AdaBoost-KNN Project Report

Author: 章翰宇
ID: 3220104133

本次作业的 notebook 代码以及报告等相关内容均已经上传至我的仓库

INTRODUCTION

Method

Reference

我参考了两篇文献以及 CSDN 论坛

一篇是改进的 AdaBoost 集成学习方法研究[D].暨南大学,2021.DOI:10.27167/d.cnki.gjinu.2020.001350.

一篇是 Zhu, H. Zou, S. Rosset, T. Hastie, "Multi-class AdaBoost", 2009.

CSDN 内容：https://blog.csdn.net/weixin_43298886/article/details/110927084

experiment environment

我使用 Anaconda 中的环境，使用 jupyternotebook 进行编辑，使用了 **torch** 库，其版本为 2.2.1

```
[23]: print("PyTorch 版本:", torch.__version__)

PyTorch 版本: 2.2.1
```

Figure 1: 检查版本

CODES AND RESULTS

代码如下：

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import math

class KNN(nn.Module):
    def __init__(self, k):
        super(KNN, self).__init__()
        self.k = k

    def forward(self, x_train, y_train, x_test,w_train):
        dists = torch.cdist(x_test, x_train)  # 计算测试样本与训练样本之间的距离，输出是：第一行是 test 中的第一个和 train 中的每一个分别的欧氏距离，第二行是 test 中第二个和 train 中每一个的欧氏距离
        _, indices = torch.topk(dists, self.k, largest=False)  # 选择最近的 K 个样本的索引，存在 indices 中

        knn_labels = y_train[indices]  # 获取最近的 K 个样本的标签
        # print(y_train[indices])

        vote_pool=torch.zeros(x_test.size()[0],x_train.size()[0])  # 新建一个投票池，列数本来应该是类
```

```
的个数，但因为未知类的个数，不妨就取 train 几个样本，就有几个列，因为样本数一定大于类的个数，因此，类别数不能乱写，比如一共就 4 个样本，不能有样本是第 9 类的
        #print("vote init",vote_pool)

        #print("hey",indices.size()[0])
        for j in range(0, indices.size()[0]):
            # print("this is j",j)

            for i in indices[j]:  # 对于每一个 x_test 都有各自的投票池，这是给第 j 个池子投票
                #print("this",i)   #i 的含义就是哪几个是 k 个最近点
                vote_pool[j][y_train[i]] += w_train[i] # y_train[i]表示 i 所在类的那个指标，往这个类的投票箱子里投权重那么多票

        #print("vote number",vote_pool)
        # 下面开始每一行各归某一行统票
        max_indices = torch.argmax(vote_pool, dim=1) #torch.argmax 对于 dim=1 这个维度考察，找到票数最多的，存入 1 行多列的 max_indices 中

        #print("max_indices",max_indices)

        # 这个 max_indices 代表胜出的类，第一个分量就是 x_test 中的第一个点应该归到的类，第二个分量就是 x_test 中第二个点应该归到的类
        return max_indices


def adjust_weight(pred_labels,y_test,w_train,w_test,N):   # 传入参数 N，即分类的总个数，相当于上图中的 N
    # 方法是，根据上面得到的 pred_labels 和我的 y_test 一个分量一个分量对比过去，如果第 i 个分量不同，就代表第 i 个数据是归类错误的
    error_index=torch.nonzero(pred_labels-y_test)
   # print("error_index",error_index)
    # 先计算分类错误率，error（就是上图中的 r_error）
    numerator=0   # 分子先设置为 0
    for i in error_index:
      #print("index=",i.item())
      #print("w_test[i.item()]",w_test[i.item()])
      numerator += w_test[i.item()]
      # print("numerator is",numerator)

    error = numerator/sum(w_test).item()  # 分母是 test 集中的权重之和，分子是 tets 集里分错的权重之和
    print("error",error)

