中山大學
SUN YAT-SEN UNIVERSITY

《人工智能实验》

实验报告

（实验三）

学 院 名 称 ：	数据科学与计算机学院
专业（班级）：	17 计算机科学与技术
学 生 姓 名 ：	仲逊
学　　号	16327143
时　　间 ：	2019　年　9　月　26　日

实 验 3 ：感知机算法与逻辑回归

一. 实验目的

1. 掌握如何使用感知机学习算法和逻辑回归算法对数据进行二元分类。
2. 在给定数据集分别使用感知机学习算法和逻辑回归算法训练并预测未知样本标签。

二. 算法原理

1. 感知机

感知机学习算法（Perceptron Learning Algorithm）是一种非常朴素的二元线性分类模型，它的思想就是用一条直线（或者一个超平面）把只有两种标签的数据集$D = \{(\boldsymbol{x_1}, y_1), (\boldsymbol{x_2}, y_2), \dots, (\boldsymbol{x_m}, y_m)\}$分成两部分，落在这个超平面两边的数据分别对应不同的标签。这个超平面的方程我们可以用$W^T\boldsymbol{x} + b = 0$来表示，现在在训练集的每个样本$\boldsymbol{x_i}$中增加一个$x_{i0} = 1$，并在$W$中增加$W_0 = b$，超平面方程即可表示成 $\widehat{W}^T\hat{\boldsymbol{x}} = 0$，那么落在其两边的点就对应了$\widehat{W}^T\hat{\boldsymbol{x}} > 0$和$\widehat{W}^T\hat{\boldsymbol{x}} < 0$，因此只需将超平面两侧的标签值转化为1和-1，其最终结果就可以简单地用$sign(\widehat{W}^T\hat{\boldsymbol{x}})$来表示。

感知机算法的核心在于权重向量W如何更新最后才能将数据集划分成这样的两部分，方法如下：多次遍历所有训练样本，每当找到一个预测错误的样本$\boldsymbol{x_i}$，则按照如下公式更新权重向量：

$$\mathrm{W_{t+1}} \leftarrow \mathrm{W_t} + [\![y_i \neq sign(W_t^T\boldsymbol{x_i})]\!]\mathrm{y_i}\mathbf{x_i}$$

直到所有的训练样本都预测正确为止。

这个更新公式在直观上是很容易理解的，首先$\widehat{W}^T\hat{\boldsymbol{x}}$这个向量内积的正负可以由向量$\widehat{W}$和向量$\hat{\boldsymbol{x}}$的夹角得到，当正样本被误分类为负样本时，即$\widehat{W}$和$\hat{\boldsymbol{x}}$夹角应该为锐角但现在为钝角，则根据公式更新后$\widehat{W}$会向$\hat{\boldsymbol{x}}$方向靠近，同样当负样本被误分类为正样本时，根据公式更新后$\widehat{W}$会向$\hat{\boldsymbol{x}}$的反方向靠近，即远离$\hat{\boldsymbol{x}}$，都会离正确标签更近。

感知机学习算法的缺点是无法处理线性不可分的数据集，并且只根据当前选取的误分类样本进行权重更新，所以很可能出现之前分类正确的样本在权值更新后反而又变为分类不正确的情况。当数据集线性不可分时，每次遍历数据集都会发现误分类样本，然后以此更新权值，最后无法收敛到一个稳定的值，只能调整遍历的次数来适应。

2. 逻辑回归

逻辑回归也是一种常用的二元线性分类模型，它的思想和PLA是比较相似的，也是用一条直线（或者一个超平面）把只有两种标签的数据集$D = \{(\boldsymbol{x_1}, y_1), (\boldsymbol{x_2}, y_2), \dots, (\boldsymbol{x_m}, y_m)\}$分成两部分，沿用感知机的符号，在处理完数据集后，平面方程即表示成 $\widehat{W}^T\widehat{\boldsymbol{x}} = 0$，不同于PLA将最终结果简单地用$sign(\widehat{W}^T\widehat{\boldsymbol{x}})$来表示，逻辑回归的得到的结果是一个概率值，它使用的是sigmoid函数，最终结果为正类的概率值为：

$$h(x) = \frac{1}{1 + e^{-\widehat{W}^T\widehat{x}}}$$

直观上也非常容易理解，sigmoid是S形函数，它将$(-\infty, +\infty)$的值映射到$(0,1)$之间，$\widehat{W}^T\widehat{\boldsymbol{x}} = 0$时$h(x) = 0.5$。确定预测表达式后应当选择的就是损失函数，虽然逻辑回归中有回归二字，但事实上它是用于分类问题，故损失函数没有使用线性回归的均方误差，而是使用了分类问题常用的交叉熵作为损失函数，其表达式为：

$$L(W) = -\sum_{i=1}^{N}(y_i log\ \ h(x_i) + (1 - y_i) log\ (1 - h(x_i)))$$

具体含义即为当标签y_i为1时，预测值$h(x_i)$越接近1损失函数值越小，越接近0损失函数越大，标签y_i为0时同理。现在的目标转化为了损失函数最小化，使用梯度下降法即可。

$$\nabla L(W) = -\sum_{i=1}^{N}\big(y_i - h(x_i)\big)x_i$$

$$W_{t+1} \leftarrow W_t - \eta \nabla L(W_t)$$

其中$\nabla L(W)$为$L(W)$梯度，η为学习率。

三. 伪代码

■ PLA伪代码：

输入:训练集$D = \{(\boldsymbol{x_1}, y_1), (\boldsymbol{x_2}, y_2), \dots, (\boldsymbol{x_m}, y_m)\}$;

处理数据集，每个样本增加$\boldsymbol{x_{i0}} = 1$;

处理标签，使得标签取值为$\{+1, -1\}$;

初始化权值向量W为0;

for $k\ in\ range$(0:遍历数据集次数$cycle$) ***do***:

    for 数据集的所有个样本 ***do***:

x_i分类标签预测值$h(x_i)$为$sign(W_t^T \boldsymbol{x_i})$;

if $\boldsymbol{x_i}$被误分类(即$h(x_i) \neq y_i$):

更新$W_{t+1} \leftarrow W_t + y_i \boldsymbol{x_i}$;

end if

end for

end for

输出: 权重向量W_{final}

- LR伪代码:

输入:训练集$D = \{(\boldsymbol{x_1}, y_1), (\boldsymbol{x_2}, y_2), \ldots, (\boldsymbol{x_m}, y_m)\}$;

处理数据集，每个样本增加$\boldsymbol{x_{i0}} = 1$;

初始化权值向量$\boldsymbol{W}$为0;

for k *in* $range$(0:迭代次数$cycle$) ***do***:

预测概率值$h(x_i) = sigmoid(Matrix(D) * W_t^T)$;

误差error = 预测概率值$h(x_i)$ − 真实标签Y;

更新$W_{t+1} \leftarrow W_t + \eta \sum_{i=1}^{N} \left(y_i - h(x_i)\right) x_i$;

end for

输出: 权重向量W_{final}

四. 代码截图

- PLA加载数据集

```
def loadData(path):
    """
    从给定路径的 csv 数据集中加载数据
    Args:
        path: 字符串 代表数据集绝对路径

    Returns:
        train,labels 二元组
        train numpy 数组，增广后的(补充 x_0=1)训练数据
        labels numpy 数组，处理后的训练集标签(转化为{+1,-1})
    """
```

```
    dataSet=pd.read_csv(path,header=None)
    #取出数据集标签
    labels=dataSet.pop(len(dataSet.columns)-1).values.tolist()
    #标签是0和1，转化成-1和1
    labels=np.array([-1 if label==0 else 1 for label in labels])
    #数据集第一列补x0=1
    dataSet.insert(0,'0',np.ones(len(dataSet)))
    train=dataSet.values

    return train,labels
```

■ PLA更新权重（也相当于对于其损失函数的梯度下降法）

```
def PLAGradDescent(train,labels,alpha=0.001,cycle=50):
    """
    在给定训练集上执行感知机学习算法的梯度下降法
    Args:
        train numpy数组，增广后的(补充x0=1)训练数据
        labels numpy数组，处理后的训练集标签(只有{+1,-1})
        alpha 学习率
        cycle 感知机学习中遍历数据集的变数
    Returns:
        weights numpy矩阵类型 列向量
        例如：np.mat([[1],[2],[3],[4]])
        表示权重w为列向量(1,2,3,4)
    """

    #训练数据转化为矩阵形式
    trainMat = np.mat(train)
    #label转化成列向量
    labelVec = np.mat(labels).transpose()
    sampleNum,featNum = np.shape(trainMat)

    #weights初始化为0，定义为列向量方便矩阵运算
    weights = np.zeros((featNum,1))
    #遍历整个数据集cycle轮
    for i in range(cycle):
        for j in range(sampleNum):
            #预测分类
            h=sign(trainMat[j]*weights)
            #遇到误分类样本使用公式更新权重
            if h!=labelVec[j][0]:
                weights=weights+alpha*trainMat[j].transpose()*labelVec[j]
    return weights
```

■ 逻辑回归加载数据集

```
    def loadData(path):
    """
    从给定路径的 csv 数据集中加载数据
    Args:
        path：字符串 代表数据集绝对路径

    Returns:
        train,labels 二元组
        train numpy 数组，增广后的(补充 x0=1)训练数据
        labels numpy 数组，训练集标签
    """
    dataSet=pd.read_csv(path,header=None)
    #取出数据集标签
    labels=dataSet.pop(len(dataSet.columns)-1).values
    #数据集第一列补 x0=1
    dataSet.insert(0,'0',np.ones(len(dataSet)))
    train=dataSet.values
    return train,labels
```

■ 逻辑回归损失函数的梯度下降法

```
def LRGradDescent(train,labels,alpha=0.001,cycle=500):
    """
    在给定训练集上执行 logistics 回归交叉熵损失函数的梯度下降法
    Args:
        train numpy 数组，增广后的(补充 x_0=1)训练数据
        labels numpy 数组，训练集标签
        alpha 学习率
        cycle 梯度下降法的迭代次数
    Returns:
        weights numpy 矩阵类型 列向量
        例如：np.mat([[1],[2],[3],[4]])
        表示权重 w 为列向量(1,2,3,4)
    """
    #训练数据转化为矩阵形式
    trainMat = np.mat(train)
    #labels 转化成列向量，方便矩阵运算
    labelVec = np.mat(labels).transpose()
    sampleNum,featNum = np.shape(trainMat)

    #weights 初始化为 0，定义为列向量方便矩阵运算
    weights = np.zeros((featNum,1))
    for i in range(cycle):
```

```
        #预测概率值
        h=sigmoid(trainMat*weights)
        #和实际的误差
        err=labelVec-h
        #梯度下降到指定阈值可以认为收敛，退出循环
        if np.linalg.norm(trainMat.transpose()*err)/sampleNum < 1e-4:
            break
        #Logistics 回归 交叉熵损失函数 批梯度下降法公式
        weights=weights+alpha/sampleNum*trainMat.transpose()*err
    return weights
```

- **K折交叉验证划分数据集**

```
def KFold(dataSet,k):
    """
    进行 K 折交叉验证的数据集划分
    Args:
        dataSet numpy 数组 含标签的数据集
        k 整型 将数据集划分为 k 个部分
    Returns:
        列表 每个元素代表数据集的一部分
    """
    #获取打乱后的数据集下标
    rowIndex = np.arange(dataSet.shape[0])
    np.random.shuffle(rowIndex)
    foldSize = int(len(dataSet)/k)
    foldList = []
    counts=k
    #每次取 int(len(dataSet)/k)个 sample 作为一个 fold
    while(counts>1):
        foldList.append(dataSet[rowIndex[(k-counts)*foldSize:
                                         (k-counts+1)*foldSize]])
        counts-=1
    foldList.append(dataSet[rowIndex[(k-1)*foldSize:]])

    return [fold.tolist() for fold in foldList]
```

五. 实验结果及分析

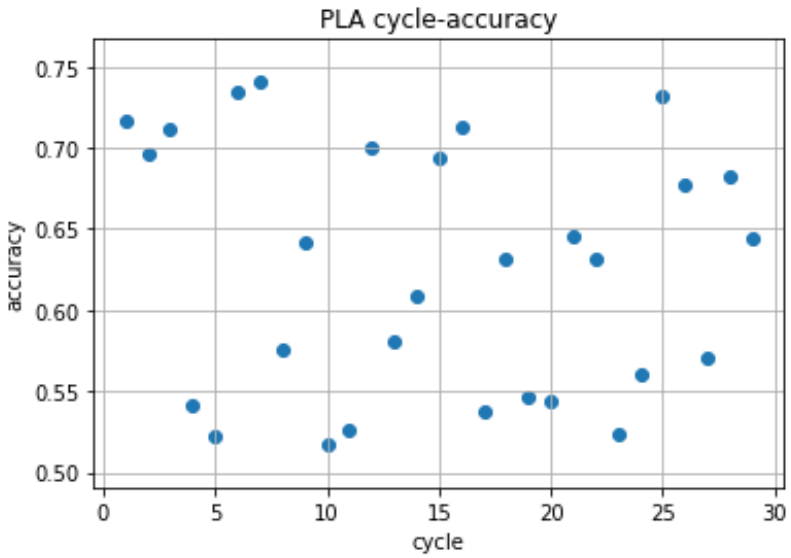
1. 将整个数据集用于训练

- 感知机学习算法

取学习率α为1，遍历轮数为1-10，得到准确率结果为：

```
In [8]: runfile('D:/AI Lab/lab3/PLA.py',
cycle 1 accuracy: 0.71625
cycle 2 accuracy: 0.6965
cycle 3 accuracy: 0.711875
cycle 4 accuracy: 0.5415
cycle 5 accuracy: 0.522
cycle 6 accuracy: 0.734875
cycle 7 accuracy: 0.740625
cycle 8 accuracy: 0.575125
cycle 9 accuracy: 0.641875
cycle 10 accuracy: 0.51725
```

可以看出PLA的准确率随着遍历轮数的不同震荡，没有收敛，为了进一步观察，作cycle-准确率的散点图如下：



由此可知该数据集不是严格线性可分的，因此PLA最终不会收敛。因为PLA每次遍历数据集都会发现存在误分类样本，然后以此更新权值，最后无法收敛到一个稳定的值，只能调整遍历的次数来适应。

■ 逻辑回归

一开始我使用的梯度下降法中使用的更新公式为$W_{t+1} \leftarrow W_t + \eta \sum_{i=1}^{N} \left(y_i - h(x_i)\right) x_i$，由于所有样本的梯度之和非常大，学习率α常常要取$10^{-4}$以下才能保证收敛，后来我使用了比较常用的将梯度之和除以训练样本数的做法，即$W_{t+1} \leftarrow W_t + \frac{\eta}{N} \sum_{i=1}^{N} \left(y_i - h(x_i)\right) x_i$作为更新公式，这时取学习率α只要取0.1以下即可收敛。

取学习率α为0.1，迭代次数为9000，得到的结果如下：

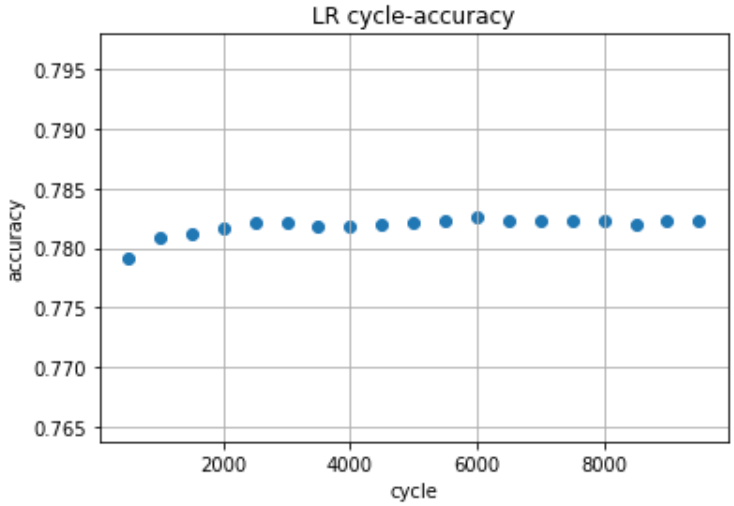
```
Gradient L-2 Norm: 0.9733223447743862
Gradient L-2 Norm: 0.6198193261647535
Gradient L-2 Norm: 0.486531787806936
Gradient L-2 Norm: 0.41297291130952085
Gradient L-2 Norm: 0.35704240257977493
```

第1-5次迭代

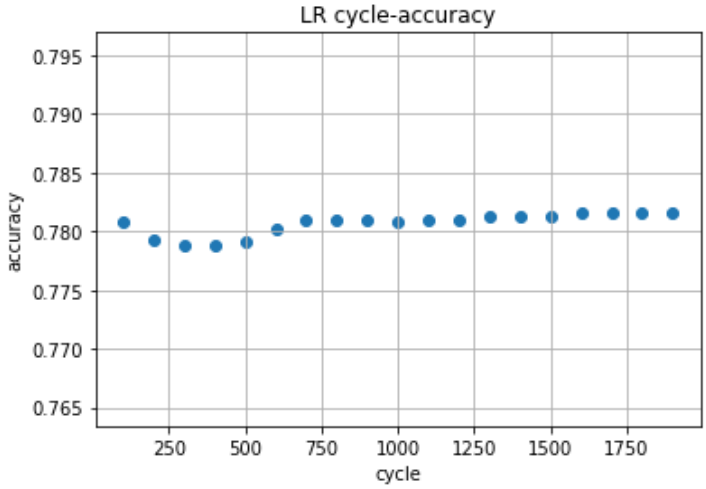
```
Gradient L-2 Norm: 0.00026986073441279996
Gradient L-2 Norm: 0.00026980327673859167
Gradient L-2 Norm: 0.0002697458313130181
Gradient L-2 Norm: 0.00026968839813334581
Gradient L-2 Norm: 0.00026963099771973098
```

第8995-9000次迭代

可以明显看出梯度（这里指$\frac{\eta}{N} \sum_{i=1}^{N} \left(y_i - h(x_i)\right) x_i$）的$L_2$范数在收敛，且开始收敛的很快，越接近于0就收敛越慢，最后几乎不变，最终得到的训练集准确率为0.78225。为了进一步观察，作cycle-准确率的散点图如下：

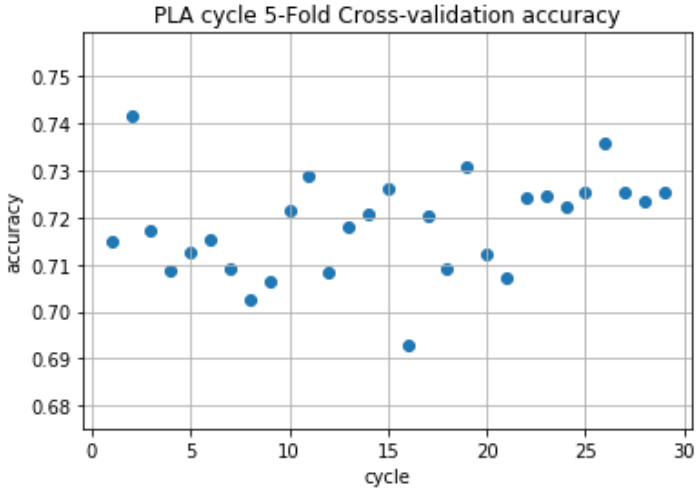


可以看出由于学习率较大，很快就接近收敛，然后准确率无较大波动。作更精细一些的散点图如下，可以得到同样的结果。

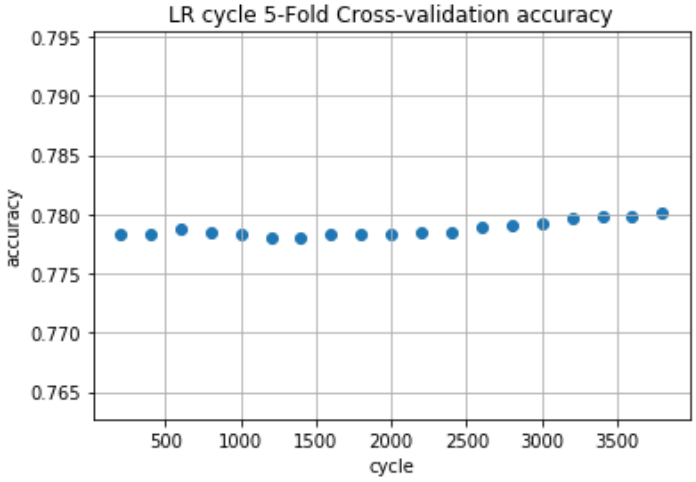


2. 使用5折交叉验证

- 感知机学习算法（学习率$\alpha = 1$）



- 逻辑回归（学习率$\alpha = 0.1$）



我们可以看出感知机学习算法由于数据集非线性可分，准确度随着遍历次数的不同而在69%-75%之间震荡，因为PLA每次遍历数据集都会发现存在误分类样本，然后以此更新权值，最后无法收敛到一个稳定的值。而调整好学习率（$\alpha = 0.1/$样本数）的逻辑回归，因为梯度下降法可以很快收敛，所以迭代次数从100开始增大变化也不明显，但因为越来越接近最优解，因此准确率总体呈现上升趋势，最终收敛到78%左右。

六. 思考题

1. 有什么手段可以使PLA适用于非线性可分的数据集?

①设置迭代次数，到一定次数就返回此时的权值w，不管是否满足所有训练集。

②使用口袋算法（Pocket Algorithm），口袋算法不像PLA那样要求严格线性分割数据集，它选择的是最优的超平面，即误分类数最少的超平面。它每次遇到误分类的样本，都会比较修正后的超平面误分类数和当前误分类数，只有当其减少才会更新权重。

③使用多层感知机组成神经网络，能够拟合任意函数。

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η过大，模型可能一开始就发散，也可能快速到达一个靠近最优解的值附近开始发散，无法收敛到最优解。

而如果学习率η过小，则模型收敛速度会变得很慢，要耗费大量的时间才能接近最优解。

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?

不合适，靠近最优解的时候梯度接近0，但是下降速度变得非常慢，或者已经不再下降，因此使用梯度模长为0作为终止条件需要耗费大量时间，或者算法永远不会停止。

判断模型收敛的方式:

①设定一个合适的阈值（如10^{-5}），当梯度的模长小于该阈值时可以认为模型已经收敛。

②设定合理的迭代次数，当到达这个迭代次数时，可以认为模型已经收敛（注意这种方法需要观察梯度的模长是否真的在减小，只有学习率合适才能通过调整迭代次数来判断模型收敛）

七. 参考文献

1) https://docs.scipy.org/doc/numpy/

2) https://pandas.pydata.org/pandas-docs/stable/

3) 《机器学习实战》Peter Harrington page74-88 北京.人民邮电出版社

ISBN: 9787115317957