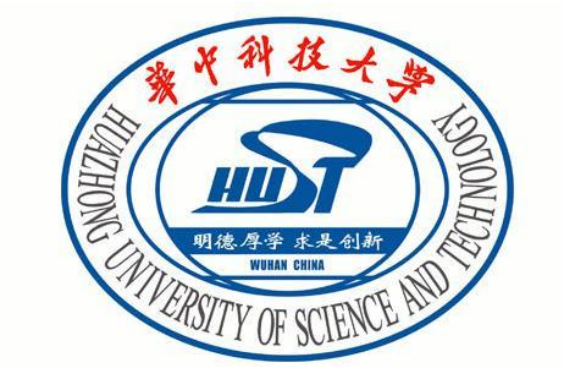
华中科技大学

计算机科学与技术学院

《机器学习》结课报告



专　　业：计算机科学与技术

班　　级：

学　　号：

姓　　名：raindaydream

成　　绩：

指导教师：何琨

完成日期：　2022 年　7　月　6　日

目录

Adaboost 算法实现

1. 实验要求

（1）总体要求：

要求分别实现以对数几率回归和决策树桩为基分类器的 AdaBoost 算 法，其中对数几率回归请参考课件第六讲，决策树桩指只有一层的决策树。

（2）所做工作：

①实现对数几率回归：最主要的是损失函数和参数调整，最终的参数调整为 learning_rate=0.05。

②决策树桩实现：最主要的是选择出最重要的特征值和步长

③adaboost 实现：最主要的是调整权重

2. 算法设计与实现

2.1 数据预处理

数据归一化处理。这次的实验中所给的数据是稀疏并且规则性较弱的，因此需要进行相关处理，即归一化。在读取了数据以后，便将每个值在自己所在的特征下进行等比例的放缩，使得最小值为 0，最大值为 1，其他的按照 x = (x-min)/(max-min)求得归一化的值。

2.2 对数几率回归实现

1. 对数几率函数

Sigmoid 函数是实现线性回归映射到二分类的重要函数。同时该函数还是预测函数，根据特征得到二分类的分类值。

```
#对数几率函数
def sigmoid(self, z):
    return np.where(z >= 1, 1 / (1 + np.exp(-z)), np.exp(z) / (1 + np.exp(z)))
```

图 1 sigmoid 函数实现

2.损失函数

在调整参数 w 和 b 的过程中，需要用到预测结果和实际结果，一般来讲是用两个数组的均方误差作为损失函数的结果。

$$L = -\left[y \log \hat{y} + (1-y) \log (1-\hat{y})\right]$$

图 2 损失函数

但是在这个数据下，该损失函数的正确率并不高，因此采用了另一种损失函数的求法：以实际值和样本值的差值与权重相乘得到最后的损失函数值。

```
def log_loss(self,data, label, w, b, weights):
    result = self.sigmoid(np.dot(data, w) + b)
    loss = (label-result)*weights
    dw = np.dot(data.T, loss)
    db = np.sum(loss)
    return result, loss, dw, db
```

图 3 损失函数的实现

3.模型训练

对数几率回归的模型训练最重要的是训练出最后的 w 和 b 参数，这里需要得到每次的 dw 和 db 来类比对数几率函数在该处的导数，在损失函数中可以得到每次的 dw 和 db，在初始的时候需要设置 learning_rate 来作为系数以保证每次调整 w 和 b 的时候是向损失函数减小的方向调整的，而且一次调整的不能过大，以保证没有越过使得损失函数达到最小值的参数值。

因此调试 learning_rate，使其较为接近 0，不断测试、调参来提高正确率。

4. 预测函数

在训练出合适的 w 和 b 之后，将其用于测试集上进行测试，采用 sigmoid 函数得到最后的分类。

但是 sigmoid 函数所得到的值并非是 0 和 1 两种分立的值，而是[0,1]这个区间中连续的值，因此需要进一步进行区分，也就是以 0.5 为分界线，大于即为 1，否则为 0。

2.3 决策树桩实现

1.决策树桩构建弱分类器

首先先取数据的某一个特征的所有数据，对其进行相关的计算。最重要的是该特征的阈值和判断条件。

（1）阈值

最后返回的阈值应该为在该特征的判断下使得正确率达到最高的值，因此需要不断调试得到这个最佳阈值。

实现找到最佳阈值的方法是设置一个步长，通过步长实现对数据的最小值到最大值的遍历，在遍历的过程中计算该阈值下的错误率，比较错误率得到错误率最小的阈值。

```
length_step = (features_max - features_min) / self.learning_rate
flag = 0
result = None
for i in range(1, int(length_step)):
    v = features_min + self.learning_rate * i
```

