Introduction to Machine Learning – Analyzing Enron Fraud

Q1. 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 one of the largest companies in the United States collapsed into bankruptcy due to widespread corporate fraud in 2002. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record (available at: https://www.cs.cmu.edu/~./enron/), including tens of thousands of emails and detailed financial data for top executives. The goal of this project is to use machine learning techniques to explore and investigate the dataset and identify persons of interest based on the email exchanges among employees as well as the financial information contained in the dataset. Some of the important characteristics of the dataset are the following:

i. There are records of 146 employees of which 18 are persons of interest
ii. Each record has 21 features related to email exchanges and financial transactions; there is 1 bool feature which tells whether the record is a person of interest or not.
    1. These are 6 features related to email and their type is string -
        - to_messages
        - shared_receipt_with_poi
        - from_messages
        - from_this_person_to_poi
        - email_address
        - from_poi_to_this_person
    2. These are 14 financial features and their type is integer -
        - salary
        - deferral_payments
        - total_payments
        - exercised_stock_options
        - bonus
        - restricted_stock
        - restricted_stock_deferred
        - total_stock_value
        - expenses
        - loan_advances
        - other
        - director_fees
        - deferred_income
        - long_term_incentive
iii. I created a couple of scatter plots using financial variables such as salary, bonus, stock values, to check the relative values of the variables. The scatter plots show the existence of an outlier record in the data set with unusually high values. This is the record with the name TOTAL, which isn't a PoI and hence which I removed before doing further analysis. Apart from this, there were outliers such as executives, Ken Lay and Jeffrey Skilling, who can be reasonably retained in the dataset. I also tried to search for "NaNs" in email_address and salary field which I considered important to be missing. Still I found 27 of them, of which one record was peculiar, 'THE TRAVEL AGENCY

IN THE PARK' which had almost all the fields as NaNs and hence I removed it. Investigating the NaNs further shows that all fields except poi have at least 20 NaNs going up to 142 NaNs for Loan Advances.

Q2. 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 importance 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”]

For new feature creation I utilized some of the apparently important features to create new ones. For e.g. ownership of stocks is important from ownership perspective of a company. Usually the stock ownership versus salary ratio of an individual rises non-linearly as an employee becomes more important. Hence I created a 'Stock_Salary' feature as a ratio of the 2 (if both of them were not NaNs). Stock has only 20 NaNs while Salary has around 50 NaNs. Hence this field will have about 90 values which is a good number as well. Another set of features I created are around email interaction with the PoIs. The variables are 'msg_from_poi', 'msg_to_poi' and 'emp_poi_interact', the first to count the number of messages sent from and sent to a PoI respectively by an employee, divided by the total messages they received and total messages they sent. The 'emp_poi_interact' is the ratio of the sum of all messages received from and sent to a PoI to the sum of all the messages sent and received by an employee. Since the email variables are in string format, they had to be cast to float to compute these new features.

For feature selection, I deploy VarianceThreshold and SelecKBest algorithms. Using VarianceThreshold I remove 3 features whose variance is below 80%. Next I use the SelectKBest algorithm to select the 7 highest scoring features. Further I used the MinMaxScaler to rescale the features as while the email features are in 100s, the financial features could range up to millions. The final 7 features selected are:

Feature	**Score**
exercised_stock_options	24.81508
total_stock_value	24.1829
bonus	20.79225
salary	18.28968
deferred_income	11.45848
long_term_incentive	9.922186

restricted_stock	9.212811

Q3. 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 used multiple classifiers including Naïve Bayes, Logistic Regression, Kmeans, AdaBoost, Support Vector Machine, Random Forest, Stochastic Gradient Descent. The table below shows the performance of each using sklearn's library's metrics functions, along with results from tester.py

	sklearn				Tester.py			
	Accuracy	**Precision**	**Recall**	**F1**	**Accuracy**	**Precision**	**Recall**	**F1**
Naïve Bayes	0.851	0.458	0.392	0.391	0.846	0.456	0.384	0.417
Logistic Regression	0.862	0.428	0.221	0.272	0.864	0.569	0.2	0.305
Kmeans	0.595	0.366	0.404	0.227	0.779	0.261	0.298	0.278
AdaBoost	0.838	0.311	0.208	0.221	0.828	0.349	0.221	0.271
SVM	0.851	0.283	0.079	0.115	Lack of true positive predictions			
Random Forest	0.861	0.349	0.148	0.186	0.85	0.443	0.179	0.255
Gradient Descent	0.785	0.318	0.232	0.192	0.586	0.118	0.294	0.169

The Naïve Bayes seems to be performing better in all evaluation metrics than others while SVM seems to be the worst.

Q4. 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? (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”]

Tuning a machine learning algorithm is about tweaking the various input parameters that define the machine learning classification model and coming up with the right combination of all or most of the important parameters so that the performance of the algorithm is optimized in terms of the validation metrics that are chosen. I have used the GridSearchCV which searches for optimum parameter combination for a classifier in an exhaustive manner.

Since Naïve Bayes doesn't have any parameters for tuning, I have used the Kmeans classifier to try out the GridSearchCV function using a limited set of parameters as shown below:

```
parameters = dict(
        n_clusters = [2, 4],
        max_iter = [300, 500],
```

```
            n_init = [10, 50],
            init =['k-means++', 'random'],
            tol=[0.001, 0.0001]
        )
```

I also used the Stratified ShuffleSplit cross-validator which helps against overfitting by using stratified sampling. The results I got are as follows:

```
The best parameters are {'tol': 0.001, 'init': 'random',
'max_iter': 500, 'n_init': 10, 'n_clusters': 4}
```

It can be seen that tol or tolerance, init or Method for Initialization, Max_Iter are all non-default values for the K-Means Clustering algorithm function in SKLearn library.

Q5. 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"]

Once a model is selected with a set of parameters, it is important to try to know in advance how will it perform in the real world scenario where it might have to handle scenarios which it didn't come across when working with the training data. Hence the classifier has to undergo validation with another set of data called testing data set. Usually a given data set is split into training and testing data, as is done by for e.g. the train_test_split function in SKLearn's cross validation library. I used the same function in ClassifierPerf function to validate the performance of all the classifiers. The testing data was set as 30% while training data was 70% of the dataset. As an additional technique I used Stratified Shuffle Shift with the GridSearchCV to validate the K-Means classifier.

Q6. 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"]

I used Accuracy, Precision, Recall and the F1 score to measure the performance of all the classifiers. In context of this project these evaluation metrics can be understood in the following manner:

- Accuracy measures the absolute correctness of the classifier, for example if all PoIs and non-PoIs are correctly identified the accuracy will be 1, and a deviation from correctly identifying a PoI or a non-PoI will lead to reduction in accuracy score. Hence it assumes equal cost for all kinds of errors.
- Precision tells us about how good is the prediction, so if we are looking for a PoI, whether the classifier misses any or catches the most it can. If the classifier predicts a non-PoI as PoI then precision goes down.

- Recall tells us about how correct is the prediction when the classifier actually encounters a particular class of interest. For example if the classifier marks a PoI as a non-PoI, the recall will go down.
- F1 Score is also a measure of the test's accuracy as it considers both Precision and Recall, it can be considered a weighted average of Precision and Recall.