Machine Learning Project

Enron Dataset Investigation

Done by ***Javeed Basha H***

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”]

The goal of this project is to investigate the Enron dataset, which consists of the financial and email data of the company named “***Enron***”. We have to build an analytical model using this dataset which could be used to identify the “*PoI -> Person of Interest*”, which is actually an individual who may have committed a fraud. Now, since the dataset consists of a labelled data, persons who have committed a fraud where listed as “*PoI*”. Machine Learning algorithms are useful because they can process data much faster, analyze it and also provide accurate results, which are useful in making predictions without any human intervention.

Dataset Information:

The Enron Dataset consists a total of 146 employees, of which 18 are ‘PoI’. A total of 21 features are present in the dataset including the financial features and the email features.

```
Features = ['poi',
            'salary',
            'bonus',
            'deferral_payments',
            'deferred_income',
            'director_fees',
            'exercised_stock_options',
            'expenses',
            'loan_advances',
            'long_term_incentive',
            'other',
            'restricted_stock',
            'restricted_stock_deferred',
            'total_payments',
            'total_stock_value',
            'from_messages',
            'from_poi_to_this_person',
            'from_this_person_to_poi',
            'shared_receipt_with_poi',
            'to_messages']
```

Outlier’s:

I found three outliers namely;

- *TOTAL* – Total Values
- *THE TRAVEL AGENCY IN THE PARK* – Not an individual
- *LOCKHART EUGENE E* – Record contains no useful data present

I removed these outliers from the dataset using *dict.pop()* function.

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"]

I used a total of 10 features using the SelectKBest() feature selection algorithm and I did perform features scaling operation using the MinMaxScaler(), this is an important step cause the scaler helps to optimize the features as it had different units present in it. Features Scaling ensures that for all the applicable classifiers, the features will be evenly weighted.

Before applying the SelectKBest() and MinMaxScaler() to features, I engineered two additional features namely;

total_sum -> Sum of all the finance features namely salary, bonus, exercised_stock_options and total_stock_value.

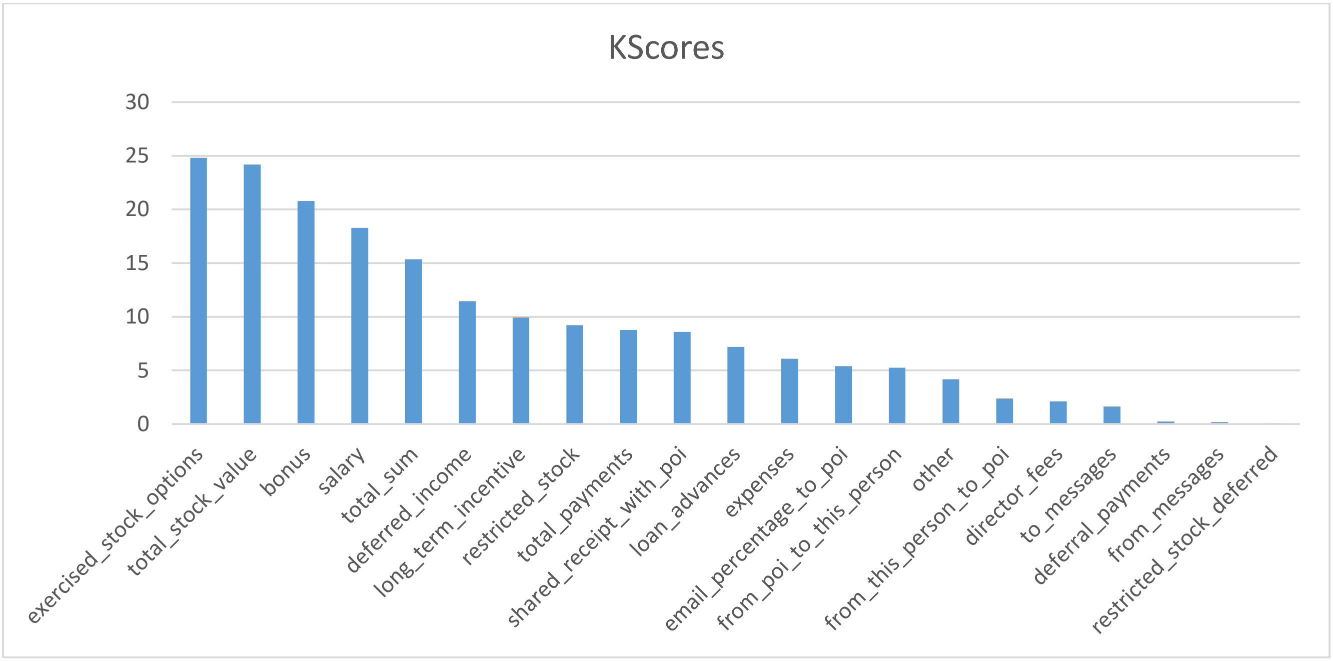
email_percentage_to_poi -> Proportions of emails sent and received by 'PoI'.

