Enron financial and email data analysis

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.

I am building a person of interest identifier based on financial and email data made public as a result of the Enron scandal. we have data 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.

The algorithm can be used to find POI and run investigations for involvement in scandal.

Check what features available in data set:

```
print 'features: ', data_dict[data_dict.keys()[0]].keys()
```

```
Number of features:  21
features:  ['salary', 'to_messages', 'deferral_payments', 'total_payments', 'exercis
ed_stock_options', 'bonus', 'restricted_stock', 'shared_receipt_with_poi', 'restrict
ed_stock_deferred', 'total_stock_value', 'expenses', 'loan_advances', 'from_messages
', 'other', 'from_this_person_to_poi', 'poi', 'director_fees', 'deferred_income', 'l
ong_term_incentive', 'email_address', 'from_poi_to_this_person']
```

Let's verify quality of data:

```
for emp_features in data_dict.values():
    for feature in features:
        if feature in emp_features.keys():
            if feature == 'email_address':
                if emp_features[feature] != '':
                    features_data[feature][0] += 1
                    if emp_features['poi'] == True:
                        features_data[feature][2] += 1
            elif type(emp_features[feature]) != type('str'):
                features_data[feature][0] += 1
                if emp_features['poi'] == True:
                    features_data[feature][2] += 1
#calculate percentage of employees having data
for key, value in features_data.items():
    features_data[key][1] = (value[0]*100/len(data_dict))
```

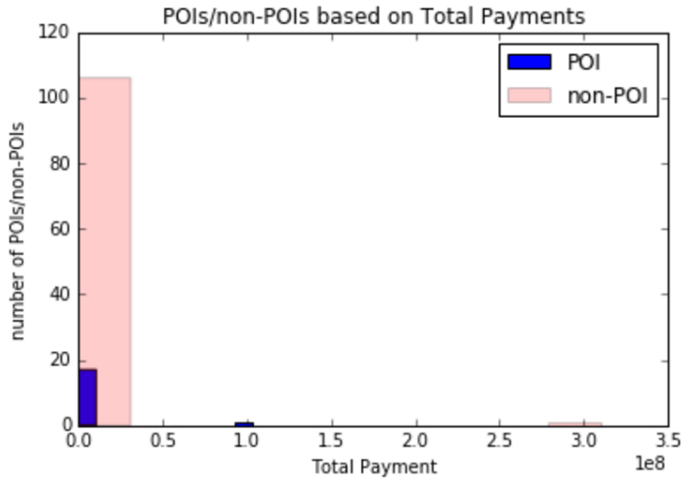
```
Total Number of Employees:  146
Total Number of POIs:  18
*** Data Quality ***
Feature: [Total Available, Percentage, POIs]
 loan_advances: [4, 2, 1]
 director_fees: [17, 11, 0]
 restricted_stock_deferred: [18, 12, 0]
 deferral_payments: [39, 26, 5]
 deferred_income: [49, 33, 11]
 long_term_incentive: [66, 45, 12]
 bonus: [82, 56, 16]
 to_messages: [86, 58, 14]
 shared_receipt_with_poi: [86, 58, 14]
 from_messages: [86, 58, 14]
 from_poi_to_this_person: [86, 58, 14]
 from_this_person_to_poi: [86, 58, 14]
 other: [93, 63, 18]
 salary: [95, 65, 17]
 expenses: [95, 65, 18]
 exercised_stock_options: [102, 69, 12]
 restricted_stock: [110, 75, 17]
```

```
total_payments: [125, 85, 18]
total_stock_value: [126, 86, 18]
email_address: [146, 100, 18]
poi: [146, 100, 18]
```

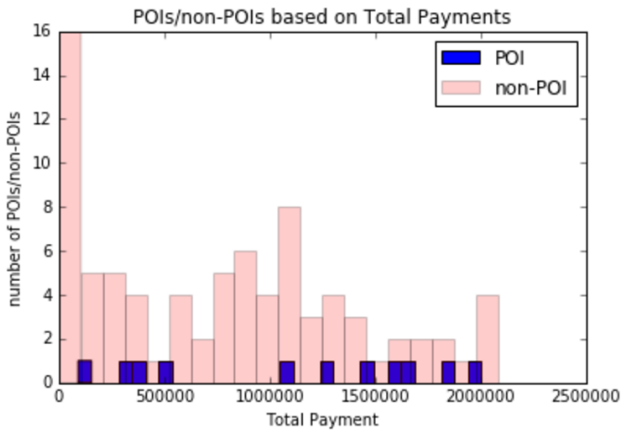
We have only 12% of employees in data are marked as POI. Lets try to understand if there is any criteria in data by plots. So we can identify what are the best features can be used to train the classifier. *Data not available in 6 features for more than 50% of employees, we will try algorithm later without these features*

