Identify Fraud from Enron Email

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.

In this project, our target is building a person of interest identifier based on financial and email data made public as a result of the Enron scandal. This data is combined with a hand-generated list of persons of interest in the fraud case, which means individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

By classification algorithms in machine learning, we expect to classify Enron employees into two categories: POI (person of interest) and non-POI. By using machine learning, we can detect more complicated patterns in a dataset with many features.

Data Exploration

- Total number of data points is 146 for the Enron dataset.
- Dataset includes 18 POIs and 128 non-POIs. Number of instances of POIs far exceeds non-POIs, so this causes **class imbalance** problem.
- 21 features is used for this data set. > - 14 financial features

```
- Payments -> Salary, Bonus, Long Term Incentive, Deferred Income, Deferr
al Payments, Loan Advances, Other,  Expenses, Director Fees, Total Paymen
ts
- Stock Value -> Exercised Stock Options, Restricted Stock, Restricted St
ock Deferred, Total Stock Value
```

> - 6 email features

```
- to messages, email address, from poi to this person, from messages, fro
m this person to poi, shared receipt with poi
```

> - 1 POI feature (will be used as label for this project)
- Except POI, features contain many missing values.

Features with Missing Values

In [3]:

```
with open("final_project_dataset.pkl", "r") as data_file:
    data_dict = pickle.load(data_file)
df = pd.DataFrame.from_dict(data_dict, orient='index')
df.replace(to_replace='NaN', value=np.nan, inplace=True)
146-df.count().sort_values()
```

Out[3]:

```
loan_advances                142
director_fees                129
restricted_stock_deferred    128
deferral_payments            107
deferred_income               97
long_term_incentive           80
bonus                         64
from_poi_to_this_person       60
to_messages                   60
shared_receipt_with_poi       60
from_this_person_to_poi       60
from_messages                 60
other                         53
salary                        51
expenses                      51
exercised_stock_options       44
restricted_stock              36
email_address                 35
total_payments                21
total_stock_value             20
poi                            0
dtype: int64
```

Especially, *loan_advances, director_fees, restricted_stock_deferred, deferral_payments, deferred_income* and *long_term_incentive* features has no values for more than half of the 146 data points.

Outlier Investigation

When we started to investigate data points, which one does not have any value is found:

In [4]:

```
temp = df[df.columns.difference(['poi'])]
temp[temp.isnull().all(axis=1)]
```

Out[4]:

	bonus	deferral_payments	deferred_income	director_fees	email_addres
LOCKHART EUGENE E	NaN	NaN	NaN	NaN	NaN

So we can remove this point. At the same time, email_address feature is object type;so it will not help us for classification in that moment. We can also remove this feature.

In [5]:

```
df.drop('LOCKHART EUGENE E', inplace = True)
df.drop(["email_address"], axis=1, inplace = True)
```

When I examined insiderpay document in project folder, I realized **THE TRAVEL AGENCY IN THE PARK** as a data point whic is not a person. However, I don't think we should remove it like an outlier. This point is not real person but it is related with real people. This firm is coowned by the sister of Enron's former Chairman according to footnotes, so I didn't remove it.

We recreate our feature list to be able to use functions for extraction features and labels in next steps. "poi" should be placed at first.

In [6]:

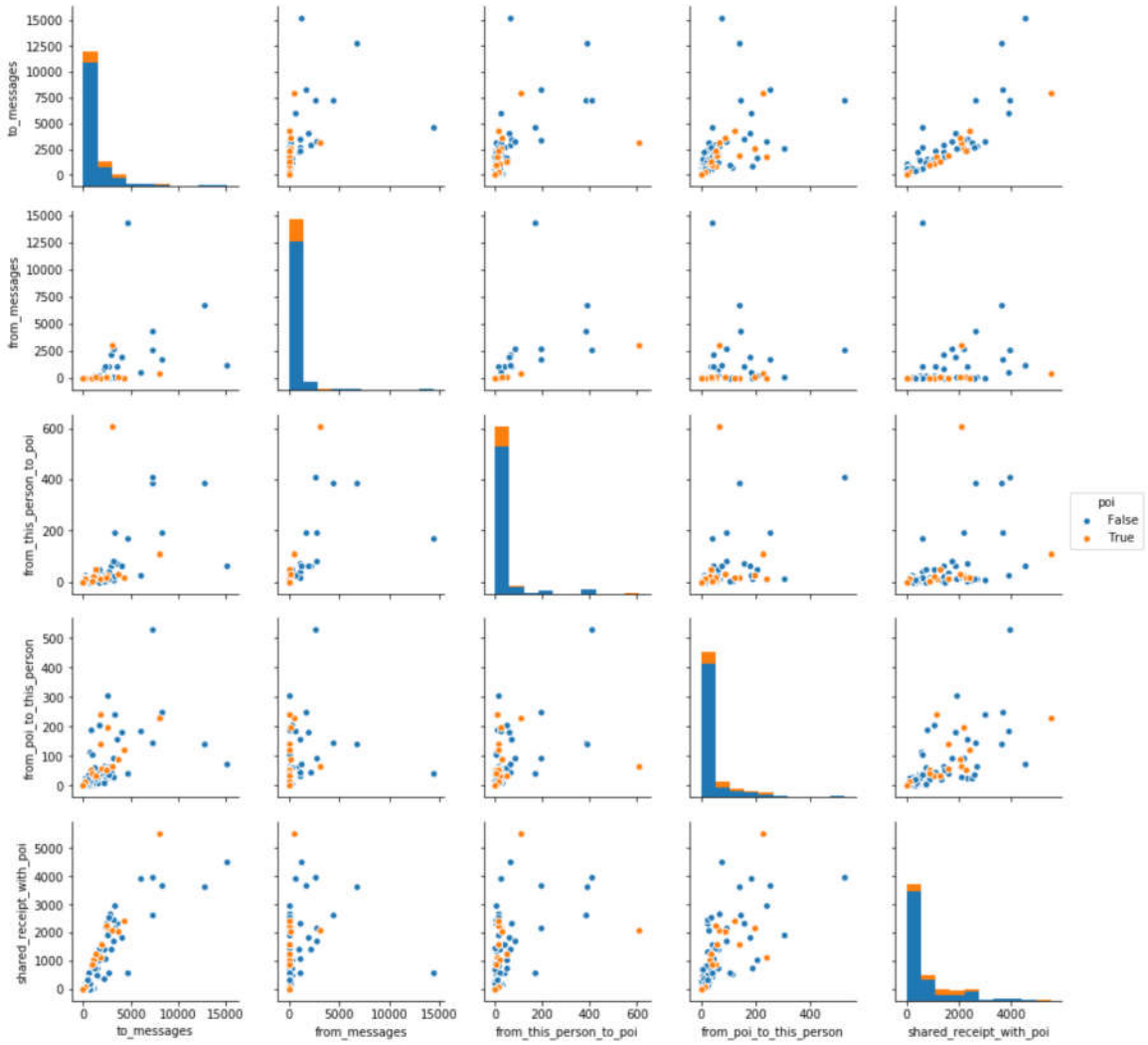
```
features_list = df.columns.tolist()
features_name = features_list[:]
features_name.remove('poi')
features_list = ['poi'] + features_name

df.fillna(0, inplace =True)
```

I observed features with minumum null value to get insight and also detect outliers.

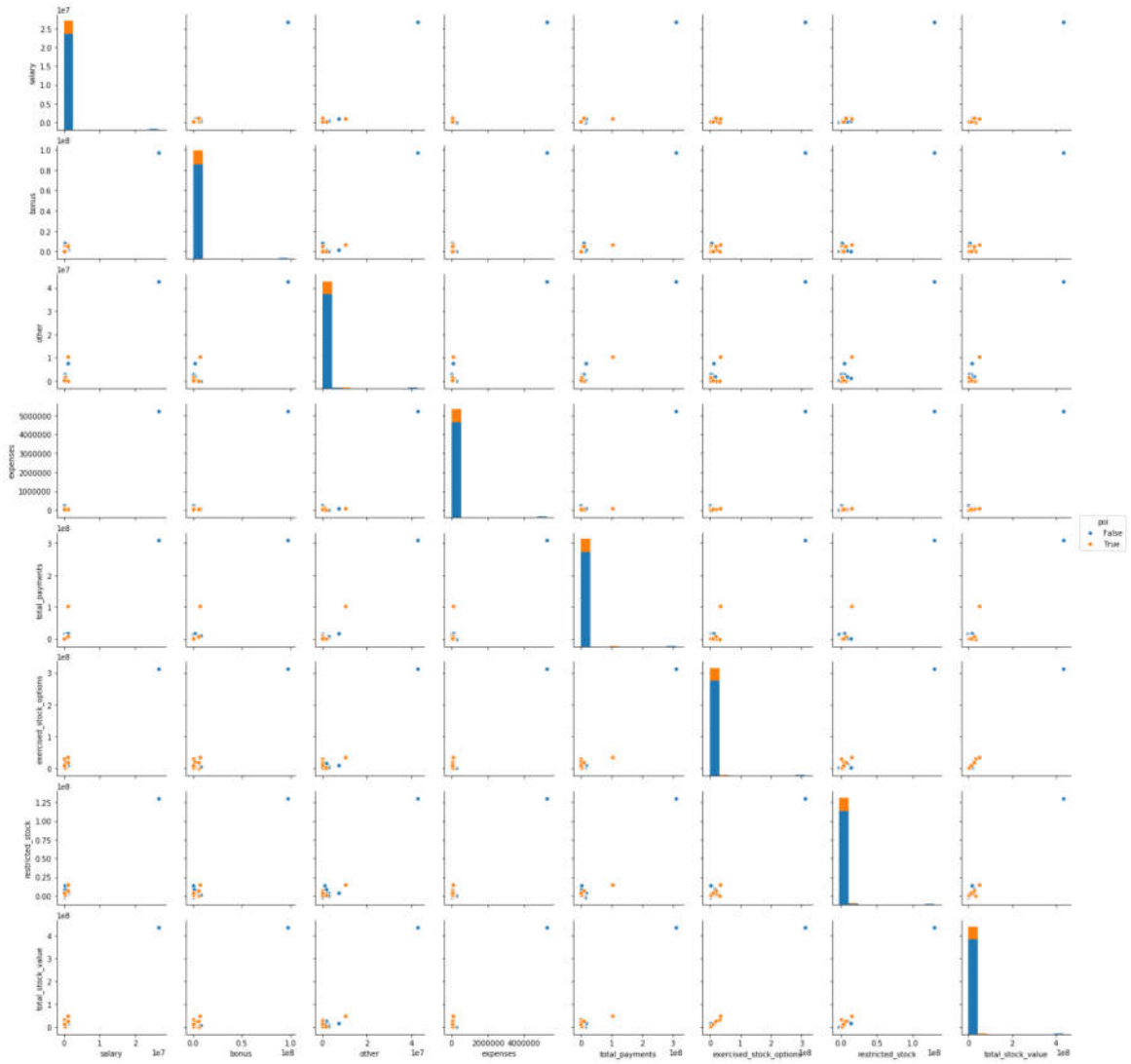
In [7]:

```
sns.pairplot(df, vars=['to_messages',
 'from_messages',
 'from_this_person_to_poi',
 'from_poi_to_this_person',
 'shared_receipt_with_poi'], hue="poi");
```