total_sum feature helps to find out the total wealth of the person, and email_percentage_to_poi helps to determine the proportion of messages communicated between PoI, compared to the rest of the individuals. Now, applying the SelectKBest() automated univariate feature selection algorithm.

All Features with K-Scores:

```
exercised_stock_options       24.8150797332
total_stock_value             24.1828986786
bonus                         20.7922520472
salary                        18.2896840434
total_sum                     15.3692768646
deferred_income               11.4584765793
long_term_incentive           9.92218601319
restricted_stock              9.21281062198
total_payments                8.77277773009
shared_receipt_with_poi       8.58942073168
loan_advances                 7.18405565829
expenses                      6.09417331064
email_percentage_to_poi       5.39937028809
from_poi_to_this_person       5.24344971337
other                         4.187477507
from_this_person_to_poi       2.38261210823
director_fees                 2.12632780201
to_messages                   1.64634112944
```

```
deferral_payments             0.224611274736
from_messages                 0.169700947622
restricted_stock_deferred     0.0654996529099
```



From the above chart, we can notice that the biggest cutoff is at total_sum and deffered_income features, but, in order to select many more features I have used the second cutoff between the loan_advance and shared_receipt_with_poi. Hence, my 10 best features and their score are given below;

K Best Features:

exercised_stock_options	-	24.815
total_stock_value	-	24.183
bonus	-	20.792
salary	-	18.290
total_sum	-	15.369
deferred_income	-	11.458
long_term_incentive	-	9.922
restricted_stock	-	9.213
total_payments	-	8.772
shared_receipt_with_poi	-	8.589

exercised_stock_options have the highest k-score of 24.815, followed by the total_stock_value feature, also additional feature namely total_sum is also figured in the K-best features with a feature score of 15.369

The best 10 features excluding the 'PoI' identifier were selected according to the K-Scores obtained, apart from the features mentioned above the other features were excluded since they had low K-Scores. The above mentioned features were used because they slightly enhanced the precision and accuracy of the algorithms tested.

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 a total of five algorithms namely;

Supervised Algorithm's:

- Naive Bayes Algorithm
- Decision Tree Algorithm
- AdaBoost Algorithm
- Random Forest Classifier Algorithm – Tree Ensembles

Unsupervised Algorithm:

- KMeans Cluster Algorithm

The final algorithm I ended up using is the ***Gaussian Naive Bayes*** Algorithm, because the final precision and recall score is relatively higher than the prescribed minimum output of 0.3. Gaussian Naive Bayes Algorithm, is a super simple algorithm and doesn't have much parameters to tune, a Naive Bayes classifier will converge quicker than discriminative models like logistic regression, so less training data is required and even in cases where the NB assumption doesn't hold, a NB classifier still often performs surprisingly well in practice, also I have tried various other algorithms, with Random Forest algorithm performing reasonably well and sufficient along with the AdaBoost Classifier algorithm.

Algorithm	**Precision**	**Recall**
Naive Bayes Algorithm	0.37401	0.32950
Decision Tree Algorithm	0.25310	0.07150
AdaBoost Algorithm	0.43529	0.14800
Random Forest Classifier Algorithm	0.42293	0.17150
KMeans Cluster Algorithm	0.14785	0.06350

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”]

For each and every algorithm a number of parameters are tuned using the GridSearchCV, the parameter tuning helps in fine tuning the algorithm, and select the best parameters needed to provide the best result.

