Raphael Tam

Introduction

The goal of this project is to use machine learning techniques to identify "person of interest" from a group of 146 Enron employees. Enron was a US company that went bankrupt due to corporate malfeasance. Many of its officers and employees were indicted, convicted or bargained with authorities for immunity or reduced sentences. They are labeled "person of interest" (POI) in this project. There are 18 POIs.

Drawn from public records, the dataset is made up of the financial information and their emails. Financial information is made up of different components of an individual's pay while serving as an employee at Enron: salary, bonus, total stock value, etc. The data set also provides the 'meta-data' on each individual's emails: the number of messages sent/received, and the number of these messages that came from/sent to/shared receipt with persons of interest. Each person is labeled as either a POI or not. The label is assigned manually based on public information. There are 21 features in the dataset.

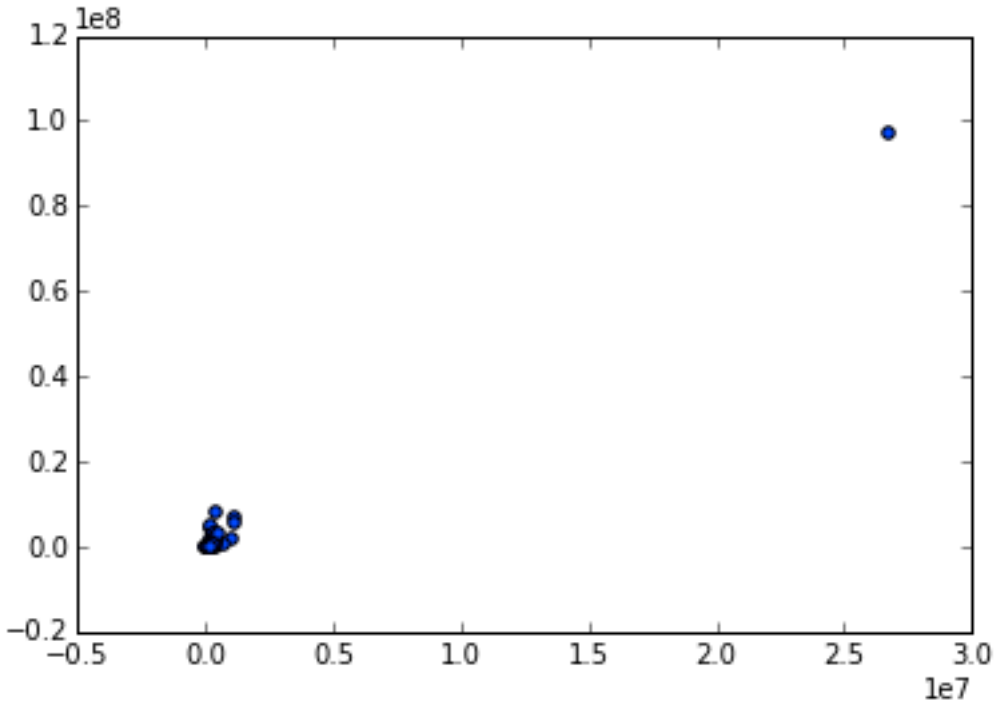
Machine learning algorithms are good at learning from the data the characteristics that are shared more commonly among the POIs than non-POIs and as such can distinguish somewhat between the two classes. These are often subtle characteristics that are not easily discernible by simply visualizing the data. The algorithms are based on statistical principals such as information gain and minimizing the sum of squared errors between learned predictions and actual data. Evaluation metrics are used to assess how well can machine learning predict whether a person is a POI.

