AdaBoost代码解析

准确率基本为100%

```
"""
* @author   孟子喻
* @time     2021.5.16
* @file     classify_iris.py
*           adaboost.py
"""
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from adaboost import AdaBoost
import matplotlib.pyplot as plt
import csv
from sklearn.model_selection import train_test_split

# 鸢尾花(iris)数据集
# 数据集内包含 3 类共 150 条记录，每类各 50 个数据，
# 每条记录都有 4 项特征：花萼长度、花萼宽度、花瓣长度、花瓣宽度，
# 可以通过这4个特征预测鸢尾花卉属于（iris-setosa, iris-versicolour, iris-virginica）中的哪一品种。
# 这里只取前100条记录，两项特征，两个类别。

def create_data():
    iris = load_iris()
    df = pd.DataFrame(iris.data, columns=iris.feature_names)
    df['label'] = iris.target
    df.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'label']
    data = np.array(df.iloc[:100, [0, 1, -1]])
    for i in range(len(data)):
        if data[i, -1] == 0:
            data[i, -1] = -1
    # print(data)
    return data

def calc_acc(y, y_hat):
    idx = np.where(y_hat == 1)
    TP = np.sum(y_hat[idx] == y[idx])
    idx = np.where(y_hat == -1)
    TN = np.sum(y_hat[idx] == y[idx])
    return float(TP + TN)/len(y)


if __name__ == "__main__":

    trainset = create_data()

    features, labels = trainset[:, :2], trainset[:, -1]
    x = features
    y = labels

```

```
    ada = AdaBoost()
    clf_num = ada.fit(x, y)  # 返回使用的弱分类器数量
    print(clf_num)
    ada.show_para() # 显示一些参数

    # 进行预测
    y_hat = ada.output_final_pred(x, clf_num)

    acc = calc_acc(y, y_hat)    # 计算准确率
    print("Accuracy: ", str(100* acc), "%")

    plt.figure()

    for i in range(clf_num):
        if ada.classifier[i].decision_feature == 0:
            plt.plot([ada.classifier[i].decision_threshold,
ada.classifier[i].decision_threshold], [0, 10])
        else:
            plt.plot([0, 10], [ada.classifier[i].decision_threshold,
ada.classifier[i].decision_threshold])

    for item in trainset:
        if item[-1] == 1:
            plt.scatter(item[0], item[1], s=30, marker='+', color='blue')
        else:
            plt.scatter(item[0], item[1], s=30, marker='_', color='red')
    plt.xlim(3, 8)
    plt.ylim(1, 6)
    plt.show()

```

这一段代码和我之前的代码一样，用于把数据集分类，然后输入模型

下面是AdaBoost的代码详解

初始化模型

```
class AdaBoost:
    def __init__(self):
        self.W = {}                 # 权重weight
        self.classifier = {}        # 分类器字典，存储用到的多个弱分类器
        self.alpha = {}             # 每个分类器的影响权重
        self.pred = {}              # 每个分类器预测的值
        self.classifier_num = 0     # 弱分类器数量
```

以下是具体的训练代码，每一句的作用基本都放在注释里了

```
# 训练
    def fit(self, x, y, max_clf_num=15):

        pred = {}
        data_size = len(y)  # 直接将所有实例一个batch进行训练，所以要经常用到一个
batch的大小

        for iter in range(max_clf_num):
```

```
            # 在允许的最大分类器数量范围内，对每个弱分类器进行初始化
            self.W.setdefault(iter)              # 每一个变量的权重，不同弱分类器对应的权重不一样
            self.classifier.setdefault(iter)     # 分类器字典
            self.alpha.setdefault(iter)          # 该分类器的alpha值
            self.pred.setdefault(iter)           # 该分类器的预测值
            # setdefault为检查iter是否存在于字典的keys中，若不存在则新建key并将value设置为None

        # 进行迭代
        for iter in range(max_clf_num):
            # 第一次迭代时初始化实例权重W，此时每个实例对分类器影响相同，所以设置为1/data_size
            if iter == 0:
                self.W[iter] = np.ones(data_size) / data_size
                self.W[iter] = self.W[iter].reshape([data_size, 1])
            # 如果不是第一次迭代，则更新weight
            else:
                self.W[iter] = self.cal_W(iter, y, pred[iter - 1])
            # pred[iter-1]是计算出的新的预测值

            # 使用弱分类器进行分类
            self.classifier[iter] = LineClassifier()
            self.classifier[iter].fit(x, y, self.W[iter])
            pred[iter] = self.classifier[iter].pred     # 使用弱分类器进行一轮预测

            # 计算误差
            error = self.cal_e(y, pred[iter], self.W[iter])
            # 计算alpha
            self.alpha[iter] = self.cal_alpha(error)
            # 计算最终预测值（每次更新）
            final_predict = self.cal_final_pred(iter, self.classifier, data_size)

            print('iteration:%d' % (iter + 1))
            self.classifier_num = iter

            # 输出最终结果
            if self.cal_final_e(y, final_predict) == 0 or error == 0:
                print('cal_final_predict:%s' % (final_predict))
                break
            print('self.decision_key=%s' % (self.classifier[iter].decision_key))
            print('self.decision_feature=%d' % (self.classifier[iter].decision_feature))
            print('decision_threshold=%f' % (self.classifier[iter].decision_threshold))
            print('error:%f alpha:%f' % (error, self.alpha[iter]))
            print('\n')
        # 输出弱分类器数量
        return self.classifier_num
```

