人工智能lab3 实验报告

> 学号：
>
> 姓名：TRY
>
> 专业：计算机科学与技术

PLA感知机学习方法

一、算法原理

- PLA是针对二元分类问题，它可以用来解决二维或者高维的**线性可分**问题。输入是样本的特征向量 $x \in R^n$，输出是类别 $y \in \{+1, -1\}$。
- 感知机函数利用`sign`函数实现类别判定：
  $$f(x) = sign(w \cdot x + b)$$
  其中，`w`为n维的**权重向量**$w = (w_1, w_2, \ldots, w_n)$，`x`为某个样例的n维**特征向量** $x = (x_1, x_2, x_3 \ldots x_n)$，`b`为阈值。

  优化：将阈值融入到权重向量中，即将`b`设为 w_0，并对`x`增加第0维$x_0 = 1$，使得 $W = (w_0 = b, w_1, w_2, \ldots, w_n)$，$X = (+1, x_1, x_2, \ldots, x_n)$ 感知机函数改写如下：
  $$f(x) = sign(\sum_{j=0}^{n} w_j x_j) = sign(\vec{W}^T \vec{X})$$
  - *实际上，在此处也可以不做这个优化，直接用w和b也可。*

  定义：如果内积的结果大于0，就判为正例，输出`+1`；反之则属于反例，输出`-1`，这样就起到了二分类的效果。
- PLA是通过**损失函数**来作为优化标准的，损失函数是"所有误分类点到分离超平面的距离之和"，其和应尽可能小：
  $$L(w, b) = -\sum_{x_i \in M} y_i(w \cdot x_i + b),\ M为误分类点集合$$
- 通过随机梯度下降来对损失函数进行优化。损失函数的梯度如下：
  $$\nabla_w L(w, b) = -\sum_{x_i \in M} y_i x_i$$
  $$\nabla_b L(w, b) = -\sum_{x_i \in M} y_i$$
  选择一个误分类点(x_i, y_i)对参数进行更新，其中η表示学习率：
  $$w = w + \eta y_i x_i$$
  $$b = b + \eta y_i$$
- **划分数据集**：采用交叉验证法**k-fold**，具体原理已在实验2的报告中解释，此处省略。
- 因此，**PLA的算法步骤**是：

1. 先将权重向量和阈值合并，并给每一个样本特征向量前加一个`1`。
2. 随机初始化一个权重向量，一般初始化为全0向量。
3. 设置学习率，如设置为1。

4. 从头遍历所有的样本点，计算每个样本点的特征向量`w`和权重矩阵`x`的内积，判断结果是大于0还是小于0（使用符号函数`sign`来计算），得到预测结果。找到**第一个**预测错误的样本，通过 $W_{t+1} \leftarrow W_t + \eta y_i x_i$ **更新**权重向量，结束本轮遍历。
5. 重复步骤4，直到所有的训练样本都**预测正确（即结果收敛）**或**到达固定迭代次数**。
6. 训练结束后我们会得到一个权重向量，通过这个权重矩阵和要预测的x做内积来预测它的标签。
7. 如果要分析实验结果，则通过验证集统计标签预测正确的样例个数，计算准确率。

二、伪代码

```
Function PLA    /*这是没有将w和b合并起来的版本*/
Input: dataset, iteration, learnint_rate/*数据集，固定迭代次数，学习率*/
Output: w,b

    n := 训练集样本的维数(去掉label)
    w := 大小为n的全零数组
    b := 0
    i := 0
    dataset1 := dataset的numpy形式
    while i<iteration do
        flag := true
        for data in dataset1:
            x := data对应的特征向量
            temp := x*w +b
            if data对应的标签为0 then
                data标签转为-1  /*！！这里一定要转，否则无法更新w,b*/
            end if
            if sign(temp)不等于data标签值 then
                flag := False
                更新w,b
                break   /*记得跳出！！*/
            end if
        end for
        if flag=True then
            break   /*代表没有误判点，停止迭代*/
        end if
        i += 1
    end
    return w, b
```

三、代码

- PLA函数：需要将负例的label由0改为-1，否则无法正确更新w和b。同时，要记得找到第一个误判点完成更新后跳出循环。

```python
def PLA(dataset, iteration, learning_rate):
    """
    返回固定迭代次数中计算的w和b。
    :param dataset: 数据集
    :param iteration: 迭代次数
    :param learning_rate: 学习率
    :return:
    """
```

```python
    n = len(dataset[0]) - 1
    w = np.zeros(n)
    b = 0
    i = 0
    dataset1 = np.array(dataset)    # 在外面转换，提速！！
    while i < iteration:
        flag = True
        for data in dataset1:
            x = data[:-1]   # 左闭右开，即去除最后的label
            temp = x.dot(w) + b
            if data[-1] == 0:   # 这里一定要转换0位-1，否则无法更新w,b！！
                data[-1] = -1
            if np.sign(temp) != data[-1]:   # 出现误判点，注意-1在csv中为0！！！
                flag = False
                w = w + learning_rate * data[-1] * x
                b = b + learning_rate * data[-1]
                break
        if flag is True:    # 意味着此时没有误判点
            break
        i += 1
    return w, b
```