In [8]:

```
sns.pairplot(df, vars=['salary',
 'bonus',
 'other',
 'expenses',
 'total_payments',
 'exercised_stock_options',
 'restricted_stock',
 'total_stock_value'], hue="poi");
```



In [9]:

```
print "Point with Maximum Value:", [df["salary"].argmax(),
df["deferral_payments"].argmax(),
df["restricted_stock"].argmax(),
df["exercised_stock_options"].argmax(),
df["bonus"].argmax(), df["total_payments"].argmax(),
df["total_stock_value"].argmax()]
```

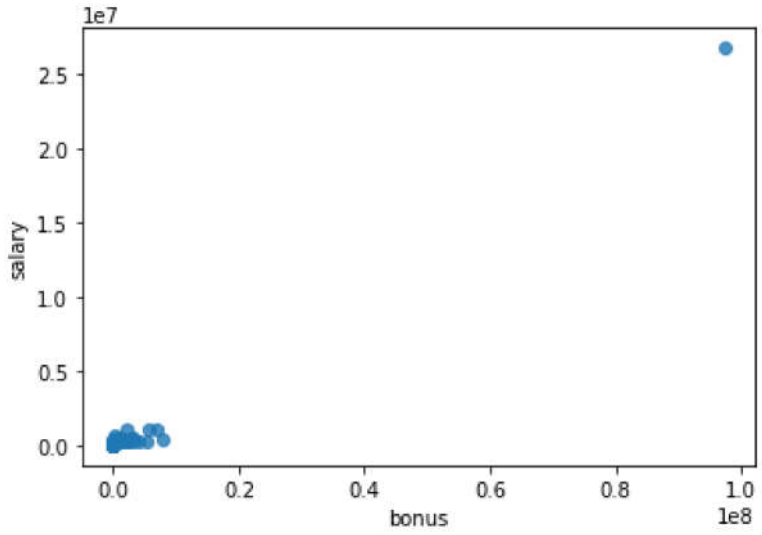
```
Point with Maximum Value: ['TOTAL', 'TOTAL', 'TOTAL', 'TOTAL', 'TOTAL', 'T
OTAL', 'TOTAL']
```

There is an outlier almost every graphs. This is total record of financial variables and we can seperate it from dataset.

For example, this is Salary vs Bonus graph before removing outlier:

In [10]:

```
sns.regplot(x="bonus", y="salary", data=df,fit_reg=False);
```



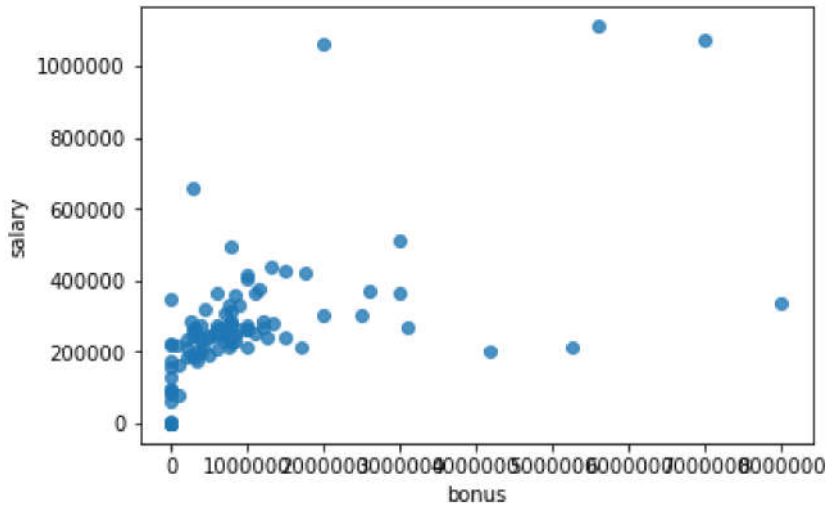
In [11]:

```
df.drop('TOTAL', inplace = True)
```

After the removal of outlier data point "TOTAL":

In [12]:

```
sns.regplot(x="bonus", y="salary", data=df, fit_reg=False);
```



List of features

I prefered to include all features at the beginning.

