PROJECT: Identify Fraud from Enron Email

Introduction

In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives.

The objective of this project is to build an algorithm to identify Enron employees who may have committed fraud based on the public Enron financial and email dataset. Such employees are referred to as "person's of interest", or, POIs.

Machine Learning is extremely useful in problems like this as it is able to work with relatively high dimensional data and find any relationships that may exist. This can then be used to make predictions about the data, such as classifying a person as a person of interest as in this case.

Dataset

In [1]:

```
# Import required modules

import sys
import pickle
import pandas as pd
import numpy as np
from time import time
sys.path.append("../tools/")

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

from sklearn.feature_selection import SelectKBest,f_classif
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier,AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC,LinearSVC
from sklearn.pipeline import Pipeline
from sklearn.grid_search import GridSearchCV

from collections import OrderedDict

import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

```
C:\ProgramData\Anaconda3\envs\DAND\lib\site-packages\sklearn\cross_validatio
n.py:41: DeprecationWarning: This module was deprecated in version 0.18 in f
avor of the model_selection module into which all the refactored classes and
functions are moved. Also note that the interface of the new CV iterators ar
e different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
C:\ProgramData\Anaconda3\envs\DAND\lib\site-packages\sklearn\grid_search.py:
42: DeprecationWarning: This module was deprecated in version 0.18 in favor
of the model_selection module into which all the refactored classes and func
tions are moved. This module will be removed in 0.20.
  DeprecationWarning)
```

In [2]:

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

In [3]:

```
# Load the dictionary into a dataframe and examine it
dataset_df=pd.DataFrame.from_dict(data_dict,orient='index')
```

In [4]:

```
dataset_df.head()
```

Out[4]:

	salary	to_messages	deferral_payments	total_payments	exercised_stock_option
ALLEN PHILLIP K	201955	2902	2869717	4484442	172954
BADUM JAMES P	NaN	NaN	178980	182466	25781
BANNANTINE JAMES M	477	566	NaN	916197	404615
BAXTER JOHN C	267102	NaN	1295738	5634343	668054
BAY FRANKLIN R	239671	NaN	260455	827696	Nal

5 rows × 21 columns

In [5]:

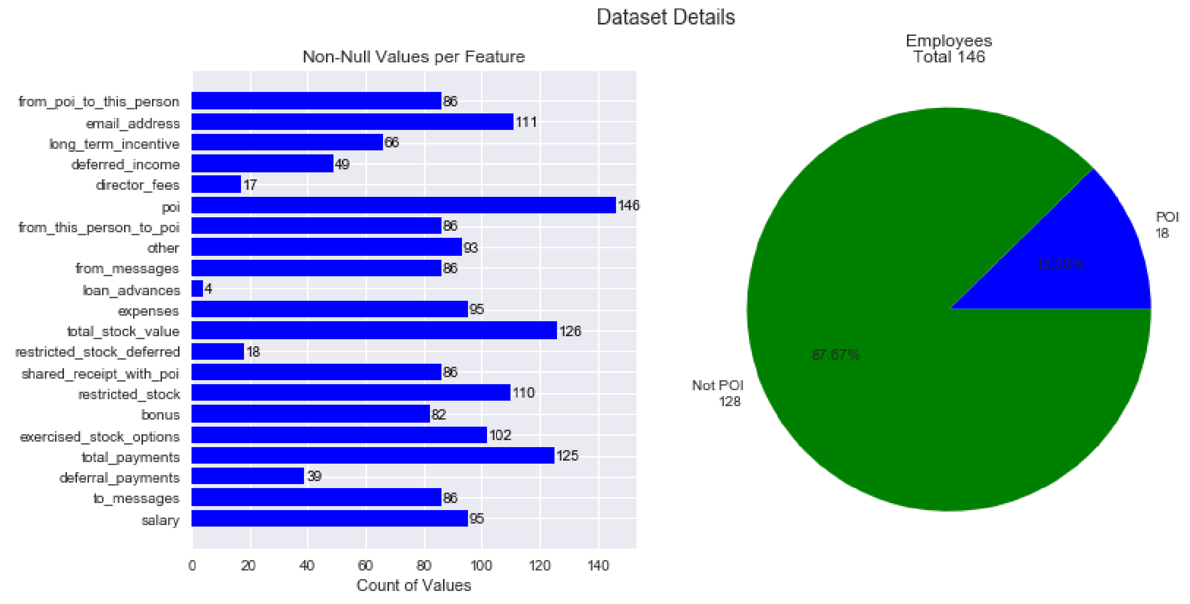
```
# Replace 'NaN' string with Null (NaN)
dataset_df.replace('NaN',np.nan,inplace=True)
```

In [6]:

```
# Dataset details
num_employees=len(dataset_df)
num_poi=len(dataset_df[dataset_df['poi']==True])
num_non_poi=num_employees-num_poi
num_vals=num_employees-dataset_df.isnull().sum()
```

In [7]:

```
# Plot POI and feature count
plt.figure(figsize=(12,6))
plt.suptitle('Dataset Details',fontsize=14)
# Left plot
plt.subplot(1,2,1)
plt.barh(range(len(num_vals.index)),num_vals,height=-0.8,color=['blue'])
plt.yticks(range(len(num_vals.index)),num_vals.index)
for i,v in enumerate(num_vals):
    plt.text(v+0.4,i-0.2,str(v),color='black')
    plt.title('Non-Null Values per Feature')
plt.xlabel('Count of Values')
# Right plot
plt.subplot(1,2,2)
plt.title('Employees\nTotal '+str(num_employees))
plt.pie([num_poi,num_non_poi],labels=['POI\n'+str(num_poi),'Not POI\n'+str(num_non_poi)],
        autopct='%.2f%%',colors=['blue','green'])
plt.axis('equal')
plt.show()
```



Features

In [8]:

```
### 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".
# You will need to use more features
```

In [9]:

```
# Which features have less than 10% data
df=dataset_df.count()/len(dataset_df)<0.1
list(df[df==True].index)
```

Out[9]:

```
['loan_advances']
```

In [10]:

```
# Which features have less than 10% data for POI
df=dataset_df[dataset_df['poi']==True].count()/dataset_df.count()<0.1
list(df[df==True].index)
```

Out[10]:

```
['restricted_stock_deferred', 'director_fees']
```

In [11]:

```
# Selected features
POI_label=['poi'] # Boolean, represented as integer

financial_features=['salary','deferral_payments','total_payments','bonus','deferred_income'
                    'total_stock_value','expenses','exercised_stock_options','other',
                    'long_term_incentive','restricted_stock'] # Units are in US dollars

email_features=['to_messages','from_poi_to_this_person','from_messages',
                'from_this_person_to_poi','shared_receipt_with_poi'] # Units are number of

# We will ignore:
# 'email_address' - not numerical data
# 'restricted_stock_deferred' and 'director_fees' - less than 10% data for POI
# 'loan_advances' - less than 10% data

features_list=(POI_label+financial_features+email_features)
print 'Number of initial features: ',len(features_list)
```

```
Number of initial features:  17
```

Correct Dataset

In [12]:

```
#Drop email address since we are not using it in this analysis
dataset_df.drop('email_address',axis=1,inplace=True)
```

In [13]:

```
#Read in data from supplied enron61702insiderpay.pdf file for comparison
#(edited so column names and employee names coincide and converted to csv)
pdf=pd.read_csv('PDF.csv',index_col=0)
```

In [14]:

```
pdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 146 entries, ALLEN PHILLIP K to Total
Data columns (total 14 columns):
salary                       95 non-null float64
bonus                        82 non-null float64
long_term_incentive          66 non-null float64
deferred_income              50 non-null float64
deferral_payments            38 non-null float64
loan_advances                4 non-null float64
other                        92 non-null float64
expenses                     97 non-null float64
director_fees                16 non-null float64
total_payments               125 non-null float64
exercised_stock_options      101 non-null float64
restricted_stock             111 non-null float64
restricted_stock_deferred    18 non-null float64
total_stock_value            126 non-null float64
dtypes: float64(14)
memory usage: 16.5+ KB
```

In [15]:

```
#Calculate differences between initial dataset and correct data
df=dataset_df.rsub(pdf).fillna(0.0)
df.sum()
```

Out[15]:

```
bonus                               0.0
deferral_payments                   0.0
deferred_income                     0.0
director_fees                   99215.0
exercised_stock_options      12851800.0
expenses                            0.0
from_messages                       0.0
from_poi_to_this_person             0.0
from_this_person_to_poi             0.0
loan_advances                       0.0
long_term_incentive                 0.0
other                               0.0
poi                                 0.0
restricted_stock              5208980.0
restricted_stock_deferred   -18148966.0
salary                              0.0
shared_receipt_with_poi             0.0
to_messages                         0.0
total_payments              -15417641.0
total_stock_value                   0.0
dtype: float64
```

In [16]:

```
#Correct dataset
for row in range(len(df)):
    for col, vals in df.iteritems():
        if vals[row] != 0.0:
            old_val=dataset_df.loc[df.index[row],col]
            new_val=pdf.loc[df.index[row],col]
            dataset_df.loc[df.index[row],col]=new_val
            print df.index[row],col,'corrected from',old_val,'to',new_val
```