- 训练完成后，我们会得到一个权重向量`w`和阈值`b`，利用它们来对验证集进行验证， 计算验证集的准确率：

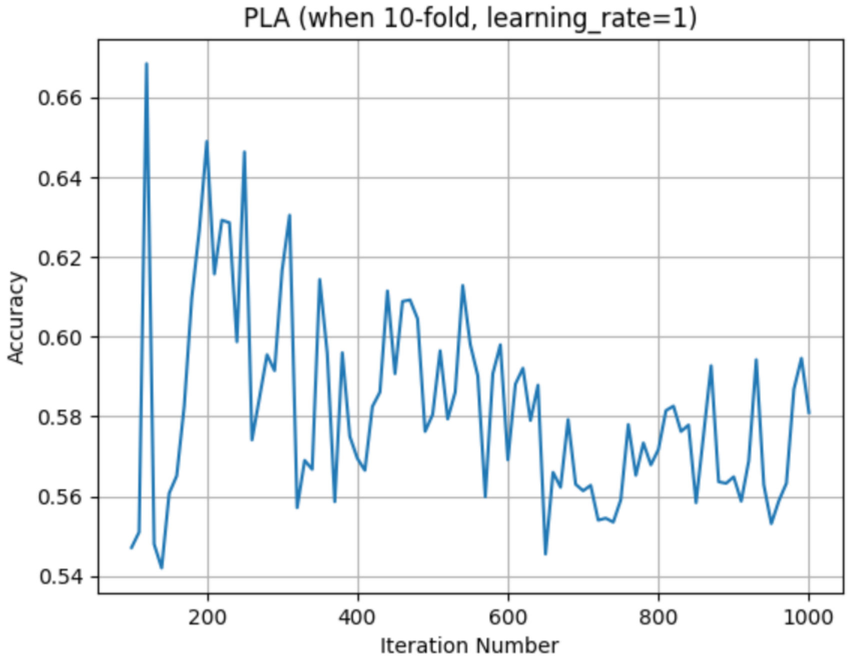
```python
def k_fold(dataset, k, i):
    """
    将数据集划分成训练集和验证集
    :param dataset:数据集
    :param k: 划分成k分
    :param i: 取第i份为验证集
    :return: 返回训练集和验证集
    """
    total = len(dataset)
    step = total // k   # 这样可以返回下取整：步长
    start1 = i * step
    end1 = start1 + step
    train_set = np.vstack((dataset[:start1], dataset[end1:]))   # vstack用于联结矩阵，要用2个括号
    valid_set = dataset[start1:end1]
    return train_set, valid_set

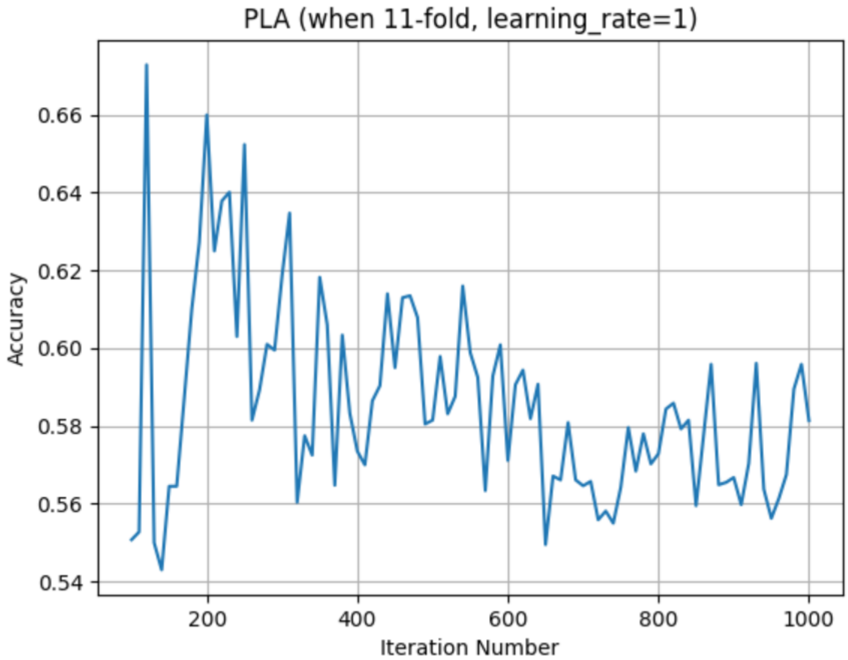

def validation(valid_set, w, b):
    """
    返回w,b下，验证集的准确率
    :param valid_set: 验证集
    :param w: 训练集得到的w
    :param b: 训练集得到的b
    :return: 返回准确率
    """
    total = len(valid_set)
    cnt = 0
    valid_set1 = np.array(valid_set)
    for data in valid_set1:
        x = data[:-1]  # 左闭右开，即去除最后的label
        temp = x.dot(w) + b
```

```
        if data[-1] == 0:  # 这里一定要转换0位-1，否则无法更新w,b！！
            data[-1] = -1
        if np.sign(temp) == data[-1] or temp == 0:   # temp=0也是预测正确的
            cnt += 1
    return cnt/total
```

