Udacity Nanodegree - Data Science Fundamentals II

Machine Learning Module Final Project

Author: Gabriel S. Gonçalves

1) Introduction and Initial Exploratory Data Analysis

Nowadays, fraud detection is one key concern for many businesses and Machine Learning algorithms are helping Data Scientist and Engineers tackle this problem. Banks, credit card providers, online shops and many other types of services have to face daily problems with attempts of fraud throughout the world. One of the most famous cases of corporate fraud in our recent history is the Enron scandal that happened in 2001. Enron Corporation was an American energy company with a market value of billions of dollars. The fraud occurred on the financial division of corporation, with key directors of the company falsifying numbers to investors and general public, late charged with crimes. The goal of the current project is to access data generated out of employees that were part of the fraud inside Enron (labeled Person of Interest, a.k.a POI) and regular employees that were not prosecuted criminally. The dataset contains different types of numeric data like salary, number of email exchanged with POI, amount of stocks, etc. and also some text data contained on the emails exchanged by employees. We will focus on the numeric data to do our exploratory data and model prediction, with the goal to predict the tested subjects as being a POI or not. The data that were provided for the project consisted of a pickle file that could be read on a Python dictionary, with keys represented by the name of the employee, and values being a dictionary containing the features and respective values. In order to inspect the data in a more detailed way, the dictionary was converted into a Pandas DataFrame, with the names of each subject as index, and each feature as columns. By inspecting the Dataframe created with .info() it is noticed that it has 146 rows and 21 columns. One problem that is initially visualized is that the data type for most of the columns are non-null object type (the equivalent for string in Pandas), and that the missing values are coded as 'NaN' strings. The only exception to it is the 'poi' column which is a boolean type column (Figure1). By converting the columns to numeric type, we can have a better notion of the missing values for each feature column.

```
<class 'pandas.core.frame.DataFrame'>
Index: 146 entries, ALLEN PHILLIP K to YEAP SOON
Data columns (total 21 columns):
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
restricted_stock_deferred     18 non-null float64
total_stock_value             126 non-null float64
expenses                      95 non-null float64
loan_advances                 4 non-null float64
from_messages                 86 non-null float64
other                         93 non-null float64
from_this_person_to_poi       86 non-null float64
poi                           146 non-null bool
director_fees                 17 non-null float64
```

```
deferred_income              49 non-null float64
long_term_incentive          66 non-null float64
email_address                0 non-null float64
from_poi_to_this_person      86 non-null float64
dtypes: bool(1), float64(20)
memory usage: 24.1+ KB
```

Figure 1. Enron Dataframe information.

Some classes of algorithms from Scikit-Learn do not accept null values, so we replaced them with 0 using the .fillna(0.0) method for Pandas Dataframes.
A traditional way to spot outliers is by using boxplots or scatter plots. But as our dataset has so many features, it becomes a daunting and complex task. We used PCA to reduce the number of dimensions of our data to 2 components, and find samples that diverge 3 or more standard deviations from the mean, by using the z-score. We ended up finding 5 samples that diverged more than 3 standard deviations from the mean of one of the components generated (Figure 2).

```
['DELAINEY DAVID W', 'KAMINSKI WINCENTY J', 'LAVORATO JOHN J', 'SHAPIRO RICHARD S',
'TOTAL']
```

Figure 2. Listed outliers on the dataset determined by z-score of PCA analysis.

We can visualize the outliers, POI and non-POI on a scatter plot with values generated by PCA analysis with 2 components (Figure 3). The outliers were removed from the following steps of the analysis.

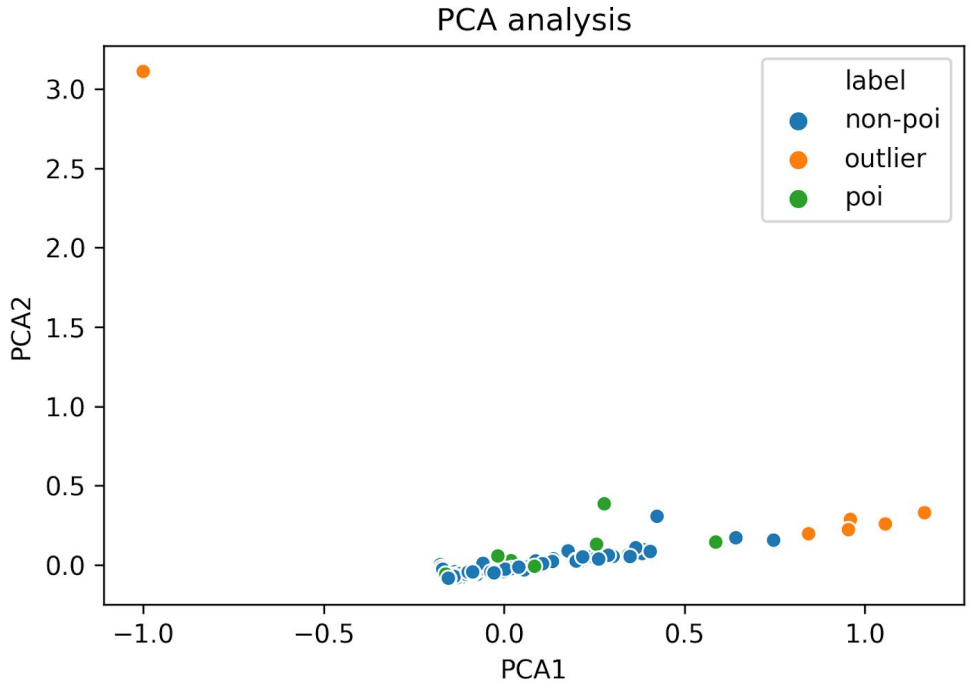

Figure 3. PCA analysis for the Enron Dataset.

