In [1]:

```
import sys
import pickle
import numpy
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA
from sklearn import cross_validation
from sklearn import tree
from sklearn.metrics import accuracy_score
sys.path.append("../tools/")
from time import time
from sklearn.model_selection import GridSearchCV
from feature_format import featureFormat, targetFeatureSplit
from tester import dump_classifier_and_data,test_classifier
from sklearn.cross_validation import train_test_split
from sklearn.cross_validation import StratifiedShuffleSplit
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
```

```
C:\Users\xyang\Anaconda2\lib\site-packages\sklearn\cross_validation.py:44: D
eprecationWarning: This module was deprecated in version 0.18 in favor of th
e model_selection module into which all the refactored classes and functions
are moved. Also note that the interface of the new CV iterators are differen
t from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

Task 1: Select what features you'll use.

In [2]:

```
### features_list is a list of strings, each of which is a feature name.
### Load the dictionary containing the dataset
with open("final_project_dataset.pkl", "r") as data_file:
    data_dict = pickle.load(data_file)
```

In [3]:

```
Name = [x for x in data_dict]
```

In [4]:

```
for i in Name:
    print i
```

```
METTS MARK
BAXTER JOHN C
ELLIOTT STEVEN
CORDES WILLIAM R
HANNON KEVIN P
MORDAUNT KRISTINA M
MEYER ROCKFORD G
MCMAHON JEFFREY
HORTON STANLEY C
PIPER GREGORY F
HUMPHREY GENE E
UMANOFF ADAM S
BLACHMAN JEREMY M
SUNDE MARTIN
GIBBS DANA R
LOWRY CHARLES P
COLWELL WESLEY
MULLER MARK S
JACKSON CHARLENE R
```

In [5]:

```
feature = [x for x in data_dict[Name[0]]]
```

In [6]:

```
feature
```

Out[6]:

```
['salary',
 'to_messages',
 'deferral_payments',
 'total_payments',
 'exercised_stock_options',
 'bonus',
 'restricted_stock',
 'shared_receipt_with_poi',
 'restricted_stock_deferred',
 'total_stock_value',
 'expenses',
 'loan_advances',
 'from_messages',
 'other',
 'from_this_person_to_poi',
 'poi',
 'director_fees',
 'deferred_income',
```

In [7]:

```
feature = [x for x in feature if x!= 'poi' and x!='email_address'and x!='from_this_person_
features_list = ['poi']+feature
```

In [8]:

```
print "Numbers of features are",len(features_list)
```

```
Numbers of features are 15
```

In [9]:

```
print "Numbers of data points are",len(data_dict)
```

```
Numbers of data points are 146
```

In [10]:

```
print "Percentage of POI in the dataset", float(len([x for x in data_dict if data_dict[x][
```

```
Percentage of POI in the dataset 0.123287671233
```

In [11]:

```
d = {}
for i in feature:
    d[i] = 0
    for ii in Name:
        if data_dict[ii][i] == 'NaN':
            d[i] += 1
```

In [12]:

```
for i in d:
    print "Feature",i,"contains",float(d[i])/len(Name), "Percent Missing values"
```

```
Feature salary contains 0.349315068493 Percent Missing values
Feature to_messages contains 0.41095890411 Percent Missing values
Feature deferral_payments contains 0.732876712329 Percent Missing values
Feature long_term_incentive contains 0.547945205479 Percent Missing values
Feature loan_advances contains 0.972602739726 Percent Missing values
Feature bonus contains 0.438356164384 Percent Missing values
Feature restricted_stock_deferred contains 0.876712328767 Percent Missing va
lues
Feature expenses contains 0.349315068493 Percent Missing values
Feature exercised_stock_options contains 0.301369863014 Percent Missing valu
es
Feature from_messages contains 0.41095890411 Percent Missing values
Feature other contains 0.36301369863 Percent Missing values
Feature deferred_income contains 0.664383561644 Percent Missing values
Feature restricted_stock contains 0.246575342466 Percent Missing values
Feature director_fees contains 0.883561643836 Percent Missing values
```

In [13]:

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

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

In [14]:

