人工智能实验报告（三）

- 院系/年级专业：数据科学与计算机学院/18级计科
- 学号/姓名：18340236/朱煜

一、实验题目

感知机&逻辑回归

二、实验原理

1、感知机（PLA）

（1）定义

- PLA是二分类的线性模型，其输入是实例的特征向量，输出的是事例的类别，分别是+1（正例）和-1（反例），感知机将对应于输入空间（特征空间）中将实例划分为正负的分离超平面，属于判别模型。
- 假设输入空间$X \in R^n$，输出空间$Y = \{+1, -1\}$。输入$x \in X$表示实例的特征向量，对应于输入空间的点；输出$y \in Y$表示实例的类别。由输入空间到输出空间的函数为

  $$f(x) = sign(w \cdot x + b)$$

  其中w和b为感知模型参数，$w \in R^n$叫做权值向量，$b \in R$叫做偏置，$w \cdot x$表示向量w和向量x的内积。

（2）线性可分

- 对于给定的数据集

  $$T = \{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$$

  其中，$x_i \in X = R^n$，$y_i \in Y = \{+1, -1\}, i = 1, 2, \cdots, N$，如果存在某个超平面$S$：$w \cdot x + b = 0$，使得能够将数据集的正实例和负实例完全正确地划分到超平面的两侧。即对所有$y_i = +1$的实例x_i，有$w \cdot x_i + b > 0$，对所有$y_i = -1$的实例x_i，有$w \cdot x_i + b < 0$，则称数据集T为线性可分数据集；否则，称数据集T线性不可分。

（3）损失函数

- 输入空间R^n中的任一点x_0到超平面S的距离为

  $$\frac{1}{||w||}|w \cdot x_0 + b|$$

  其中$||w||$是w的L_2范数。

  对于误分类点(x_i, y_i)，当$w \cdot x + b > 0$时，$y_i = -1$，当$w \cdot x + b < 0$时，$y_i = +1$，假设超平面S的误分类点集合为M，则有误分类点到分离超平面S的距离之和为

  $$-\frac{1}{||w||}\sum_{x_i \in M} y_i(w \cdot x_i + b).$$

  不考虑$\frac{1}{||w||}$，得到感知机的损失函数为

  $$L(w, b) = -\sum_{x_i \in M} y_i(w \cdot x_i + b)$$

(4) 感知机学习算法

- 感知机学习算法是对上述损失函数进行极小化，求得w和b，感知机学习算法是误分类驱动的，采用随机梯度下降法。极小化过程中不是一次使用M中所有误分类点的梯度下降，而是一次随机选取一个误分类点使其梯度下降。目标函数如下

$$\min_{w,b} = -\sum_{x_i \in M} y_i(w \cdot x_i + b)$$

- 损失函数的梯度为

$$\nabla_w L(w,b) = -\sum_{x_i \in M} y_i x_i$$

$$\nabla_b L(w,b) = -\sum_{x_i \in M} y_i$$

  则随机选择一个误分类点（x_i, y_i），对w, b进行更新

$$w \leftarrow w + \eta y_i x_i$$

$$b \leftarrow w + \eta y_i$$

  其中，$\eta(0 < \eta \leq 1)$为学习率。

- 感知机学习算法伪代码如下

```
Input:  train_data_set = {(x1,y1),(x2, y2),...,(xN,yN)},
        learning_rate = n
        max_iter = MAX_ITER
Output: w, b

w = w0
b = b0
Loop:
    for i in range(1, N):
        if sign(w * xi) != yi
            w = w + n*yi*xi
            b = b + n*yi
            max_iter = max_iter - 1
            break
    if there are still error points and max_iter > 0:
        goto Loop
return w, b
```

2、逻辑回归（LR）

(1) 定义

- 逻辑回归（LR）是一种线性回归模型，它假设数据服从伯努利分布，通过极大似然函数的方法，运用梯度下降法来求解参数，进而达到二分类的目的。
- 假设输入空间$X \in R^n$，输出是样本属于某个类别$Y = \{+1, -1\}$的概率。输入$x \in X$表示实例的特征向量，对应于输入空间的点；输出实例的类别的概率为

$$P(y=1|x) = \frac{e^{w \cdot x + b}}{1 + e^{w \cdot x + b}}, P(y=0|x) = 1 - P(y=1|x)$$

