人工智能实验

感知机和逻辑回归

第 3 次实验

姓名：陈家豪

班级：2018 级计科 1 班

学号：18308013

目录

1 实验任务 1：感知机（PLA）

1.1 算法原理

1.1.1 基本公式

PLA 算法是一种经典的机器学习算法，其主要用于二分类问题上。其输入为特征向量 $\mathbf{X} \in \mathrm{R}^n$，输出的是样本类别 $y \in \{-1, +1\}$。

PLA 算法的实质，是在一个样本数据集上，寻找一个超平面，使得其能正确划分所有的样本点。假设我们有 n 个样本，第 i 个样本拥有特征向量 $\mathbf{X}_i = \begin{bmatrix} X_{i1} & X_{i2} & \cdots & X_{im} \end{bmatrix}^{\mathrm{T}}$，该样本的类别为 y_i，则我们需要寻找一个权值参数向量 $\mathbf{W} = \begin{bmatrix} W_1 & W_2 & \cdots & W_m \end{bmatrix}^{\mathrm{T}}$，使得

$$h(\mathbf{X}) = \mathrm{sign}(\mathbf{W}^T\mathbf{X} + b) \tag{1}$$

当 $\mathbf{X} = \mathbf{X}_i$ 时，$h(\mathbf{X}_i) = y_i$。其中，sign() 是符号函数，b 是偏置参数。

我们可以令 $\widetilde{\mathbf{X}}_i = \begin{bmatrix} 1 & X_{i1} & X_{i2} & \cdots & X_{im} \end{bmatrix}^{\mathrm{T}}$，$\widetilde{\mathbf{W}} = \begin{bmatrix} W_0 & W_1 & W_2 & \cdots & W_m \end{bmatrix}^{\mathrm{T}}$，则式1可以被简化为：

$$h(\widetilde{\mathbf{X}}) = \mathrm{sign}(\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}}) \tag{2}$$

1.1.2 迭代过程和算法步骤

既然我们知道了 PLA 算法的基本公式，接下来的任务就是寻找一个合适的参数向量 $\widetilde{\mathbf{W}}$，使得预测函数的准确率尽可能高。

在探讨如何获得 $\widetilde{\mathbf{W}}$ 前，我们需要知道向量点积的几何含义：如果 $\vec{u} \cdot \vec{v}$ 为正数，则向量 $\vec{u}$ 和 $\vec{v}$ 所成的角是锐角；如果是负数，则所成角是钝角。那么，对于一个误分类点，有以下两种出错情况：

(1) $\widetilde{\mathbf{W}}$ 和 $\widetilde{\mathbf{X}}$ 本该是锐角（$y = 1$，即理论上 $\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}} > 0$），但实际上成了钝角（$y = -1$，即实际上 $\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}} < 0$）；

(2) $\widetilde{\mathbf{W}}$ 和 $\widetilde{\mathbf{X}}$ 本该是钝角（$y = -1$，即理论上 $\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}} < 0$），但实际上成了锐角（$y = 1$，即实际上 $\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}} > 0$）；

对于出错情况 (1)，我们需要让 $\widetilde{\mathbf{W}}$ 和 $\widetilde{\mathbf{X}}$ 远一点，形成一个钝角；对于出错情况 (2)，我们需要让 $\widetilde{\mathbf{W}}$ 和 $\widetilde{\mathbf{X}}$ 靠近一点，形成一个锐角。那么如何让两个向量之间的夹角变大/变小呢？我们知道，假设两个向量 $\vec{u}$ 和 $\vec{v}$，那么 $\vec{u} + \vec{v}$ 相比 $\vec{u}$ 更靠近 $\vec{v}$（即夹角更小）；那么 $\vec{u} - \vec{v}$ 相比 $\vec{u}$ 更远离 $\vec{v}$（即夹角更大）。

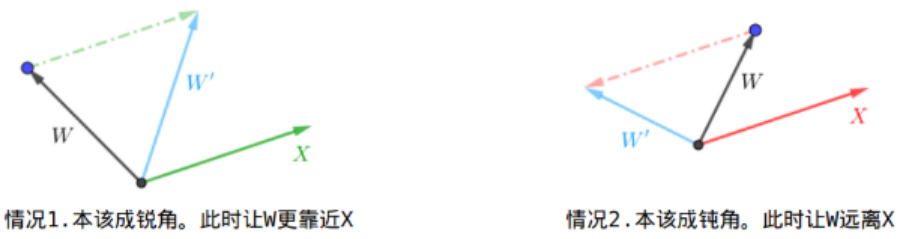

图 1: 两种错误的处理方法

更进一步，我们可以设置学习率 η，那么对于一个误分类点设其特征向量为 $\widetilde{\mathbf{X}}_i$，所属类别为 y_i，我们用式3对 $\widetilde{\mathbf{W}}$ 进行更新：

$$\widetilde{\mathbf{W}} = \widetilde{\mathbf{W}} + \eta y_i \widetilde{\mathbf{X}}_i \tag{3}$$