    # 下面计算 alpha
    if error >0 and error < 1 :
      alpha = math.log((1-error)/error)+math.log(N-1)   # 这里的几段是为了防止分母为 0 的报错，error 得在 0,1 中间
    #print("alpha",alpha)
    elif error==1:
        alpha=0
    else:           # 此时 error=0，证明训练很好了，给它安排较大的权重，比如 10
        alpha = 1000
    # 下面更新测试的样本权重，仅仅对于分错的遍历就行了，因为分对的那些权重是不变的（见上面的公式）
    for i in error_index:
      w_test[i.item()]=w_test[i.item()]*math.exp(alpha)
    #print("new wieght",w_test)
```

```
    # 下面归一化权重，此时是将所有的样本一起归一化，要包括 train 和 test 两者
    totalsum=sum(w_train)+sum(w_test)
    w_train=w_train/totalsum
    w_test=w_test/totalsum
    #print("new wieght",w_train,w_test)
    return w_train, w_test, alpha

# 先打乱
def generate_test_and_train(x_data,y_data,w_data):
    permuted_indices = torch.randperm(x_data.size(0)) # 这里取其他的几个也是可以的，反正列数都是一样的
    #print(permuted_indices)

    # 使用这个排列对张量进行重新排列
    x_data = x_data[permuted_indices]
    y_data = y_data[permuted_indices]
    w_data = w_data[permuted_indices]

    # 打乱后把前 3 个分出去，代表 test，剩下的代表 train

    x_train = x_data[4:]
    y_train = y_data[4:]
    w_train = w_data[4:]

    x_test = x_data[:4]
    y_test = y_data[:4]
    w_test = w_data[:4]

    #print("随机打乱第 0 维度后的张量: \n", x_data,y_data,w_data)
    #print(x_test,x_train)
    #print(y_test,y_train)
    return x_train,x_test,y_train,y_test,w_train,w_test # 输出一共 6 个变量

################################## 主函数 ##################################

class ModelStructure:
    def __init__(self, tensor1, tensor2, tensor3,tensor4):
        self.tensor1 = tensor1
        self.tensor2 = tensor2
        self.tensor3 = tensor3
        self.tensor4 = tensor4

classifier_array = []

def Classify(classifier_array,x_data,y_data,knn):
    # 构造一个大投票池
    big_vote_pool = torch.zeros(x_data.size()[0],3) #x_data().size()[0]行 3 列，3 其实是分的类别数，每一行代表考察的样本，那一行就是他的投票池

    for subclassifier in classifier_array:
      pred_labels = knn(subclassifier.tensor1, subclassifier.tensor2, x_data ,
subclassifier.tensor3)
      # 这里，带预测的数据就是 x_data 原本的样本集，而其他三个位置分别是 classifier_array 中的某一个结构体中的几个张量
```

```
        #print("pred_labels",pred_labels,"alpha",subclassifier.tensor4)
        for row in range(0,x_data.size()[0]):  # 对于每一个样本遍历，此时是某一个固定的子分类器上台投票
            big_vote_pool[row][pred_labels[row]] += subclassifier.tensor4 # 投出大小为 alpha 的一票

    #print(big_vote_pool)
    final_decision = torch.argmax(big_vote_pool, dim=1)
    #print(final_decision)
    predict_error_index=torch.nonzero(final_decision-y_data)
    print(predict_error_index.size(0))
    print("error rate",predict_error_index.size(0)/final_decision.size(0))



def main():
    # 这是原始输入数据，请在此自拟
    x_data = torch.tensor([[1.0, -2.0], [-2.0, 3.0], [3.0, 4.0], [1.2, -0.5],[-3.2,4.5],[0.02,
3.5], [1.0, 2.0],[4,12],[10,-2],[3,14],[2.7,6],[5,1],[-4,2],[6,-2],[-0.5,2],[2,8],
                          [3,0],[2,1],[-2,2],[-1.5,5],[2,4.7],[3.8,10],[0.5,3],[0.6,1.8],[2,3],
[0.2,1.5],[-0.5,1],[-0.8,0.8],[-1,2.1]])
    y_data = torch.tensor([0,2,1,0,2,  2,1,1,0,1,  1,0,2,0,2,1, 0,0,2,2,1, 1,2,1,1,1,1,1,2])
    w_data = torch.tensor([1,1,1,1,1,  1,1,1,1,1,  1,1,1,1,1,1, 1,1,1,1,1, 1,1,1,1,1,1,1,1]) #
初始权重，可以自己设置

    knn = KNN(3) #knn 中 k 取几

    for cnt in range(0,5):

        x_train,x_test,y_train,y_test,w_train,w_test =
generate_test_and_train(x_data,y_data,w_data)

        pred_labels = knn(x_train, y_train, x_test,w_train)

        w_train, w_test,alpha = adjust_weight(pred_labels,y_test,w_train,w_test,3) # 一共 3 类进行
测试
        #print("weight_train",w_train,"weight_test",w_test)

