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	**M1**	专业（方向）	**移动信息工程（互联网方向）**
学号	**15352102**	姓名	**韩硕轩**

一、 实验题目

感知机算法（initial & pocket）

二、 实验内容

1. 算法原理

对于一组 n 维的向量文本和它们各自的标签，找到一个同样维度的向量 w。w 和每一个文本相乘，然后分类。目标是使得分类的结果和标签相等的文本个数尽量多。

Initial 的方法是设定一个迭代次数的上限。首先随机一个初始的 w，每次迭代都让 w 与每一个文本相乘一遍并分类，如果发现分类错误就按照下图公式更新 w，算作一次迭代，并用更新后的 w 继续往下做。结束条件是 w 能满足所有的文本，或者迭代次数达到上限。

$$w_{t+1} \leftarrow w_t + y_n x_n$$

Pocket 的方法也是先设定初始的 w 和迭代次数上限。不同之处是要另外设定一个全局最优的 w_best，保存能满足最多文本的 w。每次迭代之后，判断新的 w 能否比 w_best 满足更多的文本，如果可以就更新 w_best。

迭代结束之后，initial 用最后的 w，pocket 用 w_best 来跑验证集和测试集。

2. 伪代码

initial 算法：

```
读入 train_set，个数为 num
for I from 0 to num
    文本 i 首位插入 0
    记录 label[i]
初始化 w
设置迭代上限
while (迭代没到上限)
    for j from 0 to num
        给文本 j 分类
        if 分类错误
            迭代次数增加
```

```
                更新 w
            if w 满足所有文本
                提前结束迭代

        读入 validation_set
        for I from 0 to 验证集文本数
            用 w 给文本 i 分类
            统计 tp, tn, fp, fn
        计算 accuracy, recall, precision, f1

        读入 test_set
        for I from 0 to 测试集文本数
            用 w 给文本 i 分类
            输出结果
```

pocket 算法：

```
        读入 train_set，个数为 num
        for I from 0 to num
            文本 i 首位插入 0
            记录 label[i]
        初始化 w
        初始化 w_best
        设置迭代上限
        while (迭代没到上限)
            for j from 0 to num
                给文本 j 分类
                if 分类错误
                    迭代次数增加
                    更新 w
            for j from 0 to num
                给文本 j 分类
                if 分类错误
                    w 的错误个数加一
            if w 比 w_best 更优
                更新 w_best
            if w 最优
                提前结束迭代

        读入 validation_set
        for I from 0 to 验证集文本数
            用 w_best 给文本 i 分类
            统计 tp, tn, fp, fn
        计算 accuracy, recall, precision, f1
```

读入 test_set
for I from 0 to 测试集文本数
用 w_best 给文本 i 分类
输出结果

3. 关键代码截图（带注释）

Initial 文本读入的部分：

```
for row in train_set:
    ts[i].append(1)  #给每一个文本首部插入1
    for j in range(0,len(row)):
        ts[i].append(string.atof(row[j])) #读入是string要转成数字
    i+=1  #文本个数加一
    label.append(string.atoi(row[len(row)-1]))  #记录标签
```

读完以后记录关键数据，并初始化 w：

```
w=[]
col=len(ts[0]) #记录列数即文本维度
num=i  #记录文本个数
for i in range(0,col):
    w.append(0) #生成初始的w
```

迭代和更新 w 的过程：

```
iteration=10000   #迭代上限1w次
i=1
while i<=iteration:
    flag=1
    for j in range(0,num): #遍历每一个文本
        wsum=0
        for k in range(0,col):
            wsum += w[k]*ts[j][k]  #计算乘积
        if wsum>=0:  #分类到+1或-1
            wsum=1
        else:
            wsum=-1
        if wsum != label[j]: #如果分类错误
            flag=0
            for k in range(0,col): #就更新w
                w[k]=w[k]+label[j]*ts[j][k]
            i+=1  #这算一次迭代
    if flag == 1:  #如果已经满足所有了就
        break;     #跳出迭代否则会造成死循环
```