显然，更新若干次后，$\widetilde{\mathbf{W}}$ 和 $\widetilde{\mathbf{X}}_i$ 之间可以形成一个锐角

基于上述方法，我们可以得到 PLA 算法的算法步骤：

(1) 随机初始化 $\widetilde{\mathbf{W}}$，设置学习率 η；

(2) 对所有样本进行分类，选取其中一个误分类点 $(\widetilde{\mathbf{X}}_i, y_i)$，对参数进行更新：$\widetilde{\mathbf{W}} = \widetilde{\mathbf{W}} + \eta y_i \widetilde{\mathbf{X}}_i$，直到该误分类点被正确分类为止；

(3) 选取下一个误分类点，重复步骤 (2)。直到训练集中没有误分类点为止。

事实上，只有当样本数据集是线性可分的，我们才能找到一个 $\widetilde{\mathbf{W}}$ 使得训练集中没有误分类点。如果样本数据集是线性不可分的，我们可以设置最大迭代次数，或者寻找一个 $\widetilde{\mathbf{W}}$ 使得预测准确率最高。

1.1.3 损失函数

同样，PLA 算法也有其损失函数。对于一个误分类点 (x_i, y_i)，其到超平面的距离为：

$$d = \frac{1}{\|\widetilde{\mathbf{W}}\|} \|\widetilde{\mathbf{W}}^T \widetilde{\mathbf{X}}\| = -\frac{1}{\|\widetilde{\mathbf{W}}\|} y_i \widetilde{\mathbf{W}}^T \widetilde{\mathbf{X}} \tag{4}$$

假设误分类点集合为 M，则我们可以不考虑 $\frac{1}{\|\widetilde{\mathbf{W}}\|}$，得到损失函数：

$$L(\widetilde{\mathbf{W}}) = -\sum_{x_i \in M} y_i (\widetilde{\mathbf{W}}^T \widetilde{\mathbf{X}}) \tag{5}$$