```
features_train, features_test, labels_train, labels_test = cross_validation.train_test_spl
```

In [15]:

```
ure_list():
cisionTreeClassifier()
(features_train,labels_train)
edict(features_test)
{}
core(pred, labels_test) > 0.8:
ures = sorted(range(len(clf.feature_importances_)), key=lambda i: clf.feature_importances_[
ures = [x for x in sort_features if clf.feature_importances_[x]>0.1]
sort_features:
tance[x] = clf.feature_importances_[x]
rt_features,importance

,[]
```

In [16]:

```
def vari_class_test(clf):
    ### 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.Stratifi

    ### 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.

    test_classifier(clf, my_dataset, features_list_2)
```

In [17]:

```
feature_list = []
importance_dict = {}

for i in xrange(10000):
    sort_features,importance = generate_feature_list()
    feature_list.append(sort_features)
    for ii in importance:
        try:
            if importance_dict[ii] < importance[ii]:
                importance_dict[ii] = importance[ii]
        except:
            importance_dict[ii] = importance[ii]

feature_list = set.union(*[set(list) for list in feature_list])
```

In [18]:

```
feature_list_2 = {}
for idx in feature_list:
    feature_list_2[importance_dict[idx]] = feature[idx]
    print "The feature",feature[idx],"has max importance",importance_dict[idx]
```

```
The feature salary has max importance 0.207046151794
The feature deferral_payments has max importance 0.141787624141
The feature exercised_stock_options has max importance 0.268973074462
The feature bonus has max importance 0.251040098579
The feature expenses has max importance 0.210280242128
The feature from_messages has max importance 0.176311433664
The feature other has max importance 0.320334162065
The feature deferred_income has max importance 0.138786827862
```

let's try how many feature used would optimize the performace

In [19]:

```
f_list_2 = []
for i in feature_list_2:
    f_list_2.append(feature_list_2[i])
for i in xrange(len(f_list_2)):
    clf = tree.DecisionTreeClassifier()
    features_list_2 = ['poi']+f_list_2[i:]
    print "use",len(f_list_2)-i,"features"
    vari_class_test(clf)
```

```
use 7 features
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=No
ne,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
        Accuracy: 0.81140       Precision: 0.30082      Recall: 0.31300
F1: 0.30679     F2: 0.31049
        Total predictions: 15000        True positives:  626    False po
sitives: 1455   False negatives: 1374   True negatives: 11545

use 6 features
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=No
ne,
            max_features=None, max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best')
```

We can find use 6 features would get better result Accuracy: 0.81360 Precision: 0.31297 Recall: 0.33300 F1: 0.32267 F2: 0.32879

In [20]:

```
features_list_2 = ['poi']+f_list_2
```

In [21]:

```
features_list_2
```

Out[21]:

```
['poi',
 'from_messages',
 'salary',
 'other',
 'deferred_income',
 'deferral_payments',
 'expenses',
 'exercised_stock_options',
 'bonus']
```

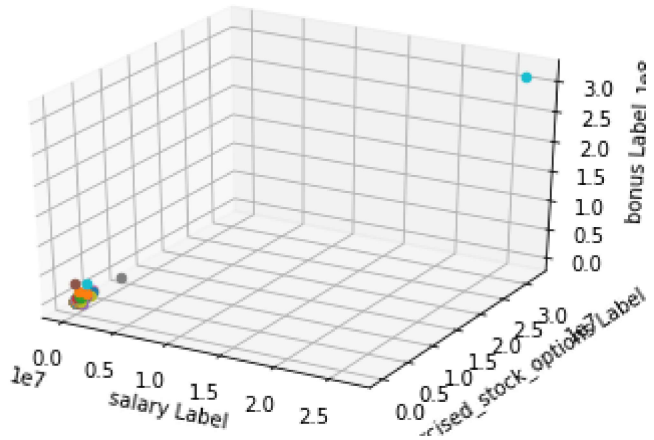
Task 2: Remove outliers

In [21]:

```
### Find outliers from plot
def plot3d(data):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    for point in data:
        salary = point[1]
        exercised_stock_options = point[3]
        bonus = point[4]
        ax.scatter(salary, exercised_stock_options,bonus)
    ax.set_xlabel('salary Label')
    ax.set_ylabel('exercised_stock_options Label')
    ax.set_zlabel('bonus Label')
    plt.show()
```

In [22]:

```
plot3d(data)
```



In [23]:

```
for i in data_dict:
    if data_dict[i]['salary'] > 2e+07 and data_dict[i]['salary']!="NaN":
        print i
```

TOTAL

In [24]:

```
### plot after pop out Total
data_dict.pop('TOTAL')
```

Out[24]:

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

We would also exclude those has almost all features 'NaN' point

In [25]:

```
d = {}
for i in data_dict:
    d[i] = 0
    for ii in feature:
        if data_dict[i][ii] == 'NaN':
            d[i] += 1
miss_value =  [x for x in d if d[x] > 10]
print miss_value
```

```
['NOLES JAMES L', 'LOWRY CHARLES P', 'WALTERS GARETH W', 'WODRASKA JOHN', 'U
RQUHART JOHN A', 'WHALEY DAVID A', 'MENDELSOHN JOHN', 'CLINE KENNETH W', 'LE
WIS RICHARD', 'WAKEHAM JOHN', 'DUNCAN JOHN H', 'LEMAISTRE CHARLES', 'PIRO JI
M', 'WROBEL BRUCE', 'MEYER JEROME J', 'BADUM JAMES P', 'GATHMANN WILLIAM D',
'GILLIS JOHN', 'LOCKHART EUGENE E', 'PEREIRA PAULO V. FERRAZ', 'BLAKE JR. NO
RMAN P', 'PRENTICE JAMES', 'THE TRAVEL AGENCY IN THE PARK', 'CHRISTODOULOU D
IOMEDES', 'WINOKUR JR. HERBERT S', 'BROWN MICHAEL', 'YEAP SOON', 'HAYSLETT R
ODERICK J', 'FUGH JOHN L', 'SCRIMSHAW MATTHEW', 'SAVAGE FRANK', 'GRAMM WENDY
L']
```

In [26]:

```
for i in miss_value:
    data_dict.pop(i)
```

If we want to analyze based on features list, we should not include one has all these as "NaN"

In [27]:

```
pop_list = []
for i in data_dict:
    count = 0
    for feature in features_list_2:
        if data_dict[i][feature]== "NaN":
            count += 1
    if count > len(features_list_2)-2:
        pop_list.append(i)
print pop_list
```

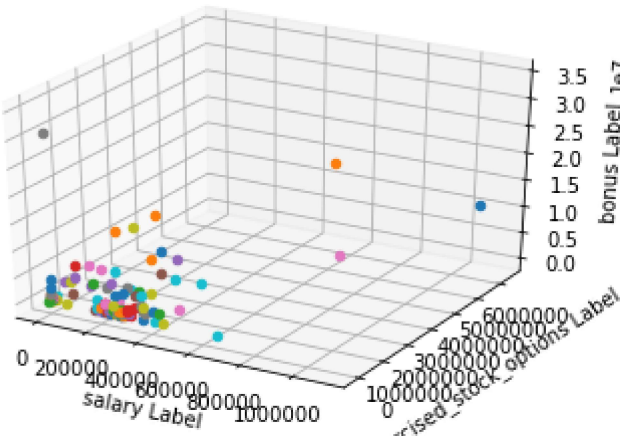
```
[]
```

In [28]:

```
### plot after pop out Total
for x in pop_list:
    data_dict.pop(x)
```

In [29]:

