中山大学数据科学与计算机学院

移动信息工程专业-人工智能

本科生实验报告

(2017-2018 学年秋季学期)

课程名称：Artificial Intelligence　　任课教师：饶洋辉　　**批改人(此处为 TA 填写)：**

年级+班级	15C1	专业（方向）	移动信息工程（物联网）
学号	15352126	姓名	黄健平
电话	18312162661	Email	1041915206@qq.com
开始日期	2017/10/24	完成日期	2017/10/25

一、 实验题目

用感知机学习算法解决二元分类问题。

二、 实验内容

1. 算法原理

a) 感知机学习算法

感知机学习算法是基于线性回归的最小二乘法演化而来的二元分类算法，只能适用于线性可分的数据集，而且是可以证明能在有限的迭代中得到一组系数，使得相应的超平面将线性可分的样本点完全二分。

超平面是平面中的直线、空间中的平面的推广，例如二维空间中满足二元一次方程 ax+by=m 点的全体就构成了二维空间的超平面，三维空间中满足 ax+by+cz=m 点的全体就构成了三维空间的超平面，同理满足 $w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + w^{(3)}x^{(3)} + \cdots + w^{(n)}x^{(n)} = 0$ 点的全体就构成了 n 维空间的超平面。

二元分类问题就是将特征向量是 n 维的样本点代入训练集得到的超平面方程，得到的结果如果大于0就分为第一类，如果小于0就分为第二类。因此可以定义一个符号函数：

$$y = \text{sign}\left(\sum_{i=1}^{n} w^{(i)}x^{(i)} - \theta\right) = \text{sign}\left(\sum_{i=0}^{n} w^{(i)}x^{(i)}\right)$$

于是可以知道如果 y 大于 0 就分为第一类，如果 y 小于 0 就分为第二类。问题转换为求解系数向量 w。

求解系数向量 w 的思想源自于线性回归的最小二乘法。最小二乘法用于求解线性回归直线的系数，使得利用该直线得到的预测值$\hat{y}$与实际值 y 的差最小。这就引入了损失函数。

b) 损失函数

定义损失函数使得损失函数极小化是很多机器学习模型求解参数的常用方法。根据李航博士在《统计学习方法》一书中所述，感知机采用的损失函数是误分类点到超平面 S 的总距离。空间 $R^{(n)}$ 中任一点 x_0 到超平面 S 的距离是$\frac{|w \cdot x_0 + b|}{\|w\|}$（可以通过归纳法证明）。对于误分

类的数据(x_i, y_i)来说有 $-y_i(w \cdot x_i + b) > 0$，因此误分类点 x_i 到超平面 S 的距离是$-\frac{y_i(w \cdot x_i + b)}{\|w\|}$。如果超平面 S 的误分类点集合为 M，那么所有误分类点到超平面的总距离为$-\frac{\sum_{x_i \in M} y_i(w \cdot x_i + b)}{\|w\|}$。如果去掉$\frac{1}{\|w\|}$，最终得到的损失函数为$L(w, b) = -\sum_{x_i \in M} y_i(w \cdot x_i + b)$，其中 M⊆N，N 是全部样本点的集合。至于为什么可以去掉分母，有两个原因：①感知机是误分类驱动的，只关心误分类的个数，而不关心误分类的误差距离，因此最终损失函数的目标是 0，去掉分母对最终目标没有影响；②w 的方向向量其实是超平面的法向量，没去掉分母之前该损失函数计算的是误分类点到超平面的垂直距离，去掉分母之后计算的是函数距离，函数距离是对垂直距离的等比例放大，也就是说垂直距离大的样本点，函数距离会更大，而垂直距离小的样本点，函数距离会更小，因此去掉分母后计算变得简单却不影响对误分类点误差的定性判断。

为了简便运算，将 b 当成 w 的第 0 维数据，最终得到感知机的损失函数：

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

c) 梯度下降法

得到损失函数后，参数 w 便可以通过方程$w = \arg\min(L(w))$求解。最好的情况就是让损失函数值为 0，参照最小二乘法的思想，只要误分类样本点的数量大于特征向量的维数且误分类样本点是线性可分的，那么总会求出一组满足条件的 w。但是用最小二乘法求解有很多问题：①误分类样本点的数量不能保证，谁知道初始的误分类样本点有多少个？②实际中的样本点集通常不是完全线性可分的，总有噪声点的干扰。

最终只能退而求其次，让损失函数值最小。于是感知机学习算法最终转化为目标函数最优化问题，用起了最优化算法？个人觉得不是这样的，感知机学习算法是机器学习中最基础和最简单的算法，如果加入一堆复杂的最优化算法，便会违背感知机学习算法产生的初衷——简单有效。所以最好的选择就是用最简单的优化算法——梯度下降算法。

