Introduction

In 2000, Enron was one of the biggest companies in USA.By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting federal investigation a significant amount of private data entered into the public domain including thousands of emails and financial data of top executives. In this project i applied my machine learning skills to create person of interest identifier based on financial and email data made public by Federal Agency after Enron Scandal.

1.Datasets and Outliers

The dataset provided consist of 146 data points and have 21 features. Out of this 146 persons 18 are labeled as Person of Interest. After plotting the “salary” feature against “poi” feature ,one outlier can be detected with the index”TOTAL”. Hence this was removed from the dataset. ”poi” feature column is used as labels here.

2.Feature creation and selection

After carefully examining the dataset, intuitively it becomes obvious that combined features can be generated by combining “salary”,”bonus”,”exercised_stock_options” ,”long_term_incentive” .I stored the this new feature in my dataframe column named ”wealth”. Apart from that i generated two other features “fraction_from_poi”, and ”fraction_to_poi”, which contains the fraction of emails received from person of interest and sent to person of interest respectively. On selection part, intuitively i selected all the features at first except “email-address” and “director_fees” and fed to the SelectKBest algorithm to identify top scoring features.

3. What features did I end up using in your POI identifier, and what selection process did I use to pick them? Did I have to do any scaling? Why or why not?

In this project, first I used scikit-learn's SelectKBest module to get the 7 best features among the 21 features present. This was because in the recursive feature selection, I saw a peak at around 7 for the features vs cross validation score. But this had to be a half manual iterative process as the dataset is very skewed towards non POIs and only 18 POIs were there in the dataset. So when I was using the 7 features suggested by SelectKBest algorithm, I was getting an accuracy of only ~72% with very low precision and recall score (0.2 and 0.333 respectively). In the below table you can see the top 7 features along with their scores. The K-best approach is an automated univariate feature selection algorithm, and in using it, I was concerned with the lack of email features in the resulting dataset. Thus, I engineered four features, wealth, poi_message, fraction_to_poi and fraction_from_poi which were the combined feature , proportion of email interaction with POIs, proportion of email fraction to POIs and proportion of email fraction from POIs. So here we mutate our original data dictionary and add these new features to the features' list. The scores of the 7 best

features selected by K-best approach is given below. Apart from these 7 features, I also added 3 more features regarding emails: 'poi_ratio', 'fraction_to_poi' and 'fraction_from_poi' .

1. exercised_stock_options: 23.968332134035151
2. total_stock_value: 23.3293437062336
3. bonus: 19.989875526651559
4. salary: 17.432087382499653
5. fraction_to_poi: 15.714789421485834
6. deferred_income: 11.04775908067936
7. wealth : 14.125536081651695

After much iterative manual testing with different combinations of the features and trying out by deleting them one at a time, I finally came up with these 4 features to be used as my final features list:
1. fraction_from_poi
2. fraction_to_poi
3. shared_receipt_with_poi

Before training the machine learning algorithm classifiers, I scaled all features using a min-max scaler. This was vitally important, as the features had different units and varied significantly by several orders of magnitude. Feature-scaling ensured that for the applicable classifiers, the features would be weighted evenly. Only after this step, I split my data into training and testing set.

4. What algorithm did I end up using? What other one(s) did I try? How did model performance differ between algorithms?

I used Decision Tree Classifier to predict the possible POIs in the dataset. The main motivation for me behind using this classifier was that the Nonlinear relationships between parameters do not affect tree performance and Decision trees implicitly perform variable screening or feature selection. I also noticed that this was one of the classifiers which was giving one of the highest accuracies. Apart from Decision trees, I also tried fitting the train data and predict the test data with Naive Bayes, Adaboost, K Nearest Neighbours and Standard Vector Machine classifiers. Adaboost gave almost as high precision metrices as decision trees. Naive Bayes and SVM also gave higher accuracy scores around 92-93% but the precision and the recall score of SVM was zero. With Decision trees, at this point I was getting an accuracy score around 96.15% with a precision score of 1 and recall score of 0.500. Given below are the precision metrices for all the algorithms I tried.

1. Decision Tree

a) Accuracy - 96.153%

b) Precision Score - 1

c) Recall Score - 0.500

2. Naive Bayes

a) Accuracy: 92.31%

b) Precision Score: 0

c) Recall Score: 0

4. Adaboost

a) Accuracy: 96.153%

b) Precision Score: 1

c) Recall Score: 0.5
5. Standard Vector Machine

a) Accuracy: 92.31%

b) Precision Score: 0

c) Recall Score: 0

6. K nearest neighbours

a) Accuracy: 84.62%

b) Precision Score: 0

c) Recall Score: 0

5. What does it mean to tune the parameters of an algorithm, and what can happen if I don't do this well? How did I tune the parameters of your particular algorithm? What parameters did I tune?