```
data = featureFormat(data_dict, features_list)
plot3d(data)
```



Task 3: Create new feature(s) using TF-IDF

In [30]:

```
import os
import pickle
import re
import sys
from nltk.stem.snowball import SnowballStemmer
import string
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
```

In [31]:

```
a = os.listdir("./emails_by_address")
print len(a)
```

```
4662
```

get the email from POI

In [32]:

```
with open("final_project_dataset.pkl", "r") as data_file:
    data_dict = pickle.load(data_file)
Name = [x for x in data_dict]
feature = [x for x in data_dict[Name[0]]]
first_name = []
last_name = []
for i in Name:
    if data_dict[i]['poi'] == True:
        for idx,ii in enumerate(i.split(' ')):
            if idx == 0 :
                last_name.append(ii)
            else:
                if len(ii) >= 3:
                    first_name.append(ii)
print "POI's first name and last name:"
```

```
POI's first name and last name:
```

In [33]:

```
d = {}
remove_list = {}
for i in a:
    stats = i.split('_')[0]
    name = i.split('_')[1].split('@')[0].strip('0123456789').replace('enron','').upper()
    f_name = ''
    l_name = ''

    for idx, ii in enumerate(name.split('.')):
        if idx == 0:
            f_name = ii
        else:
            if len(ii) >= 3:
                l_name = ii
        if f_name != '' or l_name != '':
            remove_list[f_name] = 1
            remove_list[l_name] = 1
        if f_name != '' and l_name != '':

            for iii in xrange(len(first_name)):
                if f_name in first_name[iii] and l_name in last_name[iii]:
                    d[(first_name[iii],last_name[iii],stats)] = i
print "POI's first, last name and email txt:"
for i in d:
    print i,d[i]
from_POI = []
to_POI = []
for i in d:
    if "from" in i:
        from_POI.append(d[i])
    if "to" in i:
        to_POI.append(d[i])
```

```
POI's first, last name and email txt:
('TIMOTHY', 'BELDEN', 'from') from_tim.belden@enron.com.txt
('DAVID', 'DELAINEY', 'from') from_david.delainey@enron.com.txt
('RAYMOND', 'BOWEN', 'to') to_raymond.bowen@enron.com.txt
('DAVID', 'DELAINEY', 'to') to_david.delainey@enron.com.txt
('JEFFREY', 'SKILLING', 'to') to_jeff.skilling@enron.com.txt
('KEVIN', 'HANNON', 'to') to_kevin.hannon@enron.com.txt
('KENNETH', 'RICE', 'to') to_ken.rice@enron.com.txt
('TIMOTHY', 'BELDEN', 'to') to_tim.belden@enron.com.txt
('KENNETH', 'LAY', 'from') from_kenneth.lay@enron.com.txt
('CHRISTOPHER', 'CALGER', 'from') from_christopher.calger@enron.com.txt
('REX', 'SHELBY', 'to') to_rex.shelby@enron.com.txt
('PAULA', 'RIEKER', 'to') to_paula.rieker@enron.com.txt
('KENNETH', 'LAY', 'to') to_kenneth.lay@enron.com.txt
('KENNETH', 'RICE', 'from') from_ken.rice@enron.com.txt
('MARK', 'KOENIG', 'to') to_mark.koenig@enron.com.txt
('RICHARD', 'CAUSEY', 'from') from_richard.causey@enron.com.txt
('RICHARD', 'CAUSEY', 'to') to_richard.causey@enron.com.txt
('REX', 'SHELBY', 'from') from_rex.shelby@enron.com.txt
('RAYMOND', 'BOWEN', 'from') from_raymond.bowen@enron.com.txt
('JEFFREY', 'SKILLING', 'from') from_jeff.skilling@enron.com.txt
('WESLEY', 'COLWELL', 'from') from_wes.colwell@enron.com.txt
('PAULA', 'RIEKER', 'from') from_paula.rieker@enron.com.txt
('WESLEY', 'COLWELL', 'to') to_wes.colwell@enron.com.txt
```

