中山大學
SUN YAT-SEN UNIVERSITY

实 验 报 告

人工智能实验报告–PLA 与 LR

学　　院：数据科学与计算机学院

专　　业：计算机科学与技术

学生姓名：刘斯宇

学　　号：17341110

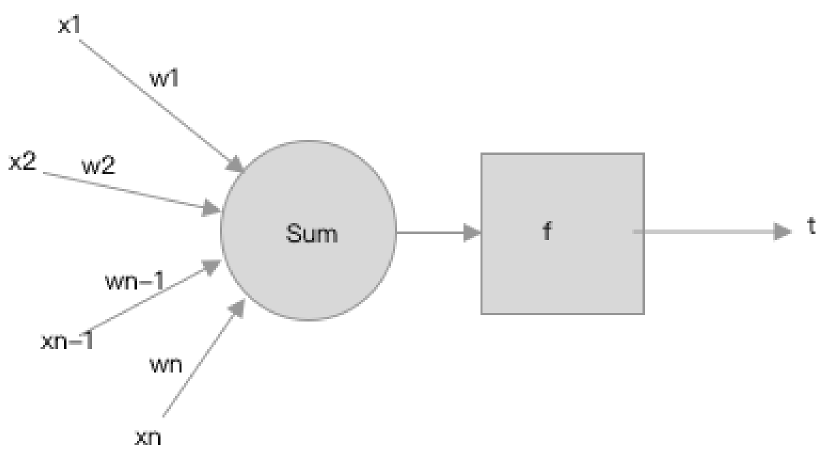
2019 年 9 月 25 日

目录

1 PLA

PLA 的全称是 Percetron Learning Algorithm，又称感知机器学习。PLA 用于解决的是对于二维或者高维的线性可分问题的分类，最终将问题分为两类——是或者不是。感知机 (Perceptrons) 也是一种人工神经网络，是一种最简单形式的前馈式人工神经网络，是一种二元线性分类器。

1.1 算法原理



如图所示，我们设

$$t = \sum_{d=1}^{n} w_d x_d$$

我们为这个“神经元”的激发值设一个阈值 threshold, 当 $t > threshold$，则判定结果为 1，否则为 -1 。我们可以重新修订一下这个表达式，使其能够用矩阵表示

$$\begin{aligned} h(x) &= sign(\textstyle\sum_{d=1}^{n} w_d x_d - threshold) \\ &= sign(\textstyle\sum_{d=1}^{n} w_d x_d + (-threshold) * (+1)) \\ &= sign(\textstyle\sum_{d=0}^{n} w_d x_d) \\ &= sign\left(\tilde{\mathbf{W}}^{\mathrm{T}} \tilde{\mathbf{X}}\right) \end{aligned}$$

其中，$x_0 = 1, w_0 = -threshold$,$W = (w_0, w_1, w_2, \ldots, w_d)$, $X = (x_0, x_1, x_2, \ldots, x_d)$

1.2 算法实现

(i) 先初始化 $W_{(0)}$, 然后根据 D 来修正 W

(ii) 遍历每一个数据 $(X_{i(t)}, Y_{i(t)})$, 直到找到 $sign\left(\tilde{\mathbf{W}}_{(t)}^{\mathrm{T}} \tilde{\mathbf{X}}_{i(t)}\right) \neq \mathcal{Y}_{i(t)}$

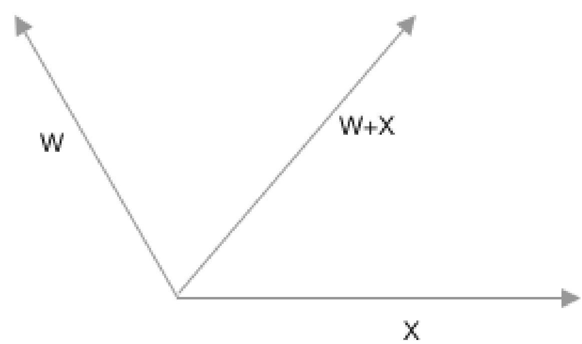
(iii) 如果找到错误的话，按照下面的方法进行 W 的修正

$$\tilde{\mathbf{w}}_{(t+1)} \leftarrow \tilde{\mathbf{w}}_{(t)} + y_{i(t)}\tilde{\mathbf{x}}_{i(t)}$$

