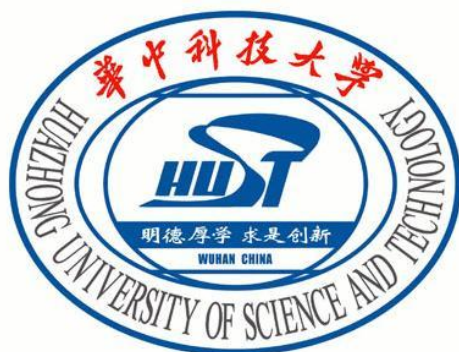


# 华中科技大学

## 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： CS2101

学    号： U202115325

姓    名： 宁毓伟

成    绩：

指导教师： 张腾

完成日期： 2023 年 4 月 18 日

## 目录

1	实验要求	3
2	算法设计与实现	3
2.1	Adaboost 算法	3
2.1.1	Adaboost 算法实现	3
2.2	分类器需要暴露的接口	4
2.3	DecisionTreeStump 实现	5
2.3.1	初始化权重以及错误率	5
2.3.2	<i>DecisionTreeStump</i> :: <i>fit</i> ( $\mathcal{S}, \mathcal{D}$ ) 接口实现	5
2.3.3	<i>DecisionTreeStump</i> :: <i>predict</i> ( $\mathbf{x}$ ) 接口实现	7
2.4	LogisticRegression 实现	7
2.4.1	初始化权重以及错误率	8
2.4.2	<i>LogisticRegression</i> :: <i>fit</i> ( $\mathcal{S}, \mathcal{D}$ ) 接口实现	8
2.4.3	<i>LogisticRegression</i> :: <i>predict</i> ( $\mathbf{x}$ ) 接口实现	10
3	实验环境与平台	10
4	结果与分析	11
4.1	测试数据说明	11
4.2	交折验证的方法	11
4.3	基于 DecisionTreeStump 分类器的 Adaboost 算法结果	12
4.4	基于 LogisitcRegression 分类器的 Adaboost 算法结果	13
4.5	结果分析	15
4.5.1	以 DecisionTreeStump 作为基分类器的结果	15
4.5.2	以 LogisticRegression 作为分类算法的结果	15
4.5.3	LogisticRegression 作为分类器表现差的原因	15
5	个人体会	16

---

5.1 我对 Adaboost 算法的新认识 . . . . .	16
5.2 基分类器类型，超参数设置对模型性能的影响 . . . . .	16

## 1 实验要求

- 实现决策树桩 (Decision Tree Stump) 算法。
- 实现逻辑回归 (Logistic Regression) 算法。
- 基于上述两个分类器实现自适应增强 (Adaptive Boost, Adaboost) 算法, 使得该算法可以选择基分类器 (Base Classifier) 的类型以及基分类器的数目。
- 实现上述算法时不使用任何机器学习模型库。

在本实验中, 仅模型使用 numpy 完成。

- 将给定的数据集 (Data Set) 随机划分成 10 个子集, 使用其中一个子集作为验证集 (Validation Set), 并使用 Adaboost 算法对模型进行训练和交叉验证。
- 将实现的 Adaboost 算法封装成 Python 脚本, 并使用命令行参数暴露接口, 使得算法可以指定基分类器的类别, 新的测试数据的目录以及输出文件的目录。

## 2 算法设计与实现

### 2.1 Adaboost 算法

Adaboost 算法是一种集成学习方法, 它通过组合多个弱分类器来构建一个强分类器。在每次迭代中, Adaboost 算法会根据上一次迭代的结果调整样本权重, 使得上一次分类错误的样本权重变大, 而分类正确的样本权重变小。这样, Adaboost 算法能够更加关注那些难以分类的样本, 从而提高分类的准确率。

Adaboost 算法的基本思想是将多个弱分类器组合成一个强分类器。在每次迭代中, Adaboost 算法会根据上一次迭代的结果调整样本权重, 使得上一次分类错误的样本权重变大, 而分类正确的样本权重变小。这样, Adaboost 算法能够更加关注那些难以分类的样本, 从而提高分类的准确率。

#### 2.1.1 Adaboost 算法实现

首先初始化一个向量  $\mathcal{D}_1 = [\frac{1}{m}]_{1 \times m}$  作为最初的样本权重。接着进行  $\mathcal{T}$  轮迭代。在每一轮迭代中以样本权重  $\mathcal{D}_i$  计算加权训练分类器  $h_i$  并计算加权错误率  $\epsilon$ , 接着使用公式  $\alpha_i \leftarrow \frac{1}{2} \ln \frac{1-\epsilon}{\epsilon}$  计算, 接着使用公式  $\mathcal{D}_{i+1} \leftarrow \mathcal{D}_i \odot \exp(-\alpha_i \mathbb{I}(h_i(\mathbf{x}) = y))$  计算新的样本权重, 最后对权重进行归一化。迭代结束后算法返回  $\text{sign}(\sum_{t \in [\mathcal{T}]} \alpha_t h_t)$  作为最终的分类器。