Outliers

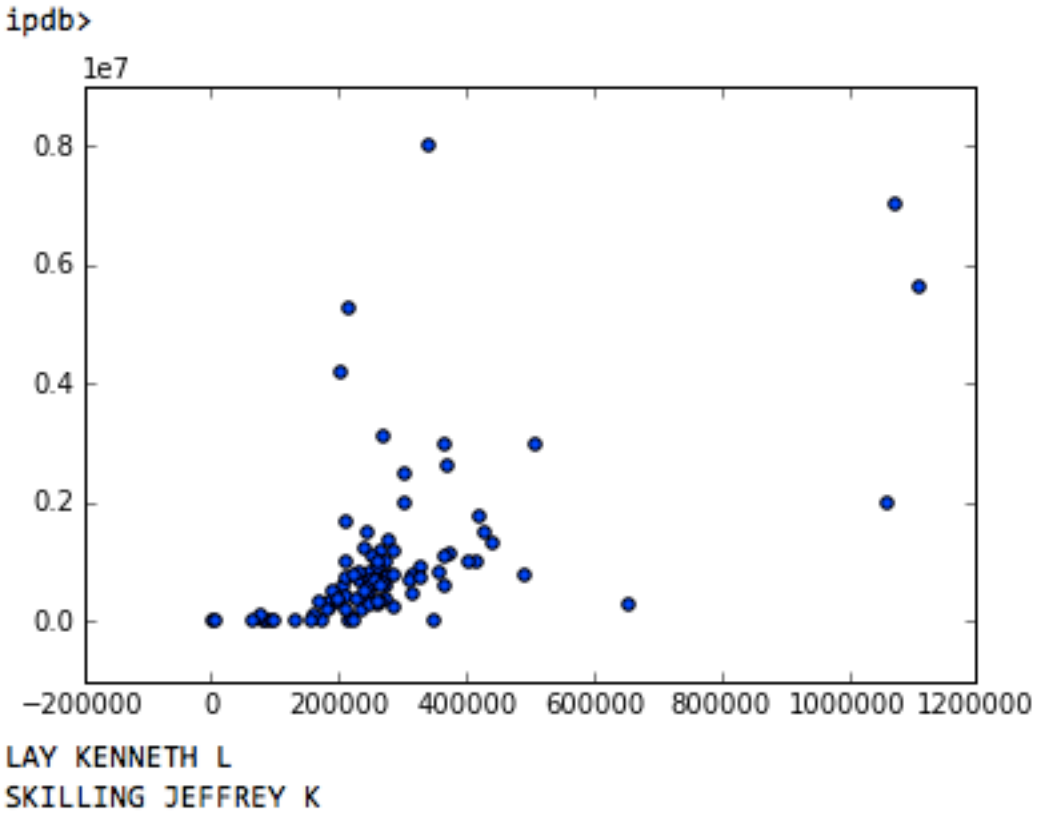
One of the first steps in the analysis is to visualize the data to check for outliers. Outliers are data points that look suspicious within the context of the problem. Sometimes it is obvious why the data point(s) does not belong. Other times we rely on our best subjective judgment to remove them from the dataset.

Plotting salary and bonus with salary_bonus_scatter_plot.py below, one can see that there is one data point that is out there by itself. The name of the person for that data point is "TOTAL". It is obviously a data entry error. So this data point is removed from the dataset.

```
ipdb>
```



With TOTAL out of the dataset, I visualize the data again with Outlier.check2.py to look for data points that may still look suspicious. The output of the script is shown below.



Since Ken Lay and Jeff Skilling are top officials of the company and also POIs in the heart of scandal, these are probably valid data points. The search for outliers stops here.

Feature Engineering

I use my intuition to select the following features and test them with machine learning algorithms.

stk_pay_ratio: The ratio of an individual's total stock value and total payments. Total payments include salary, bonus, long term incentive, deferred income, deferred payments, loan advances, other, expenses and director fees. Total stock value includes exercised stock options, restricted stock and restricted stock deferred. Since Enron's culture was to do anything, legal or otherwise, to boost its stock price and the POIs were known to engineer innovative but questionable ways to accomplish this goal, it was likely that they got a disproportionately high percentage of their rewards from stock than from salary and other payments.

bonus_salary_ratio: The ratio of an individual's bonus and salary. The idea is similar to above with the caveat that bonus is a short-term reward while total stock value is longer term.

to_poi_ratio: The ratio of "from_this_person_to_poi" and "from_messages". The idea is that a POI is more likely to send a disproportionately higher percentage of emails to other POIs in order to create, plan and coordinate ways to game the system.

'from_poi_ratio': The ratio of "from_poi_to_this_person" and "to_mesages". Same reasoning applies as the ratio above.

'from_cross_to': The product of from_poi_ratio and to_poi_ratio. The idea is that if the from_poi_ratio and the to_poi_ratio are good features, the product could be even better.

'from_squared': The feature 'from_poi_ratio' squared - similar to the 'from_cross_to' idea above.

Feature Selection Using Naïve Bayes Classifier

The process used was to run NBC with different combinations of the engineered features. 'poi_id.py' is a workflow script that allows me to do the following:

- load the data
- remove outliers
- engineer features
- select features through a features_list
- instantiate a classifier
- tune the classifier with different parameters
- test the classifier with the data using stratified k-fold shuffle splits
- print the evaluation metrics for each feature set

Engineered features described above are added one at a time to the features_list and tested.

The features that produce the best precision and recall using Naïve Bayes Classifier (NBC) are: stock_pay_ratio, from_poi_ratio and to_poi_ratio. They produce better precision, recall, F1 and F2 scores than any other feature sets. Features are not scaled because NBC does not need scaling.

