大数据机器学习第六次理论作业

2026 年 1 月 22 日

1　某公司招聘职员考察身体、业务能力、发展潜力这 3 项。身体分为合格 1、不合格 0 两级，业务能力和发展潜力分为上 1、中 2、下 3 三级。分类为合格 1 不合格-1 两级。已知 10 个人的数据，如下表所示。假设弱分类器为决策树桩。试用 AdaBoost 算法学习一个强分类器。

表 8.5　应聘人员情况数据表

	1	2	3	4	5	6	7	8	9	10
身体	0	0	1	1	1	0	1	1	1	0
业务能力	1	3	2	1	2	1	1	1	3	2
发展潜力	3	1	2	3	3	2	2	1	1	1
分类	−1	−1	−1	−1	−1	−1	1	1	−1	−1

1. 相关知识点

1)． 弱分类器：决策树桩

只要能够找到比随机分类效果略好的分类器，就找到了一个弱分类器。我们可以首先构建一个弱分类器决策树桩进行最基本的二分类。

决策树桩（Decision Stump）也叫做单层决策树，主要通过给定阈值（threshold）进行分类，$x < v$ 或 $x > v$，可以看成一个根节点直接连接两个叶节点的简单决策树。

2)． AdaBoost 算法学习强分类器

我们用:

- w_{mi} 表示第 i 个样本第 m 轮分类的权重
- G_m 表示第 m 个分类器
- e_m 表示第 m 个分类器的错误率
- α_m 表示前 m 个分类器的权重
- f_m 表示前 m 个分类器的线性组合。
- G_x 表示最终的分类器

AdaBoost 算法的输入是：

$$T = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\},$$
$$x_i \in \mathcal{X} \subseteq \mathbf{R}^n, y_i \in \mathcal{Y} = \{-1, +1\}$$

输出是：最终的分类器 G_x。

算法的步骤是：

- 初始化训练数据的起始的权值分布，每个数据的权值：$w_{1i} = \frac{1}{N}, \quad i = 1, 2, \cdots, N$
- 对 m 个弱分类器:
  - 得到在权值 D_m 下的训练数据集，得到弱分类器 $G_m(x)$
  - 计算 G_m 的训练误差 $e_m = \sum_{i=1}^{N} P(G_m(x_i) \neq y_i) = \sum_{i=1}^{N} w_{mi} I(G_m(x_i) \neq y_i)$
  - 计算 G_m 的系数 $\alpha_m = \frac{1}{2} \log \frac{1-e_m}{e_m}$
  - 更新训练数据集样本的权值 $w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)), \quad i = 1, 2, \cdots, N,$ 其中 $Z_m = \sum_{i=1}^{N} w_{mi} \exp(-\alpha_m y_i G_m(x_i))$
- 构建规范分类器的线性组合,$f(x) = \sum_{m=1}^{M} \alpha_m G_m(x)$, 得到最终的分类器:$G(x) = sign(f(x))$

2. 代码实现

1). 弱分类器：决策树桩

算法思路：

- 设最小错误率
  - 对数据集的每个特征
    - 对数据集的每个步长
    - 对每个不等号, 建立一颗决策树并使用加权数据集进行测试，如果当前决策树错误率比最小错误率小，更新当前决策树为最佳决策树
- 返回最佳决策树桩