图 4　步长设置

（2）判断条件

判断条件有两种：①大于阈值判定为 1，否则为 0；②大于阈值判定为 0，否则为 1。具体选择哪种需要按照正确率来判断。

采用两种判断条件分别进行预测，然后得到一个预测结果，该预测结果和实际的结果相比较得到两种情况下的错误率，选择错误率小的进行标记，最后记录使得在该特征决定下的预测结果错误率最小的阈值、判断条件。

2. 训练得到最后的决定特征

该数据集中有很多个特征值，我们在利用决策树桩找到一个特征下的最佳阈值和判断条件之后，可以得到该特征以及其所得到的阈值和判断条件下的预测结果，该预测结果可以与实际结果比较得到错误率，找到使得错误率最小的特征值及其阈值和判断条件来作为返回值。

说的通俗一点，就是遍历数据集中的每一个特征，对每一个特征进行决策树桩的函数分析得到阈值、条件和错误率，比较得到最小错误率对应的特征。

3. 预测函数

在对一个数据集训练完决策树模型后，可以得到决定特征及其参数，接下来就可以对一个输入的数据进行分析，只需要关注该数据的决定特征，根据判断条件直接输出预测结果。

```
def judge_result(self, test, threshold, flag):
    len_test = len(test)
    if flag == 1:
        result = np.array([1 if test[i] > threshold else -1 for i in range(len_test)])
    else:
        result = np.array([-1 if test[i] > threshold else 1 for i in range(len_test)])
    return result
```

图 5　决策树判断

2.4 Adaboost 算法实现

Adaboost 算法的原理是利用多个弱分类器作为基分类器，通过上一个基分类器的预测结果和实际结果的差别调整下一个基分类器的参数来提高正确率，最后得到一个强分类器。

1.adaboost 需要的参数

（1）alpha

Alpha 是各个基分类器在最后分类的时候所占的权重，具体实现与该基分类器的分类错误率有关。

```
def _alpha(self, error):
    return 0.5 * np.log((1 - error) / error)
```

图 6　计算 alpha

（2）规范化因子 Z

Z 是权重的归一化参数，权重在每个基分类器实现后需要进行调整，调整过后可能和并不为 1，因此需要进行归一化调整，使得权重和为 1。Z 具体的实现与误差率有关。

```
def _Z(self, weights, alpha, result):
    return sum([weights[i] * np.exp(-1 * alpha * self.Y[i] * result[i]) for i in range(self.M)])
```

图 7　归一化参数 Z

（3）权重更新

在一个基分类器实现后，需要对样本的权重进行调整。错误的样本的权重要增加，正确的样本的权重要减小，以此实现在下一个基分类器中错误的样本被关注的概率更大，错误的样本被修正为正确的结果的概率更大。

权重的更新与误差率和归一化参数相关，alpha 因子表现了一个基分类器的误差率，因此可以用 alpha 因子对权重进行更新。

```
def _w(self, alpha, result, Z):
    for i in range(self.M):
        self.weights[i] = self.weights[i] * np.exp(-1 * alpha * self.Y[i] * result[i]) / Z
```

图 8　样本权重更新

2. 模型训练

（1）以决策树为基分类器

该训练流程为：训练决策树桩->调整权重->训练下一个决策树桩

①初始化参数，除了数据和标签外最重要的是 learning_rate 和 estimators，在决策树中，learning_rate 作为步长，estimators 是基分类器的个数。

②调用决策树桩的生成函数，即决策树桩的训练模型，得到决策树桩的决定特征、阈值和判断条件得到基分类器，并将基分类器相关特征保存。

③根据该基分类器得到 alpha，Z，误差率等参数，根据这些参数调整权重，并保存基分类器的权重 alpha。

④判断基分类器是否达到指定数目，若达到就结束训练，否则继续执行②

（2）以对数几率回归为基分类器

该训练流程为：训练参数 w 和 b->调整权重->训练下一个 w 和 b

①初始化参数，除了数据和标签外最重要的是 learning_rate 和 estimators，在对数几率回归中，estimators 代表的是基分类器的个数，learning_rate 代表的是梯度下降的系数。

