Identifying Fraud at Enron

Dusan Punosevac

1. Project Overview

Abstract

Enron Corporation was an American energy, commodities, and service company based in Houston, Texas. It was founded in 1985 because of a merger between Houston Natural Gas and InterNorth, both relatively small regional companies. Enron employed approximately 20,000 staff and was one of the world's major electricity, natural gas, communications and pulp and paper companies, with claimed revenues of nearly $101 billion during 2000.
Fortune named Enron "America's Most Innovative Company" for six consecutive years.

At the end of 2001, it was revealed that its reported financial condition was sustained by institutionalized, systematic and creatively planned accounting fraud, known since as the *Enron scandal*. Enron has since become a well-known example of willful corporate fraud and corruption.
Enron filed for bankruptcy in late 2001. It ended its bankruptcy during November 2004, pursuant to a court-approved plan of reorganization.

Project goal is to create machine learning application used to identify Enron Employees who may have committed fraud based on the public Enron financial and email dataset.

Data Exploration

Total number of data points	146
Number of POIs	18
Number of non-POIs	128
Number of features in the dataset	21

Existing features in the dataset are:

- salary
- to_messages
- deferral_payments
- total_payments
- exercised_stock_options
- bonus
- restricted_stock
- shared_receipt_with_poi
- restricted_stock_deferred
- total_stock_value
- expenses
- loan_advances
- from_messages
- other
- from_this_person_to_poi
- poi
- director_fees
- deferred_income
- long_term_incentive
- email_address
- from_poi_to_this_person

Features with many missing values (> 50% of NaN values) are:

- deferral_payments (73.288%)
- restricted_stock_deferred (87.671%)
- loan_advances (97.260%)
- director_fees (88.356%)
- deferred_income (66.438%)
- long_term_incentive (54.795%)

Finally, the dataset is very imbalanced between the two classes
18 persons (12.33%) of dataset is classified as *POI*. Other 128 (87.67%) is classified as non *POI*s. By combining this information, I realised my data is suffering from *Class Imbalance Problem*. Because of this problem, we will focus on Precision and Recall instead of accuracy. That means we will focus on F1 score, and have higher focus on good precision and recall metrics on the *POI*s, since this is the class that we have less than 15% in data.

Outliers

In the financial data, there is outlier *"TOTAL"*, and we should remove it since it is sum of all financial data.
There is *"LOCKHART EUGENE E"* which has all "**NaN**" values. We want to remove this record, since it has no value for us.
The third outlier is *"THE TRAVEL AGENCY IN THE PARK"* since this is not person, but agency.

I implemented a function that use LOF (Local Outlier Factor)[1] estimator to remove other outliers. LOF has several parameters, I used the "contamination" parameter (i.e. "The amount of contamination of the data set, i.e. the proportion of outliers in the data set. When fitting, this is used to define the threshold on the decision function."[2]) to control the number of outliers that will be detected. I find out that the default parameter (0.1) delete 15 entries. I decided to remove fewer employees, I decreased the "contamination" parameter to 0.02, with this value only 3 entries are removed, "*UMANOFF ADAM S*", "*LAY KENNETH L*" and "*MORAN MICHAEL P*".
I tried other outliers detectors[3] like Isolation Forest[4], they lead to similar results of the LOF estimator.
However, my further investigation shows that "*UMANOFF ADAM S*" was CEO and president of Enron Wind Corporation, "*LAY KENNETH L"* was a CEO and chairman of the *Enron* and finally "*MORAN MICHAEL P*" who was also one of the chairman of *Enron*. After this I realized that this 3 guys are shown as outliers since they were getting significant amount of money since they were high operatives in the company. I deduced that excluding them would be bad for classifier since I believe that they might be the keys in our prediction system.

[1] http://scikit-learn.org/stable/auto_examples/neighbors/plot_lof.html
[2] http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html#sklearn.neighbors.LocalOutlierFactor
[3] http://scikit-learn.org/stable/modules/outlier_detection.html
[4] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html#sklearn.ensemble.IsolationForest

2. Feature selection

Implemented 2 new features, "*from_poi_ratio*" and "*to_poi_ratio*".
This 2 features represent the ratio of messages sent and received from POIs.

We decided to split data on train and test set using train_test_set_split from model selection, but using stratify which garanties that we have same percent of both classifiers in our training and testing samples.

Data normalization (MinMaxScaling) was done on features in order to change the amplitude and effect of some features that have higher values than other.

We choosed 7 features using SelectKBest. This is the outcome:

Without PCA:

Feature name	Score
'bonus'	23.332
'salary'	22.725
'total_stock_value'	20.694
'exercised_stock_options'	20.620
'from_poi_ratio'	20.206
'deferred_income'	10.533
'total_payments'	10.354

Using PCA:

Feature name	Score
'salary'	31.463
'restricted_stock_deferred'	5.519
'shared_receipt_with_poi'	4.838
'exercised_stock_options'	2.543
'to_messages'	2.049
'bonus'	1.842
'total_stock_value'	0.953

As you can see, after Principal component analysis (PCA) salary has higher score, and other features got less impact. However, this proved to be ideal for our algorithms and evaluation metrics.

3. Pick an algorithm

After examining dataset, features and problem there were 4 algorithms chosen for the try out.

