

Identify Fraud from Enron Data

By Abdulrahman Alemadi

GOAL OF THE PROJECT:

The goal is to build a machine learning algorithm to identify persons of interest (suspects) of the Enron scandal. The algorithm uses employees' information as features, such as salary and stock options to identify suspects. The dataset has 146 points (persons) and each point has 21 features, mostly financial features such as salary and stock options, the other features are email related such as email from/to POI. We have 18 POI's out of 146 in the dataset.

Below represents the percentage of missing values in the dataset.

1.	poi	0.000000
2.	total_stock_value	0.136986
3.	total_payments	0.143836
4.	email_address	0.239726
5.	restricted_stock	0.246575
6.	exercised_stock_options	0.301370
7.	salary	0.349315
8.	expenses	0.349315
9.	other	0.363014
10.	from_messages	0.410959
11.	from_this_person_to_poi	0.410959
12.	to_messages	0.410959
13.	shared_receipt_with_poi	0.410959
14.	from_poi_to_this_person	0.410959
15.	bonus	0.438356
16.	long_term_incentive	0.547945
17.	deferred_income	0.664384
18.	deferral_payments	0.732877
19.	restricted_stock_deferred	0.876712
20.	director_fees	0.883562
21.	loan_advances	0.972603

I decided to remove the features that have more than 50% of data missing, and for the rest of the features I have decided to impute them with the mean of the feature. There was one outlier in the dataset, it was the total amount, I simply decided to remove it.

FEATURE SELECTION & ENGINEERING

I decided to feed these features into the Principal Component Analysis function. The PCA decided the most effective features that would trigger a POI prediction, then it would combine the most effective features into 1 feature. Also, I decided not to scale the features because it has significantly decreased the AUC score of the Logistic Regression Model.

```
1. features_list = ['poi_contact_ratio1', 'poi_contact_ratio2', 'salary',  
2.                 'to_messages', 'total_payments', 'bonus',  
3.                 'total_stock_value', 'shared_receipt_with_poi',  
4.                 'exercised_stock_options', 'from_messages',  
5.                 'other', 'from_this_person_to_poi', 'expenses',  
6.                 'restricted_stock', 'from_poi_to_this_person',  
7.                 'total_worth', 'exp/net']
```

The new features I have made were the following:

- **poi_contact_ratio1**: the number of emails sent from the person to POI **divided** by the total emails sent. I made this feature because it is important to account for how frequently does the person contact a POI, out of all emails sent.
- **poi_contact_ratio2**: the number of emails received from the POI to person **divided** by the total emails received.
- **total_worth**: the **sum** of these features 'salary', 'total_payments', 'bonus', 'total_stock_value', 'exercised_stock_options', 'restricted_stock'. I made this feature because I am assuming that the POI's of the scandal has a large net worth
- **exp/net**: simply, expenses **divided** by total_worth. I picked this feature from the assumption that POI's have a large proportion of spending from their net worth.

Below is a graph that indicates the importance scores of the features, the scores were obtained by using the SelectKBest function:

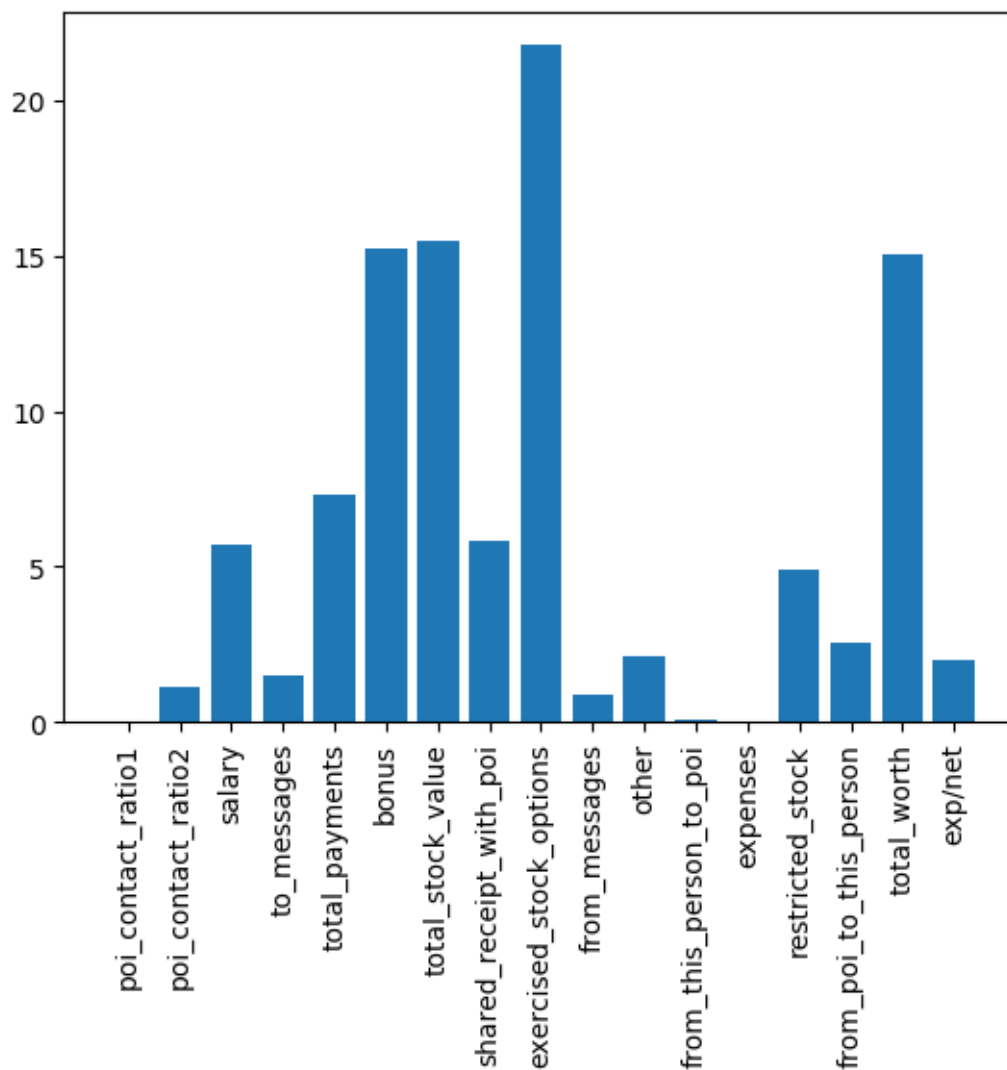


Figure 1: Feature Importance Scores

I tried to integrate SelectKBest into PCA -> LogisticRegression, but the AUC has dropped significantly to .62. Therefore, I left it out of the pipeline. I am assuming because the more features PCA has, the better.

ALGORITHM SELECTION

I decided to use logistic regression to fit and train the data. The performance metric of the models is the following:

Models	AUC	AVG F1-Score
PCA -> LogisticRegression	0.84	0.90
StandardScaler -> RandomForestClassifier	0.61	0.90
MinMaxScaler -> SelectKBest-> SVC	0.5	0.83

I had the best Area Under the Curve (AUC) results with Logistic Regression, so I have decided to use it for my final analysis.

PARAMETER TUNING

Tuning an algorithm involves trying different combination of parameters to obtain the highest yield of accuracy of the model. Tuning an algorithm is the last step before validation. Tuning is important because we want to produce a model that is fit, yet somewhat generalized to unseen data. So, it can predict most accurately throughout different datasets.

I applied GridSearchCV to fine tune my algorithm, with the following parameters:

```
1.  params = {'LR__C': np.logspace(-10, 10, 20),
2.          'LR__penalty': ['l1', 'l2'],
3.          'LR__intercept_scaling': [1, 10, 100, 1000, 10000, 1000000]}
```

The best combination of parameters is the following:

```
1.  {'LR__intercept_scaling': 100, 'LR__penalty': 'l1', 'LR__C': 0.2976351441631313}
```

MODEL VALIDATION

Validation is used to be sure that we have a good model, a model that is not prone to overfitting. If we had overfit our model, it would only have good results on the training data which is a classic mistake of validation.

To validate my model, I have utilized `tester.py`, which uses the `StratifiedShuffleSplit` function. The `StratifiedShuffleSplit` function shuffles the whole data then it splits it to training and testing subsets. Then the tester fits the training data then it predicts the outcome on the testing data. The tester iterates this process 1000 times, so it minimizes the element of chance. Finally, it aggregates the performance of each iteration to give the validation stats of the 1000 iterations.

VALIDATION METRICS

Below is the validation metrics for the testing subset.

5.	precision	recall	f1-score	support	
6.					
7.	0.0	0.97	0.90	0.94	40
8.	1.0	0.43	0.75	0.55	4
9.					
10.	avg / total	0.92	0.89	0.90	44
11.					

Precision is the score of having true positives, a low score would mean that the model has predicted many false positives (think false alarms). However, having a low recall score would mean that the model has not detected the POI's, therefore having high false negatives. In our case it would be preferable to have a higher recall score than a precision score. That way we can investigate further, instead of overlooking suspects. F1-score is the ratio of precision and recall scores, a higher f1-score simply means a better model.

When using the Tester code, I have come up with these metrics:

1.	Accuracy: 0.85080	Precision: 0.41793	Recall: 0.30300	F1: 0.35130	F2: 0.32063
2.	Total predictions: 15000	True positives: 606	False positives: 844	False negatives: 1394	True negatives: 12156