Explore data to find if any feature distinguishes POIs

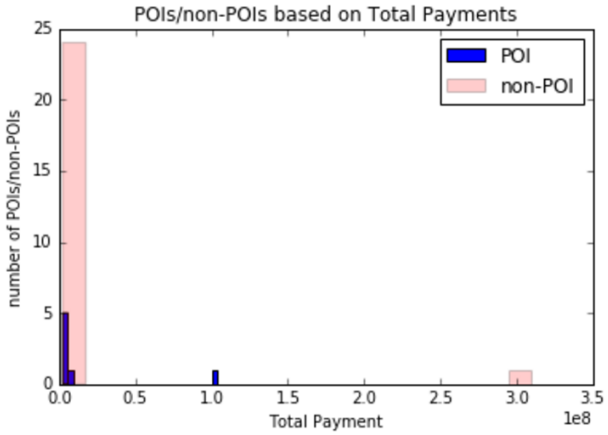
We have 85% of employees having total payment in data and all 18 POIs have this data. Let's analyze this.



We have outliers those are causing data squeezed. Let,s plot for data under IQR-Q3 and above IQR-Q3




```
number of POIs under IQR-Q3:  11
number of non-POIs under IQR-Q3:  82
```




```
number of POI in Outliers:  7
number of non-POIs in Outliers:  25
```

We do not see any clear distinguishion related to Total Payment between POIs and non POIs.

If we look into outlier plot, we have some POIs and non-POIs under 20 mm (aaprx), I would still keep these employees in my data, but let's look at 2 employees having 100mm or more

```
for key, value in data_dict.items():
    if type(value['total_payments']) <> type('str') and value['total_payments'] >= 1e8:
        print key
        pp.pprint(value)
```

```
LAY KENNETH L
{'bonus': 7000000,
 'deferral_payments': 202911,
 'deferred_income': -300000,
 'director_fees': 'NaN',
 'email_address': 'kenneth.lay@enron.com',
 'exercised_stock_options': 34348384,
 'expenses': 99832,
 'from_messages': 36,
 'from_poi_to_this_person': 123,
 'from_this_person_to_poi': 16,
 'loan_advances': 81525000,
 'long_term_incentive': 3600000,
 'other': 10359729,
 'poi': True,
 'restricted_stock': 14761694,
 'restricted_stock_deferred': 'NaN',
 'salary': 1072321,
 'shared_receipt_with_poi': 2411,
 'to_messages': 4273,
 'total_payments': 103559793,
 'total_stock_value': 49110078}
```

```
TOTAL
{'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}
```

It seems one of the data point is TOTAL, which is not an employee, it does not have email and number of emails too. Let's remove this outlier.
We have only one person around 100 mm payment, so it is not good idea to train algorithm with such an outlier. We can remove LAY KENNETH from our data

```
Number of Employees after Outlier removed:  144
```

Create new features:

Let's plot number of emails between POIs v/s non-POIs to see if there is any criteria for POI, but it's good idea to use percentage of emails from/to POI rather number, as some people might use email communication highly and some not, so percentage is good choice.

So let's create 2 new features

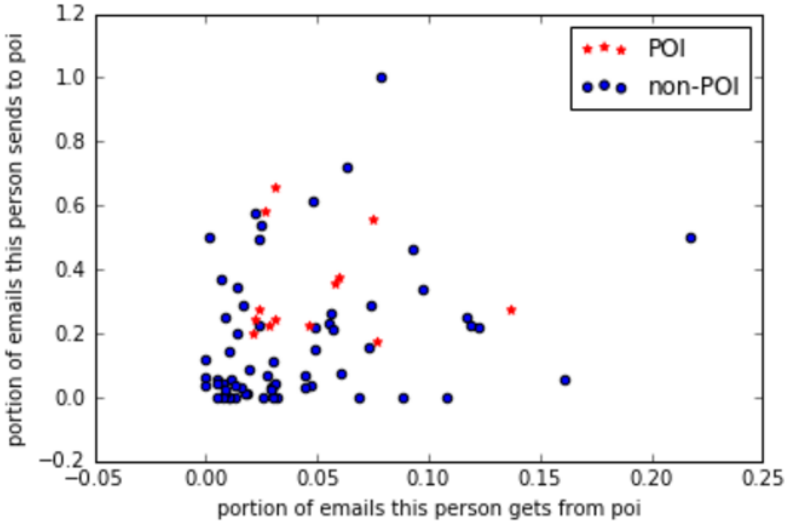
Fraction of emails from POI = Number of emails received from POI / Total number of emails received

