, 'restricted_stoc
k': 0, 'shared_receipt_with_poi': 0, 'restricted_stock_deferred': 0, 'total_s
tock_value': 0, 'expenses': 0, 'loan_advances': 0, 'from_messages': 0, 'othe
r': 0, 'from_this_person_to_poi': 0, 'poi': False, 'director_fees': 0, 'defer
red_income': 0, 'long_term_incentive': 0, 'email_address': 0, 'from_poi_to_th
is_person': 0}
THE TRAVEL AGENCY IN THE PARK {'salary': 0, 'to_messages': 0, 'deferral_payme
nts': 0, 'total_payments': 362096, 'exercised_stock_options': 0, 'bonus': 0,
'restricted_stock': 0, 'shared_receipt_with_poi': 0, 'restricted_stock_deferr
ed': 0, 'total_stock_value': 0, 'expenses': 0, 'loan_advances': 0, 'from_mess
ages': 0, 'other': 362096, 'from_this_person_to_poi': 0, 'poi': False, 'direc
tor_fees': 0, 'deferred_income': 0, 'long_term_incentive': 0, 'email_addres
s': 0, 'from_poi_to_this_person': 0}
GRAMM WENDY L {'salary': 0, 'to_messages': 0, 'deferral_payments': 0, 'total_
payments': 119292, 'exercised_stock_options': 0, 'bonus': 0, 'restricted_stoc
k': 0, 'shared_receipt_with_poi': 0, 'restricted_stock_deferred': 0, 'total_s
tock_value': 0, 'expenses': 0, 'loan_advances': 0, 'from_messages': 0, 'othe
r': 0, 'from_this_person_to_poi': 0, 'poi': False, 'director_fees': 119292,
'deferred_income': 0, 'long_term_incentive': 0, 'email_address': 0, 'from_poi
_to_this_person': 0}
```

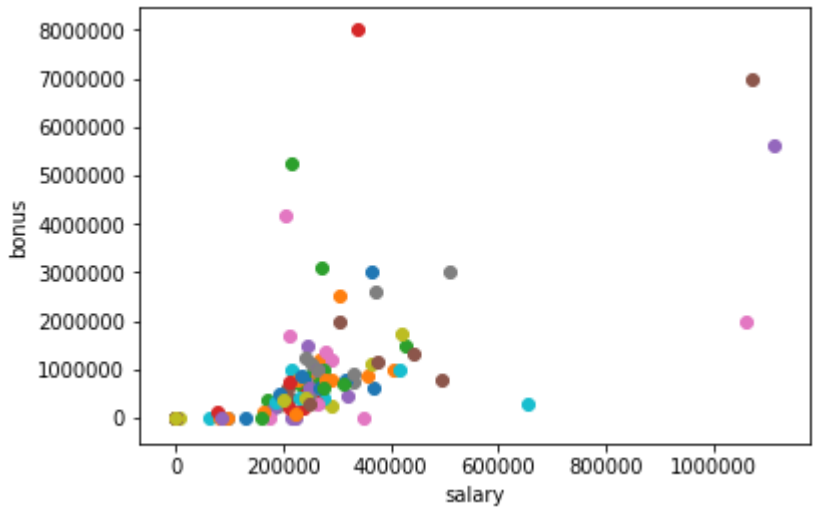
'LOCKHART EUGENE E' does not have any valid data points, so it can be removed. 'THE TRAVEL AGENCY IN THE PARK' doesn't seem like it is an actual person, so it can also be removed.

In [84]:
```
#remove the outliers
data_dict.pop('LOCKHART EUGENE E', 0)
data_dict.pop('THE TRAVEL AGENCY IN THE PARK', 0)
data_dict.pop('TOTAL', 0)
```

Out[84]:
```
{'bonus': 97343619,
 'deferral_payments': 32083396,
 'deferred_income': -27992891,
 'director_fees': 1398517,
 'email_address': 0,
 'exercised_stock_options': 311764000,
 'expenses': 5235198,
 'from_messages': 0,
 'from_poi_to_this_person': 0,
 'from_this_person_to_poi': 0,
 'loan_advances': 83925000,
 'long_term_incentive': 48521928,
 'other': 42667589,
 'poi': False,
 'restricted_stock': 130322299,
 'restricted_stock_deferred': -7576788,
 'salary': 26704229,
 'shared_receipt_with_poi': 0,
 'to_messages': 0,
 'total_payments': 309886585,
 'total_stock_value': 434509511}
```

In [85]:
```
data = featureFormat(data_dict, features_list)
for point in data:
    salary = point[1]
    bonus = point[6]
    matplotlib.pyplot.scatter( salary, bonus )

