Enron Case Study

Author-Rajdeep Shil

Introduction

The story of Enron Corporation depicts a company that reached dramatic heights only to face a dizzying fall. The fated company's collapse affected thousands of employees and shook Wall Street to its core. At Enron's peak, its shares were worth $90.75; just prior to declaring bankruptcy on Dec. 2, 2001, they were trading at $0.26. 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.

Data exploration

Question 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”, “outlier investigation”]

Answer 1:

In this project,I played the detective, and put my machine learning skills to use by building a classifier algorithm to identify Enron Employees who may have committed fraud based on the public Enron financial and email dataset. Where such large surplus of data are available, simple machine learning codes can help in narrowing in identifying the culprits and can fasten the investigation process.

In this section, I will explain the methods I used to deploy various coding functions to determine the number of data points, the number of features, determining outliers, creation of new features from existing features.

Using the poi.py given, I just queried the following lines for printing out the number of data points and features.

```
print 'Total number of datapoints are {}'.format(len(name_data_point))
```

```
Total number of datapoints are 146
```

```
#Number of features
total_features=len (data_dict[name_data_point[0]])
print 'Total number of features are {}'.format(total_features)
```

```
Total number of features are 21
```

As the data was imported in a form of nested dictionary, it was difficult to do any kind of cleaning without iterating through all the elements in the nested dictionary. I thought of using the pandas module which I learned in the first lesson of Data visualization. Pandas dataframe gives flexibility in exploring a few statistical informations like mean, standard deviation with a single line of code. After cleaning the pandas dataframe can easily be converted back to the original nested dictionary format which is pre-requisite for the tester module. This dataset can be divided into two broad categories a) Financial data b) Email data. Please see the below figure:

```
<class 'pandas.core.frame.DataFrame'>
Index: 146 entries, METTS MARK to GLISAN JR BEN F
Data columns (total 16 columns):
poi                        146 non-null float64
salary                     95 non-null float64
to_messages                86 non-null float64
deferral_payments          39 non-null float64
total_payments             125 non-null float64
exercised_stock_options    102 non-null float64
bonus                      82 non-null float64
restricted_stock           110 non-null float64
shared_receipt_with_poi    86 non-null float64
total_stock_value          126 non-null float64
expenses                   95 non-null float64
from_messages              86 non-null float64
from_this_person_to_poi    86 non-null float64
deferred_income            49 non-null float64
long_term_incentive        66 non-null float64
from_poi_to_this_person    86 non-null float64
dtypes: float64(16)
memory usage: 19.4+ KB
None
```

As from the figure above we can see there are a lot of NaN values present in the dataset.
I thought of choosing carefully the features that made the most sense to me. From initial intuition few features that were not considered are provided in the unwanted_feature list (These features would later be updated and deleted from the original feature_list).
unwanted_features=["poi","email_address"]
I removed and replaced the NaN with 0. However we can replace the financial features with **mean or media** value as well.

Cleaning of Outliers

As far as outliers are concerned, from the course module I already knew about “Total”. I investigated a little more which didn’t make sense to me.
I made a list of all financial features and removed the name where all the values were coming out to be zero. The name of the person is shown below:

```
temp_df[financial_features].loc[(temp_df[financial_features]==0).all(axis=1)].index
# found this one outliers with all finantial features as zero, hence it can be dropped
```

```
Index([u'LOCKHART EUGENE E'], dtype='object')
```

I found another outlier which had a weird name which also had almost all financial features with zero values.

```
temp_df[financial_features].loc['THE TRAVEL AGENCY IN THE PARK']
```

```
salary                          0.0
total_payments             362096.0
long_term_incentive             0.0
exercised_stock_options         0.0
bonus                           0.0
restricted_stock                0.0
total_stock_value               0.0
expenses                        0.0
deferred_income                 0.0
deferral_payments               0.0
Name: THE TRAVEL AGENCY IN THE PARK, dtype: float64
```

Hence **LOCKHART EUGENE E** and **THE TRAVEL AGENCY IN THE PARK** were also dropped from the dataset.
After cleaning I was left with **143 data points**.

Feature Engineering

Question 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”, “intelligently select features”, “properly scale features”].

