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：

指导教师：

2022 年 7 月 6 日

目录

1 实验要求

实验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，输出在不同数目基分类器条件下的 10 折交叉验证结果，并能利用拟合产生的模型对新数据进行预测，且达到一定的准确率。由于 AdaBoost 算法很难产生过拟合现象，所以不采用正则化方法。

在实现对数几率回归时需要用数值方法求系数向量的最优值，会用到梯度下降法。于是实验中调整了学习率的固定值，比较不同的学习率对对数几率回归损失函数值变化的影响。

在实现决策树桩模型时，选择划分属性的依据是（加权）错误率，即通过遍历全部的属性列和属性列下所有可能的阈值（以一定的步长）来选择令错误率最小的属性列作为划分的结点。

2 算法设计与实现

2.1 数据预处理

对于对数几率回归和决策树桩模型来说，是否标准化不会改变预测结果。但是对数几率回归中使用了梯度下降法，由于不同特征方向量纲不同，梯度下降的速度不同，对数据进行标准化处理，能够帮助梯度下降算法更快地收敛，提高代码运行效率。

此处采用 z-score 标准化，转化函数为 $y = \dfrac{y-\mu}{\sigma}$，其中 μ 为所有样本数据的均值，σ 为所有样本数据的标准差。为了保持变换的一致性，对训练集和测试集进行变换时，均采用训练集的均值、标准差，代码如下。

```
def logicStandardize(x, mean, std):
    # 通过减去均值并除以标准差的方式进行标准化
    return (x - mean) / std

std = np.sqrt(np.var(train_data, axis=0, ddof=1).tolist())  # 获取标准差
mean = np.mean(train_data, axis=0).tolist()
train_data = train_data.apply(logicStandardize, args=(mean, std), axis=1)
test_data = test_data.apply(logicStandardize, args=(mean, std), axis=1)
```

提供的测试数据集不存在缺失值，所以没有进行缺失值处理。若新的数据集存在缺失值，应先进行缺失值处理，若缺失数据较少，可以直接删去有缺失值的行；若缺失数据较多，可以使用列均值进行填充。

2.2 对数几率回归

2.2.1 算法伪代码

Algorithm 1: 对数几率回归

输入: 训练数据集矩阵 $\boldsymbol{X}$，分类真实值向量 $\boldsymbol{Y}$，训练权重 $\boldsymbol{\beta}$

1 $\boldsymbol{X} \leftarrow (\mathbf{1}; \boldsymbol{X})$; // 数据矩阵新增全为 1 的列

2 $\boldsymbol{W} \leftarrow \text{diag}(\boldsymbol{\beta})$; // 以权重为主对角线创建对角矩阵

3 $\boldsymbol{\omega} \leftarrow \mathbf{0}, \alpha = 0.5$; // 初始化系数向量和学习率

4 **while** *true* **do**

5 $\quad \hat{\boldsymbol{Y}} \leftarrow {}^{1}/_{(1+\mathrm{e}^{-\boldsymbol{\omega}^T \boldsymbol{X}})}$;

6 $\quad \boldsymbol{\nabla} \leftarrow \boldsymbol{X}^T \boldsymbol{W}(\boldsymbol{Y} - \hat{\boldsymbol{Y}})$; // 计算梯度

7 $\quad \boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \alpha \boldsymbol{\nabla}$; // 梯度下降法迭代

8 $\quad$ **if** $\|\boldsymbol{\nabla}\| < 3 \times 10^{-2}$ **then**

9 $\quad\quad$ break; // 当梯度模长小于给定值时，结束循环

10 $\quad$ **end**;

11 **end**;

12 $\hat{\boldsymbol{Y}} \leftarrow {}^{1}/_{(1+\mathrm{e}^{-\boldsymbol{\omega}^T \boldsymbol{X}})}$; // 计算预测值

13 $\hat{\boldsymbol{Y}}[\hat{\boldsymbol{Y}} > 0.5] = 1, \hat{\boldsymbol{Y}}[\hat{\boldsymbol{Y}} \leq 0.5] = 0$;

