机器学习 第8章 作业

8.1

8.3式如下:

$$P(H(x) \neq f(x)) = \sum_{k=0}^{\lfloor T/2 \rfloor} \binom{T}{k} (1-\epsilon)^k \epsilon^{T-k} \leq exp(-\frac{1}{2}T(1-2\epsilon)^2) \,.$$

第一个等式很好理解，$\binom{T}{k}(1-\epsilon)^k\epsilon^{T-k}$ 表示 T 个基分类器中有 $T-k$ 个分类器预测错误的概率，$\sum_{k=0}^{[T/2]}$ 即将所有 $H(x) \neq f(x)$ 下可能预测错误的基分类器个数遍历取总。或者可以直接套用8.43式，$P(H(x) \neq f(x)) = P(H(T) \leq [T/2])$，其中$H(T)$表示$T$个基分类器中，预测正确的个数，即可得第一个等式。

关于第二个不等式，
取8.44中 $\delta = 1-\epsilon-\frac{[T/2]}{T}$， $\frac{[T/2]}{T} \leq \frac{1}{2}$ 于是:

$$P(H(T) \leq \lfloor T/2 \rfloor) = P(H(T) \leq (1-\epsilon-\delta)T) \leq e^{-2(1-\epsilon-\frac{\lfloor T/2 \rfloor}{T})^2 T} \leq e^{-\frac{1}{2}T(1-2\epsilon)^2}$$

于是8.3得证。这里需要注意一点$e^{-2(1-\epsilon-\frac{[2/2]}{T})^2 T} \leq e^{-\frac{1}{2}T(1-2\epsilon)^2}$是建立在 $\epsilon < \frac{1}{2}$ 的基础上的，

8.3

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt


class Node(object):
    def __init__(self):
        self.feature_index = None
        self.split_point = None
        self.deep = None
        self.left_tree = None
        self.right_tree = None
        self.leaf_class = None


def gini(y, D):
    '''
    计算样本集y下的加权基尼指数
    :param y: 数据样本标签
    :param D: 样本权重
    :return:  加权后的基尼指数
    '''
    unique_class = np.unique(y)
    total_weight = np.sum(D)

    gini = 1
    for c in unique_class:
        gini -= (np.sum(D[y == c]) / total_weight) ** 2

    return gini
```

```python
def calcMinGiniIndex(a, y, D):
    '''
    计算特征a下样本集y的的基尼指数
    :param a: 单一特征值
    :param y: 数据样本标签
    :param D: 样本权重
    :return:
    '''

    feature = np.sort(a)
    total_weight = np.sum(D)

    split_points = [(feature[i] + feature[i + 1]) / 2 for i in
range(feature.shape[0] - 1)]

    min_gini = float('inf')
    min_gini_point = None

    for i in split_points:
        yv1 = y[a <= i]
        yv2 = y[a > i]

        Dv1 = D[a <= i]
        Dv2 = D[a > i]
        gini_tmp = (np.sum(Dv1) * gini(yv1, Dv1) + np.sum(Dv2) * gini(yv2, Dv2))
/ total_weight

        if gini_tmp < min_gini:
            min_gini = gini_tmp
            min_gini_point = i

    return min_gini, min_gini_point


def chooseFeatureToSplit(X, y, D):
    '''

    :param X:
    :param y:
    :param D:
    :return: 特征索引, 分割点
    '''
    gini0, split_point0 = calcMinGiniIndex(X[:, 0], y, D)
    gini1, split_point1 = calcMinGiniIndex(X[:, 1], y, D)

    if gini0 > gini1:
        return 1, split_point1
    else:
        return 0, split_point0


def createSingleTree(X, y, D, deep=0):
    '''
    这里以C4.5 作为基学习器，限定深度为2，使用基尼指数作为划分点，基尼指数的计算会基于样本权重，
```