In [13]:

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

I used pipeline after this part of the project. Inside of the pipeline, I used SelectKBest for feature selection for all used algorithms.

Creating New Feautures

After getting insight dataset from pairplots, I created three new features. Receiving and sending larger number of emails with POIs than others could be indicator of close relationship. With this purpose, I created first two features.However, one-way send or receive could be meaningless in some circumstances like hierarchical reports sending or receiving frequently. I think, two-way communication may give us more clue,so I added the third feature.

Firstly,I would like to see affect of mostly being sender side in mail correspondence:

$$\text{ratio\_to\_poi} = \frac{\#\text{ of emails sent by this person to POIs}}{\text{Total }\#\text{ of emails sent by this person}}$$

Secondly,I would like to see affect of mostly being receiver side in mail correspondence:

$$\text{ratio\_from\_poi} = \frac{\#\text{ of emails received by this person from POIs}}{\text{Total }\#\text{ of emails received by this person}}$$

Thirdly, I would like see affect of two way traffic:

$$\text{poi\_traffic} = \text{ratio\_to\_poi x ratio\_from\_poi}$$

In [14]:

```
df['ratio_to_poi'] = df['from_poi_to_this_person'] / df['to_messages']
df['ratio_from_poi'] = df['from_this_person_to_poi'] / df['from_messages']
df['poi_traffic'] = df['ratio_from_poi'] * df['ratio_to_poi']
```

New features are added in my_features_list. After adding columns to dataframe, dataset dictionary is also updated.

In [15]:

```
my_features_list = features_list[:]
my_features_list.extend(('ratio_to_poi','ratio_from_poi',"poi_traffic"))
```

In [16]:

```
df = df.fillna(0)
my_dataset = df.to_dict(orient='index')
```

In [17]:

```
data = featureFormat(my_dataset, my_features_list, sort_keys = True)
labels, features = targetFeatureSplit(data)
```

These are final distribution of data set:

In [18]:

```
df.describe()
```

Out[18]:

	salary	to_messages	deferral_payments	total_payments	exercised_st
count	1.440000e+02	144.000000	1.440000e+02	1.440000e+02	1.440000e+02
mean	1.854460e+05	1238.555556	2.220896e+05	2.259057e+06	2.075802e+06
std	1.970421e+05	2237.564816	7.541013e+05	8.846594e+06	4.795513e+06
min	0.000000e+00	0.000000	-1.025000e+05	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000	0.000000e+00	9.964825e+04	0.000000e+00
50%	2.105960e+05	347.500000	0.000000e+00	9.413595e+05	6.082935e+05
75%	2.696675e+05	1623.000000	8.535500e+03	1.945668e+06	1.683580e+06
max	1.111258e+06	15149.000000	6.426990e+06	1.035598e+08	3.434838e+07

8 rows × 22 columns

Pipeline

After this point of the project, I used pipeline. Pipeline apply sequentially a list of transforms and a final estimator. While intermediate steps of the pipeline must implement fit and transform methods, the final estimator only needs to implement fit.

Also, using pipeline is need to warrant consistent processing between test and training set when using scaling,feature selection or dimension reduction.

My pipeline structure in this project: Scaling(Optional) -> SelectKBest-> PCA -> Classifier Algorithm

In [ ]:

```
"""
from sklearn.pipeline import Pipeline
pipeline = Pipeline([("skb", skb),('pca',pca),('dt',dt)])

tuned_param_grid = { "skb__k" : [14],
              'pca__n_components':[9],
              'dt__min_samples_split': [6] }
"""
```

Scaling Features

Scaling is standardize the features of data. Scaling is not required for all classifiers and it has possibility to cause information lost in some situations.If selected algorithm is not operated in coordinate plain like Naive Bayes, Decision Tree or Random Forest Classifier, there is no need to scaling. If selected algorithm is operated in coordinate plain like SVM or K-Nearest Neighbor Classifier, scaling is necessary.

StandardScaler is used for scaling features by removing the mean and scaling to unit variance.

In [19]:

```
scale = StandardScaler()
```

Selecting Features

The next step in the pipeline is selecting the feature that using features which explain the most part of data. SelectKBest is used for applying Univariate Feature Selection to reduce the noise and simplify the computation.

For document requirement, we can compare strength of new features to initial features manually by using SelectKBest.

In [20]:

```
skb_all = SelectKBest(k="all")
skb_all.fit(features,labels)
features_all=[my_features_list[i+1]for i in skb_all.get_support(indices=True)]
feature_scores_all =skb_all.scores_
features_scores_all=[feature_scores_all[i]for i in skb_all.get_support(indices=True)]
new = []
all_scores_table = pd.DataFrame([features_all,features_scores_all,new]).T.sort_values(1
,ascending=False)
all_scores_table.columns = ["features_all","features_scores_all","new"]
all_scores_table = all_scores_table.reset_index(drop=True)
new_features = ('ratio_to_poi','ratio_from_poi',"poi_traffic")
def label_new (row):
    if row['features_all'] in new_features:
        return 'Yes'
    return '-'
all_scores_table["new"]=all_scores_table.apply (lambda row: label_new (row),axis=1)
all_scores_table
```