Parameter tuning for any algorithm in Machine Learning is a vital step. Our goal, is usually to set the parameters for the algorithm used to get optimal values that enable us to complete a learning task in the best way possible. Thus, tuning an algorithm or machine learning technique, can be simply thought of as process which one goes through in which they optimize the parameters that impact the model in order to enable the algorithm to perform the best (once, of course we have defined what "best" actual is).

With our dataset accuracy is not quite meaningful because of the disproportion among classes. So here, we are more interested in the precision score and the recall score metrices. Before tuning my algorithm, the values precision score was 1 and the recall score was 0.500 with an accuracy of 96.15%. Maybe this high value was caused by some overfitting. I tuned my Decision Tree Classifier model with GridSearchCV and validated the using Stratified Shuffle Split and Cross Validation with my customized scoring function to put a threshold for both precision_score and the recall_score. The parameters which I tuned were max_depth, min_samples_split, min_samples_leaf and criterion. The final accuracy I got after tuning my model ranged between 87-89% with a precision score around 0.433 and the recall score of 0.822.

6.What is validation, and what's a classic mistake you can make if I do it wrong? How did I validate my analysis?

In machine learning, model validation is referred to as the process where a trained model is evaluated with a testing data set. The testing data set is a separate portion of the same data set from which the training set is derived. The main purpose of using the testing data set is to test the generalization ability of a trained model. Validation is performed to ensure that a machine learning algorithm generalizes well. A classic mistake is over-fitting, where the model is trained and performs very well on the training dataset, but markedly worse on the cross-validation and test datasets. I utilized two methods for validating the analysis:

1. StratifiedShuffleSplit, folds = 1000 : The main reason why I used Startified Shuffle Split as a validation method was that the dataset was small and skewed towards non-POI. I needed a technique that accounts for that. Otherwise the risk is that in the validation phase, we would not be able to assess the real potential of our algorithm in terms of performance metrics. The chance of randomly splitting skewed and non representative validation sub-sets could be high, therefore the need to use stratification (preservation of the percentage of samples for each class) to achieve robustness in a dataset with the aforementioned limitations. I took the average precision an recall score of 1000 randomized trials with the help of test_classifier function. This data was subset into training and testing set in a 3:1 ratio.
2. Cross Validation: Cross Validation is a technique which involves reserving a particular sample of a data set on which you do not train the model. Later, you test the model on this sample before finalizing the model. Here I did a cross validation on the data by splitting the data, fitting a model and computing the score 5 consecutive times (with different splits each time). I used 'accuracy' as a scorer object.

7. Give at least 2 evaluation metrics and the average performance for each of them. Explain an interpretation of my metrics that says something human-understandable about my algorithm's performance.

The main evaluation metrics utilized were precision and recall. Precision captures the ratio of true positives to the records that are actually POIs, essentially describing how often 'false alarms' are (not) raised. Recall captures the ratio of true positives to the records flagged as POIs, which describes sensitivity. For the Decision tree algorithm which I used, following were the evaluation metrices for the two validations:

1. StratifiedShuffleSplit

a) Precision: 0.50829

b) Recall: 0.82200

2. Cross Validation

a) Precision: 0.50292

b) Recall: 0.82200

Both validations performed quite well given the fact that this dataset didn't contain plethora of data (very few POIs) and the data was noisy too. The recall score is very high. This is actually a good thing. The ratio of the false positives to true positives is much higher than the ratio of false negatives to true negatives. This is actually good for us because we would rather mark a person as POI who in reality is not a POI than not able to catch the Real POIs and marking them as False POIs. My argument here is even though we mark someone as a POI wrongly, he/she will still get through unscathed after investigation. But on the other hand, if we miss someone who is indeed guilty, then that person is just not facing the justice as he/she should have.

Conclusion

So we see by applying Stratified Shuffle Split to our Decision Tree Classifier, we are able to improve the performance metrices of our classifier. This dataset is a very sparse dataset where accuracy(almost 89% in our case) is not a good metrics given the fact it is very skewed towards non POIs. We only have 18 POIs in the dataset. Even though originally we had 36 POIs, somehow half of them are not present in this dataset. We have a high recall score(0.822) which is a good thing given how sparse this dataset is and where accuracy is not a good metrics to perform model quality. We also see that ratio of the false positives to true positives is much higher than the ratio of false negatives to true negatives. This is actually good for us because we would rather mark a person as POI who in reality is not a POI than not able to catch the Real POIs and marking them as False POIs. My argument here is even though we mark someone as a POI wrongly, he/she will still get through unscathed after investigation. But on the other hand, if we miss someone who is indeed guilty, then that person is just not facing the justice as he/she should have.

These numbers are quite good but still we can improve the accuracy of the model. One of the possible ways is to dig into the actual email messages more. Then we can find a pattern of emails to/from a specific point instead of to/from POIs. We can also do text mining to make some vocabulary pattern emerge from this huge chunks of emails to get more behavioural patterns of the POIs. As we found out that the POI data was incomplete in our dataset, the next realistic thing to try might be to extract more data from the emails.