PROJECT 5 – ENRON DATASET WITH MACHINE LEARNING

Questions:

----
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”, “outliers investigation”]

The goal of the project is to identify the persons of interest (POI's for further reference in the document) – people who were involved with the fraud in Enron scandal which led to the bankruptcy of the company.

The data we have is a compilation of financial dataset and the Enron e-mail corpus which were put together by Katie for the project. The idea is to use machine learning algorithms in the form of text learning and regressions and classifiers to to identify patterns in the dataset which help us identify the POI's. There are some data fields which are left as “NaN” because of the lack of data. We can handle this by replacing the data points with zero if it is a quantifiable field like salary. In addition when looking at the financial data to find a pattern between the financial compensation and POI list, there was one entry which had a huge value for bonus. This turned out to be the “Total” field from the financial table, which had to be taken out. Lastly, we need to be careful when combining data from different sources to ensure we not introduce biases in the dataset which can be wrongly interpreted by the algorithm.

Total number of data points – 146
Number of POI's and Non-POI's – 18, 128 respectively
Number of features given – 14 financial features and 7 e-mail features – total 21
Features with missing values – There are many features with missing values which hae been designated the value of “NaN”
Outliers – The “Total” field in the finance part of the dictionary

----
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”]

I ended up using 6 features in my POI identifier - 4 financial features and 2 features from the e-mail dataset. The financial features used were "salary","bonus", "exercised_stock_options","long_term_incentive". The e-mail features used were "fraction_emails_to_poi_from_person" and "fraction_emails_from_poi_to_person".

Scaling:
Since the analysis, used PCA, I used a minmax scaler to scale the features. It turned out that scaling the features did not improve or decrease the performance of the algorithm, but is a necessary step for PCA.

New feature:
Initially I tried a new feature "Stocks sold" - the sum of exercised_stock_options and restricted_stock_options. I initially looked p all the POI's and found that all of them except one person indicated as POI were involved in insider trading or were aware of the financial situation of the company. This would mean that they would have sold most of their stocks, when the value was high and hence would show up as outliers, compared to people who were unaware that the company was making paper profits. However, this new metric did not give the classifier any edge over the exercised_stock_options variable. So I ended up not using it.

The following results clearly show that the precision and recall metric are lower than the model which does not have the Stocks_sold feature. Hence the final model does not include this feature.

```
#################################################################################
Addition of 'Stocks_sold':
Pipeline(steps=[('randomizedpca', RandomizedPCA(copy=True, iterated_power=3,
n_components=3, random_state=None,
       whiten=False)), ('decisiontreeclassifier',
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=8, min_samples_leaf=1,
            min_samples_split=5, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best'))])
      Accuracy: 0.81892 Precision: 0.37877      Recall: 0.27650   F1: 0.31965 F2:
0.29228
      Total predictions: 13000      True positives:  553    False positives:  907
      False negatives: 1447   True negatives: 10093

#################################################################################
```

Feature Selection:
I picked up the financial features indicated above and then did a PCA on these variables. When the number of components was iterated with a given classifier (Decision tree in my case). It turned out that the best possible score could be achieved with just 3 components with the first component being able to explain 94.4% of the variance.

```
###########################################################################
"""
The explained variance from PCA is:  [ 0.94454922  0.04951619  0.00526351]
The first principal component is:  [  3.19117508e-09   2.01989184e-08
1.74080714e-07   1.09810516e-08
  -3.35661088e-15   9.59770061e-13]
The second principal component is:  [  6.25442022e-08   7.34495713e-07
-9.82234097e-08   1.87886638e-07
   4.88430913e-13   1.43540820e-11]
"""
```

Iterating number of components:

When I did the feature selection, I iterated over different number of components from 2 to 6 which had sub-optimal results compared with 3 components. Indicating the results below for number of components 5 and 6

```
Pipeline(steps=[('randomizedpca', RandomizedPCA(copy=True, iterated_power=3,
n_components=5, random_state=None,
       whiten=False)), ('decisiontreeclassifier',
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=8, min_samples_leaf=1,
            min_samples_split=5, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best'))])
      Accuracy: 0.85438 Precision: 0.54425      Recall: 0.32900   F1: 0.41010 F2:
0.35726
      Total predictions: 13000      True positives:  658    False positives:  551
      False negatives: 1342   True negatives: 10449


Pipeline(steps=[('randomizedpca', RandomizedPCA(copy=True, iterated_power=3,
n_components=6, random_state=None,
       whiten=False)), ('decisiontreeclassifier',
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=8, min_samples_leaf=1,
            min_samples_split=5, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best'))])
      Accuracy: 0.84554 Precision: 0.49693      Recall: 0.32350   F1: 0.39188 F2:
0.34777
      Total predictions: 13000      True positives:  647    False positives:  655
      False negatives: 1353   True negatives: 10345
```

###

----
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 the decision tree classifier. I tried Naive Bayes, Logistic regression and SVM's as the other algorithms. The SVM algorithm was not able to identify the POI's when applied on the test set. Hence it had the worst performance of the algorithms and thus rejected.

