Intro to Machine Learning

Final project

21 June 2015

Erni Durdevic

Identifying Fraud from Enron Email

Enron was one of the major US Companies in 2000 and bankrupted in 2002 for Corporate fraud. This project is based on a dataset of a combination of financial and e-mail data from Enron.

I tried to apply and tune a machine learning algorithm which can identify Persons of interest (also called POI) based on financial and aggregated e-mail data.

Data Exploration

I started the analysis by exploring the dataset to gain confidence over the data.

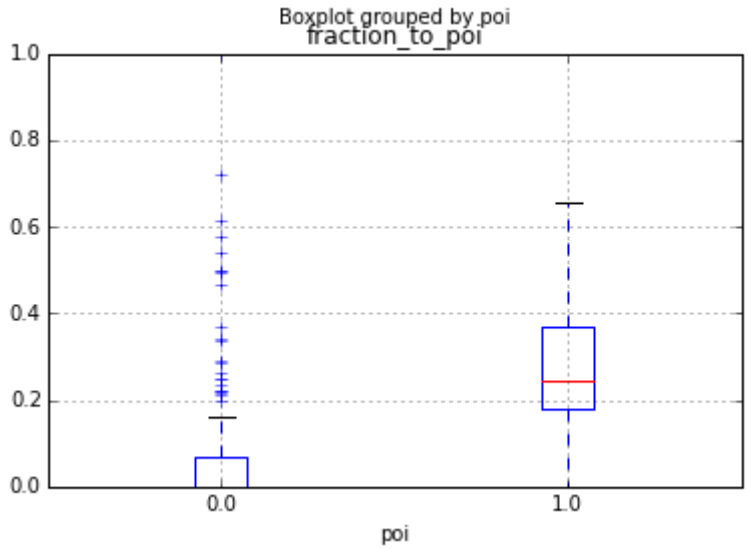
To see the complete analysis, Evaluators can re-run the submitted code setting the "print_exploratory_box_plots" variable to True. Here I will summarize the most interesting results.

- Total number of records: 145
- Total number of non-empty records: 138
- Number of POI: 18
- Number of non POI: 127
- Number of features in the initial dataset: 21
- Number of new created features: 2

The following features had many missing values:

Feature	Count of values (out of 145 records)
deferral_payments	38
deferred_income	48
restricted_stock_deferred	17
loan_advances	3
director_fees	16

An interesting box-plot displays the new created feature "fraction_to_poi" that represents the number of email sent to other POIs divided by the total number of emails sent.



Outlier investigation

The dataset contained only one outlier, it was the item 'TOTAL'. That value contained total amount paid for salary, bonuses etc. I spotted it by plotting the scatter plot of salary and bonus, then removed it by dropping the element with key 'TOTAL' from the input dictionary.

Final features

I ended up using the following features:

- 'poi',
- 'salary'
- 'exercised_stock_options'
- 'bonus'
- 'shared_receipt_with_poi'
- 'deferred_income'
- 'fraction_to_poi' (computed feature = 'from_this_person_to_poi' / 'from_messages')

Feature selection

I identified the above features combining intuition and analysis tools. I firstly identified the features that may be interesting, then I applied Lasso regression and Decision Tree algorithm (with default parameters) to analyze Lasso's 'coef_' and Decision Tree's 'feature_importances_'.

1) I analyzed the Lasso 'coef_' result, but it was only useful until I added the new 'fraction_to_poi' and 'fraction_from_poi' features. When I added the two new features, it skewed the 'coef_' values with zero values for the two new variables. This result was opposed to what I was seeing in the 'test_classifier' method results, so I stopped trusting lasso 'coef_' and continued feature selection with Decision Tree feature importances.

After all, the POI / non-POI prediction is much more suited to a decision tree (with boolean results) than to a linear regression (with continuous variable result).

My conclusion was that the Lasso algorithm is not suited to this specific problem.

2) I analyzed Decision Tree (with default parameters) 'feature_importances_', and I kept removing low importance features until I got the following feature set:

 # Classifier used:

DecisionTreeClassifier(compute_importances=None, criterion='gini',

max_depth=None, max_features=None, max_leaf_nodes=None,

min_density=None, min_samples_leaf=1, min_samples_split=2,

random_state=None, splitter='best')

 # Score on test_classifier:

Accuracy: 0.81879 Precision: 0.35695 Recall: 0.33500 F1: 0.34563 F2: 0.33917

Total predictions: 14000 True positives: 670 False positives: 1207 False negatives: 1330 True negatives: 10793

#Feature importances:

feature[salary], val: 0.259130658436
feature[exercised_stock_options], val: 0.311408469744

feature[bonus], val: 0.0128265342637
feature[shared_receipt_with_poi], val: 0.0478395061728
feature[deferred_income], val: 0.25158804126
feature[fraction_to_poi], val: 0.117206790123

Features importance

The final features importance (with tuned decision tree classifier) were the following:

Feature	Feature importance
salary	0.1939
exercised_stock_options	0.3168
bonus	0.0033
shared_receipt_with_poi	0.1792
deferred_income	0.1254
fraction_to_poi	0.1811

Feature scaling

I used feature scaling only in algorithm evaluation of k-means algorithm.

