《人工智能实验》

实验报告

Lab 3

感知机学习算法与逻辑回归

学 院 名 称 ：	数据科学与计算机学院
专业（班级）：	17 级计算机科学与技术
学 生 姓 名 ：	薛伟豪
学　　　号 ：	17341178
联 系 方 式 ：	15013041671

Lab 3：感知机学习算法与逻辑回归

1. 感知机学习算法（PLA）

1.1. 算法原理

感知机是一个针对二元分类的模型，常用于解决线性二元分类问题，是人工神经网络中最基础的两层神经网络模型。模型的输入为待分类的特征向量，输出为对应的二元类别，通常用1和-1表示。具体地，假设感知机的输入为包含有n个特征的特征向量$\vec{x} = (x_1, x_2, \dots, x_n)$，感知机模型具有权重向量$\vec{w} = (w_1, w_2, \dots, w_n)$以及阈值$\theta$，通过以下公式：

$$f(\vec{x}) = sign\left(\left(\sum_{i=1}^{n} w_i x_i\right) - \theta\right)$$

得到输入的特征向量的类别。进一步地，我们令$w_0 = -\theta$，并引入额外的辅助坐标$x_0 = 1$，将上述的公式线性化，有：

$$f(\vec{x}) = sign(\widetilde{w}^T \tilde{x})$$

当然，如果想要让感知机模型分类的效果达到最佳，那么我们必须要找到最优的增广权向量$\widetilde{w}$。感知机学习算法（PLA）就是要找到能让损失函数最小的参数。考虑集合S为所有未正确分类的点的集合，我们定义损失函数如下：

$$L(\widetilde{w}) = -\sum_{\tilde{x}_i \in S} y_i \widetilde{w}^T \tilde{x}$$

显然，在错误分类的情况下，损失函数$L(\widetilde{w}) > 0$。基于此，我们可以得到$\widetilde{w}$的更新步骤：我们随机初始化增广权向量，并对某个样本进行预测；如果预测正确则对下一个样本进行预测，直到找到一个预测错误的样本$\tilde{x}_i$。然后，我们利用这个预测错误的样本$\tilde{x}_i$，采用梯度下降法对$\widetilde{w}$进行更新，更新方式如下：

$$\widetilde{w}_{(t+1)} \leftarrow \widetilde{w}_{(t)} + \eta y_i \tilde{x}_i$$

其中，η为学习率。我们不断重复以上步骤进行迭代，直到模型收敛（如果模型是线性可分的话，即所有样本都预测正确）。这样一来，我们便找到了能够让损失函数$L(\widetilde{w})$达到最小的增广权向量$\widetilde{w}$，也就得到了一个目标的感知机模型。

1.2. 伪代码

Algorithm PLA

Input: $X \in R^{m\times n}$, $Y \in R^{n\times 1}$

Output: W_{best}

1. cycle ← Iteration Times
2. α ← Learning Rate
3. W_0 ← $\{w_i=0 \text{ for all } w_i\} \in R^{m\times 1}$
4. **for** t=0 **to** cycle **do:**
5.   **for** i=0 **to** n:
6.   **if** the sign of W_tX_i and Y_i are different **then:**
7. $W_{t+1} \leftarrow W_t + \alpha * Y_i X_i$
8.   **break** from loop line 5
9.   **end if**
10.   **end for**
11.   **if** all W_tX_i and Y_i have the same sign **then:**
12.   **break** from loop line 4
13.   **end if**
14. **end for**
15. W_{best} ← W after loop
16. return W_{best}

1.3. 代码展示

- **使用PLA算法计算出目标增广权向量W**

```python
def PLA(train_set, cycle, alpha):
    '''
    PLA 函数输入为训练集，迭代次数 cycle，学习率 alpha
    使用 PLA 算法计算出最终的增广权向量 W
    '''
    #初始化 W 为零向量
    W = np.zeros((1, len(train_set[0])))
    X = train_set[:, :-1]
    X = np.insert(X, 0, np.ones(len(X)), axis=1)
    Y = train_set[:, -1:]
    #迭代 cycle 次
    for cycle_index in range(cycle):
        #变量 is_finished 用于判断是否满足迭代停止的条件
```

```
        is_finished = True
        for i in range(len(X)):
            if np.sign(Y[i]) != np.sign(np.dot(W, X[i])):
                W = W + alpha * Y[i] * X[i]
                is_finished = False
                break
        if is_finished:
            break
    return W
```