(iv) 直到没有错误，返回最后的 W

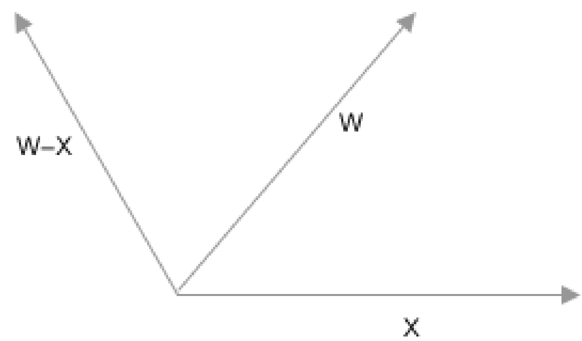
1.3 修正算法的正确性

(1) 正样例被预测为负的情况下



如果所示，在正样例被预测为负的情况下，说明两个向量的内积为负，需要调整向量使得二者的值为正,$W + X$ 会更靠近 X, 从而使得该样例的错误率降低。

(2) 负样例被预测为正的情况下



如果所示，在负样例被预测为正的情况下，说明两个向量的内积为正，需要调整向量使得二者的值为负,$W-X$ 会更靠近负, 从而使得该样例的错误率程度。

1.4 伪代码

```
        w <- vector of zeros
        for each data.X do
                newY=sign(w,X)
                if newY != Y
                        w<-w+X*Y
        while(w convergences)
```

1.5 关键代码截图

1.5.1 训练函数

方法：遍历 num 遍测试集，先计算每个测试数据与 W 的内积，如果预测错误，修正 W，否则进行下一个数据的测试。

```
import numpy as np
import matplotlib.pyplot as plt
def train(train_data,w,num):
    #num->
    for i in range(num):
        for data in train_data:
            temp=data[:-1]
```

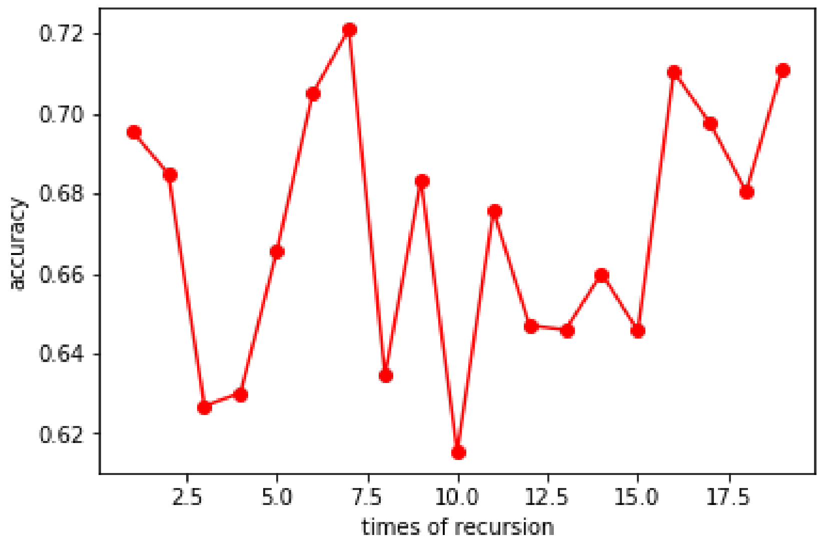
```
            res=np.dot(w,temp)
            if res>=0 and int(data[-1])==0:
                for k in range(len(w)):
                    w[k]-=data[k]
            elif res<0 and int(data[-1])==1:
                for k in range(len(w)):
                    w[k]+=data[k]
```

1.5.2 交叉验证

方法：在不同的迭代次数 (遍历每个测试数据的数目) 的情况下，将数据集按照 fold 化成训练集和测试集，分别在训练集下训练然后在测试集上面进行准确率的计算，最后计算其平均值。

```
import numpy as np
import matplotlib.pyplot as plt
for time in range(20):
        acc=0
        for i in range(fold):
            w=np.zeros(num-1)
            test_data=divide_dataset[i].copy()
            train_data=[]
            for j in range(fold):
                if j==i:
                    continue
                for data in divide_dataset[j]:
                    train_data.append(data)
            train(train_data,w,time)
            acc+=test(w,test_data)
        print('    ',time,'            ',acc/fold)
        res_list.append(acc/fold)
    x = range(20)
    plt.plot(x[1:], res_list[1:], 'ro-')
    plt.xlabel("times␣of␣recursion") #X
    plt.ylabel("accuracy") #Y
```

