# Identify Fraud From Enron Email

## Project Overview

The Enron Corporation was an American energy trading company, and it was one of the biggest companies in the US during the year 2000. The company declared bankruptcy in late 2001 and there were people who were arrested due to corporate fraud and corruption.

In this project, I will identify the persons of interest in the fraud case by constructing a POI identifier based on the public Enron financial and email dataset. As to what classifies as a person of interest, it is someone who was indicted, settled without admitting being guilty, or testified in exchange for immunity.

# Questions

**1. Summarize for us the goal of this project and how machine learning is useful in trying to accomplish it. As part of your answer, give some background on the dataset and how it can be used to answer the project question. Were there any outliers in the data when you got it, and how did you handle those? [relevant rubric items: “data exploration”, “outlier investigation”]**

The goal of this project is to use machine leaning tools to identify persons of interest involved in the Enron fraud. Since machine learning has the capability to be trained and make predictions, we were able to achieve this task.

The Enron dataset consists of 146 data points with 21 features. Basically, there are 146 total number of people in the dataset, with each person having 21 features. There are 18 people identified as persons of interest and 128 people who are not.

**Enron Dataset:**

- Total count of people in the data set: 146
- Total count of persons of interest (POI) in the data set : 18
- Total count of non persons of interest (NON POI) in the data set : 128
- Each person has 21 features and are listed below :

Total number of missing values for each feature:

- ('salary', 51)
- ('to_messages', 60)
- ('deferral_payments', 107)
- ('total_payments', 21)
- ('long_term_incentive', 80)
- ('loan_advances', 142)
- ('bonus', 64)
- ('restricted_stock', 36)
- ('restricted_stock_deferred', 128)
- ('total_stock_value', 20)
- ('shared_receipt_with_poi', 60)
- ('from_poi_to_this_person', 60)
- ('exercised_stock_options', 44)
- ('from_messages', 60)
- ('other', 53)
- ('from_this_person_to_poi', 60)
- ('deferred_income', 97)
- ('expenses', 51)
- ('email_address', 35)
- ('director_fees', 129)

There were missing values in the dataset. The total number of missing values in features is 1358, with each feature having at least 1 NaN value.

**Outliers**

I looked up the relationship between the features ‘bonus’ and ‘salary’ using a scatterplot, and I found an outlier. The outlier in the scatterplot was TOTAL. I removed the TOTAL outlier since it was irrelevant. Additionally, I removed the datapoint called “THE TRAVEL AGENCY IN THE PARK” since it did not relate to the Enron employees.

Looking at the scatterplot, it is worth noting that there were still some outliers that appear in the plot after removing the TOTAL outlier. There was no need to remove those outliers since it was not as significant as the TOTAL outlier.

**2. What features did you end up using in your POI identifier, and what selection process did you use to pick them? Did you have to do any scaling? Why or why not? As part of the assignment, you should attempt to engineer your own feature that does not come ready-made in the dataset -- explain what feature you tried to make, and the rationale behind it. (You do not necessarily have to use it in the final analysis, only engineer and test it.) In your feature selection step, if you used an algorithm like a decision tree, please also give the feature importances of the features that you use, and if you used an automated feature selection function like SelectKBest, please report the feature scores and reasons for your choice of parameter values. [relevant rubric items: "create new features", "intelligently select features", "properly scale features"]**

Before adding in my new features, I decided to test 4 algorithms, which are Naïve Bayes, Decision Tree, Support Vector Machine, and Random Forest.

| Classifier | Accuracy | Precision | Recall |
|---|---|---|---|
| Naive Bayes | 0.33 | 1.00 | 0.09 |
| Decision Tree | 0.80 | 0.00 | 0.00 |
| Support Vector Machine | 0.93 | 0.00 | 0.00 |
| Random Forest | 0.93 | 0.00 | 0.00 |

All algorithms except for Naïve Bayes have a decent accuracy. Also, Naive Bayes is the only algorithm that does not have a value of 0.0 for both precision and recall scores.

**New Features**

I created two new features, which are **'fraction_from_this_person_to_poi'** and **'fraction_from_poi_to_this_person.'**

I created these features to get an understanding of the possible persons of interest knowing that a person of interest is more likely to send or receive emails to or from other persons of interest.

I Implemented these features by looping through the Enron dataset and made the following calculation:

**fraction_from_this_person_to_poi** = from_this_person_to_poi / from_messages
**fraction_from_poi_to_this_person** = from_poi_to_this_person / to_messsages

I decided to test the same 4 algorithms but with the addition of two new features.

| Classifier | Accuracy | Precision | Recall |
|---|---|---|---|
| Naive Bayes | 0.58 | 0.50 | 0.08 |
| Decision Tree | 0.86 | 0.00 | 0.00 |
| Support Vector Machine | 0.93 | 0.00 | 0.00 |
| Random Forest | 0.93 | 0.00 | 0.00 |

As we can see, all of the algorithms' accuracies either increased or stayed the same, but the precision and recall scores of SVC did not change, as it stayed 0.0. We can also say the same thing to Random Forest and Decision Tree, as both of their precision and recall scores were 0.0.