Fraction of emails to POI = Number of emails sent to POI / Total number of emails sent

```
def dict_to_list(key,normalizer):
    new_list=[]

    for i in data_dict:
        if data_dict[i][key]=="NaN" or data_dict[i][normalizer]=="NaN":
            new_list.append(0.)
        elif data_dict[i][key]>=0:
            new_list.append(float(data_dict[i][key])/float(data_dict[i][normalizer]))
    return new_list
```

Scatter plot on new features:



We do not really see any behavior here.

Run supervised learning algorithm¶

Lets try decision tree classifier to find important features using feature_importances

If we see data quality output in prior sections, some of the features having data for less than 50% employees, so let's run algorithm for the features with at least 50% data

```
features_li = ["poi", "salary", "bonus", "fraction_from_poi_email", "fraction_to_poi_email",
                 'total_payments', 'total_stock_value', 'expenses', 'exercised_stock_options',
                 'shared_receipt_with_poi', 'restricted_stock']
features_train, features_test, labels_train, labels_test = train_test_data(features_li)
clf0 = DecisionTreeClassifier()
print '*** Decesiontree Algorithm - features with atleast 50% having data***'
print test_algorithm(clf0,features_train,features_test), '\n'
```

```
*** Decesiontree Algorithm - features with atleast 50% having data***
{'F1_Score': 0.55, 'Recall': 0.6, 'Duration_sec': 0.001, 'Precision': 0.5, 'Accuracy
': 0.88}
```

Run decision tree algorithm with all features (Note that, we don't have to include e mail address and number of emails in feature list):

```
features_list = ["poi", "salary", "bonus", "fraction_from_poi_email", "fraction_to_poi_email",
                 'deferral_payments', 'total_payments', 'loan_advances', 'restricted_stock_deferred',
                 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options',
                 'long_term_incentive', 'shared_receipt_with_poi', 'restricted_stock', 'director_fees']
features_train, features_test, labels_train, labels_test = train_test_data(features_list)

clf1 = DecisionTreeClassifier()
print '*** Decesiontree Algorithm with all features ***'
print test_algorithm(clf1,features_train,features_test), '\n'
```

```
*** Decesiontree Algorithm with all features ***
{'F1_Score': 0.22, 'Recall': 0.2, 'Duration_sec': 0.002, 'Precision': 0.25, 'Accurac
y': 0.84}

Feature Ranking:
        1 salary (0.227272727273)
        2 bonus (0.156605113636)
        3 fraction_from_poi_email (0.14678030303)
        4 fraction_to_poi_email (0.137884527089)
        5 deferral_payments (0.136134164898)
        6 total_payments (0.104429713805)
```

```
7 loan_advances (0.0473484848485)
8 restricted_stock_deferred (0.04354496542)
9 deferred_income (0.0)
10 total_stock_value (0.0)
11 expenses (0.0)
12 exercised_stock_options (0.0)
13 long_term_incentive (0.0)
14 shared_receipt_with_poi (0.0)
15 restricted_stock (0.0)
16 director_fees (0.0)
```

We have good accuracy and precision with features having more than 50% data (call it 50%Classifier). Let's try selecting top percentile features and check if we can get same or better result

```
percents = [90, 80, 70, 60, 50, 40, 30, 20, 10]
for percent in percents:
    clf2 = DecisionTreeClassifier()
    select_features(clf2, percent)
```

```
def select_features(clf, perc_fea):
    selector = SelectPercentile(f_classif, percentile=perc_fea)
    selector.fit(features_train,labels_train)
    features_train_s = selector.transform(features_train)
    features_test_s = selector.transform(features_test)
    metrics = test_algorithm(clf,features_train_s,features_test_s)
```

Percentile	Accuracy	Precision	Recall	F1_score	Time (sec)
90	86%	43%	60%	50%	0.002
80	88%	50%	60%	55%	0.001
70	88%	50%	60%	55%	0.001
…	…	…	…	…	…

If I use percentile less than 95, precision, recall and accuracy starts dropping. But again at 30th percentile, metrics approximately matches first classifier using all features, but not greater. This is kind of regularization point but we do not have to really consider performance to identify POIs, so I would still go with first classifier.