Parameters used to tune the algorithm’s:

- **Decision Tree:** {criterion, min_samples_split, max_depth, min_samples_leaf, max_leaf_nodes}
- **Naïve Bayes:** None
- **AdaBoost:** {n_estimators, algorithm, learning_rate}
- **RandomForest:** {n_estimators, n_jobs, random_state}
- **KMeans:** {n_clusters, algorithm, tol}

The above mentioned parameters where used to fine tune the algorithms to provide and optimal result. After running GridSearchCV, I was able to obtain the best parameters for various algorithms as given below;

```
Decision Tree Best Parameters:

Tried:
        Criterion: ['gini', 'entropy'],
        min_samples_split: [2, 10, 20],
        max_depth: [2, 5, 10],
        min_samples_leaf: [1, 5, 10],
         max_leaf_nodes: [5, 10, 20]
Obtained:
        criterion='gini',
        max_depth=2,
        max_leaf_nodes=5,
        min_samples_leaf=10,
        min_samples_split=2,
```

Decision Tree Algorithm: (Parameter’s Analysis)

- *criterion* – A function to measure the quality of the split, usually two features were used, namely ‘gini’ and ‘entropy’, using the GridSearchCV, the best criterion was found to ‘gini’.
- *min_samples_split* – The minimum samples required to split an internal node. min_samples_split is 2.
- *max_depth* - The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. In my case the max_depth is 2.
- *min_samples_leaf* – It’s the number of samples required to be at a leaf node. In my case its 10.

- *max_leaf_nodes* - Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes. In my case the max_leaf_nodes are 5.

```
Ada Boost Best Parameters:
Tried:
        n_estimators: [50,100,200]
        algorithm: ['SAMME', 'SAMME.R'],
        learning_rate: [0.4,0.6,1]}

Obtained:
        algorithm='SAMME',
        learning_rate=0.4,
        n_estimators=50,

Random Forest Best Parameters:
Tried:
        n_estimators: [10, 20, 30]
        n_jobs: [1, -1],
         random_state: [42, 50, 60]

Obtained:
        n_estimators=10,
        n_jobs=1,
        random_state=50,

KMeans Best Parameters:
Tried:
        algorithm: ['elkan', 'full']
        tol: [1e-2, 1e-3, 1e-4]
Obtained:
        algorithm='elkan',
        tol=0.01
```

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 performed to evaluate the machine learning algorithm's performance, usually the data is split into two sets namely the training and the test sets using the Cross-Validations technique, the split data is in the ratio of 70:30 (which can be modified), where a large part of the data is used in training set and the testing set consists a smaller part of the test data. Cross Validation is used to detect when overfitting starts.
Since, the enron dataset is small, this type of cross validation helps me to gauge my algorithms performance and provide more accurate results. A classic mistake in validation is that of over-fitting, i.e., testing the data on which it was trained on, without separating the training data from the testing data, it is difficult to assess the performance of the algorithm.

I validated the data using tester.py, where I utilized the provided K-folds evaluator to test my algorithm and assess it accordingly. The output of the this test_classifier() was to return the performance of my algorithm in terms of performance metrics.

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"]

The main evaluation metrics used are Precision, Recall and F1 Score. The relevant metric score for my selected algorithm and also the additional algorithm is given below;

Algorithm	**Precision**	**Recall**	**F1 Score**
Naïve Bayes Algorithm	0.37401	0.32950	0.35035
Random Forest Classifier Algorithm	0.42293	0.17150	0.24404

Precision – How often the class prediction is right. A high precision means PoI identified by the algorithm tends to be correct.

Recall – How often the guessed class and the actually class actually occurred. A high recall means that every time a PoI shows up in the test data, the algorithm has good chance to identify it.

F1 Score – Mean of Recall and Precision.

The sparse nature of the dataset and the limited number of PoI identifiers, attributed to a challenging aspect in the completion of the project. Most of my selected algorithms had a high precision score, such as the Random Forest and AdaBoost, but, this had to be neglected because they had a very poor recall score, in this project the higher the recall scores the better.

Based on my algorithm selection and the performance outcome of the metrics, I can conclude that ***Gaussian Naïve Bayes*** algorithm gives overall best scores, followed by Random Forest Classifier and others.