Analyzing the Enron Email List: Identifying Fraud and Persons of Interest

Max Sydow, Nov 2019

In 2001 the Enron Corporation filed for bankruptcy after it was found that several executives were involved with fraudulent financial activities. After a federal investigation data on these executives were made public, including emails, and salary and bonus amounts. Persons of interest in the investigation, POIs, are identified in the features of data sets examined in this project.

Exploring the Data

The data set has been used widely over the years, and can be accessed via multiple ways. For this project it was preprocessed in the form of a data dictionary using Python 2.7. The raw data can be found from: https://www.cs.cmu.edu/~enron/, then downloading the May 7, 2015 version as a tgz file.

Each key-value pair in the dictionary corresponds to one person. The key is the name, while the values are features that can be analyzed via machine learning algorithms. There is a total of 146 people in this set, 18 of which are designated as POI's. There are 21 features for each person, they can be categorized as follows:

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)

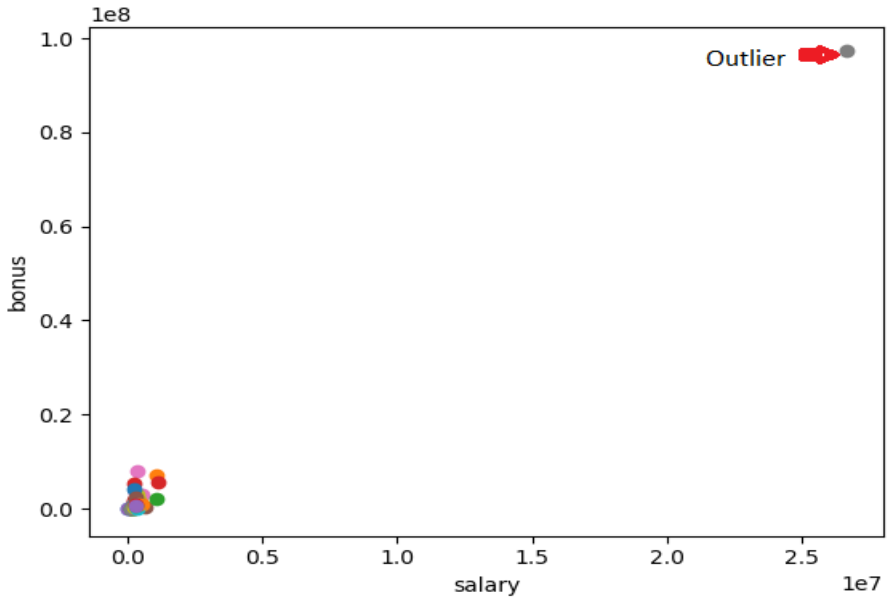
email features: ['to_messages', 'email_address', 'from_poi_to_this_person', 'from_messages', 'from_this_person_to_poi', 'shared_receipt_with_poi'] (units are generally number of emails messages; notable exception is 'email_address', which is a text string)

POI label: ['poi'] (boolean, represented as integer)

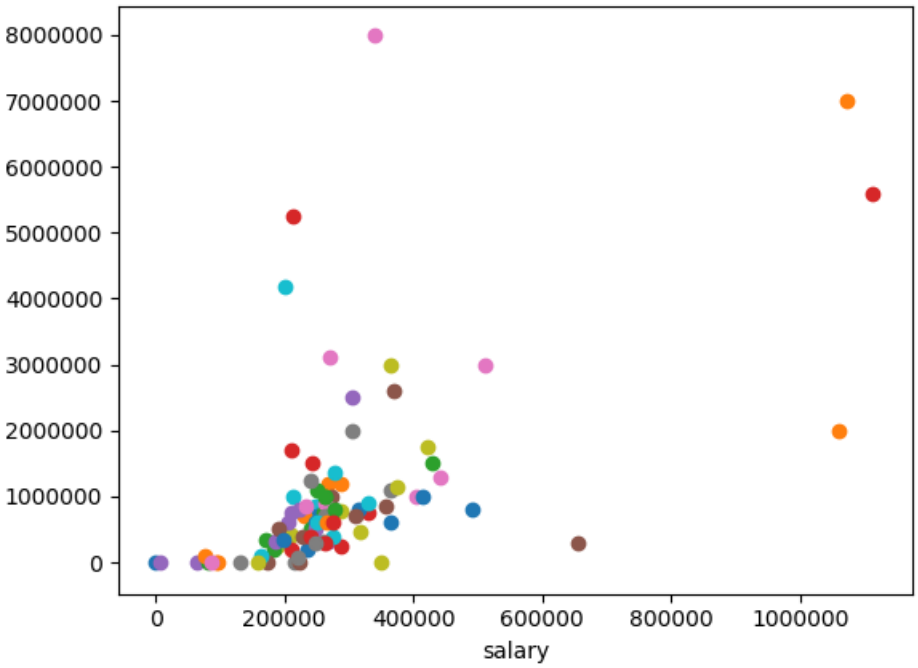
Persons of interest may email each other more frequently then others not involved with the scandal. Machine learning can be used to find evidence of correlation between POI and the features.

Reading through the dictionary keys I found one entry that doesn't look like a name. I removed 'THE TRAVEL AGENCY IN THE PARK' from this list of keys.
Plotting Bonus vs Salary, we see one outlier in the upper right corner. Running some code to loop through these 2 features reveals that the name corresponding to this point is 'TOTAL'. There's no sense in keeping this point for totals in the data set.

Bonus vs. Salary before removing outlier.



Bonus vs. Salary after removing outlier.



