Udacity Data Analyst Nanodegree
P5: Identify Fraud from Enron Emails

Megan O'Neil
2017-02-04

Goal of Project

In 2000, Enron was one of the largest companies in the United States. By 2002, it had collapsed into bankruptcy due to widespread corporate fraud. In the resulting Federal investigation, a significant amount of typically confidential information entered into the public record, including tens of thousands of emails and detailed financial data for top executives.[1]

The goal of this project is to build a person of interest identifier based on public email and financial data to identify individuals of interest in the Enron scandal, defined as those who were indicted, reached a settlement or plea deal with the government, or testified in exchange for prosecution immunity. In the dataset, the public financial and email data was combined with a hand generated list in the fraud case of who the persons of interest were. This project allows us to train a machine learning algorithm to see if we can determine which features of our data correlate with persons of interest.

Outliers

There were several outliers in the data that I removed. One clear outlier in the financial data was for 'Total', which summed the benefits received by all listed indivdiuals. This outlier was removed. Another outlier in the financial data was for the individual, 'LOCKHART EUGENE E', for whom the value of all financial features was 0. This individual also removed.

Feature Selection

I ended up using the following four features in my POI identifier, with feature scores of each following:

- bonus: 30.08
- total_stock_value: 15.96
- salary: 19.03
- fraction_to_poi: 15.60

I used 'SelectKBest' to select these features out of all of the financial and email features provided, excluding 'email_address'. I used GridSearchCV to try the estimator with multiple parameter values for K, from 1 to 14, in order to find the optimal number of features to include. The other parameter I passed for SelectKBest was the socre function 'f_classif', which computes the ANOVA F-value for the provided sample.

I used the linear scaling function, 'MinMaxScaler' in my pipeline operator. I used feature scaling because I used the Gaussian Naïve Bayes algorithm, which would be affected by features that have much larger numeric values. Since SelectKBest uses ANOVA to gauge the potential of efficacy that each feature will have during the model-fitting

[1] This section of summary is from Udacity Project Overview.

stage, linear changes to feature values from MinMaxScaler do not affect the ANOVA F-statistics computed by SelectKBest.

I included three new features in the dataset, 'fraction_to_poi', 'fraction_from_poi', and 'avg_fraction_to_from_poi'. These features were based on the fraction of emails sent between the individual and POIs; with the average fraction being a simple average of the fraction of emails to POIs and the fraction of emails from POIs.

Algorithm Selection and Performance

I used Gaussian Naïve Bayes. I also tried SVC, but model performance was lower, holding all other factors constant. I used the same approach of applying the MinMaxScaler and SelectKBest, using the GridSearchCV to optimize K from a range of values from 1-14. Using SVC, only 'bonus' was selected. While precision and recall were high for the non-POI class (0.93 and 0.96 respectively), they were 0 for the POI class.

Algorithm Tuning

Machine learning algorithms have parameters so that they can be tuned for a specific problem. Finding the best combination of parameters to optimize an algorithm is tuning. If not done well, the algorithm may not generate the best classifer for the data.

Gaussian Naïve Bayes has limited parameter tuning capability, so I did not tune parameters. I did use GridSearchCV to tune the parameters of SelectKBest.

With SVC, there would be multiple parameters to tune:

- Kernel: enables decision boundary to be linear or non-linear.
- Gamma: which defines how much influence a single training example has, the larger the gamma, the closer other examples must be to be affected.
- C: controls tradeoff between a smooth decision boundary and classifying training points correctly; low C makes decision surface smooth, high c aims at classifying all training examples correctly.

I would include multiple options for each parameter, then used GridSearchCV() to optimize them.

Validation

Validation is used to give an estimate of the performance of your estimator on an independent dataset, this is when the estimator that you have trained on the training porition of your dataset is applied to the 'test' portion of your dataset to determine how accurate the predictor is. A classic mistake is to validate your estimator on the entire dataset, or on the same data that was used to train the estimator. This will give inflated accuracy results, because you are using the exact same data that was used to train the estimator, so it should be very accurate.

I validated my results by splitting my data into training and testing datasets using sklearn cross_validation.train_test_split. The test size was 20% of the dataset. I also used StratifiedShuffleSplit to create an iterator for cross-validation in the GridSearchCV parameter optimization step.

Ultimately, I used my classifier to predict the class of the test data (pred = clf.predict(features_test).

Evaluation Metrics

My final algorithm returned the following Classification Performance Report:

Class	Precision	Recall	F1-score	Support
0.0	0.96	0.96	0.96	27
1.0	0.50	0.50	0.50	2
Avg/total	0.93	0.93	0.93	29

Where:

- Class refers to POI (0.0) or non-POI (1.0)
- Precision is equal to the ratio of true positives to all predicted positives
- Recall is equal to the ratio of true positives to all actual positives
- f1-score measures the accuracy using precision and recall, weighting them equally
- Support refers to the number of samples of the true response that lie in that class

For non-POIs, precision, recall and f1-score were all 0.96 which means that there is a 96% chance that the algorithm was correct when it predicted someone was not a POI, and a 96% chance that it would detect a non-POI. For POIs, precision, recall and f1-score were all 0.5, which means that there was a 50% chance that the algorithm was correct when it predicted someone was a person of interest, and 50% change it would detect a person of interest.

References

I used the following references in developing this project:

- Stratified Shuffle Split: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedShuffleSplit.html
- GridSearchCV: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
- Classification Report: http://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html,
- http://stats.stackexchange.com/questions/117654/what-does-the-numbers-in-the-classification-report-of-sklearn-mean
- f_classif: http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html#sklearn.feature_selection.f_classif
- f1-score: https://www.kaggle.com/wiki/MeanFScore
- cross-validating: http://nbviewer.jupyter.org/github/jming/cs109/blob/master/lec_10_cross_val.ipynb
- Udacity forum posts to develop code:
  - https://discussions.udacity.com/t/gridsearchcv-and-kfold-validation/32459/6
  - https://discussions.udacity.com/t/dont-know-how-to-validate/201521
  - https://discussions.udacity.com/t/feature-selection-vs-pca/34958
  - https://discussions.udacity.com/t/valueerror-the-least-populated-class-in-y-has-only-1-member-which-is-too-few/44984

- https://discussions.udacity.com/t/final-project-order-of-pipeline-operation-with-minmaxscaler-and-svc/164802/2

I hereby confirm that this submission is my work. I have cited above the origins of any parts of the submission that were taken from Websites, books, forums, blog posts, github repositories, etc.