2) Feature engineering and selection

One key step of Machine Learning methodology is feature selection and engineering. As we have a dataset formed by numerical data it is important to rescale the values, so that each feature can have the same weight when comparing the its respective importance. To do it we used

SciKit-Learn MinMaxScaler to transform the values from 0 to 1. To determine the features that were best suited for our analysis, we used the ExtraTreesClassifier to get the importance score for each feature (Figure 4).

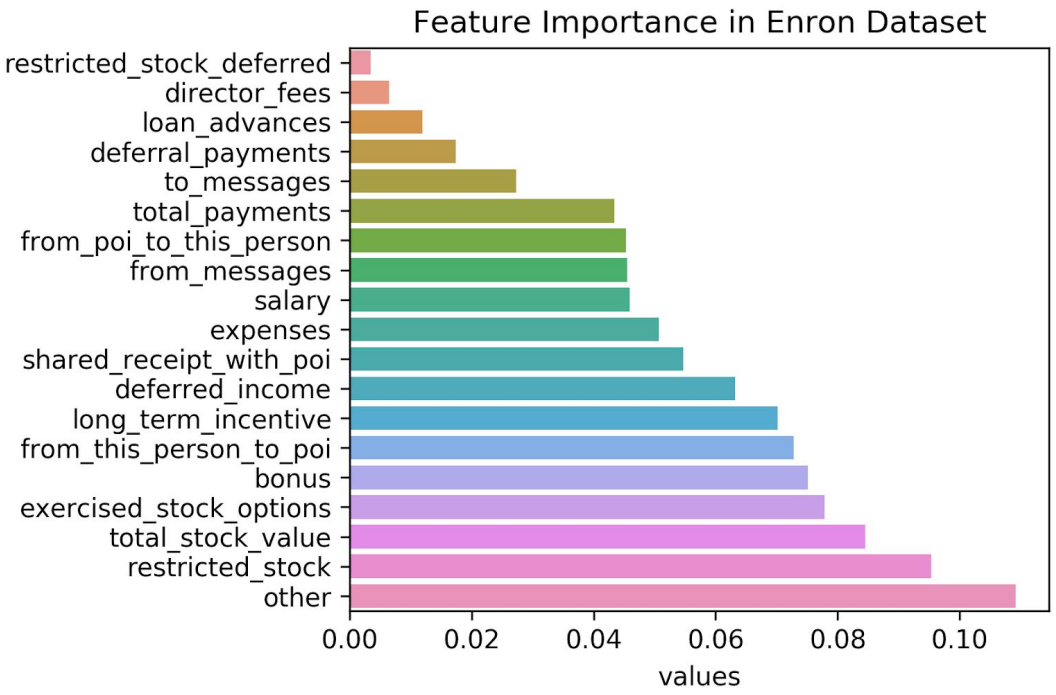

Figure 4. Feature importance for Enron Dataset

As it can be seen on Figure 4 we have different scores for each feature, and 3 of them were relatively low compared to the rest: restricted_stock_deferred, director_fees and loan_advances. So it was decided to remove these features from the analysis and keep the remaining 16. Another problem that was noticed with the dataset was the class imbalance between POI and non-POI samples. After removing the outliers the dataset presented a ratio of more than 7:1 (non-POI:POI) and it affected negatively our prediction. One traditional approach to deal with class imbalance is to oversample the class with low representation, or downsample the high representation class. We decided to use the Synthetic Minority Over-sampling Technique for generation of new samples (SMOTE) to minimize the disparity between classes.

