AM-ensembles

July 26, 2018

```
In [32]: np.argmax(np.bincount([0, 0, 1], weights=[0.2, 0.2, 0.6]))
```

```
Out[32]: 1
```

```
In [33]: ex = np.array([[0.9, 0.1],
                        [0.8, 0.2],
                        [0.4, 0.6]])

         p = np.average(ex,
                        axis=0,
                        weights=[0.2, 0.2, 0.6])
         p
```

```
Out[33]: array([ 0.58,  0.42])
```

```
In [34]: np.argmax(p)
```

```
Out[34]: 0
```

Sampling with and without replacement

```
In [3]: # Sampling is done with replacement by default
        np.random.choice(4, 12)
```

```
Out[3]: array([0, 2, 0, 3, 2, 0, 1, 3, 3, 1, 1, 2])
```

```
In [6]: # Probability weights can be given
        np.random.choice(4, 12, p=[.6, .1, .1, .2])
```

```
Out[6]: array([0, 3, 2, 0, 2, 0, 0, 0, 2, 0, 0, 0])
```

```
In [16]: import numpy as np

         l = np.array([1, 2, 1, 4, 1])
         ll = np.random.choice(l, size=l.shape, replace=False)
         print(ll)
```

```
[1 4 2 1 1]
```

```
In [11]: from sklearn.datasets import load_iris
         from sklearn import tree
         iris = load_iris()
         clf = tree.DecisionTreeClassifier()
         clf = clf.fit(iris.data, iris.target)

         import graphviz
         dot_data = tree.export_graphviz(clf, out_file=None)
         graph = graphviz.Source(dot_data)
         graph.render("iris")

Out[11]: 'iris.pdf'

In [30]: print(iris.data.shape)
         iris.data[130, :]
         clf.predict(iris.data[:5, :])

(150, 4)


Out[30]: array([0, 0, 0, 0, 0])

In [31]: # Alternatively, the probability of each class can be predicted, which is the fraction
         clf.predict_proba(iris.data[:1, :])

Out[31]: array([[ 1.,  0.,  0.]])

In [29]: teste = [1,2,3,4,5]
         teste[:3]

Out[29]: [1, 2, 3]

In [ ]:

In [24]: from sklearn.datasets import make_hastie_10_2
         from IPython.display import display
         x, y = make_hastie_10_2()
         df = pd.DataFrame(x)
         df['Y'] = y
         display(df)
```

	0	1	2	3	4	5	6	\
0	0.526489	-0.763472	-0.706099	0.654778	-0.565815	-1.163920	0.846913	
1	-1.318696	0.432710	-0.031463	-0.401374	-0.337138	-1.213596	0.835081	
2	-1.192153	-1.159879	0.861454	-0.119328	0.833415	-0.837962	1.698297	
3	0.375255	-0.605061	-0.541723	0.053363	0.189179	-1.015857	0.167054	
4	0.811765	1.916923	-0.080739	-0.694709	-1.329060	-0.430750	-0.482420	
5	-1.296543	-0.369127	-0.422352	0.372883	1.278859	-2.337437	0.727910	
6	0.250071	1.200617	-0.643814	1.437893	-2.090557	0.851917	1.668948	
7	0.013301	0.422407	-0.150393	0.631411	2.322823	0.385596	2.437358	

