Adaboost算法实践

AdaBoost算法

> 算法描述:
> *输入:*
> 训练数据集$T=\{(x_1,y_1)(x_2,y_2),\cdots,(x_N,y_N)\}, y_i\in\{-1,1\}$
> *输出*最终分类器$G(x)=\{G_1,G_2,\cdots,G_n\}$的集成
>
> 1. 初始化权值分布$D_1=(w_{11},\cdots,w_{1N}), w_{1i}=\frac{1}{N}$
> 2. 对于$m=1,2,\cdots,m$
>   a. 使用**具有权值分布D_m的训练数据集**学习, 得到基本分类器
>
> $$G_m(x):x\to\{-1,+1\}$$
>
>   b. 计算$G_m(x)$在训练数据集上的分类加权误差率:
>
> $$e_m=P(G_m(x)\neq y_i)=\sum_{i=1}^{N}w_{mi}I(G_m(x_i)\neq y_i)$$
>
>   c. 检查e_m是否大于0.5(是否优于随机预测)如果没有随机预测的效果好, 就结束
>   d. 计算$G_m(x)$在最后集成模型中的系数(错误率越高占比越小):
>
> $$\alpha_m=\frac{1}{2}\ln\frac{1-e_m}{e_m}$$
>
>   e. 更新训练数据集的权值分布(错误的数据提高权值, 正确的数据降低权值)
>
> $$w_{m+1,i}=\begin{cases}\frac{1}{Z_m}w_{m,i}e^{-\alpha_m}, G_m(x_i)=y_i\\ \frac{1}{Z_m}w_{m,i}e^{\alpha_m}, G_m(x_i)\neq y_i\end{cases}$$
>
> 3. 得到集成模型
>
> $$G(x)=sign[\sum_{m}^{M}\alpha_mG_m(x)]$$
>
> 最后的结果$G(x)$的符号断定, 因为α之和未必为1

代码

```
from numpy import *
def StumpClassify(DataMatix, Dimen, ThreshVal, ThreshIneq): #按照阈值和数据维度进行数据分类
    RetArray = ones((shape(DataMatix)[0],1))    #第一项时数据个数
    if ThreshIneq == "lt":
        RetArray[DataMatix[:, Dimen] <= ThreshVal] = -1.0   #当为"lt"是将Dimen列小于Val的索引位置置为-1.0
    else:
        RetArray[DataMatix[:, Dimen] > ThreshVal] = -1.0  # 当为"lt"是将Dimen列小于Val的索引位置置为-1.0
    return RetArray

def BuildStump(DataArr, ClassLabels, D):    #构造单层决策树
    DataMatrix = mat(DataArr)
    LabelMat = mat(ClassLabels).T
    m, n = shape(DataMatrix)
    NumSteps = 10.0; BestStump = {}; BestClassEst = mat(zeros((m, 1)))  #n是单数据属性个数， m是数据个数, Be
    MinError = inf
    for i in range(n):  #对每一个数据维度尝试分类
        RangeMin = DataMatrix[:, i].min()
        RangeMax = DataMatrix[:, i].max()
        StepSize = (RangeMax - RangeMin)/NumSteps   #计算每一次调整阈值的步长
        for j in range(-1, int(NumSteps) + 1):  #调整十次阈值
            for Inequal in ["lt", "gt"]:    #调整预测的正反向
                ThreshVal = (RangeMin + float(j)*StepSize)
                PredictVals = StumpClassify(DataMatrix, i, ThreshVal, Inequal)
                ErrArr = mat(ones((m, 1), dtype=int))
                ErrArr[PredictVals == LabelMat] = 0 #将正确项标注为0
                WeightedError = D.T*ErrArr  #计算加权错误总数
                #print("The WeightedError = {} in condition that ThreshVal = {}, Inequal = {} of the {} d
                if WeightedError < MinError:    #如果最小错误更新字典
                    MinError = WeightedError
                    BestClassEst = PredictVals.copy()
                    BestStump["Dimen"] = i
                    BestStump["Thresh"] = ThreshVal
                    BestStump["Ineq"] = Inequal
    return BestStump, MinError, BestClassEst

def AdaBoostTrain(DataArr, ClassLabels, NumIt = 40):
    WeakClassArr = []
    m = shape(DataArr)[0]
    D = mat(ones(shape = (m, 1))/m)
    AggClassEst = mat(zeros((m, 1)))
    for i in range(NumIt):
        BestStump, Error, ClassEst = BuildStump(DataArr, ClassLabels, D)
        print("D:", D.T, "\n")
        Alpha = float(0.5*log((1.0-Error)/max(Error, 1e-16)))
        BestStump["alpha"] = Alpha
        WeakClassArr.append(BestStump)  #生成当前层的决策树，将当前层的决策树加入到字典中
        print("ClassEst:", ClassEst, "\n")
        Expon = multiply(-1*Alpha*mat(ClassLabels).T, ClassEst) #通过预测的准确与否来确定更新系数
        print(ClassLabels)
        print(mat(ClassLabels))
        print(ClassEst)
        D = multiply(D, exp(Expon)) #更新D矩阵
        D = D/D.sum()
        AggClassEst += Alpha*ClassEst   #每一次生成新的学习器后，都加权加入总的决策
        print("AggClassEnt", AggClassEst, "\n")
        AggErrors = multiply(sign(AggClassEst)!=mat(ClassLabels).T, ones((m, 1))) #对应元素相乘
        ErrorRate = AggErrors.sum()/m
        print("Total errorrate", ErrorRate, "\n")
        if(ErrorRate == 0.0):
            break
```

```
    return WeakClassArr

def AdaClassfy(DataToClass, ClassifierArr): #这里Classifier是存储学习器的字典
    DataMat = mat(DataToClass)
    m = shape(DataMat)[0]
    AggClassEst = mat(zeros((m, 1)))
    for i in range(len(ClassifierArr)):
        ClassEst = StumpClassify(DataMat, ClassifierArr[i]["Dimen"],
                                 ClassifierArr[i]["Thresh"], ClassifierArr[i]["Ineq"])
        AggClassEst += ClassifierArr[i]["alpha"]*ClassEst
        print(AggClassEst)
    return sign(AggClassEst)

def LoadDataSet(filename):
    DataMat = []
    LabelMat = []
    fr = open(filename)
    NumOneLine = len(fr.readline().split('\t'))
    for line in fr.readlines():
        LineInfo = []
        Curline = line.strip().split('\t')
        for i in range(NumOneLine - 1):
            LineInfo.append(float(Curline[i]))
        DataMat.append(LineInfo)
        LabelMat.append(float(Curline[-1]))
    return DataMat, LabelMat

if __name__ == "__main__":
    DataMat, LabelMat = LoadDataSet("HorseTraining.txt")
    DataToClass, RealRes = LoadDataSet("HorseTest.txt")
    AdaBoostClassifier = AdaBoostTrain(DataMat, LabelMat, 50)
    PredictMat = list(AdaClassfy(DataToClass, AdaBoostClassifier))
    Size = len(RealRes)
    count = 0
    for i in range(Size):
        if(RealRes[i] != PredictMat[i]):
            count += 1
    print("The error rate is", count, float(count)/Size)
```