1.6 实验结果



2 LR

逻辑回归（Logistic Regression）是一种用于解决二分类（0 or 1）问题的机器学习方法，用于估计某种事物的可能性。

2.1 算法原理

回归模型就是预测一个连续变量 (如降水量，价格等)，那么一定会有一个疑问为什么既然有了线性回归模型没，还需要逻辑回归模型呢？这是因为直接使用线性回归的输出作为概率是有问题的，因为其值有可能小于 0 或者大于 1, 这是不符合实际情况的，逻辑回归的输出正是 [0,1] 区间。逻辑回归的主要思路就是用用一个函数 $h(x) = \frac{1}{1+e^{-x}}$ 将 $(-\infty, \infty)$ 映射到 (0,1) 之间。在这里，

- $h(-\infty) = 0$, 当加权分数无穷小，该数据属于正类别的概率为 0
- $h(0) = 0.5$, 当加权分数为 0，该数据属于任一类别的概率相同均为 0.5
- $h(-\infty) = 0$, 当加权分数无穷大，该数据属于正类别的概率为 1

在逻辑回归中，最常用的是代价函数是交叉熵 (Cross Entropy)，交叉熵是一个常见的代价函数，在神经网络中也会用到。交叉熵是对 suprise 的度量。神经元的目标是去计算函数 x→y=y(x)。但是我们让它取而代之计算函数 x→a=a(x)。假设我们把 a 当作 y 等于 1 的概率，1−a 是 y 等于 0 的概率。那么，交叉熵衡量的是我们在知道 y 的真实值时的平均 suprise 程度。当输出是我们期望的值，我们的 suprise 程度比较低；当输出不是我们期望的，我们的 suprise 程度就比较高。那么，逻辑回归的代价函数为

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\left(y^{(i)}\log h_\theta\left(x^{(i)}\right) + \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right]$$

那么为什么要定义这样的代价函数呢？对于单个的样本来讲，$J(\theta)$ 所对应的 $C(\theta)$ 为：

$$C(\theta)=y\log h_\theta(x)+(1-y)\log\left(1-h_\theta(x)\right)]$$

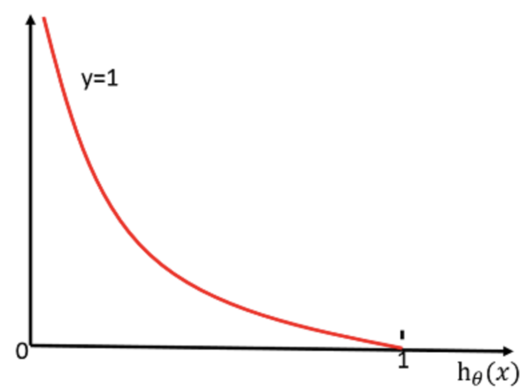
上面的方程等价于

$$C(\theta)=\left\{\begin{array}{l}-\log\left(h_\theta(x)\right),y=1\\-\log\left(1-h_\theta(x)\right),y=0\end{array}\right.,where:h_\theta(x)=\frac{1}{1+e^{-\theta^Tx}}$$

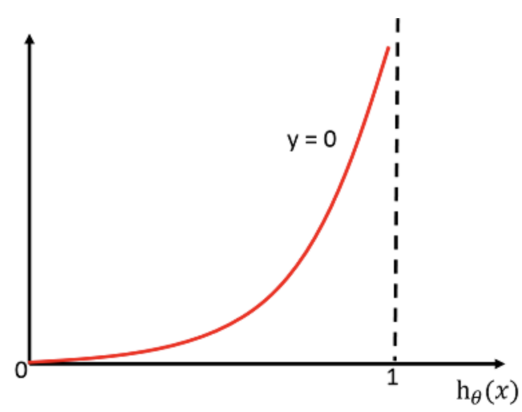
当 $y=1$ 时，

$$C(\theta)=-\log\left(h_\theta(x)\right)$$

其函数图像为



从图中可以看出，$y=1$，当预测值 $h_\theta(x)=1$ 时，可以看出代价 $h_\theta(x)=\frac{1}{1+e^{-\theta^Tx}}$ 函数 $C(\theta)$ 的值为 0，这正是我们希望的。如果预测值 $h_\theta(x)=1$ 即 $P(y=1|x;\theta)=0$, 意思是预测 $y=1$ 的概率为 0，但是事实上 $y=1$ ，因此代价函数 $C(\theta)=\infty$, 相当于给学习算法一个惩罚。同理，我们也可以画出 $y=0$ 的时候，$C(\theta)$ 的图像