8	0.245101	-0.110315	0.705483	0.184241	1.068158	0.736703	0.281864
9	0.274644	0.526407	-0.834878	0.027094	0.837857	-1.156657	0.907028
10	1.142100	1.417411	1.819498	-1.278939	-1.318678	-2.681461	0.056319
11	0.371569	-1.014978	0.513801	-0.077230	0.835562	0.553499	-0.494077
12	-0.081864	0.031625	-1.780543	1.577386	-0.098048	1.222356	-0.532805
13	1.327606	0.702513	-0.198589	1.407143	1.528707	-0.913372	2.834881
14	0.519636	-1.101724	-1.336848	-0.020455	-0.091789	-0.190226	-0.734797
15	2.308434	-1.842399	-0.069674	1.199714	0.905045	1.422081	-1.203756
16	0.968988	1.159536	-0.090298	-0.795489	-1.879691	-0.609462	-0.031963
17	1.163154	0.474395	-0.873292	0.146899	-0.176655	1.403285	0.198605
18	0.127533	0.019968	1.191380	1.027255	0.534480	0.814959	-0.660350
19	1.089202	0.815778	-2.431271	-0.786966	0.624403	1.414131	-0.562824
20	-0.754441	0.912482	0.508400	0.004482	1.352651	-0.282787	0.555767
21	0.557033	1.061032	1.425449	0.558454	0.309121	-1.149186	-0.413166
22	1.650907	-0.258760	-0.298062	-0.237932	-1.365106	-1.589686	-1.285710
23	0.378216	-0.735968	0.292748	1.220280	-0.217425	0.218336	-1.220274
24	1.551062	0.676645	-0.290905	-1.156722	-1.457788	-0.883922	-0.523748
25	-0.290830	-0.687314	0.369137	0.804209	0.449898	0.256732	0.413443
26	-1.214589	0.169434	0.783958	-0.133586	0.397853	0.408437	-0.464077
27	-0.014068	0.020463	0.803283	-0.097168	-0.846492	-0.559036	-2.360080
28	-0.268629	-0.063358	-0.320452	0.290886	-0.042790	-0.204325	-0.882782
29	-0.436593	0.931700	-0.520863	-0.184296	1.688279	1.432762	-0.075142
...	...	...	...	...	...	...	...
11970	1.080741	-0.640717	0.414929	0.614026	2.226387	0.029775	-0.862624
11971	-1.268115	0.153634	0.895223	0.034351	-0.213982	-1.101690	0.988129
11972	-0.480021	-1.863952	-1.190842	-1.164138	0.832033	1.494588	0.279965
11973	-0.067623	-1.041970	1.172865	1.200973	-1.599791	1.169615	-0.953184
11974	-1.704883	-0.498871	-1.004914	-0.527970	0.309891	0.251293	-1.201269
11975	1.557923	-0.969496	0.563230	-0.480739	-2.323281	-0.691329	-0.573250
11976	-1.169137	0.039134	0.232872	-1.109050	-0.115647	-0.946910	-0.915234
11977	-1.722215	1.347106	-0.464266	0.609084	0.551980	-0.104018	0.483701
11978	-0.002813	0.265227	-0.417637	1.906815	-1.197670	0.515705	-0.494125
11979	-0.793057	1.190724	-0.463052	-1.164846	0.479067	1.198819	-0.472603
11980	0.100805	1.656682	1.941208	2.368337	-0.234566	1.098320	-0.718115
11981	0.248531	-1.598421	0.440405	1.046646	0.322505	2.334837	-0.145578
11982	0.011217	0.416999	0.414808	-2.723120	-0.593544	1.211548	-0.065637
11983	0.210086	-0.630936	-0.614442	0.320729	-1.641955	-0.247248	-0.067407
11984	-0.668115	-0.432296	0.308561	0.327818	1.073007	0.199158	1.029413
11985	0.353536	0.016554	-0.752170	0.105523	1.106742	0.460238	-0.688491
11986	0.769236	0.900629	-0.239908	1.662989	0.839961	1.600271	1.430358
11987	-0.861658	0.927582	1.499913	-0.218049	0.692330	0.799916	-1.158869
11988	0.052431	0.881241	-0.534316	0.385234	-0.082772	-1.260843	-0.450972
11989	0.616027	0.047517	-0.603585	1.426648	0.588184	0.027613	-0.505363
11990	-0.647235	-1.816768	-0.615161	-0.867092	1.807476	-0.235322	-1.448097
11991	0.425753	-0.234748	-1.947795	1.060467	-1.194524	-2.405082	0.241939
11992	1.215309	-1.125954	-1.408679	0.816547	-0.897964	-0.794168	0.447917
11993	-0.705002	-0.890012	-0.602105	0.790843	0.975386	0.307847	0.903072
11994	-1.320432	0.235913	0.114664	2.639769	-0.031322	1.342106	0.228880