matplotlib.pyplot.xlabel("salary")
matplotlib.pyplot.ylabel("bonus")
matplotlib.pyplot.show()
```



The data looks way better now. There are a few more data points that look like they may be outliers, but I want to keep them as they may also be persons of interest.

New Features

I was interested in the ratio of emails sent to and sent from POIs as I imagined that POIs would have sent and received emails from other POIs more often than non-POIs. I also looked into pay ratios (salary, bonus, salary+bonus) of total compensation (total payments), stock ratios (salary+bonus) of total stock value, and exercised stock ratios of total stock value. If a person of interest commits fraud, then they should have received an abnormal amount of monetary benefit (very high salary, bonus, etc).

In [86]:
```
for key in data_dict:

    bonus = data_dict[key]["bonus"]
    salary = data_dict[key]["salary"]
    salary_bonus = data_dict[key]["salary"] + data_dict[key]["bonus"]
    total_payments = data_dict[key]["total_payments"]
    total_stock_value = data_dict[key]["total_stock_value"]
    to_messages = data_dict[key]["to_messages"]
    from_messages = data_dict[key]["from_messages"]
    from_poi = data_dict[key]["from_poi_to_this_person"]
    to_poi = data_dict[key]["from_this_person_to_poi"]
    cc = data_dict[key]['shared_receipt_with_poi']
    exercised_stock = data_dict[key]['exercised_stock_options']

    if bonus != 0 and total_payments != 0:
        data_dict[key]['bonus_ratio'] = round((bonus/float(total_payments)), 5
)
    else:
        data_dict[key]['bonus_ratio'] = 0
    if salary != 0 and total_payments != 0:
        data_dict[key]['salary_ratio'] = round((salary/float(total_payments)),
 5)
    else:
        data_dict[key]['salary_ratio'] = 0
    if bonus != 0 and salary !=0 and total_payments != 0:
        data_dict[key]['pay_ratio'] = round((data_dict[key]['bonus_ratio'] + d
ata_dict[key]['salary_ratio']/2), 5)
    else:
        data_dict[key]['pay_ratio'] = 0
    if bonus != 0 and salary !=0 and total_stock_value != 0:
        data_dict[key]['pay_stock_ratio'] = round((bonus+salary)/float(total_s
tock_value), 5)
    else:
        data_dict[key]['pay_stock_ratio'] = 0
    if exercised_stock != 0 and total_stock_value != 0:
        data_dict[key]['exercised_stock_ratio'] = round((exercised_stock/float
(total_stock_value)), 5)
    else:
        data_dict[key]['exercised_stock_ratio'] = 0
    if to_poi != 0 and to_messages != 0:
        data_dict[key]['email_to_poi_ratio'] = round(to_poi/float(to_messages
), 2)
    else:
        data_dict[key]['email_to_poi_ratio'] = 0
    if from_poi != 0 and from_messages != 0:
        data_dict[key]['email_from_poi_ratio'] = round(from_poi/float(from_mes
sages), 2)
    else:
        data_dict[key]['email_from_poi_ratio'] = 0
    if cc != 0 and from_messages != 0:
        data_dict[key]['cc_with_poi_ratio'] = round(cc/float(from_messages), 2
)
    else:
        data_dict[key]['cc_with_poi_ratio'] = 0
```

Feature Selection

I chose Decision Tree Classifier as my final algorithm, which is not affected by feature scaling. I used SelectKBest for feature selection, which is an automated feature selection tool. It assigns a score to each feature, and selects the ones with the highest scores.

In [87]:
```
features_list = ['poi', 'salary', 'to_messages', 'deferral_payments',
                 'total_payments', 'exercised_stock_options', 'bonus', 'restri
cted_stock',
                 'shared_receipt_with_poi', 'restricted_stock_deferred', 'tota
l_stock_value', 'expenses',
                 'loan_advances', 'from_messages', 'other', 'from_this_person_
to_poi',
                 'director_fees', 'deferred_income', 'long_term_incentive', 'f
rom_poi_to_this_person',
                 'bonus_ratio','salary_ratio', 'pay_ratio', 'pay_stock_ratio',

                 'exercised_stock_ratio', 'email_to_poi_ratio', 'email_from_po
i_ratio', 'cc_with_poi_ratio']

# total of 28 features. In my evaluation of the algorithms below, I will use S
electKBest with GridSearchCV to select the best
# features.