The screen shot of the NBC classifier metrics is provided below:

```
ipdb> GaussianNB()
        Accuracy: 0.22764       Precision: 0.17311      Recall: 0.86000 F1: 0.28820     F2: 0.47948
        Total predictions: 11000        True positives: 1720    False positives: 8216   False negatives:  280   True negatives:  784

GaussianNB()
        Accuracy: 0.69167       Precision: 0.30306      Recall: 0.65400 F1: 0.41419     F2: 0.53102
        Total predictions: 12000        True positives: 1308    False positives: 3008   False negatives:  692   True negatives: 6992

GaussianNB()
        Accuracy: 0.69792       Precision: 0.30678      Recall: 0.64500 F1: 0.41579     F2: 0.52847
        Total predictions: 12000        True positives: 1290    False positives: 2915   False negatives:  710   True negatives: 7085

GaussianNB()
        Accuracy: 0.72875       Precision: 0.30531      Recall: 0.49200 F1: 0.37679     F2: 0.43839
        Total predictions: 12000        True positives:  984    False positives: 2239   False negatives: 1016   True negatives: 7761

GaussianNB()
        Accuracy: 0.72325       Precision: 0.30476      Recall: 0.51550 F1: 0.38306     F2: 0.45287
        Total predictions: 12000        True positives: 1031    False positives: 2352   False negatives:  969   True negatives: 7648

GaussianNB()
        Accuracy: 0.70167       Precision: 0.29134      Recall: 0.55150 F1: 0.38127     F2: 0.46793
        Total predictions: 12000        True positives: 1103    False positives: 2683   False negatives:  897   True negatives: 7317
```

Tuning Decision Tree Classifier (DTC)

Now keeping the above feature set, I tuned the min_samples_split and min_samples_leaf parameters. I didn't scale these features because unlike regression classifiers, Decision Tree Classifier is not affected by data scaling.

Decision trees are prone to over-fitting. Min_samples_split sets the minimum number of samples required to continue building the tree. Min_samples_leaf sets the minimum number of samples in a leaf node and stop tree building if the number pre-split falls is below the minimum. They are both used to reduce the number of splits, control the number of tree levels and prevent over-fitting.

DTC and other classifiers are prone to over-fitting. By tuning the classifier, one can control over-fitting. The performance of many algorithms is affected by the tuning parameters. With DTC in this project, recall changes from 0.28 to 0.38. Therefore, it is important to tune the parameters.

In 'poi_id.py', 2 do loops are used to iterate over different values for these two parameters. The function test_classifier runs the DTC. 'print_feature_importances' output the importance of the 4 features on the console. 'bonus_salary_ratio' appears to be the most important feature for best tuned DTC. A screen shot of the result is shown below.

```
DecisionTreeClassifier(compute_importances=None, criterion='gini',
            max_depth=None, max_features=None, max_leaf_nodes=None,
            min_density=None, min_samples_leaf=1, min_samples_split=2,
            random_state=None, splitter='best')
        Accuracy: 0.79550       Precision: 0.38371      Recall: 0.37450 F1: 0.37905     F2: 0.37631
        Total predictions: 12000        True positives:  749    False positives: 1203   False negatives: 1251   True negatives: 8797

feature_importances
['poi', 'stk_pay_ratio', 'to_poi_ratio', 'from_poi_ratio', 'bonus_salary_ratio', 'from_cross_to', 'from_squared']
[ 0.19034033  0.06772575  0.05497039  0.26649413  0.40701832  0.01345109]

DecisionTreeClassifier(compute_importances=None, criterion='gini',
            max_depth=None, max_features=None, max_leaf_nodes=None,
            min_density=None, min_samples_leaf=2, min_samples_split=2,
            random_state=None, splitter='best')
        Accuracy: 0.79733       Precision: 0.36136      Recall: 0.28150 F1: 0.31647     F2: 0.29452
        Total predictions: 12000        True positives:  563    False positives:  995   False negatives: 1437   True negatives: 9005

feature_importances
['poi', 'stk_pay_ratio', 'to_poi_ratio', 'from_poi_ratio', 'bonus_salary_ratio', 'from_cross_to', 'from_squared']
[ 0.09448809  0.0138873   0.10424696  0.48686836  0.28044986  0.02005944]

DecisionTreeClassifier(compute_importances=None, criterion='gini',
            max_depth=None, max_features=None, max_leaf_nodes=None,
            min_density=None, min_samples_leaf=1, min_samples_split=4,
            random_state=None, splitter='best')
        Accuracy: 0.79783       Precision: 0.37815      Recall: 0.33050 F1: 0.35272     F2: 0.33904
        Total predictions: 12000        True positives:  661    False positives: 1087   False negatives: 1339   True negatives: 8913

feature_importances
['poi', 'stk_pay_ratio', 'to_poi_ratio', 'from_poi_ratio', 'bonus_salary_ratio', 'from_cross_to', 'from_squared']
[ 0.18341498  0.07120879  0.07194029  0.28598526  0.38745069  0.        ]

DecisionTreeClassifier(compute_importances=None, criterion='gini',
            max_depth=None, max_features=None, max_leaf_nodes=None,
            min_density=None, min_samples_leaf=2, min_samples_split=4,
            random_state=None, splitter='best')
        Accuracy: 0.79925       Precision: 0.36781      Recall: 0.28450 F1: 0.32083     F2: 0.29800
        Total predictions: 12000        True positives:  569    False positives:  978   False negatives: 1431   True negatives: 9022

feature_importances
['poi', 'stk_pay_ratio', 'to_poi_ratio', 'from_poi_ratio', 'bonus_salary_ratio', 'from_cross_to', 'from_squared']
[ 0.09448809  0.0138873   0.04413076  0.45276732  0.33461033  0.0601162 ]
```

