# Machine Learning on Enron Data

## Project Overview

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, I will use these email and financial data to identify persons of interests in the Enron fraud case. Persons of interests (POIs) are defined as individuals who were indicted, reached a settlement of plea deal with the government, or testified in exchange for prosecution immunity. The goal of this project is to build a POI identifier using machine learning skills. Machine learning is a powerful data mining technique, allowing us to understand potential patterns of a big dataset. This report will cover the following sessions:

1. Enron dataset
2. Feature selection
3. Algorithm selection and tuning
4. Validation and evaluation

## Enron Data

There are a total of 146 persons and 21 features in this dataset. Among the persons, there are 18 POIs. It is noted that there are some missing values, marked as NaN. Excluding these missing values will be the first step in the following analyses.

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

print "Total number of data points (people):", len(data_dict.keys())
print data_dict.keys()
```

```
Total number of data points (people):  146
```

```
['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’ …
```

```
print "Number of features:", (len(data_dict['METTS MARK']))
print data_dict['METTS MARK']
```

```
Number of features:  21

{'salary': 365788, 'to_messages': 807, 'deferral_payments': 'NaN',
'total_payments': 1061827, 'exercised_stock_options': 'NaN', 'bonus':
600000, 'restricted_stock': 585062, 'shared_receipt_with_poi': 702,
'restricted_stock_deferred': 'NaN', 'total_stock_value': 585062,
'expenses': 94299, 'loan_advances': 'NaN', 'from_messages': 29,
'other': 1740, 'from_this_person_to_poi': 1, 'poi': False,
'director_fees': 'NaN', 'deferred_income': 'NaN',
'long_term_incentive': 'NaN', 'email_address': 'mark.metts@enron.com',
'from_poi_to_this_person': 38}
```

```
# count POIs
i = 0
for key in data_dict:
    if data_dict[key]['poi'] == True:
        i = i + 1
        # print key
print "Number of POIs:", i
```

```
Number of POIs:  18
```

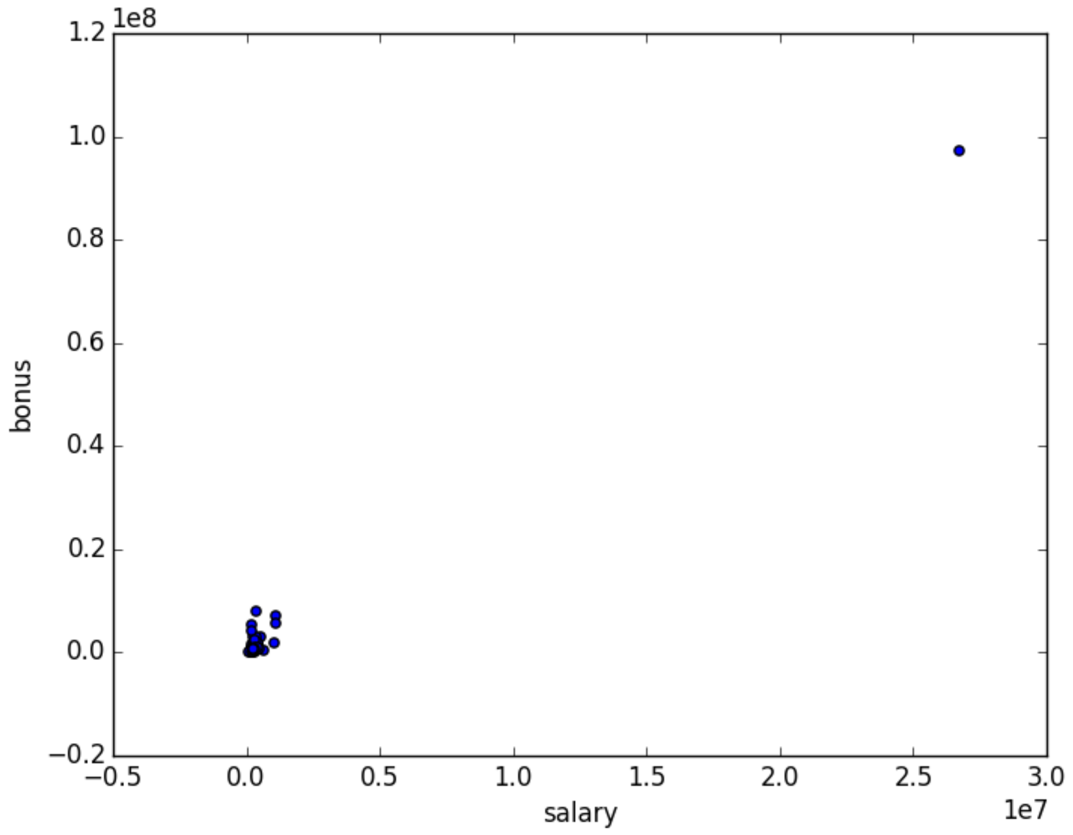
First, I would like to take a look at the financial data. For the financial data, NaN values actually represent zeros from the financial data table. They were converted as such in the featureFormat() function. To explore whether there are any outliers, I started by plotting the salary and bonus features:

```
### plot features

features = ["salary", "bonus"]
data = featureFormat(data_dict, features)

for point in data:
    salary = point[0]
    bonus = point[1]
    plt.scatter(salary, bonus)

plt.xlabel("salary")
plt.ylabel("bonus")
plt.show()
```



From the plot, one point far away looks like an outlier. To check the outliers, I printed out the top 10 salary data points:

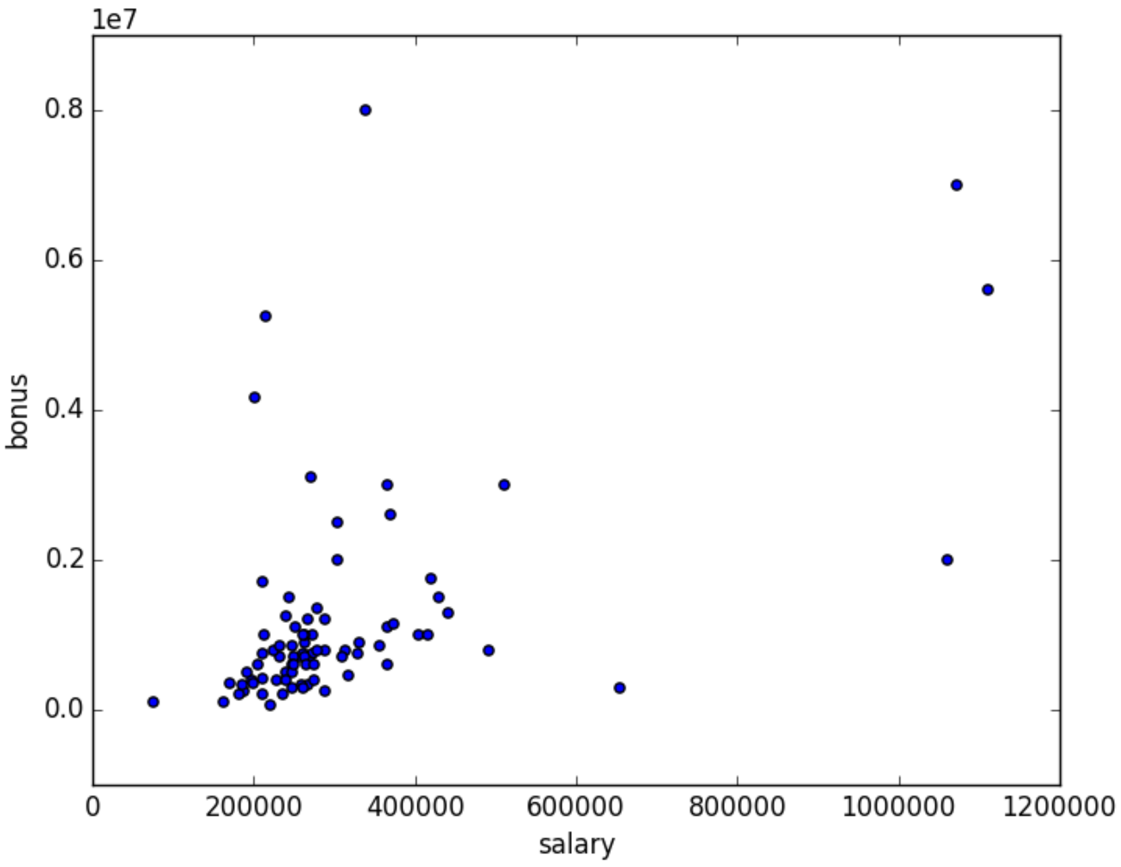
```
### check outliers

top10 = []
for key in data_dict:
    if data_dict[key]['salary'] != 'NaN':
        top10.append((key,(data_dict[key]['salary'])))

print "Top 10 salary:", (sorted (top10, key = lambda x:x[1], reverse = True)[:10])
```