### Store to my_dataset for easy export below.
my_dataset = data_dict

### Extract features and labels from dataset for local testing
data = featureFormat(my_dataset, features_list, sort_keys = True)
labels, features = targetFeatureSplit(data)
```

In the parameter tuning section below, it shows an example of how I used SelectKBest along with GridSearchCV to automatically select the optimal number of features. I experimented with the k parameter in the pipeline and discovered that the optimal number of features is 7. The top 7 features selected from the model are: bonus, salary, bonus_ratio, pay_ratio, shared_receipt_with_poi, total_stock_value, and exercised_stock_options.

Evaluation and Validation

I tried 6 different algorithms, including Gaussian Naives Bayes classifier, Decision Tree classifier, Random Forest Classifier, AdaBoost Classifier, Support Vector Machine, and K Nearest Neighbors. I chose Decision Tree Classifier as it had the highest performance. All models had a high enough precision, but low recall.

In [88]:
```
from sklearn.metrics import recall_score, precision_score, f1_score
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn import cross_validation
from sklearn.cross_validation import train_test_split, StratifiedShuffleSplit,
 StratifiedKFold, KFold, cross_val_score
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV, StratifiedShuffleSplit
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline
```

I will be using the f1-score to evaluate the performance of each classifier. F1-score is the harmonic mean of precision and recall. A high precision means that when a POI gets flagged, I can safely assume that it's a real POI and not a false alarm (reliable). A high recall means that the chance of getting false negatives is low. Every time a POI shows up, it is identified as such (accurate).

$$Precision = \frac{True\ Positive}{True\ Positive\ +\ False\ Positive}$$

$$Recall = \frac{True\ Positive}{True\ Positive\ +\ False\ Negative}$$

I chose the f1-score because of the skewness in the data (18 POIs vs 128 non-POIs). If I just look at accuracy alone, it may look very high because a high number of non-POI was identified. Yet, the f1-score can tell me how successful the classifier is when it comes to identifying POIs, which is the actual goal. A high f1-score means that my false positives and false negatives are both low. This means that if the classifier identifies a POI, then I can trust that person is most likely a POI. The opposite is true, if the classifier determines a person as non-POI, then most likely that person is a non-POI.

I have to be mindful of the f-score's main weakness. F-score reveals the power of each feature independently from others. One score is computed for the first feature, and another score is computed for the second feature. Therefore, it does not indicate anything on the combination of both features.

In [89]:
```
print "Performance Comparison Table "
# Not restricting the maximum width in characters of a column
pd.options.display.max_colwidth = 0
data = {"Algorithms":["GaussianNB",
                      "DecisionTreeClassifier",
                      "RandomForestClassifier",
                      "AdaBoostClassifier",
                      "SupportVectorMachine",
                      "KNearestNeighbors"],
        "Precision":[0.38,1.00,1.00,1.00,0.00,0.75],
        "Recall":[0.61,1.00,0.83,1.00,0.00,0.17],
        "F1":[0.47,1.00,0.91,1.00,0.00,0.27]}
algorithms = pd.DataFrame(data, columns = ["Algorithms", "Precision", "Recall"
, "F1"])
algorithms
```

```
Performance Comparison Table
```

Out[89]:

	Algorithms	Precision	Recall	F1
0	GaussianNB	0.38	0.61	0.47
1	DecisionTreeClassifier	1.00	1.00	1.00
2	RandomForestClassifier	1.00	0.83	0.91
3	AdaBoostClassifier	1.00	1.00	1.00
4	SupportVectorMachine	0.00	0.00	0.00
5	KNearestNeighbors	0.75	0.17	0.27

The table shows the performance results after running the models trained with the entire dataset. Some of the models look like they have perfect or near perfect results, but I know that classifiers such as decision tree are prone to overfitting so the results may look too good to be true. I will discuss overfitting more in the next section. Conversely, it appears that support vector machine was not able to identify any POIs. This may be attributed to the fact that the dataset is not particularly large and the number of POIs is rather few.

Validation

Validation is evaluating the performance of the model on unseen data. We want to minimize the generalization error, which is the average error for data that we have never seen. A classic mistake that someone can make is evaluating the performance of the algorithm using the same dataset it was trained on. The result will look like the algorithm is performing better than it actually is, as evidenced in the table above. This is called overfitting. It's not as difficult to guess the correct label again once the model has already memorized all of the data and its labels. However, when new, unseen data is introduced, the model will not be able to generalize that data and will not know what to do with it.

It is data mining best practice to keep a subset of data to use as test data. You would train the model with training data and examine the performance on predicting the unknown values for the target variable with the test data. Then, you would compare the predicted values from the model with the true values. This way, you can use test data to evaluate the model's performance on generalizing new data.

Cross-validation is another of separating the data into training and test data. I used StratifiedShuffleSplit, which is a variation of k-fold cross-validation. It will shuffle the data, then split the data into training and test sets. This process is repeated more than once (you can choose how many times) and it will average the results across all the tests. There are 18 POIs compared with 128 non-POIs so there is sizable class imbalance. Stratification ensures that training and test splits have a balanced class distribution that represents the overall data.

In the next table, you can see that the model's performances after splitting the data into training and test data is not as perfect as we thought they were.

In [90]:
```
# hold out 30% of the data for testing and set the random_state parameter to 4
2 (random_state controls which points go into the
# training set and which are used for testing; setting it to 42 means we know
 exactly which events are in which set, and can
# check the results you get