详细的 Adaboost 算法介绍参考文献: [1]。

伪代码见算法1。

**Algorithm 1:** Adaboost 算法

---

**输入:** 训练数据集  $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i \in [m]}$ , 基分类器算法  $\mathcal{L}$ , 基分类器数目 (迭代次数)  $\mathcal{T}$

```

1  $\mathcal{D}_1 \leftarrow [\frac{1}{m}]_{1 \times m}$ ; // 初始化权重
2 for  $t \leftarrow 1$  to  $\mathcal{T}$  do
3    $h_t \leftarrow \mathcal{L}(\mathcal{S}, \mathcal{D}_t)$ ; // 以样本权重  $\mathcal{D}_t$  训练  $h_t$ 
4    $\epsilon \leftarrow \mathcal{D}_t \odot \mathbb{I}(h_t(\mathbf{x}) \neq y)$ ; // 计算训练集的加权错误率
5   if  $\epsilon > 0.5$  then break;
6    $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon}{\epsilon}$ 
7    $\mathcal{D}_{t+1} \leftarrow \mathcal{D}_t \odot \exp(-\alpha \mathbb{I}(h_t(\mathbf{x}) = y))$ 
8    $s \leftarrow \sum \mathcal{D}_t$ 
9    $\mathcal{D}_t \leftarrow \frac{\mathcal{D}_t}{s}$ ; // 权重归一化
10 end
输出:  $\text{sign}(\sum_{t \in [\mathcal{T}]} \alpha_t h_t)$ 

```

---

算法具体实现下图:

```

sample_weight = np.ones(X.shape[0]) / X.shape[0] #? 初始化权重
for i in tqdm.tqdm(range(self._n_iter)):
    clf = self._build_clf()
    y_pred = clf.fit(X, y, sample_weight, return_pred=True) #? 以样本权重 D_t 训练 h_t
    err = clf.error_rate #? 计算训练集的加权错误率
    if err > 0.5:
        break
    alpha = 0.5 * np.log((1-err) / (err + 1e-19))
    sample_weight = sample_weight * np.exp(-1.0 * alpha * y * y_pred)
    sample_weight /= sample_weight.sum() #? 权重归一化

    self._alpha.append(alpha)
    self._classifier.append(clf) #? 保存权重

```

Fig. 1: Adaboost 算法实现

## 2.2 分类器需要暴露的接口

### 1. 用于训练的接口 $\text{fit}(\mathcal{S}, \mathcal{D}_t)$

$\text{fit}$  接口用于实现算法伪代码中的步骤3: 实现以样本权重  $\mathcal{D}_t$  训练  $h_t$ 。

*DecisionTreeStump* 的  $\text{fit}$  接口实现见: 2.3.2。

*LogisticRegression* 的  $\text{fit}$  接口实现见: 2.4.2。

### 2. 用于预测的接口 $\text{predict}(\mathbf{x})$

$\text{predict}$  接口用于实现算法伪代码中的步骤4: 计算训练集的加权错误率。

*DecisionTreeStump* 的  $\text{predict}$  接口实现见: 2.3.3。

*LogisticRegression* 的  $\text{predict}$  接口实现见: 2.4.3。

3. 用于获取训练集加权错误率的接口 `@property : error_rate`

`error_rate` 接口用于实现算法伪代码中的步骤4：计算训练集的加权错误率。

4. 用于获取分类器权重的接口 `@property : weight`

`weight` 接口用于实现算法伪代码中的输出部分10，作为 Adaboost 算法模型的权重输出。

## 2.3 DecisionTreeStump 实现

*DecisionTreeStump* 算法，也称单层决策树，是一种简单的决策树。它仅仅是基于单个特征来做决策。由于这棵树只有一次分裂过程，因此它实际上仅仅是一个树桩。

决策树桩详细内容见参考文献:[2]。

### 2.3.1 初始化权重以及错误率

*DecisionTreeStump* 模型的权重由以下几个部分组成：

- $critical\_dimension \in \{0 \dots dimension(x_i) - 1\}$

在决策树算法中，我们会依次选择训练集的一个特征维度作为进一步划分的依据。由于 *DecisionTreeStump* 是单层的决策树，因此仅会进行一次划分，这单次划分依据的特征维度就是 *critical\_dimension*。

- $thredhold \in \mathbb{R}$

*thredhold* 即阈值，作为所选择的特征的分界线。确定训练集的 *critical\_dimension* 后，使用该特征作为划分正负类样本的依据。根据样本 *critical\_dimension* 的值是否小于 *thredhold*，将样本分成两类。

