Enron Person of Interest Identifier

In 2001 Enron was one of the biggest companies in the US. However, in 2002 it declared bankrupcy due to fraud committed by a few people in the management. The federal investigators hunted down these people and they have been scrutinized. These are the people we will refer to as 'persons of Interest' or 'POI' in this document. A large amount, otherwise confidential, company data have been made public at this time.

The purpose of this machine learning project is to identify such POIs based on the publicly available information. We are trying to build a model that will be trained with the help of the available data and subsequently used to predict if any preson is a POI or not. The steps we will undertake in this project are:

1. Prepare this dataset such that the machine learning tools can handle them (including outlier data removal).
2. Experiment with various classifiers on this dataset and evaluate their performance.
3. Select the best performing classifier and understand its characteristics.

In [85]:
```
import sys
import pickle
import numpy as np
from matplotlib import pyplot as plt

# Data spliting
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit

# Feature scaling
from sklearn.preprocessing import MinMaxScaler
# Feature selection
from sklearn.feature_selection import SelectKBest, f_classif
# Principal Component Analysis
from sklearn.decomposition import PCA
# Classifiers
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

# Pipeline
from sklearn.pipeline import Pipeline

# grid search
from sklearn.model_selection import GridSearchCV

# Performance metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

# Helper functions
from feature_format import featureFormat, targetFeatureSplit
from tester import dump_classifier_and_data
from tester import test_classifier
```

Background on the dataset

In [86]:
```
# --------------------
# Load the dataset
# --------------------
with open("final_project_dataset.pkl", "r") as data_file:
    data_dict = pickle.load(data_file)
print "Number of records in the dataset:", len(data_dict)
print
# --------------------
# What are the features available in the dataset?
# --------------------
one_dict = data_dict[list(data_dict)[0]]
print "No. of features:", len(one_dict)
list_of_all_features = list(one_dict)
print "The features are:"
print list_of_all_features
print
# ----------------------
# How many POIs are there in the dataset?
# ----------------------
total_no_of_pois = 0
for key in data_dict:
    if data_dict[key]['poi'] == 1:
        total_no_of_pois +=1
print "Total number of POIs:", total_no_of_pois
# ----------------------
# How many non-POIs are there in the dataset?
# ----------------------
total_no_of_non_pois = 0
for key in data_dict:
    if data_dict[key]['poi'] == 0:
        total_no_of_non_pois +=1
print "Total number of non-POIs:", total_no_of_non_pois
print
# ------------------------------------
# Which features do not have any values
# ------------------------------------
print "Features with count of null values:"
print
print "{0:18s} \t{1:21s}".format("Feature", "Null value count")
print "{0:18s} \t{1:21s}".format("-------", "----------------")
for a_feature in list_of_all_features:
    nan_value = 0
    for key in data_dict:
        feature_val = data_dict[key][a_feature]
        if feature_val == "NaN":
           nan_value += 1
    print "{0:25s} \t{1:3d}".format(a_feature, nan_value)
```

```
Number of records in the dataset: 146

No. of features: 21
The features are:
['salary', 'to_messages', 'deferral_payments', 'total_payments', 'exercised_s
tock_options', 'bonus', 'restricted_stock', 'shared_receipt_with_poi', 'restr
icted_stock_deferred', 'total_stock_value', 'expenses', 'loan_advances', 'fro
m_messages', 'other', 'from_this_person_to_poi', 'poi', 'director_fees', 'def
erred_income', 'long_term_incentive', 'email_address', 'from_poi_to_this_pers
on']

Total number of POIs: 18
Total number of non-POIs: 128

Features with count of null values:

Feature                 Null value count
-------                 ----------------
salary                           51
to_messages                      60
deferral_payments               107
total_payments                   21
exercised_stock_options          44
bonus                            64
restricted_stock                 36
shared_receipt_with_poi          60
restricted_stock_deferred       128
total_stock_value                20
expenses                         51
loan_advances                   142
from_messages                    60
other                            53
from_this_person_to_poi          60
poi                               0
director_fees                   129
deferred_income                  97
long_term_incentive              80
email_address                    35
from_poi_to_this_person          60
```

There are 146 samples in this dataset corresponding to individuals or entities. The 21 features are divided into three different types:

1. Financial features (14)
2. E-mail features, and (6)
3. Status of a person (whether POI or not) (1)

Many of the numbers for the feature set are missing (except for POI designation). This limits the opportunity for the model to learn, but for our purpose we will fill these non-null values with '0'.

There are 18 POIs out of 146 records. Most of the individuals (88%) are non-POI. This poses a unique challenge in building a classifier model. The training dataset and the test dataset for supervised learning purpose should have proportional POI/non-POI representation in the respective set. Moreover, if the model tend to have a bias towards classifying a data as a non-POI then it will be correct most of the time. This will give a false impression of achieving high accuracy by the model.

In order to give a fair opportunity to the model to learn on various mix of labels, the training and test sets will be created with many 'folds'. We are dealing with a small dataset and it will be appropriate to use the StratifiedShuffleSplit cross validator to create a large number of training/test sets. This will be discussed more in a later section in the context of preprocessing the dataset in a pipelined mechanism.

Feature Selection

I decided to pick up the following features as a mixture of financial and e-mail related data. The first element in this array is 'poi'. This is a 'label' rather than a feature. This is chosen in such a fashion such that it can be used by a few helper functions.

Moreover, we notice that the feature *'loan_advances'* is mostly populated with NaN (142 out of 146 observations). I decide to drop this feature from the feature list.