输出: 最终的分类结果向量 $\hat{\boldsymbol{Y}}$

2.2.2 算法说明

1. 令 $\boldsymbol{\omega} = (b; \boldsymbol{\omega}), \boldsymbol{X} = (\mathbf{1}; \boldsymbol{X})$，则 $\boldsymbol{\omega}^T \boldsymbol{X} + b$ 可以简写成 $\boldsymbol{\omega}^T \boldsymbol{X}$。所以第 1 行数据矩阵新增全为 1 的列，是为了便于在第 5 行和第 10 行中计算预测值。

2. 在第 6 行中使用了损失函数的梯度。对于第 i 个案例，其对数损失为

$$L_i = -\beta_i \left[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \right]$$

暂不考虑权重，由链式法则得 $\dfrac{\partial L}{\partial \omega} = \dfrac{\partial L}{\partial \hat{y}} \dfrac{\partial \hat{y}}{\partial z} \dfrac{\partial z}{\partial \omega} = \left(\dfrac{\hat{y} - y}{\hat{y}(1 - \hat{y})} \right) \Big(haty(1 - \hat{y}) \Big) x = (\hat{y} - y)x$，然后加入权重项可以得到

$$\nabla_k = \left(\frac{\partial L}{\partial \omega} \right)_k = \sum_{i=1}^{n} \beta_k (\hat{y}_i - y_i) x_i^{(k)}, 1 \leq k \leq m$$

写成矩阵形式即为第 6 行中的公式。

3. 学习率和梯度向量的临界值都是兼顾精度和运行速度得到的实验值。

2.3 决策树桩实现

2.3.1 算法伪代码

Algorithm 2: 决策树桩

输入: 训练数据集矩阵 $\boldsymbol{X}$，分类真实值向量 $\boldsymbol{Y}$，训练权重 $\boldsymbol{\beta}$

num = 30;
minError = ∞; // 初始化错误率
for $i \leftarrow 1$ ***to*** n **do**
  step = $\dfrac{\max(\boldsymbol{X}[,i]) - \min(\boldsymbol{X}[,i])}{\text{num}}$; // 计算寻找临界值的步长
  for $j \leftarrow -1$ ***to*** *num*+1 **do**
    $\hat{\boldsymbol{Y}} = \mathbf{1}, \boldsymbol{\eta} = \mathbf{1}$;
    $\theta = \min(\boldsymbol{X}[,i]) + j * \text{step}$; // 计算临界值
    for $\ell \in [\leq, \geq]$ **do**
      if $\ell == \leq$ **then**
        $\hat{\boldsymbol{Y}}\left[\boldsymbol{X}[,i] \leq \theta\right] = 0$;
      else
        $\hat{\boldsymbol{Y}}\left[\boldsymbol{X}[,i] \geq \theta\right] = 0$;
      $\boldsymbol{\eta}[\hat{\boldsymbol{Y}} == \boldsymbol{Y}] = 0$;
      error = $\boldsymbol{\beta}^T\boldsymbol{\eta}$; // 计算加权错误率
      // 若错误率更低，则更新分类结果
      if *error* < *minError* **then**
        $\hat{\boldsymbol{Y}}_{\text{last}} = \hat{\boldsymbol{Y}}$;
        minError = error;
      end;
    end;
  end;
end;

输出: 最终的分类结果 $\hat{\boldsymbol{Y}}_{\text{last}}$

2.3.2 算法说明

1. 该决策树桩训练出的分类模型形如 $\begin{cases} 1, & \boldsymbol{x}[i] > \theta \\ 0, & \boldsymbol{x}[i] \leq \theta \end{cases}$ 或 $\begin{cases} 1, & \boldsymbol{x}[i] < \theta \\ 0, & \boldsymbol{x}[i] \geq \theta \end{cases}$。代码中的三层循环，分别确定最优的数据特征，临界值以及不等号方向。

