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

教学班级	1506	专业（方向）	移动信息工程（互联网）
学号	15352116	姓名	洪子淇
联系电话	13726205766	邮箱	1102229410@qq.com
开始日期	2017/10/24	完成日期	2017/10/25

一、 实验题目

1. 理解感知学习算法（PLA），了解 PLA 算法的适用场景；
2. 实现 PLA 原始算法和口袋算法；
3. 了解四个评测指标以及各自的含义。

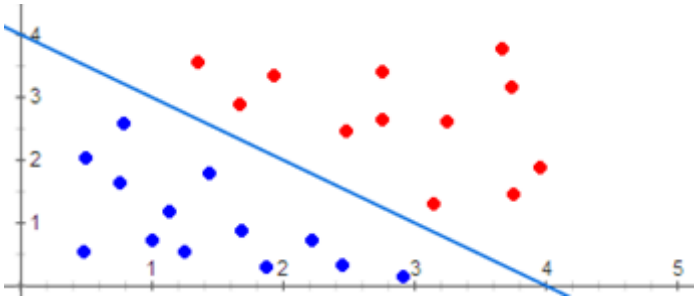
二、 实验内容

A. 算法原理

a) 原始感知学习算法和口袋算法

原始的感知学习算法是处理二分类问题的一个算法，基于线性可分的数据集，找出一个超平面将样本进行分类。

（插播一下二分类问题）在二维空间很容易对样本进行直观上的映射，x，y 分别表示样本的特征向量，那么就可以在平面上表示所有的样本数据，直观上存在着一条线可以对样本进行二分类。



（图[1]）

推广到 n 维的特征向量，那么就可以映射到一个 n 维度的空间，并且存在着一个超平面对 n 维的空间进行切割实现对样本的二分类。那么就可以定义一个 n+1 维的特征向量 x = {1，x1，x2，x3，....，xn}和一个 n+1 维的权重向量 w = {w0，w1，w2，....，wn}来表示这个超平面，超平面的表达公式为：

$$w0 + w1 * x1 + w2 * x2 + \cdots + wn * xn = 0$$

超平面将 n 维的样本数据划分为两类，一类为正样本，另一类为负样本。定义一个符号函数 sign（x）{ return x > 0 ? +1 : -1}，预测的 y 就可以表示为：

$$y = \text{sign}\left(\sum_{i=0}^{n} wi * xi\right)$$

原始的感知学习机是随机定义权重向量，遍历训练集，一旦 $\hat{y} != y$ 那么就根据 wi+1 <— wi + y*x 进行更新，直到所有的预测都正确。由于现实生活中真正线性可分的模型是极少的，所以感知学习机需要设置通过设置阈值或者迭代次数的方法停止感知。

口袋算法是基于原始感知机上的一种改进。由于原始感知机得到的是最后的 w，对于非线性模型来说，更改后的 w 不一定会比原来的 w 好，所以需要在更改 w 的时候需要对比较好的进行保存，遍历结束后就可以得到遍历过程中最好的 w。

[基于原始的感知学习算法和口袋算法的更新 w 的方式，这里可以通过随机预测样本数据并且通过阈值进行优化（写在这里是因为我没有实际实现，预测效果理论上应该会有所提高）]

b) 随机梯度下降与批量梯度下降算法

在详细分析算法之前，先引入损失函数的定义：损失函数是指一种将一个事件（在一个样本空间中的一个元素）映射到一个表达与其事件相关的经济成本或机会成本的实数上的一种函数[摘自百度百科]。通俗地将就是损失函数就是一个度量拟合的依据。

在线性回归中，损失函数通常为样本标准值与预测值的差取平方。

$$J(w) = \frac{1}{2}\sum_{i=1}^{M}(h_w(x^i) - y^i)\text{^}2$$

其中 h(x)表示样本的预测值，那么另损失函数最小，就意味着拟合程度越高，模型越好，为了求出极小值，对损失函数进行偏导，使得权重向量根据下降最快的方向进行更新，往往在更新的时候胡设置一个步长，步长的大小表示沿着梯度下降的程度。

$$w_j = w_j - \alpha\frac{\partial}{\partial w_j}J(w)$$

其中，α是学习效率，当α过大的时候可能会越过最小值，太小就会导致收敛速度较慢。

随机梯度下降算法和批量梯度算法只有一点不同，批量算法迭代更新 w 值的使用了整个样本值，算法复杂度为 O（Mn）。[注意：w 是同步更新的]。而随机梯度下降每读取一条样本就对 w 进行更新，算法复杂度为 O（n）。对于大数据，可能读取一部分数据就会使函数收敛，所有导致了可能不能收敛于最小值，而是在最小值附近震荡。