四、实验结果以及分析

1. 结果展示和分析

- 

- 

- 以上为交叉验证k=10或11、学习率=1时，准确率随固定迭代次数的变化。由图像可知，当迭代次数在120左右时，准确率最高，接近0.67。且随着固定迭代次数的增加，准确率不断浮动，整体下降。
- 因此，可以看出数据集不是线性可分的，所以PLA找不到一个最优的解，每次迭代更新后准确率就会波动，可能上升也可能下降。 可见PLA对于线形不可分的数据集表现并不是很出色。

2. 模型性能展示和分析

- 在PLA实验过程中，由于每次都是找到第一个误判点就进行更新，所以运行速度较快（基本上k-fold是秒出），故进行了k值指标和固定迭代次数指标的优化。
- 以下比较基于**“k-fold交叉验证法”**划分数据集得到：

	k值	固定迭代次数	准确率
初始	10	100	54.71%
优化1	10	120	66.85%
优化2	11	120	67.28%
最优效果	11	120	67.28%

调参变量为k值（k-fold）和 固定迭代次数，判断标准为验证集上的准确率。

LR 逻辑回归

一、算法原理

- 分类模型有两类：**硬分类模型和软分类模型**。其中，**前者**是是用一个决策函数来直接判断样本的类别，如PLA、决策树；而**后者**是先算出每个类别的概率， 根据概率的大小来判断样本的类别，如逻辑回归。
- 逻辑回归通常针对**二分类**问题，输入是样本的特征向量$x \in R^n$，输出是样本属于某个类别$y \in \{0,1\}$的概率。
- 回归函数可以表示为

$$\pi(x) = \frac{1}{1+e^{-w\cdot x}}$$

  - $w = (w^T, b)^T, x = (x^T, 1)^T$.
  - 输入空间是$-\infty$到$+\infty$，而输出空间正好是概率的范围`[0,1]`
- 因此，**某个样本**`x`属于某个类别`y`的概率是：

$$f(x) = p(y|x) = \pi(x)^y(1-\pi(x))^{1-y}$$

  `f(x)`称为**似然函数.**

  考虑**整个数据集**的样本，有：

$$\text{似然函数} = \prod_{i=1}^{N} p(y|x_i) = \prod_{i=1}^{N} \pi(x_i)^{y_i}(1-\pi(x_i))^{1-y_i}$$

取对数，得：

$$\begin{aligned} L(w) &= \sum_{i=1}^{N}[y_i log\pi(x_i) + (1-y_i)log(1-\pi(x_i))] \\ &= \sum_{i=1}^{N}[y_i(w\cdot x_i) - log(1+e^{w\cdot x_i})] \end{aligned}$$

- 对$L(w)$取负，将$-L(w)$作为逻辑回归模型的**损失函数**，并使用**梯度下降法**对损失函数进行优化

  损失函数的**梯度**为：

$$
-\nabla_w L(w) = -\sum_{i=1}^{N}[y_i - \pi(x_i)]x_i
$$

  因此，**权重参数更新公式**为：

$$
w = w + \eta\sum_{i=1}^{N}[y_i - \pi(x_i)]x_i
$$

- **划分数据集**：
  - 采用交叉验证法**k-fold**，具体原理已在实验2的报告中解释，此处省略。
  - 也可以`random.shuffle`先对数据集进行洗牌，然后将固定大小的样本划分为验证集，其余为训练集。这样可以大大**提高速度**！
- 因此，**LR算法的步骤**如下：
  1. 先将权重向量和阈值合并，并给每一个样本特征向量前加一个`1`。
  2. 随机初始化一个权重向量`w`，一般初始化为全0向量。
  3. 设置学习率，如设置为1或0.00001。
  4. 计算当前梯度（通过`numpy`实现并行以提速），并对参数`w`进行更新
  5. 重复步骤4，直到所有的训练样本都**梯度收敛（小于阈值）**或**到达固定迭代次数。**
  6. 训练结束后会得到一个权重向量`w`，通过这个权重矩阵和要预测的`x`做内积来预测它的标签。
  7. 如果要分析实验结果，则通过验证集统计标签预测正确的样例个数，计算准确率。