验证集的读入部分也是一样。不一样的是在做验证集的时候要统计 tp/tn/fp/fn：

```
if wsum!=label[i]:
    if label[i]==1:
        fn+=1
    else:
        fp+=1
else:
    if label[i]==1:
        tp+=1
    else:
        tn+=1
```

根据这四个计算准确率、精确率、召回率和 F 值：

```
accuracy=(tp+tn)*1.0/(tp+tn+fp+fn)
recall=tp*1.0/(tp+fn)
precision=tp*1.0/(tp+fp)
f1=2*precision*recall/(precision+recall)
```

最后测试集的读入和计算部分与之前相同，而输出答案的语法是这样的：

```
res1=file('res1.csv','wb')
writer=csv.writer(res1)
writer.writerow([wsum])
```

Pocket 里面相同的部分不再重复截图，不同的部分是决定最后 w 的过程：

```
num_false=0 #统计当前w不能满足的文本个数
for j in range(0,num): #遍历一遍文本
    wsum=0
    for k in range(0,col): #计算
        wsum+=w[k]*ts[j][k]
    if wsum>=0: #分类
        wsum=1
    else:
        wsum=-1
    if wsum!=label[j]: #如果不满足
        num_false+=1   #就记下来
if num_false<min_false: #新的w更优
    min_false=num_false
    w_best=w  #更新错误记录和w_best
if num_false==0: #如果已经全满足
    break  #就必须跳出因为不可能再更新了
```

其余部分就没有关键代码了，直接复制 initial 的就可以。

4. 创新点&优化（如果有）

有一些小的常数优化，没有算法复杂度上的优化。

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）

我设计了两个小的训练集。

第一个是 and 问题的：

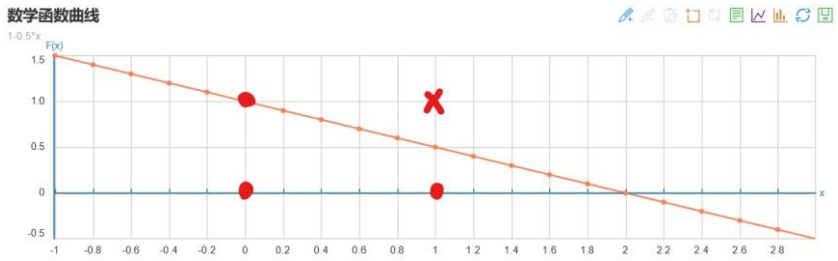
	Standard	Standard	Standard
1	0	0	1
2	1	0	1
3	0	1	1
4	1	1	-1

结果是（前者为 initial，后者为 pocket）两条直线方程一样：

```
[1, 0.0, 0.0, 1.0]
[1, 1.0, 0.0, 1.0]
[1, 0.0, 1.0, 1.0]
[1, 1.0, 1.0, -1.0]
3
[2, -1.0, -2.0]
[Finished in 0.0s]
```

```
[1, 0.0, 0.0, 1.0]
[1, 1.0, 0.0, 1.0]
[1, 0.0, 1.0, 1.0]
[1, 1.0, 1.0, -1.0]
3
2 -1.0 -2.0
[Finished in 0.1s]
```

转化成可视化的结果就是下图。可见这条直线将四个点分成了两部分。



第二个是 xor 问题的：

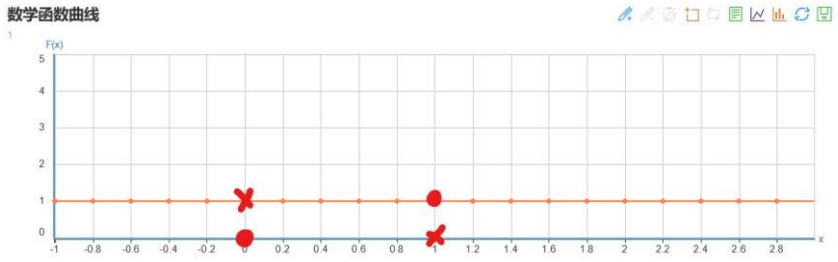
	Standard	Standard	Standard
1	0	0	1
2	1	0	-1
3	0	1	-1
4	1	1	1

