AdaBoost Demo

AdaBoost (Adaptive Boosting) is a Meta Machine Learning algorithm. It defines a procedure to learn and combine a set of `weak classifiers` to form a strong classifier. The AdaBoost learning procedure consist of `M` iterations. In each iteration a weak classifier is learned using an appropriate Machine Learning algorithm. In each iteration the training samples that are classified correctly obtain a smaller weight, whereas the weight of all trainingsamples that are not correctly classified remain the same. The effect is that in the training procedure of the next iteration, the training samples that have been classified correctly, have less influence than the not correctly classified training samples. Thus the model in the next iteration is more adapted to the previously misclassified samples. The final classifier is a linear combination of all `M` weak classifiers. The coefficients in this linear combination are also determined in the iterations: Weak classifiers with a small error rate obtain a larger weight, than weak classifiers with a high error rate.

AdaBoost is applied e.g. in the Viola-Jones face detector. Here, in each iteration a weak classifier is learned that just applies a threshold on a single feature. The learned weak classifier is the single feature with a corresponding threshold, that best discriminates positive (face subwindows) from negative samples (non-face subwindows). Note that this optimal single feature and the corresponding threshold depends on the varying weights of the training samples. AdaBoost in Viola-Jones face detection thus not only learns a classifier but also the most informative features out of a very large set of features.

Python Code of Demo

In the following example AdaBoost is applied to a set of 10 training samples, which are separated into 2 classes. Each sample is described by 3 features. The training samples are defined in matrix `X`, the corresponding class labels are defined in the vector `C`. Inititally all training samples obtain the same weight `w=1/10`. The weights are stored in variable `W`. The number of iterations (i.e. the number of weak classifiers) is defined to be `M=5`.:

```
import numpy as np
from sklearn import linear_model
from sklearn import metrics

X=np.array([[3,5,2],
                        [5,5,5],
                        [3,0,2],
                        [3,4,1],
                        [5,0,2],
                        [2,1,3],
                        [4,3,1],
                        [2,5,1],
                        [0,2,1],
                        [2,1,5]])
C=np.array([0,0,0,0,0,1,1,1,1,1]) # class labels
print "Example data - 3 features, last column is label"
print np.hstack((X,np.transpose(np.atleast_2d(C))))
```

```
W=1.0/len(C)*np.ones(C.shape[0])
print "Initial weights"
print W
M=5 #Number of iterations
```

In each iteration of the following loop a weak classifier is learned. This is realized by determining for each of the three features the threshold, that best separates the negative (class label =0) from the positive (class label = 1) training samples. The learning algorithm applied for this is the `LogisticRegression` module form `scikit-learn`. Once the best feature and corresponding threshold of the current iteration is determined, the weighted error `J` on the training data is applied to compute the variables `beta` and `alpha`. Variable `beta` is applied to adapt the weights of the training samples for the next iteration. Variable `alpha` is the coefficient that determines how strong the currently learned weak classifier contributes to the final classifier.:

```
reg=linear_model.LogisticRegression()
bestClassifier=np.zeros((M,3)) # Array to store 1.)best feature, 2.)threshold, 3.)alpha of
for F in range(M):
        print "\n\n"
        print "#"*30
        print "Start of Iteration %d"%F
        IndicatorList=[]
        betaList=[]
        thresholdList=[]
        bestVal=10
        bestFeat=10
        for i in range(X.shape[1]):
                print "\nClassification with respect to feature %d"%i
                indata= np.transpose(np.atleast_2d(X[:,i]))
                reg.fit(indata,C)
                threshold=-1*reg.intercept_/reg.coef_
                print "Threshold = ",threshold
                CE=reg.predict(indata)
                print "target :     ",C
                print "predicted :  ",CE
                Indicator = np.abs(CE-C)
                errors=np.dot(Indicator,W)
                print "J =",errors
                if errors < bestVal:
                        bestVal=errors
                        bestFeat=i
                J=errors
                beta=1.0*J/(1-J)
                IndicatorList.append(Indicator)
                betaList.append(beta)
                thresholdList.append(threshold)
                print '-'*30

        I=IndicatorList[bestFeat]
        beta= betaList[bestFeat]
        threshold=thresholdList[bestFeat]
        alpha=np.log(1.0/beta)
        print "BestFeature:          ",bestFeat
        print "Threshold =           ",threshold
        print "Best Beta =           ",beta
        print "Corresponding alpha = ",alpha
        bestClassifier[F,:]=np.array([bestFeat,threshold,alpha])
        for r in range(C.shape[0]):
                W[r]=W[r]*np.power(beta,1-I[r])
        Wnorm=np.sum(W)
        W=W/Wnorm
        print "New weights for training samples: "
        print W
        print "#"*30
        print"\n\n"
```