二、伪代码

```
Function logistic    /*将w,b合起来作为权重向量*/
input: dataset, iteration, learning_rate/*数据集，固定迭代次数，学习率*/
output: w

    n := dataset中data的维数（包括label）
    w := 大小为n的全零numpy数组       /*初始化w*/
    i := 0
    diff := 1e-4     /*模型收敛的判据*/
    dataset1 := dataset的numpy形式
    while i<iteration do
        dataset_copy = dataset1
        dataset2 := dataset_copy的前n-1列 + 增广的“1”组成的矩阵
        pi := 利用dataset2得到pi(x)
        y := dataset的最后一列label值
        temp := y-pi
        将temp从列向量扩展成为矩阵，利用numpy并行化更新w为w_new
        diff1 := 计算w和w_new之间的二范数
        if diff1 <= diff then
            print("梯度下降已收敛")
            break
        end if
        w := w_new
        i += 1
    end
    return w
```

三、代码

- 以下呈现LR核心代码（其他代码与PLA相似，此处忽略）：

```
def logistic(dataset, iteration, learning_rate):
    n = len(dataset[0])
    w = np.zeros(n)
    w = w.reshape((-1, 1))      # 弄成列向量，否则后面会出错
    i = 0
    diff = 1e-4      # 模型收敛的判据
    dataset1 = np.array(dataset)
    while i < iteration:
        dataset_copy = dataset1
        dataset_copy = np.delete(dataset_copy, dataset_copy.shape[1] - 1,
axis=1)  # 删除最后一列
        temp_one = np.ones(dataset_copy.shape[0])
        dataset2 = np.insert(dataset_copy, dataset_copy.shape[1],
values=temp_one, axis=1)  # 插入一列1到最后一列
        pi = h(w, dataset2)      # 返回的是列向量（在列向量的基础上做numpy）
        y = dataset1[:, dataset.shape[1] - 1]    # 取出最后一列，这时是数组
        y = y.reshape((-1, 1))    # 转成列向量
        temp = y - pi
        temp = np.repeat(temp, dataset2.shape[1], axis=1)     # 沿着列扩展成二
维矩阵
        sum = np.sum(temp * dataset2, axis=0)    # 沿着行求和，即对x的各分量求和
        sum = sum.reshape((-1, 1))    # 转成列向量
        w_new = w + learning_rate * sum
        diff1 = np.linalg.norm(w_new - w)    # 二范数：求两者的欧式距离（倒数第一次
为真实值，倒数第二次为预测值）
        if diff1 <= diff:
            print("梯度下降已收敛")
            break
        w = w_new
        i += 1
    return w
```

- 计算LR的预测函数$\pi(x)$的函数如下：
  - 其中，如果学习率η取`1`的话会**溢出**。因此需要利用“数值计算”的相关知识，分成`if\else`语句来防止溢出：

```
def h(w, x):
    w = w.reshape((-1, 1))    # w原为数组，通过reshape(1,-1)转成行向量；
reshape(-1,1)转成列向量.而且要赋值回去！！
    temp = np.dot(x, w)     # numpy也有dot！！！
    '''
     # 防止溢出：
    for i, data in enumerate(temp):
        if data >= 0:
            temp[i] = 1.0 / (1 + np.exp(-1 * data))
        else:
            temp[i] = np.exp(data) / (1 + np.exp(data))
    '''
    temp = 1 / (1 + np.exp(-temp))
    return temp
```

- 通过`random.shuffle`划分数据集：

```
def split_data(dataset):
    random.shuffle(dataset)
    train_set = dataset[:7000]
    valid_set = dataset[7000:]
    return train_set, valid_set
```

四、创新点

1. 在LR中，我利用了`numpy`来并行化计算权值向量`w`的**更新**：

$$w = w + \eta \sum_{i=1}^{N} [y_i - \pi(x_i)] x_i$$

正常按照公式，应该计算出每个样本对应的$\pi(x_i)$，然后将每个样本的 $[y_i - \pi(x_i)]x_i$相加，再更新`w`。

而我在实验中，是通过`numpy`操作，就算出所有样本对应的$\pi(x_i)$（用列向量记录），并通过`numpy.repeat`操作扩展成为矩阵，再与x_i 进行矩阵乘法，并用`numpy.sum`进行求和，加速了程序的运行。