每一个基学习器是一个简化的单层决策树, 决策树的构建没有按照计算信息熵等方法来进行, 而是采用试探的办法找到最佳属性和阈值

StumpClassify函数对于给定的阈值和阈值两侧的划分关系和属性的序号对数据集进行预测

BuildStump函数对于给定的数据集和标签按照权重计算损失构建最佳决策树

AdaBoostTrain函数按照算法训练模型, 得到小于NumIt个学习器

AdaClassfy函数根据训练好的模型对数据集进行预测

实验效果:

```
Total errorrate 0.28523489932885904
Total errorrate 0.28523489932885904
Total errorrate 0.2483221476510067
Total errorrate 0.2483221476510067
Total errorrate 0.2483221476510067
Total errorrate 0.24161073825503357
Total errorrate 0.24161073825503357
Total errorrate 0.2214765100671141
Total errorrate 0.2483221476510067
Total errorrate 0.2214765100671141
Total errorrate 0.23154362416107382
Total errorrate 0.22483221476510068
Total errorrate 0.21476510067114093
Total errorrate 0.2214765100671141
Total errorrate 0.22818791946308725
Total errorrate 0.22818791946308725
Total errorrate 0.22818791946308725
Total errorrate 0.21476510067114093
Total errorrate 0.2181208053691275
Total errorrate 0.2080536912751678
Total errorrate 0.22483221476510068
Total errorrate 0.20469798657718122
Total errorrate 0.2181208053691275
Total errorrate 0.22483221476510068
Total errorrate 0.23154362416107382
Total errorrate 0.2181208053691275
Total errorrate 0.22483221476510068
Total errorrate 0.21476510067114093
Total errorrate 0.22818791946308725
Total errorrate 0.21140939597315436
Total errorrate 0.2181208053691275
Total errorrate 0.20134228187919462
Total errorrate 0.2181208053691275
Total errorrate 0.20134228187919462
Total errorrate 0.20469798657718122
Total errorrate 0.20134228187919462
Total errorrate 0.21476510067114093
Total errorrate 0.19798657718120805
Total errorrate 0.1912751677852349
Total errorrate 0.19798657718120805
Total errorrate 0.20134228187919462
Total errorrate 0.20134228187919462
Total errorrate 0.21140939597315436
Total errorrate 0.1912751677852349
Total errorrate 0.20134228187919462
Total errorrate 0.18791946308724833
Total errorrate 0.20134228187919462
Total errorrate 0.18120805369127516
Total errorrate 0.20134228187919462
Total errorrate 0.18791946308724833

The error rate is 13 0.19696969696969696
```

使用50组集成用加权平均的办法, 将单个19-28的错误率降低到19, 提升了泛化性能.

心得体会

1. 模型训练过程中的学习器参数保留通过字典来保存是很好的选择

2. 尽量把循环交给numpy来做，比如
    - 循环相乘可以使用矩阵Multiply方法和*运算符
    - 循环找符合条件的值可以通过Array/mat[Condition].sum()/max()来实现