- $compare\_method \in \{0, 1\}$

*compare\_method* 即比较方法，用于划分正负类样本。

- 当  $compare\_method = 0$  时， $value(critical\_dimension) < thredhold$  的样本将会被划分成负类；
- 当  $compare\_method = 1$  时， $value(critical\_dimension) \geq thredhold$  的样本将会被划分成负类。

在开始训练前，权重将会被初始化为 *None*，加权错误率 *err* 将会被初始化为 1。

### 2.3.2 *DecisionTreeStump* :: *fit*(*S*, *D*) 接口实现

对训练集的特征维度进行遍历，目的是找到最好的特征维度作为 *critical\_dimension*。在每一次遍历的过程中，在特征维度的最小值和最大值之间均匀取点作为 *thredhold*。接着对于  $compare\_method \in \{0, 1\}$  的两种取值，计算当前的加权错误率 *err*。

执行完上述步骤后选取加权错误率  $err$  最小的特征维度、阈值以及比较方法分别作为  $critical\_dimension$ ,  $thredhold$  以及  $compare\_method$ 。

伪代码见算法2

---

**Algorithm 2:** DecisionTreeStump::fit

---

**输入:** 训练数据集  $S = \{(\mathbf{x}_i, y_i)\}_{i \in [m]}$ , 样本权重  $\mathcal{D}$

```

1  steps  $\leftarrow$  200 ;                                // 选取 thredhold 的个数
2  for d  $\leftarrow$  0 to dimension( $x_i$ ) - 1 do
3      max  $\leftarrow$  特征维度 d 上的最大值
4      min  $\leftarrow$  特征维度 d 上的最小值
5      step_size  $\leftarrow$   $\frac{max-min}{steps}$ 
6      for step  $\leftarrow$  0 to steps do
7          thredhold  $\leftarrow$  min + step  $\cdot$  step_size ;           // 当前遍历的阈值
8          for method in {0,1} do
9               $\hat{y} \leftarrow predict_{method, thredhold, d}(\mathbf{x})$ 
10             err  $\leftarrow$   $\langle \mathbb{I}(h_t(\mathbf{x}) \neq y), \mathcal{D} \rangle$  ;           // 计算加权误差
11             记录 err 最小的 d、thredhold 以及 method ;           // 权重保存
12         end
13     end
14 end
```

**输出:**  $err$  最小的  $d$ 、 $thredhold$  以及  $method$

---

具体代码见下图:

```

num_steps = 200
for dim in range(X.shape[1]):
    dim_min = np.min(X[:, dim])
    step_size = (np.max(X[:, dim]) - dim_min) / num_steps

    for step in range(num_steps + 1):
        thredhold = dim_min + step * step_size           #? 当前遍历的遍历的阈值
        for compare_method in [0, 1]:
            y_pred = self.predict(X, compare_method, thredhold, dim)
            err = np.inner((y_pred != y), sample_weight) #? 计算加权误差
            if err < self._min_err:                       #? 权重保存
                self._min_err = err
                self._thredhold = thredhold
                self._compare_method = compare_method
                self._critical_dimension = dim
```

Fig. 2: DecisionTreeStump::fit 实现

### 2.3.3 *DecisionTreeStump* :: *predict*(**x**) 接口实现

根据给定的 *critical\_dimension*、*thredhold* 以及 *compare\_method* 划分 **x**。

具体而言，当 *compare\_method* = 0 时，将 *critical\_dimension* 的值小于 *thredhold* 的样本划分成负类，将 *critical\_dimension* 的值大于等于 *thredhold* 的样本划分成正类；当 *compare\_method* = 1 时，将 *critical\_dimension* 的值小于 *thredhold* 的样本划分成正类，将 *critical\_dimension* 的值大于等于 *thredhold* 的样本划分成负类。

伪代码见算法3

---

#### Algorithm 3: *DecisionTreeStump*:*predict*

---

输入: 数据集的特征部分  $\mathbf{X}_{m \times n}$

```

1  $\hat{\mathbf{y}} \leftarrow [0 \dots 0]_{1 \times m}$ ; //  $\hat{y}$  作为预测值
2 for  $i \leftarrow 0$  to  $m - 1$  do
3    $d \leftarrow \text{critical\_dimension}$ 
4   if  $\mathbf{X}_{i,d} < \text{thredhold}$  then
5     if  $\text{compare\_method} = 0$  then  $\hat{y}_i \leftarrow -1$  else  $\hat{y}_i \leftarrow 1$ ;
6   else
7     if  $\text{compare\_method} = 0$  then  $\hat{y}_i \leftarrow 1$  else  $\hat{y}_i \leftarrow -1$ ;
8   end
9 end