②调用对数几率回归函数，得到 w 和 b 两个重要参数得到基分类器，并将基分类器相关特征保存。

③根据该基分类器得到 alpha，Z，误差率等参数，根据这些参数调整权重，并保存基分类器的权重 alpha。

④判断基分类器是否达到指定数目，若达到就结束训练，否则继续执行②

3.预测函数

预测函数即根据已经训练好的模型判断输入的数据所属的类别。根据 base 的值决定调用哪个模型，如果调用决策树模型，就可以调用决策树的预测函数，同理有对数几率回归。

不管调用哪个模型，adaboost 都会根据基分类器及其加权进行最后的结果判断，在训练模型的时候，我们保存了每个模型对应的 alpha 值，该值即为模型在

最后判断类别时所占的权重，在利用每一个基分类器判断后得到一个判断结果，最后再将这些结果加权平均，得到最后的预测值。

在基分类器加权平均后，得到的结果并不是 0 和 1 两个离散量，而是一些连续的值。对于决策树，加权平均的结果通过判断是否大于 0 来决定最后的结果是否为 1，这里采用 0 作为分界值是因为在前面的决策树模型中我们设定的预测值为 1 和-1；对于对数几率回归，加权平均的结果以 0.5 为分界值判定最终结果为 0 还是 1。

3. 实验环境与平台

Python 版本：3.8

实验平台：anaconda+spyder、pycharm

4. 结果与分析

（1）预测结果：

```
D:\users\anaconda\anaconda3\python.exe "D:/study/coding practice/machine learning/U202015562_CS2009_徐雨梦_adaboost/evaluate.py"
0.9071311159815189
0.9128376377206491
0.9152832898945622
0.8981081921573273
```

图 9　对数几率回归的正确率（learning_rate=0.005）

```
D:\users\anaconda\anaconda3\python.exe "D:/study/coding practice/machine learning/code/evaluate.py"
0.8831529143466412
0.8888594360857717
0.8913050882596847
0.8945659578249021

Process finished with exit code 0
```

图 10 决策树桩的正确率（learning_rate=0.01)

（2）不同因素对结果的影响：

①基分类器的数目

基分类器太少会导致正确率比较低，因为错误的结果所产生的影响不够大，因此错误样本不足以被修正；基分类器太多也会导致正确率下降，因为会对错误样本过多的修正，因此有可能造成过拟合现象。

而且选择不同的模型作为基分类器也是差别很大的。如图所示，对数几率回归的正确率要比决策树的正确率高。

②学习率

决策树中学习率为阈值步长，如果学习率过低会导致运行时间过长但得到的结果并不能较好的提高正确率，如果学习率过高会导致阈值不够精确而使得正确率下降。

③数据预处理

这里数据预处理采用的是归一化，在归一化之前，模型训练所需要的时间更长并且正确率不够高，归一化之后，时间性能和正确率都得到了改善。具体原因分析以决策树为例，归一化之前各个特征的区域宽度不一样，相同的步长运行次数不一样，而且权重分布也会根据不同的比例分配，这样会降低正确率。

(3) 问题回答：你对 AdaBoost 算法有何新的认识？

Adaboost 充分利用了多个弱分类器的有点，最后得到了一个强分类器，相较于其他的直接就是强分类器的算法，它更好实现并且还可以达到很高的正确率，启发我们可以从另一个角度看待并没有那么优秀的算法。

5. 个人体会

这次的实验算是一个比较大的挑战，因为对 python 还有机器学习的了解在学习这门课之前并不多，而且在考试周的时间并不充裕，所以还是难度还是不小的。但是在完成这门实验的过程中，从一开始对语法不熟悉到最后可以写出来整个的 adaboost，收获还是很大的。

我先从 adaboost 开始了解学习，通过上网、查阅书籍等方式先弄懂了 adaboost 的原理，然后了解了 adaboost 的各个模块之间的关系和实现的流程，虽然尚未到代码层面，但是还是理解了不少东西；接下来对于需要用到的决策树和对数几率回归，我也是在理解层面上先搞懂了它的原理，然后看了一些经典代码学习这些算法的实现形式，之后在自己写出来适配于 adaboost 的实现形式；最后在 adaboost 中实现这两个算法的调用和相关参数的实现，算是终于把它完成了。

由于某些原因，我只能从各种考试结束后才开始做这个实验，因此时间特别紧迫，所以在一开始写的过程中也特别绝望，也导致了自己特别烦，最后是真的哭了一场才恢复到正常的学习状态。也好在哭了一场宣泄了情绪，所以才在后面学习的过程中逐渐开窍然后写出来了可以运行的代码，然后一点点调，得到最后的结果。

过程是曲折的，所以给这门课程的建议就是希望这门课以后可以提前布置一下作业，来减少同学完成作业的曲折度。