1.2 伪代码/流程图

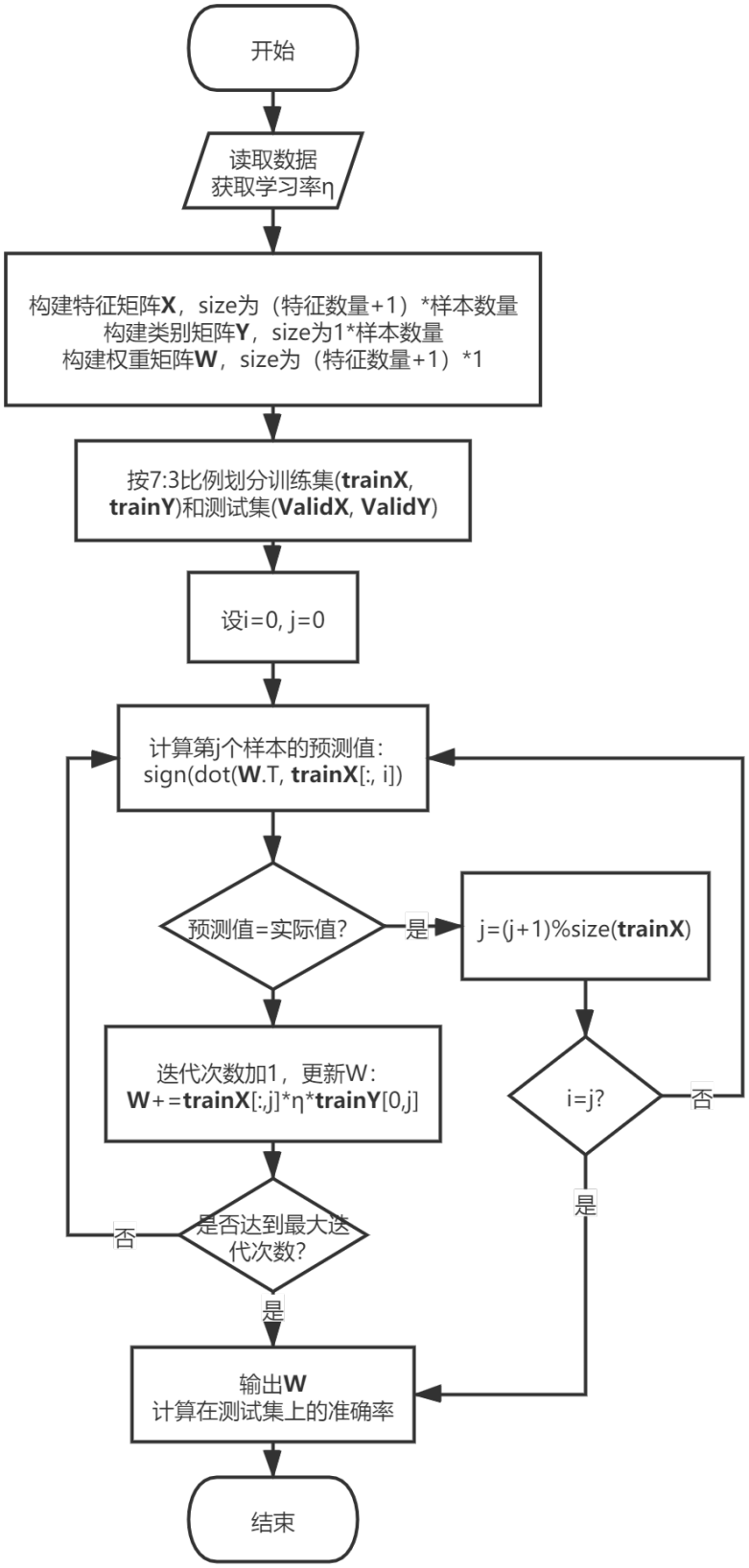

图 2: PLA 算法流程图

1.3 关键代码

PLA 算法中用于计算更新权重矩阵 **W** 的代码如下所示：

```
def PLA(X, Y, W, Eta, iteration_num):
"""
PLA算法，计算权重矩阵W
"""
startIndex=0          # 起始索引
tempIndex=startIndex# 当前索引
iterNO=0              # 当前迭代次数
while iterNO<iteration_num:
if np.sign(np.dot(W.T,X[:,tempIndex:tempIndex+1]))==Y[:,tempIndex:tempIndex+1]:
# 如果第tempIndex个样本预测值与实际值一致
    tempIndex=(tempIndex+1)%np.size(Y)# 索引喜加一
    if tempIndex==startIndex:          # 如果所有样本的预测值和实际值均一致
        break                          # 结束迭代
else:
# 如果第tempIndex个样本预测值与实际值不一致
    W+=X[:,tempIndex:tempIndex+1]*(Eta*Y[0,tempIndex])# 更新权重矩阵
    iterNO+=1                                          # 迭代次数加一
    startIndex=tempIndex
    #if iterNO%20==0:
        #print('#'+str(iterNO)+': done')
#返回迭代后的权重矩阵
return W
```

其中输入参数 X 是特征矩阵，size 为（特征数量 +1, 样本数量）；Y 是类别矩阵，size 为（样本数量,1）；W 是权重矩阵，size 为（特征数量 +1,1）；Eta 即为学习率 η；$iteration\_num$ 为最大迭代次数。该函数最终输出更新后的权重矩阵 W。

PLA 算法中另一部分关键代码就是计算准确率和对数据集预测的代码，如下所示：

```
def calculate_accuracy(X,Y,W):
"""
计算当前权重W的准确率
"""
# 计算预测值
result=np.sign(np.dot(W.T,X))
# 统计结果
correct=0
for i in range(np.size(Y)):
if result[0,i]==Y[0,i]:
    correct+=1
# 返回准确率
return correct/np.size(Y)
```

```
def predict(W,X):
"""
预测数据集X的值
```

```
    """
    # 计算预测值
    return np.sign(np.dot(W.T,X))
```

1.4 实验过程与结果分析

1.4.1 数据集的划分

与上次实验一致，本次实验我依然按照 7:3 的比例将原始数据集划分为训练集和验证集。数据集共有 8000 个样本，前 5600 个样本为训练集，其余样本为验证集。

1.4.2 PLA 算法结果分析

我们设置初始权重向量值全为 0，分别选取学习率 $\eta = 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75$ 和 2.0。因为训练集样本数为 5600，为保证所有样本均至少被遍历一次，设置最大迭代次数为 15000，每隔 25 次迭代计算一次在验证集上的准确率。将所有数据可视化，结果如图3所示。

