Identification of people possibly involved in Enron Scandal using machine learning

Project objective

"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[1]". In this project, a person of interest (POI: indicted, settled without admitting guilt, testified in exchange for immunity) identifier will be built based on publicly available financial and email data. Financial data includes features like salary, bonus and incentives of some of the Enron employees and some non-employee directors. It is well known that some of the most influential employees of the company received exorbitant salary and benefits before bankruptcy. Hence, this data may be helpful in identifying POIs. Similarly, from the email corpus we can derive the information like how many emails a person received or sent, whom these emails were sent to or received from (POI or non-POIs). This information along with the content of the emails can be used to identify the patterns that may be useful to identify POIs. This identifier may be helpful to predict whether an employee (not included in this dataset) of Enron is POI or not given his financial and email details.

Important characteristics of the dataset

code is in file 'project_code/dataset_characterstics.py':

- total number of data points: 146
  The small size of the dataset may pose significant challenges in building an efficient model.

- allocation across classes (POI/non-POI): POIs = 18, non-POIs = 128
  The skewed distribution of classes too may pose significant challenges.

- number of features:
  - financial features: 14
  - email features: 6
- number of classes: 2 (POI or non-POI)

Outlier removal: There are two prominent outliers; TOTAL and 'THE TRAVEL AGENCY IN THE PARK'. TOTAL is a spreadsheet quirk that added up all the data points (Figure 1). I do not know what 'THE TRAVEL AGENCY IN THE PARK' is, but it is definitely not a person name. Therefore, I have removed these two data points. After removing outliers, there are 144 data points, of which 18 are POIs and 126 are non-POIs. Features 'loan_advances', 'director_fees' and 'restricted_stock_deferred' have, 141, 128 and 127 missing values, respectively. It looks like certain features are not applicable to all the employees. For instance director_fees is cash payments and/or value of stock grants made in lieu of cash payments to non-employee directors, and there are only 16 non-employee directors in this data set. The

highest salary received was $1,111,258 (Skilling Jeffrey K: CEO[2]), and the highest bonus received was $8,000,000 (Lavorato John J: a top executive[3]). Although these huge figures seem to be outliers (mean salary = $284,088; mean bonus = 1,201,773), these are valid data points pertaining to Enron's most influential employees, therefor are not removed. Number of NaNs, mean, min and max of each features can be obtained by running project_code/dataset_characterstics.py.

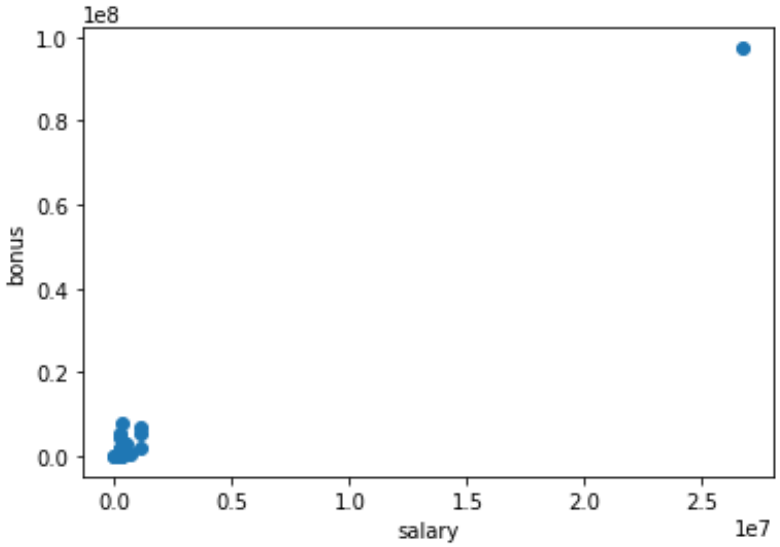

Figure 1. Scatterplot of bonus and salary showing an extreme outlier. This outlier corresponds to data point 'TOTAL', which is removed (code in 'project_code/enron_outliers.py' file).