features_train, features_test, labels_train, labels_test = train_test_split(fe
atures, labels, test_size=0.3, random_state=42)
```

In [91]:
```
print "Performance After One Training/Test Split"

pd.options.display.max_colwidth = 0
data = {"Algorithms":["GaussianNB",
                      "DecisionTreeClassifier",
                      "RandomForestClassifier",
                      "AdaBoostClassifier",
                      "SupportVectorMachine",
                      "KNearestNeighbors"],
        "Precision":[0.50,0.50,0.67,0.50,0.00,1.00],
        "Recall":[0.40,0.20,0.40,0.40,0.00,0.20],
        "F1":[0.44,0.29,0.50,0.44,0.00,0.33]}
algorithms = pd.DataFrame(data, columns = ["Algorithms", "Precision", "Recall"
, "F1"])
algorithms
```

```
Performance After One Training/Test Split
```

Out[91]:

	Algorithms	Precision	Recall	F1
0	GaussianNB	0.50	0.4	0.44
1	DecisionTreeClassifier	0.50	0.2	0.29
2	RandomForestClassifier	0.67	0.4	0.50
3	AdaBoostClassifier	0.50	0.4	0.44
4	SupportVectorMachine	0.00	0.0	0.00
5	KNearestNeighbors	1.00	0.2	0.33

Parameters Tuning

Tuning the parameters of an algorithm means customizing and finding the most robust set of parameters to optimize an algorithm's performance. Algorithms are assigned with default parameters, which may work in some cases, but it might not work the best for the dataset features in question. If you don't tune your parameters well, then you will not achieve the optimization and full potential of your algorithm, making its prediction outcome less accurate than it could be. You can also make the performance worse than it actually is. The opposite is true. An algorithm may have poor performance initially, but by trying different parameter values, you may find that the algorithm may work very well when the right set of parameters are used.

I tuned the parameters using GridSearchCV, which takes in sets of parameters and applies cross-validation to each possible combination of parameters to find the most optimal combination. For my decision tree algorithm, I tuned the **min_samples_split** and found out that 7 minimum sample splits provided the best performance. **Min_samples_split** provides a minimum threshold of observations in an internal node. If there is less than 7 people, then no further splitting can be done. I also tuned **max_depth**, which determines when the splitting of decision tree node stops. I want to avoid very deep trees as it performs well on known data (training data), but not as well on unknown data (test data) so it does not generalize well.

In [92]:
```
pca = PCA()
skb = SelectKBest()
clf_DT_pca = DecisionTreeClassifier()
target_names = ["Not POI", "POI"]

pipe = Pipeline([("skb", skb), ("pca",pca), ("DT",clf_DT_pca)])
parameters = {"skb__k":[5,7,10,15,'all'], "pca__n_components": range(2, 6), \
              "pca__svd_solver":['randomized'], "DT__min_samples_split":[5,7,1
0,20], "DT__max_depth":[1,2,5,10]}
split = StratifiedShuffleSplit(n_splits= 3, test_size=0.3, random_state=42)
grid = GridSearchCV(pipe, parameters, cv=split, scoring ='f1')
clf_DT_pca = grid.fit(features_train, labels_train)
print "best estimator: ", clf_DT_pca.best_estimator_
print "best parameters: ", clf_DT_pca.best_params_
prediction_DT_pca = clf_DT_pca.best_estimator_.predict(features_test)