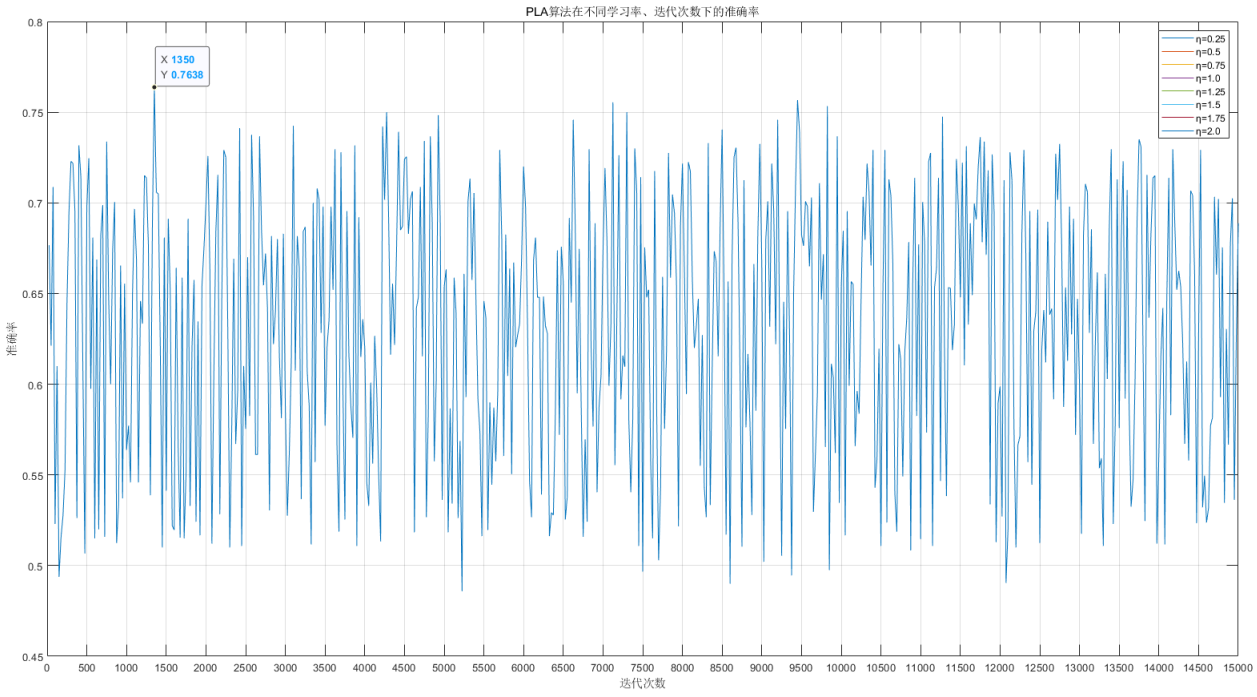

图 3: PLA 算法在初始权重为 0，不同学习率、迭代次数下的准确率

从图中可见，准确率随着迭代次数变化波动较大，没有明显规律。个人认为这是因为 PLA 算法在更新权重时只是针对单一样本进行更新，这就导致了模型在整体样本上的准确率会出现波动。在第 1350 次迭代时准确率达到最高，为 76.38%。

我们选取 $\eta = 1.0$，查看迭代 1350 次后对验证集的预测值，验证集前 10 个样本的预测值、准确值如表1所示：

序号	预测类别	真实类别
5601	-1	-1
5602	1	1
5603	1	-1
5604	-1	-1
5605	-1	1
5606	-1	-1
5607	-1	-1
5608	-1	-1
5609	1	1
5610	1	1

表 1: $\eta = 1.0$，迭代 1350 次后对验证集前 10 个样本的预测

可见，前 10 个样本中有 8 个预测正确。

此外，我们可以看到，在同一迭代次数下，无论学习率取值多少，准确率都没有任何变化……事实上，学习率取值不同，在同一迭代次数下准确率应该也会有所变化。个人认为这是一种偶然现象。我后来使用 random.seed(0) 设置随机种子为 0，初始权重设为相同的随机向量，再次计算准确率，将所有数据可视化后结果如图4所示，可见同一迭代次数下，学习率取值的不同会导致准确率有所不同。最高准确率出现在 $\eta = 1.5$ 时迭代 13550 次，准确率为 76.00%。