2.4 AdaBoost 算法实现

2.4.1 算法伪代码

Algorithm 3: AdaBoost 算法

输入: 训练数据集矩阵 $\boldsymbol{X}$，分类真实值向量 $\boldsymbol{Y}$，迭代次数 T

1 $\boldsymbol{\beta} \leftarrow {}^{1}/_{n}$;
2 **for** $t \leftarrow 1$ ***to*** T **do**
3 $\quad h_t \leftarrow \Gamma(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{\beta})$; // 此处 Γ 指对数几率或决策树桩
4 $\quad \varepsilon_t = \sum_{i=1}^{n} \beta_i \mathbb{I}(y_i \neq h_t(\boldsymbol{x}_i))$;
5 $\quad$ **if** $\varepsilon_t > 0.5$ **then**
6 $\quad\quad$ break; // 若基分类器比随机猜测还差则中止算法
7 $\quad$ **end**;
8 $\quad \alpha_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}$; // 计算 h_t 的权重
9 $\quad$ **for** $i \leftarrow 1$ ***to*** n **do**
10 $\quad\quad \boldsymbol{\beta}_i = \boldsymbol{\beta}_i \exp\left[\alpha_t \mathbb{I}\big(y_i \neq h_t(\boldsymbol{x}_i)\big)\right]$; // 更新权重
11 $\quad$ **end**;
12 $\quad \boldsymbol{\beta} \leftarrow {}^{\beta}/_{\sum \beta_i}$; // 归一化权重
13 **end**;
14 $\hat{\boldsymbol{Y}} \leftarrow \sum_{i=1}^{n} \alpha_t h_t(\boldsymbol{x}_i)$; // 获取预测值的符号
15 $\hat{\boldsymbol{Y}}[\hat{\boldsymbol{Y}} \leq 0] = 0, \hat{\boldsymbol{Y}}[\hat{\boldsymbol{Y}} > 0] = 1$;

输出: 最终的分类结果向量 $\hat{\boldsymbol{Y}}$

2.4.2 算法说明

1. 如第 5 行所示，AdaBoost 算法每一轮训练都要检查错误率是否优于随机猜测，所以算法有可能在未达到初始设置的学习轮数时便停止，导致最终集成时只包含很少的基学习器而性能不佳。查阅相关资料后发现，可以采用“重采样法”（训练时用重采样的样本集进行训练）避免这一问题，但由于时间问题，本实验中未采用这种方法。

2. 虽然该算法在训练弱分类器时每一次都是贪心地获取局部最佳弱分类器，但是却不能确保选择出来加权后的是整体最佳。

3 实验结果分析

3.1 交叉验证结果

对提供的数据进行测评，得到如下表1所示的数据。

表 1: 交叉验证分类准确率

	1	5	10	100
logistic regression	0.9005	0.9003	0.8970	0.8981
decision stump	0.6443	0.8277	0.8321	0.8853

从表中的数据来看，我对 AdaBoost 算法有了以下新的认识：

- AdaBoost 作为分类器时，分类的精度比较高。两种经过训练的模型，最终的交叉验证准确率都在 90% 左右。
- AdaBoost 的训练错误率上界，随着迭代次数的增加，会逐渐（呈指数）下降，这一点在决策树桩上可以明显看出，数据恰好呈一个上升的趋势。
- AdaBoost 算法的训练时间比较长。算法实现过程中，从加权错误率向预期值逐渐逼近来构造级联分类器，迭代训练生成大量的弱分类器后才能实现这一构造过程，所以训练的时间比较长。
- AdaBoost 算法不容易出现过拟合问题，但不是绝对的。当弱学习器的复杂度很大或训练数据含有较大的噪声时，可能会发生过拟合。

关于基分类器类型、超参数设置对最终模型性能的影响，我发现：

- 在 AdaBoost 的框架下，可以使用各种回归分类模型来构建弱学习器，最终得到的模型拟合效果都比较好，非常灵活。
- 最终准确率和基分类器有关系。决策树准确率稳步上升，而对数几率回归准确率稳定，不符合上述认识，可能是因为基分类器的分类准确率已经很高了（本实验中对数几率回归不是一个弱分类器），后续产生了过拟合。
- 模型的性能受所选超参数值的影响很大。对数几率模型中的主要超参数是学习率和迭代次数，但在该模型中迭代停止条件是梯度的模长足够小，学习率可以设定为固定值，不要过大或过小即可。AdaBoost 算法的主要超参数是基分类器数目，一般来讲，基分类器数目越大，准确率会越高。