***Let's** try naive bayes algorithm and compare result.*

```
clf3 = GaussianNB()
select_features(clf3, percent)
```

Percentile	Accuracy %	Precision %	Recall %	F1_score %	Time (sec)
90	86	40	40	40	0.001
80	88	50	60	55	0.001
70	86	43	60	50	0.001
60	86	40	40	40	0.001
50	88	50	60	55	0.001
40	86	43	60	50	0.001
…	…	…	…	…	…

The best metrics available at 50% features and which matches to our 50%Classfier and then metrics start dropping. This is kind of regularization point. We do not have to really consider performance to identify POIs. Any way I would still go with 50%classifier which relies on features having data.

Let's try naive bayes algorithm and compare result

```
percents = [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
for percent in percents:
    clf3 = GaussianNB()
    select_features(clf3, percent)
```

Percentile	Accuracy %	Precision %	Recall %	F1_score %	Time (sec)
100	33	15	10	26	0.001
90	33	15	10	26	0.002
80	65	25	10	40	0.002
…	…	…	…	…	…

All the metrics are too low comparing to Decision Classifier, so using Decision classifier is better choice.

Let's try manually selecting payment features

```
features_manual = ["poi", "salary", "bonus",
                   'deferral_payments', 'total_payments', 'loan_advances', 'restricted_stock_deferred',
                   'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options',
                   'long_term_incentive', 'restricted_stock', 'director_fees']
```

```
*** Decesiontree Algorithm with only payments features ***
{'F1_Score': 0.31, 'Recall': 0.4, 'Duration_sec': 0.002, 'Precision': 0.25, 'Accurac
y': 0.79}
```

Precision is too low, we are still better with 50%classifier.

Parameter tuning

We have run algorithms with default parameters till now, but algorithm might do more accurate fitting for decision surface by changing parameters. Let's repeat 50%classifier with different classifier parameters to see if we can achieve better result with any other parameters in algorithm

```
parameters = {'criterion':('gini', 'entropy')}
dtc = DecisionTreeClassifier()
clf5 = grid_search.GridSearchCV(dtc, parameters)
```

```
Run Decesion Tree classifier with GridSearchCV
{'F1_Score': 0.25, 'Recall': 0.2, 'Duration_sec': 0.023, 'Precision': 0.33, 'Accurac
y': 0.86}
```

These metrics are too low, so the best algorithm is Decision Tree Classifier with default parameters i.e., 50%Classfier

Conclusion

- We got good precision (50%) and Recall (60%). There is 60% of chance that algorithm marks fraudulent as POI and 50% of chance that further investigations will find POIs involvement in scandal.

Let's try to understand data behind above statement using Test data and prediction data from the result of our best classifier:

Test labels:

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0
.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0
, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]
```

Prediction:

```
[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  1.,
1.,  0.,  1.,  0.,  0.,  0.,  1.,  0.,  0.,  0.]
```

Precision = True Positives / (True Positives + False Positives)

```
Compare POIs in prediction to Test Labels, we have 3 true POIs and other 3 predict
ed POIs are incorrect.

Precision = 3 / (3 + 3) = 0.5 -> 50%

So only 50% of the identified POIs are true POIs.

Recall = True Positives / (True Positives + False Negatives)
look 2 POIs are predicted as non-POIs, so these are False negatives

Recall = 3 / (3 + 2) = 0.6 -> 60%

So, 60% of probability that POI will be identified.
```

- We have run many algorithms to validate supervised learning, but we did that without any target on metrics, Ex: we did not target 99% precision as Google might target in self driving car algorithms. We just did cross validation and chosen better one. My idea is, we should be continuously evaluating algorithms and adjusting as we get more data and more investigations happen on POIs.
- If employee not marked as POI in my classifier, I will always use another classifier with 50 percentile features to train and predict. The reason is that, employees to be predicted might have little or no communication emails with already marked POIs, but they might have emails with another set of fraudulent employees which are not part of our trained data yet.