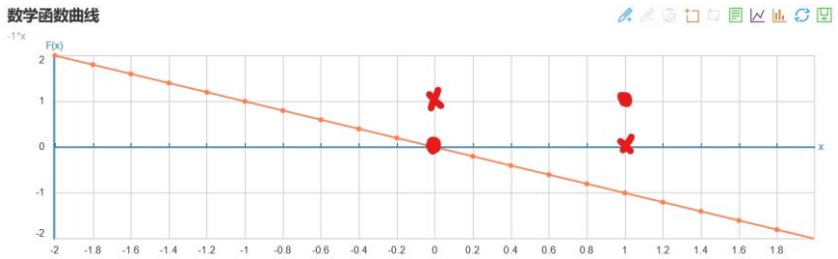
结果是（前者为 initial，后者为 pocket）：

```
[1, 0.0, 0.0, 1.0]
[1, 1.0, 0.0, -1.0]
[1, 0.0, 1.0, -1.0]
[1, 1.0, 1.0, 1.0]
3
[-1, 0.0, 1.0]
[Finished in 0.1s]
```

```
[1, 0.0, 0.0, 1.0]
[1, 1.0, 0.0, -1.0]
[1, 0.0, 1.0, -1.0]
[1, 1.0, 1.0, 1.0]
3
0 -1.0 -1.0
[Finished in 0.1s]
```

转化成可视化的结果就是下图。可见这两条直线都没能将四个点分成两部分，这四个点非线性可分。





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）

Initial迭代10000次以后跑验证集的结果截图如下。正确率0.831，表示验证集中有83.1%的文本被正确预测；召回率为0.394，表示+1的文本中有39.4%的文本被预测正确；精确率为0.467，表示被预测为+1的文本中有46.7%的文本是真的标签为+1；F值为0.427，表示算法综合效果为42.7%（F值的意义在思考题中分析）。

```
63 768 72 97
accuracy= 0.831
recall= 0.39375
precision= 0.466666666667
f1= 0.427118644068
[Finished in 1.4s]
```

Pocket迭代10000次以后跑验证集的结果截图如下。每个数据的分析和initial的相同。二者相比，pocket的各项数据均优于initial，说明pocket的预测效果更好。不过可以从时间上看出，pocket的效率比initial低了不少。

```
72 771 69 88
accuracy= 0.843
recall= 0.45
precision= 0.510638297872
f1= 0.478405315615
[Finished in 556.4s]
```

四、 思考题

1. 有什么其他的手段可以解决数据集非线性可分的问题？

答：①可以设置许多感知机，每一个的 w 值的初始化都不同，迭代完之后取平均数或者众数；②更改更新方法，用梯度下降的方法；③将线性问题转换为能够拟合的问题；④支持向量机，使得所有错误点和直线距离的和最短。

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：ppt 中看出的准确率（accuracy）表示被正确预测的标签所占比例，召回率（recall）表示所有+1 的标签中能被预测正确的个数所占比例，精确率（precision）表示所有被预测为+1的标签中正确的个数所占比例，F值将召回率和精确率1:1加权算调和平均数，综合评价了召回率和精确率。

参考网上资料以后可以更形象地总结。如果我们需要从一堆文本中推荐出其中标签为+1 的文本，召回率衡量了我们的算法能从所有真正满足需要的文本中发现多少，而精确率表示了我们发现的所有文本中有多少是真正需要的。我们总是需要二者都尽可能高，但实际上二者经常无法同时都很高。比如，我们返回所有的文本，那么召回率就是100%，但精确率就很低，而如果只找到一个

你认为最“稳”的就急着返回，那精确率就是 100%，但召回率就惨不忍睹。所以我们用 F 值，精确率和召回率的调和平均数，来综合评价算法的推荐效果。F 值越高，表示效果越好。
（参考博客：http://bookshadow.com/weblog/2014/06/10/precision-recall-f-measure/）

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

PS：可以自己设计报告模板，但是内容必须包括上述的几个部分，不需要写实验感想