Answer 2:

From the course module on feature selection, I got the hint of creating two features from the existing email features from the data set. I engineered one myself. The three engineered parameters were :

1. **fraction_from_poi:** fraction of from poi to this person("from_poi_to_this_person")/ total messages("to_messages")
2. **fraction_to_poi:**fraction of to poi from this person("from_poi_to_this_person")/ total messages("from_messages")
3. **total_profit:** Sum of bonus , salary and incentives. From Initial investigation poi had a significantly higher sum of all the three.

Except email_id, I have used all the engineered features as well as the original financial and email features present in the dataset.

My feaure_list looked like:

['poi',
'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',
'director_fees',
'deferred_income',
'long_term_incentive',
'from_poi_to_this_person',
'fraction_from_poi',
'total_profit',
'fraction_to_poi']

To select the best features, as taught in the class module I used **SelectKbest**.

Impact of new features

I have used all the features above (old existing features and new engineered features) in all the algorithms.

However to differentiate the impact of new engineered features in poi identification, I ran the classifier with old existing features as well.

I have selected Decision Tree Classifier for poi-detection, reasons of which are explained in detail in subsequent answers.

When I ran the decision tree classifier with old existing features: Used Pipeline for selecting best features before running classifier.

Without New features

Pipeline(memory=None,
steps=[('SKB', SelectKBest(k=13, score_func=<function f_classif at 0x000000000E2B2748>)), ('tree', DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=5,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=0,splitter='best'))])

CONFUSION MATRIX for Trees without New features

	pred_neg	pred_pos
neg	**30**	**8**
pos	**2**	**3**

--------Accuracy of Decision Tree without new features----------
0.7674418604651163
----------Precision of Decision Tree without new features ----------
0.2727272727272727

----------Recall of Decision Tree without new features ----------
0.6

--------The f1-score of Decision Tree without new features------:
0.37499999999999994

With New features

Pipeline(memory=None,
steps=[('SKB', SelectKBest(k=19, score_func=<function f_classif at 0x000000000DE84C18>)), ('tree', DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=5,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=0,
splitter='best'))])

CONFUSION MATRIX for Trees With New Features

	pred_neg	pred_pos
neg	36	2
pos	1	4

---------Accuracy of Decision Tree With New Features----------
0.9302325581395349

----------Precision of Decision Tree With New Features ----------
0.6666666666666666
----------Recall of Decision Tree With New Features ----------
0.8
--------The f1-score of Decision Tree With New Features-----:
0.7272727272727272

Cleary new engineered features have helped me significantly in increasing the precision of capturing the culprits.

Classifier used in this study are:

- SVC
- Naive Bayes Gaussian
- Adaboost
- Decision Trees

SVC:

In this classifier, I have used **PCA , MinMax scaler** and **GridSearchCV** to fine tune the parameters. SelectkBest was not used. PCA components were manually modified and the results were checked based on confusion matrix. To properly understand the importance of PCA see the confusion matrix of SVC with and without PCA. Other SVC components were tuned using the **GridSearchCV.**

SVC without PCA

CLASSIFICATION REPORT of SVC

	precision	recall	f1-score	support
0.0	0.88	1.00	0.94	38
1.0	0.00	0.00	0.00	5
micro avg	0.88	0.88	0.88	43
macro avg	0.44	0.50	0.47	43
weighted avg	0.78	0.88	0.83	43

CONFUSION MATRIX of SVC
[[38 0]
 [5 0]]
--------The f1-score of SVC classifer------:

0.0

Grid parameters without PCA:

SVC(C=1000.0, cache_size=200, class_weight=None, coef0=0.0,
 decision_function_shape='ovr', degree=3, gamma=0.0001, kernel='rbf',
 max_iter=-1, probability=False, random_state=None, shrinking=True,
 tol=0.001, verbose=False)

SVC with PCA

CLASSIFICATION REPORT of SVC with PCA

	precision	recall	f1-score	support
0.0	0.92	0.89	0.91	38
1.0	0.33	0.40	0.36	5
micro avg	0.84	0.84	0.84	43
macro avg	0.63	0.65	0.64	43
weighted avg	0.85	0.84	0.84	43

CONFUSION MATRIX of SVC with PCA
[[34 4]
 [3 2]]
--------The f1-score of SVC classifer with PCA------:
0.3636363636363636

Grid parameters with PCA:

SVC(C=10000.0, cache_size=200, class_weight=None, coef0=0.0,
 decision_function_shape='ovr', degree=3, gamma=0.01, kernel='rbf',
 max_iter=-1, probability=False, random_state=None, shrinking=True,
 tol=0.001, verbose=False)

Scaler

I also used the **minmax scaler** for the SVC classifier, as SVC is highly affected by the scales of features.

