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	[redacted]		
[redacted]			

一、 实验题目

感知机学习算法（PLA）

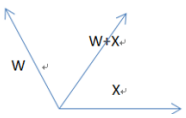
二、 实验内容

1. 算法原理

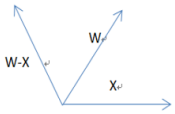
（1）PLA 算法：

PLA 算法本质上是一个线性分类器。它的目标就是找到一条直线或者是一个超平面能够将训练集分成正负两部分。在建立模型时，首先我们是任意选择一条直线(即任意设置初始 W，W 是分割线的法向量),然后通过用 W 去预测训练集中的样例，每当样例预测错误，则修正分割线，让分割线向预测错误的点靠拢（即更新 W,如下图），再从头开始预测。通过不断的调整分割线，最后找到一条直线将训练集分成两部分（即找到最优的 W）。当我们找到了最优的 W 时，我们的 PLA 模型就建立好了，我们就可以用 PLA 模型[y=sign(W*x)]去预测 validation 的结果。

正样例被预测为负的情况下：



负样例被预测为正的情况下：



在 PLA 原始算法中，我们是假设训练数据集是线性可分的。所以当出现数据集线性不可分的时候就会出现死循环的情况。我们可以通过设置迭代次数或用 PLA 口袋算法来解决这个问题。

（2）PLA 口袋算法：

PLA 口袋算法基本上和 PLA 原始算法相同，只是 PLA 口袋算法会记录一个最优的权重向量 best_w。当我们用 W 去预测训练集样本的结果时，如果预测错误，则更新 W，每次更新 W 时就和之前记录的 best_w 进行比较，比较更新的 W 分类错误的样本更少，则将用 best_w 将更新的 w 记录起来。简单来说就是我们一直握着一个比较优的权重向量 best_w，只有出现比 best_w 更优的 w 时我们才更新 best_w。其实 PLA 口袋算法有点类似于贪心算法，我们只能找到一个较优的解，而不是最优的解。

2. 伪代码

（1）PLA 算法：

```
/*作用：将一行文本切割成一个个数字*/
vector<double> get_xi(string temp){
    设置切割标签为逗号；
    for(遍历字符串 temp){
        找到出现“，”的位置；
        将这个逗号和上一个逗号之间的数字切割出来放到向量 xi 中。
    }
    返回 xi.
}
```

```
/*作用：读取 train 文件，将 train 的每一行切割成一个个单词，放入 train 中，结果 y 放入 train_y 中*/
void get_train(const char *filename){
    if（读取文件失败） 返回；
    else{
        while（一行一行地遍历 train 样本）{
            调用 get_xi 函数将每一行样本切割成一个个数字。
            给样本加上一个常数项 1
        }
    }
}
```

```
/*作用：用 w 预测 train 的每一行的结果，如果预测出错，则更新 w（w[n+1]=w[n]+x*y)*/
void renew_w(int num_of_predict){
    for（遍历 train 的每一行样本）{
        用 w 来预测每一行样本的结果：cal_y= xi[n]*w[n]。
        if（预测的结果和真实的结果不同）{
            更新 w $w_{t+1} \leftarrow w_t + y_n x_n$；
            记录迭代次数；
            将遍历的行数设置为-1，即从头开始遍历 train 样本；
        }
        当达到指定的迭代次数，遍历结束。
    }
}
```

```
/*作用：用完全更新好的 W 来预测 vali 的结果*/
void predict(){
    for（遍历 vali/test 的每一行样本）{
```

用 w 来预测每一行样本的结果。$sign(w_t^T x_n)$

if(预测的结果为+1，真实的结果为+1) TP++；

if(预测的结果为+1，真实的结果为-1) FN++；

if(预测的结果为-1，真实的结果为+1) FP++；

if(预测的结果为-1，真实的结果为-1) TN++；

}

}

/*计算四个评测指标*/

void standard(){

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

$$\text{Recall}=\frac{TP}{TP+FN}$$

$$\text{Precision}=\frac{TP}{TP+FP}$$

$$F1=\frac{2*Precision*Recall}{Precision+Recall}$$

}

（2）PLA 口袋算法：

PLA 口袋算法只有更新 W 的步骤和 PLA 算法不同，所以这里在写更新部分的伪代码。

/*作用：用 w 预测 train 的每一行的结果，如果预测出错，则更新 w（w[n+1]=w[n]+x*y) 将更新 w 的 F 值和记录的最优的 F 值比较，大则更新 best_w.*/

void renew_w(int num_of_predict){

for（遍历 train 的每一个样本）{

用 w 计算出预测结果 y.