梯度下降中的梯度指的是代价函数对各个参数的偏导数，偏导数的方向决定了在学习过程中参数下降的方向，学习率（通常用 表示）决定了每步变化的步长，有了导数和学习率就可

以使用梯度下降算法（Gradient Descent Algorithm）更新参数了, 即求解使 $J(\theta)$ 最小的参数 θ

$$
\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) & =\frac{\partial}{\partial \theta_j}\left[-\frac{1}{m} \sum_{i=1}^m\left[y^{(i)} \log \left(h_\theta\left(x^{(i)}\right)\right)+\left(1-y^{(i)}\right) \log \left(1-h_\theta\left(x^{(i)}\right)\right)\right]\right] \\
& =-\frac{1}{m} \sum_{i=1}^m\left[y^{(i)} \frac{1}{h_\theta\left(x^{(i)}\right)} \frac{\partial}{\partial \theta_j} h_\theta\left(x^{(i)}\right)-\left(1-y^{(i)}\right) \frac{1}{1-h_\theta\left(x^{(i)}\right)} \frac{\partial}{\partial \theta_j} h_\theta\left(x^{(i)}\right)\right] \\
& =-\frac{1}{m} \sum_{i=1}^m\left[y^{(i)} \frac{1}{h_\theta\left(x^{(i)}\right)}-\left(1-y^{(i)}\right) \frac{1}{1-h_\theta\left(x^{(i)}\right)}\right] \frac{\partial}{\partial \theta_j} h_\theta\left(x^{(i)}\right) \\
& =-\frac{1}{m} \sum_{i=1}^m\left[y^{(i)} \frac{1}{h_\theta\left(x^{(i)}\right)}-\left(1-y^{(i)}\right) \frac{1}{1-h_\theta\left(x^{(i)}\right)}\right] \frac{\partial}{\partial \theta_j} g\left(\theta^T x^{(i)}\right)
\end{aligned}
$$

because

$$
\begin{aligned}
\frac{\partial}{\partial \theta_j} g\left(\theta^T x^{(i)}\right) & =\frac{\partial}{\partial \theta_j} \frac{1}{1+e^{-\theta^T x^{(i)}}} \\
& =\frac{e^{-\theta^T x^{(i)}}}{\left(1+e^{-\theta^T x^{(i)}}\right)^2} \frac{\partial}{\partial \theta_j} \theta^T x^{(i)} \\
& =g\left(\theta^T x^{(i)}\right)\left(1-g\left(\theta^T x^{(i)}\right)\right) x_j^{(i)}
\end{aligned}
$$

So

$$
\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) & =-\frac{1}{m} \sum_{i=1}^m\left[y^{(i)}\left(1-g\left(\theta^T x^{(i)}\right)\right)-\left(1-y^{(i)}\right) g\left(\theta^T x^{(i)}\right)\right] x_j^{(i)} \\
& =-\frac{1}{m} \sum_{i=1}^m\left(y^{(i)}-g\left(\theta^T x^{(i)}\right)\right) x_j^{(i)} \\
& =\frac{1}{m} \sum_{i=1}^m\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right) x_j^{(i)}
\end{aligned}
$$

其实代价函数还有一个其他的理解，我们考虑整个训练集的似然函数：

$$
likelihood=\prod_{i=1}^N P\left(\mathrm{y} \mid x_i\right)=\prod_{i=1}^N h\left(x_i\right)^{y_i}\left(1-h\left(x_i\right)\right)^{1-y_i}
$$

取对数之后，便可以得到：

$$
\begin{aligned}
L(\theta) & =-log(likelihood) \\
& =-\sum_{i=1}^N\left(y_i \log \left(h\left(x_i\right)\right)+\left(1-y_i\right) \log \left(1-h\left(x_i\right)\right)\right)
\end{aligned}
$$

我们的目的是取 $L(\theta)$ 最小时的作为模型最后的参数

2.2 算法实现

- 给每一个样本的特征向量前加一维常数项 1
- 随机初始化 $d+1$ 维的权重向量 $_0$
- 计算当前梯度 $\nabla L\left(\mathbf{w}_t\right)=-\sum_{\mathbf{n}=1}^{N}\left(y_{\mathbf{n}}-h\left(x_{\mathbf{n}}\right)\right)\left(x_{\mathbf{n}}\right)$
- 根据梯度更新权重 $w_{\mathrm{t+1}} \leftarrow w_t-\eta \nabla L\left(w_t\right)$
- 重复上述操作，知道 W 收敛

2.3 伪代码

```
w <- vector of zeros
for each data.X do
        newY<-1/(1+exp(-wX))
        if newY != Y
                w<-w+X*Y
while(w convergences)
```