11995	-0.593089	-0.240664	1.928806	0.543357	-1.806218	1.365356	0.937527
11996	-0.131980	1.080677	2.018846	0.021023	-0.228484	-0.365011	-0.749460
11997	1.933225	1.367346	-0.113121	-0.843168	-0.542130	-0.077084	-0.257551
11998	0.355506	-0.455227	-0.501207	0.359476	0.402211	0.946462	-0.354442
11999	1.196846	-0.823285	1.352338	0.153614	-0.433364	-1.292426	-0.852783

	7	8	9	Y
0	-0.087337	1.697751	-1.719061	1.0
1	0.406709	2.397764	-1.198686	1.0
2	-1.717610	1.203911	1.373523	1.0
3	0.487222	-0.266284	-1.252365	-1.0
4	0.308459	-0.204133	-0.398917	-1.0
5	-1.102067	0.965088	0.214855	1.0
6	-0.686672	0.677713	-1.497522	1.0
7	-0.587649	-0.470205	-0.538169	1.0
8	-0.064058	1.804906	0.195508	-1.0
9	2.042502	1.875864	0.523830	1.0
10	0.736869	0.793001	-0.839679	1.0
11	-0.040593	-1.907335	-0.512432	-1.0
12	0.665610	-0.931202	0.746039	-1.0
13	-0.671587	-0.342724	-0.258018	1.0
14	0.886841	-0.136725	1.423513	-1.0
15	0.276217	1.104350	0.186909	1.0
16	1.205557	0.937312	-1.641719	1.0
17	0.725696	-0.940340	2.086016	1.0
18	-0.096970	0.005468	-0.128997	-1.0
19	0.239094	0.308788	-0.255931	1.0
20	-1.016883	-1.350860	-0.377891	-1.0
21	0.557653	1.450935	-0.256716	-1.0
22	0.077498	-0.379991	-1.697518	1.0
23	-0.551126	-0.447186	-0.394474	-1.0
24	0.522219	1.138418	-0.115384	-1.0
25	-0.268746	-1.106405	-0.291310	-1.0
26	1.312108	0.440888	-0.430293	-1.0
27	0.138802	-0.839130	-0.967505	-1.0
28	-0.715637	0.328914	-0.475876	-1.0
29	1.153781	0.062349	2.511368	1.0
...	...	...	...	...
11970	-0.699141	-0.140362	-0.895713	-1.0
11971	-0.130478	0.158070	-1.020550	-1.0
11972	-1.901112	-0.390070	-2.531511	1.0
11973	0.109430	0.611519	-0.006195	-1.0
11974	-0.858465	-1.847225	-0.158563	1.0
11975	2.199009	0.528653	-2.122688	1.0
11976	-1.291573	-1.148093	-1.728172	1.0
11977	1.516035	1.023592	0.212098	-1.0
11978	-1.246913	-0.695033	0.269412	-1.0
11979	1.041798	1.830802	-0.585302	1.0

```
11980 -0.094834  0.265646 -0.234797  1.0
11981 -0.618502 -0.095953  1.512859  1.0
11982 -1.087121 -0.355205  0.257238  1.0
11983  1.495478 -0.819823 -0.562095 -1.0
11984  0.011328 -0.415676  0.503885 -1.0
11985 -0.472120  2.491637  0.646600  1.0
11986 -1.092227 -0.280400  0.483720  1.0
11987 -0.398752 -0.961400 -0.783196 -1.0
11988 -0.834626 -0.328984  0.051586 -1.0
11989  1.020306  1.089035  0.500788 -1.0
11990 -0.187300 -0.494765 -0.835635  1.0
11991  1.818089  0.107790 -1.016992  1.0
11992  1.444354  0.924593 -0.355847  1.0
11993  0.352290  0.830260 -1.257872 -1.0
11994  0.470425 -0.204348 -0.082697  1.0
11995 -0.676040 -0.826311  1.062377  1.0
11996  0.213097  1.588406  1.607310  1.0
11997  1.575491 -0.174692 -0.160170 -1.0
11998  0.351253 -0.543542 -0.803243 -1.0
11999 -2.559351 -0.686868  0.491907  1.0

[12000 rows x 11 columns]
```

```
In [8]: range(10, 410, 10)
```