```
('BEN', 'GLISAN', 'from') from_ben.glisan@enron.com.txt
('BEN', 'GLISAN', 'to') to_ben.glisan@enron.com.txt
('KEVIN', 'HANNON', 'from') from_kevin.hannon@enron.com.txt
('MARK', 'KOENIG', 'from') from_mark.koenig@enron.com.txt
('CHRISTOPHER', 'CALGER', 'to') to_christopher.calger@enron.com.txt
```

get the email from non-POI

In [34]:

```
first_name = []
last_name = []

for i in Name:
    if data_dict[i]['poi'] == False:
        for idx,ii in enumerate(i.split(' ')):
            if idx == 0 :
                last_name.append(ii)
            else:
                if len(ii) >= 3:
                    first_name.append(ii)
                else:
                    first_name.append('')


fir_name = []
las_name = []

for i in xrange(len(last_name)):
    if first_name[i]!="" and last_name[i]!="":
        fir_name.append(first_name[i])
        las_name.append(last_name[i])

d = {}

for i in a:
    stats = i.split('_')[0]
    name = i.split('_')[1].split('@')[0].strip('0123456789').replace('enron','').upper()
    f_name = ''
    l_name = ''

    for idx, ii in enumerate(name.split('.')):
        if idx == 0:
            f_name = ii
        else:
            if len(ii) >= 3:
                l_name = ii

                for iii in xrange(len(fir_name)):
                    if las_name[iii].startswith(l_name) and fir_name[iii].startswith(f_nam
                        d[(fir_name[iii],las_name[iii],stats)] = i
print "None-POI's first, last name and email txt:"
for i in d:
    print i,d[i]
No_from_POI = []
No_to_POI = []
for i in d:
    if "from" in i:
        No_from_POI.append(d[i])
    if "to" in i:
        No_to_POI.append(d[i])
```

```
None-POI's first, last name and email txt:
('MARK', 'METTS', 'from') from_mark.metts@enron.com.txt
('JAMES', 'KAMINSKI', 'to') to_j.kaminski@enron.com.txt
('MARK', 'METTS', 'to') to_mark.metts@enron.com.txt
('JAMES', 'KAMINSKI', 'from') from_j.kaminski@enron.com.txt
```

get the corpus

In [35]:

```
def parseOutText(f):
    stemmm = SnowballStemmer("english")
    f.seek(0)  ### go back to beginning of file (annoying)
    all_text = f.read()

    ### split off metadata
    content = all_text.split("X-FileName:")
    words = ""
    if len(content) > 1:
        ### remove punctuation
        text_string = content[1].translate(string.maketrans("", ""), string.punctuation)
        ### split the text string into individual words, stem each word,
        ### and append the stemmed word to words (make sure there's a single
        ### space between each stemmed word)
        for i in text_string.split():
                i_stem = stemmm.stem(i)
                words += ' ' + stemmm.stem(i)

    return words
```

In [36]:

```
from_data = []
word_data = []
a = {}
for i in from_POI:
    a[i] = "POI"
for i in No_from_POI:
    a[i] = "NoPOI"

for i in a:
    from_POI_address = open("./emails_by_address/"+i,"r")
    for path in from_POI_address:
        try:
            path = path.replace("enron_mail_20110402/","")
            path = os.path.join(os.pardir, path[:-1])
            ### n represent how many levels to climb
            email = open(path, "r")
            ### use parseOutText to extract the text from the opened email
            text = parseOutText(email)
            ### use str.replace() to remove any instances of the words
            for item in remove_list:
                 text = text.replace(item, "")
            ### append the text to word_data
            word_data.append(text)
            ### append a 1 to from_data if email is from POI, and 0 if email is not from P
            if a[i] == "POI":
                from_data.append(0)
            if a[i] == "NoPOI":
                from_data.append(1)

        except:
            skip = True
        email.close()
    print i,"process finished"
    from_POI_address.close()

pickle.dump( word_data, open("your_word_data.pkl", "w") )
pickle.dump( from_data, open("your_email_authors.pkl", "w"))
```