代码实现单层决策树桩 (参考 PPT)：

```
import numpy as np
x = np.array([[0,1,3],
              [0,3,1],
              [1,2,2],
              [1,1,3],
              [1,2,3],
              [0,1,2],
              [1,1,2],
              [1,1,1],
              [1,3,1],
              [0,2,1]])
y = np.array([-1,-1,-1,-1,-1,-1,1,1,-1,-1])
dataMatrix = np.matrix(x)
labelMatrix = np.matrix([-1,-1,-1,-1,-1,-1,1,1,-1,-1]).T
def stumpClassify(dataMatrix,dim,threshold,threshInequal):
    # 分类函数，通过阈值threshold对数据进行分类
    returnArray = np.ones((np.shape(dataMatrix)[0],1)) # 初始化
    # 小于：lt less than；大于gt  greater than
    if threshInequal == 'lt':
        returnArray[dataMatrix[:,dim]<=threshold] = -1.0
```

```
    else:
        returnArray[dataMatrix[:,dim]>threshold] = -1.0
    return returnArray
def buildStump(dataMatrix,classLabels,D):
    # 构建当前数据集的最佳决策树，遍历不同的阈值且计算分类误差， 找到分类误差最小的单层决策树
    # D:样本权重
    m,n = np.shape(dataMatrix)
    numSteps = 50.0 # 步长
    bestStumpInfo = {} # 最佳单层决策树的信息
    bestClassEst = np.mat(np.zeros((m,1))) # 最佳的分类结果
    minErr =  float('inf') # 最小的误差
    for i in  range(n):
        # 计算当前第i个特征的最大和最小值
        stepSize = (dataMatrix[:,i]. max() - dataMatrix[:,i]. min())/numSteps
        for j in  range(-1, int(numSteps)+1):
            for inequalType in ['lt','gt']:
                threshold = (dataMatrix[:,i]. min() + stepSize* float(j))
                predictVals = stumpClassify(dataMatrix,i,threshold,inequalType)
                # 计算误差，初始化误差矩阵
                errMatrix = np.matrix(np.ones((m,1)))
                errMatrix[classLabels== predictVals] = 0
                weightedErr = np.mean(D.T * errMatrix)
                # print(i,threshhold,inequal,weighted_err)
                print(f'SPLIT[{j}]:dim {i}, threshold {threshold:.2f}, inequalType {
                    inequalType}, weighted error {weightedErr:.3f}')
                if weightedErr < minErr:
                    minErr = weightedErr
                    bestClassEst = predictVals.copy()
                    bestStumpInfo['dim'] = i
                    bestStumpInfo['thresh'] = threshold
                    bestStumpInfo['ineq'] = inequalType
    return bestStumpInfo,minErr,bestClassEst
# 计算权重
D = np.matrix(np.ones((dataMatrix.shape[0],1))/dataMatrix.shape[0])
bestStumpInfo,minErr,bestClassEst = buildStump(dataMatrix,labelMatrix,D)
print(f'bestStumpInfo {bestStumpInfo}')
print(f'minErr {minErr}')
print(f'bestClassEst {bestClassEst.T}')
print(f'labelMatrix {labelMatrix.T}')
```

输出结果

```
SPLIT[48]:dim 2, threshold 2.92, inequalType gt, weighted error 0.500
SPLIT[49]:dim 2, threshold 2.96, inequalType lt, weighted error 0.500
SPLIT[49]:dim 2, threshold 2.96, inequalType gt, weighted error 0.500
SPLIT[50]:dim 2, threshold 3.00, inequalType lt, weighted error 0.200
SPLIT[50]:dim 2, threshold 3.00, inequalType gt, weighted error 0.800
bestStumpInfo {'dim': 0, 'thresh': -0.02, 'ineq': 'gt'}
minErr 0.2
bestClassEst [[-1. -1. -1. -1. -1. -1. -1. -1. -1. -1.]]
labelMatrix [[-1 -1 -1 -1 -1 -1  1  1 -1 -1]]
```