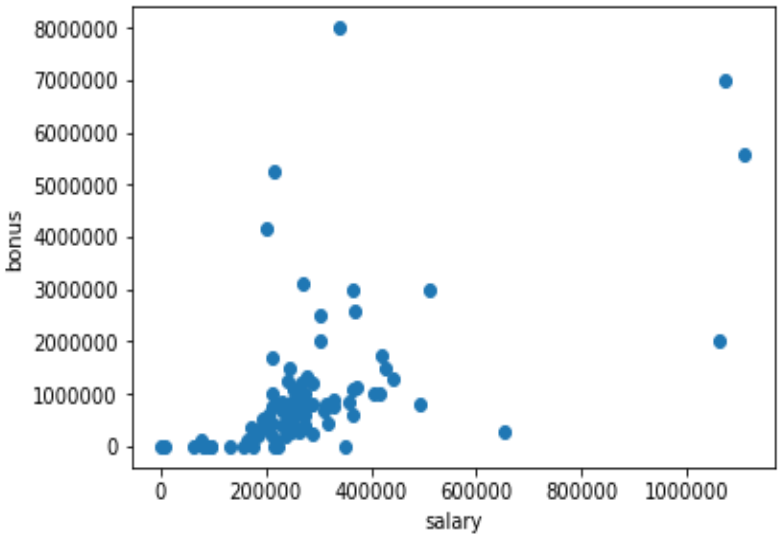

Figure 2. Scatterplot of bonus and salary after removing the data point 'TOTAL'. There seems to be a few more outliers, but these are all valid data points pertaining to Enron's most influential employees.

Feature engineering and feature selection

The following four new features have been created (code in 'project_code/poi_id.py' file).

1. **bonus_salary_ratio**
   This is bonus to salary ratio. The people involved in the scam are likely to be the most influential persons in the company, and these people are likely to have higher bonus to salary ratio in the year leading to bankruptcy. These people are likely to either determine their own benefits, or very close to decision makers.

2. **prop_from_this_person_to_poi**
   This is the proportion of the messages a person sent to POI out of total messages that the person sent. A POI is likely to send higher proportion of his/her messages to the other POIs.
3. **prop_from_poi_to_this_person**
   This is the proportion of the messages a person received from POI out of total messages that the person received. A POI is likely to receive higher proportion of messages from the other POIs.
4. **sum_prop_correspondences_with_poi**
   This is the sum of feature 2 and feature 3 above. Some POIs may write many emails to the other POIs, however, may receive a few emails from the other POIs. This could be the other way round too. The *sum_prop_correspondences_with_poi* feature accounts for such POIs.

Out of four new features, I ended up using only sum_prop_correspondences_with_poi and ratio of bonus to salary, as inclusion of other two features did not improve the model performance.

I verified the extent of influence of each feature on the target using SelectKBest function in sklearn (Table 1). I tried different numbers of top-scoring features in DecisionTreeClassifier to assess model performance. I have shown a few examples in Table 2.

Table 1. Feature scores from SelectKBest function (code in 'project_code/feature_selection_poi.py' file)

Features	Score
exercised_stock_options	19.01
total_stock_value	17.99
bonus	15.23
prop_from_this_person_to_poi	13.73
sum_prop_correspondences_with_poi	13.37
salary	12.23
deferred_income	11.89
bonus_salary_ratio	9.78

total_payments	7.81
shared_receipt_with_poi	7.38
loan_advances	7.28
long_term_incentive	7.15
restricted_stock	5.87
other	4.57
from_poi_to_this_person	4.54
expenses	4.30
from_this_person_to_poi	3.10
prop_from_poi_to_this_person	2.91
director_fees	2.14
to_messages	1.57
deferral_payments	0.19
from_messages	0.08
restricted_stock_deferred	0.06

Table 2: Performance of the DecisionTree Classifier with different combinations of top-scoring features

Features used	**Performance of the DecisionTree Classifier**
'exercised_stock_options, 'total_stock_value'	Accuracy: 0.76923 Precision: 0.23060 Recall: 0.21400 F1: 0.22199 F2: 0.21713
'exercised_stock_options', 'total_stock_value', 'bonus'	Accuracy: 0.80323 Precision: 0.37260 Recall: 0.40800 F1: 0.38950 F2: 0.40039
'exercised_stock_options', 'total_stock_value', 'bonus', 'prop_from_this_person_to_poi'	Accuracy: 0.79600 Precision: 0.32914 Recall: 0.31400 F1: 0.32139 F2: 0.31692

Note: Adding 4th top scoring feature reduced the performance (the last row in table 2).

Although the classifier derived using three top-scoring features ('exercised_stock_options', 'total_stock_value', 'bonus') achieved highest precision (0.37260) and recall (0.40800) among the combinations tried, the two newly engineered features - *sum_prop_correspondences_with_poi* and *bonus_salary_ratio* - performed better

(table 3; Precision = 0.49712, Recall = 0.43200). Therefore, I used these two features for further analysis.

It should be noted that in SelectKBest function, each feature is considered in isolation, therefore, it misses feature interaction. When there are multiple features, the relationship between each feature and the target has to be evaluated while controlling for the possible influence of the other features. That is the reason why the model derived from combination of the above two features performed better than the one with three features that scored well with SelectKBest fucntion.