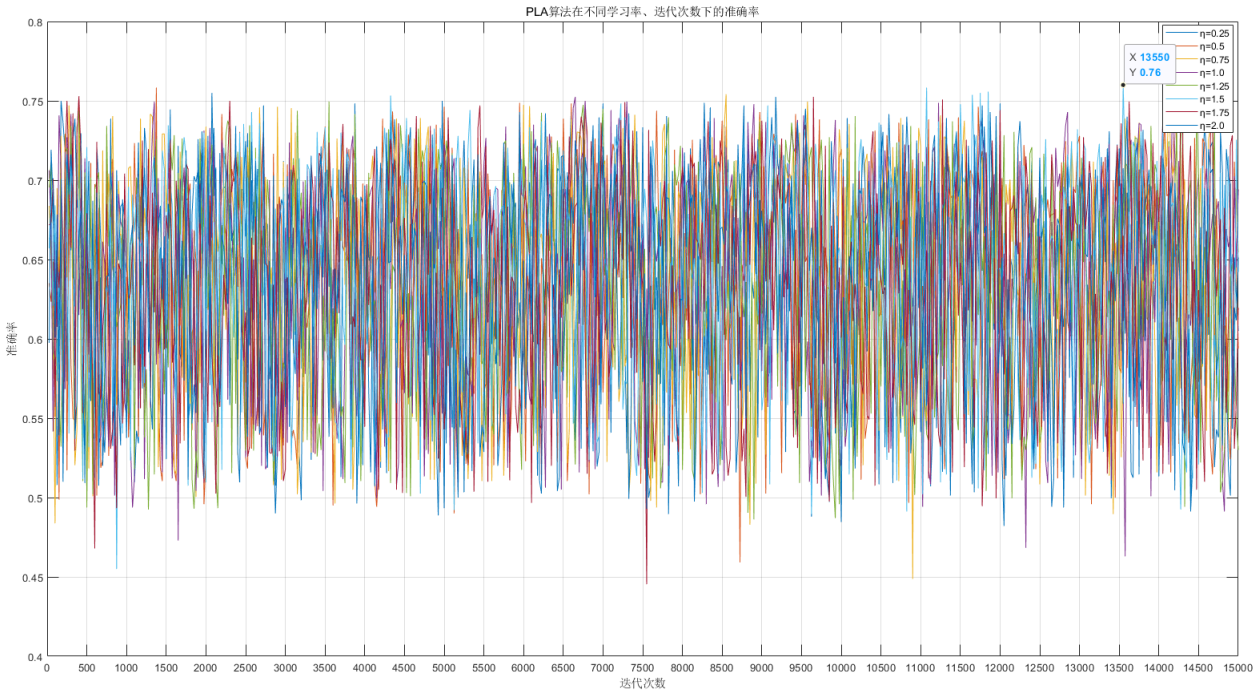

图 4: PLA 算法在初始权重为随机向量，不同学习率、迭代次数下的准确率

2 实验任务 2：逻辑回归（Logistics Regression）

2.1 算法原理

2.1.1 基本公式

逻辑回归是另一种分类模型，常用于二分类。其输入是样本的特征向量 $\mathbf{X} \in \mathrm{R}^n$，输出是样本属于某个类别 $y \in \{0, 1\}$ 的概率。

与线性回归分类不同的是，逻辑回归使用 Logistic 函数计算样本的类别概率。假设我们有 n 个样本，第 i 个样本拥有特征向量 $\mathbf{X}_i = \begin{bmatrix} X_{i1} & X_{i2} & \cdots & X_{im} \end{bmatrix}^{\mathrm{T}}$，该样本的类别为 y_i，权值参数向量为 $\mathbf{W} = \begin{bmatrix} W_1 & W_2 & \cdots & W_m \end{bmatrix}^{\mathrm{T}}$，则

$$p(y=1|\mathbf{X}) = \frac{e^{\mathbf{W}^T\mathbf{X}+b}}{1+e^{\mathbf{W}^T\mathbf{X}+b}} \tag{6}$$

$$p(y=0|\mathbf{X}) = \frac{1}{1+e^{\mathbf{W}^T\mathbf{X}+b}} \tag{7}$$

其中，$p(y=1|\mathbf{X})$ 是该样本为标签 1 的概率，$p(y=0|\mathbf{X})$ 是该样本为标签 0 的概率。

同样，我们可以令 $\widetilde{\mathbf{X}}_i = \begin{bmatrix} 1 & X_{i1} & X_{i2} & \cdots & X_{im} \end{bmatrix}^{\mathrm{T}}$，$\widetilde{\mathbf{W}} = \begin{bmatrix} W_0 & W_1 & W_2 & \cdots & W_m \end{bmatrix}^{\mathrm{T}}$，则可以被简化为：

$$p(y=1|\mathbf{X}) = \frac{e^{\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}}}}{1+e^{\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}}}} \tag{8}$$

$$p(y=0|\mathbf{X}) = \frac{1}{1+e^{\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}}}} \tag{9}$$

我们可以将这两个函数合并起来，得到似然函数 $f(\mathbf{X})$。如式10所示，某个样本 $\mathbf{X}$ 属于某个类别 y 的概率为：

$$f(\mathbf{X}) = p(y|\mathbf{X}) = (\frac{e^{\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}}}}{1+e^{\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}}}})^y(\frac{1}{1+e^{\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}}}})^{1-y} \tag{10}$$