Also, looking at names/keys that have NaN above a threshold percentage as values it was found that ''LOCKHART EUGENE E' has all NaN for feature values. Others had over 85% NaN values, but some non-NaN values had strong K-Best-Features scores, so I decided to keep them.

Features

It stands to reason that emails sent to and from POIs may be an indicator of the person sending or receiving is also a POI. The raw number of such emails doesn't give away much without a comparison. Two new features were created for proportions of emails sent/received to/from POI to total emails sent received to/from that person: "prop_email_to_poi", and "prop_email_from_poi".

These new features were used along with the rest in the SelectKBest selection tool. The top 5 highest scoring features are listed in the table below. Of note: the "prop_email_to_poi" feature is the 5th highest scoring feature.

Feature	Score
exercised_stock_options	24.82
total_stock_value	24.18
bonus	20.79
salary	18.29
prop_email_to_poi	16.41

Algorithms

Default parameters were tried for each of the 4 algroithms used: Gaussian NB (Naïve Bayes), SVC (Support Vector Classification), Decision Tree, and AdaBoost. My 'features_used' were of the form [['poi' , '*feature*'], where '*feature*' is one of the 5 individual features above. To help choose which algorithm to focus on the accuracy was tabulated with and without scaling for each feature in each algorithm. The scaling used was the MinMaxScaler, which is the ratio of the distance of each value from the minimum to the range of values.

Feature Summary by Algorithm

Naive Bayes

Accuracy	Bonus	Exercised_stock_options	Prop_email_to_poi	Salary	Total_stock_value	Avg
Scaled	.848	.951	.852	.763	.84	.851
Not Scaled	.848	.951	.852	.763	.84	.851

SVC

Accuracy	Bonus	Exercised_stock_options	Prop_email_to_poi	Salary	Total_stock_value	Avg
Scaled	.848	.902	.852	.737	.78	.824
Not Scaled	.697	.878	.852	.737	.76	.781

Decision Tree

Accuracy	Bonus	Exercised_stock_options	Prop_email_to_poi	Salary	Total_stock_value	Avg
Scaled	.788	.854	.667	.763	.72	.758
Not Scaled	.788	.854	.667	.763	.72	.758

AdaBoost

Accuracy	Bonus	Exercised_stock_options	Prop_email_to_poi	Salary	Total_stock_value	Avg
Scaled	.788	.854	.667	.763	.72	.758
Not Scaled	.788	.854	.667	.763	.72	.758

The GaussianNB algorithm has the highest average metrics for all features used. It is clear to see that 'excercised_stock_options' has the highest accuracy feature in all algorithms used. The only algorithm affected by scaling was SVC, and I chose to focus on it for reasons that will be explained in the next section.

Parameter Tuning

SVC has more parameters to adjust than Gaussian NB and trying just a few different kernels resulted in different accuracies. This is an example of parameter tuning. Instead of running the code and noting the accuracy with each parameter adjustment, grid search can be used to optimize accuracy over a range of parameters. I'd like to see if there are other parameters for the SVC algorithm that may increase accuracy.

For this I used the following parameters:

'kernel': ('linear', 'poly', 'rbf', 'sigmoid'),

'C': [1, 10, 100, 1000, 10000],

'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1]

A better accuracy of 0.927 was obtained.

Validation

Validation refers to how accurate a model can make predictions on a testing data set based on its performance on a training set. One thing to be wary of is making sure that training and testing data sets don't contain patterns that could distinguish them. For example, if there is a pattern in the training set it would be applied to the testing set, but if no such pattern exists in the testing set then those points wouldn't be classified well. The data should be randomized to avoid this.

Overfitting is a common pitfall with validation. The extreme case is when the training and testing sets are the same – of course the same predictions will occur, but it's as if you're not really making a prediction at all. Another case is when an algorithm is too sensitive to small changes in the training set. Small changes generally should be attributed to noise, but the excess sensitivity in the model will look for similar patterns in the testing set when they probably should be ignored.

Adjusting the size of training and testing sets can lead to a balance model that avoids overfitting and maximizes performance. I started with a test size of 0.4, since that was used in the course mini projects. The accuracies for different test size proportions are summarized below:

test size	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
accuracy	.909	.905	.968	.927	.882	.852	.901	.901

Here we see that accuracy increases as testing set proportion increases, then peaks at 0.3, but starts to increase again after 0.6. This increase where over half the data set is being used for testing is where overfitting starts to take hold. Using 3/10 of the data for testing gives the best accuracy.

A better method is K-fold Cross Validation which partitions the testing and training sets into k number of folds. It is better, because it iterates over all data using different partitions of the same size for training and testing. For example, a 3-fold validation starts with 2/3 of the data used for training, and the remaining 1/3 for testing. The next iteration uses another partition of 2/3 of the data for testing and so forth for 3 iterations. Validation scores for each iteration can be found using the cross_val_score function. Using 3 folds, I obtained an average accuracy of 0.88.

Evaluation Metrics

Precision is the ratio of true positive to true positive plus false positive. In this context, it is the ratio of those that are truly POIs to the sum of true POIs and mistakenly identified POIs.

Recall is the ratio of true positive to true positive plus false negative. In other words, it is the ratio of those that are truly POIs to the sum of true POIs and POIs that were mistakenly not identified.

The SVC algorithm using grid search yielded a precision of 1.0. This means that my model did not produce any false positives in its prediction. My recall was 0.667, so the model produced some false negatives.

The following websites were used as sources:

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

https://scikit-learn.org/stable/modules/cross_validation.html

https://github.com/falodunos/intro-to-machine-learning-udacity-final-project