

Machine Learning Report

Enron Data Analysis

The analysis of the Enron data is aimed at identifying persons of interesting regarding the fraud case that led to the company's bankruptcy. Given the enormous amount of data released to the public, applying traditional methods to uncover new persons of interest that might have been overlooked could turned out to be messy and tedious. Contrarily, Machine Learning techniques are great to generalize from big datasets, to predict new outcomes, and spot trends in the data that we would probably no be able to do with traditional methods.

The features found in the Enron data could be organized into two main categories: financial and networking. The financial data covers salary, bonus, deferral payments stocks operations, and ownership. Networking within the company can be explore through features such as the number os messages sent or received and their recipients. Once the row with all zeros got removed, we were left off with 145 data points. The original data actually carries a significant amount of missing values that the function `featureFormat()` fixes by zeroing them out. In a nutshell, a person of interest may be someone that made a lot money and was in constant contact with influential people.

Data Preprocessing

In order to spot some possible outliers in the data, I've plotted a scatter for all numerical features. The most notorious outlier was the 'TOTAL' key value, which corresponds to the sum of various financial values in a given column. Clearly this one should be dropped from the dictionary, which I did using `.pop()`. After that, I ran linear regressions for each of the features in an attempt to discard the top 10% values with the

largest residual errors. However, applying the largest residual errors method always turned out with a dataset short in a significant (40%-55%) amount of positive POI on the training set. I interpreted this as an indication that these anomalies in the data could actually be what I'm looking for. So, I have decided not to drop any other outliers for now.

Being the problem at hand a classification of two groups (or classes), POI and no-POI, the method applied for feature selection was the Recursive Feature Elimination (RFE) based on Logistic Regression. While Logistic Regression should be a good algorithm for classifying dual classes, RFE is a SelectKBest method that searches for the best combination of features among all possible subsets of those features. Then, using those two methods combined is in line with the characteristics of the data.

The initial target number for selecting features was set to 13 as a conservative start, which represents around 70% of the all available features. This initial set could be revisited during the performance evaluation phase. The selection process got run twice, one time with not-scaled features and another time with scaled features. Scaling the data was really important as the features hold numbers in different magnitudes and the Logistic Regression algorithm is very sensitive to components' scales. In both scenarios the proportion of financial and networking features were the same, even though the selected features were not always the same (see table below).

The table below summarizes all the reported procedures. The blue-filled rows indicate when not-scaled and scaled feature got selected in both scenarios.

Features	Not Scaled Selection	Scaled Selection	Possible Outlier
email address	Not-selected	Not-selected	No
Salary	Not-selected	Selected	Yes
deferral_payments	Not-selected	Selected	Yes
total_payments	Selected	Not-selected	Yes
loan_advances	Selected	Selected	Yes
bonus	Selected	Not-selected	Yes
restricted_stock_deferred	Selected	Not-selected	Yes

deferred_income	Not-selected	Selected	Yes
director_fees	Selected	Selected	Yes
total_stock_value	Selected	Selected	Yes
expenses	Not-selected	Selected	Yes
exercised_stock_options	Selected	Selected	Yes
other	Not-selected	Selected	Yes
long_term_incentive	Selected	Not-selected	Yes
restricted_stock	Selected	Not-selected	Yes
to_message	Selected	Selected	Yes
from_poi_to_this_person	Selected	Not-selected	Yes
from_this_person_to_poi	Not-selected	Selected	Yes
from_message	Selected	Selected	Yes
shared_receipt_with_poi	Selected	Selected	Yes

Beyond the original features set, I have included the two additional not ready-made features aiming in better measuring how POI networked, just as discussed in one of the classes. The two features assess the rate people exchanged messages with POI and no-POI. The first one calculates the fraction of one's all messages that were sent to a POI (`fraction_from_this_person_poi`), the other one measures the other way around, meaning the fraction of messages one received from POI (`fraction_to_this_person_from_poi`).

Experimenting with Classifiers

The candidate algorithms are some of the classifiers explored in the classes: Gaussian Naive Bayes, Support Vector Classifier (SVC), Decision Trees, and K Near Neighbors. These algorithms are referenced as satisfactorily good binary classifiers. Initially I tried them out using the `cross_val_score()`, a Gridsearch cross-validation method from Scikit-Learn. Cross-validation is an excellent technique to test your model

and yet avoid overfitting it. It works by splitting the training set into a smaller training and a validation sets, then testing the model across these sets. This strategy was extensively laborious for the hyperparameters had to be manually tuned.

Nonetheless, it was yielding very poor results, way under the rubric 0.3 expectation, specially with the precision measurements. This was very worrisome for the precision is the main goal in this project, since I want my true positives prediction to be as good as possible (you don't want to false accuse someone). The following tables present the best manual selection with its best hyperparameters settings both for precision and recall.