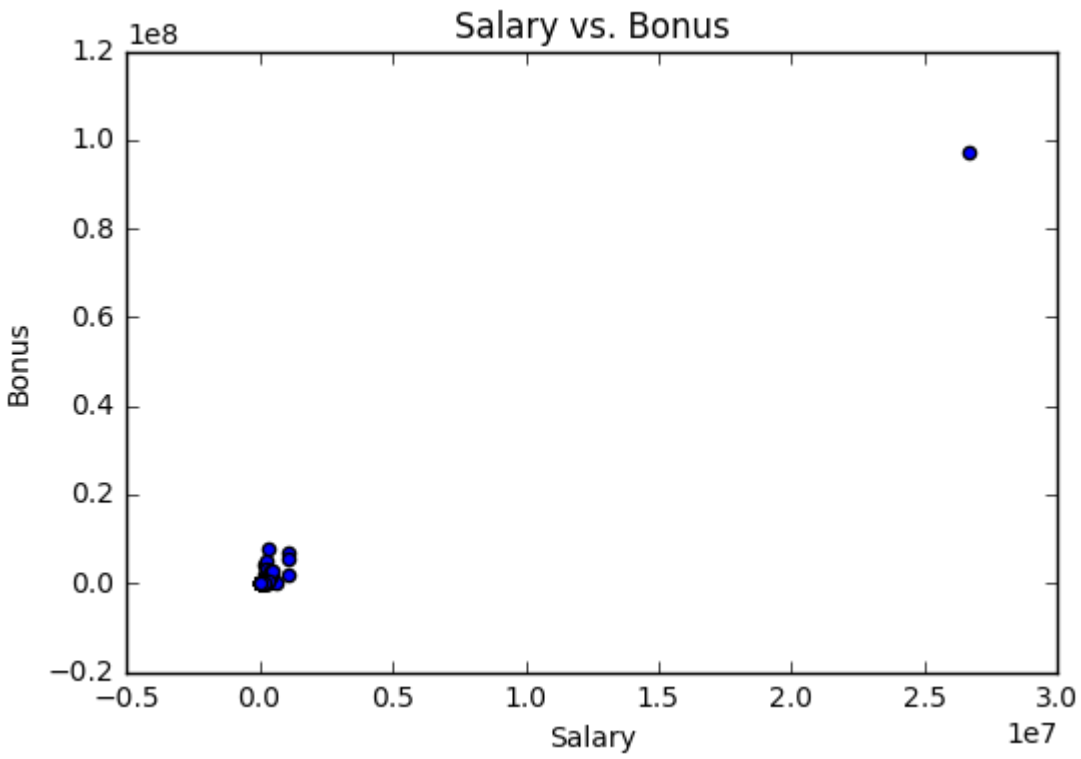
In [87]:
```
features_list = ['poi','salary', 'bonus',
                 'deferral_payments', 'total_payments',
                 'exercised_stock_options', 'restricted_stock',
                 'restricted_stock_deferred', 'total_stock_value',
                 'expenses', 'director_fees', 'deferred_income',
                 'long_term_incentive', 'to_messages', 'from_messages',
                 'from_poi_to_this_person', 'from_this_person_to_poi' ]
```

Outlier Detection/Removal

In [88]:
```
print "Checking for any outliers ...."
print
# Create numpy arrays out of the selected features in the dataset
data = featureFormat(data_dict, features_list, sort_keys = True)

for point in data:
    plt.scatter(point[1], point[2])
plt.xlabel("Salary")
plt.ylabel("Bonus")
plt.title("Salary vs. Bonus")
plt.show()
```

```
Checking for any outliers ....
```



We see a clear outlier. Lets investigate this data item.

In [89]:
```
sorted_data = sorted(data, key=lambda x:x[1], reverse=True)
outlier_salary = int(sorted_data[0][1])
outlier_bonus = sorted_data[0][2]

for key in data_dict:
    if data_dict[key]["salary"] == outlier_salary:
        print "Outlier key = {}, Outlier salary = {}".format(key, data_dict[ke
y]["salary"])
        print data_dict[key]
```

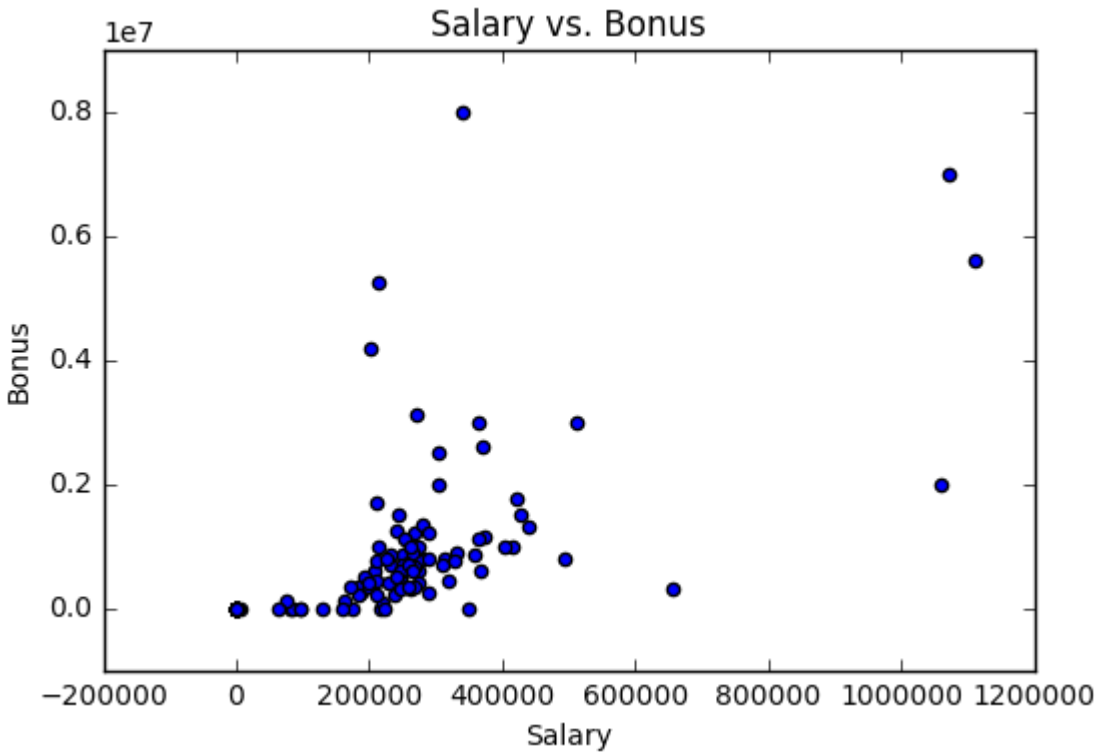
```
Outlier key = TOTAL, Outlier salary = 26704229
{'salary': 26704229, 'to_messages': 'NaN', 'deferral_payments': 32083396, 'to
tal_payments': 309886585, 'exercised_stock_options': 311764000, 'bonus': 9734
3619, 'restricted_stock': 130322299, 'shared_receipt_with_poi': 'NaN', 'restr
icted_stock_deferred': -7576788, 'total_stock_value': 434509511, 'expenses':
 5235198, 'loan_advances': 83925000, 'from_messages': 'NaN', 'other': 4266758
9, 'from_this_person_to_poi': 'NaN', 'poi': False, 'director_fees': 1398517,
 'deferred_income': -27992891, 'long_term_incentive': 48521928, 'email_addres
s': 'NaN', 'from_poi_to_this_person': 'NaN'}
```

This record corresponds to the sum total of all the salaries/bonuses of all the individuals in the dataset. This is not a person and we will remove it.