```

输出:  $\hat{\mathbf{y}}$

---

具体代码见下图:

```

y = np.ones(X.shape[0])
if compare_method == 0:
    y[np.argwhere(X[:, dim] < thredhold)] = -1.0
else:
    y[np.argwhere(X[:, dim] >= thredhold)] = -1.0
return y

```

Fig. 3: *DecisionTreeStump*::*predict* 实现

## 2.4 LogisticRegression 实现

Logistic Regression (逻辑回归) [3] 是一种用于解决二分类问题的机器学习方法，用于估计某种事物的可能性。它是一种广义线性模型，假设因变量  $y$  服从伯努利分布。与线性回归不同的是，逻辑回归使用了 sigmoid 函数，将线性回归的结果从  $(-\infty, \infty)$  映射到  $(0, 1)$ 。

逻辑回归的数学表达式见等式2



$$f(x) = \sigma(\theta \cdot x + bias) \quad (1)$$

其中  $\sigma$  为 sigmoid 函数:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (2)$$

### 2.4.1 初始化权重以及错误率

*LogisticRegression* 模型的权重由以下几个部分组成:

- $\theta_{1 \times n+1}$

$\theta$  的维度比数据集中的  $\mathbf{x}$  的维度要多 1。对于  $\mathbf{x} = (x_1, x_2 \dots x_n)$  以及  $\theta = (\theta_1, \theta_2 \dots \theta_n)$ , 我们引入  $\tilde{x} = (1, x_1, x_2 \dots x_n)$  以及  $\tilde{\theta} = (bias, \theta_1, \theta_2 \dots \theta_n)$

如此以来, 等式2将化简为等式3

$$f(x) = \sigma(\tilde{\theta} \cdot \tilde{x}) \quad (3)$$

这将在我们使用代码实现逻辑回的时候提供足够的遍便利。

- $mean_{1 \times n}$

$mean$  是  $\mathbf{X}_{m \times n}$  在第一个维度上的均值, 用于对数据集进行正则化处理。注意, 在计算  $mean$  的过程中, 使用的是训练集的特征进行计算。在对验证集进行预测的时候, 使用训练集特征计算出来的  $mena$  进行正则化。这样做保证了模型中不含有验证集的任何特征。

- $std_{1 \times n}$

$std$  是  $\mathbf{X}_{m \times n}$  在第一个维度上的标准差, 用于对数据集进行正则化处理。

在开始训练前, 权重将会被初始化为 *None*, 加权错误率 *err* 将会被初始化为 1。

### 2.4.2 *LogisticRegression* :: *fit*(*S*, *D*) 接口实现

对于给定的数据集  $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i \in [m]}$  以及样本权重  $\mathcal{D}$ , 定义损失函数为

$$l_i = -y_i \ln y_i - (\hat{y}_i - y_i) \ln (\hat{y}_i - y_i) \quad (4)$$

$$Loss \leftarrow \sum_{i \in [m]} \mathcal{D}_i \cdot l_i \quad (5)$$

其中  $\hat{y}$  是真实值  $y$ 。

在训练的过程中进行多次迭代, 每一次迭代将会对损失函数进行求导, 并通过梯度下降的方式更新权重。损失函数的梯度为:

$$\nabla_w L(w) = \sum_{i=1}^m \mathcal{D}_i (\sigma(\theta^T x^{(i)}) - y^{(i)}) x^{(i)} \quad (6)$$

其中,  $m$  是样本数量,  $\sigma(z)$  是 sigmoid 函数,  $\theta$  是模型参数,  $x^{(i)}$  是第  $i$  个样本的特征向量,  $\mathcal{D}_i$  是第  $i$  个样本的权重,  $y^{(i)}$  是第  $i$  个样本的标签。

伪代码见算法4

---

**Algorithm 4:** LogisticRegression:fit
 

---

**输入:** 训练数据集  $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}_{i \in [m]}$ , 样本权重  $\mathcal{D}$

```

1  $iter \leftarrow 1000$ ; // 迭代的次数
2  $\theta \leftarrow (1 \dots 1)_{1 \times n}$ ; // 初始化权重
3  $\eta \leftarrow 0.001$ ; // 初始化学习率
4  $mean \leftarrow \frac{\sum_{i \in [m]} \mathbf{x}_i}{m}$ ; // 计算均值
5  $std \leftarrow \sqrt{\frac{1}{m} \sum_{i \in [m]} (\mathbf{x}_i - mean)^2}$ ; // 计算标准差
6  $\mathbf{x}_i \leftarrow (\mathbf{x}_i - mean) / std$  for  $i \in [m]$ ; // 正则化处理
7 for  $i \leftarrow 0$  to  $iter$  do
8    $\hat{y} \leftarrow \text{sigmoid}(\theta \mathbf{X})$ 
9    $l_i \leftarrow -y_i \ln y_i - (\hat{y}_i - y_i) \ln (\hat{y}_i - y_i)$  for  $i \in [m]$ 
10   $Loss \leftarrow \sum_{i \in [m]} \mathcal{D}_i \cdot l_i$ ; // 计算加权损失
11   $\theta \leftarrow \theta - \eta \cdot \frac{\partial Loss}{\partial \theta}$ 
12 end
输出:  $\theta, mean, std$ 

