MACHINE LEARNING: ENRON POI IDENTIFIER

-KISHAN CHOUDHURY

9/17/2016

Playing a detective

The following is an analysis of the Enron dataset which is freely available in the open. Our objective is to find the suspects of fraud committed at Enron during its years of business.

MACHINE LEARNING: ENRON POI IDENTIFIER

PLAYING A DETECTIVE

Contents

Being a fan of Sherlock Holmes, I have always been obsessed with detective stories. Never had I thought in my wildest dreams that this field of data science would allow me to fulfill my secret wish. Let us begin.

1. INTRODUCTION

Enron Corporation was an American energy, commodities, and services company based in Houston, Texas. At the end of 2001, it was revealed that it's reported financial condition was sustained by institutionalized, systematic, and creatively planned accounting fraud, known since as the Enron scandal. This project employs Machine Learning to uncover the persons who might be involved in the fraud. Such people are known as Persons of Interest(POI).

The dataset includes financial features as well as a large set of emails. The emails are used to create a set of email features. Below is the list of features:

- 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

Outliers:
On exploring the data, we found an entry for an individual named "TOTAL". All the measures were very high for this particular entry. It is very clear that this must have been wrongly introduced during the data generation phase. We removed this entry from the dataset.

Since the number of data points is less, let us have a look at the names of the individuals to check if there are any obvious errors.
After going through the list , we see that there is an entry for **THE TRAVEL AGENCY IN THE PARK.** This obviously doesn't seem to be a person. Hence, we consider this as an outlier and delete it from the dataset.

Now, that the outliers have been taken care of, let's have a look at the basic statistics:

Total Number of Individuals(data points): 144
Number of POI:18

As we can see, the number of data points is very less. Moreover, the data is unbalanced i.e. the allocation to the classes POI and non-POI is not in the same range. There are a very few POI's. As a result of this, we need to carefully think about our validation strategy. Having separate train, test and validation sets will result in a very small training dataset thereby resulting in weaker model. Also, accuracy will not be a good metric to measure model performance. Let's discuss the validation strategy in detail in the validation section.

2. FEATURE ANALYSIS AND SELECTION

At first the above feature list was narrowed down based on intuition. The original email related features were removed and three new features were created.

- percentage_from_poi= from_poi_to_this_person/ to_messages
- percentage_to_poi= from_this_person_to_poi/ from_messages0
- percentage_shared_receipt_from_poi = shared_receipt_with_poi/ to_messages

These features were created because it made more sense to look at the ratio of emails related to poi than the absolute values.

SelectKBest function was used to find the scores for the various features.

Features	Score
exercised_stock_options	22.34897541
total_stock_value	22.51054909
Bonus	20.79225205
Salary	18.28968404
percentage_to_poi	16.40971255
deferred_income	11.42489149
long_term_incentive	9.92218601
restricted_stock	8.82544222
percentage_shared_receipt_from_poi	9.10126874
total_payments	9.28387362
loan_advances	7.18405566
Expenses	5.41890019
Other	4.2024363
percentage_from_poi	3.12809175
director_fees	2.13148399
deferral_payments	0.22885962
restricted_stock_deferred	0.76814634

Now that we have the scores for each of the features, how do we decide how many features to choose in order to get the best performance.

Also, going through the enron61702insiderpay.pdf , it seems the features total_payments and total_stock_value were derived from the rest of the sub features.
total_payments, which is a combination of:

- salary
- bonus
- director_fees
- deferral_payments
- deferred_income
- loan_advances
- long_term_incentive
- expenses
- other

total_stock_value, a combination of:

- restricted_stock
- exercised_stock_options
- restricted_stock_deferred

Let’s try to validate the data to make sure the calculations are correct.

On Cross-Checking, it seems there are problems with two individuals:

individuals with incorrect total_payments:
['BELFER ROBERT', 'BHATNAGAR SANJAY']
individuals with incorrect total_stock_value:
['BELFER ROBERT', 'BHATNAGAR SANJAY']

Let’s dig into enron61702insiderpay.pdf to check if the problems can be corrected.