3) Algorithm Selection

After doing our initial EDA and feature engineering a few different Machine Learning classification algorithms were tested to decide which one performed better for our dataset. The approach used consisted on running the algorithms with default parameters and measure performance metrics (precision, recall and f1-score). The algorithm with best f1-score would be used on further model tuning to try to optimize it. We tested a Naive Bayes classifier (GaussianNB), Random Forest Classifier (RandomForestClassifier) and Logistic Regression (LogisticRegression). The results are presented on the Figure 5.

```
GaussianNB(priors=None, var_smoothing=1e-09)

ROC AUC score = 0.8583156779661016

              precision    recall  f1-score   support

       False       0.55      0.85      0.67        59
        True       0.72      0.36      0.48        64
```

```
   micro avg       0.59      0.59      0.59       123
   macro avg       0.63      0.60      0.57       123
weighted avg       0.64      0.59      0.57       123

Confusion Matrix - GaussianNB
[[50  9]
 [41 23]]

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

RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
            max_depth=None, max_features='auto', 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, n_estimators='warn',  n_jobs=None,
oob_score=False, random_state=42, verbose=0, warm_start=False)

ROC AUC score = 0.9615995762711865

              precision    recall  f1-score   support

       False       0.87      0.92      0.89        59
        True       0.92      0.88      0.90        64

   micro avg       0.89      0.89      0.89       123
   macro avg       0.89      0.90      0.89       123
weighted avg       0.90      0.89      0.89       123

Confusion Matrix - RandomForestClassifier
[[54  5]
 [ 8 56]]

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

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=42, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)

ROC AUC score = 0.8344809322033898

              precision    recall  f1-score   support

       False       0.80      0.66      0.72        59
        True       0.73      0.84      0.78        64

   micro avg       0.76      0.76      0.76       123
   macro avg       0.76      0.75      0.75       123
weighted avg       0.76      0.76      0.75       123

Confusion Matrix - LogisticRegression
[[39 20]
 [10 54]]
```

Figure 5. Initial performance metrics for the 3 tested algorithms (GaussianNB, RandomForestClassifier, LogisticRegression)

As we can see on Figure 5, the Random Forest algorithm outperformed the other 2 algorithms in each measured metric (Precision, Recall, F1-Score and ROC AUC score). So we decided to use

Random Forest as our predictive model algorithm and explore other options for parameters to increase the performance.

4) Model optimization with parameter tuning

After deciding which algorithm to use on our model, we evaluated which parameters would better fit our data. This is one key step on Machine Learning methodology, and using the proper approach can increase significantly the performance of the model. In order to do it we used the GridSearchCV from SciKit-Learn to explore different combinations of parameters and return the best options. After deciding a range of values for each parameter, it's also set which metric the GridSearchCV should use to measure performance, as there are trade-offs between precision and recall. Depending on the goal of the model, picking the right metric is essential. The tested parameters and respective values are described on Figure 6.

```
param_grid = {
    'max_depth': [80, 90, 100, 110,150, 200],
    'max_features': [None, 'auto', 2, 3, 6, 9, 12],
    'min_samples_leaf': [2, 4, 6, 12],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000],
    'class_weight': ['balanced','balanced_subsample',
                     {0:1, 1:6}, {0:1, 1:3}] }
```

Figure 6. Parameters tested on GridSearchCV.

It was tested the maximum depth of the tree (max_depth), maximum number of features for best split (max_features), minimum number of samples per leaf (min_samples_leaf), minimum number of samples per split (min_samples_split), number of evaluated trees in the forest (n_estimators) and weights associated to each class in the dataset (class_weight).
In our case, as we are dealing with fraud detection we should be concerned more with Recall than precision, minimizing false positive rate, to avoid not investigating potential fraud cases. The precision rate for a fraud detection model can be lower than recall rate, as the case of investigating cases of regular transactions as fraud is acceptable. The selected parameters are described on figure 7.

```
clf = RandomForestClassifier(
            bootstrap=True,
            class_weight='balanced_subsample',
            criterion='gini', max_depth=100, max_features=6,
            max_leaf_nodes=None, min_impurity_decrease=0.0,
            min_impurity_split=None, min_samples_leaf=2,
            min_samples_split=10, min_weight_fraction_leaf=0.0,
            n_estimators=100, n_jobs=None, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
```

Figure 7. Parameters defined for RandomForestClassifier with GridSearchCV.

5) Validating the model

In order to check if the chosen samples for training and testing didn't have bias a cross validation can be performed. The bias can happen when of your training set does not represent the diversity observed in tested set. Cross Validation consists of splitting the dataset and checking if the metrics generated for each split is consistent. We used StratifiedShuffleSplit from Scikit-Learn with 10 splits and measured the recall score, precision score and f1-score for each split, getting a

a final mean of 0.945, 0.866, and 0.901 respectively.

6) Evaluation metric

As described above, the main goal of our model is to detect employees that were involved in fraud inside Enron. By maximizing performance over F1-Score, we tried lowering the number of false positive individuals on our test set, meaning we tried to avoid not investigating individuals that were potential criminals, but also having a good precision. As we can see from our resulted metrics (Figure 8), our model has showed a great results for precision, recall and F1-score.

```
 ROC AUC score = 0.95

              precision    recall  f1-score   support

       False       1.00      0.85      0.92        34
        True       0.89      1.00      0.94        40

   micro avg       0.93      0.93      0.93        74
   macro avg       0.94      0.93      0.93        74
weighted avg       0.94      0.93      0.93        74

Confusion Matrix - Optimized RandomForestClassifier
[[29  5]
 [ 0 40]]
```

Figure 8. Evaluation Metrics for Optimized Random Forest Classifier.

One remarkable thing is that in this simulation we had 0 false positives, meaning that all POI would be caught using our model.

7) Final remarks

Based on the initial goal and the results from our recall score it can be said that our model performed well under the conditions presented by the dataset. We optimized our model to try to identify all potential POI (high recall) and still get a good precision. The next step for this project would be evaluate the rest of the employees from Enron, and also try to use our model on other companies that experienced similar cases of financial fraud.