In [90]:
```
print "Removing the record of 'TOTAL' from the dataset ..."
del data_dict['TOTAL']
print "New samples in the dataset:", len(data_dict)

# Extract the feature set again after removing the outlier
data = featureFormat(data_dict, features_list, sort_keys = True)
# How does the plot now look like?
for point in data:
    plt.scatter(point[1], point[2])
plt.xlabel("Salary")
plt.ylabel("Bonus")
plt.title("Salary vs. Bonus")
plt.show()
```

```
Removing the record of 'TOTAL' from the dataset ...
New samples in the dataset: 145
```



We notice that there are a few data points that stand out as outside of the norm. I ask the question - are they among any more outliers that should be removed? Are they real persons or some entities that are of no interest to the investigators? We will check who they are.

In [91]:
```
# Are there any more outliers?
# Lets consider some threshold of numbers for us to be interested
high_salary = 1000000
high_bonus = 7000000
print "Who are the people who got > 1M salary or > 7M bonus?:"
for key in data_dict:
    key_salary = data_dict[key]['salary']
    key_bonus  = data_dict[key]['bonus']
    if key_salary != 'NaN' and key_bonus != 'NaN':
       if key_salary > high_salary or key_bonus > high_bonus:
          print key, key_salary, key_bonus
```

```
Who are the people who got > 1M salary or > 7M bonus?:
LAVORATO JOHN J 339288 8000000
LAY KENNETH L 1072321 7000000
SKILLING JEFFREY K 1111258 5600000
FREVERT MARK A 1060932 2000000
```

These are the individuals and we will retain them in the dataset.

In [92]:
```
# ------------------------------
# Any non-person in the dataset? Such entities will typically have
# no salary, bonus, restricted stock and possibly no e-mail address
# ------------------------------
for key in data_dict:
    key_salary = data_dict[key]['salary']
    key_bonus = data_dict[key]['bonus']
    key_restricted_stock = data_dict[key]['restricted_stock']
    key_email_address = data_dict[key]['email_address']
    if key_salary == 'NaN' and \
       key_bonus == 'NaN' and \
       key_restricted_stock == 'NaN' and \
       key_email_address == 'NaN':
       print key
```

```
WALTERS GARETH W
BELFER ROBERT
URQUHART JOHN A
WHALEY DAVID A
MENDELSOHN JOHN
WAKEHAM JOHN
DUNCAN JOHN H
LEMAISTRE CHARLES
WROBEL BRUCE
MEYER JEROME J
LOCKHART EUGENE E
PEREIRA PAULO V. FERRAZ
BLAKE JR. NORMAN P
THE TRAVEL AGENCY IN THE PARK
WINOKUR JR. HERBERT S
BADUM JAMES P
YEAP SOON
FUGH JOHN L
SAVAGE FRANK
GRAMM WENDY L
```

There is a travel agency in the dataset. I wanted to confirm if this entity carries any meaningful information.

In [93]:
```
print data_dict['THE TRAVEL AGENCY IN THE PARK']
```

```
{'salary': 'NaN', 'to_messages': 'NaN', 'deferral_payments': 'NaN', 'total_pa
yments': 362096, 'exercised_stock_options': 'NaN', 'bonus': 'NaN', 'restricte
d_stock': 'NaN', 'shared_receipt_with_poi': 'NaN', 'restricted_stock_deferre
d': 'NaN', 'total_stock_value': 'NaN', 'expenses': 'NaN', 'loan_advances': 'N
aN', 'from_messages': 'NaN', 'other': 362096, 'from_this_person_to_poi': 'Na
N', 'poi': False, 'director_fees': 'NaN', 'deferred_income': 'NaN', 'long_ter
m_incentive': 'NaN', 'email_address': 'NaN', 'from_poi_to_this_person': 'Na
N'}
```

Looks like this entity does carry some finanacial information. I will let it remain in the dataset.

New Feature Creation

One may intuitively think that the POIs will be exchaning a lot of e-mails among themselves. However, it is possible that an individual may broadcast a lot of e-mails to the entire company that includes the POIs as recipient. But that does not indicate that the sender is a POI. Similarly, the POI may be broadcasting e-mails to a lot of people including non-POIs. Rather, a better metric to consider is the fractions of e-mails sent to or received from POIs.

We create the following two new features:

1. Fraction of all e-mails that went from this person to poi
2. Fraction of all e-mails that came from poi to this person If any of the relevant e-mail messages is 'NaN' then the fraction is considered as '0'

In [94]:
```
my_dataset = data_dict
for key in my_dataset:
    # ----
    # Calculate fraction of messages from this person to poi
    # ----
    key_fr_msgs = my_dataset[key]["from_messages"]
    key_fr_msgs_to_poi = my_dataset[key]["from_this_person_to_poi"]
    if key_fr_msgs_to_poi != 'NaN' and key_fr_msgs != 'NaN':
       fraction_msgs_to_poi = key_fr_msgs_to_poi/float(key_fr_msgs)
    else:
       fraction_msgs_to_poi = 0
    my_dataset[key]["fraction_from_this_person_to_poi"] = fraction_msgs_to_poi
    # -----
    # Calculate fraction of messages from poi to this person
    # -----
    key_to_msgs = my_dataset[key]["to_messages"]
    key_to_msgs_fr_poi = my_dataset[key]["from_poi_to_this_person"]
    if key_to_msgs_fr_poi != 'NaN' and key_to_msgs != 'NaN':
       fraction_msgs_fr_poi = key_to_msgs_fr_poi/float(key_to_msgs)
    else:
       fraction_msgs_fr_poi = 0
    my_dataset[key]["fraction_to_this_person_from_poi"] = fraction_msgs_fr_poi
```

