ABOUT THIS PROJECT

Enron, one of the largest companies in the US in 2000, collapsed into bankruptcy by 2002 due to widespread corporate fraud. During the resulting Federal Investigation, confidential emails and financial data for top executives were entered into public record. The goal of this project is to leverage the available data and build a 'Person of Interest' (POI) Identifier. Basically, the existing information of executives who were (POI), and weren't (non-POI), party to fraud will be investigated using machine learning. Hence, if email and financial information is received for an executive that is not a part of this study, the Identifier will reveal if this executive is a POI.

To do this, I extract/identify useful features that represent the available data. Afterwards, I engineer the features, pick and tune a machine learning algorithm (classifier) to build the Identifier, then test and evaluate it.

CHARACTERISTICS OF THE DATA – Data Exploration

- There are 146 data points, which theoretically, should represent top Enron Executives
- 18 of these people are Persons of Interest (POI)
- Each person is associated with 21 features, which fall into three major types:
    - 14 Financial Features: ['salary', 'deferral_payments', 'total_payments', 'loan_advances', 'bonus', 'restricted_stock_deferred', 'deferred_income', 'total_stock_value', 'expenses', 'exercised_stock_options', 'other', 'long_term_incentive', 'restricted_stock', 'director_fees'] (all units are in US dollars)
    - 6 Email Features: ['to_messages', 'email_address', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi'] (units are generally number of email messages; notable exception is 'email_address', which is a string)
    - 1 POI label: ['poi'] (Boolean, represented as integers)
- Table 1 shows that most of these features have missing values.

Table 1: Percent of missing values in each feature

Feature	Num_Missing_Values	Percent_Missing (%)
loan_advances	142	97.3
director_fees	129	88.4
restricted_stock_deferred	128	87.7
deferral_payments	107	73.3
deferred_income	97	66.4
long_term_incentive	80	54.8
bonus	64	43.8
from_poi_to_this_person	60	41.1
shared_receipt_with_poi	60	41.1
from_this_person_to_poi	60	41.1
to_messages	60	41.1
from_messages	60	41.1
other	53	36.3
salary	51	34.9
expenses	51	34.9
exercised_stock_options	44	30.1
restricted_stock	36	24.7
email_address	35	24
total_payments	21	14.4
total_stock_value	20	13.7
poi	0	0

Missing values could introduce some bias to the POI Identifier. Later, the distribution of missing values across classes is discussed. This will improve understanding of the introduced bias, and its effect on the POI Identifier.

OUTLIER INVESTIGATION – Exploratory Data Analysis

A scatter plot of the features, 'salary' vs. 'bonus', shown in Figure 1, reveals the first outlier that was removed from the study. This outlier corresponds to the data point, 'TOTAL'. Visually inspecting the data source (enron61702insiderpay.pdf), I see that 'TOTAL' is the summation of inputs for the Enron Executives listed in the spreadsheet.