Since DecisionTreeClassifier, DecisionTree and Naive Bayes algorithms are not sensitive to feature scaling, there was no need to scale features in those situations.

In particular, the chosen algorithm, Naive Bayes, has the "naive" assumption of independence between every pair of features, therefore by definition there is no need for feature scaling.

New features

Based on the intuition that POIs interact more with each other, there were two features worth considering: 'from_poi_to_this_person' and 'from_this_person_to_poi'. This two features have to be adjusted by the total number of email received/sent in order to be meaningful for our purpose. Therefore I created two new features:

'fraction_from_poi' = 'from_poi_to_this_person' / 'to_messages'
'fraction_to_poi' = 'from_this_person_to_poi' / 'from_messages'

Of the two created features, surprisingly, only the 'fraction_to_poi' was having a sensible impact on feature importance. Therefore I removed the 'fraction_from_poi' and kept only the 'fraction_to_poi' feature.

Pick an algorithm

I ended up using GaussianNB because it gave me the best compromise between accuracy and recall scores. I also tried DecisionTreeClassifier, KMeans with feature scaling, RandomForestClassifier and SVC.

After tuning the algorithms that can be tuned with GridSearchCV, I got comparable results to GaussianNB algorithm, but the best scores were achieved with GaussianNB.

Random Forest

For instance I was able to tune the RandomForestClassifier to score 0.54078 in Precision, but with only 0.242 in Recall.

```
RandomForestClassifier(bootstrap=True, compute_importances=None,
        criterion='gini', max_depth=None, max_features='auto'
        max_leaf_nodes=None, min_density=None, min_samples_leaf=1,
        min_samples_split=4, n_estimators=38, n_jobs=1,
        oob_score=False, random_state=1, verbose=0)

Accuracy: 0.86236      Precision: 0.54078      Recall: 0.24200 F1: 0.33437      F2: 0.27206
```

Decision Tree

The best tune I found for Decision Tree had both Recall and Precision scores above 0.3, but still scored less than GaussianNB:

```
DecisionTreeClassifier(compute_importances=None, criterion='gini',
        max_depth=None, max_features=None, max_leaf_nodes=None,
        min_density=None, min_samples_leaf=1, min_samples_split=2,
        random_state=1, splitter='best')

 Accuracy: 0.81850      Precision: 0.35527      Recall: 0.33200 F1: 0.34324      F2: 0.33641
```

GaussianNB

This algorithm had the best results, with 0.41 in Precision and 0.33 in Recall:

```
GaussianNB()

Accuracy: 0.83750      Precision: 0.41476      Recall: 0.33450 F1: 0.37033      F2: 0.34797
```

SVC

With SVC algorithm I was getting only extreme All-POIs (recall = 1.0, but very low precision) or Zero-POIs (This gave division by zero exception on test_classifier method) results.

The SVC tries to minimize the distance of the separation line with the nearest points of each group, but in this case there was no clear distinction between the groups, so the SVC algorithm was useless for my purpose.

Tune the algorithm

Tuning an algorithm means finding the input parameters that maximizes one or more evaluation metrics. If an algorithm is not well tuned it can lead to overfitting or to a poor evaluation metric (accuracy, precision or recall).

To tune the parameters in my project, for the classifiers that can be tuned, I used the sklearn GridSearchCV algorithm.

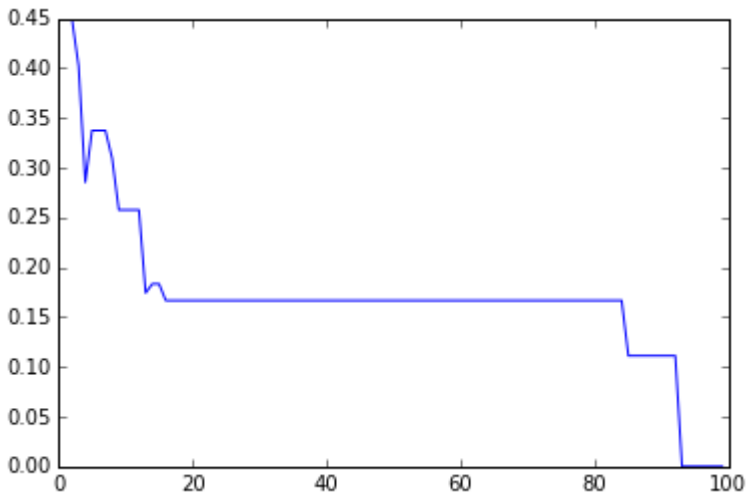
I started by tuning one parameter at a time and by plotting the 'resultgrid_scores_' in order to get an idea of the scoring of any parameter independently. Then I applied the GridSearchCV to multiple parameters restricting the range of each one to the most meaningful range found on the previews plot.

Finally I run the GridSearchCV with different target metrics (setting the 'score' parameter to "accuracy", "precision" or "recall") to better understand the characteristics of the algorithm.

