DAND PROJECT

Identify Fraud from Enron Email

By Alex Mathew 19/10/2017

Project Overview

In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives. In this project, you will play detective, and put your new skills to use by building a person of interest identifier based on financial and email data made public as a result of the Enron scandal. To assist you in your detective work, we've combined this data with a hand-generated list of persons of interest in the fraud case, which means individuals who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity.

I intend build a person of interest (POI) identifier based on financial and email data part of the dataset from the Enron scandal. We have the email and financial data for 146 executives at Enron to identify persons of interest in the fraud case. A person of interest (POI) is was guilty of fraud or settled. This report documents the machine learning techniques used in building a POI identifier.

Enron Submission Free-Response 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?

Ans. The goal of this project is to build a predictive model that can identify POI based on features included in the Enron dataset. Such model could be theoretically used to find suspects who were not indicted during the original investigation, or to find persons of interest during fraud investigations at other businesses.

The dataset contains a total of **146 data points** with **21 features**. Of the 146 records, **18 are labeled as POI.**

Missing Data for Each Feature:

salary 51
to_messages 60
deferral_payments 107
total_payments 21
exercised_stock_options 44
bonus 64
restricted_stock 36
shared_receipt_with_poi 60
restricted_stock_deferred 128
total_stock_value 20
expenses 51
loan_advances 142
from_messages 60
other 53
from_this_person_to_poi 60
poi 0
director_fees 129
deferred_income 97
long_term_incentive 80
email_address 35
from_poi_to_this_person 60

Drawing a plot using bonus and salary, I found an outlier 'TOTAL' which should not be part of data here. Also removed THE TRAVEL AGENCY IN THE PARK as it is not an individual.

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.

Ans. I used scikit-learn SelectKBest to select best influential features. I used a list of many features to narrow down the best 5 features which had the higher scores.

This I narrowed them down to :

Features	**Score**
salary	18.28
bonus	20.79
deferred_income	11.45
total_stock_value	24.182
exercised_stock_options	24.815

I made two features: **Fraction_from_poi (5.12)** and **fraction_to_poi (4.09)**

I did this based on the idea that POI's must have higher interaction with POI's as compared to others. However these features got low scores using the selectKbest and hence I didn't use them further onwards. I did use MInMaxScaling as the units and ranges of the various features are diverse.

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

Ans. I tested three algorithms

GaussianNB*(priors=None)*

Accuracy: 0.85464 **Precision: 0.48876** **Recall: 0.38050** F1: 0.42789

F2: 0.39814 Total predictions: 14000 True positives: 761 False positives: 796

False negatives: 1239 True negatives: 11204

KNeighborsClassifier*(algorithm='auto', leaf_size=5, metric='minkowski',*

metric_params=None, n_jobs=1, n_neighbors=10, p=3, weights='distance')

Accuracy: 0.85793 **Precision: 0.52174** **Recall: 0.06600** F1: 0.11718

F2: 0.07997 Total predictions: 14000 True positives: 132 False positives: 121

False negatives: 1868 True negatives: 11879

KMeans*(algorithm='auto', copy_x=True, init='k-means++', max_iter=300, n_clusters=2, n_init=10, n_jobs=1, precompute_distances='auto', random_state=None, tol=0.001, verbose=0)*

Accuracy: 0.79271 **Precision: 0.28234** **Recall: 0.29250** F1: 0.28733

F2: 0.29041 Total predictions: 14000 True positives: 585

False positives: 1487 False negatives: 1415 True negatives: 10513

KNeighboursClassifer got the maximum Precision value of 52% but a low recall score.

Gaussian Naive Bayes got a precision and Recall score of above 0.3, ie 0.48 and 0.38 respectively.

So Naive Bayes is better overall.

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? What parameters did you tune?

Parameters tuning refers to the adjustment of the algorithm when training, in order to improve the fit on the test set. Parameter can influence the outcome of the learning process, the more tuned the parameters, the more biased the algorithm will be to the training data & test harness. The strategy can

be effective but it can also lead to more fragile models & overfit the test harness but don't perform well in practice.

I tried to tune the KMeans classifier with minimal effect. I managed to increase the accuracy and recall by a very little amount.

```
KMeans(copy_x=True, init='k-means++', max_iter=500, n_clusters=2, n_init=20,
      n_jobs=1, precompute_distances='auto', random_state=None, tol=0.005,
     verbose=0)
```

Following parameters are used to tune an algorithm:

- Select K Best (SelectKBest) : k
- Gaussian Naive Bayes (`GaussianNB`) : None

You can configure *k*-Means by specifying any of the following:

- Number of clusters
- Growth factor for memory allocated to hold clusters
- Convergence tolerance
- Distance Function. The default distance function is Euclidean.
- Split criterion. The default criterion is the variance.
- Number of iterations for building the cluster tree.

Tuned Parameters

max_iter : 300 -> 500

n_init : 10 - >20

tol : 0.001 -> 0.005

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

Ans. Validation allows us to assess how well the chosen algorithm generalizes beyond the dataset used to train it—that is, identify the risk of overfitting.

One of the biggest mistakes one can make is therefore to use the same data for training and testing.

In order to validate my analyses, I used the cross validation using StratifiedShuffleSplit part of the given tester.py file through test_classifier

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.

Ans.

As my evaluation metric,I made use of the precision_score , recall_score, and accuracy

As we know,

- ⇨ precision = number of true positives / (number of true positives + number of false positives

Ie % of correctly identified POIs.

I got a precision score of 0.52 with K-neighbours and 0.48 with naïve bayes.

- ⇨ recall = number of true positives / (number of true positives + number of false negatives)

ie. Proportion of all POIs identified correctly

- ⇨ Accuracy of my algorithm with a score of 0.85 means that my model will be 85% correct in identifying POIs or not.