Project Description

In this project I will build an algorithm to identify “Person of Interest” among Enron Employees. POI is someone who may have committed fraud based on the public Enron financial and email dataset. Dataset is organized as a dictionary of dictionaries in which each key-value pair corresponds to one person. The dictionary key is the person's name, and the value is another dictionary, which contains the names of all the features and their values for that person. The dataset has 146 data points with 21 feature (variable) for each, including ‘poi’ feature which specifies whether the related person is a suspect of fraud or not. The ‘poi’ will be used as a label to classify our data to two classes: POI and non-POI! There is only 18 data points which are labeled as POI and the rest is non-POI.

In order to have a better sense of data and also to find outliers, I visualized data with feature_visualization[1] function which generates scatter plots for its input features. In some of the diagrams, like the scatter plot of ‘bonus’ vs ‘salary’, there is a very significant outlier for which both ‘bonus’ and ‘salary’ is at least 10 times higher than any other data point. It later became clear that the name of this data point is ‘TOTAL’ and the value for its variables is the sum of the other data points, and it should be removed from dataset. Then I checked the consistency of the key names in the data set and I found 'THE TRAVEL AGENCY IN THE PARK' as a non-person data point that should also be removed from the data set. I also got suspicious if ‘YEAP SOON’ is a real person or not; after I google it, I found that it is real! Finally, I used person_missing_values function to check the number of missing values (NaN values) for every person, and I found 'LOCKHART EUGENE E' with no data for any of its features.

Features

First of all, I removed “email_address” from the features, because it contains no useful information and it causes “could not convert string to float” error in FeatureFormat function. I also removed “shared_receipt_with_poi” because I knew from “Intro to Machine Learning” that this feature basically is encoding the ‘poi’ label for each person as a feature! Also knowing from “Intro to Machine Learning”, I added two new features: "fraction_from_poi" and "fraction_to_poi" which are basically scaled features: first one is "from_poi_to_this_person" divided by "to_messages", and the second one is "from_this_person_to_poi" divide by "from_messages". After adding and removing some features, I used SelectKBest to find out the univariate score of each feature. (Table-1)

I also used feature_missing_values function to calculate the number of missing values for each feature (Table-2) Since there is no data for "loan_advances" in 140 out of 143 data points, I removed this feature from the data set.

Before I select any group of features based on their SelectKBest scores or number of their missing values, I will test different combination of features to see which combination results in a better score. Since the number of data points is low, high number of features increases the risk of overfitting specially for Decision Tree algorithm. In order to avoid overfitting, I will test the combinations with less than 5 features for decision tree algorithm.

[1] The code of this function and all other functions that are mentioned in this document is represented in functions.py

Table-1: SelectKBest Scores

exercised_stock_options	24.8	total_stock_value	24.2	bonus	20.8
salary	18.3	fraction_to_poi	16.4	deferred_income	11.5
long_term_incentive	9.9	restricted_stock	9.2	total_payments	8.8
loan_advances	7.2	expenses	6.1	from_poi_to_this_person	5.2
other	4.2	fraction_from_poi	3.1	from_this_person_to_poi	2.4
director_fees	2.1	to_messages	1.6	deferral_payments	0.2
from_messages	0.2	restricted_stock_d	0.1		

Table-2: Number of missing values for each feature

exercised_stock_options	42	total_stock_value	18	bonus	62
salary	49	fraction_to_poi	57	deferred_income	95
long_term_incentive	78	restricted_stock	34	total_payments	20
loan_advances	140	expenses	49	from_poi_to_this_person	57
other	52	fraction_from_poi	57	from_this_person_to_poi	57
director_fees	127	to_messages	57	deferral_payments	105
from_messages	57	restricted_stock_d	126		

best_features is the function that selects the best combination of features. It works as follow:

1) Select a combination of features
2) Tune the algorithm if possible (I will describe this step in Question 4)
3) Fit the classifier Algorithm and Predict the targets
4) Find the test scores through the "**test_classifier"** function
5) If the sum of Precision and Recall is higher than the current maximum sum, update the maximum score

In Table-5 there are some of the feature combinations and their test results for Decision Tree algorithm. The features importances are also presented in the same table.
I also used **MinMaxScaler** to scale the features but it did not improve the results!

Algorithms & Modeling

Since the Enron Fraud Detection is a classification problem (not a regression), I started with two classification algorithms: Decision-Tree, and Naïve-Bayes. Then for each algorithm, and for every combination of sub-features, I calculated the evaluation metrics. In Table-5 and Table-6, you can see some of the results for each algorithm for different combinations of features.
The recall score for Naïve Bayes is no better than 0.3, which is not good! Also the sum of precision and recall is higher for Decision Tree; as a result, I chose the Decision Tree as the best algorithm for Enron fraud detection. You can see the best result for each algorithm in Table-3

Table-3: Best Results for each algorithm

Algorithm	Features	Accuracy	Precision	Recall
Decision Tree	'bonus', 'fraction_to_poi'	0.82	0.57	0.46
Naïve Bayes	'total_stock_value', 'fraction_to_poi'	0.87	0.69	0.30

Parameter Tuning

The goal of parameter tuning is to get the best performance out of the algorithm. It also can help avoiding the overfitting situation. For example, small amount of Min-Sample-Split will let the Decision Tree to go very deep and in that case it would be prone to overfitting.