Listing down the attribute values for the two individuals:
BELFER ROBERT

Feature Name	Pickle file	Pdf
salary	0	0
bonus	0	0
director_fees	3285	102500
deferral_payments	-102500	0
deferred_income	0	-102500
loan_advances	0	0
long_term_incentive	0	0
expenses	0	3285
other	0	0
total_payments	102500	3285
restricted_stock	0	44093
exercised_stock_options	3285	0
restricted_stock_deferred	44093	-44093
total_stock_value	-44093	0

BHATNAGAR SANJAY

Feature Name	Pickle file	Pdf
salary	0	0
bonus	0	0
director_fees	137864	0
deferral_payments	0	0
deferred_income	0	0
loan_advances	0	0
long_term_incentive	0	0
expenses	0	137864
other	137864	0
total_payments	15456290	137864
restricted_stock	-2604490	2604490
exercised_stock_options	2604490	15456290
restricted_stock_deferred	15456290	-2604490
total_stock_value	0	15456290

This seems to have been introduced due to human error. If there were a lot of problematic data points, it would not have been possible to check these manually but since we already have so few points, getting rid of these points does not seem to be a great idea. Let's try to correct these points in the dataset.

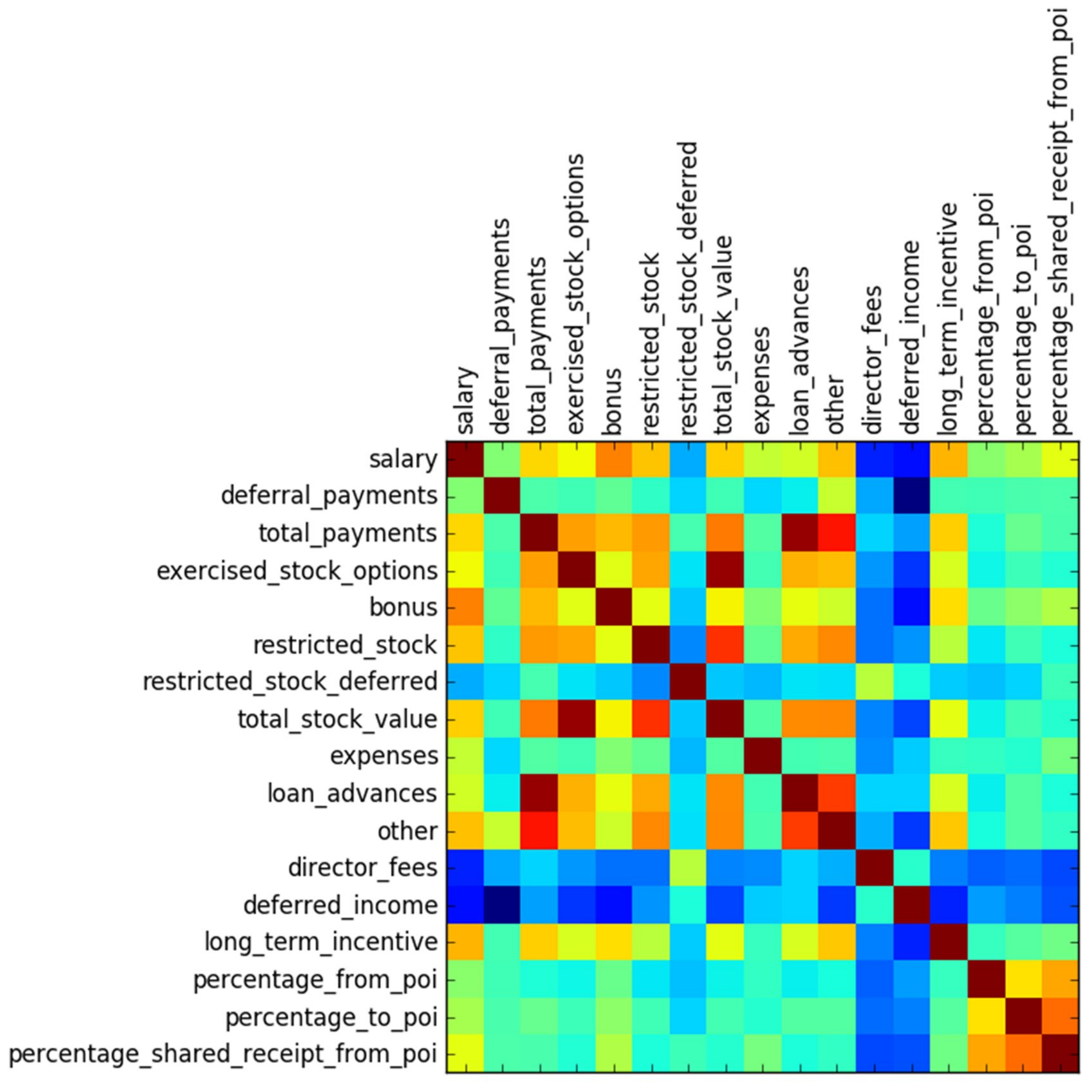
Now that the aggregated values have been corrected for all the points, another problem that we have to decide on is whether to include the aggregated features or the sub features, or to include both. The only way to be sure is to try out all the options. Please note that, since Machine Learning is an iterative process of gradually trying out new things, thereby, improving the model, basic level testing has been carried out before this step which suggests we narrow our focus to two algorithms:

- Naive Bayes
- Decision Trees.