```
BELFER ROBERT director_fees corrected from 3285.0 to 102500.0
BELFER ROBERT restricted_stock_deferred corrected from 44093.0 to -44093.0
BELFER ROBERT total_payments corrected from 102500.0 to 3285.0
BHATNAGAR SANJAY exercised_stock_options corrected from 2604490.0 to 1545629
0.0
BHATNAGAR SANJAY restricted_stock corrected from -2604490.0 to 2604490.0
BHATNAGAR SANJAY restricted_stock_deferred corrected from 15456290.0 to -260
4490.0
BHATNAGAR SANJAY total_payments corrected from 15456290.0 to 137864.0
```

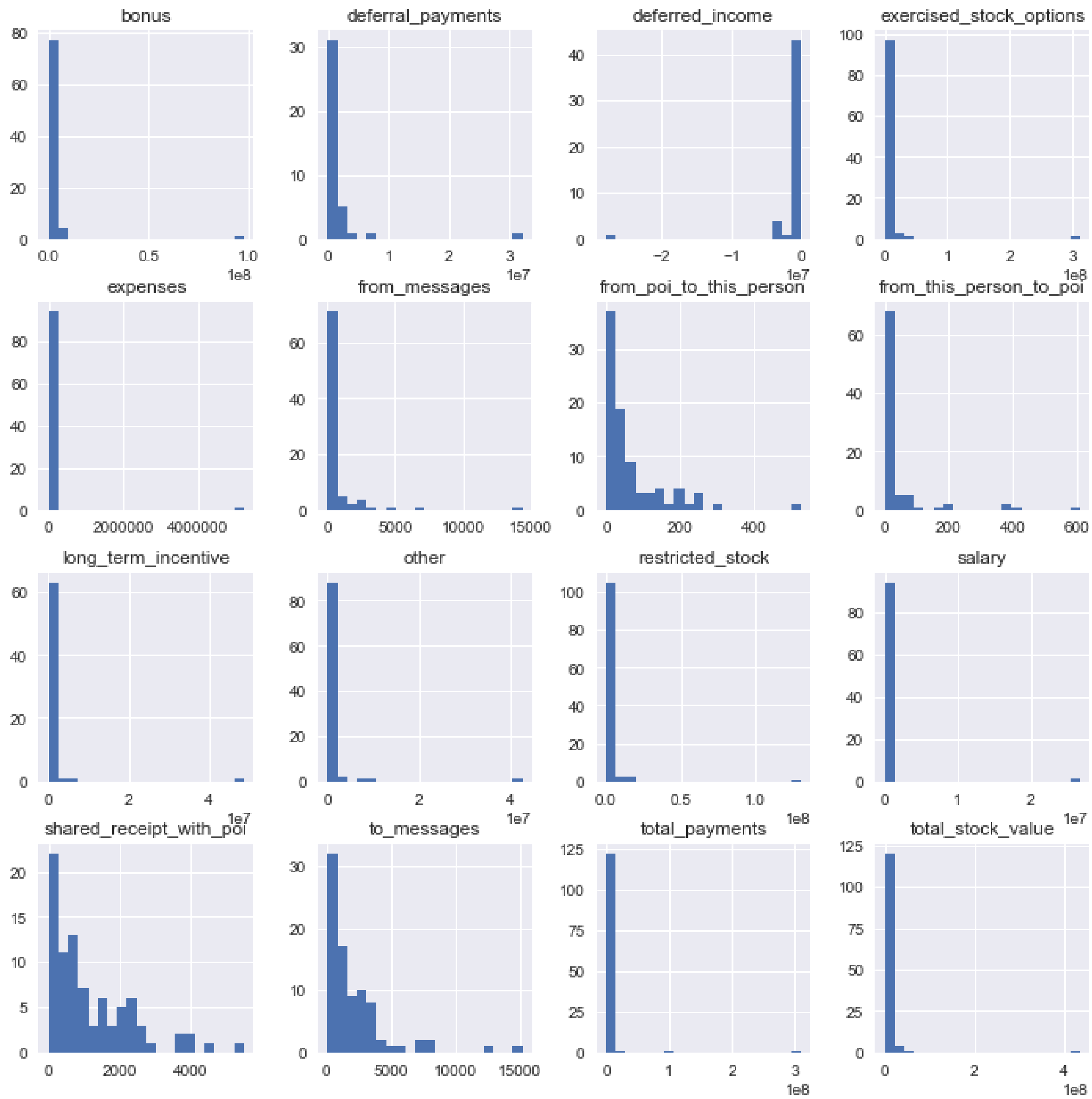
Remove Outliers

In [17]:

```
### Task 2: Remove outliers
```

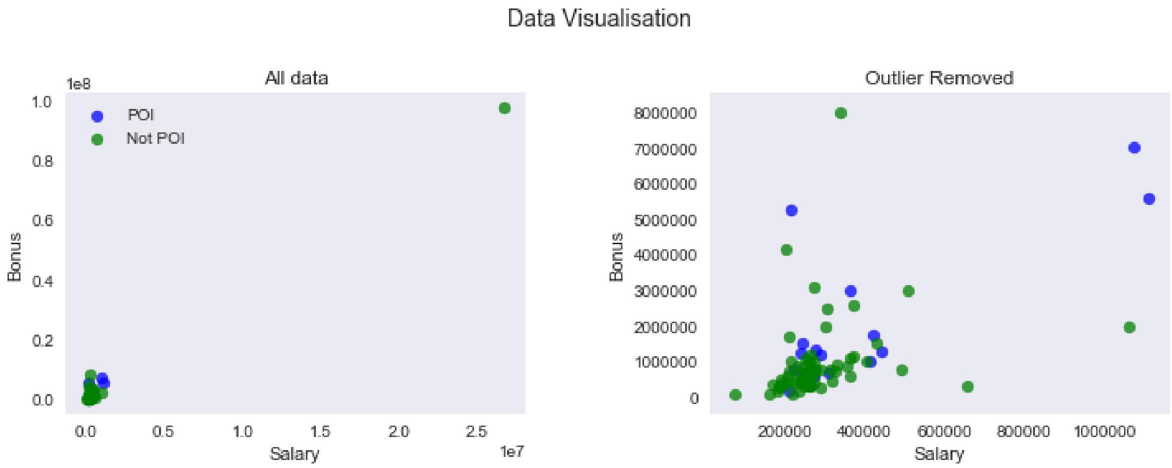
In [18]:

```
# Visualise data to show any outliers
dataset_df[features_list[1:]].hist(figsize=(12,12),bins=20);
```



In [19]:

```
#Plot of data with suspected outlier
plt.figure(figsize=(12,4))
plt.suptitle('Data Visualisation',fontsize=14)
# Left plot
plt.subplot(1,2,1)
x1=dataset_df[dataset_df['poi'] == True]['salary']
y1=dataset_df[dataset_df['poi'] == True]['bonus']
x2=dataset_df[dataset_df['poi'] == False]['salary']
y2=dataset_df[dataset_df['poi'] == False]['bonus']
plt.scatter(x1,y1,color='blue',alpha=0.75)
plt.scatter(x2,y2,color='green',alpha=0.75)
plt.title('All data')
plt.ylabel('Bonus')
plt.xlabel('Salary')
plt.legend(('POI','Not POI'))
plt.grid()
# Right plot
plt.subplot(1,2,2)
outlier=dataset_df['salary'].idxmax()
df=dataset_df[dataset_df.index != outlier]
x1=df[df['poi'] == True]['salary']
y1=df[df['poi'] == True]['bonus']
x2=df[df['poi'] == False]['salary']
y2=df[df['poi'] == False]['bonus']
plt.scatter(x1,y1,color='blue',alpha=0.75)
plt.scatter(x2,y2,color='green',alpha=0.75)
plt.title('Outlier Removed')
plt.ylabel('Bonus')
plt.xlabel('Salary')
plt.subplots_adjust(wspace=0.4,top=0.8)
plt.grid()
plt.show()
```



In [20]:

```
# Outlier
print 'Outlier found in row:',outlier
```

```
Outlier found in row: TOTAL
```

In [21]:

```
# Check for entries with no numerical data
df=dataset_df.drop(['poi'],axis=1).nunique(axis=1).sort_values()
df.head()
```

Out[21]:

```
LOCKHART EUGENE E              0
GRAMM WENDY L                  1
WHALEY DAVID A                 1
WROBEL BRUCE                   1
THE TRAVEL AGENCY IN THE PARK    1
dtype: int64
```

In [22]:

```
# Empty row
dataset_df.loc['LOCKHART EUGENE E']
```

Out[22]:

```
salary                         NaN
to_messages                    NaN
deferral_payments              NaN
total_payments                 NaN
exercised_stock_options        NaN
bonus                          NaN
restricted_stock               NaN
shared_receipt_with_poi        NaN
restricted_stock_deferred      NaN
total_stock_value              NaN
expenses                       NaN
loan_advances                  NaN
from_messages                  NaN
other                          NaN
from_this_person_to_poi        NaN
poi                          False
director_fees                  NaN
deferred_income                NaN
long_term_incentive            NaN
from_poi_to_this_person        NaN
Name: LOCKHART EUGENE E, dtype: object
```

In [23]:

```
# Drop the following:
# TOTAL - Spreadsheet aggregation included by mistake (outlier)
# LOCKHART EUGENE E - Does not contain any numerical data
# THE TRAVEL AGENCY IN THE PARK - Not an individual (Alliance Worldwide - co-owned by the s

dataset_df.drop(['TOTAL','LOCKHART EUGENE E','THE TRAVEL AGENCY IN THE PARK'],axis=0,inplac
```

Engineered Features

We will engineer several new features in order to provide further insights into the data.

In [24]:

```
### Task 3: Create new feature(s)
```

In [25]:

```
# New financial features:
dataset_df['fraction_bonus_salary']=dataset_df['bonus']/dataset_df['salary']
dataset_df['fraction_bonus_total']=dataset_df['bonus']/dataset_df['total_payments']
dataset_df['fraction_salary_total']=dataset_df['salary']/dataset_df['total_payments']
dataset_df['fraction_stock_total']=dataset_df['total_stock_value']/dataset_df['total_paymer
```

In [26]:

```
# New email features:
dataset_df['fraction_to_poi']=dataset_df['from_this_person_to_poi']/dataset_df['from_messag
dataset_df['fraction_from_poi']=dataset_df['from_poi_to_this_person']/dataset_df['to_messag
```

In [27]:

```
# Add new features to feature list
new_features_list=['fraction_bonus_salary',
                        'fraction_bonus_total',
                        'fraction_salary_total',
                        'fraction_stock_total',
                        'fraction_to_poi',
                        'fraction_from_poi']
extended_features_list=features_list+new_features_list
print 'Number of extended features: ',len(extended_features_list)
```

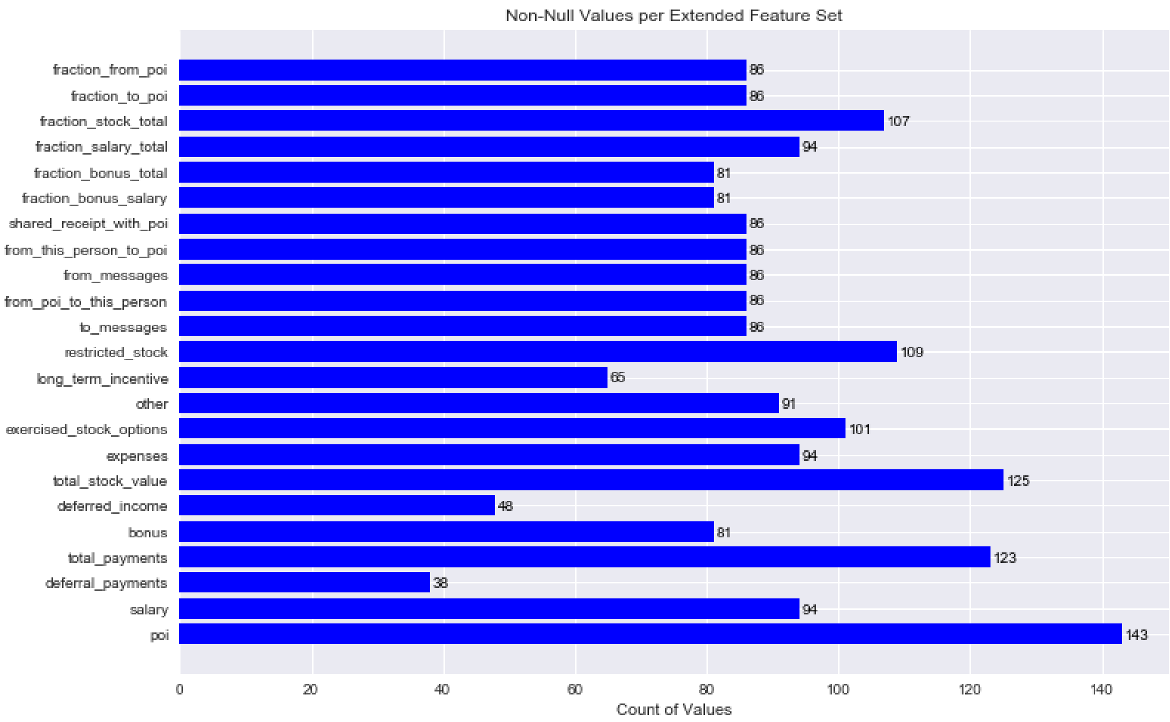
```
Number of extended features:  23
```

In [28]:

```
# Cleaned and trimmed dataset
num_employees=len(dataset_df[extended_features_list])
num_poi=len(dataset_df[dataset_df['poi']==True])
num_non_poi=num_employees-num_poi
num_vals=num_employees-dataset_df[extended_features_list].isnull().sum()
```

In [29]:

```
#Plot POI and feature count
plt.figure(figsize=(12,8))
#plt.subplot(1,2,1)
plt.barh(range(len(num_vals.index)),num_vals,height=-0.8,color=['blue'])
plt.yticks(range(len(num_vals.index)),num_vals.index)
for i,v in enumerate(num_vals):
    plt.text(v+0.4,i-0.2,str(v),color='black')
    plt.title('Non-Null Values per Extended Feature Set')
plt.xlabel('Count of Values')
plt.show()
```



In [30]:

```
# Replace Null (NaN) entries with 0.0 to prevent errors in algorithms
dataset_df.fillna(value=0.0,inplace=True)
```

In [31]:

```
dataset_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 143 entries, ALLEN PHILLIP K to YEAP SOON
Data columns (total 26 columns):
salary                        143 non-null float64
to_messages                   143 non-null float64
deferral_payments             143 non-null float64
total_payments                143 non-null float64
exercised_stock_options       143 non-null float64
bonus                         143 non-null float64
restricted_stock              143 non-null float64
shared_receipt_with_poi       143 non-null float64
restricted_stock_deferred     143 non-null float64
total_stock_value             143 non-null float64
expenses                      143 non-null float64
loan_advances                 143 non-null float64
from_messages                 143 non-null float64
other                         143 non-null float64
from_this_person_to_poi       143 non-null float64
poi                           143 non-null bool
director_fees                 143 non-null float64
deferred_income               143 non-null float64
long_term_incentive           143 non-null float64
from_poi_to_this_person       143 non-null float64
fraction_bonus_salary         143 non-null float64
fraction_bonus_total          143 non-null float64
fraction_salary_total         143 non-null float64
fraction_stock_total          143 non-null float64
fraction_to_poi               143 non-null float64
fraction_from_poi             143 non-null float64
dtypes: bool(1), float64(25)
memory usage: 28.6+ KB
```

In [32]:

```
### Store to my_dataset for easy export below.
my_dataset=dataset_df.to_dict('index')

### Extract original features and labels from dataset
data=featureFormat(my_dataset,features_list,sort_keys=True)
labels,features=targetFeatureSplit(data)

### Extract extended features and labels from dataset
data=featureFormat(my_dataset,extended_features_list,sort_keys=True)
labels2,features2=targetFeatureSplit(data)
```

Feature Selection

Feature selection is the process by which the machine learning algorithm automatically selects those features select those features that have the strongest relationship with the target variable. Feature selection can be used either to improve accuracy scores or to increase performance on high-dimensional data.

Three benefits of performing feature selection are:

- Reduces Overfitting: Less irrelevant data means less opportunity to make decisions based on noise.
- Improves Accuracy: Less misleading data means modeling accuracy improves.
- Reduces Training Time: Less data means algorithms train faster.

We will use *SelectKBest* to select features according to the *k* highest scores, according to the *f_classif* scoring function which computes the ANOVA F-value for the data. In this instance we use *k=12*.

In [33]:

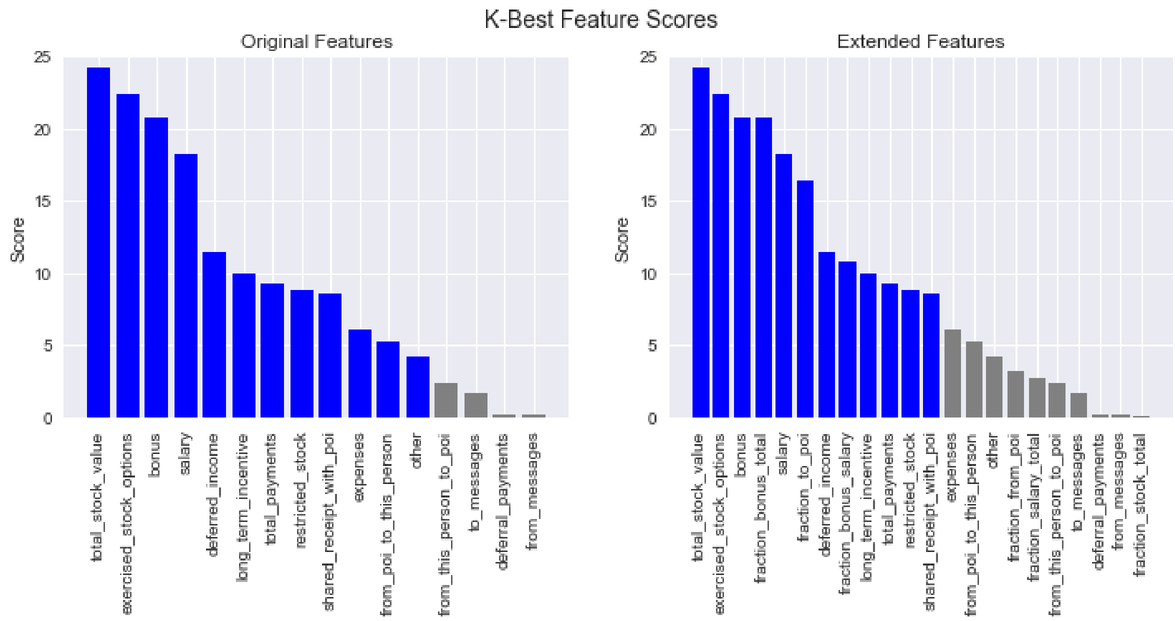
```
# Select K-Best features
n=12

k_best1=SelectKBest(score_func=f_classif,k=n)
k_best1.fit(features,labels)
feature_scores1=zip(features_list[1:],k_best1.scores_)
k_best_features1=OrderedDict(sorted(feature_scores1,key=lambda x: x[1],reverse=True))

k_best2=SelectKBest(score_func=f_classif,k=n)
k_best2.fit(features2,labels2)
feature_scores2=zip(extended_features_list[1:],k_best2.scores_)
k_best_features2=OrderedDict(sorted(feature_scores2,key=lambda x: x[1],reverse=True))
```

In [34]:

```
#Plot KBest feature scores
plt.figure(figsize=(12,4))
plt.suptitle('K-Best Feature Scores',fontsize=14)
# Left plot
plt.subplot(1,2,1)
plt.bar(range(len(k_best_features1)),k_best_features1.values(),
        align='center',color=['blue']*n+['grey']*(len(k_best_features1)-n))
plt.xticks(range(len(k_best_features1)),k_best_features1.keys(),rotation='vertical')
plt.title('Original Features')
plt.ylabel('Score')
plt.ylim(0,25)
# Right plot
plt.subplot(1,2,2)
plt.bar(range(len(k_best_features2)),k_best_features2.values(),
        align='center',color=['blue']*n+['grey']*(len(k_best_features2)-n))
plt.xticks(range(len(k_best_features2)),k_best_features2.keys(),rotation='vertical')
plt.title('Extended Features')
plt.ylabel('Score')
plt.ylim(0,25)
plt.show()
```



Feature Scaling

Scaling is a common requirement for many machine learning algorithms. Some algorithms may behave badly if the features are not more or less normally distributed.

While *Decision Trees* and *Random Forests* classifiers are able to handle un-scaled features, other classifiers, like *SVM* cannot.

Since our data is highly skewed, we will employ *StandardScaler* where appropriate to scale our data.

In [35]:

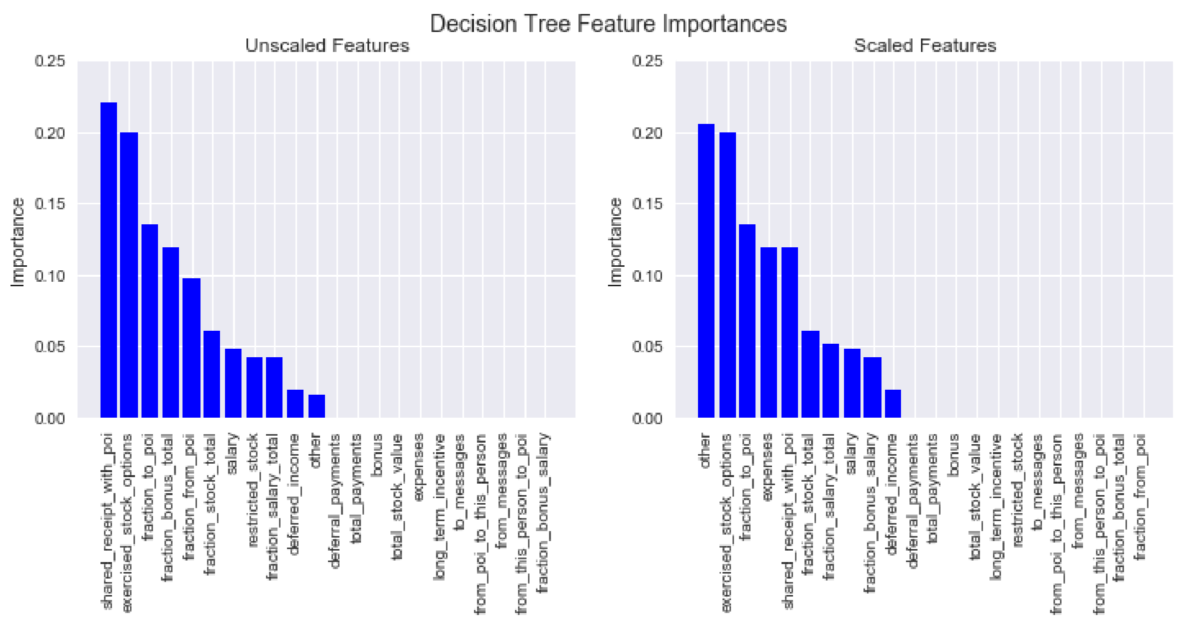
```
# Fit Decision Tree to unscaled features and get feature importances
clf1=DecisionTreeClassifier()
clf1=clf1.fit(features2,labels2)
feature_importances1=zip(extended_features_list[1:],clf1.feature_importances_)
important_features1=OrderedDict(sorted(feature_importances1,key=lambda x: x[1],reverse=True