令 $\pi(x) = \frac{e^{\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}}}}{1+e^{\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}}}}$。显然，$y=1$ 时 $f(\mathbf{X}) = p(y=1|\mathbf{X}) = \pi(x)$，$y=0$ 时 $f(\mathbf{X}) = p(y=0|\mathbf{X}) = 1-\pi(x)$。

基于整个训练集，我们可以得到整个样本集的似然函数：

$$\prod_{i=1}^{N}[\pi(x_i)]^{y_i}[1-\pi(x_i)]^{1-y_i} \tag{11}$$

对似然函数取对数，我们可以得到对数似然函数：

$$L(\widetilde{\mathbf{W}}) = \sum_{i=1}^{N}[y_i\log\pi(x_i) + (1-y_i)\log(1-\pi(x_i))] = \sum_{i=1}^{N}[y_i(\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}}) - \log(1+e^{\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}}})] \tag{12}$$

对 $L(\widetilde{\mathbf{W}})$ 求极大值，即可得到 $\widetilde{\mathbf{W}}$ 的估计值。故我们对 $L(\widetilde{\mathbf{W}})$ 取负作为逻辑回归模型的损失函数：

$$L(\widetilde{\mathbf{W}}) = -\sum_{i=1}^{N}[y_i(\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}}) - \log(1+e^{\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}}})] \tag{13}$$

则损失函数的梯度为

$$-\nabla_{\widetilde{\mathbf{W}}}L(\widetilde{\mathbf{W}}) = -\sum_{i=1}^{N}[y_i - \pi(x_i)]x_i \tag{14}$$

因此更新权重的函数为：

$$\widetilde{\mathbf{W}} = \widetilde{\mathbf{W}} + \eta \sum_{i=1}^{N} [y_i - \pi(x_i)] x_i \tag{15}$$