下面依次是随机梯度下降和批量梯度下降更新 w 的公式：

$$w_j = w_j - \alpha(h_w(x^i) - y^i)x_j^i$$

$$w_j = w_j + \alpha\sum_{i=1}^{M}(h_w(x^i) - y^i)x_j^i$$

随机梯度下降和批量梯度下降的方法对于原始 PLA 的优化在于收敛速度快，迭代次数较少就可以找到拟合较好的权重向量。不过这是针对线性回归问题的，不能处理非线性模型。由于本次实验中给出的训练集不是线性可分的，所以下面考虑了通过逻辑回

归来进行优化。参考资料：https://www.tuicool.com/articles/MRbee2i

（虽然都写了，包括下面的逻辑回归效果都比较差，如果有时间我就进行评测，没有时间就.....当我进行理论分析吧）

c) 逻辑回归

逻辑回归其实是一种分类方法，主要用于二分类。构造了一个预测函数 h：

$$h_w(x) = g(w^T x) = \frac{1}{1 + e^{-w^T x}}$$

利用了 Sigmoid 函数的特性，自变量大于 0 的时候收敛到 1，小于 0 的时候收敛到 0，换句话说就是自变量越大，是 1 的概率也就越大。逻辑回归中一般采用对数损失函数。

$$\text{Cost}(h_w(x), y) = \begin{cases} -\log(h_w(x)) & if\ y = 1 \\ -\log(1 - h_w(x)) & if\ y = 0 \end{cases}$$

同样利用梯度下降去收敛到损失函数的最小值，根据梯度下降方向更新权重向量，具体推导过程可以参考 http://blog.csdn.net/pakko/article/details/37878837 ，这里就不展开了。最后得到权重向量的更新公式：

$$w_j = w_j + \frac{\alpha}{M} \sum_{i=1}^{M} (h_w(x^i) - y^i) x_j^i$$

虽然和线性回归中的更新公式一样，可是由于损失函数的不同，他们有不一样的性质。

B. 伪代码

a) 原始感知学习算法

```
Get X and y; //the data matrix
initial w;       //weight vector
for i to times:
    if   sign(w^Txk) != yk    //xk means kth Sample in X
         w = w + xk*yk;
end
get the final w;
```

b) 口袋算法

```
Get X and y;
initial w;
initial pocket;
for i to times:
    if   sign(w^Txk) != yk
         w = w + xk*yk;
         check the cost(w,x); //here compare F1 norm
         if the new w if better than the order one:
              update the new w in pocket;
end
get the w in the pocket;
```

c) 逻辑回归

Get X and y;
initial w;
define the H(x) = 1/(1+e^(-x)); // forecasting function
for i to times:
  for j to n; //dimensionality of feature vector
    $w_j = w_j + \frac{\alpha}{M}\sum_{i=1}^{M}(H_w(x^i) - y^i)x_j^i$ // updata the w
  caculate the cost(X,y,new_w);
  if cost is lower:
    continue;
  else :
    shorten the α and back to loop;
    if new w − w <epsion: // it means w has convergenced
      exit the loop;
end
get w from the loop;

C. 关键代码截图（代码+注释）

a) 原始感知学习算法

```
#*****************************原始PLA算法*************************
def initialPLA(test,w=np.ones(dim),times=1000):
    for i in range(times):   #迭代次数
        c = m;
        for item in test:
            if Data.sign(sum(item['x']*w)) !=  item['y']:
                w += item['y']*np.array(item['x'])  #找到预测错误的样本就更新w
            else:
                c -= 1
        if c == 0:
            break
    return w #返回最后的w
```