主函数中用到的一些其他函数，基本都对应课本公式

```
    def cal_e(self, y, pred, W):
        ret = 0
```

```
        for i in range(len(y)):
            if y[i] != pred[i]:
                ret += W[i]
        return ret

    # 迭代更新alpha
    def cal_alpha(self, e):
        if e == 0:
            return 10000
        elif e == 0.5:
            return 0.001
        else:
            return 0.5 * np.log((1 - e) / e)  # 对应课本公式8.2，返回Gm的系数

    # 计算最终的预测值
    def cal_final_pred(self, i, classifier, data_size):
        ret = np.array([0.0] * data_size)
        for j in range(i + 1):
            ret += self.alpha[j] * classifier[j].pred
        return np.sign(ret)
```

以下是弱分类器

初始化部分

```
class LineClassifier:
    def __init__(self):
        self.W = None                       # 传入的每个实例的权重
        self.decision_key = None            # 取大于阈值的一侧还是小于的一侧（因为是划定正方形一个区域，所以要有方向）
        self.decision_feature = None        # 哪个特征起到分类作用x或y
        self.decision_threshold = None      # 分类的阈值
        self.pred = None                    # 该分类器本次的预测值
```

训练部分

```
 # 训练
    def fit(self, X, y, W):
        self.W = W                  # W是特征权重
        dic = self.cal_dic(X)       # 计算用每个实例的每个特征做阈值时，所有实例点在阈值两侧的分类情况
        e_dic = self.cal_e_dic(y, dic)          # 计算生成对应的error字典
        e_min, self.decision_key, self.decision_feature, e_min_i = self.cal_e_min(e_dic)     # 计算最小的error
        self.decision_threshold = X[e_min_i, self.decision_feature]      # 把最小error对应的阈值取出来输出
        self.pred = dic[self.decision_key][self.decision_feature][e_min_i]
    # 用本次训练出的弱分类器进行一次预测
```

统计数据以便于后续阈值选取部分

```
    def cal_dic(self, X):
        ret_gt = {}     # gt代表greater than
        for i in range(X.shape[1]):
            # 此处为取x或y,记为特征i
            ret_gt[i] = []
```

```
            for j in range(X.shape[0]):
                # 此处为取一个实例，记为实例j
                temp_threshold = X[j, i]    # 将取到的某个实例j的特征i作为阈值
                temp_line = []
                for k in range(X.shape[0]):     # 继续取所有实例j的特征i，如果超过了阈值，则标记为1存入temp_line
                    if X[k, i] >= temp_threshold:
                        temp_line.append(1)
                        # 如果
                    else:                       # 如果没超过阈值，则标记为-1存入temp_line
                        temp_line.append(-1)
                ret_gt[i].append(temp_line)     # 将取到的与阈值的比较情况存入ret_gt[i]

        ret_lt = {}     # lt代表lower than
        for i in range(X.shape[1]):
            ret_lt[i] = []
            for j in range(X.shape[0]):
                temp_threshold = X[j, i]
                temp_line = []
                for k in range(X.shape[0]):
                    if X[k, i] <= temp_threshold:
                        temp_line.append(1)
                    else:
                        temp_line.append(-1)
                ret_lt[i].append(temp_line)
        ret = {'gt': ret_gt, 'lt': ret_lt}  # 是一个字典，里面存储了每一个实例的每一个特征作为阈值时，有几个点大于或小于它
        return ret
```