# Scale features
scaler=StandardScaler(copy=True)
scaled_features=scaler.fit_transform(features2)

# Fit Decision Tree to scaled features and get feature importances
clf2=DecisionTreeClassifier()
clf2=clf2.fit(scaled_features,labels2)
feature_importances2=zip(extended_features_list[1:],clf2.feature_importances_)
important_features2=OrderedDict(sorted(feature_importances2,key=lambda x: x[1],reverse=True
```

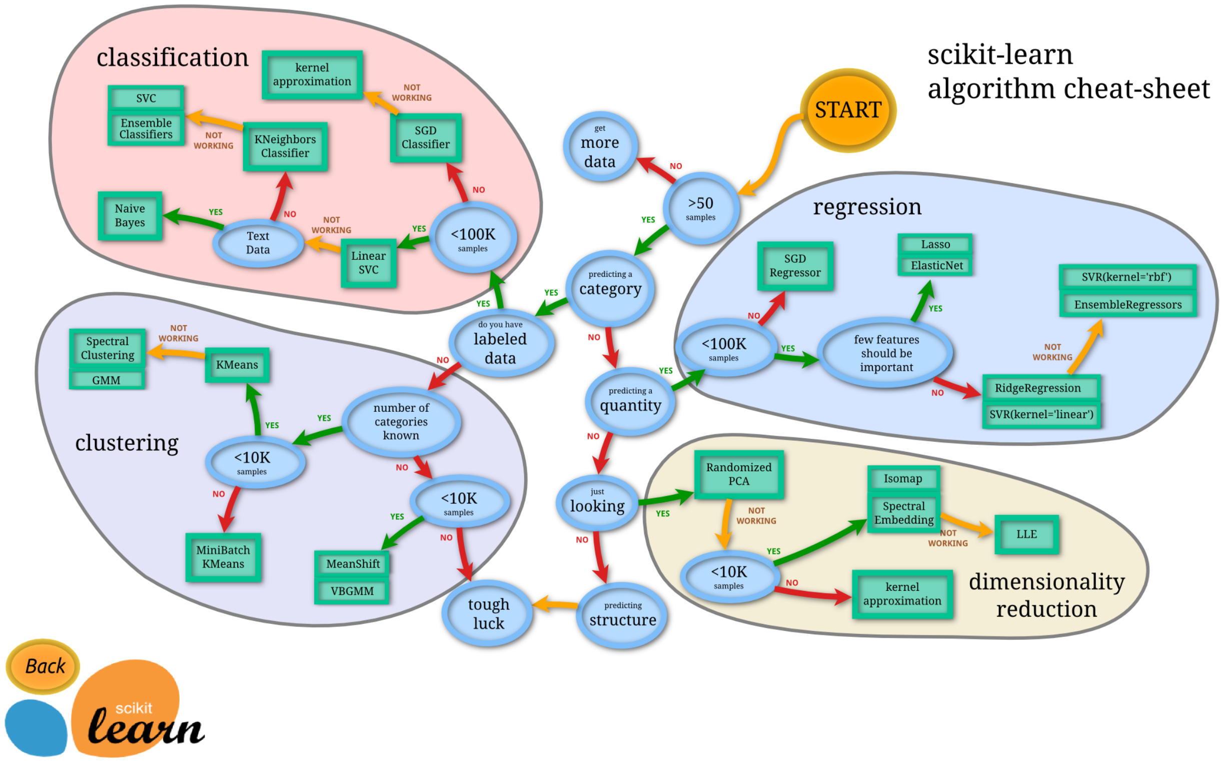
In [36]:

```
#Plot Decision Tree feature importances
plt.figure(figsize=(12,4))
plt.suptitle('Decision Tree Feature Importances',fontsize=14)
# Left plot
plt.subplot(1,2,1)
plt.bar(range(len(important_features1)),important_features1.values(),align='center',color=[
plt.xticks(range(len(important_features1)),important_features1.keys(),rotation='vertical')
plt.title('Unscaled Features')
plt.ylabel('Importance')
plt.ylim(0,0.25)
# Right plot
plt.subplot(1,2,2)
plt.bar(range(len(important_features2)),important_features2.values(),align='center',color=[
plt.xticks(range(len(important_features2)),important_features2.keys(),rotation='vertical')
plt.title('Scaled Features')
plt.ylabel('Importance')
plt.ylim(0,0.25)
plt.show()
```



Algorithms

We will try a variety of supervised learning classifiers according to the following chart:



Performance

The performance of each classifier will be scored using the following metrics:

$$
\begin{aligned}
True\ Positives\ (TP) &= Correctly\ classified\ POI \\
False\ Positives\ (FP) &= Incorrectly\ classified\ POI\ (Type\ I\ error) \\
False\ Negatives\ (FN) &= Incorrectly\ classified\ Non-POI\ (Type\ II\ error) \\
True\ Negatives\ (TN) &= Correctly\ classified\ Non-POI
\end{aligned}
$$

$$
\begin{aligned}
Accuracy &= \frac{(TP+TN)}{(TP+TN+FP+FN)} \\
Precision &= \frac{TP}{(TP+FP)} \\
Recall &= \frac{TP}{(TP+FN)} \\
F_1 &= 2 \cdot \left(\frac{(Precision \cdot Recall)}{(Precision + Recall)} \right) \\
F_2 &= (1+2^2) \cdot \left(\frac{(Precision \cdot Recall)}{(2^2 \cdot Precision + Recall)} \right)
\end{aligned}
$$

Accuracy is the ratio of individuals correctly classified as a POI to the total number of individuals.

Precision is the ratio of individuals correctly classified as a POI to the total number of individuals classified as a POI.

Recall is the ratio of individuals correctly classified as a POI to the total number of individuals that were actually a POI.

The F-scores are the weighted harmonic mean of the precision and recall. For F1, recall and precision are equally important, whereas for F2, recall is weighted more than precision by a factor of 2.

In [37]:

```
### Task 4: Try a varity of classifiers
### Please name your classifier clf for easy export below.
### Note that if you want to do PCA or other multi-stage operations,
### you'll need to use Pipelines. For more info:
### http://scikit-learn.org/stable/modules/pipeline.html
```

In [38]:

```
def plot_metrics(perf_labels,perf_metrics):
    #Plot Performance Metrics
    plt.figure(figsize=(12,4))
    plt.suptitle('Classifier Performance',fontsize=14)
    # Left plot
    plt.subplot(1,2,1)
    plt.bar(range(len(perf_labels[:5])),perf_metrics[:5],align='center',color=['blue'])
    plt.xticks(range(len(perf_labels[:5])),perf_labels[:5],rotation='vertical')
    plt.title('Scores')
    plt.ylabel('Score')
    plt.axhline(0.3,color='k',linestyle='-',linewidth=2,label='Target')
    # Right plot
    plt.subplot(1,2,2)
    plt.bar(range(len(perf_labels[5:])),perf_metrics[5:],align='center',color=['blue'])
    plt.xticks(range(len(perf_labels[5:])),perf_labels[5:],rotation='vertical')
    plt.title('Predictions')
    plt.ylabel('Count')
    plt.show()
    return
```

In [39]:

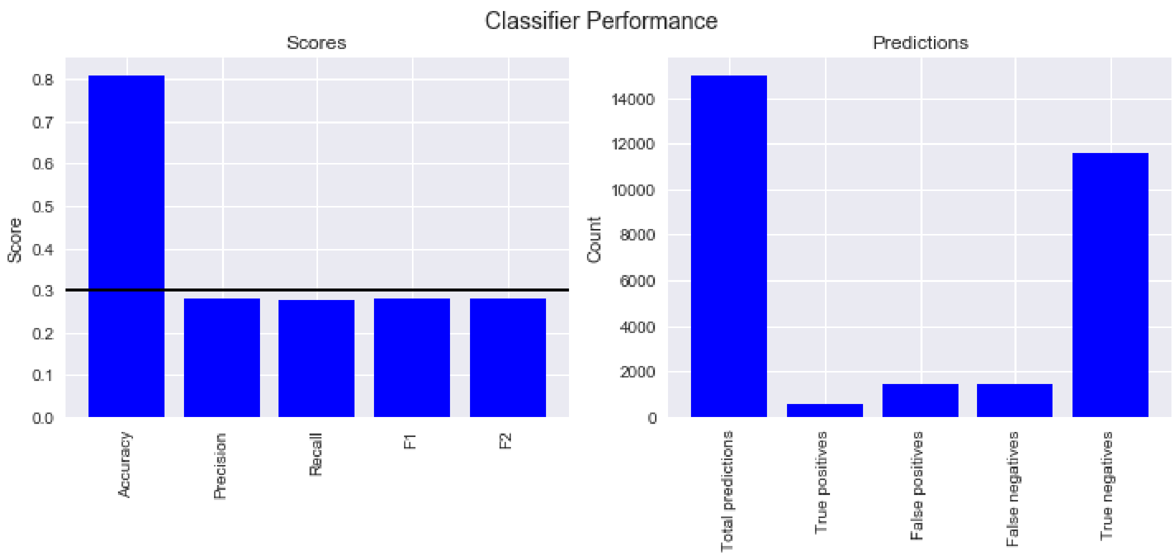
```
def classify(clf):
    perf_labels=('Accuracy',
                 'Precision',
                 'Recall',
                 'F1',
                 'F2',
                 'Total predictions',
                 'True positives',
                 'False positives',
                 'False negatives',
                 'True negatives')
    t0=time()
    s='Classifier: '+str(clf.named_steps.clf)[:str(clf.named_steps.clf).find('(')]
    u='-'*len(s)
    print(s+'\n'+u)
    perf_metrics=test_classifier(clf,my_dataset,extended_features_list)
    print('Elapsed time: %0.3fs' % (time()-t0))
    plot_metrics(perf_labels,perf_metrics)
    return perf_labels,perf_metrics
```

In [40]:

```
#Decision Tree Classifier (Unscaled)
pipeline=Pipeline([('kbest',SelectKBest()),
                   ('clf',DecisionTreeClassifier())])
clf=pipeline.fit(features2,labels2)
perf_labels,perf_metrics=classify(clf)
```

```
Classifier: DecisionTreeClassifier
----------------------------------
Pipeline(memory=None,
     steps=[('kbest', SelectKBest(k=10, score_func=<function f_classif at 0x
077CD030>)), ('clf', DecisionTreeClassifier(class_weight=None, criterion='gi
ni', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'))])
        Accuracy: 0.80967       Precision: 0.28266      Recall: 0.27800 F1:
0.28031 F2: 0.27892
        Total predictions: 15000        True positives:  556    False positi
ves: 1411       False negatives: 1444   True negatives: 11589

Elapsed time: 1.902s
```

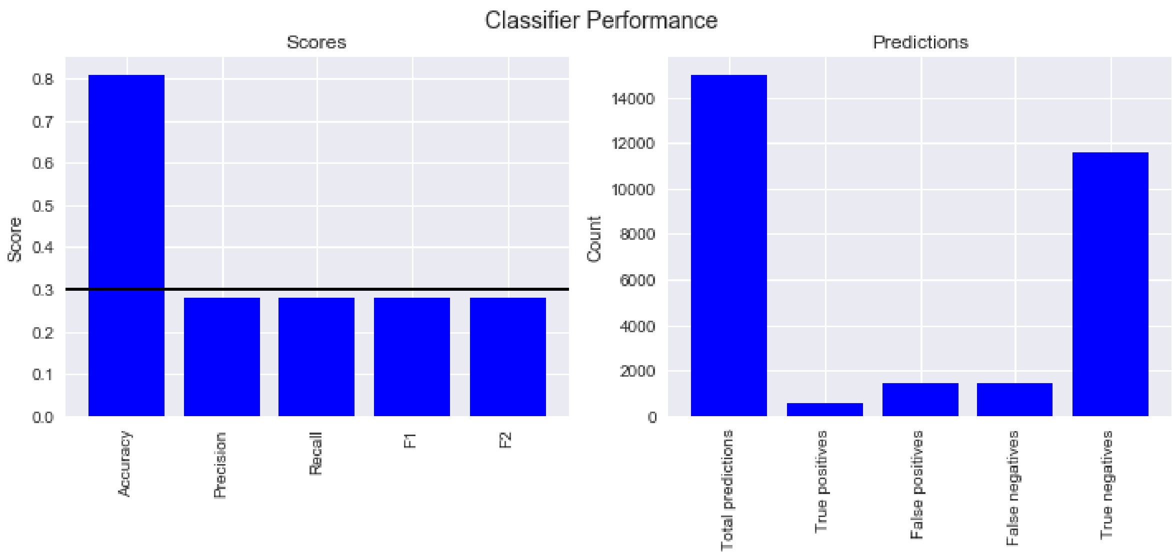


In [41]:

```
#Decision Tree Classifier (Scaled)
pipeline=Pipeline([('scaler',StandardScaler()),
                   ('kbest',SelectKBest()),
                   ('clf',DecisionTreeClassifier())])
clf=pipeline.fit(features2,labels2)
perf_labels,perf_metrics=classify(clf)
```

```
Classifier: DecisionTreeClassifier
----------------------------------
Pipeline(memory=None,
     steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=Tr
ue)), ('kbest', SelectKBest(k=10, score_func=<function f_classif at 0x077CD0
30>)), ('clf', DecisionTreeClassifier(class_weight=None, criterion='gini', m
ax_depth=None,
            max_features=None, max_leaf_nodes=None,
          ...      min_weight_fraction_leaf=0.0, presort=False, random_state
=None,
            splitter='best'))])
        Accuracy: 0.80860       Precision: 0.28149      Recall: 0.28050 F1:
0.28099 F2: 0.28070
        Total predictions: 15000        True positives:  561    False positi
ves: 1432       False negatives: 1439   True negatives: 11568

Elapsed time: 2.190s
```



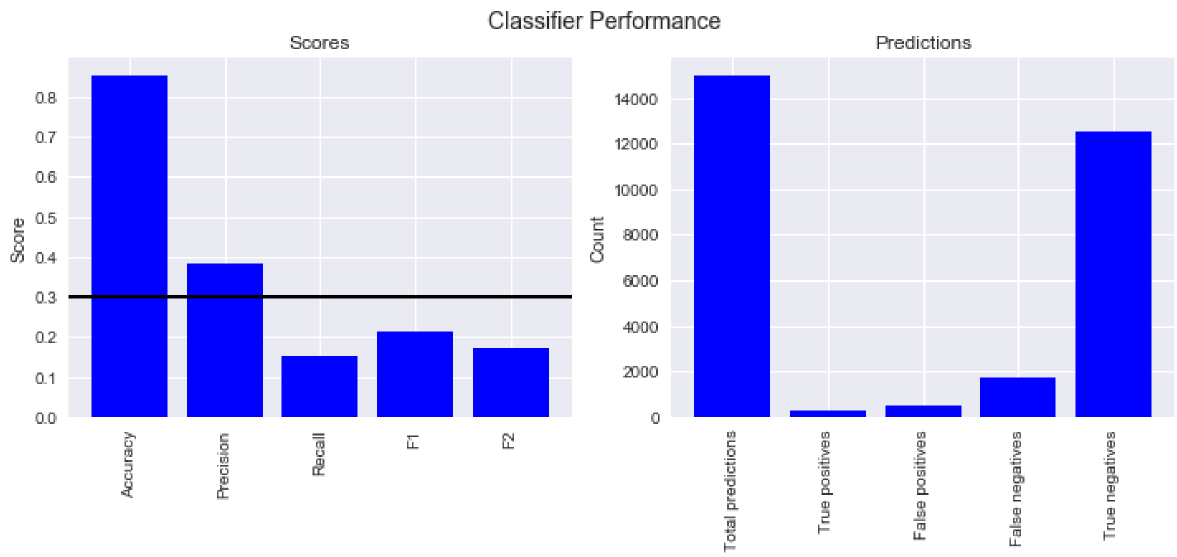
The above two examples confirm that the *Decision Tree Classifier* is not overly affected by feature scaling.

In [42]:

```
#Random Forest Classifier (Unscaled)
pipeline=Pipeline([('kbest',SelectKBest()),
                   ('clf',RandomForestClassifier())])
clf=pipeline.fit(features2,labels2)
perf_labels,perf_metrics=classify(clf)
```

```
Classifier: RandomForestClassifier
----------------------------------
Pipeline(memory=None,
     steps=[('kbest', SelectKBest(k=10, score_func=<function f_classif at 0x
077CD030>)), ('clf', RandomForestClassifier(bootstrap=True, class_weight=Non
e, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None...n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False))])
        Accuracy: 0.85420       Precision: 0.38119      Recall: 0.15000 F1:
0.21529 F2: 0.17071
        Total predictions: 15000        True positives:  300    False positi
ves:  487       False negatives: 1700   True negatives: 12513

Elapsed time: 27.574s
```

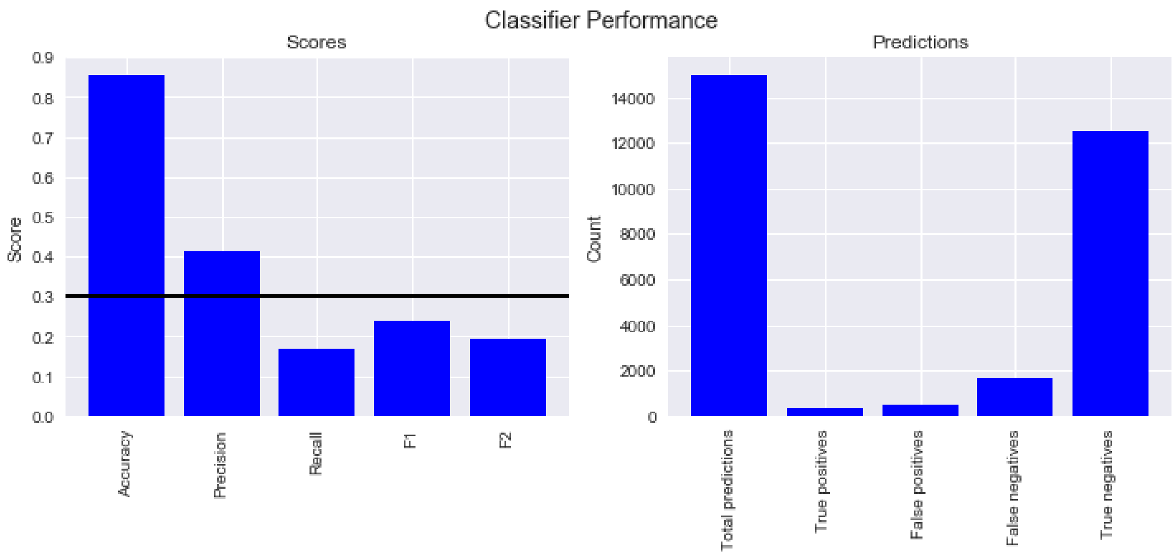


In [43]:

```
#Extra Trees Classifier (Unscaled)
pipeline=Pipeline([('kbest',SelectKBest()),
                   ('clf',ExtraTreesClassifier())])
clf=pipeline.fit(features2,labels2)
perf_labels,perf_metrics=classify(clf)
```

```
Classifier: ExtraTreesClassifier
--------------------------------
Pipeline(memory=None,
     steps=[('kbest', SelectKBest(k=10, score_func=<function f_classif at 0x
077CD030>)), ('clf', ExtraTreesClassifier(bootstrap=False, class_weight=Non
e, criterion='gini',
           max_depth=None, max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=None, verbose=0, warm_start=Fals
e))])
        Accuracy: 0.85687       Precision: 0.41112      Recall: 0.17000 F1:
0.24054 F2: 0.19259
        Total predictions: 15000        True positives:  340    False positi
ves:  487       False negatives: 1660   True negatives: 12513

Elapsed time: 24.586s
```

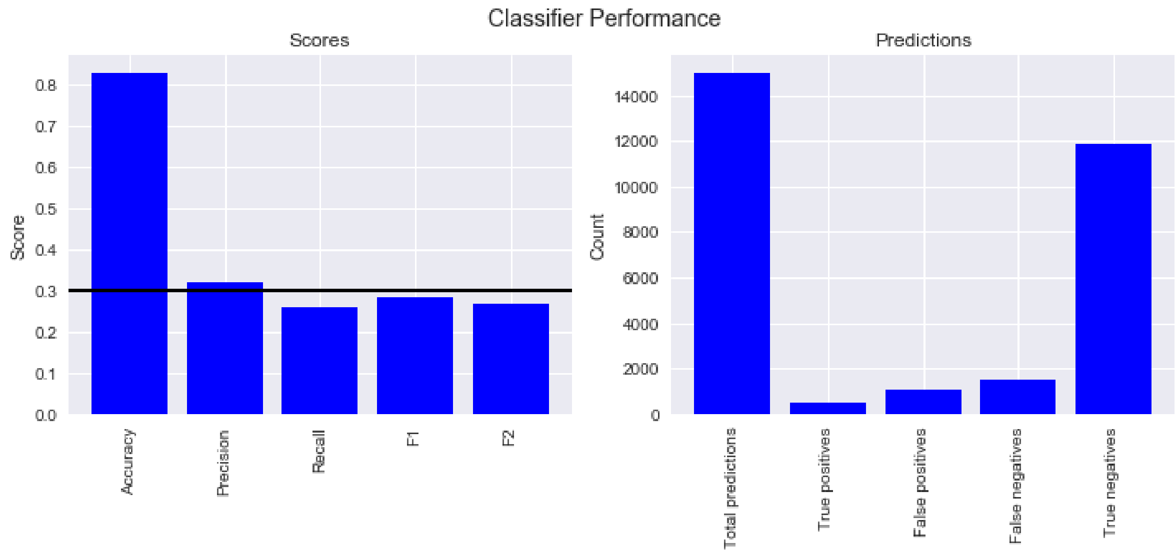


In [44]:

```
#AdaBoost Classifier (Unscaled)
pipeline=Pipeline([('kbest',SelectKBest()),
                   ('clf',AdaBoostClassifier())])
clf=pipeline.fit(features2,labels2)
perf_labels,perf_metrics=classify(clf)
```

```
Classifier: AdaBoostClassifier
------------------------------
Pipeline(memory=None,
     steps=[('kbest', SelectKBest(k=10, score_func=<function f_classif at 0x
077CD030>)), ('clf', AdaBoostClassifier(algorithm='SAMME.R', base_estimator=
None,
          learning_rate=1.0, n_estimators=50, random_state=None))])
        Accuracy: 0.82747       Precision: 0.31829      Recall: 0.25750 F1:
0.28469 F2: 0.26773
        Total predictions: 15000        True positives:  515    False positi
ves: 1103       False negatives: 1485   True negatives: 11897