2. 在计算预测函数$\pi(x)$的函数中，如果设置学习率为1，会导致溢出。经过查资料，利用了"数值计算"中学习到的方法，设置了`if-else`语句进行**防止溢出**的操作：

```
# 防止溢出：
    for i, data in enumerate(temp):
        if data >= 0:
            temp[i] = 1.0 / (1 + np.exp(-1 * data))
        else:
            temp[i] = np.exp(data) / (1 + np.exp(data))
```

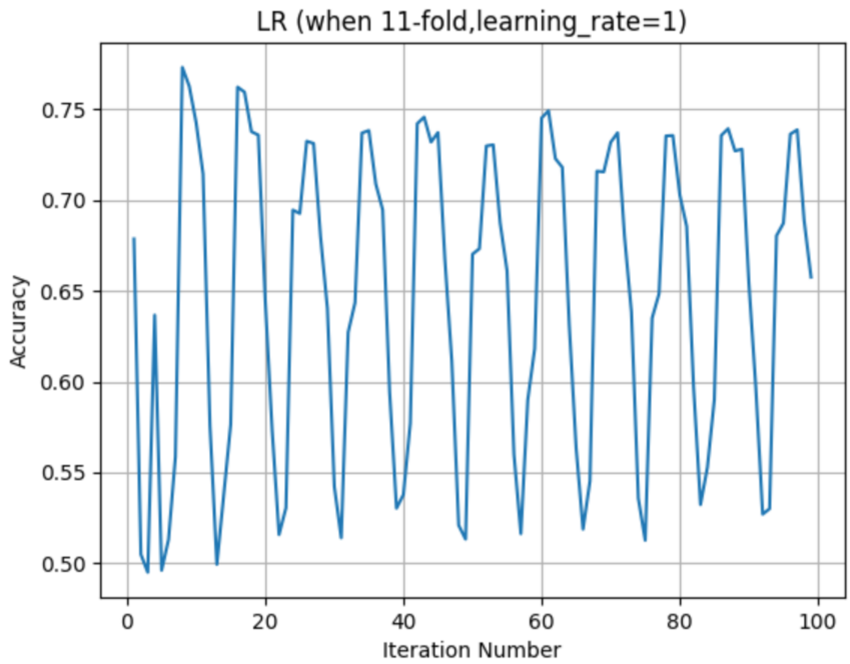
用于替换本来的`temp = 1 / (1 + np.exp(-temp))`。

3. 由于LR算法的原因，在使用交叉验证k-fold的时候运行速度较慢；因此，实验中使用了`random.shuffle`操作，先对数据集进行**洗牌**，同样起到了**"随机"**的效果，并且由于不需运行`k`次，大大**提高了运行速率**！并且，此时由于并不需要计算`k`次，在计算$\pi(x)$的时候**不会溢出**，可以使用不分类讨论的版本，同样提高运行速率。