Out[20]:

	features_all	features_scores_all	new
0	exercised_stock_options	25.0975	-
1	total_stock_value	24.4677	-
2	bonus	21.06	-
3	salary	18.5757	-
4	ratio_from_poi	16.6417	Yes
5	deferred_income	11.5955	-
6	long_term_incentive	10.0725	-
7	restricted_stock	9.3467	-
8	total_payments	8.86672	-
9	shared_receipt_with_poi	8.74649	-
10	loan_advances	7.24273	-
11	expenses	6.2342	-
12	from_poi_to_this_person	5.34494	-
13	poi_traffic	4.7384	Yes
14	other	4.20497	-
15	ratio_to_poi	3.21076	Yes
16	from_this_person_to_poi	2.42651	-
17	director_fees	2.10766	-
18	to_messages	1.69882	-
19	deferral_payments	0.217059	-
20	from_messages	0.164164	-
21	restricted_stock_deferred	0.0649843	-

We can oberve from this table, "ratio_from_poi" is 5th important feature for determining "poi" label. So it can be useful in next algorithms. Other two new features, "poi_traffic" and "ratio_to_poi" are less important nearly half of the features.

In [21]:

```
skb = SelectKBest()
```

In [24]:

```
"""
param_grid = { "skb__k":range(12,17),
               "pca__n_components":range(6,11),
               'dt__min_samples_split': [ 4, 6, 8]}
"""
```

Out[24]:

```
'\nparam_grid = { "skb__k":range(12,17), \n              "pca__n_component
s":range(6,11),\n              \'dt__min_samples_split\': [ 4, 6, 8]}\n'
```

Inside of the pipeline parameters definition, we defined range of K value with "skb__k". K value is the number of top features to select. Range is selected (12,16) so 12,13,14 and 15 integer values of k in SelectKBest have been tested.

According to Decision Tree Classifier pipeline, k=14 represented the best results.

In [28]:

```
scores_table # After processing Decision Tree Classifier
```

Out[28]:

	features_selected	features_scores_selected
0	exercised_stock_options	22.8469
1	total_stock_value	22.3346
2	salary	16.9609
3	bonus	15.4914
4	ratio_from_poi	13.806
5	restricted_stock	8.61001
6	total_payments	8.50624
7	loan_advances	7.34999
8	shared_receipt_with_poi	7.0634
9	deferred_income	6.19467
10	long_term_incentive	5.66331
11	expenses	5.28385
12	from_poi_to_this_person	5.05037
13	other	4.42181
14	poi_traffic	3.90447
15	ratio_to_poi	3.5745

After this selection, we will use PCA(Principal component analysis) to reduce dimension by projecting data to a lower dimensional space.

In [22]:

```
pca = PCA(random_state=42)
```

n_components value is the number of components to keep. Range is selected (5,11) so 5,6,7,8,9 and 10 integer values of n_components in PCA have been tested.

According to Decision Tree Classifier pipeline, n_components=8 gave the best results

Picking an Algorithm

I attempted use Gaussian Naive-Bayes, Decision Tree Classifier and K-Nearest Neighbors algorithms.

In [23]:

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()

from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier (random_state=42)