  其中w和b为感知模型参数，$w \in R^n$叫做权值向量，$b \in R$叫做偏置，$w \cdot x$表示向量w和向量x的内积。为方便表示,将w表示为$\tilde{w} = (w^T, b)^T$，将x表示为$\tilde{x} = (x^T, 1)^T$，即

$$P(y=1|x) = \frac{e^{\tilde{w} \cdot \tilde{x}}}{1 + e^{\tilde{w} \cdot \tilde{x}}} = \frac{1}{1 + e^{-\tilde{w} \cdot \tilde{x}}}, P(y=0|x) = 1 - P(y=1|x)$$

(2) 损失函数

- 令$\pi(x)=\frac{1}{1+e^{-w\cdot x}}$，在模型下， 利用给定数据$x$得到给定标签的概率，称为该问题的似然，逻辑回归中的似然函数为

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

在整个训练集上考虑似然函数，为

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

为方便运算，将似然函数取对数，得到对数似然函数为

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

对$L(w)$取负， 将$-L(w)$作为逻辑回归模型的损失函数，则损失函数为

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

(3) 逻辑回归算法

- 逻辑回归算法是对上述损失函数进行极小化，求得w，采用批梯度下降法。目标函数如下

$$\min_{w}=-\sum_{i=1}^{N}[y_i(w\cdot x_i)-log(1+e^{w\cdot x_i})]$$

- 损失函数的梯度为

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

则对所有样本计算梯度和，对w进行更新

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

其中，$\eta(0<\eta\leq 1)$为学习率。

- 逻辑回归算法伪代码如下

```
Input:  train_data_set = {(x1,y1),(x2, y2),...,(xN,yN)},
        learning_rate = n
        max_iter = MAX_ITER
Output: w, b

w = w0
Loop:
    for i in range(1, N):
        pi_x_i = 1/(1+e^(w*x_i))
        wnew = w + n*[y_i- pi_x_i]x_i
    if norm(wnew-w) < 0.0001:
        w = wnew
        break
    w = wnew
    if max_iter > 0:
        goto Loop
return w
```

三、实验过程

1、感知机（PLA）

（1）关键代码

- 由于本次实验只有一个数据集，需要自行划分验证集与测试集。本次采用的为分层随机抽样，保证验证集中正反例的比例与数据集的比例相同，具体划分函数如下。

```
def Divide(data_set, proportion):
    # 计算原数据集中正反例的个数
    label_dict = {'0': 0, '1': 0}
    for i in range(1, len(data_set)):
        if data_set[i][-1] == '0':
            label_dict['0'] += 1
        else:
            label_dict['1'] += 1
    print(label_dict)
    # 根据比例得到验证集中正反例应该有的数目
    for value in label_dict:
        label_dict[value] = int(label_dict[value]*proportion)
    select_set = [data_set[i] for i in range(1, len(data_set))]

    train_set = []
    test_set = []
    while len(select_set) > 0:
        # 生成随机数，随机抽样
        i = random.randint(0, len(select_set)-1)
        # 进行分层抽样
        if select_set[i][-1] == '0':
            if label_dict['0'] > 0:
                train_set.append(select_set[i])
                label_dict['0'] -= 1
                select_set.pop(i)
            else:
                test_set.append(select_set[i])
                select_set.pop(i)
        else:
            if label_dict['1'] > 0:
                train_set.append(select_set[i])
                label_dict['1'] -= 1
                select_set.pop(i)
            else:
                test_set.append(select_set[i])
                select_set.pop(i)
    return train_set, test_set
```