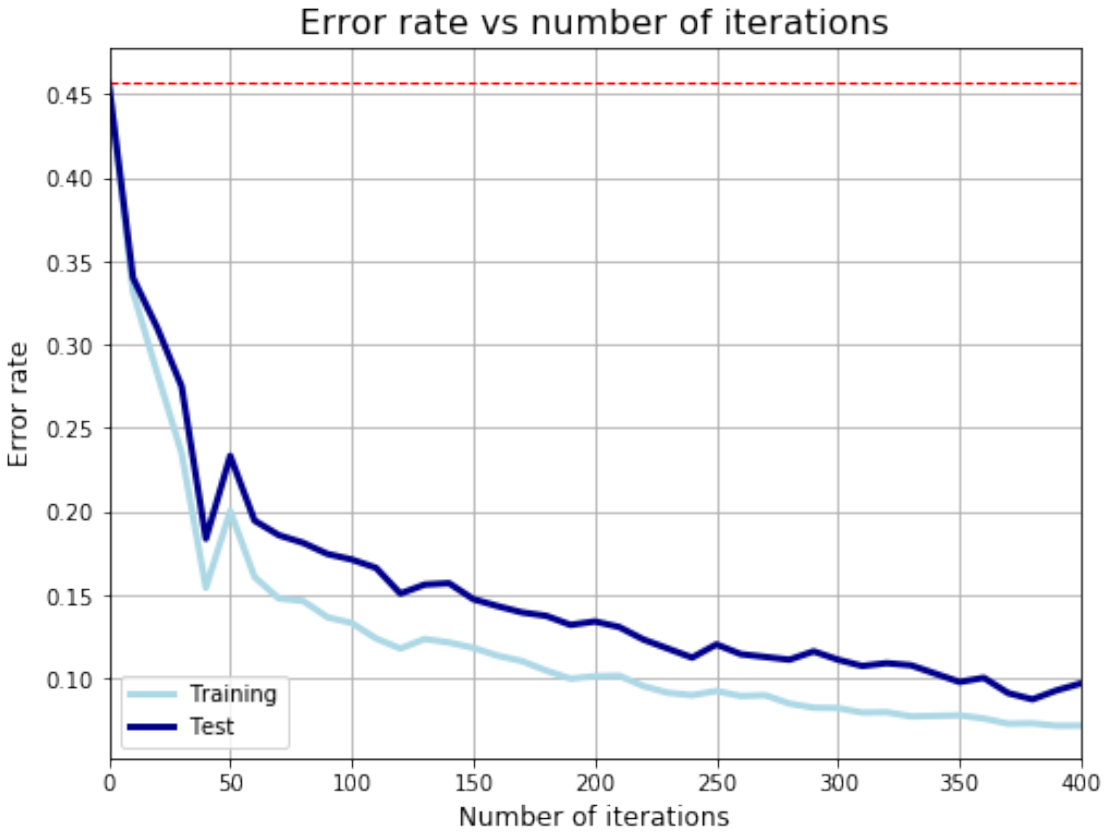
```
Out[8]: range(10, 410, 10)
```

```
In [7]: ## Script adaptado de https://github.com/jaimeps/adaboost-implementation/blob/master/ada
        import pandas as pd
        import numpy as np
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.model_selection import train_test_split
        from sklearn.datasets import make_hastie_10_2
        import matplotlib.pyplot as plt

        """ HELPER FUNCTION: GET ERROR RATE ========================================="""
        def get_error_rate(pred, Y):
            return sum(pred != Y) / float(len(Y))

        """ HELPER FUNCTION: PRINT ERROR RATE ======================================="""
        def print_error_rate(err):
            print ('Error rate: Training: %.4f - Test: %.4f' % err)

        """ HELPER FUNCTION: GENERIC CLASSIFIER ====================================="""
        def generic_clf(Y_train, X_train, Y_test, X_test, clf):
            clf.fit(X_train,Y_train)
            pred_train = clf.predict(X_train)
```

```
    pred_test = clf.predict(X_test)
    return get_error_rate(pred_train, Y_train), \
           get_error_rate(pred_test, Y_test)

""" ADABOOST IMPLEMENTATION ================================================="""
def adaboost_clf(Y_train, X_train, Y_test, X_test, M, clf):
    n_train, n_test = len(X_train), len(X_test)
    # Initialize weights
    w = np.ones(n_train) / n_train
    pred_train, pred_test = [np.zeros(n_train), np.zeros(n_test)]

    for i in range(M):
        # Fit a classifier with the specific weights
        clf.fit(X_train, Y_train, sample_weight = w)
        pred_train_i = clf.predict(X_train)
        pred_test_i = clf.predict(X_test)
        # Indicator function
        miss = [int(x) for x in (pred_train_i != Y_train)]
        # Equivalent with 1/-1 to update weights
        miss2 = [x if x==1 else -1 for x in miss]
        # Error
        err_m = np.dot(w,miss) / sum(w)
        # Alpha
        alpha_m = 0.5 * np.log( (1 - err_m) / float(err_m))
        # New weights
        w = np.multiply(w, np.exp([float(x) * alpha_m for x in miss2]))
        # Add to prediction
        pred_train = [sum(x) for x in zip(pred_train,
                                          [x * alpha_m for x in pred_train_i])]
        pred_test = [sum(x) for x in zip(pred_test,