At this stage all `M=5` weak classifiers are learned. The parameters `feature index`, `threshold` and `alpha` of each weak classifier are stored in the numpy-array `bestClassifiers`. These parameters uniquelly define the final strong classifier. The strong classifier is finally applied to three different input vectors, defined in the array `newdata`. The decision criteria is: If the linear combination of the strong classifiers is larger than half of the sum over all 5 `alpha`-values, then the predicted class label is 0, otherwise it is 1.:

```
print "*"*50
print "Best Classifiers in each iteration:"
print "Feature|Threshold|Alpha"
print bestClassifier
halfSumAlpha=0.5*np.sum(bestClassifier[:,2])
print "Threshold of combined Classifier  :  ",halfSumAlpha
newdata=np.array([[1,5,1],[2,10,2],[5,3,1]])
for u in range(newdata.shape[0]):
        data=newdata[u,:]
        sum=0
        for v in range(M):
                if newdata[u,bestClassifier[v,0]] > bestClassifier[v,1]:
                        sum=sum+bestClassifier[v,2]
        print "Data : ",newdata[u,:]
        if sum > halfSumAlpha:
                print "Classified as Class 0"
        else:
                print "Classified as Class 1"
```

The output generated by this program is:

```
Example data - 3 features, last column is label
[[3 5 2 0]
 [5 5 5 0]
 [3 0 2 0]
 [3 4 1 0]
 [5 0 2 0]
 [2 1 3 1]
 [4 3 1 1]
 [2 5 1 1]
 [0 2 1 1]
 [2 1 5 1]]
Initial weights
[ 0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1]


##############################
```

```
Start of Iteration 0

Classification with respect to feature 0
Threshold =  [[ 2.02868822]]
target :      [0 0 0 0 0 1 1 1 1 1]
predicted :   [0 0 0 0 0 1 0 1 1 1]
J = 0.1
------------------------------

Classification with respect to feature 1
Threshold =  [[ 1.85487592]]
target :      [0 0 0 0 0 1 1 1 1 1]
predicted :   [0 0 1 0 1 1 0 0 0 1]
J = 0.5
------------------------------

Classification with respect to feature 2
Threshold =  [[ 1.64177335]]
target :      [0 0 0 0 0 1 1 1 1 1]
predicted :   [0 0 0 1 0 0 1 1 1 0]
J = 0.3
------------------------------
BestFeature:           0
Threshold =            [[ 2.02868822]]
Best Beta =            0.111111111111
Corresponding alpha =  2.19722457734
New weights for training samples:
[ 0.05555556  0.05555556  0.05555556  0.05555556  0.05555556  0.05555556
  0.5         0.05555556  0.05555556  0.05555556]
##############################





##############################
Start of Iteration 1

Classification with respect to feature 0
Threshold =  [[ 2.02868822]]
target :      [0 0 0 0 0 1 1 1 1 1]
predicted :   [0 0 0 0 0 1 0 1 1 1]
J = 0.5
------------------------------

Classification with respect to feature 1
Threshold =  [[ 1.85487592]]
target :      [0 0 0 0 0 1 1 1 1 1]
predicted :   [0 0 1 0 1 1 0 0 0 1]
J = 0.722222222222
------------------------------

Classification with respect to feature 2
Threshold =  [[ 1.64177335]]
target :      [0 0 0 0 0 1 1 1 1 1]
predicted :   [0 0 0 1 0 0 1 1 1 0]
J = 0.166666666667
------------------------------
BestFeature:           2
Threshold =            [[ 1.64177335]]
```

```
Best Beta =            0.2
Corresponding alpha =  1.60943791243
New weights for training samples:
[ 0.03333333  0.03333333  0.03333333  0.16666667  0.03333333  0.16666667
  0.3         0.03333333  0.03333333  0.16666667]