- PLA学习算法函数通过传入的`max`参数确定迭代次数的上限（在本次实验中为500），通过传入的`n`参数确定学习率，而在每次迭代过程中使用测试集计算正确率以评估`w`与`b`的好坏，并在迭代过程中实时打印出正确率，最终选择正确率最高的作为最终输出的`w`与`b`，具体实现代码如下

```
def PLA(train_set, w, b, n, max_i, test_set):
    print("迭代次数  正确率")
    # 判断是否存在误分类点
    test = True
    AC_set = []
    best_w = [i for i in w]
    best_b = b
```

```
    while test == True and max_i > 0:
        test = False
        for line in train_set:
            x_i = line[:-1]
            x_i = Str2Float(x_i)
            y_i = 1*(line[-1] == '1') + (-1)*(line[-1] == '0')
            # 存在误分类点，对w进行更新
            if y_i*(numpy.matmul(w, x_i)+b) <= 0:
                # w = w + n*yi*xi
                w = list(numpy.array(w) + n*y_i*numpy.array(x_i))
                b = b + n*y_i
                test = True
                # 迭代次数减少
                max_i -= 1
                AC = PLA_AC(test_set, w, b)
                AC_set.append(AC)
                # 保存最好的w与b
                if AC >= max(AC_set):
                    best_w = [i for i in w]
                    best_b = b
                print("%-10d%f" % (500-max_i, AC))
                break
    print("PLA最好迭代次数  最好正确率")
    print("%-15d%f" % (AC_set.index(max(AC_set))+1, max(AC_set)))

    return best_w, best_b
```

- 使用`w`与`b`作为参数，对测试集终端例子进行分类，分类结果与实际结果符号相同则判断为分类正确，具体实现代码如下

```
def PLA_AC(test_set, w, b):
    count = 0
    for line in test_set:
        x_r = line[:-1]
        x_r = Str2Float(x_r)
        y_r = 1*(line[-1] == '1') + (-1)*(line[-1] == '0')
        # 判断分类是否正确
        if y_r*(numpy.matmul(w, x_r)+b) > 0:
            count += 1
    return count/len(test_set)
```

(2) 实验结果

- 运行PLA，设置学习率为1，打印出迭代结果（部分），由于采用的是分层抽样，因此不能保证抽取到的样本为线性可分。在迭代次数上限为500左右时，可得最好迭代次数为23，最好正确率为73.5%，其正确率存在一定偶然性，在经过足够的迭代过程后，正确率大致保持在65%左右。

```
478       0.633750
479       0.630000
480       0.658125
481       0.638125
482       0.648125
483       0.650625
484       0.659375
485       0.636250
486       0.665625
487       0.637500
488       0.631250
489       0.634375
490       0.640625
491       0.650000
492       0.653125
493       0.640625
494       0.639375
495       0.639375
496       0.600000
497       0.623750
498       0.633750
499       0.636250
500       0.646250
PLA最好迭代次数   最好正确率
23              0.735000
```

2、逻辑回归（LR）

（1）关键代码

- 在LR中，验证集与测试集的划分方式与PLA相同，实现代码同上。
- 逻辑回归算法函数通过传入的`max`参数确定迭代次数的上限（在本次实验中为500），通过传入的`n`参数确定学习率，而在每次迭代过程中使用测试集计算正确率以评估`w`的好坏，并在迭代过程中实时打印出正确率，而迭代过程中计算批梯度的大小以判断收敛与否，具体实现代码如下

```
def LR(train_set, w, n, max_i, test_set):
    print("迭代次数   正确率")
    while max_i > 0:
        # 批梯度
        lw = numpy.array([0.0 for i in range(0, len(w))])
        for line in train_set:
            x_i = line[:-1]
            x_i = Str2Float(x_i)
            x_i.append(1.0)
            y_i = 1*(line[-1] == '1') + (0)*(line[-1] == '0')
            # 防止数据溢出 对梯度进行求和
            if -numpy.matmul(x_i, w) > 200:
                lw += n*(-y_i)*numpy.array(x_i)
          else:
                lw += n*((1/(1+math.exp(-numpy.matmul(x_i, w)))-y_i)
                       * numpy.array(x_i))
        # 更新w
        w = list(numpy.array(w) - lw)
        max_i -= 1
        print("%-10d%f" % (500-max_i, LR_AC(test_set, w)))
        # 根据范数判断是否收敛
        if numpy.linalg.norm(lw, ord=2) < 0.001:
            break
    return w
```