b) 口袋算法

```
import common_method as cm
import numpy as np
fpath = 'Lab3/'
train = cm.getData(fpath+'train.csv', target=True)
val = cm.getData(fpath+'val.csv', target=True)
test = cm.getData(fpath+'test.csv', predict=True)

def pocketPLA(test_x,w,times):
    pocket = [-1,w]          #利用F1参数进行评测指标，对F1进行初始化
    for i in range(times):  #设置迭代次数
        for item in test_x:
            if cm.sign(sum(item['x']*wo)) !=  item['y']:
                w += item['y']*np.array(item['x'])    #与原始一样，预测不同就更新w
        (Ac,Re,Pre,F1) = cm.check(test_x,w) #遍历完样本后将w与之前的w进行比较
        if pocket[0] < F1:   #如果得到的比较好，就将w放进口袋，同时更新F1指标
            pocket = [F1,w]
    print(w)
    print('Time : ' + str(i) + '    mypocket_F1 :' + str(pocket[0]))
    return pocket           #返回口袋
```

c) **逻辑回归**

```
def logistic_regression(train_X, train_y, weights=None, times=1000):
    if weights == None:
        weights = [0.0] * train_X.shape[1]
    alpha = 1
    epsilon = 1e-10
    kp = train_y[train_y == 1].shape[0] / train_y.shape[0]
    kn = 1 - kp
    k = (kp, kn)
    w0 = np.array([weights]).transpose() # dim x 1
    p = sigmoid(np.dot(train_X, w0)) # m x 1
    J0 = logistic_cost(p[:], train_y, k)
    for t in range(times):
        # w = w0
        # for j in range(train_X.shape[1]):
        #   x_j_T = train_X[:, j:j+1].transpose()
        #   # 写成train_X[:, j]会变成一维向量
        #   w[j] = w0[j] - alpha * np.dot(x_j_T, p - train_y)
        p_y = p - train_y
        p_y[train_y == 0] *= k[0]
        p_y[train_y == 1] *= k[1] #由于模型的负值类过多，所以设置一个权重
        w = w0 - alpha * np.dot(train_X.transpose(), p_y) #每一个w的更新和整个训练集有关
        p = sigmoid(np.dot(train_X, w))
        J = logistic_cost(p, train_y, k)
        if J < J0: #没有收敛，就继续循环，并且更新损失函数以及更新之后的w
            J0 = J
            w0 = w
        else:       #损失变大了，可能越过了极小值，用原来的w，步长减半再次计算损失函数
            alpha /= 2
            #如果更新后的w与上一个w相差不大，就证明收敛了，退出循环
            if np.linalg.norm(w - w0) < epsilon * np.linalg.norm(w0):
                print('times:', t)
                break
    weights = w0.transpose().tolist()[0] #转换成list
    return weights
```

D. 创新点&优化

创新点（并没有优化效果）：实现了逻辑回归算法，并且根据样本中正负样本的比例设置损失代价。

三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

小数据集数据展示：

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

手算：w = {1,1,1}

sign（w^Tx1）= -1 （错误，更新 w)

w = w + y1*x1 = {2,-3,0}

sign（w^Tx2）= 1 （错误，更新 w)

w = w + y2*x2 = {1,-3,-3}

sign（w^Tx1）= 1 and sign（w^Tx2）= -1 （so w = {1,-3,-3} in the end)

sign（w^Tx3）= -1 （预测为-1）

使用了原始的 PLA 算法得到 w 的结果如下：

```
PS C:\Users\小柒\Desktop\大三上\人工智能\实验\Lab3 PLA> py .\PLA_initial_15352116.py
[ 1. -3. -3.]
```

由输出结果可知，模型与手算结果得到的答案一致。

2. 评测指标展示即分析（如果实验题目有特殊要求，否则使用准确率）

a) 原始感知学习算法

迭代次数 500，初始化权重向量为 1：

```
[ -2908.          -8734.2557        1490.92666       -161.769       -21499.58689999
   1438.4134      -10843.2368         97.073         11019.3628       -1782.0981
   5392.4136         77.6317        -376.074          5511.30035      -1595.41982
    886.66858    -20589.373          3290.             2172.           -125.797
    913.67131    -10238.             8895.             -362.2236       7058.75226
  -1138.986        4547.19891         234.8306         1069.8931       3409.0615
  -1075.271       -1700.90016       -1305.78714        2416.635         324.6869
 -15726.40275     -5111.22815       -8468.74795         581.8799       -639.28
  13553.3688      -7685.            -7799.4391         2291.83418      -1689.
   -635.776        6213.4311        -4459.73745       -1034.74308       7485.3518
   -305.           2769.           -11735.8161          895.            -866.1
 -11460.79430001   8844.             7409.0726         2931.12923      -6602.5179
  -4487.2343      12673.            -3812.             6867.69056
  27351.75143998 -26066.2125     ]
************** This is the result of initial PLA ****************
train:
[TP, FN, TN, FP] :
[228, 398, 3162, 212]
Ac: 0.8475       Re: 0.364217252396       Pre: 0.518181818182      F1: 0.427767354597
validation:
[TP, FN, TN, FP] :
[55, 105, 780, 60]
Ac: 0.835        Re: 0.34375      Pre: 0.478260869565      F1: 0.4
```