Best Preliminary Manually-tuned Precision Results			
Naive Bayes	SVM-SVC (C= 100.0, kernel='linear')	Decision Tree (min_samples_split=40)	K Near Neighbors (n_neighbors=3)
Score Mean: 0.11825396825396825 Score STD: 0.0991317179598383	Score Mean: 0.0 Score STD: 0.0	Score Mean: 0.1 Score STD: 0.30000000000000004	Score Mean: 0.2 Score STD: 0.4000000000000001

Best Preliminary Manually-tuned Recall Results			
Naive Bayes	SVM-SVC (C= 100.0, kernel='linear')	Decision Tree (min_samples_split=40)	K Near Neighbors (n_neighbors=3)
Score Mean: 0.85 Score STD: 0.32015621187164245	Score Mean: 0.0 Score STD: 0.0	Score Mean: 0.1 Score STD: 0.30000000000000004	Score Mean: 0.1 Score STD: 0.30000000000000004

Cross-validation indicated me that these rather good algorithms were underfitting for some reason. Facing such poor results, before trying to change anything regarding the features or revisiting my initial position towards the outliers, I decided to expand and automate the fine-tuning by running a Randomized Search on SVM-SVC, Decision Tree, and KNN (the hyperparameters for naive bayes are far too simpler for running a random search). The scikit-learn tool provides a random search at [https://scikit-](https://scikit-learn.org/stable/modules/randomized_search.html)

learn.org/stable/auto_examples/model_selection/plot_randomized_search.html. The following tables summarize the best performances.

Precision Results Using The Best Tuning Hyperparameters		
SVM-SVC	Decision Tree	K Near Neighbors
Mean validation score: 0.143 (std: 0.315) Parameters: {'kernel': 'linear', 'gamma': 0.001, 'C': 500}	Mean validation score: 0.378 (std: 0.443) Parameters: {'min_samples_split': 8, 'min_samples_leaf': 8, 'max_features': 9}	Mean validation score: 0.096 (std: 0.294) Parameters: {'weights': 'uniform', 'n_neighbors': 9, 'leaf_size': 10}

Recall Results Using The Best Tuning Hyperparameters		
SVM-SVC	Decision Tree	K Near Neighbors
Mean validation score: 0.248 (std: 0.398) Parameters: {'kernel': 'poly', 'gamma': 0.1, 'C': 100}	Mean validation score: 0.448 (std: 0.408) Parameters: {'min_samples_split': 8, 'min_samples_leaf': 8, 'max_features': 9}	Mean validation score: 0.096 (std: 0.294) Parameters: {'weights': 'uniform', 'n_neighbors': 3, 'leaf_size': 10}

The random search isolated a fine-tuned set of hyperparameters for the decision tree that can perform over 0.3 for both precision and recall. The average and the standard deviation scores achieved by the best hyperparameters are written in blue at the tables aforementioned. Regarding the recall score, decision tree performed slightly better than the other two. However, the results for precision score proved decision tree to be at least 2.6 times more precise than SVC and KNN, being the latter the worst performer.

Decision tree even had a better precision score on the test set (**0.6**). The feature importances, as shown below, indicates that only five features are explanatory for the dataset: deferred_income, expenses, other, fraction_to_this_person_from_poi, and fraction_from_this_person_poi.

Features Importances
Feature salary holds importance of 0.0
Feature deferral_payments holds importance of 0.0
Feature loan_advances holds importance of 0.0
Feature deferred_income holds importance of 0.28070397435491884
Feature total_stock_value holds importance of 0.0
Feature expenses holds importance of 0.144615120414684
Feature exercised_stock_options holds importance of 0.0
Feature other holds importance of 0.42148073425523574
Feature director_fees holds importance of 0.0
Feature to_messages holds importance of 0.0
Feature from_messages holds importance of 0.0
Feature from_this_person_to_poi holds importance of 0.0
Feature shared_receipt_with_poi holds importance of 0.0
Feature fraction_to_this_person_from_poi holds importance of 0.02267575460088313
Feature fraction_from_this_person_poi holds importance of 0.13052441637427833

Considering all the results at hand, and the goal set by the project specifications, I chose decision tree as my final classifier. This classifier holds a true positive rate of 4 out of 10 in average, and a false negative rate of 7 out of 10 in average. Even though these results are enough for the purposes of the project, this classifier is either extremely underfoot or only having 145 instances available is not enough to properly train the algorithms. Oddly enough when you apply PCA on the current settings, all the classifiers' performance became worse still. In conclusion, the whole feature engineering should be revisited, and possibly making a greater effort to gather more instances for training the algorithms.