```
Top 10 salary: [('TOTAL', 26704229), ('SKILLING JEFFREY K', 1111258),
('LAY KENNETH L', 1072321), ('FREVERT MARK A', 1060932), ('PICKERING
MARK R', 655037), ('WHALLEY LAWRENCE G', 510364), ('DERRICK JR. JAMES
V', 492375), ('FASTOW ANDREW S', 440698), ('SHERRIFF JOHN R', 428780),
('RICE KENNETH D', 420636)]
```

The highest salary point is ‘TOTAL’, which should be removed. Even though the salary of SKILLING JEFFREY K, LAY KENNETH L and FREVERT MARK A are also very high, they are definitely the POIs I would like to keep in the dataset. After excluding the 'TOTAL' outlier, the plot looks better:



Using the similar strategy, I also checked the rest of financial features, including 'total_payments', 'total_stock_value' and so on. LAY KENNETH L and SKILLING JEFFREY K look like two outliers in some of the plots. However, they should stay in the dataset, since they are two obvious POIs.

## Feature Selection

I created two new features: 'from_poi_ratio' and 'to_poi_ratio' in the email data. Replacing the original features by these newly created ratios will help to reduce the number of features in email data.

```
for key in data_dict:
    if data_dict[key]['from_poi_to_this_person'] != 'NaN' and data_dict[key]['from_messages'] != 'NaN':
        from_ratio = float (data_dict[key]['from_poi_to_this_person'])/float(data_dict[key]['from_messages'])
    else:
        from_ratio = float(0.)
    data_dict[key]['from_poi_ratio'] = from_ratio


for key in data_dict:
    if data_dict[key]['from_this_person_to_poi'] != 'NaN' and data_dict[key]['to_messages'] != 'NaN':
        to_ratio = float(data_dict[key]['from_this_person_to_poi'])/float(data_dict[key]['to_messages'])
    else:
        to_ratio = float(0.)
    data_dict[key]['to_poi_ratio'] = to_ratio
```

Feature selection was deployed using Decision Tree Classifier with 10-fold cross-validation. Firstly, the feature list contained all 14 financial features and 3 email features (i.e. 'from_poi_ratio', 'to_poi_ratio', 'share_receipt_with_poi'). The sum feature scores from 10-fold cross-validation were calculated.

```
### feature selection using Decision Tree Classifier with KFold validation
from sklearn.cross_validation import KFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

acc = []
precision = []
recall = []
feature_scores = []

t0 = time()
k = KFold(len(labels), 10)
for train_index, test_index in k:
    features_train = [features[ii] for ii in train_index]
    features_test= [features[ii] for ii in test_index]
    labels_train = [labels[ii] for ii in train_index]
    labels_test= [labels[ii] for ii in test_index]

    DTclf = DecisionTreeClassifier(random_state = 11)
    DTclf.fit(features_train, labels_train)
    pred = DTclf.predict(features_test)
    acc.append(accuracy_score(labels_test, pred))
    precision.append(precision_score(labels_test, pred))
    recall.append(recall_score(labels_test, pred))
    feature_scores.append(DTclf.feature_importances_)

print "Time:", round(time()-t0, 3), "s"
print "Accuracy:", acc, np.mean(acc)
print "Precision:", precision, np.mean(precision)
print "Recall:", recall, np.mean(recall)

sum_feature_scores = [sum(x) for x in zip(*feature_scores)]
print "10-fold sum feature scores:", sum_feature_scores
for i in range (len(features_list)-1):
    print (i+1, features_list[i+1], sum_feature_scores[i])
```

```
(1, 'salary', 0.36728608908510579)
(2, 'deferral_payments', 0.20965623256098082)
(3, 'expenses', 1.541280470710239)
(4, 'deferred_income', 0.72946514445348498)
(5, 'long_term_incentive', 0.28616799188236075)
(6, 'restricted_stock_deferred', 0.047514619883040933)
(7, 'loan_advances', 0.0)
(8, 'director_fees', 0.0)
(9, 'bonus', 1.6760131774587923)
(10, 'other', 0.59543989946874643)
(11, 'total_stock_value', 0.27783075673694224)
(12, 'restricted_stock', 0.98469053351129443)
(13, 'total_payments', 0.38752528297454669)
(14, 'exercised_stock_options', 1.3539454859545119)
(15, 'shared_receipt_with_poi', 0.33809083747559593)
(16, 'to_poi_ratio', 0.99056336590612526)
(17, 'from_poi_ratio', 0.21453011193823246)
```

Therefore, I selected the features with their sum feature scores greater than 0.7, including **'bonus', 'expenses', 'exercised_stock_options', 'to_poi_ratio', 'restricted_stock', ‘deferred_income'.**

In this dataset, the allocation of labels (i.e. POI/non-POI) is not balanced. According to documentations and discussions on machine learning classifiers, 2-dimensional classifiers such as SVM and K-means may generate tricky results for the imbalanced dataset. Since Decision Tree classifier was deployed here, feature scaling was not applicable for this method.

## Algorithm

With the 6 chosen features, besides Decision Tree Classifier I used for feature selection, I also tried AdaBoost and GaussianNB classifiers.