- **利用得到的增广权向量W对测试集进行分类**

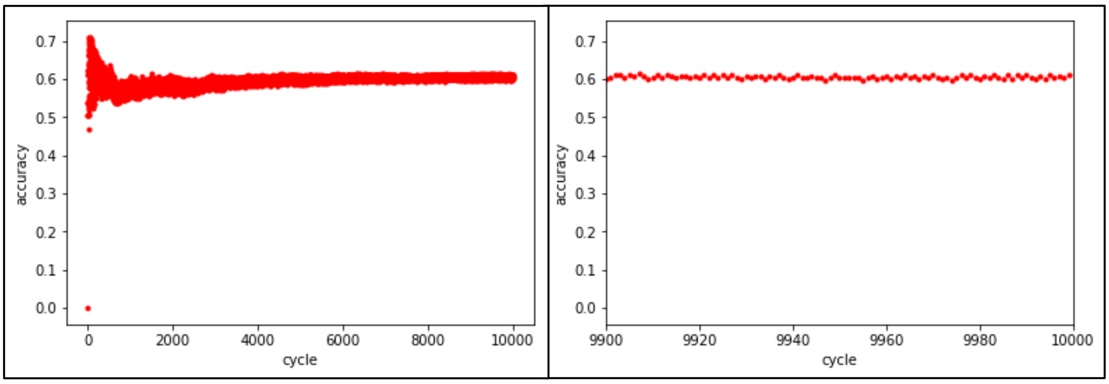
```
def classify(W, test_set):
    '''
    利用增广权向量 W 对测试集进行分类
    函数返回回测试集的分类预测结果列表
    '''
    predict_Y = []
    test_set = np.insert(test_set, 0, np.ones(len(test_set)), axis=1)
    for item in test_set:
        label = np.sign(np.dot(W, item))
        predict_Y.append(label)
    return predict_Y
```

- **将预测标签和实际标签对比计算模型预测的准确率**

```
def cal_accuracy(W, test_set, cycle):
    '''
    计算预测准确率
    '''
    predict_Y = classify(W, test_set[:, :-1])
    real_Y = test_set[:, -1:]
    count = 0
    for index in range(len(real_Y)):
        if predict_Y[index] == real_Y[index]:
            count += 1
    return count / len(real_Y)
```

1.4. 实验结果及分析

我们首先观察迭代次数对于准确率的影响。在学习率为0.1的情况下，我们对W进行迭代10000次，观察每次迭代后预测准确率的变化情况。同时放大后期的散点图，观察后期的收敛情况。这里我们将全部的数据集作为训练集和测试集，散点图如下所示：



我们可以发现，在学习率为0.1的情况下，使用PLA算法进行增广权向量的迭代，前期的预测准确率波动很大。最大的预测准确率也出现在前期，为70.93%。而在迭代后期，预测准确率稳定在60%左右，波动不大。在调整了其他不同的学习率后，依然呈现出这样的趋势，而且不同的学习率在该数据集上几乎没有影响。

为了更好的评估模型的效果，我们对数据集不同划分下的预测准确率进行评估。由于学习率对模型几乎没有影响，考虑学习率为1和不同的迭代次数，模型的表现如下：

迭代次数	40	100	500	1000	2000
训练集[0 : 8000] 验证集[0 : 8000]	0.7093	0.5364	0.5761	0.5794	0.5900
训练集[0 : 7500] 验证集[7500:8000]	0.6820	0.5080	0.5320	0.5480	0.5680
训练集[0 : 4000] 验证集[4000:8000]	0.7083	0.5385	0.5665	0.5620	0.5745
训练集[0 : 500] 验证集[500:8000]	0.7083	0.5335	0.5724	0.5747	0.5868

不难看出，使用PLA模型对验证集进行分类，前期的预测准确率波动较大，在迭代次数为40左右时出现了高准确率，接近71%，。随着迭代次数的增加，预测准确率也稳定下来，接近60%，这符合散点图的分布。此外我们还可以看到，大多数情况下验证集数量大于训练集数量时，普遍呈现出稍好的预测准确率。

2. 逻辑回归算法（LR）

2.1.算法原理

和感知机学习算法一样，逻辑回归算法（LR）是解决二元分类问题的一个重要算法。逻辑回归算法属于软分类算法，通过预测出的概率值进行分类。

再次使用PLA算法描述中提出的假设。假设算法输入为有n个特征的特征向量$\vec{x} = (x_1, x_2, \dots, x_n)$，我们希望找到一个最优权重向量$\vec{w} = (w_0, w_1, \dots, w_n)$，能够让最小二乘法的回归模型$\widetilde{w}^T\tilde{x} = w_0 + \sum_{i=1}^{n} w_i x_i$起到最优的分类效果。我们这里采用0和1作为二元分类问题的类别，很显然，对于二元分类问题$\widetilde{w}^T\tilde{x}$取值范围为(-∞, +∞)，预测出的概率可能会超出区间[0, 1]，故我们需要对$\widetilde{w}^T\tilde{x}$进行映射。在逻辑回归模型中，这个映射函数就是sigmoid函数。

我们规定，$h(x)$为某个样本预测为1的概率，在二元分类问题中，$1 - h(x)$便为该样本预测为0的概率。由sigmoid函数可得

$$h(x) = p(y = 1 \mid x) = \frac{1}{1 + e^{-\widetilde{w}^T\tilde{x}}}$$

那么该样本属于某个类别y，即0或1的概率可以用以下通式表示：

$$p(y \mid x\,;\,\widetilde{w}) = h(x)^y \big(1 - h(x)\big)^{1-y}$$

考虑整个训练集样本的似然函数如下：

$$\prod_{i=1}^{n} p(y \mid x\,;\,\widetilde{w}) = \prod_{i=1}^{n} h(x_i)^{y_i}\big(1 - h(x_i)\big)^{1-y_i}$$

我们希望找到一个增广权向量$\widetilde{w}$，使得似然函数达到最大值。将似然函数取负对数，得到目标函数$L(\widetilde{w})$如下。我们要做的就是找到一个最优的$\widetilde{w}$，使得$L(\widetilde{w})$达到最小值

$$L(\widetilde{w}) = -\sum_{i=1}^{n}\Big(y_i \, log\, h(x_i) + (1 - y_i)\, log\big(1 - h(x_i)\big)\Big)$$

我们采用梯度下降法进行求解。将上式对$\widetilde{w}$求偏导数，有：

$$\frac{\partial L(\widetilde{w})}{\partial \widetilde{w}} = -\sum_{i=1}^{n}\left[\left(y_i - \frac{1}{1 + e^{-\widetilde{w}^T\tilde{x}_i}}\right)\tilde{x}_i\right]$$

由此我们得到了增广权向量$\widetilde{w}$的更新公式：

$$\widetilde{w}_{(t+1)} \leftarrow \widetilde{w}_{(t)} - \eta \frac{\partial L(\widetilde{w})}{\partial \widetilde{w}}$$

其中，η为学习率。我们重复以上步骤进行迭代，直到模型收敛。这样一来，我们便找到了能够让损失函数$L(\widetilde{w})$达到最小的增广权向量$\widetilde{w}$，也就得到了一个目标的逻辑回归模型。

2.2. 伪代码

Algorithm PLA

Input: $X \in R^{m\times n}$, $Y \in R^{n\times 1}$

Output: W_{best}

1. cycle ← Iteration Times
2. α ← Learning Rate
3. W_0 ← {w_i=0 for all w_i } $\in R^{m\times 1}$
4. **for** t=0 **to** cycle **do:**
5. gradient ← X(Y - sigmoid(W·X))
6. W_{t+1} ← W_t + α*gradient
7. **if** gradient is not changed **then:**
8. **break** from loop line 4
9. **end if**
10. **end for**
11. W_{best} ← W after loop
12. return W_{best}

2.3. 核心代码展示

- **使用LR算法计算出目标增广权向量W**

```
def LR(train_set, cycle, alpha):
    '''
    PLA 函数输入为训练集，迭代次数 cycle，学习率 alpha
    使用 PLA 算法计算出最终的增广权向量 W
    '''
    #初始化 W 为零向量
    W = np.zeros((1, len(train_set[0])))
    X = train_set[:, :-1]
    X = np.insert(X, 0, np.ones(len(X)), axis=1)
    Y = train_set[:, -1:]
```

```
    #迭代 cycle 次
    for i in range(cycle):
        gradient = np.dot((Y - sigmoid(np.dot(W,
X.transpose()))).transpose()).transpose(), X)
        if (abs(gradient) < 1e-9).all():
            break
        W = W + alpha * gradient
    return W
```

- **利用得到的增广权向量W对测试集进行分类**

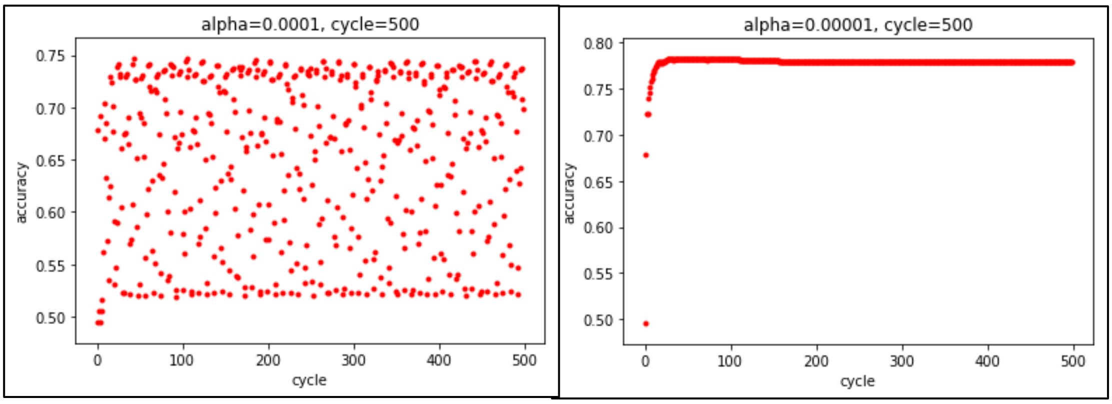
```
def classify(W, test_set):
    '''
    利用增广权向量 W 对测试集进行分类
    函数返回测试集的分类预测结果列表
    '''
    predict_Y = []
    test_set = np.insert(test_set, 0, np.ones(len(test_set)), axis=1)
    for item in test_set:
        p = sigmoid(np.dot(W, item))
        if p >= 0.5:
            predict_Y.append(1)
        else:
            predict_Y.append(0)
    return predict_Y
```

- **利用得到的增广权向量W对测试集进行分类**

```
def classify(W, test_set):
    '''
    利用增广权向量 W 对测试集进行分类
    函数返回测试集的分类预测结果列表
    '''
    predict_Y = []
    test_set = np.insert(test_set, 0, np.ones(len(test_set)), axis=1)
    for item in test_set:
        p = sigmoid(np.dot(W, item))
        if p >= 0.5:
            predict_Y.append(1)
        else:
            predict_Y.append(0)
    return predict_Y
```

2.4. 实验结果及分析

为了观察迭代次数以及学习率对于准确率的影响。在学习率为0.0001和0.00001的情况下，我们分别对W进行迭代500次，观察每次迭代后预测准确率的变化情况。这里我们将全部的数据集作为训练集和测试集，散点图如下所示：



我们可以发现，在学习率为0.0001的情况下，使用LR算法进行增广权向量的迭代，预测准确率剧烈波动，未能收敛，且此时最高准确率不超过75%。由此可以推知，0.0001的学习率对于该数据集来说过大，无法收敛。而在学习率为0.00001的情况下，使用LR算法进行迭代，能够清楚地看到，预测准确率在后期收敛，准确率稳定在77.88%。

为了更好的评估模型的效果，我们对不同学习率下的预测准确率进行评估。考虑收敛情况下的学习率以及迭代次数5000，对于不同训练集和验证集情况，模型的表现如下：

学习率	0.00001	0.000001	0.0000001	0.00000001
训练集[0 : 8000] 验证集[0 : 8000]	0.7819	0.7788	0.7824	0.7460
训练集[0 : 7000] 验证集[7000:8000]	0.7600	0.7600	0.7620	0.7380
训练集[0 : 4000] 验证集[4000:8000]	0.7738	0.7750	0.7765	0.7238
训练集[0 : 2000] 验证集[2000:8000]	0.7722	0.7748	0.7752	0.7160
训练集[0 : 1000] 验证集[1000:8000]	0.7633	0.7721	0.7590	0.7084

综合来看，当迭代次数都为5000次的时候，学习率为0.000001时，模型的表现最好。

3. 思考题

3.1. 有什么手段可以使PLA适用于非线性可分的数据集?

- 可以设置迭代次数，当达到给定的迭代次数后就返回此时的W，不管这个W是否能够完美划分所有的训练集。
- 可以使用口袋算法。我们可以找到一个W，使得在训练集里用这个W划分后，分类错误的样本数量最少。具体而言，我们可以设置一个全局的最优权重向量。每次迭代更新W之后，计算出该W下分类预测的准确率，如果准确率得到提高，那么再将最优权重向量更新为W。否则不进行更新。

3.2. 不同的学习率η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。

- 如果学习率过大，尽管收敛速度有可能变快，但也有可能会出现W会在最优解附近来回跳动的情况，甚至还会导致模型发散不收敛。
- 如果学习率过小，则模型发散的风险较小，但是会导致模型收敛速度过慢，十分耗时。

3.3. 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?

- 使用梯度的模长是否为零作为梯度下降的收敛条件不合适。在使用梯度下降法靠近最优解的时候，有可能出现梯度接近零的情况，但此时梯度下降的速率很慢，同时也有可能会发生无法做到梯度恰好为零的情况，这些都可能会导致迭代无法停止，耗费大量时间。
- 为了判断模型是否收敛，我们可以采取以下方式:

1) 设置合适的迭代次数。如果算法达到该迭代次数时，则可认为模型已经收敛。需要注意的是，采用这种方法时我们还需要观察梯度的模长是否在减小。
2) 每次迭代后判断梯度的模长是否小于一个阈值，如果小于，则可认为模型收敛。