- 使用最终得到的`w`作为参数，对测试集终端例子进行分类，推测正例概率大于等于0.5则为正例，小于0.5则为反例，具体实现代码如下

```
def LR_AC(test_set, w):
```

```
    count = 0
    for line in test_set:
        x_r = line[:-1]
        x_r = Str2Float(x_r)
        x_r.append(1.0)
        y_r = 1*(line[-1] == '1') + (0)*(line[-1] == '0')
        # 防止数据溢出
        if -numpy.matmul(w, x_r) > 200:
            pi_x = 0
        else:
            pi_x = 1/(1 + math.exp(-numpy.matmul(w, x_r)))
        # 判断分类是否正确
        if (pi_x >= 0.5 and y_r == 1) or (pi_x < 0.5 and y_r == 0):
            count += 1
    return count/len(test_set)
```

(3) 实验结果

- 运行LR，设置学习率为0.00001，学习率的选择详见实验分析，打印出迭代结果（部分）。在迭代次数上限为500左右时，可得迭代次数为76时已经满足收敛条件，收敛后正确率保持在77.5%左右。

```
47      0.777500
48      0.777500
49      0.777500
50      0.776875
51      0.776875
52      0.776875
53      0.776875
54      0.775625
55      0.775625
56      0.775625
57      0.775625
58      0.775625
59      0.774375
60      0.773750
61      0.773750
62      0.774375
63      0.774375
64      0.774375
65      0.774375
66      0.774375
67      0.774375
68      0.773750
69      0.773750
70      0.773750
71      0.773750
72      0.773750
73      0.773750
74      0.773750
75      0.774375
76      0.775000
```

四、实验分析

1. 本次实验中，PLA的数据并非线性可分，则PLA的结果很大程度上取决于抽样的训练集样本与随机梯度下降时选择的数据样本，因此其正确率在迭代刚开始时会出现大幅度变化，有时会出现实验结果中迭代次数较少而正确率高的情况，而随着迭代进行，其正确率才会趋于稳定。但由于数据集的性质，其正确率不尽人意。
2. 在LR时，选择了0.00001作为学习率，这是由于在选择过程中，较大的学习率出现了不收敛的情况，导致正确率在一定范围内振荡，没有趋于稳定，具体学习率与收敛情况大致如下

学习率	是否收敛\收敛次数	正确率
1	否\无	50%~78%振荡
0.1	否\无	50%~78%振荡
0.01	否\无	60%~75%振荡
0.001	否\无	60%~75%振荡
0.0001	否\无	60%~75%振荡
0.00005	是\500以上	65%
0.00001	是\70~80	78%
0.000005	是\85	79%
0.000001	是\145	78%

可见学习率在0.0001与0.000005左右时可使LR最终收敛，而其收敛次数在500以内，最终正确率稳定在78%左右。根据上表可知，自始至终保持同样的学习率其实不太合适。因为迭代刚开始的时候，参数和最优解相距较远，需要保持一个较大的学习率逼近最优解。而迭代到一定次数后，保持最初的学习率，容易跨过最优点，在最优点附近来回振荡，因此学习率应顺着迭代进行变化。

3. 在LR时，选择了批梯度下降的方法，批梯度下降会获得全局最优解，缺点是在更新每个参数的时候需要遍历所有的数据，计算量会很大，并且会有很多的冗余计算。

五、实验思考

- 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。
    1. 学习率的大小决定了收敛速度，当学习率过小时，会导致模型收敛速度很慢，时间开销大。学习率适中能有效地帮助模型在适宜的迭代次数后达到收敛。
    2. 学习率决定了每一次迭代过程中，往梯度下降方向移动的距离，如果学习率过大，w会在最优点附近来回跳动，这样导致模型不收敛。而学习率小可以提高精度，降低发散的风险度。
- 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?
    1. 当模型输入的数据量较多时，由于使用批梯度下降，难以做到梯度为零，这样的收敛终止条件难以达到，需要经过大量时间的迭代，这是不可取的；而某些代价函数的设置并不是简单的单峰函数，会存在有局部最优解，梯度的模长为零作为梯度下降的终止条件会导致错过全局最优解。
    2. 判断模型收敛可通过参数的模长变化，当迭代过程中模长基本不变化，则可以认为模型收敛，这也可在测试集的正确率上体现，当参数模长变化小时，测试集正确率基本稳定，此时可认为模型收敛。