2). AdaBoost 算法学习强分类器

代码实现 AdaBoost 算法学习强分类器 (参考 PPT):

```
def adaBoostDSTrain(dataMatrix,classLabels,tol = 0.05,iter_max = 40):
    weakClassArray = []
    m = np.shape(dataMatrix)[0]
    D = np.matrix(np.ones((m,1))/m) # 初始化权重
    aggClassEst = np.mat(np.zeros((m,1))) # 初始化估计值
    for i in  range(iter_max):
        bestStumpi,erri,classEsti = buildStump(dataMatrix,classLabels,D)
        print(f'D {D.T}',end=',')
        alpha =  float(0.5*np.log((1.0-erri)/ max(erri,1e-15))) # 避免分母为0
        bestStumpi['alpha'] = alpha # alpha: 弱分类器权重
        weakClassArray.append(bestStumpi) #存储弱分类器
        print(f'classEsti {classEsti.T}',end=',')
        expon = np.multiply(-1*alpha*labelMatrix,classEsti) # 指数
        D = np.multiply(D,np.exp(expon))
        D = D/D. sum() # 更新样本权重，除以规范化因子
        aggClassEst += alpha*classEsti
        print(f'aggClassEsti {aggClassEst}',end=',')
        # 计算误差
        aggErr = np.multiply(np.sign(aggClassEst)!=labelMatrix,np.ones((m,1)))
        errRate = aggErr. sum()/m
        print(f'total error {errRate:.3f}')
        # 直到误差为0
        if errRate == 0:
            break
    return weakClassArray,aggClassEst

weakClassArray, aggClassEst = adaBoostDSTrain(dataMatrix, labelMatrix)
print(f'weakClassArray {weakClassArray}')
print(f'aggClassEst {aggClassEst}')
def adaBoostDSClassify(dataMatrix,classifierArray):
    m = dataMatrix.shape[0]
    aggClassEst = np.matrix(np.zeros((m,1)))# 初始化估计值
    # 遍历分类器
    for i in  range( len(classifierArray)):
        classEsti = stumpClassify(dataMatrix,classifierArray[i]['dim'],classifierArray[i]['
            thresh'],classifierArray[i]['ineq'])
        aggClassEst += classifierArray[i]['alpha']*classEsti
        print(f'aggClassEst {aggClassEst.T}')
    return np.sign(aggClassEst)
def cal_error(labelMatrix, predLabelMatrix):
    # 计算错误率
    err = 0
    n = labelMatrix.shape[0]
    for i in  range(n):
        if labelMatrix[i] != predLabelMatrix[i]:
            err += 1
    return err / n
aggClassEstAdaDS = adaBoostDSClassify(dataMatrix, weakClassArray)
print(f'aggClassEstAdaDS {aggClassEstAdaDS.T}')
```

```
print(f'lableMatrix {labelMatrix.T}')
print(f'error {cal_error(labelMatrix, aggClassEstAdaDS)}')
```

输出结果

```
aggClassEstAdaDS [[-1. - 1. - 1. - 1. - 1. - 1.  1.  1. - 1. - 1.]]
lableMatrix [[-1 - 1 - 1 - 1 - 1 - 1  1  1 - 1 - 1]]
error 0.0
```

结果表明： 以决策树桩作为弱分类器，使用 AdaBoost 算法学习的强分类器的错误率下降。

2　比较支持向量机、AdaBoost、逻辑斯谛回归模型的学习策略与算法。

模型 / 对比方式	支持向量机	AdaBoost	逻辑斯谛回归
特点	可以通过核函数解决非线性可分问题	将多个弱分类器按权重组合成强分类器	难以处理非线性分类问题
学习策略	最小间隔最大化	加法模型逐一学习基函数	极大对数似然函数
算法	SMO 算法求解凸二次规划对偶问题	前向分步算法（损失函数是指数损失函数）	梯度下降算法、牛顿法、拟牛顿法求解极大似然估计

1)． 相似性与不同点

相似性：

- 都可用于分类。
- 可以把 AdaBoost 弱分类器看成 SVM 的核函数，让 AdaBoost 模拟最小间隔最大化的过程。SVM 的目标是找到让每个支持向量距离超平面间隔不少于 1 的权值。AdaBoost 的目标是优化分类器的权值 α，通过对弱分类器根据权重组合来构造强分类器。

不同点： 高维数据降维后用不同模型分类的决策边界差异如图所示。

- AdaBoost 通过多个弱分类器的组合，让决策边界更加复杂，处理了难以分类的样本，让分类效果更好。
- 线性核 SVM 通过最小间隔最大化获得比逻辑斯谛回归的模型更优的效果。