So, we will restrict ourselves to the above mentioned two algorithms.

Test Description	Accuracy	Precision	Recall	F1
NB with All features	0.78793	0.29389	0.42100	0.34615
NB all features except Aggregated Values	0.65787	0.22303	0.63050	0.32950
NB with Aggregated Values and email features	0.85453	0.41078	0.20950	0.27748
DT with All features	0.80500	0.28112	0.29700	0.28884
DT all features except Aggregated Values	0.80907	0.29171	0.30250	0.29701
DT with Aggregated Values and email features	0.80453	0.24646	0.22650	0.23606

So, after trying out the various scenarios, it seems we cannot eliminate the detailed level features and just work with the aggregated features. We will have to rely on SelectKBest to find the best features. However, theoretically, the aggregated features should be correlated to the detailed level features that were used to create them. We can also use PCA to reduce the number of features. Let's have a look at the correlation matrix.



From the above correlation matrix, we can see darker shades for the aggregated features and each of the detail level features used to create them.

As we will see in the next section, since we narrowed down our analysis to Naive Bayes and Decision Trees, we do not need to perform any feature scaling. If we decide to use PCA for feature reduction, we shall need to normalize the features before applying PCA. However, in this particular scenario, it might be better to perform automated feature selection through SelectKBest.

3. ALGORITHM SELECTION

We tried using the below three algorithms:

- Naive Bayes
- SVM with rbf kernel
- Decision Trees

Let's have a look at their performance using all the features.

Classifier	Accuracy	Precision	Recall	F1
Naive Bayes	0.78793	0.29389	0.42100	0.34615
SVM with rbf kernel	No positives predicted			
Decision Tree	0.80500	0.28112	0.29700	0.28884

It seems Gaussian Naive Bayes and Decision Tree performs better compared to SVN.

4. TUNING THE ALGORITHM

Our ultimate goal in this project is to find the best algorithm to design a model and maximize its performance. Performance is measured using various performance metrics which we will be discussing later. One of the difficulties is that, learning algorithms (eg. decision trees, random forests, clustering techniques, etc.) require us to set parameters before we use the models (or at least to set constraints on those parameters). How we set those parameters can depend on a whole set of factors. That said, our goal, is usually to set those parameters to their optimal values that enable us to complete a learning task in the best way possible. Thus, tuning an algorithm is a machine learning technique that can be simply thought of as a process which one goes through in which they optimize the parameters that impact the model in order to enable the algorithm to perform the best.

Gaussian Naive Bayes:

In case of Gaussian Naive Bayes, there are no such parameters to be tuned.
So, the question is, now that we know the scores for the various features using SelectKBest, how many features should we use.

We will just use SelectKBest and try the Naive Bayes classifier with different number of features. This is done easily through pipeline and GridSearchCV

We are trying with 1-10 features and trying out to find the number of features resulting in best performance. Although this is done in one run through GridSearchCV, but to do a comparative study, let's list down the performance parameters for each of the configurations:

Number of Features	Accuracy	Precision	Recall	F1
1	0.82753	0.21084	0.29700	0.28884
2	0.83327	0.27773	0.15650	0.20019
3	0.83807	0.33307	0.21400	0.26058
4	0.84020	0.36301	0.26300	0.30502
5	0.84320	0.38959	0.31050	0.34558
6	**0.84313**	**0.39192**	**0.32000**	**0.35233**
7	0.84353	0.39177	0.31400	0.34860
8	0.84307	0.38797	0.30650	0.34246
9	0.84267	0.38266	0.29350	0.33220
10	0.84573	0.39363	0.29050	0.33429

From the above table,best performance is for k=6

When , we use GridSearchCV, it also reflects our understanding so far. Below is the best_estimator_ for GridSearchCV:

Pipeline(steps=[('anova', SelectKBest(k=6, score_func=<function f_classif at 0x0000000007CFA4A8>)), ('NB', GaussianNB())])

DecisionTreeClassifier:

The parameter that we shall try to tune is min_samples_split. But before that, we will try to find the number of features resulting in best performance similar to Naive Bayes. But since there can sometimes be interactions between various steps in the analysis, being able to optimize over the full Pipeline is really a useful feature of GridSearchCV. So, in the final code, we will try to tune both of the parameters together but for now, let's just try to understand how the performance varies with the number of features in case of DecisionTreeClassifier

Again, we are trying with 1-10 features and listing down the performance metrics for each setting just for a comparative study.

