Udacity – Data Analyst Nanodegree, Project 4

Intro to Machine Learning

Giulio Ministeri

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.

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.

Considering the number of feature provided and the potentially big number of features that can be extracted from raw emails, the dataset cardinality is very small and skewed. Only 146 employees of which only 18 are POIs. I need to pay lot of attention to overfitting, and also adopt a more sophisticated evaluation metric like precision&recall or F1. Simple ones like accuracy is useful only in balanced dataset where positives and negatives entries have almost the same cardinality.

Also the evaluation process needs to be chosen by considering this dataset leak. Train/test split can be used in very big dataset where removing lot of entries for testing does not affect the train step performances.

As first, I analyzed finance data looking for outliers. After just showing the labels I found the two employees names "TOTAL" and "THE TRAVEL AGENCY IN THE PARK", which of course are not real employees and must be removed.

The outliers analysis has been made with the following steps

i) I classified the entries based on the number of features with meaningful values, resulting in (25 has been chosen as simple threshold to split dataset in 4 subsets):

6 employees with less than 25% valued features,
37 employees with 25% to 50% valued features,
52 employees with 50% to 75% valued features,
49 employees with more than 75% valued features.

After analyzing the 11 employees belonging to lessThan25% set I found out that none of them are POIs so I decided to removed them from the dataset. Here it is the list:{WODRASKA JOHN, WHALEY DAVID A, WROBEL BRUCE, SCRIMSHAW MATTHEW, LOCKHART EUGENE E, GRAMM WENDY L}

ii) I classified the given features based on the number of entries with meaningful values, resulting in (25 has been chosen as simple threshold to split dataset in 4 subsets):

3 features with less than 25% valued entries,

3 features with 25% to 50% valued entries,
10 features with 50% to 75% valued entries,
6 features with more than 75% valued entries.

As for the entries, I decided to remove those features with less than 25% valued entries. Here it is the list: {restricted_stock_deferred, loan_advances, director_fees}.

iii) I used linear regression (using “total_payments” feature as basis) to get an outliers list for each finance-related feature (i.e. above the .9 percentile).

iv) I merged the lists by computing, for each name, how many times it appears in these lists. Here it is the final list:

employees who are outliers in 1 features: 22
employees who are outliers in 2 features: 9
employees who are outliers in 3 features: 6
employees who are outliers in 4 features: 1
employees who are outliers in 5 features: 1
employees who are outliers in 6 features: 0
employees who are outliers in 7 features: 0

v) Same process for email-related features using “from_messages” as base feature to analyze “from_this_person_to_poi” feature, and “to_messages” as base feature to analyze ["from_poi_to_this_person", "shared_receipt_with_poi"] features. Here it is the final list:

employees who are outliers in 1 features: 14
employees who are outliers in 2 features: 5
employees who are outliers in 3 features: 1

vi) I analyzed the compositions of these subsets and realized that among the most occurring outliers, two names are POIs. So I decided not to remove any entries but instead to modify values of those features which are outliers with the ones predicted by linear regression.

2.

For feature selections I first tried treebased feature selection, but after noticing that the importance coefficients and ranks where totally different after more than one tries, I decided to move to another solution. I used a method in which at each iteration I remove a feature from the dataset, and then, using a stratified shuffle cross-validation, I train a tree and accumulate the f1 score. Finally I ended up with these final scores:

removing salary, score: 121.5
removing to_messages, score: 108.833333333
removing deferral_payments, score: 109.666666667
removing total_payments, score: 121.866666667
removing exercised_stock_options, score: 107.9
removing bonus, score: 114.666666667
removing restricted_stock, score: 124.5
removing shared_receipt_with_poi, score: 111.5
removing total_stock_value, score: 123.666666667
removing expenses, score: 112.166666667
removing from_messages, score: 114.9
removing other, score: 121.333333333
removing from_this_person_to_poi, score: 123.833333333
removing deferred_income, score: 114.166666667
removing long_term_incentive, score: 113.0
removing from_poi_to_this_person, score: 100.666666667

And finally, using 90 percentile on the scores, I decided to remove

I created two sets of artificial 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 and created the log and squared version and added the mixed products of these features.
Similarly to what I did for features selection, here I progressively added to the dataset the artificial features built on each possible pair of features (to be able to make the mixed product), and analyzed how the performances of a simple tree changed:

```
feature:  salary  score:  1340.56666667
feature:  to_messages  score:  1491.46666667
feature:  deferral_payments  score:  1548.16666667
feature:  total_payments  score:  1505.23333333
feature:  bonus  score:  1566.63333333
feature:  total_stock_value  score:  1355.96666667
feature:  shared_receipt_with_poi  score:  1521.96666667
feature:  from_poi_to_this_person  score:  1591.66666667
feature:  exercised_stock_options  score:  1884.0
feature:  from_messages  score:  1545.5
feature:  other  score:  1353.83333333
feature:  deferred_income  score:  1519.4
feature:  expenses  score:  1681.3
feature:  long_term_incentive  score:  1514.56666667
```

So I decided, based on 90 percentile of the scores, to add ['exercised_stock_options', 'expenses'], to the set of feature for which to create the artificial ones.