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
```

Tuning the Algorithm

Tuning is to adjust parameters of algorithms in order to maximize the evaluation metrics are determined.This metric is selected as f1 for this case. GridSearchCV is used for tuning the algorithms by experimenting for the best parameters from specified array in this project.

Tuning and validation on the same data without cross validation has posibility to overfit or underfit. Cross validation step prevents this situation. So validation method is defined in GridSearchCv to be sure not to encounter with overfitting.

Gaussian Naive Bayes has no parameters for tuning. For meeting the requirement, I also picked Decision Tree Classifier and K-Nearest Neighbor for tuning.

For Decision Tree Classifier, "min_samples_split" parameter is tuned. 4,6 and 8 is tested for min_samples_split and best result is given by 8.

- dt__min_samples_split': [4, 6, 8]}
- dt__min_samples_split': [6]

For K-Nearest Neighbor Classifier, "n_neighbors" parameter is tuned. Range is defined as (2,6) and best result is given by 3.

- knn__n_neighbors': range(2,8)
- knn__n_neighbors': [3]

Evaluation Metrics

	Predicted POI	**Predicted non-POI**	
Actual POI	TP	FN	
Actual non-POI	FP	TN	

- True Positive: Actual POI and Predicted as POI
- False Negative: Actual POI and Predicted as non-POI
- False Positive: Actual non-POI and Predicted as POI
- True Negative: Actual non-POI and Predicted as non-POI

Accuracy shows the ratio between right classifications and the total number of predicted labels. For our case, accuracy is the ratio between "number of right POI and right non-POI predictions" and "all total predictions".

$$\text{Accuracy} = \frac{\text{Actual POI and Predicted as POI} + \text{Actual non-POI and Predicted as non-POI}}{\text{Total Predictions}}$$

Accuracy is not a strong metric for this dataset, because POI/non-POI distribution is not balanced. A model predicting that everyone is non-POI will give high accuracy but no inference for case.

Precision is the ratio of correct classifications over all observations with positive label. The higher precision score,the less false positives. For our case, precision is "number of right POI predictions" over "all total predictions as POI".

$$\text{Precision} = \frac{\text{Actual POI and Predicted as POI}}{\text{Actual POI and Predicted as POI} + \text{Actual non-POI and Predicted as POI}}$$

Recall is the ratio of correct classifications as positive label over all actual positive labels. The higher recall score, less false negatives.For our case, racall is "number of right POI predictions" over "number of all actual POI".

$$\text{Recall} = \frac{\text{Actual POI and Predicted as POI}}{\text{Actual POI and Predicted as POI} + \text{Actual POI and Predicted as non-POI}}$$

We want to be sure all possible POIs included at this stage even if some people will be absolved after next investigation. So, recall is more important according to this insight.

F1 is weighted average of the precision and recall.

$$\text{F1} = 2\frac{(\text{Recall x Precision})}{\text{Recall} + \text{Precision}}$$

Validation

Validation is model performance evaluation by excluding training data to prevent overfitting. With this aim, data is partitioned into different training set and validation set. Model is fitted by using training set and the validation set is used to measure its performance. Without validation, our model performs mostly performs excellent with training data but performs poor with new data.

Validation Strategy

When tuning parameters, 'StratifiedShuffleSplit' should be used directly for creating multiple test/train splits instead of a standard test/train split and 'StratifiedShuffleSplit' .The dataset is small and imbalanced so we maximize the number of POIs in test subsets to improve validation by applying 'StratifiedShuffleSplit' directly.

If we used standard test/train split, we have 102 observations in training subset and 44 observations in teet. test subsst subset. On average 5-6 POIs in the test subset.If we use 'StratifiedShuffleSplit' then, POIs will be decreased due to test size 0.3. So we should use 'StratifiedShuffleSplit' directly for this project.

Algorithm Performance

Algorithm	Accuracy	Precision	Recall	F1
Gaussian Naive Bayes	0.85540	0.43977	0.30850	0.36262
Decision Tree Classifier	0.83220	0.36330	0.34350	0.35312
K-Nearest Neighbor	0.83213	0.22737	0.10800	0.14644

We want to be sure all possible POIs included at this stage even if some people will be absolved after next investigation. So, recall is more important according to this insight.

For this reason, my selection will be **Decision Tree Classifier**.

Selected Algorithm Pipeline

Decision Tree Classifier

Tested Parameters:

- **SKB:** k: [12, 13, 14, 15, 16]
- **PCA:** n_components: [6, 7, 8, 9, 10]
- **DT:** min_samples_split=[4,6,8]

Best Parameters:

- **SKB:** k: [16]
- **PCA:** n_components: [8]
- **DT:** min_samples_split=[6]

Scores:

- Accuracy: 0.83220 Precision: 0.36330 Recall: 0.34350 F1: 0.35312 F2: 0.34729

In [24]:

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier (random_state=42)

pipeline = Pipeline([("skb", skb),('pca',pca),('dt',dt)])
param_grid = { "skb__k":range(12,17), "pca__n_components":range(6,11),'dt__min_samples_
split': [4,6,8]}
#tuned_param_grid = { "skb__k":[16], "pca__n_components":[9], 'dt__min_samples_split':
 [6]}

folds = 100
sss = StratifiedShuffleSplit(labels, folds, test_size=0.3, random_state=42)
dt_grid = GridSearchCV(pipeline, param_grid, cv=sss, scoring='f1')
dt_grid.fit(features,labels)
```

Out[24]:

```
GridSearchCV(cv=StratifiedShuffleSplit(labels=[ 0.  0. ...,  1.  0.], n_it
er=100, test_size=0.3, random_state=42),
       error_score='raise',
       estimator=Pipeline(steps=[('skb', SelectKBest(k=10, score_func=<fun
ction f_classif at 0x7f40fd5e1758>)), ('pca', PCA(copy=True, iterated_powe
r='auto', n_components=None, random_state=42,
  svd_solver='auto', tol=0.0, whiten=False)), ('dt', DecisionTreeClassifie
r(class_weight=None, criterion='gini', max_depth=None,
   ...plit=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=42, splitter='best'))]),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'pca__n_components': [6, 7, 8, 9, 10], 'dt__min_samples
_split': [4, 6, 8], 'skb__k': [12, 13, 14, 15, 16]},
       pre_dispatch='2*n_jobs', refit=True, scoring='f1', verbose=0)
```

In [25]:

```
clf = dt_grid.best_estimator_
clf
```

Out[25]:

```
Pipeline(steps=[('skb', SelectKBest(k=16, score_func=<function f_classif a
t 0x7f40fd5e1758>)), ('pca', PCA(copy=True, iterated_power='auto', n_compo
nents=9, random_state=42,
  svd_solver='auto', tol=0.0, whiten=False)), ('dt', DecisionTreeClassifie
r(class_weight=None, criterion='gini', max_depth=None,
      ...plit=6, min_weight_fraction_leaf=0.0,
            presort=False, random_state=42, splitter='best'))])
```