skb_steps = clf_DT_pca.best_estimator_.named_steps['skb']
tree = clf_DT_pca.best_estimator_.named_steps['DT']
feature_scores = ['%.2f' % elem for elem in skb_steps.scores_ ]
feature_scores_pvalues = ['%.3f' % elem for elem in skb_steps.pvalues_ ]
features_selected=[(features_list[i+1], feature_scores[i], feature_scores_pval
ues[i]) \
                   for i in skb_steps.get_support(indices=True)]
features_selected = sorted(features_selected, key=lambda feature: float(featur
e[1]) , reverse=True)
print 'Selected Features, Scores, P-Values'
print features_selected
accuracy_DT_pca = accuracy_score(prediction_DT_pca, labels_test)
precision_DT_pca = precision_score(labels_test, prediction_DT_pca)
recall_DT_pca = recall_score(labels_test, prediction_DT_pca)
f1_DT_pca = f1_score(labels_test, prediction_DT_pca)

print "f1-score pca:", f1_DT_pca
print classification_report(y_true=labels_test, y_pred=prediction_DT_pca, targ
et_names=target_names)

feature_importances = tree.feature_importances_
print "Feature Importances: ",feature_importances


best_features_list = ['poi', 'bonus', 'salary', 'bonus_ratio', 'pay_ratio',
                      'shared_receipt_with_poi', 'total_stock_value', 'exercised_st
ock_options']

# total of 7 features

my_dataset = data_dict

best_data = featureFormat(my_dataset, best_features_list, sort_keys = True)
labels, best_features = targetFeatureSplit(best_data)
best_features_train, best_features_test, labels_train, labels_test = train_tes
t_split(best_features, labels, test_size=0.3, random_state=42)
clf_DT_split = DecisionTreeClassifier(min_samples_split=7, max_depth=10, rando
m_state=42)
clf_DT_split.fit(best_features_train, labels_train)
print clf_DT_split.feature_importances_
```

```
best estimator:  Pipeline(memory=None,
     steps=[('skb', SelectKBest(k=7, score_func=<function f_classif at 0x0000
00000BDD8DD8>)), ('pca', PCA(copy=True, iterated_power='auto', n_components=
2, random_state=None,
  svd_solver='randomized', tol=0.0, whiten=False)), ('DT', DecisionTreeClassi
fier(class_weight=None, criterion='gini', max_depth=1...      min_weight_frac
tion_leaf=0.0, presort=False, random_state=None,
            splitter='best'))])
best parameters:  {'DT__min_samples_split': 7, 'DT__max_depth': 10, 'pca__svd
_solver': 'randomized', 'pca__n_components': 2, 'skb__k': 7}
Selected Features, Scores, P-Values
[('bonus', '30.73', '0.000'), ('salary', '15.86', '0.000'), ('bonus_ratio',
'13.07', '0.000'), ('pay_ratio', '11.66', '0.001'), ('shared_receipt_with_po
i', '10.72', '0.001'), ('total_stock_value', '10.63', '0.002'), ('exercised_s
tock_options', '9.68', '0.002')]
f1-score pca: 0.333333333333
             precision    recall  f1-score   support

    Not POI       0.92      0.87      0.89        38
        POI       0.29      0.40      0.33         5

avg / total       0.84      0.81      0.83        43

Feature Importances:  [ 0.52600861  0.47399139]
[ 0.23764125  0.14041514  0.          0.          0.32442044  0.15309617
  0.144427  ]
```

The precision falls under the desired level of 0.30. However, after evaluating the performances for each model using tester.py, the Decision Tree model performed the best. Bonus and salary are the top two features with the highest scores. The bonus ratio and pay ratio features that I created are derived from bonus and salary, so naturally they also have high scores. I'm surprised that none of the email features were selected.

Feature Selection Revisited

In the pipeline, SelectKBest selects the top k features with the highest scores and applies it to the data. By combining it with GridSearchCV, different combinations of hyperparameters were tuned to find the best one, including the number of k features. I provided a list of possible choices for number of features (5,7,15, and all) for GridSearchCV to choose from. After trying each one, it found that the top 7 features provided the best performance.

In [103]:
```
print "Final Performance Using tester.py After Cross-Validation and Parameters
 Tuning"
pd.options.display.max_colwidth = 0
data = {"Algorithms":["GaussianNB",
                      "DecisionTreeClassifier",
                      "RandomForestClassifier",
                      "AdaBoostClassifier",
                      "SupportVectorMachine",
                      "KNearestNeighbors"],
        "Parameters":["Default",
                      "criterion='entropy', min_samples_split=7, n_estimators=
7, max_depth=10",
                      "n_estimators=7",
                      "algorithm='SAMME.R', n_estimators=50",
                      "kernel='sigmoid', C=0.5",
                      "weights='uniform', n_neighbors=5, p=2"],
        "Precision":[0.55162,0.38208,0.32333,0.31883,1.00000,0.38741],
        "Recall":[0.28050,0.31350,0.14550,0.16850,0.04250,0.16000],
        "F1":[0.37189,0.34441,0.20069,0.22048,0.08153,0.22647],
        "F2":[0.31108,0.32517,0.16348,0.18604,0.05257,0.18128]}
algorithms = pd.DataFrame(data, columns = ["Algorithms", "Parameters", "Precis
ion", "Recall", "F1", "F2"])
algorithms
```