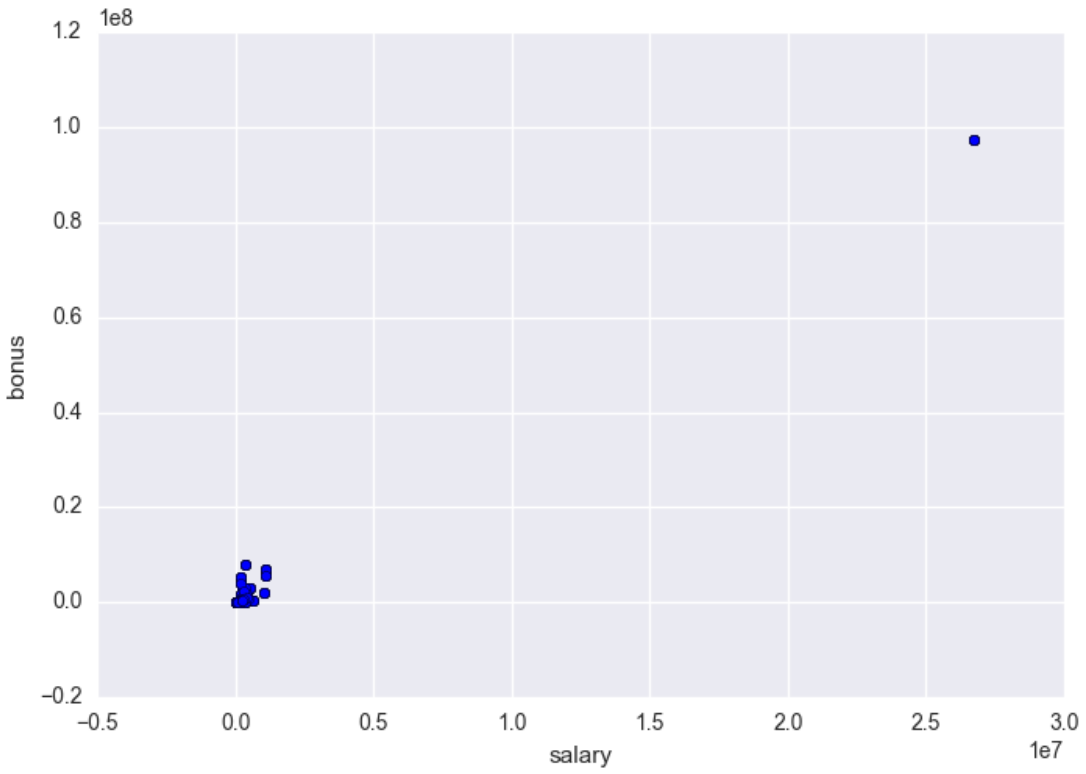

Figure 1: Bonus Vs. Salary

During the visual inspection of enron61702insiderpay.pdf, I noticed two things that resulted in the removal of two more data points:

- There is another data point grouped with 'TOTAL' called 'THE TRAVEL AGENCY IN THE PARK'. This point was removed because it does not represent a person.
- There is a lot of missing values in this (financial) data source. It is possible that there are people in this dataset with missing values for all their features. If there is also no email information for any of these people, then there is neither financial nor email information available. These people provide no information for this study and can be removed.
    - On exploring the available data sources in Python, only one person, 'LOCKHART EUGENE E', fell into this category. He was removed from the study.

Now that 'TOTAL', 'THE TRAVEL AGENCY IN THE PARK', and 'LOCKHART EUGENE E' have been removed from the study, the total number of data points is down to 143.

FEATURE ENGINEERING – Feature exclusion based on Human Intuition

The available data sources have lots of missing values (entered as 'NaN'). Missing values can be passed into classifiers, however, they can introduce some bias to the classifier. It is important to understand how the bias affects the classifier. Table 2 shows the percentage of missing values in each feature for POIs and non-POIs.

Table 2: Percent of missing data per feature

Feature	POI_Percent_Missing	Non-POI_Percent_Missing	Percent_Difference
restricted_stock_deferred	100.0	86.4	13.6
director_fees	100.0	87.2	12.8
exercised_stock_options	33.3	28.8	4.5
poi	0.0	0.0	0.0
deferral_payments	72.2	73.6	-1.4
loan_advances	94.4	98.4	-4.0
total_stock_value	0.0	14.4	-14.4
total_payments	0.0	16	-16
shared_receipt_with_poi	22.2	42.4	-20.2
to_messages	22.2	42.4	-20.2
from_messages	22.2	42.4	-20.2
from_this_person_to_poi	22.2	42.4	-20.2
from_poi_to_this_person	22.2	42.4	-20.2
restricted_stock	5.6	26.4	-20.8
long_term_incentive	33.3	57.6	-24.3
email_address	0	25.6	-25.6
deferred_income	38.9	70.4	-31.5
salary	5.6	38.4	-32.8
bonus	11.1	48	-36.9
expenses	0.0	39.2	-39.2
other	0.0	41.6	-41.6

In the dataset, there are very few POIs (18) compared to non-POIs (125 remain). Ideally, it is always better to have no missing values. However, with the available data, it is better to have a higher percentage of missing values for non-POIs entries for two reasons:

- Since there are far less data points for POIs, there are less values describing trends.
- Classifiers group data points into classes based on 'NaN' entries. For any feature, when a new data point is passed into the Identifier has an 'NaN' entry for said feature, the feature will likely suggest that the data point be grouped into the class with the higher percentage of 'NaN's. In general, since there are less POIs in the world, the probability that a new data point with lots of missing values is a non-POI is higher. Hence, when a data point has lots of missing entries, I am okay with the person being grouped into the non-POI class.
    - Because of the mentioned reasons, I excluded 'restricted_stock_deferred' and 'director_fees' from the study. Table 1 shows that these features are missing most data points.
    - A visual inspection of enron61702insiderpay.pdf reveals that 'total_stock_value' and 'total_payments' respectively depend on 'restricted_stock_deferred' and 'director_fees'. Hence, 'total_stock_value' and 'total_payments' were excluded.

Email addresses for all POI are known, but some are missing for non-POI. Missing an email should not signify that a person belongs in the non-POI class. The feature, 'email_address' was excluded from the study.

In total, 5 features will be excluded from the study based on human intuition: [‘restricted_stock_deferred’, ‘director_fees’, ‘total_payments’, ‘total_stock_value’, ‘email_address’]. This brings the total number of features down to 16.

FEATURE ENGINEERING – Feature creation based on human intuition

Lecture content introduced the idea that transforming the features, 'from_this_person_to_poi' and 'from_poi_to_this_person', could provide different insight about a person's relationship with POI. To do this, these features were converted into fractions of a total, rather than just counts. For example:

- Person A sent a total of 15 emails, and 10 of them were to POI
- Person B sent a total of 100 emails, and 20 of them were to POI

Based on count comparison, the feature 'from_this_person_to_poi' will reveal that Person B is more of a POI than Person A. However, 67% of Person A's sent emails are to POI, in comparison to 20% of Person B. Therefore, based on “fractions of a total” comparison, Person A is more of a POI.

Applying this same transformation to 'shared_receipt_with_poi', I created a third feature containing fractions of a total. This feature reveals the fraction of all received emails, for each person, that was also received by other POI. It could emphasize scenarios where information requested from Non-POI was sent to multiple POI only.

In total, 3 new features were created; 'frac_from_poi', 'frac_shared_with_poi', and 'frac_to_poi'. Both 'frac_from_poi' and 'frac_shared_with_poi' will respectively represent counts from 'from_poi_to_this_person' and 'shared_receipt_with_poi' as fractions of 'to_messages'. The new feature, 'frac_to_poi' will represent counts from 'from_this_person_to_poi' as fractions of 'from_messages'. Adding these three features brings the total number of features up to 19.

FEATURE ENGINEERING – Univariate Feature Selection

To identify the features that provide the most insight, the univariate feature selection method, SelectKBest, was deployed. Table 3 shows the score for each feature. Features with higher scores provide more insight.

Table 3: SelectKBest Feature Scores

Feature	Score
exercised_stock_options	24.53
bonus	20.52
salary	18.00
frac_to_poi	16.18
deferred_income	11.32
long_term_incentive	9.77
restricted_stock	9.08
frac_shared_with_poi	8.91
shared_receipt_with_poi	8.43
loan_advances	7.13
expenses	5.95
from_poi_to_this_person	5.14
other	4.13
frac_from_poi	3.05
from_this_person_to_poi	2.34
to_messages	1.59
deferral_payments	0.23
from_messages	0.18

"Feature" and "Information" are two different things. Features can provide access to information. However, "more features" does not mean "more information". Quality is more important than quantity. After varying the number of features plugged into selected classifiers, and viewing the effect on performance, only the 4 best features, ['exercised_stock_options', 'bonus', 'salary', 'frac_to_poi'], were selected.

Besides SelectKBest, other options for revealing the features with the most insight include:

- Deploying a tuned Decision Tree Algorithm (or any Meta Classifier) and utilizing its feature_importances attribute.
- Applying SelectPercentile. This is especially helpful when working with higher number of features (tens, hundreds, thousands).
- Carrying out feature scaling, then deploying Principal Component Analysis (PCA). Like SelectPercentile, this is helpful when working with higher number of features. The main goal here is dimensionality reduction. Features are compressed into Principal Components (PC) that model the variation/patterns in the data. The drawback here is that, because PCs are linear combinations of several input features, it may not be very easy to explain the PCs.

CROSS-VALIDATION

To estimate the performance of the POI Identifier, the available dataset is used to train and test the classifier. Stratified K-Folds cross-validator is deployed to split data into training and testing groups.

Stratified K-Folds allows every data point to be used for both training and testing. In this study, the data is split into 4 bins/folds. Each bin will get the opportunity to be the test set, while the others are training sets. The average performance of the 4 test results is returned. This cross-validator is particularly helpful for this study because there are very few POI data points, but no tradeoffs are being made by setting some data points completely for training, and others completely for testing. A higher number of training data points will typically improve the performance of a classifier.

One classic mistake that can occur with cross-validation is that either class (i.e. POI or non-POI) can be over-represented in the training or testing dataset. This is even more probable when one class has a lot more data points than another, just like in this study. It could cause the trained POI Identifier to perform poorly because it is trained with data primarily from one class, but tested with data from another class. Stratified K-Folds ensures that all bins have a similar percent of POI and non-POI data.

PICK AND TUNE CLASSIFIERS

While completing coursework, I got the opportunity to work with several supervised learning classifiers, which include: Gaussian Naïve Bayes, SVMs, Decision Trees, and Adaboost. For this project, I decided to learn how to work with K-Nearest Neighbors and Random Forest.

While deploying these classifiers, the input parameters were tuned using GridSearchCV. GridSearchCV works through multiple combinations of parameter tunes, cross-validating as it goes, to determine which settings yield the best performance. In this case, GridSearchCV is set up to choose the parameter options that maximize the F1 score. "F1 score" and the decision to choose this performance setting is explained in the Classifier Evaluation section of this document.

When parameters are not tuned well, two things can happen:

I. The classifier has high variance. This means that it is overfit to the training data, splitting training data points into classes almost perfectly. However, when deployed on training data, the algorithm performs poorly.
II. The classifier has high bias. It learns very little from training data, performing poorly. It also performs poorly on the training data.

Table 4 shows the parameters tuned (and suggested settings) applied to K-Nearest Neighbors and Random Forest. Table 5 shows the performance of both classifiers. K-Nearest Neighbors was chosen as the preferred classifier for the POI Identifier because it has the higher F1 score.

Table 4: Tuned Parameters of Deployed Classifiers

Classifier	Parameter	Settings	Best Setting (all testing bins)
K-Nearest Neighbors	n_neighbors	3, 5, 7, 9	3
	weight	uniform, distance	uniform
	algorithm	auto, ball_tree, kd_tree, brute	auto
Random Forest	n_estimators	10, 50, 100, 200	varies per bin
	criterion	entropy, gini	varies per bin
	min_samples_split	2, 5, 10	varies per bin
	max_features	auto, log2, None	varies per bin

Table 5: Performance of Deployed Classifiers

Classifier	Accuracy	Precision	Recall	F1 Score
K-Nearest Neighbors	0.8804	0.5417	0.3250	0.4028
Random Forest	0.8589	0.2750	0.3000	0.2864

CLASSIFIER EVALUATION

According to the rubric for this project, a POI Identifier that has precision and recall of 0.3 or above is adequate. This assessment is based on the Stratfied ShuffleSplit cross-validator within tester.py. To determine if the POI Identifier is adequate, the dataset, selected features, and selected classifier (K-Nearest Neighbors) was passed into tester.py.

- Testing the POI Identifier using tester.py, the results are:
    - Accuracy: 0.8690, Precision: 0.6481, Recall: 0.3250, F1: 0.4329

The performance metric, Accuracy, reveals that the POI Identifier classified 86.9% of all test data points correctly. This means that 86.9% of people were correctly identified as POI or non-POI. However, because most of the test data points are non-POI, this information reveals more about the Identifier's accuracy of non-POI predictions, than it does POI.

The Precision and Recall performance metrics reveal more about the classifier's ability to identify POI. Precision shows that 64.9% of people that the classifier grouped into the POI class where correctly identified. On the other hand, Recall shows that only 32.5% of all POI were identified. F1 score is the harmonic average of Precision and Recall.

The POI Identifier has better Precision than Recall. This means that, whenever a POI gets flagged in the test set, I am quite confident that it is likely to be a real POI, and not a false alarm. On the other hand, the price I pay for this is that the classifier sometimes misses real POIs, since it is reluctant to pull the trigger on edge cases. Looking at the input data, this makes sense. There is a lot of missing data, however, for most features, Table 2 shows that there is a higher percent of missing data within the non-POI class. Hence, when a person's entry for a feature is 'NaN', that feature will suggest that this person is non-POI. Since the entire dataset has lots of missing data, the classifier is likely to group uncertain cases as non-POI since the data point (person) probably has a few 'NaN' feature entries.