In [26]:

```
test_classifier(clf,my_dataset,my_features_list)
```

```
Pipeline(steps=[('skb', SelectKBest(k=16, score_func=<function f_classif a
t 0x7f40fd5e1758>)), ('pca', PCA(copy=True, iterated_power='auto', n_compo
nents=9, random_state=42,
  svd_solver='auto', tol=0.0, whiten=False)), ('dt', DecisionTreeClassifie
r(class_weight=None, criterion='gini', max_depth=None,
      ...plit=6, min_weight_fraction_leaf=0.0,
            presort=False, random_state=42, splitter='best'))])
        Accuracy: 0.83220       Precision: 0.36330      Recall: 0.34350 F
1: 0.35312      F2: 0.34729
        Total predictions: 15000        True positives:  687    False posi
tives: 1204     False negatives: 1313   True negatives: 11796
```

Feature Scores

In [27]:

```
dt_grid.best_estimator_.named_steps['skb'].get_support()
features_selected=[my_features_list[i+1]for i in dt_grid.best_estimator_.named_steps['s
kb'].get_support(indices=True)]

feature_scores =dt_grid.best_estimator_.named_steps['skb'].scores_
features_scores_selected=[feature_scores[i]for i in dt_grid.best_estimator_.named_steps
['skb'].get_support(indices=True)]

scores_table = pd.DataFrame([features_selected,features_scores_selected]).T.sort_values
(1,ascending=False)
scores_table.columns = ["features_selected","features_scores_selected"]
scores_table = scores_table.reset_index(drop=True)
scores_table
```

Out[27]:

	features_selected	features_scores_selected
0	exercised_stock_options	22.8469
1	total_stock_value	22.3346
2	salary	16.9609
3	bonus	15.4914
4	ratio_from_poi	13.806
5	restricted_stock	8.61001
6	total_payments	8.50624
7	loan_advances	7.34999
8	shared_receipt_with_poi	7.0634
9	deferred_income	6.19467
10	long_term_incentive	5.66331
11	expenses	5.28385
12	from_poi_to_this_person	5.05037
13	other	4.42181
14	poi_traffic	3.90447
15	ratio_to_poi	3.5745

Initial Features with Selected Algorithm

In [29]:

```
test_classifier(clf,my_dataset,features_list)
```

```
Pipeline(steps=[('skb', SelectKBest(k=16, score_func=<function f_classif a
t 0x7f40fd5e1758>)), ('pca', PCA(copy=True, iterated_power='auto', n_compo
nents=9, random_state=42,
  svd_solver='auto', tol=0.0, whiten=False)), ('dt', DecisionTreeClassifie
r(class_weight=None, criterion='gini', max_depth=None,
      ...plit=6, min_weight_fraction_leaf=0.0,
            presort=False, random_state=42, splitter='best'))])
        Accuracy: 0.83093       Precision: 0.35699      Recall: 0.33450 F
1: 0.34538      F2: 0.33877
        Total predictions: 15000        True positives:  669    False posi
tives: 1205     False negatives: 1331   True negatives: 11795
```

In [30]:

```
dt_grid.best_estimator_.named_steps['skb'].get_support()
features_selected=[features_list[i+1]for i in dt_grid.best_estimator_.named_steps['skb'
].get_support(indices=True)]

feature_scores =dt_grid.best_estimator_.named_steps['skb'].scores_
features_scores_selected=[feature_scores[i]for i in dt_grid.best_estimator_.named_steps
['skb'].get_support(indices=True)]

scores_table2 = pd.DataFrame([features_selected,features_scores_selected]).T.sort_value
s(1,ascending=False)
scores_table2.columns = ["features_selected","features_scores_selected"]
scores_table2 = scores_table2.reset_index(drop=True)
scores_table2
```

Out[30]:

	features_selected	features_scores_selected
0	exercised_stock_options	22.8469
1	total_stock_value	22.3346
2	salary	16.9609
3	bonus	15.4914
4	restricted_stock	8.61001
5	total_payments	8.50624
6	loan_advances	7.34999
7	shared_receipt_with_poi	7.0634
8	deferred_income	6.19467
9	long_term_incentive	5.66331
10	expenses	5.28385
11	from_poi_to_this_person	5.05037
12	other	4.42181
13	director_fees	1.76607
14	to_messages	1.48694
15	from_messages	0.811423

Results Initial Features vs. Latest Features

After training selected classifier in the original subset of features, we could assess results with the dataset with the newly generated ones to determine the impact of the new features.

Algorithm	Accuracy	Precision	Recall	F1
Initial Features	0.83093	0.35699	0.33450	0.34538
Latest Features	0.83220	0.36330	0.34350	0.35312

As we can see, our new created features "ratio_from_poi","poi_traffic" and "ratio_to_poi" impacted on raising recall score which is important for this case.

