9. AdaBoost

March 28, 2022

[1]:
```
import numpy as np
```

[2]:
```
def stumpClassify(dataMatrix,dimen,threshVal,threshIneq):
    retArray = np.ones((np.shape(dataMatrix)[0],1))
    if threshIneq == 'lt':
        retArray[dataMatrix[:,dimen] <= threshVal] = -1.0
    else:
        retArray[dataMatrix[:,dimen] > threshVal] = -1.0
    return retArray
```

[3]:
```
def buildStump(dataArr,classLabels,D):    # D is the weight vector
    dataMatrix = np.mat(dataArr)
    labelMat = np.mat(classLabels).T
    m,n = np.shape(dataMatrix)
    numSteps = 10.0
    bestStump = {}  # empty dictionary
    bestClasEst = np.mat(np.zeros((m,1)))
    minError = np.inf
    for i in range(n):                    # for each feature
        rangeMin = dataMatrix[:,i].min()
        rangeMax = dataMatrix[:,i].max();
        stepSize = (rangeMax-rangeMin)/numSteps
        # loop over values with changing thresholds in each iteration
        for j in range(-1,int(numSteps)+1):
            for inequal in ['lt', 'gt']:
                threshVal = (rangeMin + float(j) * stepSize)
                # create stump for current threshold
                predictedVals = stumpClassify(dataMatrix,i,threshVal,inequal)
                errArr = np.mat(np.ones((m,1)))
                errArr[predictedVals == labelMat] = 0   # find error
                weightedError = D.T*errArr              # find weighted error
                # check with error of previous best stump
                if weightedError < minError:
                    minError = weightedError
                    bestClasEst = predictedVals.copy()
                    bestStump['dim'] = i
                    bestStump['thresh'] = threshVal
```

```
                    bestStump['ineq'] = inequal
    return bestStump,minError,bestClasEst
```

[4]:
```
def adaBoostTrainDS(dataArr,classLabels,numIt=40):
    weakClassArr = []
    m = np.shape(dataArr)[0]
    D = np.mat(np.ones((m,1))/m)      # initial weight matrix
    aggClassEst = np.mat(np.zeros((m,1)))

    for i in range(numIt):           # numIt is number of iterations
        # find best stump for current dataset
        bestStump,error,classEst = buildStump(dataArr,classLabels,D)
        print("D:",D.T)             # sample weights of current iteration

        # find performance of best stump
        alpha = float(0.5*np.log((1.0-error)/max(error,1e-16)))
        # alpha value is added to bestStump dictionary
        bestStump['alpha'] = alpha
        # dictionary is appended to the list
        weakClassArr.append(bestStump)
        print("classEst: ",classEst.T)  # predicted classes by the stump

        # vector of exp; returns -ve value for correct predictions
        expon = np.multiply(-1*alpha*np.mat(classLabels).T,classEst)
        D = np.multiply(D,np.exp(expon))        # calculate new weights
        D = D/D.sum()                           # normalize weights

        aggClassEst += alpha*classEst
        print("aggClassEst: ",aggClassEst.T)
        aggErrors = np.multiply(np.sign(aggClassEst) != np.mat(classLabels).T, \
                                np.ones((m,1)))
        errorRate = aggErrors.sum()/m
        print("total error: ",errorRate,"\n")
        if errorRate == 0.0: break
    return weakClassArr
```

[5]:
```
def adaClassify(datToClass,classifierArr):
    dataMatrix = np.mat(datToClass)
    m = np.shape(dataMatrix)[0]
    aggClassEst = np.mat(np.zeros((m,1)))
    for i in range(len(classifierArr)):
        classEst = stumpClassify(dataMatrix,classifierArr[i]['dim'], \
                classifierArr[i]['thresh'], classifierArr[i]['ineq'])
        aggClassEst += classifierArr[i]['alpha']*classEst
        print(aggClassEst)
    return np.sign(aggClassEst)
```

[6]:
```
datMat = [[ 1. , 2.1],[ 2. , 1.1],[ 1.3, 1. ],[ 1. , 1. ],[ 2. , 1. ]]
classLabels = [1.0, 1.0, -1.0, -1.0, 1.0]

classifierArr = adaBoostTrainDS(datMat,classLabels,30)
print(classifierArr)
```

```
D: [[0.2 0.2 0.2 0.2 0.2]]
classEst:  [[-1.  1. -1. -1.  1.]]
aggClassEst:  [[-0.69314718  0.69314718 -0.69314718 -0.69314718  0.69314718]]
total error:  0.2

D: [[0.5   0.125 0.125 0.125 0.125]]
classEst:  [[ 1.  1. -1. -1. -1.]]
aggClassEst:  [[ 0.27980789  1.66610226 -1.66610226 -1.66610226 -0.27980789]]
total error:  0.2

D: [[0.28571429 0.07142857 0.07142857 0.07142857 0.5       ]]
classEst:  [[1. 1. 1. 1. 1.]]
aggClassEst:  [[ 1.17568763  2.56198199 -0.77022252 -0.77022252  0.61607184]]
total error:  0.0

[{'dim': 0, 'thresh': 1.3, 'ineq': 'lt', 'alpha': 0.6931471805599453}, {'dim':
1, 'thresh': 1.0, 'ineq': 'lt', 'alpha': 0.9729550745276565}, {'dim': 0,
'thresh': 0.9, 'ineq': 'lt', 'alpha': 0.8958797346140273}]
```

[7]:
```
adaClassify([0, 0],classifierArr)
```

```
[[-0.69314718]]
[[-1.66610226]]
[[-2.56198199]]
```

[7]:
```
matrix([[-1.]])
```

[8]:
```
adaClassify([[5, 5],[0,0]],classifierArr)
```

```
[[ 0.69314718]
 [-0.69314718]]
[[ 1.66610226]
 [-1.66610226]]
[[ 2.56198199]
 [-2.56198199]]
```

[8]:
```
matrix([[ 1.],
        [-1.]])
```