P5. Identify fraud from Enron email dataset

Project goal and dataset description

The goal of this project is to try to build algorithm which can find Enron employees who committed fraud using dataset that contains financial information and data from Enron emails dataset. Dataset consists of 146 observations across 20 variables. 19 variables contain quantitative data and one - textual (e-mail). 128 of observations are non POI and 18 are POI so we have here dataset with imbalanced classes (88 % of observations belongs to “not POI” class).
Several features have many missing values:

Feature	Missing values	%
deferral_payments	107	73
restricted_stock_deferred	128	88
loan_advances	142	97
director_fees	129	88
deferred_income	97	66

In search for outliers I plotted histograms for every feature. “Salary” has a big outlier above 25 millions, this data point is from row named “Total”, so I deleted this row. Also I deleted “THE TRAVEL AGENCY IN THE PARK” that is not a person and “LOCKHART EUGENE E” that has no data.
Many features also have outliers, but those look like legitimate data points and often belong to POI so I decided to save it.

Features

I created one new financial feature as a ratio of “exercised stock options” to the sum of “salary” and “bonus”, because I thought that people who sold more stocks relative to their income possibly had information about real state of affairs in the company; and two based on email counts: “to_fraction” as a ratio of messages from POI to this person to all messages and “from_fraction” as a ratio of massages to POI from this person to all messages because this will select not people who just write and receive more mail but people who spent more time communicating with POI.
I used SelectKBest for future selection and used grid search to find number of features to be selected. First I this method on initial set of features used and got this result with k = 5 and best features ['exercised_stock_options', 'total_stock_value', 'bonus', 'salary', 'deferred_income'] -
Accuracy: 0.86086	Precision: 0.51686	Recall: 0.39850	F1: 0.45003	F2: 0.41763

After that I added my features to the list and again used SelectKBest and grid search. This time I also get best result at k = 5 but with another list of features: ['exercised_stock_options', 'total_stock_value', 'bonus', 'salary', 'from_fraction'] but with added features model performance got lower - Accuracy: 0.85929 Precision: 0.51070 Recall: 0.35800 F1: 0.42093 F2: 0.38077.
Feature scores:
exercised_stock_options - 24.815080, total_stock_value 24.182899, bonus 20.792252, salary 18.289684, from_fraction 16.409713, deferred_income 11.458477, long_term_incentive 9.922186, total_payments 8.772778, shared_receipt_with_poi 8.589421,loan_advances 7.184056, expenses 6.094173, from_poi_to_this_person 5.243450, other 4.187478, to_fraction 3.128092, from_this_person_to_poi 2.382612, director_fees 2.126328, to_messages 1.646341, fin1 0.603212, from_messages 0.169701, restricted_stock_deferred 0.065500.
But overall algorithm performance got lower, so for final algorithm I used first best feature combination : exercised_stock_options, total_stock_value, bonus, salary and deferred_income. I haven't used scaling for the best algorithm, but tried scaling for K nearest neighbours and SVM there it is required.

Algorithm

The best result was achieved using Naive Bayes algorithm (precision: 0.510706, recall: 0.35800, f1: 0.42093). K-Nearest Neighbours was the second best (precision: 0.32555, recall: 0.32750, f1: 0.32652) and I also tried SVM and Random Forest.

Parameter tuning

Parameter tuning is a process of searching for values of hyper-parameters that maximize algorithm performance. For this task I used GridSearchCV for every algorithm except Naive Bayes which don't have parameters, and after that chose the best performing variant. For Random forest I tuned 2 parameters with following values: n_estimators':[10,50,100,200], 'max_features':[17,10,5,"auto"] using GridSearchCV. For K Nearest Neighbours I tuned number of neighbours with values [1,3,5,10].

Validation

To validate an algorithm is to check its ability to generalise on unseen data. Without validation it is easy to overfit classifier so it will get perfect performance on train data but will not give good result on new data. Usually to avoid this data are divided in two sets: train data and test data, and only train data are used to fit classifier but in this case we have small amount of data and this division lower available data even more. So for validation I used cross-validation on all available data using stratified shuffle split to make splits with the same proportion of classes.

Evaluation

For algorithm evaluation I used F1, precision and recall because of class imbalance, which make accuracy misleading as classifier that always predict negative label gets higher accuracy. Precision shows how many positive results are true positives. Recall shows that fraction of positives was identified. I can’t use just recall because maximizing recall can lower precision below acceptable level so I use F1 as harmonic measure. Recall is more important for this task of classifying POI because it is better to identify more POI even if number of false positives will get higher. My best result using Naive Bayes classifier - precision: precision: 0.510706, recall: 0.35800, f1: 0.42093.