Udacity – Data Analyst Nanodegree, Project 4

Intro to Machine Learning

Giulio Ministeri

1.

The goal of this project is to create a predictor capable of detecting people who were involved in some way in the Enron fraud, based principally on the analysis of their financial data and sent and received e-mails. Machine learning algorithms come in handy in these cases, where there are no specific features or ease-to-detect patterns in the data such that one can simply checks for these evidences and reliably predicts the outcomes. Heuristics solutions, i.e. long list of "if...then...else", are unfeasible, while machine learning algorithms learn, reshape and rework input features for us giving back a "box" ables to predict the output if fed with new inputs.

The dataset includes e-mails and finance data of about 146 Enron employees; 18 of them are POIs (person of interest). For each employee there are 21 finance-related features, whose values are not always consistent; missing values are labeled as "NaN". Email-related features includes a set of summary infos like total received and sent mails, received and sent mails to poi people. Finally also the entire mail dataset is provided (about 150 thousands emails) together with a useful set of file which indexes all the emails by email-address: for each person there are two files which contains the relative paths to sent and received emails.
As first, I analyzed finance data looking for outliers. The analysis has been made in 4 steps i) I removed employees who has less than the 25% of features valorized ii) I removed the features which have less than 25% entries valorized; iii) I used linear regression (using "total_payments" feature as basis) to get an outliers list for each feature iv) I merged the lists by computing, for each name, how many times it appears in these lists; iv) I removed the entries for which the total count where more than a threshold. I ended up with 114 nonpois and 16 pois.

2.

After the outlier detection and removal described above I used tree-based feature selection using the entire dataset as base for the tree learning procedure. I did not use any feature scaling since it is not necessary when using trees. I removed all the features for which the importance was less than 1% of the total. I ended up using these original features:
"
salary - importance: 0.0334957653128
total_payments - importance: 0.0697672793439
bonus - importance: 0.0376703015518
total_stock_value - importance: 0.20659605532
shared_receipt_with_poi - importance: 0.0401087671012
from_poi_to_this_person - importance: 0.0402868094026
exercised_stock_options - importance: 0.0844430572247
from_messages - importance: 0.0175292882474
other - importance: 0.0821453852443
from_this_person_to_poi - importance: 0.062352803036
deferred_income - importance: 0.130920898752
expenses - importance: 0.0733546492809
restricted_stock - importance: 0.0422604228233
long_term_incentive - importance: 0.0642953860411

"

I created two sets of new features; the former was created by manipulating these original features; the latter by elaborating the huge e-mail dataset.
As first part I took the most important features (importance >= 0.1) and created the log and squared version of them, I also added the mixed products of them.
The second set of features has been created by elaborating the email dataset looking for the 10 most important words to detect pois. Since the dataset is very large and my PC was unable to elaborate all mails at once, I applied a procedure in which a random subset is repeatedly picked from the set (keeping the pois-nonpois ratio) and a dictionary of most important word is updated every loop to save and accumulate the importance coefficients. At the end the most important words are chosen based on the accumulated importance coefficients.
These features were finally tested to check their gains in the classifier. The final features list is the following:
deferred_income^2 importance: 0.0514773040273
total_stock_value*deferred_income importance: 0.0143809794781
deferred_income importance: 0.0172014021245
from_poi_to_this_person importance: 0.0307847995208
locat importance: 0.0157891159757
shared_receipt_with_poi importance: 0.113615975724
other importance: 0.0254023460068
log_total_stock_value importance: 0.0630175034308
long_term_incentive importance: 0.0320149313404
total_messages importance: 0.017324721161
colwellhouectect importance: 0.0180225478794
bonus importance: 0.0864072664056
total_stock_value importance: 0.13866454412
from_this_person_to_poi importance: 0.0291452113248
expenses importance: 0.0483518178469
restricted_stock importance: 0.0505022129125
7133457774 importance: 0.0225824186553
salary importance: 0.0173412038361
total_payments importance: 0.048032193811
total_stock_value^2 importance: 0.0857804082342
exercised_stock_options importance: 0.0590175307631

3.

I tested 4 different algorithms: DecisionTree, RandomForest, Adaboost and SupportVectorMachine.
All these 4 algorithms has been pipelined with PCA algorithm; I also added feature MinMaxScaling only to SVM pipeline, since it is the only algorithm that needs this step to perform better.

4.

Machine learning algorithms are generic models that can mimic a lot of different phenomena. Since they are general and can be used in an enormous variety of situations, they have a set of parameters that must be tuned to better fit the model or the situation in which they are used. Practically there is no mathematical formula to find the best parameters, so they are chosen based on try&test process where an initial set of possible parameters (chosen after some considerations on the situation and the input data) are used and performances are tested. No tuning means that the algorithm will use default parameters which probably will be a way distant from the best ones, resulting in an useless predictor.
I used gridsearchCV to tune parameters of the different algorithms, here the list of algorithms and tuned parameters for each:

PCA: number of components
Decision tree: minimum samples to split
Random Forest: number of estimators
AdaBoost : number of estimators
SVM: C parameter
GridSearchCV has been set to use “f1” as score indicator. I used sklearn train_test_split to create two subset of data, one for classifier training and one for testing and evaluating the algorithm.

5.

Validation process is important in evaluating the classifier performances. It is a very important step in order to avoid overfitting: a circumstance where the model has learned to replicate the data used to train it, and not replicate the general phenomenon.
As explained in 4.) I splitted the data into two subsets, used to train and validate the trained models respectively.
Finally, for test and compute classifier performances I used the tester script from the course base code.
The final classifier I chose is a pipeline of a PCA with 10 elements and a decision tree classifier with min_samples_split parameter equals to 2. The final performances are:

```
Accuracy: 0.79969       Precision: 0.33676      Recall: 0.31150 F1: 0.32364     F2: 0.31624
        Total predictions: 13000        True positives:  623    False positives: 1227   False negatives:
1377   True negatives: 9773
```

REFERENCES:
[1] Sklearn packet documentation: http://scikit-learn.org/stable/index.html
[2] Stackoverflow: http://stackoverflow.com/
[3] Learning python animal book from O'Reilly
[4] http://machinelearningmastery.com/
[5] udacity data analyst nanodegree forum