**Univariate Feature Selection**

I decided to take a look at my top performing features according to their F-score using Naïve Bayes.

- ('poi', 1.0, 1.0, 1.0)
- ('total_stock_value', 0.6439909297052154, 0.284, 0.3941707147814018)
- ('exercised_stock_options', 0.5140271493212669, 0.284, 0.36586151368760067)
- ('bonus', 0.5256257449344458, 0.2205, 0.3106727721028531)
- ('deferred_income', 0.5074626865671642, 0.221, 0.30790665273423895)
- ('long_term_incentive', 0.5569823434991974, 0.1735, 0.2645825390773923)
- ('restricted_stock_deferred', 0.15118300703000984, 1.0, 0.26265677326154047)
- ('director_fees', 0.14955507365587378, 1.0, 0.26019644831848043)
- ('fraction_from_this_person_to_poi', 0.27346570397111913, 0.1515, 0.19498069498069498)
- ('salary', 0.5217391304347826, 0.114, 0.1871153057037341)
- ('restricted_stock', 0.5193621867881549, 0.114, 0.18696186961869618)
- ('total_payments', 0.43761996161228406, 0.114, 0.1808806029353431)
- ('loan_advances', 0.5, 0.0515, 0.09338168631006347)
- ('other', 0.25495049504950495, 0.0515, 0.08569051580698835)
- ('from_this_person_to_poi', 0.176, 0.055, 0.0838095238095238)
- ('shared_receipt_with_poi', 0.17457627118644067, 0.0515, 0.07953667953667953)
- ('deferral_payments', 0, 0, 0)
- ('expenses', 0, 0, 0)
- ('to_messages', 0, 0, 0)
- ('from_poi_to_this_person', 0, 0, 0)
- ('from_messages', 0, 0, 0)
- ('fraction_from_poi_to_this_person', 0, 0, 0)

Most of the features are performing well. However, we have to take into account that the features 'restricted_stock_deferred' and 'director_fees' have a lot of missing values. Therefore, I did not include those two features in my final features_list.

I ended up using the following features in my features_list:

poi, total_stock_value, exercised_stock_options, bonus, deferred_income, long_term_incentive, restricted_stock, salary, total_payments, other,
shared_receipt_with_poi.

When I used my newly created feature ‘fraction_from_this_person_to_poi’ onto my final classifier, I did not see any improvement. As a matter of fact, it lowered both the precision and recall scores. For this reason, I decided to not include any of my newly created feature in my final feature_list.

I tested the same 4 algorithms with my final set of features.

| Classifier | Accuracy | Precision | Recall |
|---|---|---|---|
| Naive Bayes | 0.93 | 1.00 | 0.50 |
| Decision Tree | 0.73 | 0.00 | 0.00 |
| Support Vector Machine | 0.93 | 0.00 | 0.00 |
| Random Forest | 0.93 | 1.00 | 0.50 |

The accuracy for Naïve Bayes increased significantly from 58 to 93. The rest of algorithms’ accuracies also increased except for Decision Tree.

**3. What algorithm did you end up using? What other one(s) did you try? How did model performance differ between algorithms? [relevant rubric item: “pick an algorithm”]**

For this project, I tested Naïve Bayes, Decision Tree, SVC and Random Forest using the final set of features that I chose. After testing, I found that Naïve Bayes performed better than the rest of the algorithms, considering it had the highest accuracy, precision score, and recall score. Note that I had not tuned any of the algorithms yet.

After tuning all the algorithms, Naïve Bayes still produced better results. It still had the highest accuracy, precision score, and recall score, which is why I ended up using Naïve Bayes.

**4. What does it mean to tune the parameters of an algorithm, and what can happen if you don’t do this well? How did you tune the parameters of your particular algorithm? What parameters did you tune? (Some algorithms do not have parameters that you need to tune -- if this is the case for the one you picked, identify and briefly explain how you would have done it for the model that was not your final choice or a different model that does utilize parameter tuning, e.g. a decision tree classifier). [relevant rubric items: “discuss parameter tuning”, “tune the algorithm”]**

As we saw in the algorithms we tested above, most of them have a high accuracy but all except for Naïve Bayes have precision and recall scores of zero. This means that performance measures such as those I have mentioned above are not quite as good as they could be if the model is not modified to the particular dataset's features. Therefore, it is important to do parameter tuning to make the best use of the classification models to achieve the best possible result.