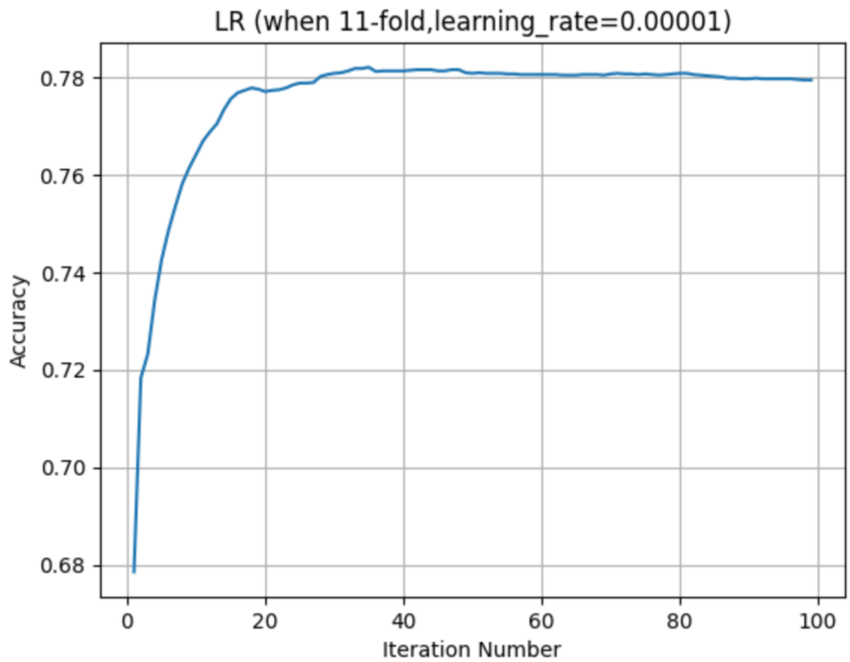
五、实验结果以及分析

1.结果展示和分析

-

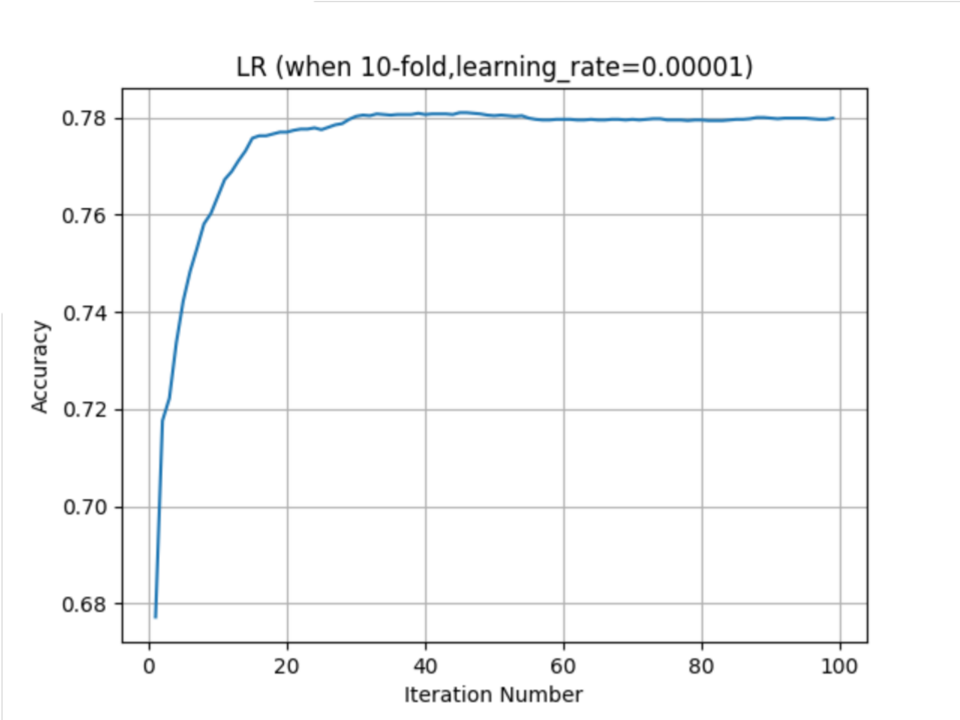


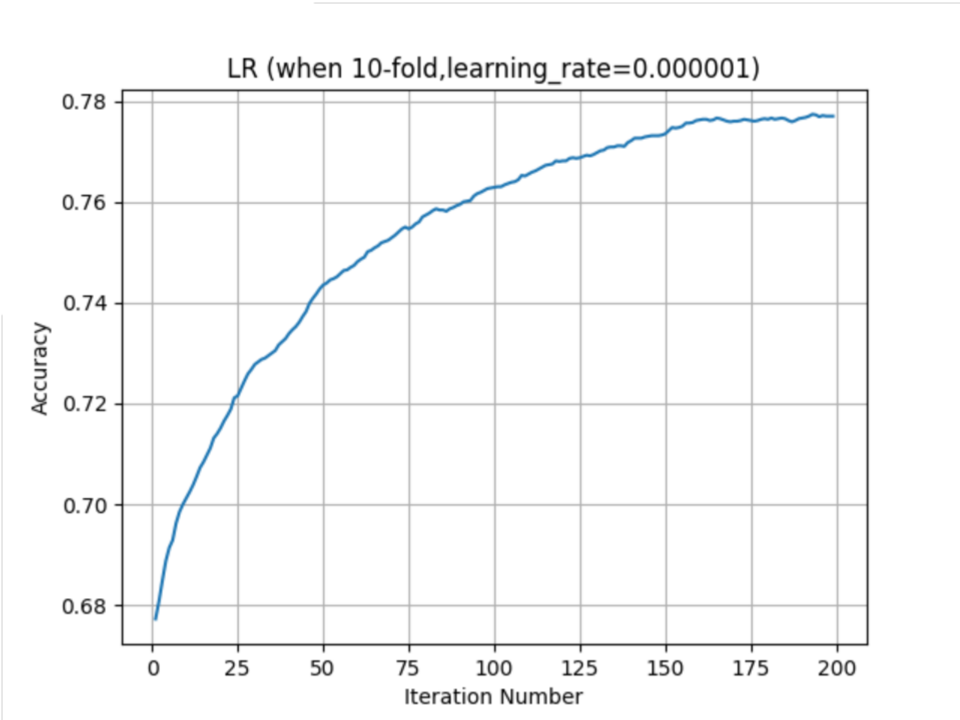
LR (when 11-fold,learning_rate=1)
Accuracy
Iteration Number