- Support Vector Machines (SVC)
- DecissionTreeClassifier
- AdaBoostDecisionTreeClassifier
- KNeighborsClassifier

We did measure these algorithms, with and without use of PCA.
At the end, there were 2 algorithms which gave really good results in combination **with PCA**, based on our metrics. These algorithms are:

- *AdaBoostClassifier(DecisionTreeClassifier(min_samples_split=100), algorithm="SAMME")*
- *KNeighborsClassifier(n_neighbors=5, weights="distance", algorithm="auto")*

You can read more about evaluation metrics, results and which one we decided to use and why in 5th chapter (Validation, evaluation metrics and common pitfalls).

4. Tuning algorithm parameters

Parameter tuning is really important thing that you should do when you make your own machine learning. Parameter tuning can improve quality of your classification and depending on some parameters, change results completely for performing poorly, to good
Used GridSearchCV to tune up parameters of tried algorithms. Also, using algorithm parameter "auto" in KNeighborsClassifier to decide which algorithm is the best one for given training set. Tuning algorithm parameters is really important task, since if we are not tuning our classifier, we can get into situation that our classifier perform poorly.

5. Validation, evaluation metrics and common pitfalls

Validation is really important part of machine learning. It is mechanism that enables us to validate how actually good our classifier and model is. The common mistake is that when people think about classification model and machine learning in general, their first intuition is to check **accuracy** metric. Sometimes it is more important to have better ratio of precision and recall, known as f1-score and sacrifice part of your accuracy in order to achieve this. That happened in our case, since we didn't hunt only for good accuracy but for good recall and precision among our smaller part of dataset, our *POI*s.

Our dataset suffers from *Imbalanced Class Problem*, as we mentioned before. Because of this, we decided to focus on Precision and Recall (F1 score) of *POI*s. But instead of looking on average value of this, we want to pay additional attention to this value on *POI*s classes. By our definition, the good algorithm is the one that has good Precision and Recall of *POI*s, good accuracy and good Precision and Recall of the non *POI*s.

One more common mistake in validation metrics is that people **tend to use whole dataset**, **instead of splitting the set into training and test set**. If you do not do this, you cannot validate your model correctly and tune your algorithm. Also you are **prone to overfitting** the data.

Results

This is the results of tuned parameters for algorithms without PCA:

SVC: (C=10.0, gamma=0.001) without PCA				
poi	precision	recall	f1-score	support
0 (non POIs)	0.86	1.00	0.93	25
1 (POIs)	0.00	0.00	0.00	4
avg / total	0.74	0.86	0.80	29

DecisionTreeClassifier(min_samples_split=100) without PCA				
poi	precision	recall	f1-score	support
0 (non POIs)	0.87	0.80	0.83	25
1 (POIs)	0.17	0.25	0.20	4
avg / total	0.77	0.72	0.75	29

AdaBoostClassifier(DecisionTreeClassifier(min_samples_split=100), algorithm="SAMME") without PCA				
poi	precision	recall	f1-score	support
0 (non POIs)	0.88	0.84	0.86	25
1 (POIs)	0.20	0.25	0.22	4
avg / total	0.78	0.76	0.77	29

KNeighborsClassifier(n_neighbors=10, weights="uniform", algorithm="auto") without PCA				
poi	precision	recall	f1-score	support
0 (non POIs)	0.86	1.00	0.93	25
1 (POIs)	0.00	0.00	0.00	4
avg / total	0.74	0.86	0.80	29

This is the results with PCA:

SVC: (C=10.0, gamma=0.001) with PCA				
poi	precision	recall	f1-score	support
0 (non POIs)	0.85	1.00	0.92	23
1 (POIs)	0.00	0.00	0.00	4
avg / total	0.73	0.85	0.78	27

DecisionTreeClassifier(min_samples_split=100) with PCA				
poi	precision	recall	f1-score	support
0 (non POIs)	0.95	0.78	0.86	23
1 (POIs)	0.38	0.75	0.50	4
avg / total	0.86	0.78	0.80	27

AdaBoostClassifier(DecisionTreeClassifier(min_samples_split=100), algorithm="SAMME") with PCA				
poi	precision	recall	f1-score	support
0 (non POIs)	0.95	0.87	0.91	23
1 (POIs)	0.50	0.75	0.60	4
avg / total	0.89	0.85	0.86	27

KNeighborsClassifier(n_neighbors=10, weights="uniform", algorithm="auto") with PCA				
poi	precision	recall	f1-score	support
0 (non POIs)	0.92	0.96	0.94	23
1 (POIs)	0.67	0.50	0.57	4
avg / total	0.88	0.89	0.88	27

Seeing the results, we came to the conclusion that our *AdaBoostDecisionTreeClassifier* and *KNeighborsClassifer* are 2 of the best, since their precision and recall are really good, especially for the *POI*s which are really our focus. At the end, we decided to use *KNeighborsClassifier* since it **had better avg/total f1 score**.

6. Summery

In this project, you got to know more about Enron Corporation and it's fraud. You went through data exploration, checking what the dataset is in this project and what outliers are and how it can affect our classification model.
You could see the normalization of the data, what is features scaling and PCA, and how it affects classification model results.
We went through picking of the algorithm, tuning it's parameters and creating validation and evaluating metrics.

Final results are that we used *KNeighborsClassifer* which gave us the best performances.
The last test_classifier method which is provided in tester.py gave us this results:

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=5, p=2, weights='distance')
Accuracy: 0.88471 Precision: 0.66812 Recall: 0.38350 F1: 0.48729 F2: 0.41922
Total predictions: 14000 True positives: 767 False positives: 381
False negatives: 1233 True negatives: 11619