Identify Fraud from Enron Email

Nicole Mister

Introduction

Enron, an energy commodities and services company, declared bankruptcy in 2001. The company then went under investigation for fraud. A corpus of emails was used in the investigation to determine persons of interest (POI) in the case. The goal of this project is to use this corpus of emails as well as employee financial data to create a machine learning algorithm to predict if an Enron employee is a POI.

Understanding the Dataset

Data Exploration

Our dataset contains 146 records with 21 features. Of the 146 records, 18 are POI.

Outlier Investigation

I graphed a scatterplot of salaries and bonuses and I could see that there were some outliers. After a quick visual inspection of enron61702insiderpay.pdf, I was able to determine “TOTAL” and “THE TRAVEL AGENCY IN THE PARK” were outliers that should be removed. I used the dicitionary pop function to remove those records. While another scatterplot indicated that the data still included additional outliers, I did not remove any additional data. The outliers included several POIs. I would therefore potentially lose pertinent data by removing these points.

Optimize Feature Selection/Engineering

Create new features

Because I believe the proportion of emails to and from a POI may be more relevant than just the sum of emails, I created two new features, from_ratio and to_ratio. The to_ratio is the number of emails from this person to a POI divided by the total number of emails sent by this person. The from_ratio is the number of emails from this person to a POI divided by the total number of emails from this person. The from_ratio was selected by the SelectKBest function as the second best function with an F-score of 8.77.

Intelligently select features

To select features, I used a pipeline and GridSearchCV to apply SelectKBest with all possible values of k (1-21). with this, I was able to determine that best number of features is 8. The features selected and F-score are as follows:

Feature	F-score
[long_term_incentive	18.289684043404513
from_ratio	8.7727777300916756
exercised_stock_options	8.589420731682381

Identify Fraud from Enron Email

Nicole Mister

bonus	5.2434497133749582
total_stock_value	2.3826121082276739
salary	1.6463411294420076
restricted_stock	0.22461127473600989
deferred_income	0.16970094762175533

Properly scale features

I did not use scaling for these features.

Pick and Tune an Algorithm

Pick an algorithm

I tried three different algorithms: Decision Tree, AdaBoost and Naïve Bayes. I ran the code several times. The scores varied with each run as expected, but the F1 score for Decision Tree was consistently higher than the other algorithms in each run. The results of one run are below.

Decision Tree	Precision: 0.43478	Recall: 0.50000	F1: 0.46512
AdaBoost	Precision: 0.47059	Recall: 0.40000	F1: 0.43243
Naïve Bayes	Precision: 0.44444	Recall: 0.40000	F1: 0.42105

Parameter tuning

Tuning the parameters means to adjust settings of the algorithm which may impact how well the algorithm performs on a dataset. I used GridSearchCV and StratifiedShuffleSplit to test several settings for three different parameters for Decision Tree. GridSearchCV tests every parameter setting given and returns the settings with the best F1 score. Ever run of the code would return different combinations of parameters, so the parameter settings did not make a clear difference in the F1 score. Ultimately, I decided there wasn't enough of a gain to use a complex set of parameters and used Decision Tree's default settings. Below are the results demonstrating that the default parameters scored a better F1 then some of the more complex settings.

`Default Parameter Settings`	`Precision: 0.33981`	`Recall: 0.35000`	`F1: 0.34483`
`class_weight=None, criterion='entropy', max_depth=None, max_features='sqrt', max_leaf_nodes=None, min_impurity_split=1e-07, min_samples_leaf=1,`	`Precision: 0.39024`	`Recall: 0.24000`	`F1: 0.29721`

min_samples_split=20, min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best'			
class_weight=None, criterion='gini', max_depth=None, max_features='sqrt', 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: 0.30594	Recall: 0.33500	F1: 0.31981

Validate and Evaluate

Usage of Evaluation Metrics

The evaluation metrics I used to evaluate the algorithms in this analysis were precision and recall. Precision is the percent of employees labeled as POIs that were actually POIs. The final classifier had a precision of 0.340 which means that of all the individuals the classifier identified as a POI, 34.0% of them actually were POIs. Recall is the percent of POIs correctly identified. The final classifier had a recall of 0.350 which means that of all the POIs, 35.0% of them were correctly identified by the classifier.

Validation and its importance.

Response addresses what validation is and why it is important.

Validation tests an algorithm's performance against a set of data not used in the training. If validation is not performed correctly, for instance, testing on the same data used to train, you can overfit your data so that the effectiveness of the algorithm seems better than it actually is.

Validation Strategy

I used the test_classifier from the tester module to validate my algorithm. The test_classifier users StratifiedShuffleSplit. StratifiedShuffleSplit returns stratified randomized folds that preserve the percentage of samples for each class. This method is useful for this unbalanced dataset since it preserves the percentage of samples for each class. Otherwise the classifier could train to a sample that does not contain a POI and would not be useful in predicting POIs. Machine learning models are sensitive to balance sampling. StratifiedShuffleSplit samples separately for testing and training to avoid overfitting, although there is no gurantee that every sample will be different. The test_classifier uses 1000 folds which means it samples, trains and tests 1000 times.