final_project_questions-Erwin

January 11, 2018

1 Identifying Fraud From Enron Data

by ** Erwin **

1.1 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"]

1.2 Goals

- to understand and explore a public dataset
- to explore various tools for describing and visualizing the data
- to wrangle data, i.e. to clean up erronous and outlier data
- to select and tune compatible features
- to select appropriate machine learning model
- to understand potential pitfalls in data analysis, especially in engineering machine learning model
- to understand evaluation metrics
- to interpret model's performance
- to create useful reports containing the insights, findings, reusable codes.

In this project, I will use various machine learning model to determine if a person is a POI for Enron fraud cases. To do so, I will split the dataset into a subset of training data (where I use the data to engineer the model), and test data (where I use the data to evaluate the engineered model).

The following python packages are used:

- **pandas** - for data structure and analysis
- **numpy** - for useful numerical data modification
- **scikit-learn** - for machine learning model
- **matplotlib** - for basic data visualization
- **seaborn** - for more elegant and robust data visualiation

1.3 Project Background

Enron was one of the largest corporates in the United States. However, It went into bankruptcy due to widespread corporate fraud. In the subsequent investigation, significant amount of confidential information was published publicly. This includes emails and detailed financial data from top executives.

For this project, we are going to make use of the Enron public data to create a machine learning model to classify whether a person in the Enron dataset is a person of interest (POI) for the Enron corporate fraud case. For more information of the Enron scandal, read here: https://en.wikipedia.org/wiki/Enron_scandal

In the dataset, there are: - 146 records with 6 email features, 14 financial, and 1 labeled feature (i.e. POI) - Out of the 146 records, there are 18 records labeled as persons of interest.

In this case, the machine learning model will predict the POI which is the target feature (label) by using the other features (financial and email features).

1.4 Outliers

From this project, I think the best way to find outliers is:

1) to have an high-level understanding of the data. This includes:

- understanding each feature, especially the important ones (e.g. bonus, salary, poi, from_this_person_to_poi, from_poi_to_this_person, etc)
- to have an overview of the data type, number of records, available features.
- to see if there is any missing data, wrong data type, etc.

2) After we understand the data, we can visualize either by using bivariate plots or univariate plots to check the potential outliers. In this case, I use scatterplot to examine the important variables (e.g. bonus, salary) to visually locate which records are more suspicious.

I found out there are two data-points that are most likely outliers:

- TOTAL: this is the total aggregation of all the records. Think of it like a "Total" in a spreadsheet data.
- THE TRAVEL AGENCY IN THE PARK: This is definitely not a name of an employee in Enron.

To handle the outliers, first I sort descendingly the dataset keys to ensure that those records are indeed outliers with very high salary / bonus.

The second outlier (THE TRAVEL AGENCY IN THE PARK) is a bit more tricky and require more visual attention to scroll through the list of records one by one.

After ensuring those are really outliers, I remove the records from the dictionary using the pop method. Those outliers must be removed to ensure that they don't affect negatively to the machine learning model.

1.5 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 doesn't 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.) If you used an algorithm like a decision tree, please also give the feature importances of the features that you use. [relevant rubric items: "create new features", "properly scale features", "intelligently select feature"]

1.6 Selected Features

Two ways I used to find out the features to use:

- Calculate the correlation between each feature and the label (POI) then sort them descendingly. This way I can see which features have high correlation with the label.
- The most accurate way is to use the built-in function SelectKBest to determine the top-N features to be used to predict the POI.

Usually, the second method is sufficient to select the appropriate features. However, I used the first option for sanity check to see if I have done any blunder in the SelectKBest method.

Using the above method, I filtered out to only 10 best features to be used. The table below shows the score and the number of valid record (non-NaN) for each feature.

Feature	Score	#Valid_Record
exercised_stock_options	24.815	101
total_stock_value	24.183	125
bonus	20.792	81
salary	18.290	94
deferred_income	11.458	48
long_term_incentive	9.922	65
restricted_stock	9.212	109
total_payments	8.772	123
shared_receipt_with_poi	8.589	86
loan_advances	7.184	3

In the table above, we can see that all the features generally looking fine for the model, with only `loan_advances` has a high score but only 3 valid (non-NaN) values.

1.6.1 Feature Engineering

In addition, there is a lack of email-related features in the model. So, I engineered three new features:

- poi_ratio = number of messages sent to + received from poi / total sent + received messages
- fraction_to_poi = number of messages sent to poi / total sent messages
- fraction_from_poi = number of messages received from poi / total received messages

The objective is to explore if there is any significant impact if we include the email features into the model.

Next, we will test out the effect before and after we added the 3 new features to the machine learning model (DecisionTree) we used.