##############################






##############################
Start of Iteration 2

Classification with respect to feature 0
Threshold =  [[ 2.02868822]]
target :      [0 0 0 0 0 1 1 1 1 1]
predicted :   [0 0 0 0 0 1 0 1 1 1]
J = 0.3
------------------------------

Classification with respect to feature 1
Threshold =  [[ 1.85487592]]
target :      [0 0 0 0 0 1 1 1 1 1]
predicted :   [0 0 1 0 1 1 0 0 0 1]
J = 0.433333333333
------------------------------

Classification with respect to feature 2
Threshold =  [[ 1.64177335]]
target :      [0 0 0 0 0 1 1 1 1 1]
predicted :   [0 0 0 1 0 0 1 1 1 0]
J = 0.5
------------------------------
BestFeature:           0
Threshold =            [[ 2.02868822]]
Best Beta =            0.428571428571
Corresponding alpha =  0.847297860387
New weights for training samples:
[ 0.02380952  0.02380952  0.02380952  0.11904762  0.02380952  0.11904762
  0.5         0.02380952  0.02380952  0.11904762]
##############################






##############################
Start of Iteration 3

Classification with respect to feature 0
Threshold =  [[ 2.02868822]]
target :      [0 0 0 0 0 1 1 1 1 1]
predicted :   [0 0 0 0 0 1 0 1 1 1]
J = 0.5
------------------------------

Classification with respect to feature 1
Threshold =  [[ 1.85487592]]
```

```
target :      [0 0 0 0 0 1 1 1 1 1]
predicted :   [0 0 1 0 1 1 0 0 0 1]
J = 0.595238095238
------------------------------

Classification with respect to feature 2
Threshold =  [[ 1.64177335]]
target :      [0 0 0 0 0 1 1 1 1 1]
predicted :   [0 0 0 1 0 0 1 1 1 0]
J = 0.357142857143
------------------------------
BestFeature:           2
Threshold =            [[ 1.64177335]]
Best Beta =            0.555555555556
Corresponding alpha =  0.587786664902
New weights for training samples:
[ 0.01851852  0.01851852  0.01851852  0.16666667  0.01851852  0.16666667
  0.38888889  0.01851852  0.01851852  0.16666667]
##############################




##############################
Start of Iteration 4

Classification with respect to feature 0
Threshold =  [[ 2.02868822]]
target :      [0 0 0 0 0 1 1 1 1 1]
predicted :   [0 0 0 0 0 1 0 1 1 1]
J = 0.388888888889
------------------------------

Classification with respect to feature 1
Threshold =  [[ 1.85487592]]
target :      [0 0 0 0 0 1 1 1 1 1]
predicted :   [0 0 1 0 1 1 0 0 0 1]
J = 0.462962962963
------------------------------

Classification with respect to feature 2
Threshold =  [[ 1.64177335]]
target :      [0 0 0 0 0 1 1 1 1 1]
predicted :   [0 0 0 1 0 0 1 1 1 0]
J = 0.5
------------------------------
BestFeature:           0
Threshold =            [[ 2.02868822]]
Best Beta =            0.636363636364
Corresponding alpha =  0.451985123743
New weights for training samples:
[ 0.01515152  0.01515152  0.01515152  0.13636364  0.01515152  0.13636364
  0.5         0.01515152  0.01515152  0.13636364]
##############################


**************************************************
Best Classifiers in each iteration:
```

```
Feature|Threshold|Alpha
[[ 0.          2.02868822  2.19722458]
 [ 2.          1.64177335  1.60943791]
 [ 0.          2.02868822  0.84729786]
 [ 2.          1.64177335  0.58778666]
 [ 0.          2.02868822  0.45198512]]
Threshold of combined Classifier  :   2.8468660694
Data :  [1 5 1]
Classified as Class 1
Data :  [ 2 10  2]
Classified as Class 1
Data :  [5 3 1]
Classified as Class 0
```