- 

LR (when 11-fold,learning_rate=0.00001)
Accuracy
Iteration Number

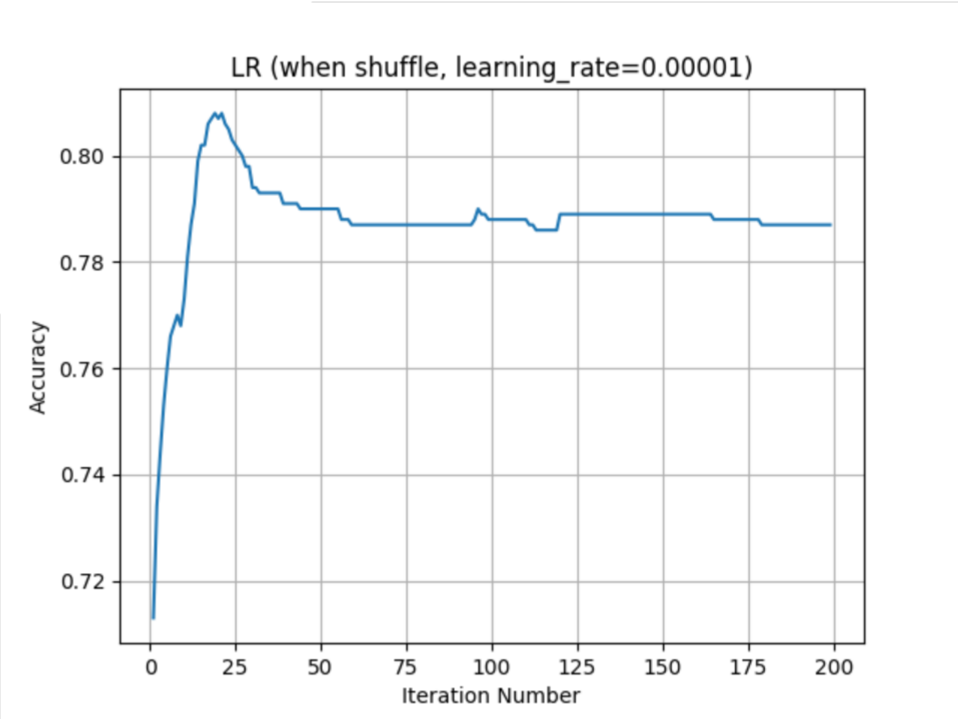
-

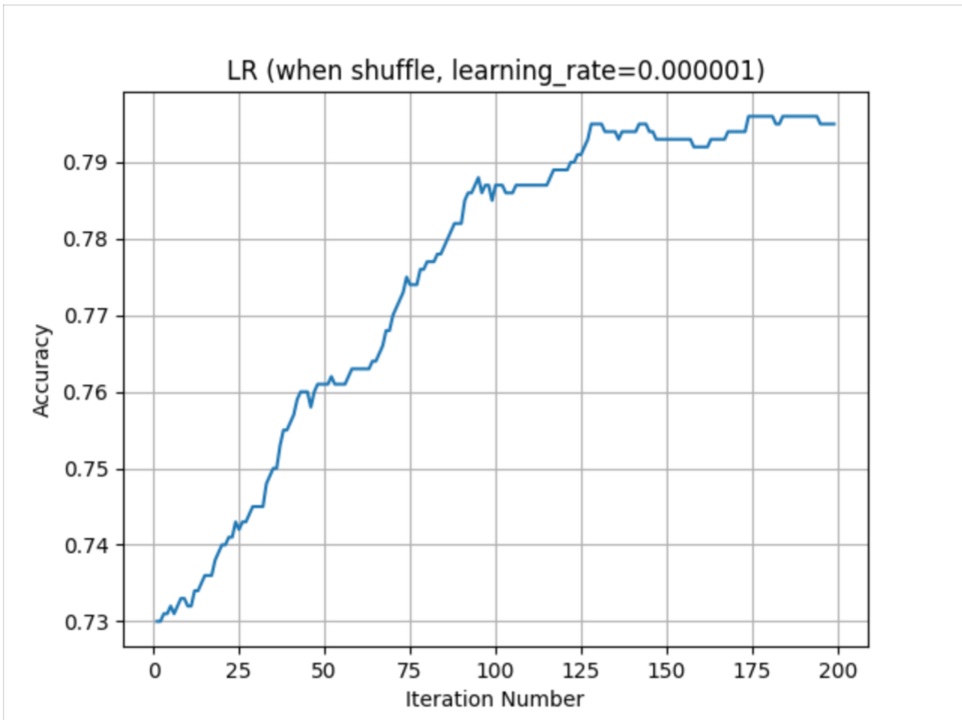


- 


- ```
  梯度下降已收敛
  LR在迭代次数为69时，对数据集进行10折划分后的准确率为0.7794999999999999
  梯度下降已收敛
  梯度下降已收敛
  梯度下降已收敛
  LR在迭代次数为70时，对数据集进行10折划分后的准确率为0.7794999999999999
  梯度下降已收敛
  梯度下降已收敛
  梯度下降已收敛
  梯度下降已收敛
  LR在迭代次数为71时，对数据集进行10折划分后的准确率为0.779375
  梯度下降已收敛
  梯度下降已收敛
  梯度下降已收敛
  梯度下降已收敛
  梯度下降已收敛
  LR在迭代次数为72时，对数据集进行10折划分后的准确率为0.7795
  ```
- 在运行时，我通过遍历固定迭代次数从1到100/200来查看最佳的固定迭代次数。
  ```

- 对于**k-fold**来说：
    - **学习率的选择**：当学习率为1时，因为学习率太大，导致准确率一直在最优点两侧波动，达不到最优值。当把学习率调整到10^{-5}或10^{-6}时，可以看到准确率图像呈现先上升后平稳的趋势，符合认知。因此可以看出，**学习率的设置**在梯度下降法中是十分重要的，只有设置合适的学习率才会有比较稳定的结果。
    - **趋于平稳的速度：**且当学习率取到10^{-6}时，准确率也会趋于平稳，只不过**趋于平稳的速度**慢于$10^{-}5$的速度，且两者达到的稳定状态相似，都接近于78%。因此可以看出，当学习率越大时，趋于平稳的速度越快。
    - **是否梯度收敛**：由最后一图看出，当阈值设置为10^{-3}时，开始出现某些`k-fold`出来的训练集**输出"梯度下降收敛"**的语句，也就是此时的w的变化很小，接近于最优点，证明此时确实是梯度收敛，与预测相符。

- 

- 

- 对于**shuffle**来说：

- 和k-fold类似，验证集准确率**先上升，后稳定到一定的水平**，且有可能出现先升后降的情况（学习率=0.00001时）。*（虽然看起来最后不大稳定，但实际上由于纵坐标的精度较大，在学习率=0.00001时浮动的范围也是从0.786到0.789）*

- **对比两种方法**：在学习率相同的情况下，对比“`k-fold`”和`shuffle`运行的结果，发现两者的准确率都会在某一时刻趋于稳定，只不过“`k-fold`”趋于稳定的**速率**快于`shuffle`，且两者稳定的准确率相近，都是78%左右。