```
### Try out the Adaboost Classifier
from sklearn.ensemble import AdaBoostClassifier

acc = []
precision = []
recall = []

t0 = time()
k = KFold(len(labels), 10)
for train_index, test_index in k:
    features_train = [features[ii] for ii in train_index]
    features_test= [features[ii] for ii in test_index]
    labels_train = [labels[ii] for ii in train_index]
    labels_test= [labels[ii] for ii in test_index]

    ABclf = AdaBoostClassifier(n_estimators = 100, random_state = 11)
    ABclf.fit(features_train, labels_train)
    pred = ABclf.predict(features_test)
    acc.append(accuracy_score(labels_test, pred))
    precision.append(precision_score(labels_test, pred))
    recall.append(recall_score(labels_test, pred))

print "Time for Adaboost:", round(time()-t0, 3), "s"
print "Accuracy for Adaboost:", acc, np.mean(acc)
print "Precision for Adaboost:", precision, np.mean(precision)
print "Recall for Adaboost:", recall, np.mean(recall)


### Try out the GaussianNB
from sklearn.naive_bayes import GaussianNB

acc = []
precision = []
recall = []

t0 = time()
k = KFold(len(labels), 10)
for train_index, test_index in k:
    features_train = [features[ii] for ii in train_index]
    features_test= [features[ii] for ii in test_index]
    labels_train = [labels[ii] for ii in train_index]
    labels_test= [labels[ii] for ii in test_index]

    NBclf = GaussianNB()
    NBclf. fit(features_train, labels_train)
    pred = NBclf.predict(features_test)
    acc.append(accuracy_score(labels_test, pred))
    precision.append(precision_score(labels_test, pred))
    recall.append(recall_score(labels_test, pred))

print "Time for GaussianNB:", round(time()-t0, 3), "s"
print "Accuracy for GaussianNB:", acc, np.mean(acc)
print "Precision for GaussianNB:", precision, np.mean(precision)
print "Recall for GaussianNB:", recall, np.mean(recall)
```

Parameter tuning in a machine learning algorithm is to find out the optimal values that enable the algorithm to complete a learning task in the best possible way. Tuning parameter is important because the values of each parameter can affect how good the algorithm ‘learn from’ the training variable data.

In order to figure out the best parameters for the algorithms, GridSearchCV was used to tune parameters in Decision Tree and AdaBoost with 5-fold cross-validation. Scoring metric was changed to precision and recall in GridSearchCV.

```
from sklearn.grid_search import GridSearchCV

### Tune Decision Tree Classifier
parameters = {"min_samples_split": [2, 3, 4, 5, 6, 7, 8],
              "criterion": ["gini", "entropy"],
              "splitter": ["best", "random"],
              }
scores = ['precision', 'recall']

for score in scores:
    print ("Tuning hyper-parameters for", score)
    tree = DecisionTreeClassifier(random_state = 11, max_features = "auto", class_weight = "auto", max_depth = None)
    DTclf = GridSearchCV(tree, param_grid = parameters, cv=5, scoring = score)
    DTclf.fit(features, labels)
    print "The best parameters for decision tree:"
    print (DTclf.best_params_)


### Tune Adaboost Classifier
parameters = {"n_estimators": [1, 10, 50, 100, 200],
              }
scores = ['precision', 'recall']
for score in scores:
    print ("Tuning hyper-parameters for", score)
    tree = DecisionTreeClassifier(random_state = 11, max_features = "auto", class_weight = "auto", max_depth = None)
    Adaboost_tuned = AdaBoostClassifier(base_estimator = tree, random_state = 11)
    ABclf = GridSearchCV(Adaboost_tuned, param_grid = parameters, cv=5, scoring = score)
    ABclf.fit(features, labels)
    print "The best parameters for Adaboost:"
    print (ABclf.best_params_)
```