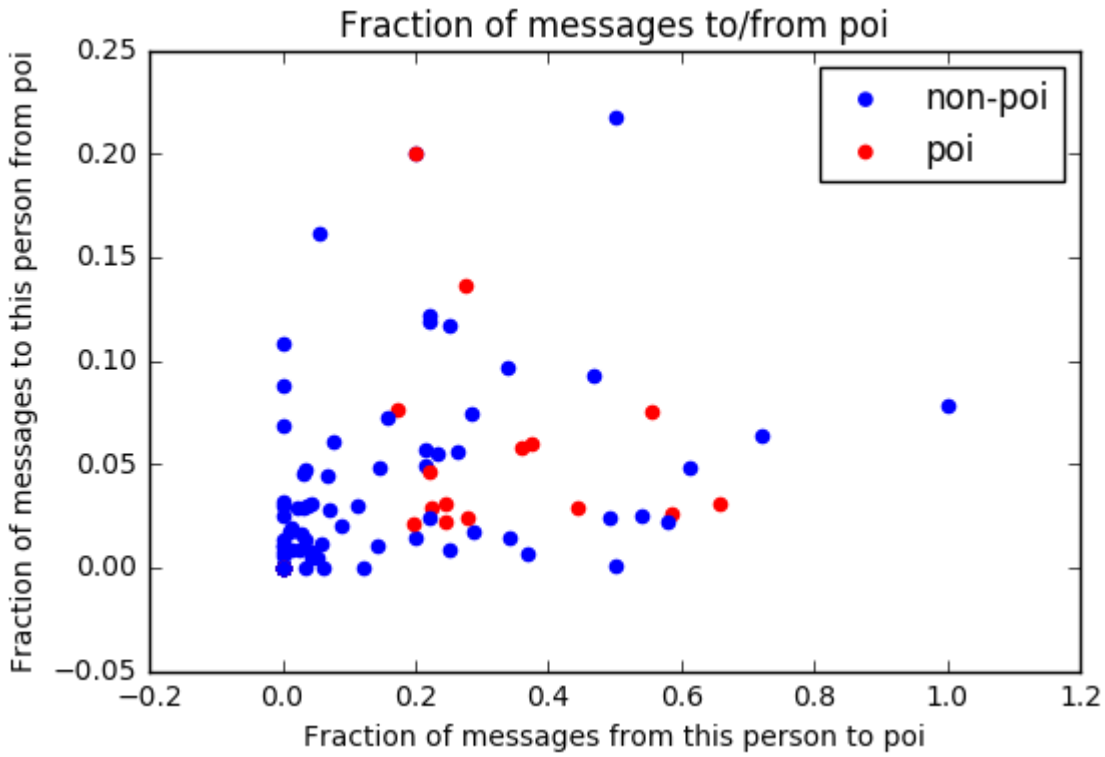
Add the two new features in the features list.

In [95]:
```
features_list.append('fraction_from_this_person_to_poi' )
features_list.append('fraction_to_this_person_from_poi' )
print features_list
# Extract the feature set again now that the outlier has been removed
# and two new features have been added.
data = featureFormat(my_dataset, features_list, sort_keys = True)
print data[0]
```

```
['poi', 'salary', 'bonus', 'deferral_payments', 'total_payments', 'exercised_
stock_options', 'restricted_stock', 'restricted_stock_deferred', 'total_stock
_value', 'expenses', 'director_fees', 'deferred_income', 'long_term_incentiv
e', 'to_messages', 'from_messages', 'from_poi_to_this_person', 'from_this_per
son_to_poi', 'fraction_from_this_person_to_poi', 'fraction_to_this_person_fro
m_poi']
[  0.00000000e+00   2.01955000e+05   4.17500000e+06   2.86971700e+06
   4.48444200e+06   1.72954100e+06   1.26027000e+05  -1.26027000e+05
   1.72954100e+06   1.38680000e+04   0.00000000e+00  -3.08105500e+06
   3.04805000e+05   2.90200000e+03   2.19500000e+03   4.70000000e+01
   6.50000000e+01   2.96127563e-02   1.61957271e-02]
```

Do the fraction of e-mail messages to/from poi indicate anything obviously important? We will plot them to visualize.

In [96]:
```
for point in data:
    if point[0] == 1:
       point_color = 'red'
    else:
       point_color = 'blue'
    plt.scatter(point[17], point[18], color=point_color)
type1 = plt.scatter(0.2, 0.2, marker = 'o', color='blue')
type2 = plt.scatter(0.2, 0.2, marker = 'o', color='red')
plt.xlabel("Fraction of messages from this person to poi")
plt.ylabel("Fraction of messages to this person from poi")
plt.legend((type1, type2), ('non-poi', 'poi'), loc='upper right', scatterpoint
s=1)
plt.title("Fraction of messages to/from poi")
plt.show()
```



The plot does not show a very strong indication about identifying the POIs based on the fraction of e-mails. However, this seems to indicate a cluster in which the POIs are confined.

I have added these two new features in the features list to be considered for the model creation. In a subsequent section of this report, I will describe how a number of best features will be selected based on a scoring approach. There, we will notice that one of these two features actually scores high in the ranking and thus considered for the development of the model.

Classifiers

I experimented with the following classifiers:

1. Gaussian Naive Bayes
2. Decision Tree
3. Random Forest
4. Support Vector Machine

Based on the performance achieved by the four classifiers, the best performance was demonstrated by the Gaussian Naive Bayes approach. The best performance was determined by the F1 score which is a weighted combination of the precision and recall metrices.

The details of the performance scores will be mentioned in the concluding part of this report (please refer to the **Conclusion** section). However, in the following section I am going to describe my pipeline approach for data processing and classification. I will focus on the steps leading to the use of the Naive Bayes classifier.

