Identify Fraud from Enron Email Project

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”]

Enron Overview

Enron Corporation was an American energy, commodities, and services company based in Houston, Texas. It was founded in 1985 as the result of a merger between Houston Natural Gas and InterNorth.

Before its bankruptcy on December 2, 2001, Enron employed approximately 20,000 staff and was one of the world's major electricity, natural gas, communications and pulp and paper companies, with claimed revenues of nearly $101 billion during 2000. Fortune named Enron "America's Most Innovative Company" for six consecutive years. (Link)

This project uses machine learning skills in order to single out the POIs* who might had involved with fraud by applying various algorithms to detect important features that lead us to them.

* POI (person of interest) means an individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

Dataset Exploration

The features in the data fall into three major types, namely financial features, email features and POI labels.

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)

Some figures about the dataset:

```
Number of data points: 146
Number of POIs: 18
Number of non-POIs: 128
```

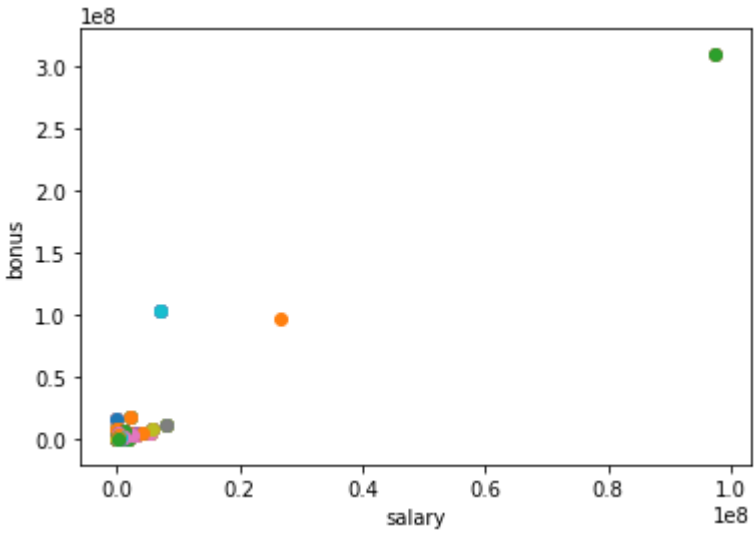
Number of NaN values in each feature:

```
salary                      51
deferral_payments          107
total_payments              21
loan_advances              142
bonus                       64
restricted_stock_deferred  128
deferred_income             97
total_stock_value           20
expenses                    51
exercised_stock_options     44
other                       53
long_term_incentive         80
restricted_stock            36
director_fees              129
to_messages                 59
email_address               34
from_poi_to_this_person     59
from_messages               59
from_this_person_to_poi     59
shared_receipt_with_poi     59
```

Many features have high number of NaNs values which could weaken the insightful reflections of them, we will not deal with them since the feature_format() method will robotically replace them with zero.

Since the vast majority of the dataset (about 87%) are labeled with non-POIs, this will spark the issue of imbalanced classes to a notable level in which classes are not represented equally. This is a challenging situation where we usually get high accuracy but later we discover they all belong to the dominated class.

In doing exploratory visualization, an extreme outlier has appeared. After digging it up, this outlier represents an erroneous one where it logically does not relate to the dataset since it is just a TOTAL of previous instances, it was then removed from the dataset. Another erroneous outlier was removed since it does not belong to an actual person (namely: THE TRAVEL AGENCY IN THE PARK). There are other outliers but can be considered as valid as well as high POIs indicators such as bonus more than 5 million for `LAY KENNETH L` and `SKILLING JEFFREY K`.



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"]

First, I have selected all features to get initial thoughts about how various algorithms would work in such imbalanced classes, then I used my sense of intuition (with support of visualizations) to select features that highly correlate with POIs. Also, I considered applying SelectKBest derived from sklearn.feature_selection, which scores the features by using a function (f_classif by default) and returns the k highest scoring features. Moreover, PCA was used it has proved by experimental to achieve higher scores since it reduces dimensionality and maximizes information variance. Pipeline & GridSearchCV were used to run up a selection of parameters and return the best of all based on f1 score.