SVC with SCALER

CLASSIFICATION REPORT of SVC with scaler

	precision	recall	f1-score	support
0.0	0.90	0.95	0.92	38
1.0	0.33	0.20	0.25	5
micro avg	0.86	0.86	0.86	43
macro avg	0.62	0.57	0.59	43

weighted avg 0.83 0.86 0.84 43

CONFUSION MATRIX of SVC with with scaler
[[36 2]
[4 1]]
--------**The f1-score of SVC with scaler**------:
0.25

This study was done solely to understand the impact of PCA and scaler. After manually fine tuning it, and combining them I decided not to choose it, as the results were not satisfactory.

After using the minmax scaler in SVC , I did not use any scaler for the decision tree classifier (My selected algorithm) , or the remaining three classifiers. Scaler was not employed because it did not affect the performance of the classifiers.

Naive Bayes Gaussian
For Naive Bayes, and all the other three, I used Pipeline. This way I didn't select any features manually. All the features were fed to the pipeline, whose first step was **SelectKBest**. K=6 (Best number of features) was determined using SKB for Naive Bayes Gaussian. No feature scaling was performed as Naive bayes classifier performance is independent of feature scaling.

Adaboost:

K=10 (Best number of features) was determined using SKB for Adaboost with Decision Tree Classifier.
No feature scaling was performed as Adaboost performance is independent of feature scaling.

Decision Tree:

K=19 (Best number of features) was determined using SKB for trees.
No feature scaling was performed as Decision tree performance is independent of feature scaling.

Gridparemeters and scores of the last three algorithms are given in subsequent answers.

Algorithm Selection

Question 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”]

Answer 3:

As mentioned above I have used four classifiers.

- SVC
- Naive Bayes Gaussian
- Adaboost
- Decision Trees

SVC classifier performance and scores are mentioned in the previous question. Now for the performance of the remaining three

Gausian:

---------Accuracy of GNB----------
0.8837209302325582
----------Precision of GNB ----------
0.5
----------Recall of GNB ----------
0.6

--------The f1-score of GNB------:
0.5454545454545454

Adaboost:
--------Accuracy of ADABOOST----------

0.8372093023255814
----------Precision of ADABOOST ----------

0.3333333333333333
----------Recall of ADABOOST ----------

0.4
--------The f1-score of ADABOOST------:

0.3636363636363636

Decision tree:
---------Accuracy of Decision Tree----------
0.9302325581395349

----------Precision of Decision Tree ----------
0.6666666666666666

----------Recall of Decision Tree ----------
0.8

--------The f1-score of Decision Tree------:
0.7272727272727272

Clearly it was not an easy choice. I had to try out a variety of things before I could get a nice recall of **0.8 in the decision tree.** As our labels are skewed(High number of non-poi to very few poi), it was very important for the classifier to be able to predict poi accurately**.** Which means maximize true positive and minimize false negative**.**
Hence I chose **Decision tree as my classifier.**

Question 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? (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 items: “discuss parameter tuning”, “tune the algorithm”].

Answer 4:

Usually all classifiers have some default parameters. It drives how the classifier creates a **decision boundary to classify different labels**.
I will explain this with the example of my chosen classifier Decision Tree.