```
    不确定这样的做法是否正确，但在西瓜书p87, 4.4节中, 处理缺失值时, 其计算信息增益的方式是将样本权重考虑在内的，
    这里就参考处理缺失值时的方法。
    :param X: 训练集特征
    :param y: 训练集标签
    :param D: 训练样本权重
    :param deep: 树的深度
    :return:
    '''

    node = Node()
    node.deep = deep

    if (deep == 2) | (X.shape[0] <= 2):  # 当前分支下，样本数量小于等于2 或者 深度达到2时，直接设置为也节点
        pos_weight = np.sum(D[y == 1])
        neg_weight = np.sum(D[y == -1])
        if pos_weight > neg_weight:
            node.leaf_class = 1
        else:
            node.leaf_class = -1

        return node

    feature_index, split_point = chooseFeatureToSplit(X, y, D)

    node.feature_index = feature_index
    node.split_point = split_point

    left = X[:, feature_index] <= split_point
    right = X[:, feature_index] > split_point

    node.left_tree = createSingleTree(X[left, :], y[left], D[left], deep + 1)
    node.right_tree = createSingleTree(X[right, :], y[right], D[right], deep + 1)

    return node


def predictSingle(tree, x):
    '''
    基于基学习器，预测单个样本
    :param tree:
    :param x:
    :return:
    '''
    if tree.leaf_class is not None:
        return tree.leaf_class

    if x[tree.feature_index] > tree.split_point:
        return predictSingle(tree.right_tree, x)
    else:
        return predictSingle(tree.left_tree, x)


def predictBase(tree, X):
    '''
```

```
    基于基学习器预测所有样本
    :param tree:
    :param X:
    :return:
    '''
    result = []

    for i in range(X.shape[0]):
        result.append(predictSingle(tree, X[i, :]))

    return np.array(result)


def adaBoostTrain(X, y, tree_num=20):
    '''
    以深度为2的决策树作为基学习器，训练adaBoost
    :param X:
    :param y:
    :param tree_num:
    :return:
    '''
    D = np.ones(y.shape) / y.shape  # 初始化权重

    trees = []  # 所有基学习器
    a = []  # 基学习器对应权重

    agg_est = np.zeros(y.shape)

    for _ in range(tree_num):
        tree = createSingleTree(X, y, D)

        hx = predictBase(tree, X)
        err_rate = np.sum(D[hx != y])

        at = np.log((1 - err_rate) / max(err_rate, 1e-16)) / 2

        agg_est += at * hx
        trees.append(tree)
        a.append(at)

        if (err_rate > 0.5) | (err_rate == 0):  # 错误率大于0.5 或者 错误率为0时，则直
接停止
            break

        # 更新每个样本权重
        err_index = np.ones(y.shape)
        err_index[hx == y] = -1

        D = D * np.exp(err_index * at)
        D = D / np.sum(D)

    return trees, a, agg_est


def adaBoostPredict(X, trees, a):
    agg_est = np.zeros((X.shape[0],))
```

```
    for tree, am in zip(trees, a):
        agg_est += am * predictBase(tree, X)

    result = np.ones((X.shape[0],))

    result[agg_est < 0] = -1

    return result.astype(int)


def pltAdaBoostDecisionBound(X_, y_, trees, a):
    pos = y_ == 1
    neg = y_ == -1
    x_tmp = np.linspace(0, 1, 600)
    y_tmp = np.linspace(-0.2, 0.7, 600)

    X_tmp, Y_tmp = np.meshgrid(x_tmp, y_tmp)

    Z_ = adaBoostPredict(np.c_[X_tmp.ravel(), Y_tmp.ravel()], trees,
a).reshape(X_tmp.shape)
    plt.contour(X_tmp, Y_tmp, Z_, [0], colors='orange', linewidths=1)

    plt.scatter(X_[pos, 0], X_[pos, 1], label='1', color='c')
    plt.scatter(X_[neg, 0], X_[neg, 1], label='0', color='lightcoral')
    plt.legend()
    plt.show()


if __name__ == "__main__":
    data_path = r'..\data\watermelon3_0a_Ch.txt'

    data = pd.read_table(data_path, delimiter=' ')