Parameter tuning will help us improve all of the algorithms' precision and recall scores.

We need to be careful though, as tuning the model's parameters too much may result in overfitting.

At first, I tested different combinations of parameters for each algorithm, but then I utilized some methods from Scikit-learn.

**Naive Bayes**

One of the methods from Scikit-learn that I used was SelectKBest. Using this method in a pipeline will select the K best features. Another method I used is called Principal Component Analysis and it lowered the dimensionality of the features. I also used MinMaxScaler to scale features, which were then given to the Naïve Bayes classifier. Lastly, I used GridSearchCV and this method will pick the best value of k for SelectKBest and n_components for Principal Component Analysis. Basically, the GridSeachCV will choose the best possible parameters for Naïve Bayes.

- 'k': [8,9,10]
- 'n_components': [6,7,8]

GridSearchSV gave me the following best parameter combination for Naïve Bayes:

- 'k':10
- 'n_components':7

The result was:

| Classifier | Accuracy | Precision | Recall | F1 Score | Training Time | Predicting Time |
|---|---|---|---|---|---|---|
| Naive Bayes | 0.86 | 0.40 | 0.40 | 0.40 | 3.16 | 0.00 |

**Decision Tree**

For this classifier, I used different parameters like:

- 'min_samples_split': [2,3,4]
- 'criterion': ['gini','entropy']

GridSearchSV gave me the following best parameter combination for Decision Tree:

- 'min_samples_split': 3
- 'criterion': 'entropy'

The result was:

| Classifier | Accuracy | Precision | Recall | F1 Score | Training Time | Predicting Time |
|---|---|---|---|---|---|---|
| Decision Tree | 0.83 | 0.20 | 0.25 | 0.22 | 11.69 | 0.00 |

**Support Vector Machine**

For this classifier, MinMaxScaler was included to the pipeline and I used different parameters like:

- 'svc__kernel': ['linear','rbf']
- 'svc__C': [0.1,1,10,100,1000]
- 'svc__gamma': [1e-3,1e-4,1e-1,1,10]

GridSearchSV gave me the following best parameter combination for SVC:

- 'svc__kernel': 'linear'
- 'svc__C': 100
- 'svc__gamma': 0.001

The result was:

| Classifier | Accuracy | Precision | Recall | F1 Score | Training Time | Predicting Time |
|---|---|---|---|---|---|---|
| Support Vector Machine | 0.86 | 0.20 | 0.33 | 0.25 | 2.88 | 0.00 |

**5. What is validation, and what's a classic mistake you can make if you do it wrong? How did you validate your analysis? [relevant rubric items: "discuss validation", "validation strategy"]**

Validation is a process of evaluating a trained model with a testing dataset. The classic mistake you can make regarding validation is not splitting the dataset into training and testing set, which leads to overfitting. While the model that is overfitted will perform very well on the training data, it will fail miserably on predicting with new or unseen data. Splitting the dataset into training and testing set will not only reduce the problem of overfitting, it will also provide a means for testing the algorithm to see if we are receiving the best possible results.

In this project I decided to use the function train_test_split of sklearn.model_selection to split 20% of my dataset for testing and 80% for training. I validated my analysis by calling the test_classifier() function that uses the StratifiedShuffleSplit function of sklearn.model_selection. I also used accuracy_score, precision_score, and recall_score from sklearn.metrics.

**6. Give at least 2 evaluation metrics and your average performance for each of them. Explain an interpretation of your metrics that says something human-understandable about your algorithm's performance. [relevant rubric item: "usage of evaluation metrics"]**

Two evaluation metrics are Recall and Precision.

Recall is an evaluation metric that calculates a classifier's ability to classify true positives over all cases that are actually positive. Recall is calculated by taking the total number of True Positive and dividing it by the total number of True Positive and False Negative or (True Positive / (True Positive + False Negative)). In our case, a True Positive is a person of interest (POI) who is correctly classified as a POI and a False Negative is a POI who is not correctly classified. For this project, I got a recall score of 0.35 when using Naïve Bayes, which means 35% of POIs were identified correctly.

Precision is an evaluation metric that calculates a classifier's ability to classify true positives from all cases that are labeled as positives. Precision is calculated by taking the total number of True Positive and dividing it by the total number of True Positive and False Positive or (True Positive / (True Positive + False Positive)). In our case, a True Positive is a POI who is correctly classified as a POI and a False Positive is someone who is not a POI but is classified as a POI. For this project, I got a precision score of 0.37 when using Naïve Bayes, which means 37% of the people labeled as POI were actually POIs.

To summarize, I picked **Naïve Bayes** as my algorithm, which resulted in recall of **0.35** and precision of **0.37**.