2. 模型性能展示和分析

- 由于LR算法每次都要对所有样本进行计算来更新`w`值，且特别再增加了“防溢出”计算后速度降低，故**运行速度较慢**。以下呈现关于划分数据集方法（`k-fold`/`random.shuffle`）、k值、学习率η和固定迭代次数的调参。

	k-fold	shuffle	学习率	k值	固定迭代次数	准确率
初始	1	0	1	11	175	51.63%
优化1	1	0	1	11	8	77.29%
优化2	1	0	0.00001	11	175	78.0667%
优化3	1	0	0.00001	10	175	78.0625%
优化4	1	0	0.000001	11	175	77.69%
优化5	0	1	0.00001	-	19	80.8%
优化6	0	1	0.00001	-	175	78.7%
优化7	0	1	0.000001	-	175	79.6%
最优效果	0	1	0.00001	-	19	80.8%

调参变量为划分数据集方法（`k-fold`/`random.shuffle`）、k值（k-fold）、学习率η和 固定迭代次数，判断标准为验证集上的准确率。

分析：由上表可知，划分数据集方法、学习率和固定迭代次数对准确率的影响较大，而k值的选取对准确率的影响较小。

六、思考题

1. **不同的学习率对模型收敛有何影响？ 从收敛速度和是否收敛两方面来回答。**

　　学习率越大，权重向量`w`更新的越快，收敛速度越大，越快到达最优点；学习率越小，权重向量`w`更新越慢，可能需要更多的迭代次数来更新权重矩阵。但是如果学习率太高，每次权重向量更新的太大，有可能在更新的时候越过了最优点，导致在最优点附近不断震荡最后无法收敛到最优值，即不收敛。

　　因此，学习率的设置对模型收敛有很重要的影响。

2. **使用梯度的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。

比如，如果逻辑回归的目标函数是一个严格的凸函数，则在优化过程中很难到达最优点（极点）。在最优点附近的梯度很小，导致`w`更新速度越来越慢，甚至会造成无限靠近最优点但始终无法到达的情况发生，此时梯度永远不会为0。如果学习率比较高，有可能造成本来已经很接近最优值，却在更新时越过最优值到了对面的情况。因此，选择梯度的模长是否为0作为判断收敛终止条件并不合适。

一般来说，选择相邻两次权重向量的差值$w_{new} - w$的二范数（也就是差值的欧氏距离）是否低于某个阈值作为收敛终止条件，当两者的差值的二范数小于阈值时，判断梯度下降收敛，停止迭代。