Pipeline steps for estimation

I have used a pipeline mechanism to perform the following steps in sequence:

1. Feature scaling

The features in the Enron dataset are very different in their ranges. E.g. the financial features like salary, bonus, stock options etc. range in millions of dollers, the number of e-mail messages range in the 100s, while the fraction of e-mail messages range in 0 - 1. Any tansformations we apply on these features have unequal influence on them depending on the range. Most machine learning algorithms do not perform well in delaing with such disparate data. Hence, it is common to 'scale' such data to fit to a common range, e.g. 0 - 1. In this exercise I have used the MinMaxScalar preprocess to map all the numerical feature data to a 0 - 1 range.

2. Select a set of best features

A set of 'best' features are selected with the SelectKBest estimator. Based on statistical scoring information the estimator picks up 'k' (user-supplied parameter) top-scoring features to feed the next step of the classification process. I will show later in my analysis the best features ranked with their scores.

3. PCA for feature reduction and transform the dataset to reduced dimention

Principal component analysis is a technique to determine the components of every data point along a few number of orthogonal dimensions. An inherent scoring mechanism is used to rank a dimension in terms of the amount of variances of the data along that dimension. E.g. the highest ranking dimension is expected to show the maximum amount of variance along that dimension.

4. Classification

Classification is the last step in the pipeline where I separately tried the four classifiers that operated on the data preprocessed by step 1 - 3.

We understand that the entire pipeline can be tuned or configured by selecting various parameters at all the stages. In this approach I have taken an automated approach by using GridSearchCV to exhaustively try all the combinations of the parameters to arrive at the most optimal performance of the classifier.

Note - the dataset is small. Hence, in order to get the maximum advantage in terms of being able to work with a large number of train/test split, I have used StratifiedShuffleSplit cross validator. Interestingly this utility ensures that the proportion of the labes remain the same in both the train and test splits.

In [97]:
```
### Extract features and labels from dataset for local testing
print "Extract features and labels from dataset for local testing"
data = featureFormat(my_dataset, features_list, sort_keys = True)

labels, features = targetFeatureSplit(data)
```

```
Extract features and labels from dataset for local testing
```

Parameter Tuning

As mentioned, I undertake a pipeline-based approach to create the model. This pipeline involves a few estimators/transformers that are allowed to learn through the available dataset. These entities are 'parameterized' in the sense that their underlying algorithms react differently depending on the parameters supplied to them (based on the nature of the dataset, of course). This ultimately leads to different kind of performances demonstrated by the model.

Moreover, this pipeline demonstrates an overall performance based on a cumulative and collective performance of each estimator. Hence it is very important that we supply a 'range' of relevant parameters to each of the estimator in the pipeline to determine what combination of the parameters give the best performance of the model. In this exercise, we will let the task of selecting the best set of parameters (we will refer to this task as 'parameter tuning') to GridSearchCV.

The advantage of a flexible parameter tuning (with a choice of a large number of input parameters to choose from), is - the model has the opportunity to be trained on a wide variety of input situation. This enables the model to react resonably well (classify) to a new set of input data. On the other hand, not having the pipeline to do parameter tuning, we run the risk of making the model to be either 'too good' ('overfit') for the training data or 'too bad' ('underfit') for a new data set to classify.

Choice of Parameters

Following are the parameters I have considered for the pipeline components:

Feature scaling:

> None

Feature selection (SelectKBest):

> Choice of number of features to select:
>
> > k: [10, 12, 16, 18]

Dimensionality reduction (PCA):

> Choice of final number of reduced dimensions:
>
> > n_components: [6, 8, 10]
>
> Whiten or not (improve accuracy at the cost of some information loss):
>
> > whiten: [True]

NOTE: Feature selection step feeds the PCA step. Hence it is important to ensure that the smallest value of '*k*' parameter of SelectKBest is greater than or equal to the highest value of the '*n_components*' parameter of PCA.

Classifiers:

Naive Bayse:

None

Decision Tree:

Threshold (minimum) number of samples to decide to split an internal node:

min_samples_split: [10, 15]

Random Forest:

Number of trees in the forest:

n_estimators":[8, 10, 12]

Number of features to consider when looking for the best split:

max_features:[2, 4]

Support Vector Machine:

Kernel:

kernel:['rbf']

Misclassification penalty parameter:

C:[1, 10, 100]

Given a user's choice of set of parameters for the components of the pipeline to consider, GridSearchCV exhaustively tries to find out which combination gives the best performance. The following example (with SVM classifier) illustrates the mechanism:

param_grid = {"SKB__k":[10, 12, 16, 18], "PCA__n_components":[6, 8, 10], "PCA__whiten":[False, True], "svm__kernel":['linear', 'rbf'], "svm__C":[1, 10, 100]}

NOTE: Per convention a parameter in the parameter grid can be represented in the following syntax:

> *pipeline component__parameter name*

E.g.

"SKB__k" denotes the 'k' parameter of the SelectKBest component.

In this case, GridSearchCV exhaustively tries to explore 4 * 3 * 2 * 2 * 3 = 144 combinations of parameters to arrive at the best combination to achieve the best result.

Naive Bayes Classifier

In [98]:
```
scaler = MinMaxScaler()
skb = SelectKBest()
pca = PCA()
gnb = GaussianNB()

# ----------------------------
# Pipeline with Naive Bayes classifier
# ----------------------------
Pipe = Pipeline(steps=[('scaling',scaler),("SKB", skb),("PCA", pca),("GNB", gn
b)])

# ------------------------
# Use the pipeline. Naive Bayes classifier does not have any parameter
# ------------------------
param_grid = {"SKB__k":[10, 12, 16, 18], "PCA__n_components":[6, 8, 10], "PCA_
_whiten":[True]}
# ------------------------------------------------
# Split the dataset into 100 folds with 30% test set
# ------------------------------------------------
stashsp = StratifiedShuffleSplit(n_splits=100, test_size=0.3, random_state=42)

gs = GridSearchCV(Pipe, param_grid=param_grid, scoring='f1', n_jobs=1, cv=stas
hsp)
gs.fit(features, labels)
print "Best parameters from GridSearchCV are:", gs.best_params_
print "Best estimator found by grid search:", gs.best_estimator_
clf = gs.best_estimator_
```

```
Best parameters from GridSearchCV are: {'PCA__n_components': 6, 'PCA__white
n': True, 'SKB__k': 10}
Best estimator found by grid search: Pipeline(steps=[('scaling', MinMaxScaler
(copy=True, feature_range=(0, 1))), ('SKB', SelectKBest(k=10, score_func=<fun
ction f_classif at 0x0000000009ABD748>)), ('PCA', PCA(copy=True, iterated_pow
er='auto', n_components=6, random_state=None,
  svd_solver='auto', tol=0.0, whiten=True)), ('GNB', GaussianNB(priors=Non
e))])
```