2.1.2 算法步骤

基于上述方法，我们可以得到 PLA 算法的算法步骤：

(1) 随机初始化 $\widetilde{\mathbf{W}}$，设置学习率 η；

(2) 计算当前梯度，并对参数 $\widetilde{\mathbf{W}}$ 进行更新；

(3) 重复步骤 (2)，直到满足一定的收敛条件。

2.2 伪代码/流程图

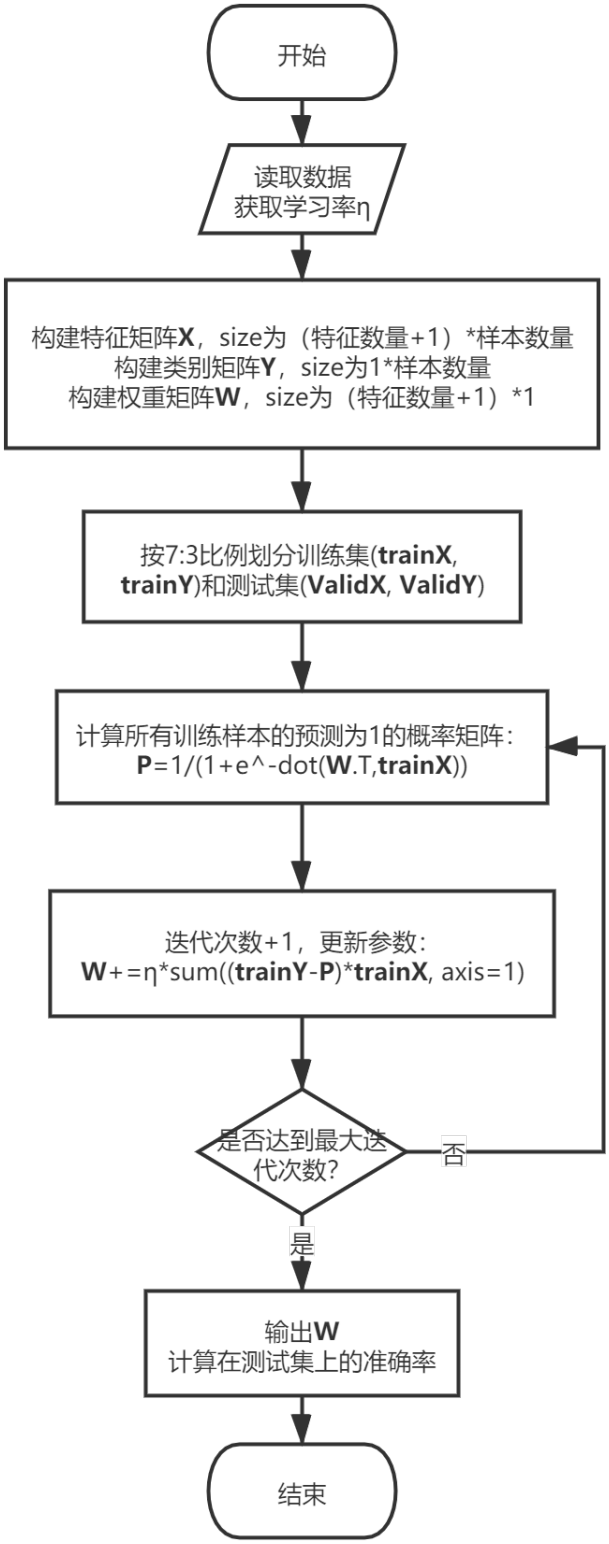

图 5: LR 算法流程图

2.3 关键代码

计算逻辑回归概率的函数代码如下所示：

```
def calculate_p(W,X):
"""
计算 (x)，即逻辑回归预测为1的概率
"""
temp=np.dot(W.T,X)
p=np.zeros_like(temp)
for i in range(np.size(temp)):
    # 根据e^(wx+b)的正负，采用不同方式计算p，避免出现数值溢出
    if temp[0,i]>0:
        p[0,i]=1/(1+math.exp(-temp[0,i]))
    else:
        p[0,i]=math.exp(temp[0,i])/(math.exp(temp[0,i])+1)
return p
```

同样，W 为权重矩阵，X 为样本的特征矩阵。因为在计算 $e^{\widetilde{\mathbf{W}}^T\tilde{\mathbf{x}}}$ 时，$\widetilde{\mathbf{W}}^T\widetilde{\mathbf{X}}$ 可能远大于/远小于 0，导致最后计算结果溢出。故我们需要区分正负值，使用不同的计算方式避免数值溢出。

逻辑回归中用于更新权重矩阵的函数代码如下所示：

```
def Logistics_regression(X,Y,W,Eta,iteration_num):
"""
逻辑回归，计算权重矩阵W
"""
for i in range(iteration_num):
    p=calculate_p(W,X)  # 计算 (x)
    temp=Y-p
    temp2=temp*X
    temp3=np.reshape(np.sum(temp2,axis=1),(np.size(W),1))
    W+=temp3*Eta          # 更新参数
# 返回更新后的参数
return W
```

此外，还有两个个关键代码，分别用于计算当前权重矩阵应用在数据集上的预测值和准确率，代码如下所示：

```
def predict(W,X,threshold):
"""
计算预测值
"""
# 计算预测值
regression=calculate_p(W,X)
total=np.size(regression)
for i in range(total):
    # 根据阈值预测类别
    if regression[0,i]>threshold:
        regression[0,i]=1
    else:
        regression[0,i]=0
return regression
```

```python
def calculate_accuracy(X,Y,W, threshold):
"""
计算当前权重W的准确率
"""
# 计算预测值
regression=calculate_p(W,X)
# 统计结果
correct=0
total=np.size(Y)
for i in range(total):
    # 根据阈值预测类别
    if regression[0,i]>threshold:
        regression[0,i]=1
    else:
        regression[0,i]=0
    # 判断是否预测正确
    if Y[0,i]==regression[0,i]:
        correct+=1
# 返回准确率
return correct/total
```

threshold 是阈值，一般设为 0.5。当概率大于 *threshold* 时预测为 1，否则为 0。最后统计正确预测的数量，返回准确率。

2.4 实验过程与结果分析

2.4.1 数据集的划分

我们依然按照 7:3 的比例将原始数据集划分为训练集和验证集。数据集共有 8000 个样本，前 5600 个样本为训练集，其余样本为验证集。

2.4.2 LR 算法结果分析

对于逻辑回归，我们同样设置初始向量值全为 0，分别选取学习率 $\eta = 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75$ 和 2.0。因为该算法是批梯度下降，故我们设置最大迭代次数为 3000，每隔 10 次迭代计算一次在验证集上的准确率。将结果可视化，结果如图6所示。

