showcase_toolbox

March 6, 2016

1 Showcase of toolbox.py

As the title says, this is only a showcase.
For actual documentation, refer to toolbox.py

```
In [1]: import toolbox as tb
```

For first testing of classifiers, we can generate multidimensional toydata easy separable in two classes.

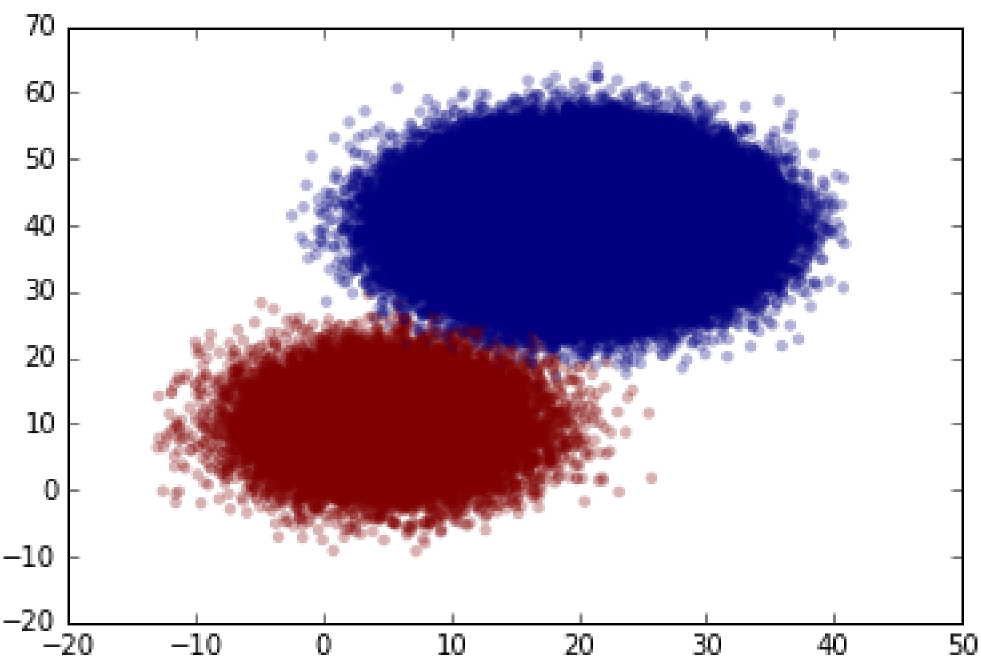
```
In [2]: toy_data = tb.createToyData(n = 550000,dim = 4,s_prob = 0.05)
        toy_weights = toy_data[:,0]
        toy_labels = toy_data[:,1]
        x = toy_data[:,2:4]
```

We also can visualize the data with pyplot.scatter.

```
In [3]: import matplotlib.pyplot as plt
        %pylab inline
        plt.scatter(x[:,0], x[:,1], edgecolor="", c=toy_labels, alpha=0.3)
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
Out[3]: <matplotlib.collections.PathCollection at 0x81e6cc0>
```



For the demonstration of more tools, we need to use kaggleData.py.

```
In [4]: import kaggleData as kD
        csv_data,csv_header = kD.csvToArray()
        sol_data,sol_header = kD.getSolutionKey(csv_data,csv_header)
```

We can calculate the AMS that would result from a perfect submission.

```
In [5]: tb.calcMaxSetAMS(sol_data)

Maximum AMS possible with this Data:
Public Leaderboard: (67.7111228951553, 691.98860771412183, 0)
Private Leaderboard: (67.711122895128, 691.98860771368709, 0)
```

This function basicly uses calcSetAMS() with the solutions labels as prediction. In general, we can calculate a AMS for any solution, using any prediction for the same amount of events.

```
In [6]: tb.calcSetAMS(toy_labels,sol_data)

Out[6]: ((0.24363377847776452, 35.22995747483769, 20888.029055869818),
         (0.2384823161428654, 34.415650297619372, 20804.23081820122))
```

Using the right decision threshold was key for succeeding in the challenge, we can estimate a good threshold using a bruteforce approach.

```
In [7]: soft_pred = np.random.rand(550000)
        tb.bestThreshold(soft_pred,sol_data,10)

Out[7]: (array([ 1.,  1.,  1., ...,  1.,  1.,  1.]), 1.079073517337079, 0.0)
```

From a prediction, we can produce an array containing all submission-relevant data.

```
In [8]: test = tb.createSubmissionArray(soft_pred)

In [9]: test

Out[9]: array([[  4.33265000e+05,   1.00000000e+00,   4.51658814e-06],
               [  7.12108000e+05,   2.00000000e+00,   4.78960215e-06],
               [  7.41817000e+05,   3.00000000e+00,   4.90349060e-06],
               ...,
               [  6.60094000e+05,   5.49998000e+05,   9.99996623e-01],
               [  4.11077000e+05,   5.49999000e+05,   9.99996778e-01],
               [  6.24393000e+05,   5.50000000e+05,   9.99997796e-01]])
```