为了充分验证代码的正确性，选用 Breast Cancer Wisconsin (Diagnostic) Data Set 再次对模型进行验证，得到的交叉验证准确率结果如表 2 所示。从这张表中得到的结论基本与上面的分析相同。

表 2: *Wisconsin Dataset* 分类准确率

	1	5	10	100
logistic regression	0.9714	0.9696	0.9571	0.9571
decision stump	0.8768	0.9321	0.9375	0.9768

3.2 不同学习率对结果的影响

对对数几率回归的基分类器进行训练时（各个案例权重项相等），我们探索了损失函数值与学习率的关系，其中损失函数由下式给出：

$$\text{loss} = -\sum_{i=1}^{n} [y \log \hat{y} + (1-y) \log(1-\hat{y})]$$

图 1 是一个相对比较合适的学习率取值（$\alpha = 0.01$），可以看到此时损失值快速收敛至 750 附近。

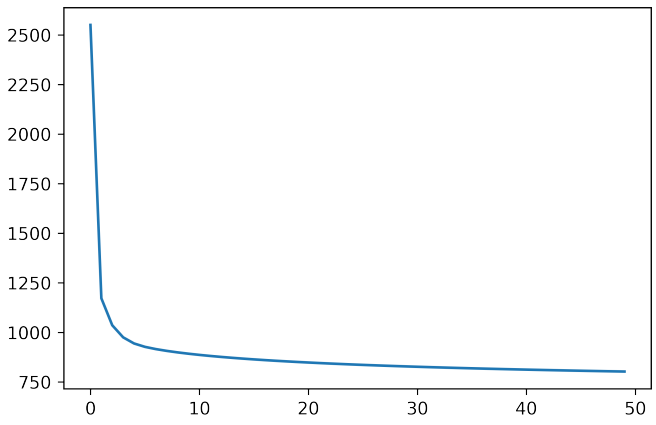

图 1: $\alpha = 0.01$

学习率越低，损失函数的变化速度就越慢，收敛过程将变得十分缓慢。而当学习率设置的过大时，梯度可能会在最值附近震荡，甚至可能无法收敛。图 2 便是学习率偏大的案例，损失函数值在来回震荡而且并未收敛。图 3 则是学习率偏小的例子，经过 50 次迭代之后，损失函数值仍未收敛到理想值（750）附近。

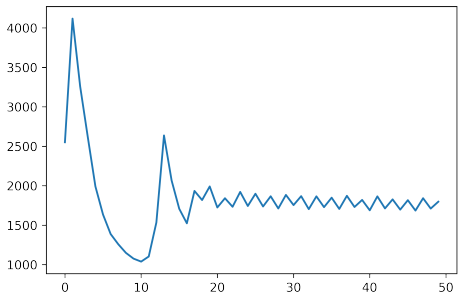

图 2: $\alpha = 0.1$

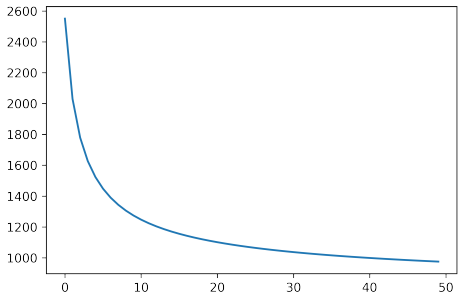

图 3: $\alpha = 0.001$

4 实验环境与平台

- 操作系统: windows 10
- 调试软件: VSCode 1.68.1
- 代码运行环境:
  - python 3.10.3
  - pandas 1.4.1
  - numpy 1.22.3

5 个人体会

对于《机器学习》这门课来说，理论学习和代码学习都很重要。理论学习过程比较枯燥，且不像技术工具那样可以立即付诸实践，但深入学习下来，对了解机器学习的重要思想、深刻理解算法是很有帮助的。在代码实现的时候，也会遇到很多许多细节上的问题，最终基本上都通过不断地调试以及查阅资料解决了。

建议课程增设习题课、讨论课等，便于大家相互交流，可以提高学习的效率。