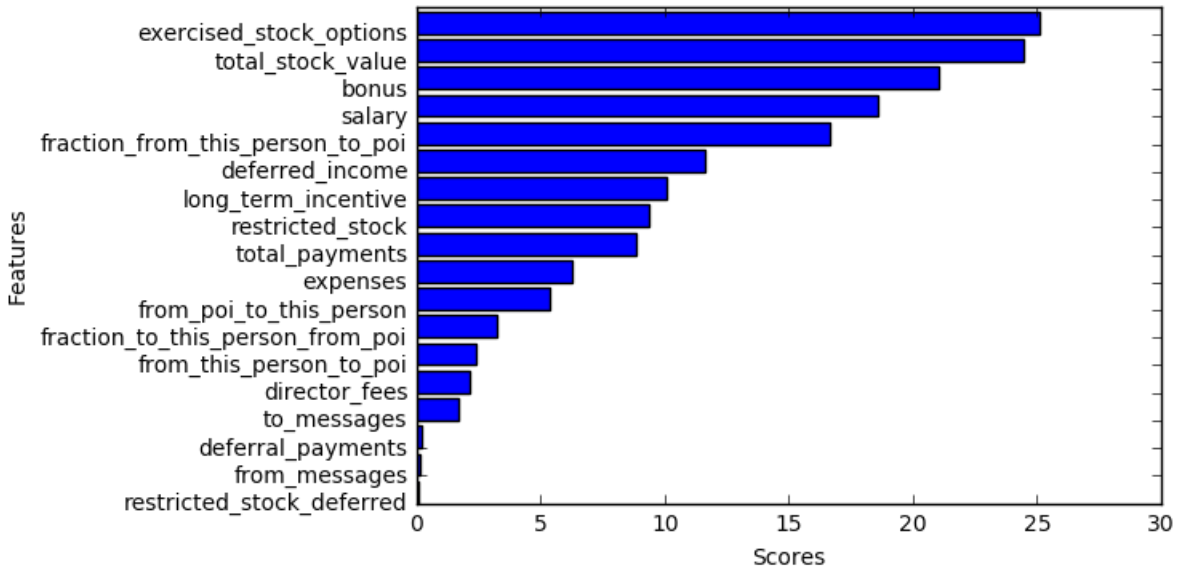
Selection of best features with their scores

SelectKBest estimator (referred to as 'SKB' in the pipeline) was used to select k best features. In this section we see a scoring as a basis of their selection.

In [99]:
```
clf.named_steps['SKB'].get_support(indices=True)
score_array = clf.named_steps['SKB'].scores_
print score_array
```

```
[ 18.57570327  21.06000171   0.21705893   8.86672154  25.09754153
   9.34670079   0.06498431  24.46765405   6.23420114   2.10765594
  11.59554766  10.07245453   1.69882435   0.1641645    5.34494152
   2.42650813  16.64170707   3.21076192]
```

In [100]:
```
# -------
# Make a local copy of features_list and remove the label from it.
# That gives us the list of features that we passed to the estimators in the p
ipleline
# -------
flist = features_list[:]
flist.remove('poi')
len(flist)
indices = np.argsort(score_array)[::]
x_features = []
y_scores = []
for i in indices:
    x_features.append(flist[i])
    y_scores.append(score_array[i])
x_pos = np.arange(len(x_features))
plt.barh(x_pos, y_scores)
plt.yticks(x_pos, x_features)
plt.xlabel("Scores")
plt.ylabel("Features")
plt.show()
```



The above chart represents the selected features in their relative abilities to influence the model. It is interesting to note that one of the new 'engineered' features ('fraction_from_this_person_to_poi') ranks among the top influencers! Intutively, this means that fraction of all the outgoing e-mails that were sent to the POIs is, indeed, an important consideration to identify a POI.

Dimensionality Reduction (PCA)

Curious to know the effect of dimensionality reduction. I wanted to know the dimensions, if they are orthogonal to each other and how much of variance exists in each dimension.

In [102]:
```
# Find out the dimensions
pc_array = clf.named_steps['PCA'].components_
# How many of them
no_of_pcs = len(pc_array)
print "Reduced number of dimensions is:", no_of_pcs
print
# How much of variances are explained by each dimension
explained_var_array = clf.named_steps['PCA'].explained_variance_
print "Two of the PCs are:"
print pc_array[0]
print pc_array[1]
print
# Trying to verify that the dimensions are indeed orthogonal,
# i.e. their scalar product is '0'
for i0 in range(no_of_pcs):
    for i1 in range(no_of_pcs):
        if i0 < i1:
           sum_of_scalar_prod = 0
           for j in range(10):
              sum_of_scalar_prod += pc_array[i0][j] * pc_array[i1][j]
           #if abs(sum_of_scalar_prod) < .000000000000001:
           if abs(sum_of_scalar_prod) < 1.0e-15:
              sum_of_scalar_prod = 0
           print "first_PC = {}, second_PC = {}, scalar product = {}".format(
                            i0, i1, sum_of_scalar_prod)

print "\nVariance explained by each dimension:\n", explained_var_array
```

```
Reduced number of dimensions is: 6

Two of the PCs are:
[ 0.50298107  0.39228911  0.1911616   0.3225715   0.25540209  0.31503301
  0.27748722 -0.26637025  0.28372024  0.24412844]
[ 0.06145362 -0.03944369 -0.05158439 -0.16382921 -0.04287921 -0.12920097
  0.9061908   0.28947139 -0.13159003 -0.15602493]

first_PC = 0, second_PC = 1, scalar product = 0
first_PC = 0, second_PC = 2, scalar product = 0
first_PC = 0, second_PC = 3, scalar product = 0
first_PC = 0, second_PC = 4, scalar product = 0
first_PC = 0, second_PC = 5, scalar product = 0
first_PC = 1, second_PC = 2, scalar product = 0
first_PC = 1, second_PC = 3, scalar product = 0
first_PC = 1, second_PC = 4, scalar product = 0
first_PC = 1, second_PC = 5, scalar product = 0
first_PC = 2, second_PC = 3, scalar product = 0
first_PC = 2, second_PC = 4, scalar product = 0
first_PC = 2, second_PC = 5, scalar product = 0
first_PC = 3, second_PC = 4, scalar product = 0
first_PC = 3, second_PC = 5, scalar product = 0
first_PC = 4, second_PC = 5, scalar product = 0

Variance explained by each dimension:
[ 0.09242219  0.03672226  0.03249784  0.02520724  0.01710748  0.00985413]
```