1.6.2 SelectKBest Table

Before adding 3 new features:

DecisionTreeClassifier	Precision	Recall	Accuracy
All Features Used	0.24164	0.22591	0.86
9KBest Features Scikit-Learn	0.27254	0.25400	0.88
10KBest Features Scikit-Learn	0.27861	0.25490	0.88
11KBest Features Scikit-Learn	0.27580	0.27885	0.88
12KBest Features Scikit-Learn	0.27775	0.28580	0.86

After adding 3 new features:

DecisionTreeClassifier	Precision	Recall	Accuracy
All Features Used	0.28085	0.27757	0.88
9KBest Features Scikit-Learn	0.29484	0.28149	0.84
10KBest Features Scikit-Learn	0.29310	0.30488	0.84
11KBest Features Scikit-Learn	0.27581	0.27885	0.88
12KBest Features Scikit-Learn	0.27776	0.28580	0.86

Because we are trying to predict a POI, so the objective is to have a higher recall because we cannot risk to miss classifying a potential POI just because we want the model to be have more precision.

One thing interesting is that in the 10KBest Features option, there is a more significant gain when the 3 new features are added into the model.

So, in this case, using 10KBest Features seems to be the most optimal one with the highest recall and only slightly lower precision and accuracy as compared to the highest ones.

1.6.3 Feature Scaling

Feature scaling is important step in doing a machine learning model because it standardizes the range of each of the selected feature, so the features with a naturally higher range of values will not dominate the other features. For example, salary or bonus usually have much higher range of values as compared to number of emails.

In this case, I use **MinMaxScaler()** function to normalize the features.

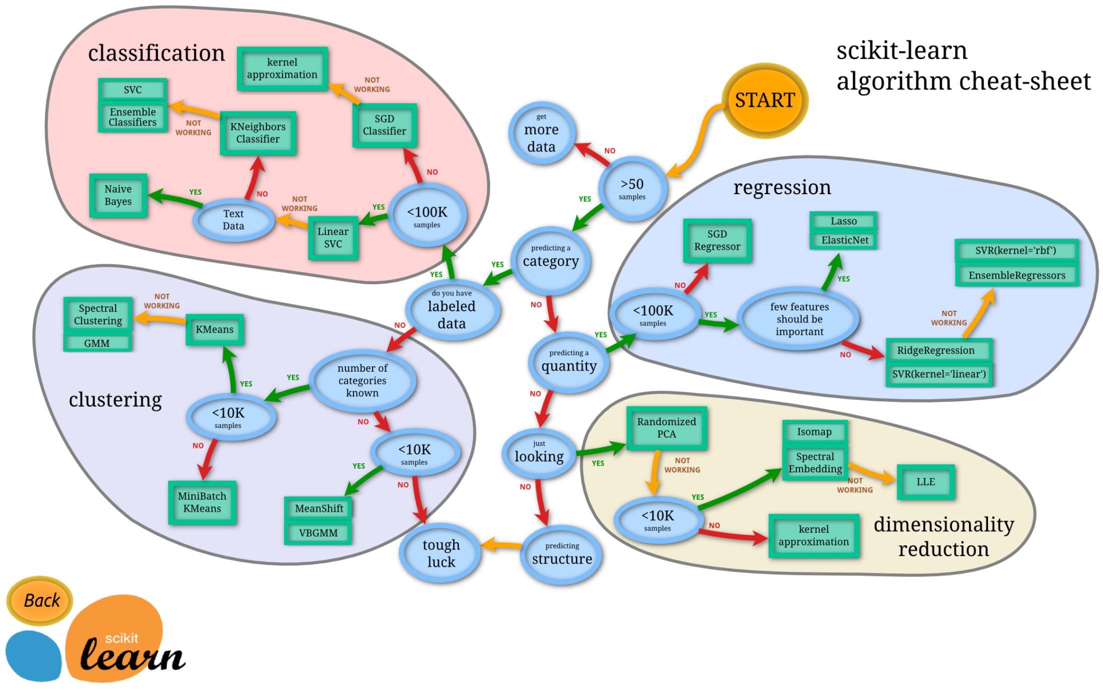
1.7 Question 3

What algorithm did you end up using? What other one(s) did you try? [relevant rubric item: “pick an algorithm”]

After some exploration, I realized there is an built-in "cheat-sheet" to help us decide on appropriate machine-learning model to use.

```
In [1]: from IPython.display import Image
        Image(filename='BZJiN.png', width=800, height=600)
```

```
Out[1]:
```



As shown by the image above, it seems that classification section seems to fit the most criteria. So, I will try various types of classification algorithm.

1.7.1 Machine Learning algorithms used

The following algorithms are tested to check their accuracy, precision, and recall in oder to pick algorithm to work with: "Nearest Neighbors", "Linear SVM", "RBF SVM", "Decision Tree", "Random Forest", "AdaBoost", "Naive Bayes", "Extra Trees" The results were as follow:

==========================

Classifier:

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=3, p=2, weights='uniform')

precision: 0.216064285714

recall: 0.0999064213564

Accuracy: 0.84 (+/- 0.00)

===========================

Classifier:

SVC(C=0.025, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma='auto', kernel='linear', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)

precision: 0.0

recall: 0.0

Accuracy: 0.93 (+/- 0.00)

=========================

Classifier:

SVC(C=1, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape=None, degree=3, gamma=2, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False)

precision: 0.0

recall: 0.0

Accuracy: 0.93 (+/- 0.00)

=========================

Classifier: **Most Optimal Option**

DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5, max_features=None, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')

precision: 0.293106403319

recall: 0.304880699856

Accuracy: 0.84 (+/- 0.00)

=========================

Classifier: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini', max_depth=5, max_features=1, max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)

precision: 0.28009047619

recall: 0.113670634921

Accuracy: 0.91 (+/- 0.00)

=========================

Classifier:

AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0, n_estimators=50, random_state=None)

precision: 0.279896681097

recall: 0.208357647908

Accuracy: 0.81 (+/- 0.00)

=========================

Classifier:

GaussianNB(priors=None)

precision: 0.328858626753

recall: 0.38744549062

Accuracy: 0.88 (+/- 0.00)

=========================

Classifier:

ExtraTreesClassifier(bootstrap=False, class_weight=None, criterion='gini', max_depth=None, max_features='auto', max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1, oob_score=False, random_state=None, verbose=0, warm_start=False)

precision: 0.347327777778

recall: 0.172377994228

Accuracy: 0.86 (+/- 0.00)

========================

Based on the above result, **DecisionTreeClassifier seems to be the most optimal to work with**.

ExtraTreesClassifier is similar to DecisionTreeClassifier and with lower precision value, so it is ignored.

GaussianNB is ignored because it is too simple with less parameter, so it cannot be further tuned.

"Linear SVM" and "RBF SVM" have 0 values for both precision and recall, thus ignored.

KNeighbourClassifier is ignored too because it has recall value close to 0.

AdaBoostClassifier and RandomForestClassifier are also ignored fom selection because of the sub-optimal scores.

1.8 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? (Some algorithms don't 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 if you used, say, a decision tree classifier). [relevant rubric item: "tune the algorithm"]

1.8.1 Parameter Tuning

Machine Learning algorithms have parameters that can be adjusted to affect the outcome of the machine learning process. When we adjust those parameters, it is called as "parameter tuning".

The objective of parameter tuning to find the sweet spot or the optimized points for the parameter values. The more fine-tuned the parameter the algorithms, the more it will be biased to the training data. As such, it will lead a model that may overfit. As such, we need to balance out the scores from the parameter tuning process.

We will start tuning the parameters for DecisionTreeClassifiers. Afterward, we will pick the best variation we have from the parameter tuning methods.

1.8.2 DecisionTreeClassifier Parameter Tuning

Method used: **grid_search.GridSearchCV** to find the optimum configuration for the parameter.

I used the following parameters to tune for the algorithm I picked:

parameters = {'max_depth': [1,2,3,4,5,6,8,9,10],'min_samples_split':[2,3,4,5],'min_samples_leaf':[1,2,3,4,5,6,7,8
'criterion':('gini', 'entropy')}

Grid search provided the following parameters and score:

DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=2, max_features=None, max_leaf_nodes=None, min_samples_leaf=2, min_samples_split=2, min_weight_fraction_leaf=0.0, random_state=None, splitter='best')

Score: 0.657142857143

Processing time: 14.54 s

1.9 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 item: "validation strategy"]

1.9.1 Validation

Validation is the process to determine how well an algorithm can fit outside the given dataset. In this case, we split the data into subset of training and test data. Hence, the test data will be used to determine how the algorithm performs.

1.9.2 Classic Mistake

A potential pitfall is **over-fitting**, in which the model is trained and performs very well on the training dataset. However, it performs worse in the test dataset. This means that the algorithm overfit the training data that was used and unable to perform well in a generalized situation.

1.9.3 Analysis Validation

To validate the analysis,two methods are used:

1: **evaluate.py** --> It takes the average precision and recall over 1000 randomized trials with the dataset sub-sectioned with a 3:1 training-to-test ratio.

2: **tester.py** --> It uses the StratifiedShuffleSplit, folds = 1000 evaluator to provide metrics on the classifiers.

1.10 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"]

1.11 Two Evaluation metrics:

Evaluation Metrics used: **Precision** and **Recall**.

Recall: (true positives)/(true positive+false negative). In this case, it means how many % of all the persons in the Enron data will be classified as POI.

Precision: (true positives)/(true positive+false positive). In this case, it means how many % of the classifed POI are truly a POI.

1.11.1 Average Perfomance

Validation 1 (StratifiedShuffleSplit, folds = 1000)

Classifier	Precision	Recall	Accuracy
DecisionTreeClassifier	0.471	0.471	0.86

Validation 2 (Randomized, partitioned trials, n=1000)

Classifier	Precision	Recall	Accuracy
DecisionTreeClassifier	0.314	0.329	0.86

1.11.2 Understanding Performance

In this case of classifying POI, recall will have more importance than precision. This is because we would rather have more people classified as POI than let the POI slipped away. It will not be a big issue even if the people classified as POI are not a true POI. In a real life scenario, at most the suspect will be released. This is better than letting a potential POI escaped. As such, the precision can be of less importance.

1.12 Resources and References

- Introduction to Machine Learning (Udacity)
- scikit-learn Documentation