Naïve Bayes does not have parameters that I need to tune. For Decision Tree, first I tried to use GridSearchCV as follow: param_grid = {'criterion': ['entropy', 'gini'], 'min_samples_split': [2,20]}

Unfortunately, the resulted scores were not satisfactory. So I incorporated the parameter tuning into the feature/algorithm selection procedure as follow:

1) Select a combination of features
2) Fit and Predict with Decision Tree with 'min_samples_split' in the range of 2 to 21
3) Test the scores with the **test_classifier** function
4) If the sum of Precision and Recall is higher than the current maximum sum, update the maximum score

Since there is only 143 data points, I chose 20 as the upper bound of the 'min_samples_split'**.** The optimum value with the best results is 13!

In Table-4, you can see some of the results for Decision-Tree with features = ['bonus', 'fraction_to_poi'] and different values of **Min-Sample-Split.**

Table-4: Tuning Decision Tree with different Min-Sample-Split values

Min-Sample-Split	**accuracy**	**precision**	**recall**
2	0.776	0.433	0.395
3	0.790	0.470	0.381
4	0.784	0.450	0.355
5	0.780	0.438	0.356
6	0.781	0.440	0.346
7	0.782	0.438	0.316
8	0.788	0.443	0.358
9	0.781	0.438	0.331
10	0.787	0.457	0.347
11	0.796	0.487	0.357
12	0.809	0.529	0.393
13	0.821	0.566	0.459
14	0.822	0.566	0.464
15	0.821	0.565	0.447
16	0.810	0.532	0.405
17	0.801	0.502	0.342
18	0.792	0.468	0.308

Validation

Validation is basically dividing the dataset into different Training and Testing groups; then fitting the algorithm with training data and evaluating its performance on the testing data. The classic mistake is to train and test the algorithm with same data, which highly increases the chance of overfitting. Since the number of data points in the Enron dataset is relatively small, by simply dividing the data to two different training and testing groups, there is still a good chance of overfitting; therefore, instead of using a simple train_test_split, it is better to use K-Fold cross validation. We can also shuffle the data points' order before splitting into folds.

Furthermore, since there is only a small number of POIs in Enron dataset, there is a huge imbalance between the size of POI and non-POI classes. In order to handle this imbalance, we can use the Stratified Cross Validation which assigns data points to folds so that each fold has approximately the same number of data points of each output class. By using **StratifiedShuffleSplit** we can address all of these concerns.

Evaluation

I used Accuracy, Precision and Recall metrics to evaluate the algorithm. Accuracy is simply the number of correctly predicted targets to the total number of targets.

Precision shows how many of all the items labeled as positive truly belong to the positive class.

Recall shows how many of all the items that are truly positive, are correctly classified as positive.

In the context of this project, for example, if our model predicts 10 POIs while only 6 of those predicted POIs are true POIs, then the precision of model would be 0.6. If there are 10 POIs in our data and the number of true POIs predicted by our model is 4, then the recall of the model would be 0.4

For the Enron dataset, Accuracy is not a significant metric. Since less than 15% of the data points are POI, if the algorithm predicts all the data point as non-POI, its accuracy would be still more than 80%. As a result, Precision and Recall are more important evaluation metrics than Accuracy.

Table-5: Decision Tree Results for different combination of features

Min-Sample-Split	**Feature combination**	**Feature importance**	**accuracy**	**precision**	**recall**
2	('exercised_stock_options', 'total_payments', 'expenses')	0.338, 0.303, 0.359	0.830	0.415	0.472
5	('bonus', 'expenses')	0.338, 0.662	0.779	0.401	0.439
7	('exercised_stock_options', 'total_payments', 'expenses')	0.608, 0.089, 0.302	0.837	0.433	0.451
10	('exercised_stock_options', 'total_payments')	0.682, 0.318	0.852	0.484	0.487
12	('bonus', 'fraction_to_poi')	0.633, 0.367	0.808	0.527	0.390
13	('bonus', 'fraction_to_poi')	0.619, 0.381	0.822	0.566	0.463

Table-6: Naïve Bayes Results for different combination of features

Feature combination	**accuracy**	**precision**	**recall**
'total_stock_value', 'fraction_to_poi'	0.869	0.691	0.300
('exercised_stock_options', 'total_stock_value', 'bonus')	0.843	0.486	0.351
('total_stock_value', 'bonus', 'restricted_stock', 'expenses')	0.859	0.511	0.342
('total_stock_value', 'bonus', 'fraction_to_poi', 'restricted_stock', 'expenses')	0.859	0.511	0.342
('exercised_stock_options', 'total_stock_value', 'bonus', 'fraction_to_poi', 'restricted_stock', 'expenses')	0.856	0.495	0.343
('exercised_stock_options', 'total_stock_value', 'bonus', 'salary', 'fraction_to_poi', 'long_term_incentive', 'expenses')	0.845	0.449	0.352