```

---

具体代码见下图:

```

self._X_mean = X.mean(axis=0)
self._X_std = X.std(axis=0)
X = (X - self._X_mean) / self._X_std    #? 正则化处理
X = np.hstack([np.ones((X.shape[0], 1)), X])
for _ in range(max_iter):
    g = self._grad(X, y, sample_weight) #? 计算梯度
    self._theta -= lr * g

```

Fig. 4: LogisticRegression::fit 实现

梯度的计算实现见下图:

```

y_delta = self._sigmoid(X.dot(self._theta)) - y
return (sample_weight * y_delta).dot(X)

```

Fig. 5: LogisticRegression::fit 实现

### 2.4.3 *LogisticRegression* :: *predict*(**x**) 接口实现

根据给定的 *theta*、*mean* 以及 *std* 将 **x** 划分正负类别。

具体而言，本实验根据给定的模型参数 *theta*、*mean* 以及 *std* 计算出  $\hat{y} \in (0, 1)$ （逻辑回归的预测值）。接着将样本中预测值小于 0.5 的部分划分成负类，将样本中预测值大于等于 0.5 的部分划分成正类。

伪代码见算法5

---

**Algorithm 5:** *DecisionTreeStump*:*predict*


---

**输入:** 数据集的特征部分  $\mathbf{X}_{m \times n}$

```

1  $\mathbf{x}_i \leftarrow (\mathbf{x}_i - \text{mean}) / \text{std}$  for  $i \in [m]$  ; // 正则化处理
2  $\hat{y} \leftarrow \text{sigmoid}(\theta \mathbf{X})$ 
3 for  $i \leftarrow 1$  to  $m$  do
4   if  $\hat{y}_i < 0.5$  then
5      $\hat{y}_i \leftarrow -1$ 
6   else
7      $\hat{y}_i \leftarrow 1$ 
8   end
9 end
输出:  $\hat{y}$ 

```

---

具体代码见下图:

```

if not train:           #? 训练过程中会在fit中正则化处理
    X = (X - self._X_mean) / self._X_std
    X = np.hstack([np.ones((X.shape[0], 1)), X])

y = self._sigmoid(X.dot(self._theta))
y[np.argwhere(y < 0.5)] = -1.0
y[np.argwhere(y >= 0.5)] = 1.0
return y

```

Fig. 6: *LogisticRegression*::*predict* 实现

## 3 实验环境与平台

- OS: Manjaro Linux x86\_64
- Kernel: 5.15.106-1-MANJARO
- CPU: AMD Ryzen 7 5800H with Radeon Graphics (16) @ 3.200GHz
- Memory: 13832MiB
- Python 3.9.13 with numpy 1.21.5

## 4 结果与分析

### 4.1 测试数据说明

数据集  $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}$ ，其中特征部分为  $\mathbf{X}_{3679 \times 57}$ ，标签部分为  $\mathbf{y}_{1 \times 57} \in \{0, 1\}$ 。