In this case PCA reduced the dimensions of this data set to six dimensions (from the original 10 dimensions as selected by SelectKBest algorithm). As expected all of them are orthogonal (scalar product is '0') and the maximum variance explained along any one dimension is 9.24%.

The reduction in dimensionality inherently results into loss of information. However, in this case the loss of information may be negligible (variance explained by the dropped dimensions should be less than .985%.

At the tail end of the pipe

At the tail end of the pipe the 'trained' model has been created. I will now dump the model into a few pickle databases for others to review.

In [103]:
```
# Dump the classifier, dataset and features list
dump_classifier_and_data(clf, my_dataset, features_list)
```

Validation Strategy

Validation is the step where we evaluate the performance of the model over a given dataset. Simply put, the relevant question we ask is - how many data points have been mis-classified (false positive and false negative). The validation is important since the user of the model needs to gain confidence to be able to use the model to classify any future new data.

One usual way to evaluate the performace of the model is to split the data into training and test sets, predict the labels of the test features and compare them with the actual labels.

In [104]:
```
features_train, features_test, labels_train, labels_test = \
    train_test_split(features, labels, test_size=0.2, random_state=42)
pred = clf.predict(features_test)
print classification_report(labels_test, pred)
print confusion_matrix(labels_test, pred)
```

```
             precision    recall  f1-score   support

        0.0       0.96      0.93      0.94        27
        1.0       0.33      0.50      0.40         2

avg / total       0.92      0.90      0.91        29

[[25  2]
 [ 1  1]]
```

In the above approach 20% of the original dataset (29 records) was allocated as the test set. The classification_report function reported three metrics on this test set following a comparison between the actual and predicted labels, as follows:

1. **Precision**: this is the ratio of the data points that were correctly predicted to be belonging to a label and all the data points that were predicted to be belonging to the same label (including 'false' predictions).

   In general terms, 'Precision' indicates the model's ability to accurately identify the targets. A higher value of this metric indicates that the model is able to keep the false positives to minimum. According to the above classification report, it shows that out of all the people that have been identified as POIs, 33% of them are correctly recognized and the rest have been falsely flagged.
2. **Recall**: this is the ratio of the data points that were correctly predicted to be belonging to a label and all the data points that actually belong to the same label (including those that have been mis-classified)

   This metric represents the model's ability to stay away from mis-classifying the targets. A higher value of this figure indicates the ability to keep the false negatives to a minimum. According to the above classification report, the model is recognizing 50% of the POIs in the test set correctly and the other 50% of the POIs are given a clean cheat.
3. **f1_score**: this is an weighted ratio involving the precision and recall metrics. This combination sort of balances out the effect of skewed metrics (i.e. high precision/low recall or low precision/high recall). This is derived by computing the harmonic mean of Precision and Recall and serves as a single metric through which the performance of the model is described.

   This metric is also an indication of how close the Precision and Recall are. If there is a wide gap between these two, then it is a good tradeoff to sacrifice part of the higher metric in favor of some increase in the other one, thus resulting in an increase of the f1_score overall. Referring to the classification report above, both the Precision and Recall for the non-POI are high. This is expected since the model would be mostly right if it predicts a person as a non-POI. This results into a higher value of the f1-score for the non-POIs. On the other hand, although the Precision and Recall are not great for the POIs, they are somewhat close. Now, consider - we sacrifice the Precision a little bit, say 0.45 and improve the Recall to 0.4. The overall performance or the f1-score improves to 0.42 (a 2% increase than the current performance). This is a little bit of performance gain - a positive tradeoff!

In this specific case the confusion_matrix can be interpreted as follows:

1. 25 non-pois have been correctly predicted as non-pois (True negative)
2. 1 poi has been correctly predicted as a poi (True positive)
3. 1 no-poi has been falsely predicted as a poi (False positive)
4. 1 poi has been falsely predicted as non-poi (False negative)

A word of caution

The performance of the model/classifier, based on the above classification report and confusion_matrix appear to be too good. The reason is we are working with only one test set. Also remember that we are working with a very sparse dataset where only 12% individuals are POIs.

Classic mistakes can happen when the model is not trained with a large number of training sets. The model must necessarily have the opportunity to face various different kind of patterns to be able to make an optimal decision on a new data. This was a challenge for a small and skewed (majority as non-POI) Enron dataset. We understand that the model will mostly be right (high accuracy) if it has a bias to classify a person as a non-POI.

In order to handle this situation I used the StratifiedShuffleShit utility to create a large number of combination of training/test sets ('folds'). This results into the model being trained with a large number of datasets and validate with a large number of test sets. I am using the 'test_classifier' function from the supplied tester.py file over 100 folds.

```
In [105]: test_classifier(clf, my_dataset, features_list, folds=100)
```

```
Pipeline(steps=[('scaling', MinMaxScaler(copy=True, feature_range=(0, 1))),
 ('SKB', SelectKBest(k=10, score_func=<function f_classif at 0x0000000009ABD7
48>)), ('PCA', PCA(copy=True, iterated_power='auto', n_components=6, random_s
tate=None,
  svd_solver='auto', tol=0.0, whiten=True)), ('GNB', GaussianNB(priors=Non
e))])
        Accuracy: 0.82159       Precision: 0.31847      Recall: 0.35907 F1:
 0.33755        F2: 0.35014
        Total predictions: 4400 True positives:  200    False positives:  428
False negatives:  357   True negatives: 3415
```

According this evaluation (with 'test_classifier' function), the Precision (.31847), Recall (.35907) and F1-score (.33755) values are far from being as impressive as shown in the previous paragraph. However, these values are more realisting as a reaizable performance achieved by the model.