Decision Tree classifier default definition looks like:

class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort='deprecated', ccp_alpha=0.0)

Performance of tree with default parameter:

CONFUSION MATRIX for decision tree

	pred_neg	pred_pos
neg	34	4
pos	4	1

---------Accuracy of Decision Tree Default----------
0.813953488372093

----------Precision of Decision Tree Default ----------
0.2

----------Recall of Decision Tree Default ----------
0.2

--------The f1-score of Decision Tree Default------:
0.20000000000000004

As mentioned earlier I used pipeline, and pipeline definition looked like:

param_grid={"SKB__k":[2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20],'tree__criterion':['entropy'],'tree__min_samples_split':[2,5,7,10,12,15],'tree__max_depth':[1,2,3,4,5]}

Here are criteria that were tuned:
'tree__min_samples_split': specifies the minimum number of samples required to split an *internal node.*

'tree__max_depth': The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. Large numbers or too much depth can lead to bias.

After tuned parameters were used in the tree classifier:

CONFUSION MATRIX for Trees

	pred_neg	pred_pos
neg	36	2
pos	1	4

---------Accuracy of Decision Tree----------
0.9302325581395349

----------Precision of Decision Tree ----------
0.6666666666666666

----------Recall of Decision Tree ----------
0.8

--------The f1-score of Decision Tree------:
0.7272727272727272

Validation

Question 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 items: "discuss validation", "validation strategy"]

Answer 5:

I knew I had a very limited set of data to train this model. For validation of the classifier model the features and label data were split into **training sets** and **testing sets**. I divided the data such that 30% of the data is preserved for testing. If I use all the data points for training, obviously the classifier will identify the poi, however it would not be able to handle any new data. The scores mentioned in the above answers describe the performance of the classifiers against the tester sets.

In a classification setting, to ensure that the train and test sets have approximately the same percentage of samples of each target class I used **StratifiedShuffleSplit.** It is beneficial to

use stratified cross validation especially when response classes are unbalanced (Very few poi to large number of non-poi). .

The cross-validator **StratifiedShuffleSplit** was used with GridSearchCV to obtain the best hyper-parameters.

Question 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"]

Answer 6:

One of the first evaluation metrics I learned in the class was Accuracy which means total correct prediction divided by total overall prediction. Although accuracy is overall a very good estimator for the performance of the classifier, it does not hold much importance for data sets like Enron where one class of poi count is almost negligible to that of non-poi.
Which means even if the classifier is not able to predict any correct poi, the overall accuracy will still show a higher number which is a false indicator.
Hence, I decided to use Precision, recall and F1 to understand the performance of my classifier.
The confusion matrix for Decision Tree:

	pred_neg	**pred_pos**
Act_neg	36	2
Act_pos	1	4

TP: True Positive. (Actual positive and predicted positive)
FP: False positive (Actual negative and predicted positive)
TN: True Negative. (Actual negative and predicted negative)
FN: False Negative (Actual positive and predicted negative)
Precision: Attempts to answer the proportion of prediction are actually correct. (refer the above table

$$\text{Precision} = TP/(TP+FP) = 4/(2+4) = 0.667$$

Recall: Attempts to answer what proportion of positives were actually predicted correctly.

$$\text{Recall} = TP/(TP+FN) = 4/(4+1) = 0.8$$

F1: Strikes balance between precision and balance. A good classifier should have a higher f1 score.

f1= 2*(Precision * Recall) /(Precision + Precision)

f1=2*(0.667*0.8)/(0.667+0.8)

f1=0.72

APPENDIX

Using tester.py for different classifier:

Decision Tree:

steps=[('SKB', SelectKBest(k=19, score_func=<function f_classif at 0x000000000D74BE48>)), ('tree', DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=5,
max_features=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=0,
splitter='best'))])

Accuracy: 0.83253
Precision: 0.37149
Recall: 0.37000
F1: 0.37074
F2: 0.37030
Total predictions: 15000
True positives: 740
False positives: 1252
False negatives: 1260
True negatives: 11748

Adaboost:

Pipeline(memory=None,
steps=[('SKB', SelectKBest(k=10, score_func=<function f_classif at 0x000000000D74BE48>)), ('ADA', AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
learning_rate=1.0, n_estimators=50, random_state=0))])

Accuracy: 0.82887
Precision: 0.32181
Recall: 0.25600
F1: 0.28516
F2: 0.26692
Total predictions: 15000
True positives: 512
False positives: 1079
False negatives: 1488
True negatives: 11921