LR_test

April 27, 2021

方理楠-201270001

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import threading
import random
from time import time
```

[ ]:
```
# sigmoid 函数
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

[ ]:
```
# 从文件中读取训练数据
def parse_train_data(filename):
    data = np.loadtxt(fname=filename, delimiter=',')
    dataMat = data[:, 0:-1]
    classLabels = data[:, -1]
    dataMat = np.insert(dataMat, 0, 1, axis=1)  # 按列在每行位置 0 插入 1，即偏置项，与 weights 对齐

    return dataMat, classLabels
```

[ ]:
```
# 从文件中读取测试数据
def parse_test_data(filename):
    data = np.loadtxt(fname=filename, delimiter=',')
    dataMat = data[:, :]
    dataMat = np.insert(dataMat, 0, 1, axis=1)  # 按列在每行位置 0 插入 1，即偏置项，与 weights 对齐

    return dataMat
```

[ ]:
```
# 损失函数
def loss_funtion(dataMat, classLabels, weights):
    m, n = np.shape(dataMat)
    loss = 0.0
    for i in range(m):
        logit = 0.0
        for j in range(n):
```

```
            logit += dataMat[i, j] * weights.T[0, j]  # 计算一个样本的
log-odds(logit)
        propability = sigmoid(logit)
        loss += classLabels[i, 0] * np.log(propability) + (
            1 - classLabels[i, 0]) * np.log(1 - propability)  # 损失函数
    return loss
```

[ ]:
```
# 梯度上升
def grad_Ascent(dataMatIn, classLabels):
    dataMatrix = np.mat(dataMatIn)  # (m,n)
    labelMat = np.mat(classLabels).T
    m, n = np.shape(dataMatrix)
    weights = np.ones((n, 1))  # 列向量
    alpha = 0.01
    maxstep = 10  # 迭代次数
    eps = 0.0001  # 损失小于一个阈值返回
    count = 0
    loss_array = []

    for i in range(maxstep):
        loss = loss_funtion(dataMatrix, labelMat, weights)

        h_theta_x = sigmoid(dataMatrix * weights)  # g(h(x))
        e = labelMat - h_theta_x  # y-g(h(x))
        new_weights = weights + alpha * dataMatrix.T * e  # 迭代
        new_loss = loss_funtion(dataMatrix, labelMat, new_weights)
        loss_array.append(new_loss)
        if abs(new_loss - loss) < eps:
            break
        else:
            weights = new_weights
            count += 1

    print("count is: ", count)
    print("loss is: ", loss)
    print("weights is: ", weights)

    return weights, loss_array
```

[ ]:
```
# 随机梯度上升
def stocGradAscent(dataMatrix, classLabels, numIter=100):
    m, n = np.shape(dataMatrix)  # 返回 dataMatrix 的大小。m 为行数,n 为列数。
    weights = np.ones(n)  # 参数初始化
    for j in range(numIter):
        dataIndex = list(range(m))  # 记录样本点索引
        for i in range(m):
            alpha = 4/(1.0+j+i)+0.01  # 降低 alpha 的大小，每次减小 1/(j+i)。
```

```
            randIndex = int(random.uniform(0, len(dataIndex)))  # 随机选取样本
            # 选择随机选取的一个样本，计算 h
            h = sigmoid(sum(dataMatrix[dataIndex[randIndex]]*weights))
            error = classLabels[dataIndex[randIndex]] - h  # 计算误差
            weights = weights + alpha * error * \
                dataMatrix[dataIndex[randIndex]]  # 更新回归系数
            del(dataIndex[randIndex])  # 删除已经使用的样本
    return weights  # 返回
```

[ ]:
```
# 分类函数 特征向量，回归系数
def classifyVector(inX, weights):
    prob = sigmoid(sum(inX*weights))
    if prob > 0.5:
        return 1.0
    else:
        return 0.0
```

[ ]:
```
def lr():
    data_train, labels_train = parse_train_data('train_data.txt')
    train_weights = stocGradAscent(data_train, labels_train, 100)

    data_test = parse_test_data('test_data.txt')
    labels_test = np.loadtxt('answer.txt')
    correct_count = 0
    m, n = np.shape(data_test)
    for i in range(m):
        if classifyVector(data_test[i], train_weights) == labels_test[i]:
            correct_count += 1
    correct_rate = (correct_count/m)*100
    print("测试集准确率为：%.2f%%" % correct_rate)
```

[ ]:
```
if __name__ == '__main__':
    start=time()
    lr()
    end=time()
    print('耗时：  %d 秒' %(end-start))
```

[ ]:
```

```