Conclusion

This report shows an approach of creating and evaluating a model to identify Enron persons of interest. Four (4) different classifiers have been used, of which Naive Bayes classifer demonstrates the best performance as indicated by it F1-score. The following table shows the performance of the four classifiers:

Classifier	Accuracy	Precision	Recall	F1	F2
Naive Bayes	0.82159	0.31847	0.35907	0.33755	0.35014
Decision Tree	0.82341	0.28764	0.26750	0.27721	0.27130
Random Forest	0.85295	0.30435	0.12567	0.17789	0.14239
SVM	0.82659	0.24378	0.17594	0.20438	0.18631

Supplemental Code

This section includes the code for the other three classifiers that I have tried. Please refer to the section "Choice of Parameters" where I have described the parameters I am considering for various classifiers.

Decision Tree Classifier

In [ ]:
```
scaler = MinMaxScaler()
skb = SelectKBest()
pca = PCA()
dt = DecisionTreeClassifier()

# ---------------------------
# Pipeline with Decision Tree classifier
# ---------------------------
Pipe = Pipeline(steps=[('scaling',scaler),("SKB", skb),("PCA", pca),("DT",
dt)])

# ------------------------
# Parameter selection
# ------------------------
param_grid = {"SKB__k":[10, 12, 16, 18], "PCA__n_components":[6, 8, 10], "PCA_
_whiten":[True],
              "DT__min_samples_split":[10, 15]}

# -------------------------------------------------
# Split the dataset into 100 folds with 30% test set
# -------------------------------------------------
stashsp = StratifiedShuffleSplit(n_splits=100, test_size=0.3, random_state=42)

gs = GridSearchCV(Pipe, param_grid=param_grid, scoring='f1', n_jobs=1, cv=stas
hsp)
gs.fit(features, labels)
print "Best parameters from GridSearchCV are:", gs.best_params_
print "Best estimator found by grid search:", gs.best_estimator_
clf = gs.best_estimator_

# Split the dataset into traning/test sets and perform usual prediction
features_train, features_test, labels_train, labels_test = \
    train_test_split(features, labels, test_size=0.2, random_state=42)
pred = clf.predict(features_test)
print classification_report(labels_test, pred)
print confusion_matrix(labels_test, pred)

# Use more intelligent splitting with StratifiedShuffleSplit and evaluate the
 performance
test_classifier(clf, my_dataset, features_list, folds=100)
```

Random Forest Classifier

In [ ]:
```
scaler = MinMaxScaler()
skb = SelectKBest()
pca = PCA()
randomf = RandomForestClassifier()

Pipe = Pipeline(steps=[('scaling',scaler),("SKB", skb),("PCA", pca),("randfore
st", randomf)])

param_grid = {"SKB__k":[10, 12, 16, 18], "PCA__n_components":[6, 8, 10], "PCA_
_whiten":[True],
              "randforest__n_estimators":[8, 10, 12], "randforest__max_feature
s":[2, 4]}

# -------------------------------------------------
# StratifiedShuffleSplit to create 100 folds with 30% test set
# -------------------------------------------------
stashsp = StratifiedShuffleSplit(n_splits=100, test_size=0.3, random_state=42)

gs = GridSearchCV(Pipe, param_grid=param_grid, scoring='f1', n_jobs=1, cv=stas
hsp)
gs.fit(features, labels)
print "Best parameters from GridSearchCV are:", gs.best_params_
print "Best estimator found by grid search:", gs.best_estimator_
clf = gs.best_estimator_

# Split the dataset into traning/test sets and perform usual prediction
features_train, features_test, labels_train, labels_test = \
    train_test_split(features, labels, test_size=0.3, random_state=42)
pred = clf.predict(features_test)
print classification_report(labels_test, pred)
print confusion_matrix(labels_test, pred)

# Use more intelligent splitting with StratifiedShuffleSplit and evaluate the
 performance
test_classifier(clf, my_dataset, features_list, folds=100)
```

SVM Classifier

In [ ]:
```
scaler = MinMaxScaler()
skb = SelectKBest(f_classif)
pca = PCA()
svr = SVC()

Pipe = Pipeline(steps=[('scaling',scaler),("SKB", skb),("PCA", pca),("svm", sv
r)])

param_grid = {"SKB__k":[10, 12, 16, 18],
              "PCA__n_components":[6, 8, 10], "PCA__whiten":[True],
              "svm__kernel":['rbf'], "svm__C":[1, 10, 100]}

# -------------------------------------------------
# StratifiedShuffleSplit to create 100 folds with 30% test set
# -------------------------------------------------
stashsp = StratifiedShuffleSplit(n_splits=100, test_size=0.3, random_state=42)

gs = GridSearchCV(Pipe, param_grid=param_grid, scoring='f1', n_jobs=1, cv=stas
hsp)
gs.fit(features, labels)
print "Best parameters from GridSearchCV are:", gs.best_params_
print "Best estimator found by grid search:", gs.best_estimator_
clf = gs.best_estimator_

# Split the dataset into traning/test sets and perform usual prediction
features_train, features_test, labels_train, labels_test = \
    train_test_split(features, labels, test_size=0.3, random_state=42)
pred = clf.predict(features_test)
print classification_report(labels_test, pred)
print confusion_matrix(labels_test, pred)

# Use more intelligent splitting with StratifiedShuffleSplit and evaluate the
 performance
test_classifier(clf, my_dataset, features_list, folds=50)
```

References

1. scikit-learn documentation http://scikit-learn.org (http://scikit-learn.org)
2. Discussion forum at Udacity
3. What is an intuitive explanation of F-sore https://www.quora.com/What-is-an-intuitive-explanation-of-F-score (https://www.quora.com/What-is-an-intuitive-explanation-of-F-score)
4. "Hands-On Machine Learning with Scikit-Learn & TensorFlow", Aurelien Geron, O'Reilly Book, March 2017