Then I implemented each of the algorithms – Decision Tree, Naive Bayes and Logistic Regression. The results have been shown below. Out of the three, Naive Bayes had the highest precision (0.44), but failed the requirement for recall marginally (0.29). Logistic regression had a good recall rate (0.46) but had a low precision score(0.25). Decision Tree, on the other hand had a balanced performance in terms of both precision and recall with scores of 0.40 and 0.36 respectively, comfortably scoring over the requirement of 0.3 for the measures. Therefore I chose, decision tree as the algorithm to be used and applied parameter tuning for the same to improve the performance further.

Hence I went ahead and used the Decision tree classifier as my POI identifier.

##

Naive Bayes:

```
Pipeline(steps=[('randomizedpca', RandomizedPCA(copy=True, iterated_power=3,
n_components=3, random_state=None,
       whiten=True)), ('gaussiannb', GaussianNB())])
      Accuracy: 0.83592 Precision: 0.44545      Recall: 0.27150   F1: 0.33737 F2:
0.29450
      Total predictions: 13000      True positives:  543    False positives:  676
      False negatives: 1457   True negatives: 10324
```

Decision Tree:

```
Pipeline(steps=[('randomizedpca', RandomizedPCA(copy=True, iterated_power=3,
n_components=3, random_state=None,
       whiten=False)), ('decisiontreeclassifier',
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=None, splitter='best'))])
      Accuracy: 0.81977 Precision: 0.40035      Recall: 0.34450   F1: 0.37033 F2:
0.35439
      Total predictions: 13000      True positives:  689    False positives: 1032
      False negatives: 1311   True negatives: 9968
```

Logistic Regression:

```
Pipeline(steps=[('randomizedpca', RandomizedPCA(copy=True, iterated_power=3,
n_components=None,
       random_state=None, whiten=False)), ('logisticregression',
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False))])
      Accuracy: 0.70600 Precision: 0.25041      Recall: 0.45700   F1: 0.32354 F2:
0.39227
      Total predictions: 13000      True positives:  914    False positives: 2736
      False negatives: 1086   True negatives: 8264
```

##

----
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? (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”]

Some algorithms have parameters which can be tuned. For example, a SVM has the following parameters which can be tuned – Kernel, Gamma and the C parameters which

can be changed to understand the compatibility to a dataset and study the effects it has on the final classification. Kernels – rbf, sigmoid and linear are examples of the same. If the tuning is not done properly, we risk under-fitting or over-fitting the algorithm to the data – also called the bias variance trade-off.

Similarly, for my algorithm, the decision tree classifier, I had a few parameters I could tune to improve the performance of the algorithm. I used the gini function to measure the quality of the split. In addition, I also used the min_samples_split parameter and the max_leaf_nodes to tune the tree.

I varied the min_samples_split parameter between 2 to 10 and found that the precision and the recall were maximum when the samples was either 5 or 6. This intuitively makes sense, since the we have around 146 samples in our dataset, we need at least 5 (3.5% of overall dataset, much higher if the training sample count is used) to make a decision. In addition, we cannot increase this any further or we run the risk of pruning the tree too much and hence under-fitting the tree.

In addition to this metric, I used the max_leaves_node to prune the tree further. This has the benefit of restricting the number of leaves at the end of the tree. In addition the tree grows more on the nodes which give a much better classification of the POI's and the non-POI's. I swept the values of the max_leaf_nodes between 2 and 15. The best value to improve the precision and recall was 9. Hence this was used in the final model.

The model tied to improve both the recall and precision simultaneously giving equal weighting to each measure when tuning the tree. For eg. when choosing the tuning parameters, the metric of Precision – 0.56 and Recall – 0.36 was favored over the tuning parameters which resulted in a Precision – 0.6 and Recall – 0.31. The idea was to make the algorithm as balanced as possible with respect to classifying a person as a POI.

----
5. 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 the process by which we can evaluate the performance of the algorithm or classifier on a unknown dataset. The classic mistake one can make is by training and testing on the same data set. This will give us an over optimistic result which leads us to believe that our algorithm is better than it actually is.
To avoid this error, my analysis, uses different datasets to train and test the algorithm. We use 30% of the overall samples to test the algorithm after it has been trained on 70% of the dataset. This ensures that we avoid us overestimating the power of our classifier.
In addition, to maximize the amount of data on which the classifier is tested, we use cross validation to increase the data points on which testing was done. This was achieved by the tester function which uses “StratifiedShuffleSplit” to split the overall data set into training and testing data sets multiple times – in this case, thousand different ways. This was used to validate the performance of the algorithm and arrive at the precision and recall values of 0.56 and 0.36 respectively.

----
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”]

The evaluation metrics chosen are Precision and Recall. Accuracy is not a good metric, as the number of POI's is very low in the dataset. Hence the outcome is extremely skewed in favor of non-POI's.

I will define the metrics with respect to our problem.

Precision – The probability of our algorithm correctly predicting that a person is POI when it predicts someone as POI. This is the likelihood of a correct identification of POI by the algorithm when someone is declared a POI by the algorithm.

Recall - The probability of our algorithm predicting that a person is a POI (as opposed to predicting him a non-POI) when he is actually a POI in reality.

To summarize, increasing precision means we will increase the possibility of a person being a POI in reality, when our algorithm classifies him as a POI. Increasing recall would be to increase the possibility that our algorithm classifies a person as POI, when he actually is a POI.