    X = data.iloc[:, :2].values
    y = data.iloc[:, 2].values

    y[y == 0] = -1

    trees, a, agg_est = adaBoostTrain(X, y)

    pltAdaBoostDecisionBound(X, y, trees, a)
```

8.5

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.utils import resample


def stumpClassify(X, dim, thresh_val, thresh_inequal):
    ret_array = np.ones((X.shape[0], 1))

    if thresh_inequal == 'lt':
```

```
        ret_array[X[:, dim] <= thresh_val] = -1
    else:
        ret_array[X[:, dim] > thresh_val] = -1

    return ret_array


def buildStump(X, y):
    m, n = X.shape
    best_stump = {}

    min_error = 1

    for dim in range(n):

        x_min = np.min(X[:, dim])
        x_max = np.max(X[:, dim])

        # 这里第一次尝试使用排序后的点作为分割点，效果很差，因为那样会错过一些更好的分割点；
        # 所以后来切割点改成将最大值和最小值之间分割成20等份。

        # sorted_x = np.sort(X[:, dim])
        # split_points = [(sorted_x[i] + sorted_x[i + 1]) / 2 for i in range(m -
1)]

        split_points = [(x_max - x_min) / 20 * i + x_min for i in range(20)]

        for inequal in ['lt', 'gt']:
            for thresh_val in split_points:
                ret_array = stumpClassify(X, dim, thresh_val, inequal)

                error = np.mean(ret_array != y)

                if error < min_error:
                    best_stump['dim'] = dim
                    best_stump['thresh'] = thresh_val
                    best_stump['inequal'] = inequal
                    best_stump['error'] = error
                    min_error = error

    return best_stump


def stumpBagging(X, y, nums=20):
    stumps = []
    seed = 16
    for _ in range(nums):
        X_, y_ = resample(X, y, random_state=seed)  # sklearn 中自带的实现自助采样的
方法
        seed += 1
        stumps.append(buildStump(X_, y_))
    return stumps


def stumpPredict(X, stumps):
    ret_arrays = np.ones((X.shape[0], len(stumps)))
```

```
    for i, stump in enumerate(stumps):
        ret_arrays[:, [i]] = stumpClassify(X, stump['dim'], stump['thresh'],
stump['inequal'])

    return np.sign(np.sum(ret_arrays, axis=1))


def pltStumpBaggingDecisionBound(X_, y_, stumps):
    pos = y_ == 1
    neg = y_ == -1
    x_tmp = np.linspace(0, 1, 600)
    y_tmp = np.linspace(-0.1, 0.7, 600)

    X_tmp, Y_tmp = np.meshgrid(x_tmp, y_tmp)
    Z_ = stumpPredict(np.c_[X_tmp.ravel(), Y_tmp.ravel()],
stumps).reshape(X_tmp.shape)

    plt.contour(X_tmp, Y_tmp, Z_, [0], colors='orange', linewidths=1)

    plt.scatter(X_[pos, 0], X_[pos, 1], label='1', color='c')
    plt.scatter(X_[neg, 0], X_[neg, 1], label='0', color='lightcoral')
    plt.legend()
    plt.show()


if __name__ == "__main__":
    data_path = r'..\data\watermelon3_0a_Ch.txt'

    data = pd.read_table(data_path, delimiter=' ')

    X = data.iloc[:, :2].values
    y = data.iloc[:, 2].values

    y[y == 0] = -1

    stumps = stumpBagging(X, y, 21)

    print(np.mean(stumpPredict(X, stumps) == y))
    pltStumpBaggingDecisionBound(X, y, stumps)
```

8.7

决策树的生成过程中，最耗时的就是搜寻最优切分属性；随机森林在决策树训练过程中引入了随机属性选择，大大减少了此过程的计算量；因而随机森林比普通决策树Bagging训练速度要快。