Some of the other feature combinations tried are as follows. However, these combinations did not yield results better than the combination of the above two features.

1. 'sum_prop_correspondences_with_poi', 'bonus', 'total_payments'
   Accuracy: 0.79143 Precision: 0.28053 Recall: 0.29400
2. 'exercised_stock_options', 'sum_prop_correspondences_with_poi', 'total_payments'
   Accuracy: 0.80443 Precision: 0.30267 Recall: 0.28300
3. 'sum_prop_correspondences_with_poi', 'salary', 'shared_receipt_with_poi'
   Accuracy: 0.82542 Precision: 0.47217 Recall: 0.40300

I did not do any feature scaling; it is not required with decision tree because the tree structure will remain the same with or without the scaling. A node of a tree partition the data in one direction, and then partition in another direction. Therefore, we don't have to worry about what's going on in one dimension, when doing something with the other dimension[4].

Algorithm selection

Seven different algorithms have been tried and their performances are reported in Table 3.

Table 3: Model performances of different classification algorithms. Features used are sum_prop_correspondences_with_poi and bonus_salary_ratio (the code is in 'project_code/poi_id.py' file).

Algorithm	**Performance**
Naïve Bayes (GaussianNB)	Accuracy: 0.75930 Precision: 0.05077 Recall: 0.01150 F1: 0.01875 F2: 0.01360
DecisionTreeClassifier	Accuracy: 0.79900 Precision: 0.49712 Recall: 0.43200 F1: 0.46228 F2: 0.44362
LogisticRegression	Accuracy: 0.78130 Precision: 0.06912 Recall: 0.00750 F1: 0.01353 F2: 0.00913
Support Vector Machine (SVC)	Accuracy: 0.78730 Precision: 0.19324 Recall: 0.02000 F1: 0.03625 F2: 0.02437
KNeighborsClassifier	Accuracy: 0.76980 Precision: 0.28490 Recall: 0.10000 F1: 0.14804 F2: 0.11492

Ensemble. AdaBoostClassifier	Accuracy: 0.75800 F1: 0.30460 F2: 0.27954	Precision: 0.35811	Recall: 0.26500
RandomForestClassifier	Accuracy: 0.79820 F1: 0.36977 F2: 0.32167	Precision: 0.49251	Recall: 0.29600

Among the seven algorithms used, DecisionTreeClassifier achieved highest accuracy, precision and recall. Hence, this algorithm has been chosen.

Parameter tuning

Parameters are arguments passed when we create a classifier before fitting the model. What parameters are used can make a huge difference in the decision boundary that the algorithm arrives at[5]. A common problem with any predictive modelling is overfitting of the data. Overfitting happens when a model learns the noise in the training data to the extent that it negatively impacts the performance of the model on new data[6]. One of the ways we can control overfitting is through parameter tuning. If we don't do the parameter tuning well, we may end up overfitting the data[7]. Well-tuned parameters can also improve the model performance.

I used GridSearchCV to tune parameters. With tuned parameters precision increased from 0.50 to 0.61, and accuracy increased from 0.80 to 0.83. The following parameter grid is used in GridSearchCV.

```
param_grid = {"criterion": ["gini", "entropy"],
              "min_samples_split": [2, 3, 5],
              "min_samples_leaf": [1, 5],
              "max_leaf_nodes": [None, 5, 10],
              }
```

Table 4: Model performance of DecisionTree classifier without and with parameter tuning using GridSearchCV function (the code is in 'project_code/poi_id.py' file)**.**

Parameters	**Performance**		
Default	Accuracy: 0.79900 0.46228 F2: 0.44362	Precision: 0.49712	Recall: 0.43200 F1:
min_samples_leaf=5	Accuracy: 0.83090	Precision: 0.60934	Recall: 0.43050

The model performance further improved when I tuned the parameter max_depth along with min_samples_leaf (max_depth = 2, min_samples_leaf = 5; **Accuracy: 0.84380 Precision: 0.65062 Recall: 0.47300**).

Validation

Model validation is the process where a trained model is evaluated with a testing data set (a separate portion of the same data set from which the training set is derived). The main purpose

of validation is to test the generalization ability of a trained model[8]. The classic mistake we do if we do not validate properly is overfitting (see above for definition), that is, the model performs well on training data, but performs poorly on new unseen data.

The validation strategy I used was to split the data into training (67% of data points) and testing set (33%) using sklearn.cross_validation function. The model was fit using the training data, and its performance was assessed using testing set (Table 5).