特征部分  $\mathbf{X}$  在各个分量上的均值见下图

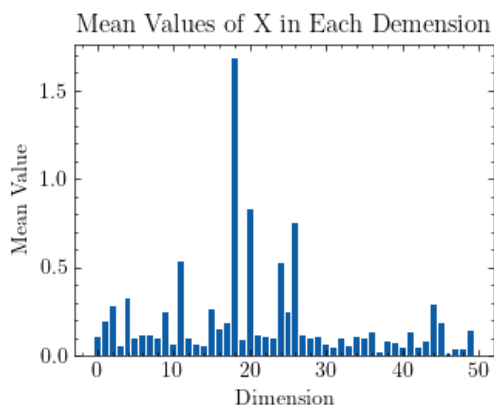


Fig. 7: 前 50 个维度

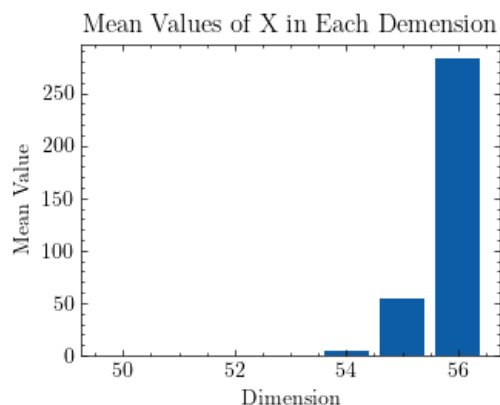


Fig. 8: 50 个之后的维度

不难发现第 55、56 个维度上的平均值要远远大于其他维度，进行在逻辑回归算法4中进行维度正则化将有利于模型的训练和收敛。

### 4.2 交折验证的方法

在本次实验中采用了 10 交折验证的方法。具体而言，实验中将会首先打乱数据集，接着按照打乱后的循序依次选取 10 个子集作为验证集。

验证集与训练集划分的样例如下图所示：



Fig. 9: Cross Validation

在本实验中，对于 10-交折验证的每一个 fold 都分别，根据实验要求分别在基分类器数量（迭代轮数）为  $\mathcal{T} \in \{1, 5, 10, 100\}$  的条件下对两种基分类器进行测试。为了便于展示结果，本实验将对  $\mathcal{T} \in \{1, 2 \dots 100\}$  的测试结果用图像进行展示，对实验要求的 5 个迭代轮次展示具体的准确率。

### 4.3 基于 DecsionTreeStump 分类器的 Adaboost 算法结果

$Fold \in \{1, 2, 3, 4, 5\}$  的准确率  $Accuracy$  随着迭代轮数  $\mathcal{T}$  的变化如下图所示

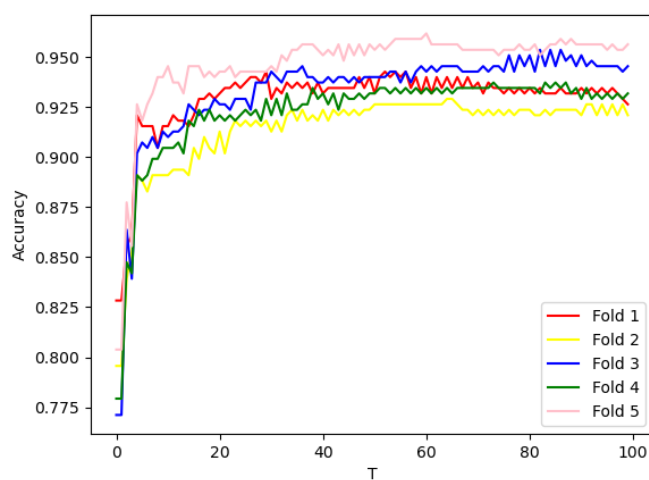


Fig. 10: 准确率

$Fold \in \{6, 7, 8, 9, 10\}$  的准确率  $Accuracy$  随着迭代轮数  $T$  的变化如下图所示

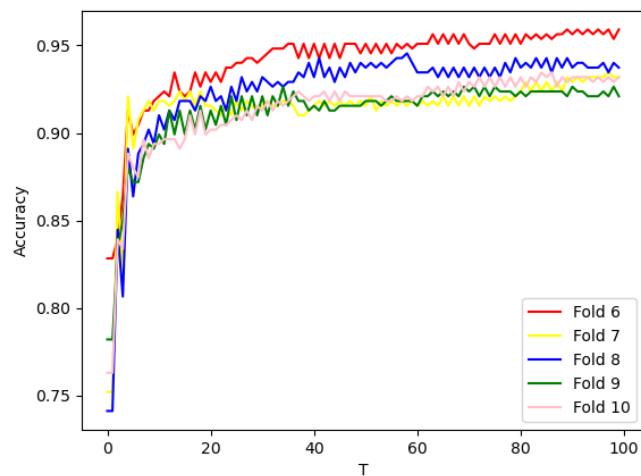


Fig. 11: 准确率

$Fold \in \{1, 2 \dots 10\}$  的准确率  $Accuracy$  与迭代轮数  $T \in \{1, 5, 10, 100\}$  的关系如下表所示：

Tab. 1: 准确率

	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6	Fold7	Fold8	Fold9	Fold10
T=1	82.83%	79.56%	77.11%	77.93%	80.38%	82.83%	75.20%	74.11%	78.20%	76.02%
T=5	92.10%	89.10%	90.19%	89.10%	92.64%	91.28%	92.10%	89.10%	88.01%	88.56%
T=10	91.55%	89.10%	91.28%	90.46%	94.01%	91.83%	91.28%	89.37%	89.10%	89.10%
T=100	92.64%	92.10%	94.55%	93.19%	95.64%	95.91%	93.19%	93.73%	92.10%	92.92%