Not selected Algorithms Pipelines

Gaussian Naive Bayes

Tested Parameters:

- **SKB:** k: [12, 13, 14, 15, 16]
- **PCA:** n_components: [6, 7, 8, 9, 10]

Best Parameters:

- **SKB:** k: [16]
- **PCA:** n_components: [8]

Scores:

- Accuracy: 0.85540 Precision: 0.43977 Recall: 0.30850 F1: 0.36262 F2: 0.32809

In [40]:

```
pipeline = Pipeline([("skb", skb),('pca',pca),('nb',nb)])
param_grid = { "skb__k":range(12,17), "pca__n_components":range(6,11)}
#param_grid = { "skb__k":[16], "pca__n_components":[8]}

folds = 100
sss = StratifiedShuffleSplit(labels, folds, test_size=0.3, random_state=42)

nb_grid = GridSearchCV(pipeline, param_grid, cv=sss, scoring='f1')

nb_grid.fit(features,labels)
```

Out[40]:

```
GridSearchCV(cv=StratifiedShuffleSplit(labels=[ 0.  0. ...,  1.  0.], n_it
er=100, test_size=0.3, random_state=42),
       error_score='raise',
       estimator=Pipeline(steps=[('skb', SelectKBest(k=10, score_func=<fun
ction f_classif at 0x7f40fd5e1758>)), ('pca', PCA(copy=True, iterated_powe
r='auto', n_components=None, random_state=42,
  svd_solver='auto', tol=0.0, whiten=False)), ('nb', GaussianNB(priors=Non
e))]),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'pca__n_components': [6, 7, 8, 9, 10], 'skb__k': [12, 1
3, 14, 15, 16]},
       pre_dispatch='2*n_jobs', refit=True, scoring='f1', verbose=0)
```

In [41]:

```
clf = nb_grid.best_estimator_
clf
```

Out[41]:

```
Pipeline(steps=[('skb', SelectKBest(k=16, score_func=<function f_classif a
t 0x7f40fd5e1758>)), ('pca', PCA(copy=True, iterated_power='auto', n_compo
nents=8, random_state=42,
  svd_solver='auto', tol=0.0, whiten=False)), ('nb', GaussianNB(priors=Non
e))])
```

In [42]:

```
test_classifier(clf,my_dataset,my_features_list)
```

```
Pipeline(steps=[('skb', SelectKBest(k=16, score_func=<function f_classif a
t 0x7f40fd5e1758>)), ('pca', PCA(copy=True, iterated_power='auto', n_compo
nents=8, random_state=42,
  svd_solver='auto', tol=0.0, whiten=False)), ('nb', GaussianNB(priors=Non
e))])
        Accuracy: 0.85540       Precision: 0.43977      Recall: 0.30850 F
1: 0.36262      F2: 0.32809
        Total predictions: 15000        True positives:  617    False posi
tives:  786     False negatives: 1383   True negatives: 12214
```

K-Neighbors Classifier

Tested Parameters:

- **SKB:** k: [12,13, 14, 15, 16]
- **PCA:** n_components: [6, 7, 8]
- **KNN:** n_neighbors=range(2,6)

Best Parameters:

- **SKB:** k: [12]
- **PCA:** n_components: [7]
- **KNN**:n_neighbors=[3]

Scores:

- Accuracy: 0.83467 Precision: 0.17828 Recall: 0.06650 F1: 0.09687 F2: 0.07603

In [43]:

```
knn = KNeighborsClassifier()

pipeline = Pipeline([("scale",scale),("skb", skb),('pca',pca),('knn',knn)])
param_grid = {"scale":[scale],
              "skb__k":range(12,17),
              "pca__n_components":range(6,11),
              'knn__n_neighbors': range(2,6)
             }
#tuned_param_grid = {"scale":[StandardScaler()], "skb__k":[12], "pca__n_components":
[7], 'knn__n_neighbors': [3]}

folds = 100
sss = StratifiedShuffleSplit(labels, folds, test_size=0.3, random_state=42)
knn_grid = GridSearchCV(pipeline, param_grid, cv=sss, scoring='f1')
knn_grid.fit(features,labels)
```

Out[43]:

```
GridSearchCV(cv=StratifiedShuffleSplit(labels=[ 0.  0. ...,  1.  0.], n_it
er=100, test_size=0.3, random_state=42),
       error_score='raise',
       estimator=Pipeline(steps=[('scale', StandardScaler(copy=True, with_
mean=True, with_std=True)), ('skb', SelectKBest(k=10, score_func=<function
f_classif at 0x7f40fd5e1758>)), ('pca', PCA(copy=True, iterated_power='aut
o', n_components=None, random_state=42,
  svd_solver='auto', tol=0.0, whiten=False)), ('knn', KNeighborsClassifier
(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'))]),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'pca__n_components': [6, 7, 8, 9, 10], 'scale': [Standa
rdScaler(copy=True, with_mean=True, with_std=True)], 'knn__n_neighbors':
[2, 3, 4, 5], 'skb__k': [12, 13, 14, 15, 16]},
       pre_dispatch='2*n_jobs', refit=True, scoring='f1', verbose=0)
```

In [44]:

```
clf = knn_grid.best_estimator_
clf
```

Out[44]:

```
Pipeline(steps=[('scale', StandardScaler(copy=True, with_mean=True, with_s
td=True)), ('skb', SelectKBest(k=12, score_func=<function f_classif at 0x7
f40fd5e1758>)), ('pca', PCA(copy=True, iterated_power='auto', n_components
=7, random_state=42,
  svd_solver='auto', tol=0.0, whiten=False)), ('knn', KNeighborsClassifier
(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=3, p=2,
           weights='uniform'))])
```