梯度下降法是沿梯度方向寻找最小值，因为在向量微积分中，标量场某一点的梯度指向标量场中增长速度最快的方向，梯度的值就是这个变化率的大小，所以梯度下降法就是要沿着梯度的负方向来求最小值。因此感知机损失函数的梯度为$\nabla_w L(w) = -\sum_{x_i \in M} y_i x_i$，运用梯度下降法，w 的更新公式为$w = w - \eta \nabla_w L(w) = w + \eta \sum_{x_i \in M} y_i x_i$，η是步长，又称为学习率。仔细观察这条梯度下降公式可以发现每次 w 的更新都涉及到整个训练集的数据，更新速度非常缓慢。因此有必要将单次迭代样本数从整个训练集替换为训练集的子集或单个样本(即$w = w + \eta y_i x_i$)。其实这就是四种经典类型的梯度下降法：batch GD、mini-batch GD、SGD 和 online GD。参考资料：http://www.cnblogs.com/richqian/p/4549590.html。

至此，感知机学习算法的基本框架都已经解释完毕了，更深层次的内容已超出本人的能力范围。

d) 对偶形式

李航博士在《统计学习方法》一书中还给出了感知机学习算法的对偶形式，因为在后面的优化算法中需要用到，本人特别不要脸地摘抄一下结论。

对偶形式的基本想法是，将 w 和 b 表示为实例 x_i 和标记 y_i 的线性组合的形式，通过求解其系数从而得到 w 和 b。从 w 和 b 更新公式($w=w+\eta y_i x_i$，$b=b+\eta y_i$)可以看出逐步修改 w，b，n 次后 w，b 关于（x_i, y_i）的增量分别是$\alpha_i y_i x_i$和$\alpha_i y_i$，这里的 $\alpha_i=n_i\eta$，n_i 记录着样本实例点

的更新次数。最后 w，b 分别表示为： $\mathrm{w}=\sum_{i=1}^{N}\alpha_i y_i x_i$，$\mathrm{b}=\sum_{i=1}^{N}\alpha_i y_i$，*感知机模型变为：*

$$\mathrm{f(x)}=\mathrm{sign}(\sum_{i=1}^{N}\alpha_i y_i x_i\cdot x+b)$$

把 b 看成第 0 维数据，x_i 变成增广向量，简化为$\mathrm{f(x)}=\mathrm{sign}(\sum_{i=1}^{N}\alpha_i y_i x_i\cdot x)$

对比之前的感知机模型：$\mathrm{f(x)}=\mathrm{sign}(\mathrm{w}\cdot x)$，看似变复杂了，其实是变简单了。①首先假设当前梯度下降用的是误分类样本 i，w 的更新公式：$w=\mathrm{w}+\eta y_i x_i$，更新的是一个向量，但是$\alpha_i$的更新公式：$\alpha_i=\alpha_i+\eta$，更新的是一个标量；②其次在重新寻找误分类样本点的过程，原始算法是通过判断$y_k(\mathrm{w}\cdot x_k)<0$，这其中涉及两个向量内积的计算。但对偶算法是通过判断$y_k(\sum_{j=0}^{N}\alpha_j y_j x_j\cdot x_k)<0$，又可以写成$y_k(\sum_{j*=0}^{N}\alpha_{j*}y_{j*}x_{j*}\cdot x_k+\eta y_i x_i\cdot x_k)$，可以看出$\sum_{j*=0}^{N}\alpha_{j*}y_{j*}x_{j*}\cdot x_k$已经在上一次更新中计算得到，因此只需要再加上$\eta y_i x_i\cdot x_k$即可计算出括号中的值。最终可以发现所有的计算都转化为样本集实例之间内积的计算，只要预先计算实例间的内积并以矩阵的形式存储，那么就可以通过查表的方式快速得到结果。这个矩阵叫做 Gram 矩阵：

$$\mathrm{G}(x_1,x_2,\dots,x_N)=\begin{bmatrix}x_1\cdot x_1 & x_1\cdot x_2 & \dots & x_1\cdot x_N\\ \vdots & \vdots & \dots & \vdots\\ x_N\cdot x_1 & x_N\cdot x_2 & \dots & x_N\cdot x_N\end{bmatrix}$$

对偶式算法对于随机梯度下降算法没有多大用处，因为这种算法每更新一次系数，都不会遍历样本集统计误分类样本点，而是随机选择一个误分类样本点，这样导致程序运行过程中可能没有计算$\sum_{j*=0}^{N}\alpha_{j*}y_{j*}x_{j*}\cdot x_k$。反之显然，对偶式算法可以用来提升 PLA 口袋算法的速度，因为它每次都需要遍历样本集统计为误分类。