                                         [x * alpha_m for x in pred_test_i])]

    pred_train, pred_test = np.sign(pred_train), np.sign(pred_test)
    # Return error rate in train and test set
    return get_error_rate(pred_train, Y_train), \
           get_error_rate(pred_test, Y_test)

""" PLOT FUNCTION ==========================================================="""
def plot_error_rate(er_train, er_test):
    df_error = pd.DataFrame([er_train, er_test]).T
    df_error.columns = ['Training', 'Test']
    plot1 = df_error.plot(linewidth = 3, figsize = (8,6),
            color = ['lightblue', 'darkblue'], grid = True)
    plot1.set_xlabel('Number of iterations', fontsize = 12)
    plot1.set_xticklabels(range(0,450,50))
    plot1.set_ylabel('Error rate', fontsize = 12)
    plot1.set_title('Error rate vs number of iterations', fontsize = 16)
    plt.axhline(y=er_test[0], linewidth=1, color = 'red', ls = 'dashed')
```

```
    plt.show()

""" MAIN SCRIPT ============================================================="""
if __name__ == '__main__':

    # Read data
    x, y = make_hastie_10_2()
    df = pd.DataFrame(x)
    df['Y'] = y

    # Split into training and test set
    train, test = train_test_split(df, test_size = 0.2)
    X_train, Y_train = train.iloc[:,:-1], train.iloc[:,-1]
    X_test, Y_test = test.iloc[:,:-1], test.iloc[:,-1]

    print('passei 1')

    # Fit a simple decision tree first
    clf_tree = DecisionTreeClassifier(max_depth = 1, random_state = 1)
    er_tree = generic_clf(Y_train, X_train, Y_test, X_test, clf_tree)

    print('passei 2')

    # Fit Adaboost classifier using a decision tree as base estimator
    # Test with different number of iterations
    er_train, er_test = [er_tree[0]], [er_tree[1]]

    x_range = range(10, 410, 10)
    for i in x_range:
        er_i = adaboost_clf(Y_train, X_train, Y_test, X_test, i, clf_tree)
        er_train.append(er_i[0])
        er_test.append(er_i[1])

    #er_i = adaboost_clf(Y_train, X_train, Y_test, X_test, 50, clf_tree)
    #er_train.append(er_i[0])
    #er_test.append(er_i[1])

    print('passei 3')

    # Compare error rate vs number of iterations
    plot_error_rate(er_train, er_test)
```

```
passei 1
passei 2
passei 3
```


Error rate vs number of iterations
Error rate
Number of iterations
Training
Test
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
50
100
150
200
250
300
350
400

In [ ]: