Project: Identify Fraud from Enron Email

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 the project is to identify employees from Enron who may have committed fraud based on the public Enron financial and email dataset, i.e., a person of interest. We define a person of interest (POI) as an individual who was indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

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, you will play detective, and put your new skills to use by building a person of interest identifier based on financial and email data made public as a result of the Enron scandal. The data has a 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.

This project involves muddled data so decent expertise of Data Wrangling and Data Visualization to clean and visualize the data, find correlation between various features and identify extreme outliers and finally using Machine Learning algorithm (to spot trends and) to predict Person of Interest 'POI'.

The dataset contains a total of 146 data points with 21 features, 18 POIs and 128 Non POIs.

Employees

There are 146 Enron employees in the dataset. 18 of them are POIs.

Features

There are fourteen (14) financial features. All units are US dollars.

- 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

There are six (6) email features. All units are number of emails messages, except for 'email_address', which is a text string.

- to_messages
- email_address
- from_poi_to_this_person
- from_messages
- from_this_person_to_poi
- shared_receipt_with_poi

There is one (1) other feature, which is a Boolean, indicating whether or not the employee is a person of interest.

- poi

After Exploratory Data Analysis, there were three 3 records needed removal:

- TOTAL: Contains extreme values for most numerical features.
- THE TRAVEL AGENCY IN THE PARK: Non-employee entry
- LOCKHART EUGENE E: Contains only NaN values.

Data before removal:

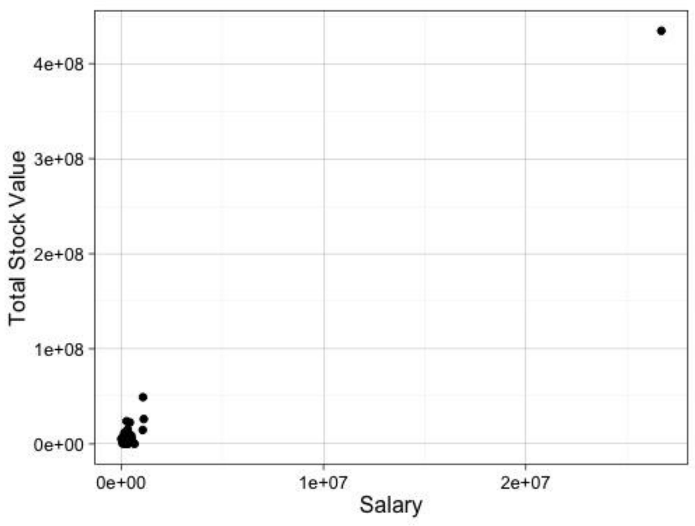

Figure 1 Total Stock Value Compared to Salary

Data after removal:

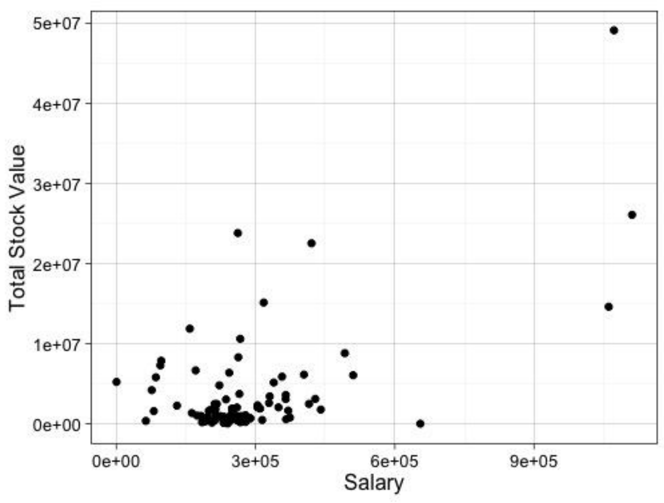

Figure 2 after Cleaning

We have some features with many missing values. Top features with missing values found by math.isnan() from python:

Feature	**NaN per feature**
Loan advances	142
Director fees	129
Restricted stock deferred	128
Deferred payment	107
Deferred income	97
Long term incentive	80

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”, “properly scale features”, “intelligently select feature”]

Along with the given 21 features, 2 additional features have been created using existing features which should help to find better results.

- *fraction_poi_communication*: Fraction of emails related to POIs. This feature helps to determine what proportion of messages are communicated between POIs compared to overall messages and helps to identify if a person has higher frequency of sending and receiving email with a POI or those who have high amount of email activity with POIs
- *total_wealth*: Salary, bonus, total stock value, exercised stock options. It helps to find the sum of all values which is related to a person’s wealth.

Using SelectKBest, the top 10 features of the dataset are:

Features	Scores
Exercised Stock Options	24.815
Total Stock Value	24.183
Bonus	20.792
Salary	18.289
Total Wealth	15.369
Deferred Income	11.458
Long Term Incentive	9.922
Restricted Stock	9.213
Total Payments	8.772
Shared Receipt with POI	8.589

The new feature **Total Wealth** appears in the top 10 best features, from which we infer that it is good to use in an algorithm to determine its performance.

Features Scaling: This is done to standardize the range of features in the data. Standardization of dataset is important for many machine learning algorithms else, they might behave irregularly if an individual features are not similar to standard normally distributed data.
Scikit-learn's, StandardScaler () function was sued to standardize the features in the dataset.

Therefore in this project the top 10 features that were selected after SelectKBest are 'poi', 'exercised_stock_options', 'total_stock_value', 'bonus', 'salary', 'total_wealth', 'deferred_income', 'long_term_incentive', 'restricted_stock', 'total_payments' and used only these features as these are the best features scored by SelectKBest. SelectKBest selects the top k features that have maximum relevance with the target variable. It takes two parameters as input arguments, "k" and the score function to rate the relevance of every feature with the target variable.

PCA (Principal Component Analysis) is used to transform input features into Principal Components (**PCs**), which is used as new features. First PCs are in directions that maximizes variance (minimizes information loss) of the data. It helps in dimensionality reduction, to find latent features, reduce noise and make algorithms to work better with fewer inputs. Maximum PCs in the data is equal to the number of features. The final dimension of PCA is determined using n_components used in scikit-learn PCA ()

A determined number of principal components were used to prevent overfitting and improve predictive performance. Pipeline fits PCA, transforms data and fits each classifier on the transformed data.

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"]**

6 different algorithms were performed using scikit-learn. The parameters were tuned using GridSearchCV.
The optimal parameters value which were achieved after performing the parameters tuning for each of the algorithm are given below.

Gaussian Naive Bayes

```
clf = GaussianNB()
```

Logistic Regression

```
number_of_best_features = 9

clf = Pipeline(steps=[
        ('scaler', StandardScaler()),
        ('pca', PCA(n_components=4, whiten=False)),
        ('classifier', LogisticRegression(tol=0.01, C=1e-08, penalty='l2',
random_state=42))])
```

tol: Tolerance for stopping criteria.

C: Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

penalty: Used to specify the norm used in the penalization. The ‘newton-cg’, ‘sag’ and ‘lbfgs’ solvers support only l2 penalties.

random_state : The seed of the pseudo random number generator to use when shuffling the data. Used only in solvers ‘sag’ and ‘liblinear’.

Support Vector Machine

```
number_of_best_features = 8

clf = Pipeline(steps=[
        ('scaler', StandardScaler()),
        ('pca', PCA(n_components=6, whiten=True)),
        ('classifier', SVC(C=1000, gamma=.001, kernel='rbf'))])
```

C : Penalty parameter C of the error term.

gamma : Kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’.

kernel : Specifies the kernel type to be used in the algorithm.

Decision Tree

```
number_of_best_features = 9

clf = Pipeline(steps=[
        ('scaler', StandardScaler()),
        ('pca', PCA(n_components=5, whiten=True)),
        ('classifier', DecisionTreeClassifier(criterion='entropy',
                                              min_samples_leaf=2,
                                              min_samples_split=2,
                                              random_state=46,
                                              max_depth=None))
```

criterion: The function to measure the quality of a split. “entropy” for the information gain.

min_samples_split: The minimum number of samples required to split an internal node:

min_samples_leaf: The minimum number of samples required to be at a leaf node:

random_state : random_state is the seed used by the random number generator

max_depth : The maximum depth of the tree.

Random Forest

```
number_of_best_features = 9

clf = Pipeline(steps=[
        ('scaler', StandardScaler()),
        ('classifier', RandomForestClassifier(max_depth=5,
                                              n_estimators=25,
                                              random_state=42))
```

n_estimators: The number of trees in the forest.

max_depth: maximum depth of the tree.

random_state : random_state is the seed used by the random number generator

Ada Boost

```
number_of_best_features = 9

clf = Pipeline(steps=[
        ('scaler', StandardScaler()),
        ('classifier', AdaBoostClassifier(learning_rate=1.5,
                                          n_estimators=30,
                                          algorithm='SAMME.R'))
    ])
```

n_estimators: The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.

learning_rate: Learning rate shrinks the contribution of each classifier by learning_rate. There is a trade-off between learning_rate and n_estimators.

algorithm: use the SAMME.R real boosting algorithm.

The uses of Principal Component Analysis (PCA) would exacerbate the performance of **Random Forest** and **Ada Boost**, so it was discarded from the algorithms selected for the evaluation.

Logistic Regression gives overall best performance using evaluation matrices. (Discussion in question #6)

Scores before adding new features:

Algorithm	Precision	Recall	F1-score
Gaussian Naive Bayes	0.383	0.317	0.347
Logistic Regression	0.41	0.448	0.428
Support Vector Machine	0.485	0.1	0.16
Decision Tree	0.21	0.19	0.20
Random Forest	0.42	0.185	0.258
Ada Boost	0.275	0.212	0.239

Scores after adding new features:

Algorithm	Precision	Recall	F1-score
Gaussian Naive Bayes	0.392	0.331	0.359
Logistic Regression	0.437	0.449	0.443
Support Vector Machine	0.490	0.151	0.230
Decision Tree	0.194	0.160	0.176
Random Forest	0.461	0.192	0.271
Ada Boost	0.316	0.230	0.266

As it is evident from the scores with new features has better performance.

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 item: "tune the algorithm"]**

Parameter tuning of an algorithm involves changing the algorithm's input parameters to a set of range and measuring the performances of each combinations in order to determine the optimal input parameters. Parameters tuning substantially improves the performance of any algorithm.

For each algorithm parameter tuning is done using GridSearchCV and StratifiedShuffleSplit where a number of operations are applied at once using Pipeline function

PCA (Principal Component Analysis) is used to transform input features into Principal Components (PCs), which is used as new features. First PCs are in directions that maximizes variance (minimizes information loss) of the data. It helps in dimensionality reduction, to find latent features, reduce noise and make algorithms to work better with fewer inputs. Maximum PCs in the data is equal to the number of features. The final dimension of PCA is determined using n_components used in scikit-learn PCA ()

Following parameters are used to tune an algorithm:

- Select K Best (SeleckKBest) : k
- Principal Components Analysis (PCA) : n_components, whiten
- Gaussian Naive Bayes (GaussianNB) : None
- Logistic Regression (LogisticRegression) : C, tol, penalty, random_state
- Support Vector Classifier(SVC) : C, gamma, kernel
- Decision Tree (DecisionTreeClassifier): min_samples_split, min_samples_leaf, criterion, max_depth, random _state.
- Random Forest (RandomForestClassifier) : n_estimators, max_depth, random_state
- Ada Boost (AdaBoostClassifier) : n_estimators, algorithm, learning_rate

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 item: "validation strategy"]**

Validation is a way to uphold a machine learning algorithm's performance, i.e., to test how well the model has been trained. It is performed to ensure that a machine learning algorithm generalizes well, and to prevent overfitting. A classic validation mistake is testing your algorithm on the same data that is was trained on. Overfitting occurs when an algorithm perform very well on the training data, but failed to perform on the testing (or, unseen) data. Without separating the training set from the testing set, it is difficult to determine how well your algorithm generalizes to new data. Therefore, it is important to split the dataset into training and testing set. Scikit-Learn's StratifiedShuffleSplit, which shuffled the data and split them into K different set, called folds. In this dataset, the data was split into 100 folds to create the training and testing data. Stratification or stratified sampling is a method of sampling from a population. In statistical surveys, when subpopulations within an overall population vary, it is advantageous to sample each subpopulation (stratum) independently. Stratification is the process of dividing members of the population into homogeneous subgroups before sampling. This often improves the representativeness of the sample by reducing sampling error. It can produce a weighted mean that has less variability than the arithmetic mean of a simple random sample of the population. It's Advantages:

- If measurements within strata have lower standard deviation, stratification gives smaller error in estimation.
- For many applications, measurements become more manageable and/or cheaper when the population is grouped into strata.
- It is often desirable to have estimates of population parameters for groups within the population.

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"]

Three evaluation matrices were used. Namely **Precision**, **Recall** and **F1-score**. Theses matrices are based on comparing the predicted values to actual ones.

- True Positive (TP): Case is positive and predicted positive.
- True Negative (TN): Case is negative and predicted negative.
- False Positive (FP): Case is negative but predicted positive.
- False Negative (FN): Case is positive but predicted negative.

Precision: Out of all items that are truly positive, how many are correctly classified as positive. It is calculated as (TP)/ (TP + FP). Here it refers to the ratio of true positive (predicted as POI) to the records that are actually POI while recall described ratio of true positives to people flagged as POI. In this case, a high precision means POIs identified by an algorithm tended to be correct.

Recall: Recall refers to the ratio of correct positive predictions made out of the actual total that were indeed positive (correct positive predictions + incorrect false negative predictions). Out of all items are predicted as positive, how many truly belong to positive cases. It is calculated as (TP)/ (TP + FN). In this case, a high recall means if there are POIs in the dataset, an algorithm has good chance to identify them.

F1-score: It is a harmonic mean of recall and precision. It is calculated as (2*recall*precision)/ (recall + precision). It reaches its best value at 1 and least value at 0.

The precision, recall and f1-score for each algorithms are given below:

Algorithm	Precision	Recall	F1-score
Gaussian Naive Bayes	0.392	0.331	0.359
Logistic Regression	0.437	0.449	0.443
Support Vector Machine	0.490	0.151	0.230
Decision Tree	0.194	0.160	0.176
Random Forest	0.461	0.192	0.271
Ada Boost	0.316	0.230	0.266

Logisitic Regression gives overall best scores for the evaluation matrices, followed by **Gaussian NB**, **Random Forest**, **AdaBoost**, **Support Vector** and **Decision Tree**.