2.4 关键代码截图

2.4.1 训练函数

方法：遍历 num 遍测试集，先计算每个测试数据与 W 的内积，再计算 $h(x)$, 如果预测错误，修正 W, 否则进行下一个数据的测试。

```
import numpy as np
import matplotlib.pyplot as plt
def train(train_data,w,num):
    #num->
    for i in range(num):
        for data in train_data:
            temp=data[:-1]
            res=np.dot(w,temp)
            if res>=0 and int(data[-1])==0:
                for k in range(len(w)):
                    w[k]-=data[k]
            elif res<0 and int(data[-1])==1:
                for k in range(len(w)):
                    w[k]+=data[k]
```

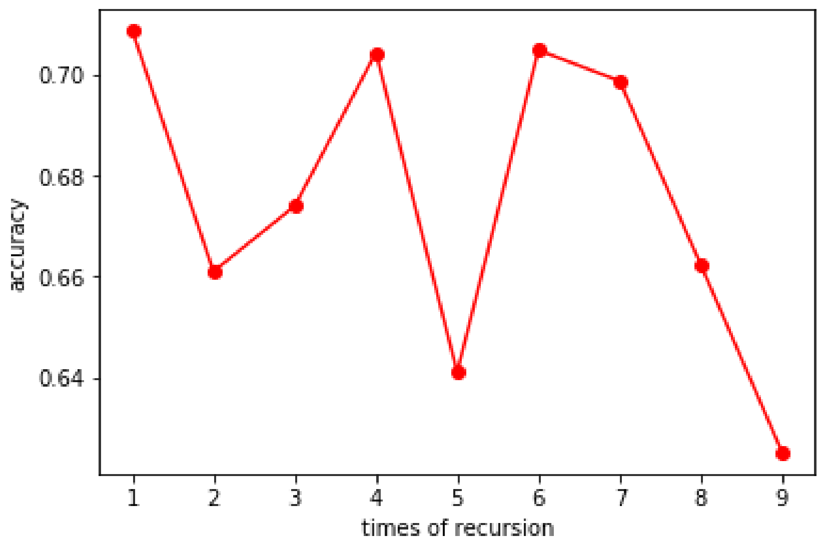
2.4.2 交叉验证

方法：在不同的迭代次数 (遍历每个测试数据的数目) 的情况下，将数据集按照 fold 化成训练集和测试集，分别在训练集下训练然后在测试集上面进行准确率的计算，最后计算其平均值。

```
if __name__ == "__main__":
    dataset=load('train.csv')
    #print(dataset)

    fold=5

    divide_dataset=[]

    index=0

    d=len(dataset)/fold

    num=0
    for data in dataset:
        if num==0:
            divide_dataset.append([])
        divide_dataset[index].append(data)
        num+=1
        if num==d:
            num=0
            index+=1
    num=len(dataset[0])
    res_list=[]
    for time in range(10):
        acc=0
        for i in range(fold):
            w=np.zeros(num-1)
            test_data=divide_dataset[i].copy()
            train_data=[]
            for j in range(fold):
                if j==i:
                    continue
                for data in divide_dataset[j]:
                    train_data.append(data)

```

```
        train(train_data,w,time)
        acc+=test(w,test_data)
    print('    ',time,'            ',acc/fold)
    res_list.append(acc/fold)
x = range(10)
plt.plot(x[1:], res_list[1:], 'ro-')
plt.xlabel("times␣of␣recursion") #X
plt.ylabel("accuracy") #Y
```

2.5 实验结果



3 实验总结

这次实验还是比较的简单的，很快就做完了，但是这个实验我的收获还是挺大的，主要在我知道了逻辑回归和线性回归的区别，以及为什么需要有逻辑回归，同时也知道了代价函数的定义，已经交叉熵的含义。

4 思考题

4.1 有什么手段可以使 PLA 适用于非线性可分的数据集？

可以多加一层神经网络，其中的激活函数用一个非线性函数就可以了。

4.2 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。

学习率过大，即下降的快，步子大，那么你很可能会在某一步跨过最优值，从而导致不收敛，学习率过小，可能导致长时间无法收敛。

4.3 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，因为一般很难找到梯度的模长为 0 的那个点。如果要找到的话，也需要花费大量的时间，没有意义。

一般采用下面的方法来判断收敛：

- 定义一个合理的阈值，当两次迭代之间的差值小于该阈值时，迭代结束。
- 设置一个大概的迭代步数，比如 1000 或 500，梯度下降法最终的迭代肯定会收敛，只要达到相应迭代次数，多了也没关系。