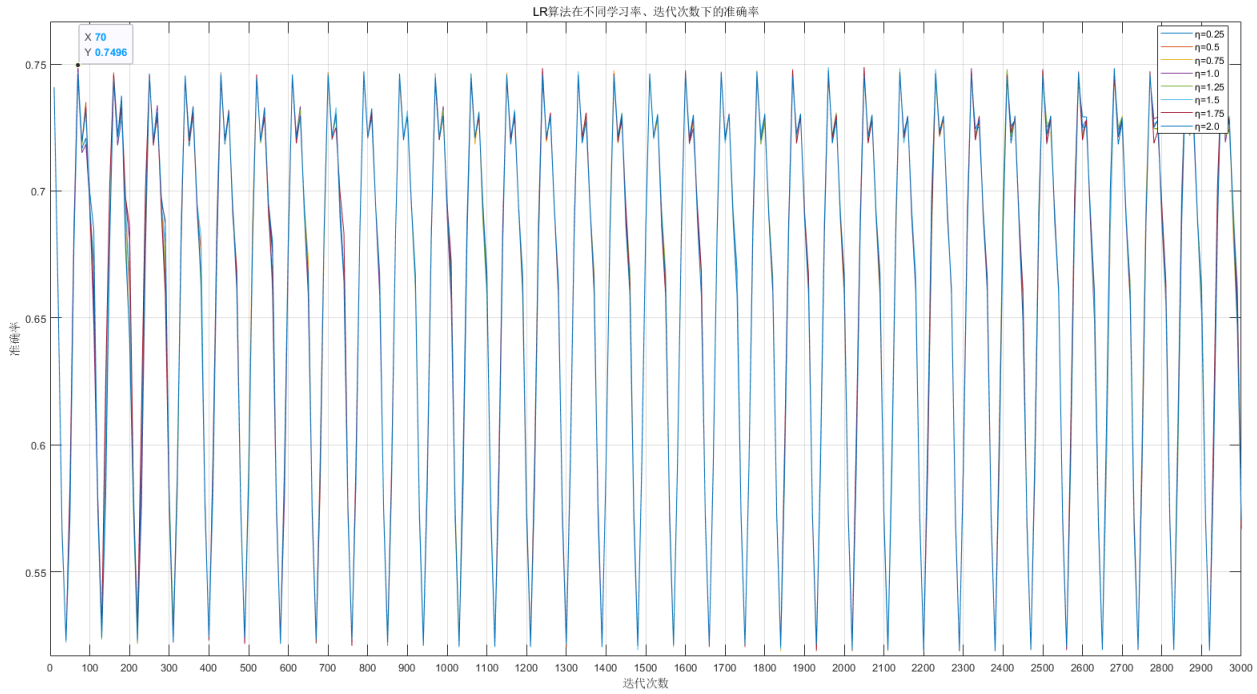

图 6: LR 算法在初始权重为 0，不同学习率、迭代次数下的准确率

该结果图像也很神奇，出现了类似周期函数的波动……最高准确率出现在 $\eta = 1.0$、迭代 70 次后，为 74.96%。个人猜测，这种现象出现的原因是因为权重值在局部最优点附近反复振荡导致的。

同样，我们选取 $\eta = 1.0$，查看迭代 70 次后对验证集的预测值，验证集前 10 个样本的预测值、准确值如表2所示：

序号	预测类别	真实类别
5601	0	0
5602	1	1
5603	1	0
5604	0	0
5605	0	1
5606	0	0
5607	0	0
5608	0	0
5609	0	1
5610	1	1

表 2: $\eta = 1.0$，迭代 70 次后对验证集前 10 个样本的预测

可见，前 10 个样本中有 7 个预测正确。

3 思考题

3.1 不同学习率对模型收敛的影响

学习率 η 是一种通过损失函数的梯度调整权重的超参数。η 越低，权重的变化速度就越慢。低学习率虽然可以保证模型最终收敛到一个局部极小值，但收敛速度会很慢，需要较长的时间才能完成收敛。

当 η 值越高时，权重的变化速度越快。在一定情况下，高学习率可以使得收敛速度极快，但由于过大的步长可能会导致局部极小值被错过，因此模型可能无法收敛。

3.2 使用梯度模长是否为零作为梯度下降的收敛终止条件是否合适，为什么？一般如何判断模型收敛？

不合适，因为在实际情况中，梯度模长几乎不可能为零，如果使用梯度模长作为梯度下降的收敛终止条件，那么该模型可能永远不会收敛。

在判断模型是否收敛时，我们可以选取以下条件：

(1) 选取一个较小的阈值，当梯度模长小于阈值时，即达到收敛终止条件，停止训练；

(2) 设定一个非常大的最大迭代次数，当迭代次数超过最大次数时就终止收敛，停止训练；

4 实验总结与感想

本次实验是人工智能实验的第三次实验，要求我们实现 PLA 算法和逻辑回归算法。本次实验的代码量并不大，但同样需要时间去消化、理解这两种经典机器学习算法的实现原理及机制。我在本次实验中遇到的最大的难题，是在实现逻辑回归算法时，计算概率时经常出现数值溢出。在参考了网上资料后，我采用两种不同的计算方式计算概率值，从而避免了溢出。

在此感谢老师和助教在实验过程中提供的帮助，希望我能顺利完成下一次实验。