```
Final Performance Using tester.py After Cross-Validation and Parameters Tunin
g
```

Out[103]:

	Algorithms	Parameters	Precision	Recall	F1	F2
0	GaussianNB	Default	0.55162	0.2805	0.37189	0.31108
1	DecisionTreeClassifier	criterion='entropy', min_samples_split=7, n_estimators=7, max_depth=10	0.38208	0.3135	0.34441	0.32517
2	RandomForestClassifier	n_estimators=7	0.32333	0.1455	0.20069	0.16348
3	AdaBoostClassifier	algorithm='SAMME.R', n_estimators=50	0.31883	0.1685	0.22048	0.18604
4	SupportVectorMachine	kernel='sigmoid', C=0.5	1.00000	0.0425	0.08153	0.05257
5	KNearestNeighbors	weights='uniform', n_neighbors=5, p=2	0.38741	0.1600	0.22647	0.18128

Resource

- http://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html)
- http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html)
- http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html (http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html)
- http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html (http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html)
- http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html)
- http://scikit-learn.org/0.17/modules/generated/sklearn.grid_search.GridSearchCV.html (http://scikit-learn.org/0.17/modules/generated/sklearn.grid_search.GridSearchCV.html)
- https://stackoverflow.com/questions/21095654/what-is-a-nonetype-object (https://stackoverflow.com/questions/21095654/what-is-a-nonetype-object)
- https://stackoverflow.com/questions/33091376/python-what-is-exactly-sklearn-pipeline-pipeline (https://stackoverflow.com/questions/33091376/python-what-is-exactly-sklearn-pipeline-pipeline)
- http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html (http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html)
- https://medium.com/@aneesha/svm-parameter-tuning-in-scikit-learn-using-gridsearchcv-2413c02125a0 (https://medium.com/@aneesha/svm-parameter-tuning-in-scikit-learn-using-gridsearchcv-2413c02125a0)
- https://datascience.stackexchange.com/questions/31232/why-not-use-scaler-fit-transform-on-total-dataframe (https://datascience.stackexchange.com/questions/31232/why-not-use-scaler-fit-transform-on-total-dataframe)
- https://stats.stackexchange.com/questions/151483/why-am-i-getting-100-accuracy-for-svm-and-decision-tree-scikit (https://stats.stackexchange.com/questions/151483/why-am-i-getting-100-accuracy-for-svm-and-decision-tree-scikit)
- https://stats.stackexchange.com/questions/35276/svm-overfitting-curse-of-dimensionality (https://stats.stackexchange.com/questions/35276/svm-overfitting-curse-of-dimensionality)
- https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#lists (https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#lists)
- https://discussions.udacity.com/t/print-feature-names-for-selectkbest-where-k-value-is-inside-param-grid-of-gridsearchcv/302844/2 (https://discussions.udacity.com/t/print-feature-names-for-selectkbest-where-k-value-is-inside-param-grid-of-gridsearchcv/302844/2)
- https://discussions.udacity.com/t/how-can-i-get-tester-py-to-print-multiple-accuracy-and-recall-values-for-documentation/301255/2 (https://discussions.udacity.com/t/how-can-i-get-tester-py-to-print-multiple-accuracy-and-recall-values-for-documentation/301255/2)
- https://discuss.analyticsvidhya.com/t/why-it-is-necessary-to-normalize-in-knn/2715 (https://discuss.analyticsvidhya.com/t/why-it-is-necessary-to-normalize-in-knn/2715)
- https://discussions.udacity.com/t/getting-feature-importances-/212958/6 (https://discussions.udacity.com/t/getting-feature-importances-/212958/6)