        # 把这个子分类器存起来，存在一个 classifier[]内部，要的就是 x_train,y_train,w_train 和 alpha，这几个
量表征了这个子分类器
        classifier_array.append(ModelStructure(x_train,y_train,w_train,alpha))

        # 把数据集和训练集重新拼起来，以便下一轮打乱

        x_data = torch.cat((x_train, x_test), dim=0)
        y_data = torch.cat((y_train, y_test), dim=0)
        w_data = torch.cat((w_train, w_test), dim=0)
        # print(x_data,y_data,w_data )

    print("\n\n Here is my classifier\n")
    for item in classifier_array:
        print("x_train:", item.tensor1)
        print("y_train:", item.tensor2)
        print("w_train:", item.tensor3)
        print("alpha:", item.tensor4)
    # 预测，试试看效果
```

```
    # 这是在原始数据集上的预测表现
    print("----------in the original data set, the prediction result is---------")
    Classify(classifier_array,x_data,y_data,knn)

    print("----------now lets test it with new input dots, the prediction result is---------")
    # 接下来这两个是全新的样本点，与之前是毫无关系的，纯粹是看看能否预测
    testdata_x = torch.tensor([[1.0, -1.0], [-2.0, 1.2], [3.0, 10], [1.2, 0],[-1,2],  [1.0,
-1.5], [-1.7, 0.5], [3, 3], [1, 0.1],[-1,2.4]])
    testdata_y = torch.tensor([0,2,1,0,2,0,1,1,0,2])
    Classify(classifier_array,testdata_x ,testdata_y ,knn)
main()
```

实验效果：我采用了如下图所示的训练样本点(x_data 与 y_data)，其中不同颜色代表不同的类别

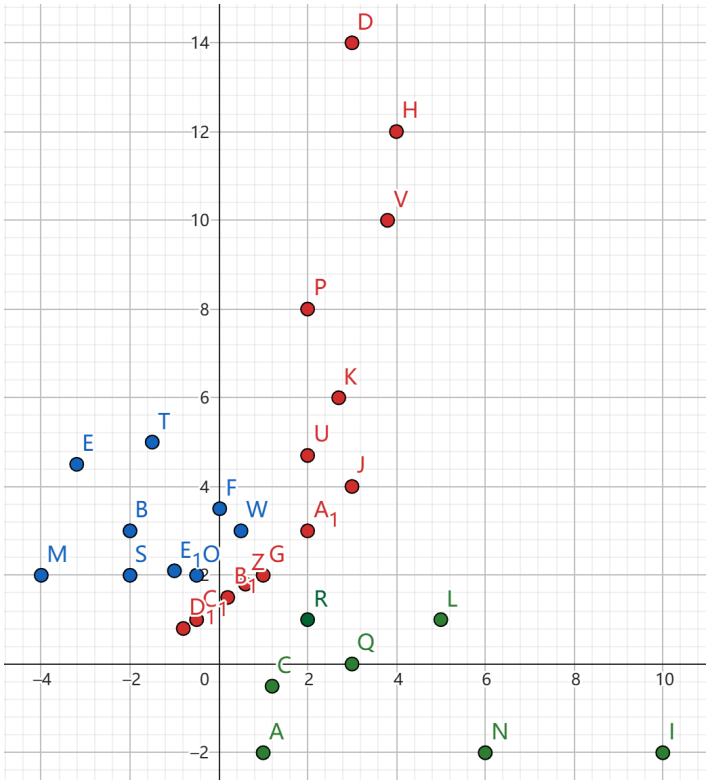

Figure 2: 代码中的 data 点

输出