Number of Features	Accuracy	Precision	Recall	F1
1	0.80753	0.20374	0.15250	0.17444
2	0.79040	0.19510	0.18300	0.18885
3	0.79793	0.26748	0.29650	0.28124
4	0.79380	0.23561	0.24350	0.23949
5	0.78407	0.22006	0.24350	0.23119
6	0.78327	0.21503	0.23600	0.22503
7	0.79367	0.24404	0.26100	0.25223
8	0.79707	0.23742	0.23600	0.23671
9	0.79407	0.23632	0.24400	0.24010
10	0.80153	0.24884	0.24200	0.24537

Now, let's use **GridSearchCV** to tune the number of features and **min_samples_split** in one go. Below is the output:

```
Pipeline(steps=[('anova', SelectKBest(k=3, score_func=<function f_classif at 0x0000000007CD4518>)), ('dtree',
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=1,
            min_samples_split=1, min_weight_fraction_leaf=0.0,
            presort=False, random_state=42, splitter='best'))])
```

So, as we can see the best performance is produced by the following parameters:

- K=3
- min_samples_split=1

To conclude, below are the performance parameters for Naive Bayes and DecisionTreeClassifier:

Algorithm	Accuracy	Precision	Recall	F1
NaiveBayes	**0.84313**	**0.39192**	**0.32000**	**0.35233**
DecisionTreeClassifier	0.79793	0.26748	0.29650	0.28124

So, as we can see, Naive Bayes with k=6 is clearly the winner here.

5. VALIDATION

Validation is the process of assessing the performance of the machine learning algorithm. After we design a model and train it using the data, we need to test how the model performs on some test data that was completely withheld from training. The separate set is known as a Test Set. If we test the model on the same dataset that was used to train the model, the performance will obviously be high as the model already has the knowledge of the data. This is known as **overfitting**, which is the most common problem if we go wrong. Moreover, in machine learning, we perform several rounds of tuning and retesting of the model. Through this process, the model indirectly learns of the test data set. This calls for the need of a third dataset called validation set.

But, splitting the data into 3 separate sets greatly reduces the data available for training. This problem is more severe in this case as there are very few data points and even fewer POI's. To address this problem, we are using StratifiedShuffleSplit. This function creates several folds from this dataset(1000 in this case). The folds are made by preserving the percentage of samples for each class (POI and non POI). Each fold has a training set and a test set. The overall performance is the mean of performance on each fold.

6. EVALUATION METRICS

In this case study, we mainly use two evaluation metrics:

- Precision: The precision is the ratio `tp / (tp + fp)` where `tp` is the number of true positives and `fp` the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.
- Recall: The recall is the ratio `tp / (tp + fn)` where `tp` is the number of true positives and `fn` the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.

Let's try to understand what these two metrics represent. If we make a prediction, precision is the measure of how accurate the prediction is. And recall is the measure of how many positive samples are we predicting. In the current scenario, precision is the measure of how accurately are we predicting a POI. That is , if the model predicts a person as a POI, what are the chances of that person actually being a POI is reflected by precision. And what percentage of people are predicted to be POI is reflected by recall.

So, if we want this model to be used just as the first step to narrow down the suspect list, we want a high recall. However, if this model is to be the final judge and jury, we definitely need a high value for precision.

Let's have a look at the average performance for each of the metrics for the two algorithms that we have used in this analysis.

Algorithm	Accuracy	Precision	Recall	F1
NaiveBayes	**0.85327**	**0.43578**	**0.34100**	**0.38261**
DecisionTreeClassifier	0.80600	0.28896	0.31150	0.29981

As can see, the precision is not very high. So, if this algorithm predicts a person as a POI, we cannot be absolutely sure that he is guilty. That person just comes under our suspect radar. This process gives us a good start and narrows down our suspect list. But it needs to be followed up with further investigation processes, focusing on the suspects.