2. 伪代码

a) PLA 原始算法

```
/*x and w is augmented matrix */
w ← w0 //initial w
for i←0 to times
   xk←arg_uneq_x(sign(w · xk), yk)
   w←w+ykxk
end
return w
```

b) PLA 口袋算法

```
/*x and w is augmented matrix */
w ← w0 //initial w
opt_w← w0 //opt_w is the optimal w
for i←0 to times
   xk←arg_uneqx(sign(w · xk), yk)
   w←w+ykxk
   for j←0 to N //N is the quantity of samples
      if sign(w · xj)=yj
         then acc_num += 1
```

```
    end
    if acc_num > opt_acc_num
        then acc_num = opt_acc_num
        opt_w = w
end
return opt_w
```

3. 关键代码截图

a) PLA 原始算法

```
w = [1] * vectorLen; #初始化w
#方法一：原始算法
K = 500 #迭代次数
for trainList in trainMatrix:
    ti = trainMatrix.index(trainList)
    tmp = sum([x * y for x, y in zip(trainList, w)])
    if tmp * trainResList[ti] < 0: #找到误分类的样本点
        w = [x + alpha * trainResList[ti] * y for x, y in zip(w, trainList)] #更新w
        K -= 1
        if K == 0:
            break
```

b) PLA 口袋算法

```
opt_w = []
w = [0] * len(trainList); #初始化w
K = 500 #迭代次数
for trainList in trainMatrix:
    ti = trainMatrix.index(trainList)
    tmp = sum([x * y for x, y in zip(trainList, w)])
    if tmp * trainResList[ti] <= 0: #找到误分类的样本点
        w = [x + alpha * trainResList[ti] * y for x, y in zip(w, trainList)] #更新w
        tmp_acc_num = 0
        for trainList2 in trainMatrix:
            ti2 = trainMatrix.index(trainList2)
            Res = sum([x * y for x, y in zip(trainList2, w)])
            if Res * trainResList[ti2] > 0: #这里为了让结果稍微好看，让TP的影响系数大一点
                if trainResList[ti2] == 1:
                    tmp_acc_num += 2
                if trainResList[ti2] == -1:
                    tmp_acc_num += 1
        if opt_acc_num < tmp_acc_num:
            opt_acc_num = tmp_acc_num
            opt_w = w #更新全局最优w
        K -= 1
        if(K == 0):
            break
```

4. 创新点&优化

实现了 PLA 算法的对偶形式，用于加速口袋算法。

三、 实验结果及分析

1. 实验结果展示示例

(附:结果展示所用的是 ppt 上的小数据集)

✧ 小数据集数据展示

小数据集:

编号	特征1	特征2	标签
Train1	-4	-1	+1
Train2	0	3	-1
Test1	-2	3	?

✧ 人工计算过程

步骤1：样本数据加常数项1

train1: $x_1 = \{1, -4, -1\}$ $y_1 = +1$

train2: $x_2 = \{1, 0, 3\}$ $y_2 = -1$

test1: $x_3 = \{1, -2, 3\}$ $y_3 = ?$

步骤2： 初始化向量$\mathbf{w} = \{1, 1, 1\}$

步骤3： 计算$\text{sign}(\mathbf{w}^T x_1) = -1 \neq y_1 \rightarrow$ train1错误

更新$\mathbf{w}$得$\mathbf{w} = \mathbf{w} + y_1 x_1 = \{2, -3, 0\}$

计算$\text{sign}(\mathbf{w}^T x_2) = +1 \neq y_2 \rightarrow$ train2错误

更新$\mathbf{w}$得$\mathbf{w} = \mathbf{w} + y_2 x_2 = \{1, -3, -3\}$

计算得$\text{sign}(\mathbf{w}^T x_1) = y_1$且$\text{sign}(\mathbf{w}^T x_2) = y_2$

预测全正确，停止迭代

预测： 计算$\text{sign}(\mathbf{w}^T x_3) = -1$，所以$y_3$预测为-1

✧ 程序计算结果

```
========== RESTART: D:\浅夏忆\
ab3\lab3.py ==========
accuracy_1 = 1.0
F1_1 = 1.0
w = [1.0, -3.0, -3.0]
```

由上图可以看出程序计算的结果与手算的结果相同，说明程序是正确的。

2. 评测指标展示及分析

(附:评测指标展示所用的训练集和验证集是实验所用的大数据集，以验证集的四个评测指标来展示。)

a) PLA 原始算法

```
原始算法的结果如下：
Accuracy_1 = 83.3%
Recall_1 = 4.375%
Precision_1 = 33.333333333333333%
F1_1 = 7.7348066298342545%
TN = 826
TP = 7
FN = 153
FP = 14
```