The table below points out the values of K used and their corresponding scores:

Algorithm	Features Selected (SelectKBest)	K Best Scores:	F Score
Naïve Bayes	2	total_stock_value: 14.69 exercised_stock_options: 13.71	0.23
	4	total_stock_value: 14.69 exercised_stock_options: 13.71 salary: 11.20 bonus: 11.13	0.36
	6	total_stock_value: 14.69 exercised_stock_options: 13.71 salary: 11.20 bonus: 11.13 shared_receipt_with_poi: 5.50 deferred_income: 5.30	0.35

Then I excluded the SelectKBest, and opted to apply dimension reduction (PCA) as the table below depicts:

Algorithm	Number of PCA Components	F Score
Naïve Bayes	4 (Optimal)	0.42
	6	0.41
	8	0.41

New Feature Engineering:

A new feature was created (ratio_of_bonus_salary) to capture unusual relations for potential POIs that receive low salaries yet high bonuses. We will test the hypothesis if this new feature will reflect better in the score metrics below.

features_list=
['poi','salary','bonus','total_payments','ratio_of_bonus_salary','deferred_income','total_stock_value','exercised_stock_options','long_term_incentive','from_poi_to_this_person','from_this_person_to_poi', 'shared_receipt_with_poi',]

Algorithm	Features Selected (SelectKBest)	Scores
Naïve Bayes	features_list	F1: 0.46 | Precision: 0.47 | Recall: 0.45
	features_list + ratio_of_bonus_salary	F1: 0.43| Precision: 0.45| Recall: 0.41

Seems the new feature has generally decreased the scores, in that case we will not include it in our final analysis.

Scaling:

Feature normalization/scaling is a process that is highly depended on the chosen algorithm as well as measurement units of those features. For instance, K Nearest Neighbors and Support Vector Machines would most likely need their features to be scaled as the algorithm is driven by Euclidean distances. Whereas Decision-Tree and Random Forests do not need scaling since it depends on a threshold value and no coordinate plane is used.

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”]

I ended up using Naïve Bayes since it has proved to give the highest f1 score against Decision-Tree and Random. (The selected parameters are shown in the next page)

Algorithm	Scores
Naïve Bayes (**Final Classifier**)	F1: 0.46 | Precision: 0.47 | Recall: 0.45
Decision Tree	F1: 0.30 | Precision: 0.36 | Recall: 0.26
Random Forest	F1: 0.39 | Precision: 0.28 | Recall: 0.60

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"]

Algorithm parameter tuning is a critical stage for achieving optimal algorithm performance and to reach a sweet spot in the bias–variance tradeoff (ultimately to avoid overfitting or underfitting the model). A common strategy to tackle the parameters tuning is to use GridSearchCV under Scikit-learn package where it builds a grid of all pre-chosen parameters and runs each combination through the model, and finally chooses the most ideal/fitting parameters.

Algorithm	Parameters Tuned (Optimal in Bold)
Naïve Bayes (**Final Classifier**)	PCA Components: [**4**,6,8]
Decision Tree	criterion: ('gini', **'entropy'**) splitter: ('best', **'random'**) min_samples_split:[**2**, 10, 20] max_depth:[**10**,15,20,25,30] max_leaf_nodes:[5,10,**30**] random_state:[**11**] PCA Components: [4,**6**,8]
Random Forest	min_samples_split: [**10**, 15, 30, 40] n_estimators: [**5**, 7, 10, 20] criterion : [**'gini'**, 'entropy'] class_weight: [**'balanced'**] random_state: [15, 20, **42**] PCA Components: [4,6,**8**]

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 process to indicate how successful the predictions scores of a model against the unseen data (test data). A typical mistake can be noted when we extensively train our model on the training data and we see the accuracy score is almost 99% believing the model is ready for production. However, this might be a case of overfitting where the model cannot generalize on unseen data, and can be dealt with by using validation techniques. Another mistake can happen when we test the data on the same training data.

I have used the accompanied Stratified Shuffle Split cross validation in the tester.py which uses stratified randomized folds to create different training/testing sets, and more importantly, it takes into account the weight of sample classes (POIs vs non-POIs) and attempts to evenly distribute both of them in the training and testing data.

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"]

Since the dataset is extensively biased towards non-POIs, a regular accuracy formula would be deceiving to apply as it will overlook the minor percentage of POIs for the advantage of non-POIs. Alternatively, there are other metrics that can handle such problematic imbalanced classes named Precision and Recall.

For my best model's result by using Naïve Bayes algorithm, I have achieved the aforementioned performances: F1: 0.40174 |Precision: 0.47986 |Recall: 0.34550.

- Precision can be simply put as how frequent our class prediction (POIs versus non-POIs) is correct when we guess that class.
- On the other hand, recall is how frequent we guess the class (POIs versus non-POIs) when the class already occurred.