Decision Tree Classifier tuning

I tuned the 'min_sample_split' parameter with different target scoring methods, and I obtained the following scores:

'precision' by 'min_sample_split'



The best estimator had 'min_sample_split' = 2. I searched the sklearn documentation for the confirmation that the peak in 'min_sample_split' = 2 was not due to overfitting, and I found that the 'grid_search' uses a cross-validation of 3 folds by default.

I concluded that the results were accurate and that the best estimator was the following:

DecisionTreeClassifier(compute_importances=None, criterion='gini',

max_depth=None, max_features=None, max_leaf_nodes=None,

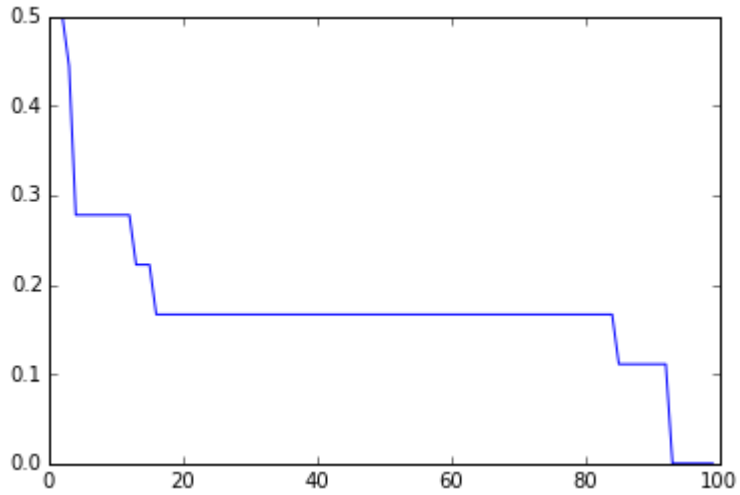
min_density=None, min_samples_leaf=1, min_samples_split=2,

random_state=1, splitter='best')

Accuracy: 0.81850 Precision: 0.35527 Recall: 0.33200 F1: 0.34324 F2: 0.33641

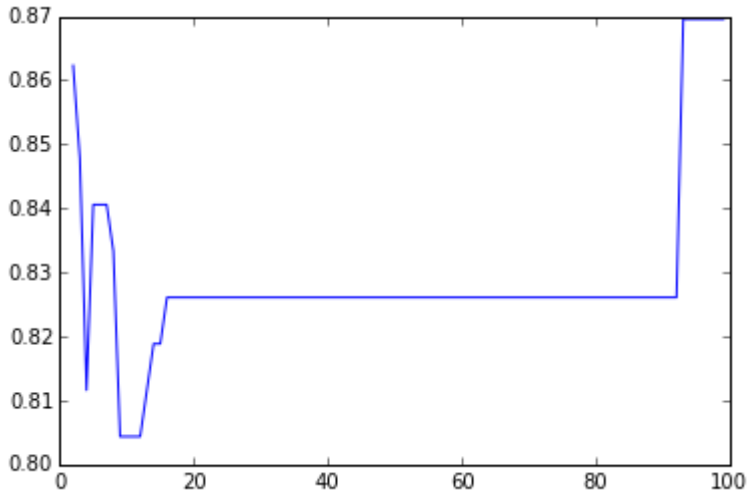
Total predictions: 14000 True positives: 664 False positives: 1205 False negatives: 1336 True negatives: 10795

'recall' by 'min_sample_split'



With recall scoring method, I obtained the best estimator with 'min_sample_split' = 2, as with precision scoring method.

'accuracy' by 'min_sample_split'



In this case the best estimator had a very high 'min_sample_split' value. This is due to the fact that POIs are few over the total, so an algorithm that guesses "all points are non-POI" has the best chances to gain a high accuracy.

This is not what we were looking for, because it skews completely the recall performance of out classifier.

Validation strategy

Validation is the process of checking the performance of a machine learning algorithm. A classic mistake in validation process is using the same dataset for training and performance testing. This approach can easily lead to overfitting.

I validated my analysis using the provided 'test_classifier' method, which uses StratifiedShuffleSplit algorithm (which is based on StratifiedKFold and ShuffleSplit algorithms).

This approach is good for small datasets, because it runs the training and testing multiple times over the same dataset, splitting training and testing data randomly each time.

Evaluation metrics

Since the dataset I was investigating was skewed (it has much more negative values for POI rather than positive values), as seen in "Tuning algorithm" section, accuracy was not a good metric to be used.

I focused on precision (how well the algorithm minimizes false positives) and recall (how well the algorithm minimizes false negatives).

Precision

The best precision score I got was 0.54, which means that for every true positive POI, the algorithm identified about another false positive POI. On average my algorithms scored from 0.3 to 0.4, the low values are due to the low number of POIs in the dataset.

Recall

The average values I got for recall were around 0.3, although Random Forest algorithm scored significantly lower. Even running GridSearchCV with Random Forest algorithm and target metrics set to 'recall' it was not able to score more than 0.24.