KFold cross-validation strategy was also used (Table 5). In this cross-validation techniique, the original sample is randomly partitioned into *k* equal sized subsamples (I used $k = 10$). Of the *k* subsamples, a single subsample is retained as the test set, and the remaining $k - 1$ subsamples are used as training data. The cross-validation process is then repeated *k* times (the *folds*), with each of the *k* subsamples used exactly once as the test set. The *k* results can then be averaged to produce a single estimation. The advantage of this method is that all observations are used for both training and validation[9].

While performing feature selection, algorithm selection, and parameter tuning, StratifiedShuffleSplit strategy is used to evaluate the performance of the model (Table 5; code in 'project_code/tester.py'). It is a variation of ShuffleSplit, a function which generates a user defined number of independent train and test dataset splits. Samples are first shuffled and then split into a pair of train and test sets. StratifiedShuffleSplit returns stratified splits, i.e splits created by approximately preserving the same percentage for each target class as in the complete set. This method is useful when there is a large imbalance in the distribution of the target classes[10].

Evaluation metrics

Table 5. Model performance with different validation techniques (the code is in 'project_code/poi_id.py' file except for StratifiedShuffleSplit, which is in 'project_code/tester.py').

Cross Validation Function	**Model performance**		
	Accuracy	**Precision**	**Recall**
StratifiedShuffleSplit	0.84380	0.65062	0.47300
train_test_split	0.87879	0.71429	0.71429
KFold	0.90000	0.75000	1.00000
	0.70000	0.00000	0.00000
	0.70000	0.00000	0.00000
	1.00000	1.00000	1.00000
	0.70000	0.33000	0.50000
	0.90000	0.00000	0.00000
	1.00000	0.00000	0.00000
	0.70000	0.25000	1.00000

	1.00000	0.00000	0.00000
KFold Average	**0.84788**	**0.30443**	**0.42143**

Explanation of evaluation metrics (taking metrics from StratifiedShuffleSplit above as a general example)

Accuracy = (true_positives + true_negatives)/total_predictions = 0.84380
The chosen model correctly classified 84% of datapoints in test set as POIs and non-POIs.

Precision = true_positives/(true_positives+false_positives) = 0.65062
Only 65% of the people this model classified as POIs in test set are actually POIs.

Recall = true_positives/(true_positives+false_negatives) = 0.47300
Only 47% of the POIs in the test set are correctly identified as POIs by this model.

Conclusion

Given the nature of corporate governance, in corporate frauds, generally only a small set of influential individuals are involved. In such cases, predictive models built using quality data such as employee financial and email data can aid investigating agencies in identifying people who are likely involved in the scam, so that only a subset of people can be subjected to further rigorous investigation. This could save a lot of time and resources of investigating agencies as they do not have to investigate every employee of the company. Furthermore, machine learning tools may be able to identify patterns in the data, which could assist authorities in devising preventive regulatory measures.

In this project, I came up with a model to predict whether a person related to Enron is involved in the Enron fraud. However, this model needs to be further improved with advanced machine learning techniques before it could be of any practical significance. As such, this model could not identify 53% of POIs in the test set (recall = 47%), which means many POIs will escape the investigation. Furthermore, 35% of the people that this model identified as POI are not actually POIs (precision = 65%), which means, some of the innocent people would be subjected to investigation. Precision of 65% may be okay in this situation; although we are investigating 35% of the innocent people, during further investigation, anyway these people will be found innocent. However, if 53% of the people, who are guilty, escape, it is really unjust.

References

1. https://classroom.udacity.com/nanodegrees/nd002/parts/0021345409/modules/317428862475461/lessons/3174288624239847/concepts/31803986370923
2. https://en.wikipedia.org/wiki/Jeffrey_Skilling
3. http://www.nytimes.com/2002/06/18/business/officials-got-a-windfall-before-enron-s-collapse.html

4. https://classroom.udacity.com/nanodegrees/nd002/parts/0021345409/modules/317428862475460/lessons/2864738562/concepts/30244886700923#
5. https://classroom.udacity.com/nanodegrees/nd002/parts/0021345409/modules/317428862475460/lessons/2252188570/concepts/23972185420923
6. http://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/
7. https://classroom.udacity.com/nanodegrees/nd002/parts/0021345409/modules/317428862475460/lessons/2252188570/concepts/30232785960923
8. http://link.springer.com/referenceworkentry/10.1007%2F978-1-4419-9863-7_233#CR02331
9. https://en.wikipedia.org/wiki/Cross-validation_(statistics)#k-fold_cross-validation
10. http://scikit-learn.org/stable/modules/cross_validation.html#cross-validation