```
('Tuning hyper-parameters for', 'precision')
The best parameters for decision tree:
{'min_samples_split': 3, 'splitter': 'random', 'criterion': ‘gini'}

('Tuning hyper-parameters for', 'recall')
The best parameters for decision tree:
{'min_samples_split': 7, 'splitter': 'random', 'criterion': ‘entropy'}

('Tuning hyper-parameters for', 'precision')
The best parameters for Adaboost:
{'n_estimators': 1}

('Tuning hyper-parameters for', 'recall')
The best parameters for Adaboost:
{'n_estimators': 1}
```

Since ’n_estimators’ equals to 1 in Adaboost, there is no benefit of using this ensemble algorithm. Thus, I chose Decision Tree classifier as the final algorithm.  Because of the imbalanced of POI/non-POI in this dataset, StratifiedShuffleSplit was used with 5 iterations to split the data into training and test sets. Averaged accuracy, precision and recall values were calculated.

```
from sklearn.cross_validation import StratifiedShuffleSplit
sss = StratifiedShuffleSplit(labels, n_iter = 5, test_size = 0.1, random_state = 0)

acc = []
precision = []
recall = []

t0 = time()
for train_index, test_index in sss:
    features_train = [features[ii] for ii in train_index]
    features_test= [features[ii] for ii in test_index]
    labels_train = [labels[ii] for ii in train_index]
    labels_test= [labels[ii] for ii in test_index]

    ### Decision Tree Classifier
    clf = DecisionTreeClassifier(random_state = 11, max_features = "auto",
                                 class_weight = "auto", max_depth = None, min_samples_split = 3,
                                 splitter = 'random', criterion = 'gini')

    clf.fit(features_train, labels_train)
    pred = clf.predict(features_test)
    acc.append(accuracy_score(labels_test, pred))
    precision.append(precision_score(labels_test, pred))
    recall.append(recall_score(labels_test, pred))

print "Final model - Accuracy:", acc, np.mean(acc)
print "Final model - Precision:", precision, np.mean(precision)
print "Final model - Recall:", recall, np.mean(recall)
```

```
Final model - Accuracy:
[0.7857142857142857, 0.8571428571428571, 0.7857142857142857,
0.7857142857142857, 0.9285714285714286]
0.828571428571

Final model - Precision:
[0.33333333333333331, 0.5, 0.33333333333333331, 0.40000000000000002,
1.0]
0.513333333333

Final model - Recall:
[0.5, 0.5, 0.5, 1.0, 0.5]
0.6
```

## Validation and Evaluation

In feature selection, 10-fold cross-validation was used in splitting the data into training set and test set.

In parameter tuning, 5-fold cross-validation was set in GridSearchCV.

In the final algorithm, StratifiedShuffleSplit was used with 5 iterations to split the data into training and test sets. StratifiedShuffleSplit method was use, with holding up 10% data with balanced classes (POI/non-POI) in the whole dataset as a test set. Having a test set is important to valid whether the classifier works in the real data. If all data points were used as training data, the algorithm would be too specific and hard to generalize to a new data point.

Accuracy, precision and recall scores were used as evaluation metrics.

```
Final model - Accuracy:
[0.7857142857142857, 0.8571428571428571, 0.7857142857142857,
0.7857142857142857, 0.9285714285714286]
0.828571428571

Final model - Precision:
[0.33333333333333331, 0.5, 0.33333333333333331, 0.40000000000000002,
1.0]
0.513333333333

Final model - Recall:
[0.5, 0.5, 0.5, 1.0, 0.5]
0.6
```

Given that the number of non-POIs is much larger than the number of POIs in this dataset, accuracy is a sub-optimal evaluation metric because that it is easily to get all non-POIs correctly classified. In this case, precision and recall are better metrics to evaluate how the classifier performs.

Precision in this dataset is the number of true POIs (who is POI and also classified as a POI) divided by the total number of individuals classified as POIs. It can be understood as the likelihood (0.513) of a person identified as a POI is actually a true POI.

Recall in this dataset is the number of true POIs (who is POI and also classified as a POI) divided by the total number of individuals who are POIs (i.e. there are 18 POIs in this case). It can be understood as how likely (0.60) a person will be flagged as a POI.

Both numbers are around 50-60%, means there are still about 50% classifications were incorrect. One possible way to increase both numbers is to further dig into the features. From the process, I noticed that feature selection is the most important step in the whole analyses, especially when the allocation of labels is imbalanced.