```
from_kevin.hannon@enron.com.txt process finished
from_mark.metts@enron.com.txt process finished
from_richard.causey@enron.com.txt process finished
from_j.kaminski@enron.com.txt process finished
from_paula.rieker@enron.com.txt process finished
from_ken.rice@enron.com.txt process finished
from_raymond.bowen@enron.com.txt process finished
from_jeff.skilling@enron.com.txt process finished
from_david.delainey@enron.com.txt process finished
from_tim.belden@enron.com.txt process finished
from_wes.colwell@enron.com.txt process finished
from_mark.koenig@enron.com.txt process finished
from_christopher.calger@enron.com.txt process finished
from_kenneth.lay@enron.com.txt process finished
from_ben.glisan@enron.com.txt process finished
from_rex.shelby@enron.com.txt process finished
```

In [37]:

```
tfidf_vectorizer1 = TfidfVectorizer(stop_words='english')
tfidf1 = tfidf_vectorizer1.fit_transform(word_data)
feature_names = tfidf_vectorizer1.get_feature_names()
print len(feature_names)
```

```
29561
```

In [38]:

```
labels = from_data
features = tfidf1
features_train, features_test, labels_train, labels_test = cross_validation.train_test_spl
```

In [39]:

```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(features_train,labels_train)
pred = clf.predict(features_test)
sort_features = sorted(range(len(clf.feature_importances_)), key=lambda i: clf.feature_imp
sort_features = [x for x in sort_features if clf.feature_importances_[x]>0.01]
for x in sort_features:
    print "important key words", feature_names[x], "has importance", clf.feature_importanc
```

```
important key words mett has importance 0.0240669536209
important key words ddelainnsf has importance 0.0443758398404
important key words kaminski has importance 0.0706294468682
important key words vkamin has importance 0.0890770833449
important key words vinc has importance 0.75629211812
```

In [40]:

```
true_negatives = 0
false_negatives = 0
true_positives = 0
false_positives = 0
for prediction, truth in zip(pred, labels_test):
    if prediction == 0 and truth == 0:
        true_negatives += 1
    elif prediction == 0 and truth == 1:
        false_negatives += 1
    elif prediction == 1 and truth == 0:
        false_positives += 1
    elif prediction == 1 and truth == 1:
        true_positives += 1
total_predictions = true_negatives + false_negatives + false_positives + true_positives
accuracy = 1.0*(true_positives + true_negatives)/total_predictions
precision = 1.0*true_positives/(true_positives+false_positives)
recall = 1.0*true_positives/(true_positives+false_negatives)
print "accuracy",accuracy
print "precision",precision
print "recall",recall
```

```
accuracy 0.99772001824
precision 0.994360902256
recall 0.996233521657
```

Although the data looks great, the tfidf model conclude only 14 POI and 2 Non-poi (since we did not have others actual email address based on their name), the prediction is focus on the exist data.

In this case, I won't use this data in further analysis

Task 4: Try a varity of classifiers

In [41]:

```
def vari_class_test(clf):
    ### 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.Stratifi

    ### 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.

    test_classifier(clf, my_dataset, features_list_2)
```

GaussianNB

In [42]:

```
clf = GaussianNB()
vari_class_test(clf)
```

```
GaussianNB(priors=None)
        Accuracy: 0.81825       Precision: 0.42232      Recall: 0.24600 F1:
 0.31090        F2: 0.26841
        Total predictions: 12000        True positives:  492    False positi
ves:  673       False negatives: 1508   True negatives: 9327
```

Adaboost

In [43]:

```
clf =  AdaBoostClassifier()
vari_class_test(clf)
```

```
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=50, random_state=None)
        Accuracy: 0.83900       Precision: 0.52252      Recall: 0.39450 F1:
 0.44957        F2: 0.41483
        Total predictions: 12000        True positives:  789    False positi
ves:  721       False negatives: 1211   True negatives: 9279
```

K_Nearest_Neighbour

In [44]:

```
clf = KNeighborsClassifier(2)
vari_class_test(clf)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=2, p=2,
           weights='uniform')
        Accuracy: 0.86900       Precision: 0.75296      Recall: 0.31850 F1:
 0.44765        F2: 0.36005
        Total predictions: 12000        True positives:  637    False positi
ves:  209       False negatives: 1363   True negatives: 9791
```

Random_Forest

In [45]:

```
clf = RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1)
vari_class_test(clf)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features=1, max_leaf_nodes=None,
            min_impurity_split=1e-07, 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=False)
        Accuracy: 0.83442       Precision: 0.51422      Recall: 0.11750 F1:
 0.19129        F2: 0.13894
        Total predictions: 12000        True positives:  235    False positi
ves:  222       False negatives: 1765   True negatives: 9778
```

Now wew add PCA to each of them to find better results:

PCA + Adaboost

In [46]:

```
pipe = Pipeline(steps=[('pca', PCA()), ('classify', AdaBoostClassifier())])
vari_class_test(pipe)
```

```
Pipeline(steps=[('pca', PCA(copy=True, iterated_power='auto', n_components=N
one, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)), ('classify', AdaBoostClassifie
r(algorithm='SAMME.R', base_estimator=None,
          learning_rate=1.0, n_estimators=50, random_state=None))])
        Accuracy: 0.81058       Precision: 0.41267      Recall: 0.32250 F1:
 0.36205        F2: 0.33724
        Total predictions: 12000        True positives:  645    False positi
ves:  918       False negatives: 1355   True negatives: 9082
```

Task 5: Tune your classifier to achieve better than .3 precision and recall

PCA + K_Nearest_Neighbour

In [47]:

```
pipe = Pipeline(steps=[('pca', PCA()), ('classify', KNeighborsClassifier(2))])
vari_class_test(pipe)
```

```
Pipeline(steps=[('pca', PCA(copy=True, iterated_power='auto', n_components=N
one, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)), ('classify', KNeighborsClassif
ier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=2, p=2,
           weights='uniform'))])
        Accuracy: 0.86900       Precision: 0.75296      Recall: 0.31850 F1:
 0.44765        F2: 0.36005
        Total predictions: 12000        True positives:  637    False positi
ves:  209       False negatives: 1363   True negatives: 9791
```

PCA + Random_Forest

In [48]:

```
pipe = Pipeline(steps=[('pca', PCA()), ('classify', RandomForestClassifier(max_depth=5, n_
vari_class_test(pipe)
```

```
Pipeline(steps=[('pca', PCA(copy=True, iterated_power='auto', n_components=N
one, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)), ('classify', RandomForestClass
ifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=5, max_features=1, max_leaf_nodes=None,
            ...imators=10, n_jobs=1, oob_score=False, random_state=None,
            verbose=0, warm_start=False))])
        Accuracy: 0.83475       Precision: 0.51638      Recall: 0.13400 F1:
 0.21278        F2: 0.15730
        Total predictions: 12000        True positives:  268    False positi
ves:  251       False negatives: 1732   True negatives: 9749
```

After adding PCA, we can find all model increase precision and recall

Then we can test different parameters effect

In [49]:

```
import optunity
import optunity.metrics
```

Every learning algorithm has its own hyperparameters:

k-NN: 1<n_neighbors<5 the number of neighbours to use

naive Bayes: no hyperparameters

random forest: 10<n_estimators<30: number of trees in the forest 5<max_features<20: number of features to consider for each split

In [50]:

```
search = {'algorithm': {'k-nn': {'n_neighbors': [1, 5]},
                        'naive-bayes': None,
                        'random-forest': {'n_estimators': [10, 30],
                                          'max_features': [1, 5]}
                        }
         }
```