Better Classifier

Compared to DTC, Naïve Bayes is a better classifier. It has a precision of 0.307 and a recall of 0.645. The best tuned tree produces a precision of 0.384 and a recall of 0.374. NBC has 0.416 and 0.528 versus DTC with 0.379 and 0.376 as the best average F1 and F2 score.

Validation

Validation is to test the model in the training phase against data held out. The data model is made up of features selected, the type of classifier and its tuning parameters. Typically, a dataset is divided into 3 groups: training, validation and testing. The model is trained by machine learning algorithms with the training set, validated on the validation set and tested on the test set.

Validation is important because it helps avoid over-fitting and it provides an estimate of the model's performance on an independent dataset. Without validation, one won't know how well does the model applies to an independent dataset. If done wrong, one could falsely believe that a model has good performance metrics when it actually fails to predict unseen test examples.

Since the dataset contains only a 145 examples, we want to maximize the number of training examples without over-fitting and still get an accurate evaluation of the model's performance. In this situation, the testing step is omitted completely since validation can be viewed as testing with held out data. Furthermore, stratified shuffle split cross validation is used.

Stratified shuffle split cross validation is a combination of stratified k-fold cross validation and ShuffleSplit. In k-fold cross validation, the original dataset is randomly portioned into k equal size subsamples. Of the k subsamples, a single subsample is retained as the validation data and the remaining k-1 subsamples are used as training data[1]. In stratified k-fold cross validation, the folds are selected so that the number of examples in each class (POI and non-POI) is about the same in all the folds.

The ShuffleSplit iterator generates a user-defined number of independent train/validation splits. Examples are shuffled randomly and then split into train and validation sets[2].

StratetiedShufflesSplit is available in the sklearn package to combine stratified k-fold cross validation and shufflesplit in one function. The test_classifier function in tester.py calls this function with a fold of 1000. As a result, the script trains and tests the model in 1000 randomly constructed train-test cycles, each generating 12 predictions for a total of 12000 predictions. These predictions are compared with the labels in the test set to construct the confusion tables and the F scores.

1. http://en.wikipedia.org/wiki/Cross-validation_(statistics)
2. http://scikit-learn.org/stable/modules/cross_validation.html#random-permutations-cross-validation-a-k-a-shuffle-split

Evaluation

The average precision and recall of the NBC using stk_pay_ratio, to_poi_ratio and from_poi_ratio as features is 0.307 and 0.645 respectively.

Precision is the ratio of two averages. It is the number of Enron employees who are POIs and are identified by the model as such over the total number of employees who are identified as POIs, correctly or otherwise. Those that are POIs and identified by the model as such are true positives. That is the numerator. The denominator is made up of employees that are correctly identified as POI and also people who are incorrectly identified as POIs. These examples for which the model incorrectly flags as POIs are false positives.

Recall is ratio of the number of POIs that are correctly identified by the model as such over the total number of POIs in all the subsamples. The numerator is the number of true positives. The denominator is made up of two components: true positive and those who are POIs but failed to be identified as such by the model (false negatives).

The NBC is pretty good in recall but not so much in its precision. That means it is better at identifying a person as a non-POI than when it assigns someone to the POI category. On average, the classifier will falsely identify a person as a POI more often than falsely identifying a person as a non-POI. In my opinion, this is a useful classifier at the early phase of an investigation. At this phase, it is better to err on

the “over zealous” side to investigate more people so as to avoid letting the guilty ones off the hook, knowing that if the model says a person is not a POI, it is less likely that the model is wrong.

References

- http://docs.scipy.org/doc/numpy/user/basics.indexing.html
- http://stackoverflow.com/questions/25217510/how-to-see-top-n-entries-of-term-document-matrix-after-tfidf-in-scikit-learn
- http://scikit-learn.org/stable/documentation.html