There is a method to sort an array with respect to a column.

```
In [10]: tb.sortByColumn(test,1)

Out[10]: array([[  4.33265000e+05,   1.00000000e+00,   4.51658814e-06],
                [  7.12108000e+05,   2.00000000e+00,   4.78960215e-06],
                [  7.41817000e+05,   3.00000000e+00,   4.90349060e-06],
                ...,
                [  6.60094000e+05,   5.49998000e+05,   9.99996623e-01],
                [  4.11077000e+05,   5.49999000e+05,   9.99996778e-01],
                [  6.24393000e+05,   5.50000000e+05,   9.99997796e-01]])
```

We also are able to directly create a submissionfile.

```
In [11]: tb.createSubmissionFile(soft_pred,fname="toysubmission.csv",threshold=0.0)
```

We can reimport this submission data with kaggleData.csvToArray(). For a os-independent path, we us the os package.

```
In [12]: import os
         scriptFolderPath = os.path.dirname(os.getcwd())
         mainFolderPath = os.path.dirname(scriptFolderPath)
         submissionPath = (mainFolderPath + "/data/submissions/")
```

```
In [13]: sol_csv_data,sol_csv_header = kD.csvToArray(csvF = (submissionPath + "toysubmission.csv"), rows
```

```
In [14]: sol_csv_data
```

```
Out[14]: array([['350000', '399789', 's'],
                ['350001', '365561', 's'],
                ['350002', '275467', 's'],
                ...,
                ['899997', '465178', 's'],
                ['899998', '192544', 's'],
                ['899999', '281632', 's']],
               dtype='<U16')
```

We can access classification runs recorded with recordRun() with getRecord().
If no parameters are given, the standard record file "records_1.csv" is accessed.
For instruction how to use recordRun(), refer to showcase_sklearn.

```
In [15]: rec_data ,rec_header = tb.getRecord()
```

To see, what kind of data has been recorded, we can refer to the header provided by getRecord(). In the standard record file, we record runs of several classifiers. Often, these have different parameter options. To record all relevant parameters, the file provides multiple "Settings"-features.

```
In [16]: rec_header
```

```
Out[16]: ['Classifier',
          'Featurelist',
          'CV_Score',
          'PublicAMS',
          'PrivateAMS',
          'time_train',
          'time_pred',
          'Settings',
          'None',
          'None',
          'None',
          'None',
          'None',
          'None',
          'None',
          'None',
          'None',
          'None',
          'None',
          'None']
```

With following code, we can access all best runs of each used classification method.
First, we sort rec_data by public AMS.

```
In [17]: rec_pubams=tb.sortByColumn(rec_data,3)
```

Then, we iterate reversed through the data.
When we find a classifier for the first time, we expand an array called "best", an save the classifiers name to a list "gotIT". For the remaining data, we ignore all data classified by the methods contained in gotIT.

```
In [18]: gotIT = []
         best = []
         for row in reversed(rec_pubams):
             if row[0] not in gotIT:
                 gotIT.append(row[0])
                 best.append(row)
         best = np.array(best)
```

We now have a list of recorded classification runs, beginning with our best xgboost run.

```
In [19]: best
```

```
Out[19]: array([['xgboost', 'header_all', '   test-ams@0.14', '3.66421262680941',
                 '3.71268472156738', '1099.13236880302', '69.8130800724029',
                 'threshold=0.855', 'steps_=2500', 'depth_=9', 'eta_=0.01',
                 'subsample_=0.9', 'eval_1=auc', 'eval_2=ams@0.14', 'None', 'None',
                 'None', 'None', 'None', 'None'],
                ['gbc', 'header_all', '0.869924', '3.28927069645223',
                 '3.42808660853669', '8322.90483593940', '10.2775928974151',
                 'threshold=0.6666', 'trees_=100', 'depth_=12', 'eta_=0.01',
                 'subsample_=0.9', 'None', 'None', 'None', 'None', 'None', 'None',
                 'None', 'None'],
                ['kNN', 'header_6', '0.81892', '3.18534579809683',
                 '3.17144377327013', '1.15410709381103', '110.166372060775',
                 'threshold=0.7777', 'k=297', 'p=1', 'None', 'None', 'None', 'None',
                 'None', 'None', 'None', 'None', 'None', 'None'],
                ['log Reg', 'header_8', '0.73912', '2.04562322781762',
                 '2.07029542090901', '6.75238895416259', '0.44402599334716',
                 'threshold=0.4444', 'C=0.1', 'penalty=l2', 'None', 'None', 'None',
                 'None', 'None', 'None', 'None', 'None', 'None', 'None'],
                ['log Reg CV', 'header_3', '0.73507', '1.96216755909995',
                 '1.98149424825609', '33.3609240055084', '0.26706004142761',
                 'threshold=0.4444', 'Cs=1', 'penalty=l1', "scoring = 'roc_a",
                 'None', 'None', 'None', 'None', 'None', 'None', 'None', 'None',
                 'None']],
               dtype='<U16')
```