(迭代次数为 500，步长为 1，初始化 w 为 0)

分析：原始算法迭代完之后立即停止，不在乎当前的分类正确率是多少。因此原始算法不能通过固定次数的迭代来保证正确率。

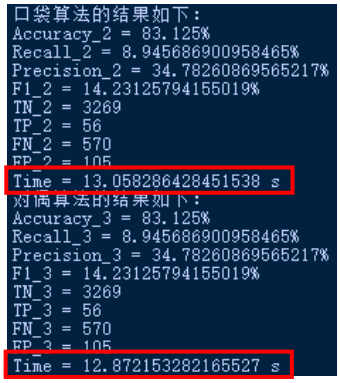
b) PLA 口袋算法

```
口袋算法的结果如下：
Accuracy_2 = 81.10000000000001%
Recall_2 = 40.0%
Precision_2 = 40.764331210191095%
F1_2 = 40.37854889589906%
F1_2 = 0.4037854889589906
TN_2 = 747
TP_2 = 64
FN_2 = 96
FP_2 = 93
```

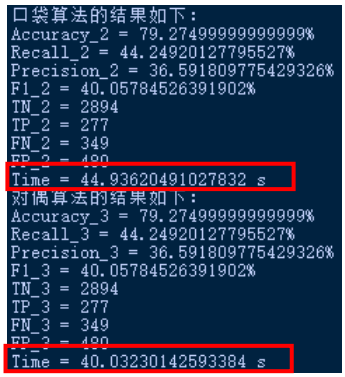
(迭代次数为 500，步长为 1，初始化 w 为 0)

分析：口袋算法比原始算法多出的只是记录当前分类效果最好的 w，因此口袋算法是为了保证正确率而设立的，迭代次数越好，找到分类效果更好的 w 几率越大。

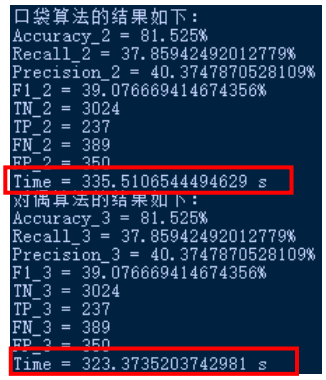
c) PLA 对偶算法



(迭代次数为 30，步长为 1)



(迭代次数为 100，步长为 1)



(迭代次数为 500，步长为 1)

分析：PLA 对偶算法和原始算法结果是一样的，只是优化了运行时间，迭代次数越多，优化效果越明显。PLA 对偶算法计算过程中不再需要计算向量的内积，所有向量的计算都转化为了标量的运算，能极大地提高运行速度。但是有个缺陷就是开始必须要计算一次样本集的 gram 矩阵进行保存。（跑的时候忘记将训练集改成验证集了，不过没关系，比较的是时间，正确率就无视吧）

四、 思考题

1. 有没有其他的手段可以解决数据集非线性可分的问题?

答： PLA 本来就不适用于非线性可分的数据集，可以采用支持向量机、人工神经网络算法或引入 sigmoid 函数映射的逻辑斯蒂回归。如果强行用 PLA 解决非线性可分的数据集，需要将特征向量升维，即将 $x=(x^{(0)}, x^{(1)}, \cdots, x^{(n)})^T$ 升维成 $x=(x^{(0)}x^{(0)}, x^{(1)}x^{(1)}, \cdots, x^{(n)}x^{(n)}, x^{(0)}x^{(1)}, \cdots, x^{(0)}x^{(n)}, x^{(1)}x^{(2)}, \cdots, x^{(1)}x^{(n)}, \ldots, x^{(n-1)}x^{(n)})^T$，同理 w 也要从 n+1 维升成(n+2)(n+1)/2 维，这样分隔面就从超平面变成了超曲面，实现非线性可分。

2. 请查询相关资料，解释为什么要用这四种评测指标，各自的意义是什么?

答：Accuracy 是预测正确的样本数占总样本数的比例。

Recall 是预测为 1 且正确的样本数占实际为 1 的总样本数的比例。

Precision 是预测为 1 且正确的样本数占预测为 1 的总样本数的比例。

F1 是 Recall 和 Precision 的调和平均数。调和平均数的特点是易受极端值的影响，且受极小值的影响比受极大值的影响更大。因此F1 的意义是综合 Recall 和 Precision 的评价效果，并且赋予较小的指标更大的权重，更客观地评价模型。