The second set of features has been created by elaborating the email dataset looking for the 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 words is updated every loop to save and accumulate the importance coefficients. At the end the most important words are chosen based on the accumulated coefficients. Before adding the words to the final dataset I removed all the words that were clearly references to phone numbers, email addresses, or peoples' names.

Then I added the two artificial features sets to the final dataset and tested again looking for useless features. The procedure is the same of the initial feature selection.
The final features I removed, based on 90 percentile of the scores, are:
['exercised_stock_options*expenses', u'broadband', u'rosali', 'other', u'committe', 'total_payments', u'embed']
As expected most are coming from the artificial features, except for “total_payments”.

The final dataset is the following:
['salary', 'to_messages', 'deferral_payments', 'exercised_stock_options', 'bonus', 'shared_receipt_with_poi', 'total_stock_value', 'expenses', 'from_messages', 'deferred_income', 'long_term_incentive', 'from_poi_to_this_person', 'exercised_stock_options^2', 'log_exercised_stock_options', 'expenses^2', 'log_expenses', u'62602pst', u'andypst', u'best', u'board', u'calgerpdxect', u'calgerpdxectect', u'corp', u'ddelainnsf', u'ene', u'everi', u'execut',

u'flemingcorpenronenron', u'guy', u'gwhall', u'gwhalleynsf', u'hotlin', u'klay', u'legals', u'let', u'mayberryjpg', u'nonprivilegedpst', u'open', u'parti', u'prestohouectect', u'robust', u'southwood', u'spreadsheet', u'tjonesnsf', u'view', u'violationnotif']

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 and cross-validation folds set to 100. I used sklearn train_test_split to create two subset of data, one for classifier training and one for testing and evaluating the best reachable performances of the classifier.

The final result for each algorithm of gridsearchCV were:

decisionTree algorithm:

```
Accuracy: 0.80914	Precision: 0.33954	Recall: 0.35550	F1: 0.34734	F2: 0.35219
	Total predictions: 14000	True positives:  711	False positives: 1383	False negatives:
1289	True negatives: 10617
```

Random forest result:

```
Accuracy: 0.82814	Precision: 0.32193	Recall: 0.18350	F1: 0.23376	F2: 0.20077
	Total predictions: 14000	True positives:  367	False positives:  773	False negatives:
1633	True negatives: 11227
```

SVC result:

```
Accuracy: 0.70671	Precision: 0.15498	Recall: 0.23650	F1: 0.18725	F2: 0.21399
	Total predictions: 14000	True positives:  473	False positives: 2579	False negatives:
1527	True negatives: 9421
```

adaboost result:

```
Accuracy: 0.81807	Precision: 0.29452	Recall: 0.19600	F1: 0.23536	F2: 0.21005
	Total predictions: 14000	True positives:  392	False positives:  939	False negatives:
1608	True negatives: 11061
```

And the final algorithm chosen is :

```
Pipeline([ ("pca", PCA(n_components = 10)), ("tree", tree.DecisionTreeClassifier
(min_samples_split= 7))])
```

5.

The validation process is important to test if the classifier is working correctly. 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 to replicate the general phenomenon.

Moreover, a common mistake when making cross-validation is not shuffling the data. By analyzing data with statistical (mean, variance) or graphical (scatter plots, histograms) tools, we ignore how raw data are ordered, and sometimes it could happen that data are grouped by the output label (e.g.: first all nonPOIs, then all POIs); so by splitting entries as they are we create unbalanced subsets where almost all entries belongs to the same label.

As explained in 4.) I split the data into two subsets, used to train and validate the classifiers 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.

6.

I evaluated the final classifier using the test_classifier script. The two important metrics I am interested in are precision and recall. Precision metric explains the goodness of the model when saying the case is positive, a small value of precision means there are too many false positives.

On the contrary recall metric explains the goodness of the model in terms of false negative cases, the smaller the recall is, the more false negatives has been detected. F1 score is a summarizing metric that combines precision and recall by the harmonic mean:

F1 = precision * recall / (precision + recall)

The final performances of the classifier are:

Precision: 0.33676
Recall: 0.31150
F1: 0.32364