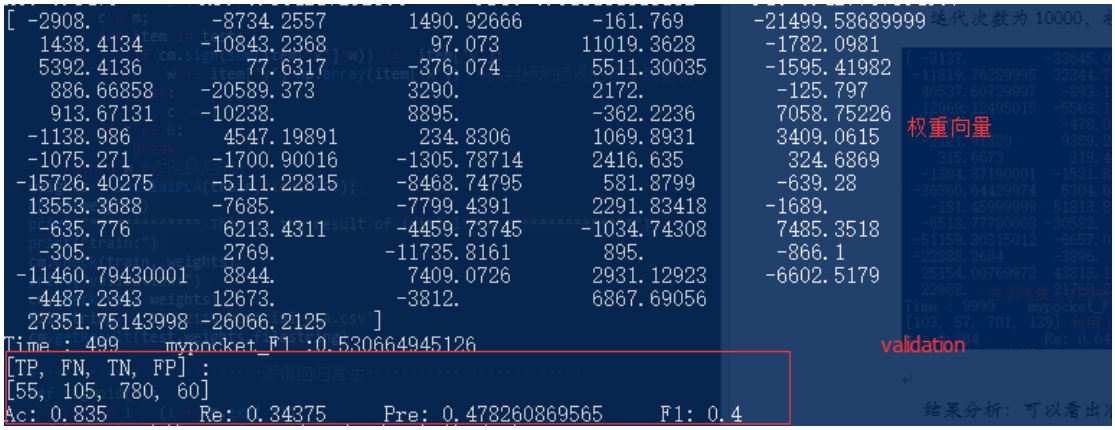
迭代次数为 1500，初始化权重向量为 1：

```
[ -3461.       -11596.03460002   5528.05523        143.058       -22655.31329995
   8708.5406     -18986.47990001   1776.9074       25698.0335       -1107.9033
  16066.4883      -271.3108        -616.6495        8330.19830001
  -2345.84326     3147.94787     -26278.45669999    5043.           3224.
   -178.668      -1267.77578      -18541.          20374.          -1199.96591
   8844.46395    -3340.05884       3312.78625       -278.5901        653.4282
   4042.76960001 -1875.7222       -1630.46827      -1698.20824      2385.561
    378.1195    -25636.87450004   -1841.58311001  -11981.09074       408.8861
   -383.22       27764.2984       -16446.          -3099.99306      -1997.38206
  -4851.          -440.1635       13528.92509998  -12369.82974      -2336.73765
  13968.9376       37.            3249.           -16686.5459        1194.
  -1110.5       -20354.15870001   27265.           16828.80930001
  12148.97426    -5687.25989999   -6103.9002       17899.           -3536.
  12715.94209001  34402.49325     -32051.4877    ]
************** This is the result of initial PLA ****************
train:
[TP, FN, TN, FP] :
[367, 259, 2906, 468]
Ac: 0.81825      Re: 0.586261980831      Pre: 0.439520958084      F1: 0.502395619439
validation:
[TP, FN, TN, FP] :
[92, 68, 724, 116]
Ac: 0.816        Re: 0.575        Pre: 0.442307692308      F1: 0.5
```

结果分析：通过比较迭代次数得到不同的准确率可知，不是迭代次数越多就会得到越高的准确率，所以在基于效果差不多的原始 PLA 和口袋算法来说，口袋算法会比原始算法保留更好的权重向量。

b) 口袋算法

迭代次数为 500，初始权重向量为 1.