Elapsed time: 126.968s
```

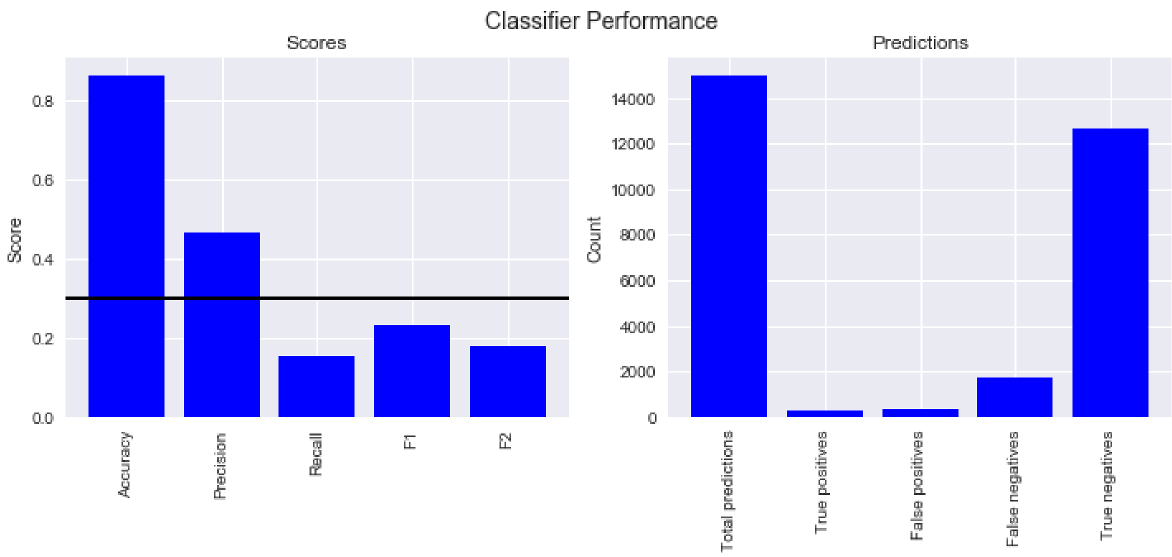


In [45]:

```
#Support Vector Classifier (Scaled)
pipeline=Pipeline([('scaler',StandardScaler()),
                   ('kbest',SelectKBest()),
                   ('clf',SVC(kernel="linear"))])
clf=pipeline.fit(features2,labels2)
perf_labels,perf_metrics=classify(clf)
```

```
Classifier: SVC
---------------
Pipeline(memory=None,
     steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=Tr
ue)), ('kbest', SelectKBest(k=10, score_func=<function f_classif at 0x077CD0
30>)), ('clf', SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])
        Accuracy: 0.86387       Precision: 0.46818      Recall: 0.15450 F1:
0.23233 F2: 0.17841
        Total predictions: 15000        True positives:  309    False positi
ves:  351       False negatives: 1691   True negatives: 12649

Elapsed time: 2.820s
```

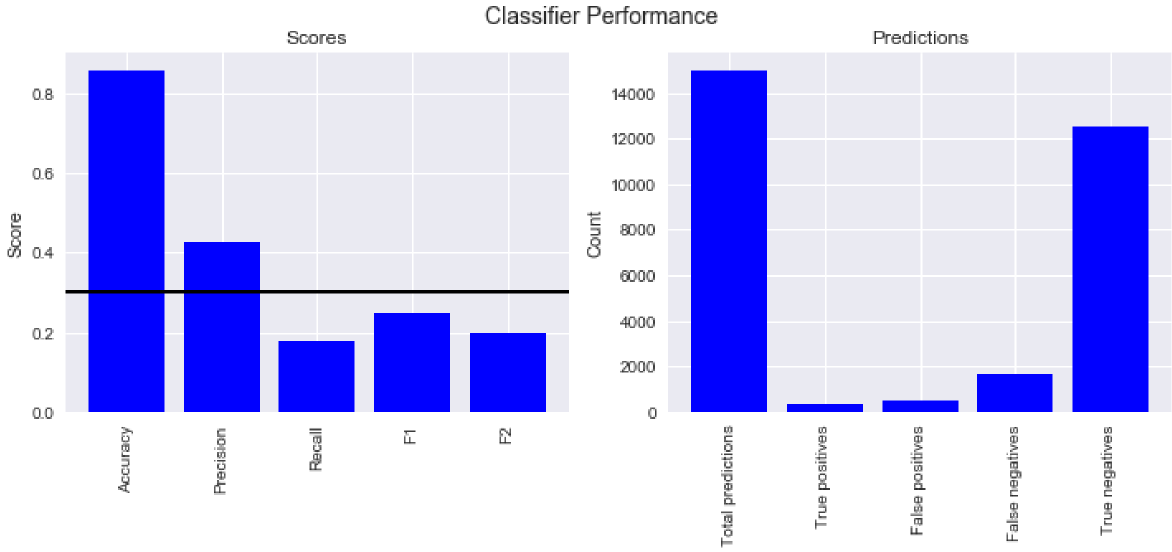


In [46]:

```
#Linear Support Vector Classifier (Scaled)
pipeline=Pipeline([('scaler',StandardScaler()),
                   ('kbest',SelectKBest()),
                   ('clf',LinearSVC())])
clf=pipeline.fit(features2,labels2)
perf_labels,perf_metrics=classify(clf)
```

```
Classifier: LinearSVC
---------------------
Pipeline(memory=None,
     steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=Tr
ue)), ('kbest', SelectKBest(k=10, score_func=<function f_classif at 0x077CD0
30>)), ('clf', LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=
True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0))])
        Accuracy: 0.85833       Precision: 0.42461      Recall: 0.17600 F1:
0.24885 F2: 0.19934
        Total predictions: 15000        True positives:  352    False positi
ves:  477       False negatives: 1648   True negatives: 12523

Elapsed time: 8.064s
```

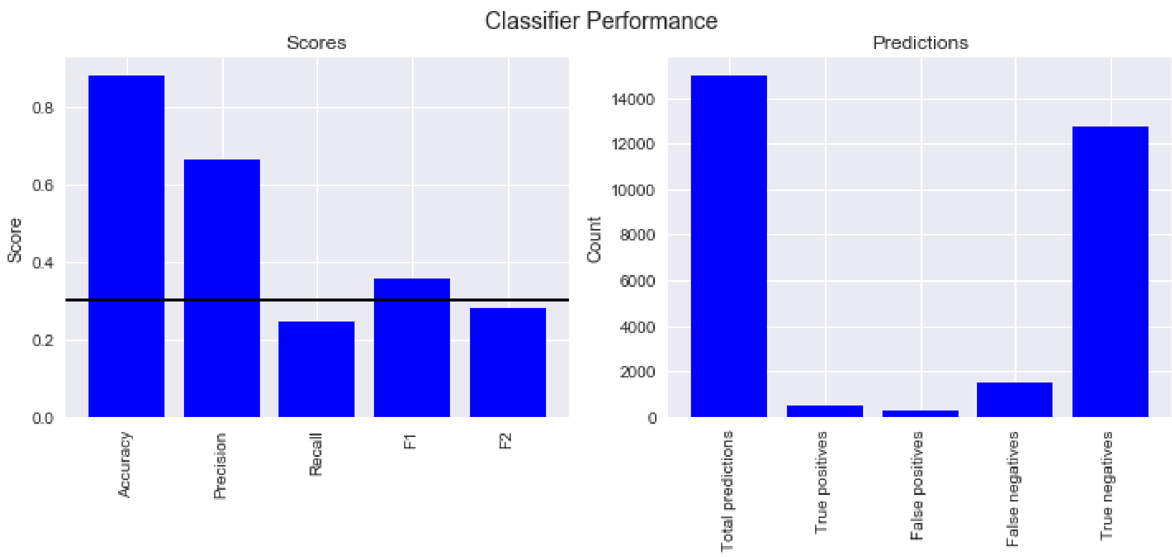


In [47]:

```
#K Neighbors Classifier (Unscaled)
pipeline=Pipeline([('kbest',SelectKBest()),
                   ('clf',KNeighborsClassifier())])
clf=pipeline.fit(features2,labels2)
perf_labels,perf_metrics=classify(clf)
```

```
Classifier: KNeighborsClassifier
--------------------------------
Pipeline(memory=None,
     steps=[('kbest', SelectKBest(k=10, score_func=<function f_classif at 0x
077CD030>)), ('clf', KNeighborsClassifier(algorithm='auto', leaf_size=30, me
tric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'))])
        Accuracy: 0.88287       Precision: 0.66531      Recall: 0.24450 F1:
0.35759 F2: 0.27991
        Total predictions: 15000        True positives:  489    False positi
ves:  246       False negatives: 1511   True negatives: 12754

Elapsed time: 2.269s
```

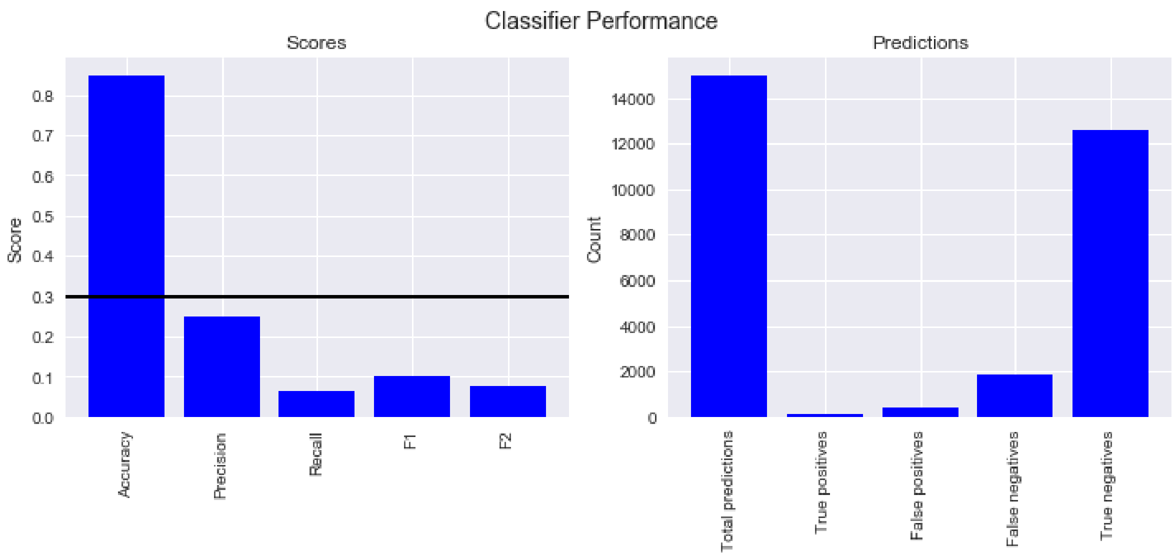


In [48]:

```
#K Neighbors Classifier (Scaled)
pipeline=Pipeline([('scaler',StandardScaler()),
                   ('kbest',SelectKBest()),
                   ('clf',KNeighborsClassifier())])
clf=pipeline.fit(features2,labels2)
perf_labels,perf_metrics=classify(clf)
```

```
Classifier: KNeighborsClassifier
--------------------------------
Pipeline(memory=None,
     steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=Tr
ue)), ('kbest', SelectKBest(k=10, score_func=<function f_classif at 0x077CD0
30>)), ('clf', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric
='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'))])
        Accuracy: 0.84907       Precision: 0.24905      Recall: 0.06550 F1:
0.10372 F2: 0.07682
        Total predictions: 15000        True positives:  131    False positi
ves:  395       False negatives: 1869   True negatives: 12605

Elapsed time: 2.363s
```



The above two examples confirm that the *KNeighbors Classifier* is adversely affected by feature scaling.

Tuning

We will tune the parameters of the best performing classifiers using *GridSearchCV* Cross Validation to try to achieve better than 0.3 for both precision and recall. Since F_1 is the harmonic mean of the precision and recall, we will use this metric to guide our tuning.

In [49]:

```
### Task 5: Tune your classifier to achieve better than .3 precision and recall
### using our testing script. Check the tester.py script in the final project
### folder for details on the evaluation method, especially the test_classifier
### function. Because of the small size of the dataset, the script uses
### stratified shuffle split cross validation. For more info:
### http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedShu
```

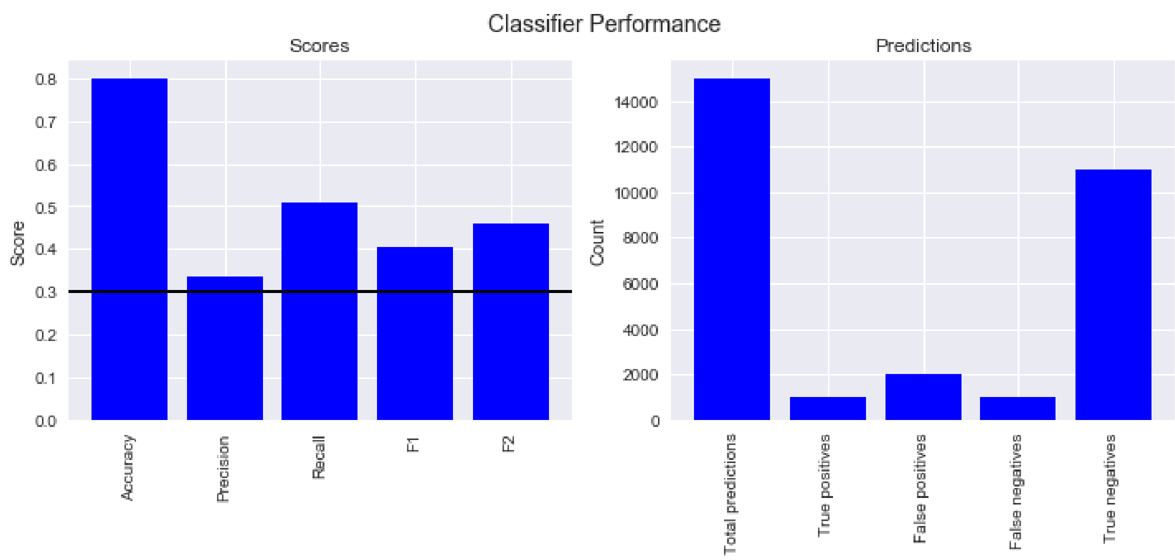
In [50]:

```
#DecisionTree Classifier
pipeline=Pipeline([('scaler',StandardScaler()),
                   ('kbest',SelectKBest()),
                   ('clf',DecisionTreeClassifier())])
param_grid=([{'kbest__k':[6,12,18],
              'clf__max_depth':[None,1,2],
              'clf__min_samples_split':[10,20,30],
              'clf__class_weight':[None,'balanced']}])
clf=GridSearchCV(pipeline,param_grid,scoring='f1').fit(features2,labels2).best_estimator_
perf_labels,perf_metrics=classify(clf)
```

```
C:\ProgramData\Anaconda3\envs\DAND\lib\site-packages\sklearn\metrics\classif
ication.py:1135: UndefinedMetricWarning: F-score is ill-defined and being se
t to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)

Classifier: DecisionTreeClassifier
----------------------------------
Pipeline(memory=None,
     steps=[('scaler', StandardScaler(copy=True, with_mean=True, with_std=Tr
ue)), ('kbest', SelectKBest(k=12, score_func=<function f_classif at 0x077CD0
30>)), ('clf', DecisionTreeClassifier(class_weight='balanced', criterion='gi
ni',
            max_depth=None, max_features=None, max_leaf_nodes=None,
    ...      min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'))])
        Accuracy: 0.80033       Precision: 0.33575      Recall: 0.50850 F1:
0.40445 F2: 0.46106
        Total predictions: 15000        True positives: 1017    False positi
ves: 2012       False negatives:  983   True negatives: 10988

Elapsed time: 3.339s
```

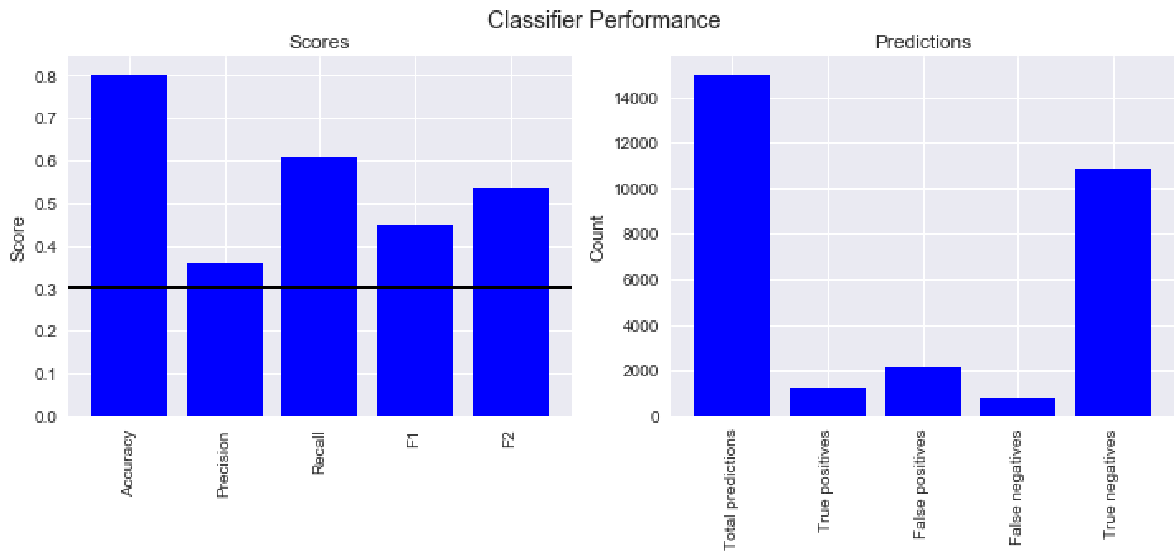


In [51]:

```
#AdaBoost Classifier
pipeline=Pipeline([('kbest',SelectKBest()),
                   ('clf',AdaBoostClassifier())])
param_grid=([{'kbest__k':[6,12,18],
              'clf__base_estimator':[DecisionTreeClassifier(class_weight='balanced',max_dep
                                     DecisionTreeClassifier(class_weight='balanced',max_dep
              'clf__n_estimators':[25,50,75],
              'clf__learning_rate':[0.01,0.1,1.0],
              'clf__algorithm':['SAMME']}])
clf=GridSearchCV(pipeline,param_grid,scoring='f1').fit(features2,labels2).best_estimator_
perf_labels,perf_metrics=classify(clf)
```

```
Classifier: AdaBoostClassifier
------------------------------
Pipeline(memory=None,
     steps=[('kbest', SelectKBest(k=12, score_func=<function f_classif at 0x
077CD030>)), ('clf', AdaBoostClassifier(algorithm='SAMME',
          base_estimator=DecisionTreeClassifier(class_weight='balanced', cri
terion='gini', max_depth=2,
            max_features=None, max_leaf_nodes=None,
            mi...e,
            splitter='best'),
          learning_rate=0.01, n_estimators=50, random_state=None))])
        Accuracy: 0.80280       Precision: 0.35854      Recall: 0.60700 F1:
0.45080 F2: 0.53311
        Total predictions: 15000        True positives: 1214    False positi
ves: 2172       False negatives:  786   True negatives: 10828

Elapsed time: 142.747s
```



In [52]:

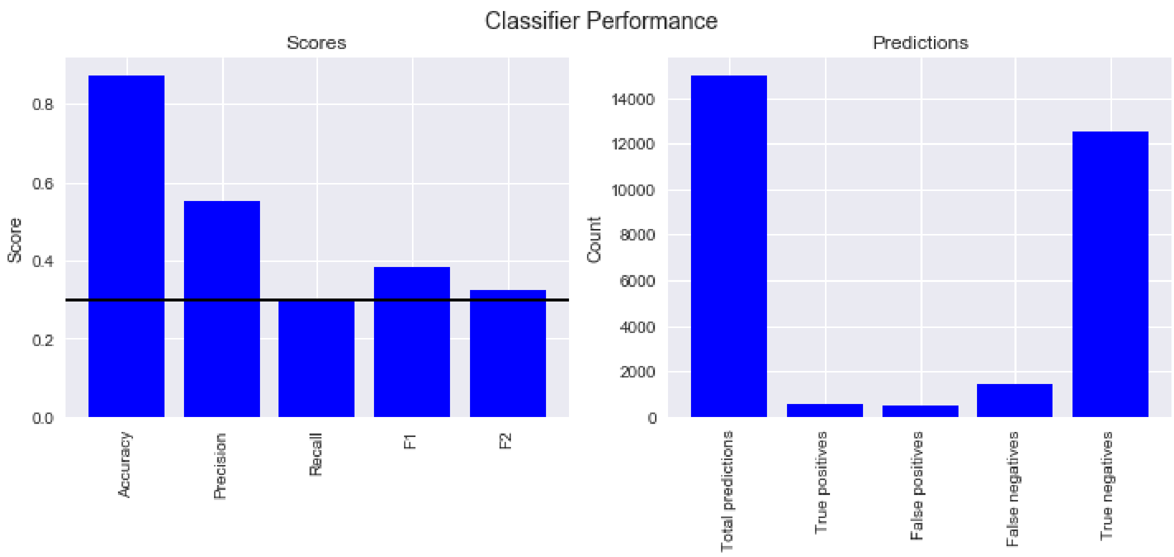
```
#Store this classifiers parameters and scores
CLF=clf
final_kbest=CLF.named_steps.kbest
final_clf=CLF.named_steps.clf
final_perf_labels=perf_labels
final_perf_metrics=perf_metrics
```

In [53]:

```
#K Neighbors Classifier (Unscaled)
pipeline=Pipeline([('kbest',SelectKBest()),
                   ('clf',KNeighborsClassifier())])
param_grid=([{'kbest__k':[6,12,18],
              'clf__n_neighbors':[3,4,5]}])
clf=GridSearchCV(pipeline,param_grid,scoring='f1').fit(features2,labels2).best_estimator_
perf_labels,perf_metrics=classify(clf)
```

```
Classifier: KNeighborsClassifier
--------------------------------
Pipeline(memory=None,
     steps=[('kbest', SelectKBest(k=12, score_func=<function f_classif at 0x
077CD030>)), ('clf', KNeighborsClassifier(algorithm='auto', leaf_size=30, me
tric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=3, p=2,
           weights='uniform'))])
        Accuracy: 0.87433       Precision: 0.55399      Recall: 0.29500 F1:
0.38499 F2: 0.32543
        Total predictions: 15000        True positives:  590    False positi
ves:  475       False negatives: 1410   True negatives: 12525