通过上面的表格可以发现，迭代轮数（基分类器数目） $T$  越大，验证集的准确率也就越高。

#### 4.4 基于 LogisiticRegression 分类器的 Adaboost 算法结果

$Fold \in \{1, 2, 3, 4, 5\}$  的准确率  $Accuracy$  随着迭代轮数  $T$  的变化如下图所示：

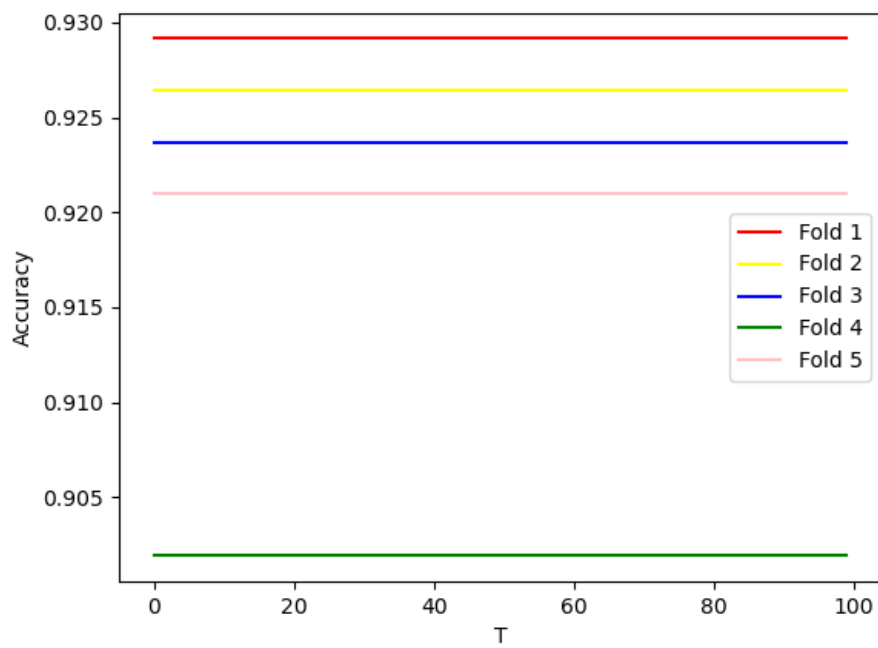


Fig. 12: 准确率

$Fold \in \{6, 7, 8, 9, 10\}$  的准确率  $Accuracy$  随着迭代轮数  $T$  的变化如下图所示:

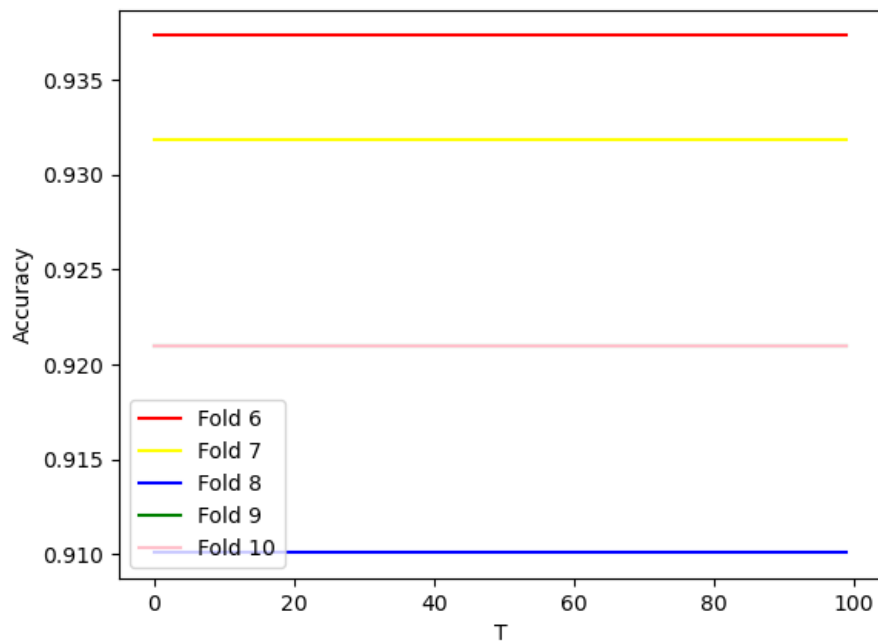


Fig. 13: 准确率

$Fold \in \{1, 2 \dots 10\}$  的准确率  $Accuracy$  与迭代轮数  $\mathcal{T} \in \{1, 5, 10, 100\}$  的关系如下表所示:

Tab. 2: 准确率

	Fold1	Fold2	Fold3	Fold4	Fold5	Fold6	Fold7	Fold8	Fold9	Fold10
T=1	92.92%	92.92%	92.37%	90.19%	92.10%	93.73%	93.19%	91.01%	92.10%	92.10%
T=5	92.92%	92.92%	92.37%	90.19%	92.10%	93.73%	93.19%	91.01%	92.10%	92.10%
T=10	92.92%	92.92%	92.37%	90.19%	92.10%	93.73%	93.19%	91.01%	92.10%	92.10%
T=100	92.92%	92.92%	92.37%	90.19%	92.10%	93.73%	93.19%	91.01%	92.10%	92.10%

## 4.5 结果分析

### 4.5.1 以 DecisionTreeStump 作为基分类器的结果

通过观察测试结果可以发现, 随着迭代轮数 (分类器数目)  $\mathcal{T}$  的增加, 对于以 DecisionTreeStump 算法作为分类器的 Adaboost 算法而言, 准确率逐渐上升。

### 4.5.2 以 LogisticRegression 作为分类算法的结果

使用 LogisticRegression 作为分类算法时, 准确率并不会随着  $\mathcal{T}$  的增加发生变化。在本实验中观察训练过程可以发现, 使用 LogisticRegression 作为分类器的 Adaboost 算法并不能完整的运行完成, 而会因为算法1中的第5行错误率  $err > 0.5$  而退出。因此随着迭代轮数的增加, 并不会有更多数目的 LogisticRegression 分类器被训练成型, 算法每次仅会用几个固定数目的 LogisticRegression 分类器进行预测, 因此预测的结果不会随着  $\mathcal{T}$  的增加发生变化。

### 4.5.3 LogisticRegression 作为分类器表现差的原因

使用 LogisticRegression 作为分类算法与 Adaboost 算法进行结合的做法从理论角度看并不可行。这是因为 LogisticRegression 作为分类算法时使用的损失函数 (Loss Function) 如下所示:

$$l_i = -y_i \ln y_i - (\hat{y}_i - y_i) \ln (\hat{y}_i - y_i) \quad (7)$$

$$Loss \leftarrow \sum_{i \in [m]} \mathcal{D}_i \cdot l_i \quad (8)$$

其中  $\hat{y}$  是真实值  $y$ 。

而 Adaboost 算法使用的损失函数是指数损失:

$$l_i = \exp(-y_i \hat{y}_i) \quad (9)$$

在运行 LogisticRegression 算法时, 会假设损失函数为公式7以及公式8, 并以此为基础对权重求梯度。然而 Adaboost 算法执行的过程中使用的损失函数为公式9, 这导致分类器所求梯度并不是损失函数的梯度, 因此分类器的权重不会被正确更新, 并且错误率逐步积累, 最终导致算法停止。



## 5 个人体会

### 5.1 我对 Adaboost 算法的新认识

#### 1. Adaboost 算法的基本思想

Adaboost 算法的基本思想是通过训练多个弱分类器，并通过弱分类器线性拟合的方式合成一个强分类器。本实验使用的拟合方式如下：

$$H_T = \text{sign}\left(\sum_{t=0}^T \alpha h_t\right) \quad (10)$$

其中  $H_T$  是 Adaboost 算法合成的强分类器， $h_t$  是迭代轮数为  $t$  时训练的弱分类器。

#### 2. Adaboost 算法提供的是算法框架

Adaboost 算法提供的是算法框架，可以使用各种类型基分类器进行训练。但在本实验中，LogisticRegression 作为基分类算法的效果并不如 DecisionTreeStump，这是因为在计算的过程中 LogisticRegression 分类器权重的梯度并没有被正确求解，导致准确率并不会随着  $T$  的增加而上升。具体原因见：4.5.3。

### 5.2 基分类器类型，超参数设置对模型性能的影响

#### 1. 基分类器类型的影响

不同的基分类器对模型的效果有着不同的影响，在本实验中以 DecisionTreeStump 作为基分类器要比 LogisticRegression 好，具体结果见：4.5。

#### 2. 迭代轮数 $T$ 的影响

随着迭代轮数  $T$  的增加，以 DecisionTreeStump 算法作为基分类器的 Adaboost 算法模型的准确率逐渐上升。准确率达到一定程度之后就不再上升了。具体见：4.3。

## 参考文献

- [1] Robert E. Schapire, Yoav Freund, Peter Barlett, and Wee Sun Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. In Proceedings of the 14th International Conference on Machine Learning, pages 322–330, Nashville, TN, 1997.
- [2] Iba, Wayne; Langley, Pat (1992). Induction of One-Level Decision Trees.ML92: Proceedings of the Ninth International Conference on Machine Learning, Aberdeen, Scotland, 1–3 July 1992. Morgan Kaufmann. pp. 233–240.
- [3] Tolles, Juliana; Meurer, William J (2016). Logistic Regression Relating Patient Characteristics to Outcomes.