迭代次数为 10000，初始权重向量为 1

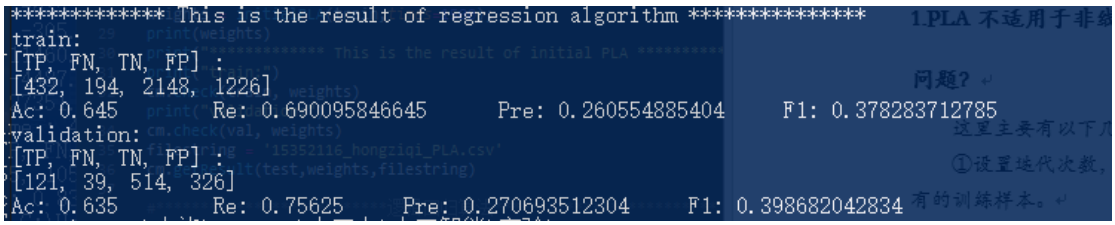
结果分析：

当迭代次数为 500 的时候，原始的 PLA 和口袋 PLA 效果一样，这是因为原始最后得到的 w 和口袋中最好的 w 值一样，准确度也比较高。

当迭代次数为 10000，可以看出准确率为 80.4%，相对比迭代 500 次，准确度有所下降，F1 指数有所提高。根据对评测指标进行分析，在保证了高准确度的情况下，这个模型在召回率比精确率有更加好的属性。

c) **逻辑回归算法**

准确率上，比原始的还要差很多。。。。(是一个比较失败的实现)



不过分析可以观测到，这个模型可以在要求召回率比较高的事件上上模拟效果。Re 指标下面有详细分析。

四、思考题

1.PLA 不适用于非线性的问题，有什么其他的手段可以解决数据集非线性可分的问题?

这里主要有以下几种方法:

①设置迭代次数，到了一定的程度就返回此时的 w，不管它到底有没有完全地划分开所有的训练样本。

②口袋算法：在每一次更改了 w 的时候就对整一个训练样本进行遍历，如果更改之后的得到更好指标，就把此时更改之后的 w 放入口袋中。

2.查询相关资料，解释四种指标各自的意义以及用四种评测指标的理由?

四个评测指标分别是 Accuracy（准确率）、Precision（精确率）、Recall（召回率）、F1（F 值），对于二分类可以划分四个参数，TP、FN、TN、FP，分别表示**真正例**（本来为 1 的预测为 1 的个数），**假正例**（本来为 1 的预测为-1 的个数），**真反例**（本来为-1 的预测为-1 的个数），**假反例**（本来为-1 的预测为 1 的个数）

真实情况	预测情况	
	正例	反例
正例	TP	FN
反例	FP	TN

而下面是四个评测指标的计算方式:

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN} \qquad \text{Recall} = \frac{TP}{TP+FN}$$

$$\text{Precision} = \frac{TP}{TP+FP} \qquad \text{F1} = \frac{2 * Precision * Recall}{Precision * Recall}$$

准确率虽然常用，不过不能满足所有任务的需求，这时候就需要引入其他的性能指标来衡量。例如在医学疾病的阴阳性问题，我们更关心的不是所有样本中预测正确的数量。

对于病人来说，他们更加关注在所有预测为阳性疾病下真正患有阳性疾病的概率，如果评测指数为 1 那么就证明如果被预测为患有阳性疾病，那么真实阳性疾病发生的概率很高。

对于医生来说，他们更加关注的是所有真实患有阳性疾病中被预测为患有阳性疾病的概率，该概率越高就证明了他们的预测水平越高。

用通俗地话来说，精确率也可以说为查准率，召回率可以说成查全率来理解。在不同的应用方面，对于这两个的指标的重视程度也会有所不同。对于商品推荐系统来说，查准率更为重要。而对于罪犯的信息检索系统，查全率更为重要。

精确率和召回率是一对相对矛盾的度量，一般生活中，精确度高时，召回率往往会偏低；召回率高时，精确率会偏低。（一些简单的任务中，这两个的概率才都会比较高。）

往往对于学习机来说，我们需要对学习机的性能进行比较。而 F1 是这两个测评指标的调和平均数，这是综合考虑精确度、召回率的性能度量的一个指标（还有其他指标，这是最常用的一个。）

|---------- 如有优化，重复 1，2 步的展示，分析优化后结果 ----------|

实验问题记录：

图[1]来自：http://blog.csdn.net/hulingyu1106/article/details/51212632