Elapsed time: 2.168s
```



Results

The best performing algorithm was the *AdaBoost Classifier* (short for Adaptive Boosting) using *SelectKBest* to select features:

In [54]:

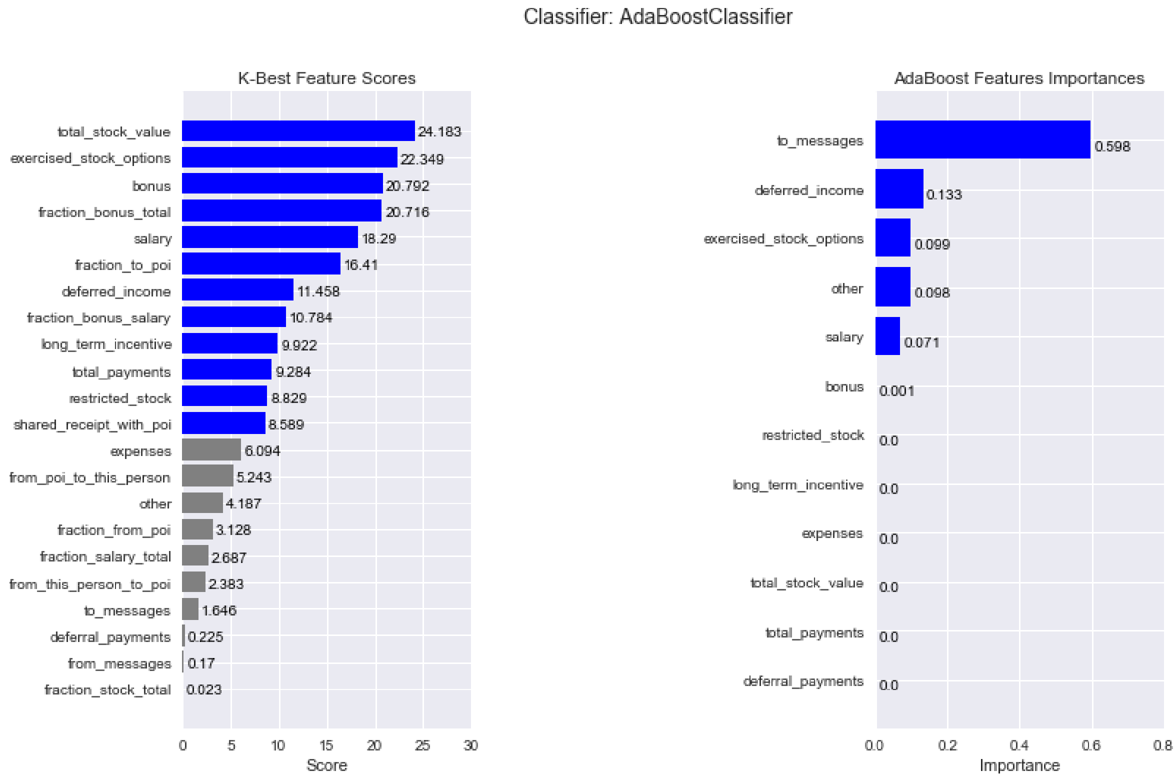
```
#Select K-Best
n=final_kbest.k
k_best=final_kbest
k_best.fit(features2,labels2)
feature_scores=zip(extended_features_list[1:],k_best.scores_)
k_best_features=OrderedDict(sorted(feature_scores,key=lambda x: x[1]))

#AdaBoost Classifier
clf=final_clf
clf=clf.fit(k_best.transform(features2),labels2)

feature_importances=zip(extended_features_list[1:],clf.feature_importances_)
important_features=OrderedDict(sorted(feature_importances,key=lambda x: x[1]))
```

In [55]:

```
#Plot final feature scores and importances
plt.figure(figsize=(12,8))
plt.suptitle('Classifier: '+str(final_clf)[:str(final_clf).find('(')],fontsize=14)
# Left plot
plt.subplot(1,3,1)
l=len(k_best_features)-n
plt.barh(range(len(k_best_features)),k_best_features.values(),align='center',
         color=['grey']*(l)+['blue']*(len(k_best_features)-l))
plt.yticks(range(len(k_best_features)),k_best_features.keys())
for i,v in enumerate(k_best_features.values()):
    plt.text(v+0.3,i-0.2,str(round(v,3)),color='black')
plt.title('K-Best Feature Scores')
plt.xlabel('Score')
plt.xlim(0,30)
# Right plot
plt.subplot(1,3,3)
plt.barh(range(len(important_features)),important_features.values(),align='center',color=['
plt.yticks(range(len(important_features)),important_features.keys())
for i,v in enumerate(important_features.values()):
    plt.text(v+0.01,i-0.2,str(round(v,3)),color='black')
    plt.title('AdaBoost Features Importances')
plt.xlabel('Importance')
plt.xlim(0,0.8)
plt.show()
```



In [56]:

```
#Output Classifier Parameters
print 'Best performing classifier: '+str(final_clf)[:str(final_clf).find('(')]+'\n'
for k in CLF.named_steps.values():
    print k
```

```
Best performing classifier: AdaBoostClassifier

AdaBoostClassifier(algorithm='SAMME',
          base_estimator=DecisionTreeClassifier(class_weight='balanced', cri
terion='gini', max_depth=2,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
          learning_rate=0.01, n_estimators=50, random_state=None)
SelectKBest(k=12, score_func=<function f_classif at 0x077CD030>)
```

In [57]:

```
#Output Performance Metrics
print '\nPerformance Metrics:\n'
print '{:25}{:15}'.format('Metric:','Score:')
for k in range(5):
    print('{:25}{:8.2%}'.format(final_perf_labels[k],final_perf_metrics[k]))
for k in range(5,10):
    print('{:25}{:8d}'.format(final_perf_labels[k],final_perf_metrics[k]))
```

```

Performance Metrics:

Metric:                  Score:
Accuracy                   80.28%
Precision                  35.85%
Recall                     60.70%
F1                         45.08%
F2                         53.31%
Total predictions           15000
True positives               1214
False positives              2172
False negatives               786
True negatives              10828
```

In [58]:

```
#Output Feature Scores and Feature Importances
print '\nFeature Scores:\n'
print('{:30}{:15}{:15}'.format('Feature:','Score:','Importance:'))
for k in extended_features_list[1:]:
    print('{:29}{:8.4f}{:14.4f}'.format(k,k_best_features[k],(important_features[k] if (k i
```

```
Feature Scores:

Feature:                      Score:         Importance:
salary                        18.2897        0.0711
deferral_payments              0.2246        0.0000
total_payments                 9.2839        0.0000
bonus                         20.7923        0.0014
deferred_income               11.4585        0.1326
total_stock_value             24.1829        0.0000
expenses                       6.0942        0.0000
exercised_stock_options       22.3488        0.0987
other                          4.1875        0.0982
long_term_incentive            9.9222        0.0000
restricted_stock               8.8287        0.0000
to_messages                    1.6463        0.5979
from_poi_to_this_person        5.2434        0.0000
from_messages                  0.1697        0.0000
from_this_person_to_poi        2.3826        0.0000
shared_receipt_with_poi        8.5894        0.0000
fraction_bonus_salary         10.7836        0.0000
fraction_bonus_total          20.7156        0.0000
fraction_salary_total          2.6874        0.0000
fraction_stock_total           0.0225        0.0000
fraction_to_poi               16.4097        0.0000
fraction_from_poi              3.1281        0.0000
```

In [59]:

```
### Task 6: Dump your classifier, dataset, and features_list so anyone can
### check your results. You do not need to change anything below, but make sure
### that the version of poi_id.py that you submit can be run on its own and
### generates the necessary .pkl files for validating your results.

#dump_classifier_and_data(CLF,my_dataset,extended_features_list)
```

Conclusion

The goal of this project was to build a machine learing classifier to identify Enron employees who may have committed fraud based on the public Enron financial and email dataset. Such employees are referred to as "person's of interest", or, POIs. The application of machine learning is extremely useful in problems like this as it is able to work with relatively high dimensional data and find any relationships that may exist. These trained models can then be used to make predictions about the data.

The original dataset consisted of records of 146 individuals, 18 of whom were labelled a 'person of interest'. Each record contained up to 21 items of data (1 POI label, 14 finance features and 6 e-mail features). Initial analysis revealed outliers and unnecessary records that were removed:

- *TOTAL* - Spreadsheet aggregation included by mistake (outlier)

- *LOCKHART* EUGENE E - Did not contain any numerical data
- *THE TRAVEL AGENCY IN THE PARK* - Not an individual (Alliance Worldwide - co-owned by the sister of Enron's former Chairman)

Of all the features in the dataset, the following were removed:

- *email_address* - not numerical data
- *loan_advances*- less than 10% in dataset
- *restricted_stock_deferred* - less than 10% in dataset for POI
- *director_fees* - less than 10% in dataset for POI

The supplied *enron61702insiderpay.pdf* file was converted to csv format, and was edited so column names and employee names coincided to the those in the dataset. Comparison of this file with the dataset identified 2 entries that were mis-aligned, which were subsequently corrected.

In addition to the supplied features, several new features were engineered to provide further insights into the data. The proportion of the employees total compensation which comes from their bonus, stocks or salary, or how frequently the employee communicates with a person of interest could potentially be quite informative. Thus, the following additional features were engineered:

- The ratio of the employees bonus to their salary
- The ratio of the employees bonus to their total compensation
- The ratio of the employees salary to their total compensation
- The ratio of the employees stocks to their total compensation
- The ratio of the employees emails to a person of interest, to the total number of emails sent
- The ratio of the employees emails from a person of interest, to the total number of emails received

Automatic feature selection was employed to improve accuracy and reduce the chances of overfitting. The final feature set used was determined using the *SelectKBest* algorithm.

Some classifiers are known to behave badly if the features are not more or less normally distributed, and require the features to be scaled. Since the data was highly skewed, *StandardScaler* was employed where appropriate.

Several supervised machine learning classifiers were tested with their default values and the relevant performance metrics calculated, with the multistage operations (Feature Scaling, Feature Selection and Classifying) processed using pipelines:

- *DecisionTreeClassifier*
- *RandomForestClassifier*
- *ExtraTreesClassifier*
- *AdaBoostClassifier*
- *SVC*
- *LinearSVC*
- *KNeighborsClassifier*

The 3 best performing classifiers were tuned using *GridSearchCV*:

- *DecisionTreeClassifier*
- *AdaBoostClassifier*
- *KNeighborsClassifier*

Parameter tuning is the process of re-running the classifier with different specified combinations of parameters in order to maximise a scoring metric. In this case the *f1* score was set as the target as it is a combination of both *precision* and *recall*. Incorrect application of parameter tuning can lead to underfitting or overfitting which may result in a poorly performing classifier. The use of pipelines enabled the tuning of the number of selected features, as well as several influential classifier parameters.

Validation ensures that the classifier performs consistently across various datasets. Validation is carried out by repeatedly splitting the data into several training and testing sets. This maximises the amount of training and testing data available. *Stratified Shuffle Split Cross Validation* was employed here since the dataset is small and sparse.

The best performing algorithm was found to be the *AdaBoost Classifier* using *SelectKBest* to select features. The classifier achieved the following performance:

- Accuracy = 80.29%
- Precision = 35.87%
- Recall = 60.70%.

These results can be interpreted as follows:

- 80.29% of all the individuals were correctly classified as a person of interest.
- Of all the individuals classified as a person of interest, 35.87% of these classifications were correct.
- Of all the actual persons of interest, 60.70% of these were correctly classified.

References

https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
https://scikit-learn.org/stable/modules/cross_validation.html
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html
https://en.wikipedia.org/wiki/Precision_and_recall