if（预测的结果和真实的结果不同）{

更新 w;

记录迭代次数；

设置从头开始遍历，即遍历行数=0；

}

调用 predict_train（）函数用更新的 w 预测 train 样本的值，并得到 F 值；

将得到的 F 值和 best_w 的 F 值进行比较；

if(更新的 w 的 F>best_w 的 F) {

best_w=w;

best_F= F;

}

}}

3. 关键代码截图（带注释）

（1）PLA 算法：

总体思路：

首先给每一个样本前加上一个常数项 1，然后初始化权重向量 w=1,遍历所有的样本，每当找到一个预测错误的样本，即$sign(w_t^T x_n) \neq y_n$,就更新$w_{t+1} \leftarrow w_t + y_n x_n$ 。每次更新 w 就又要从头开始遍历样本，直到 w 能够把所有样本都预测正确。然后用此时得到的 w 来预测 validation 的 label。

《1》读取 csv 文件中的文本内容，并且将每一行文本的 words 切割成一个个数字。并且将 string 类型的数值转换成 double 类型。

```
/*作用：将一行文本切割成一个个数字*/
vector<double> get_xi(string temp)
{
    vector<double> xi;
    string pattern=",";
    temp+=pattern;
    int size=temp.size(),pos;
    for(int i=0;i<size;i++){
        pos=temp.find(pattern,i);
        if(pos<size){
            string s=temp.substr(i,pos-i);
            double x=stringToDouble(s);
            xi.push_back(x);
            i=pos; // i=pos+pattern.size()-1;
        }
    }
    return xi;
}
```

```
/*作用：将string类型的数据转成double型 */
double stringToDouble(const string& str)
{
    istringstream iss(str);
    double num;
    iss >> num;
    return num;
}
```

《2》往存储每一行文本切割出来的数字的向量 xi 的开头插入常数项 1,然后将 xi 的最后一样（即 label）放入向量 train_y 中。

```
while(getline(ReadFile,temp)){
    vector<double> xi;
    xi=get_xi(temp);
    xi.insert(xi.begin(),1); //插入常数项1
    train_y.push_back(xi[xi.size()-1]);//最后一项即为label
    xi.pop_back(); //抛掉最后一项
    train.push_back(xi);
}
```

《3》 用 w 来预测 train 的每一行的结果 label，

```
double cal_y=0;
// 根据w,计算出预测的label。cal_y= xi[n]*w[n]
for(int column_of_train=0;column_of_train<train_column;column_of_train++){
    cal_y+=1.0*train[row_of_train][column_of_train]*w[column_of_train];
}
// 如果计算结果>0，则预测的label为1；计算结果<0，则预测的label为-1.
if(cal_y<0) cal_y=-1;
else if(cal_y>0) cal_y=1;
```

如果预测出错，则更新 w $w_{t+1} \leftarrow w_t + y_n x_n$ 。

```
//当预测的label和真实的label不相同时，预测错误，则更新w（w[n+1]=w[n]+x*y)。
if(cal_y!=train_y[row_of_train]){
    //  (w[n+1]=w[n]+x*y)
    for(int pos_w=0;pos_w<train_column;pos_w++){
        w[pos_w]+=1.0*train[row_of_train][pos_w]*train_y[row_of_train];
    }
    //w每更新一次，迭代次数--;
    num_of_predict--;
    //回到最初的样本，train[0] (PS:如果这里设置为row_of_train=0，则会回到train[1])
    row_of_train=-1;
}
```

因为这次实验给出的 train 是非线性可分的，所以我们设置了迭代次数，w 每更新一次，迭代次数--。当迭代次数为 0 时，迭代结束。

```
//w每更新一次，迭代次数--；
num_of_predict--;
```

《4》 用通过 train 更新好的 w 来预测 vali 的结果 label。

```
/*作用：用完全更新好的w来预测vali或test的结果*/
void predict()
{
    int vali_row=vali.size(),vali_column=vali[0].size();
    for(int row_of_vali=0;row_of_vali<vali_row;row_of_vali++){
        double predict_y=0;
        // predict_y= xi[n]*w[n]
        for(int column_of_vali=0;column_of_vali<vali_column;column_of_vali++){
            predict_y+=1.0*vali[row_of_vali][column_of_vali]*w[column_of_vali];
        }
        if(predict_y<0) predict_y=-1;
        else if (predict_y>0) predict_y=1;
        result_y.push_back(predict_y);
        if(vali_y[row_of_vali]==1 && predict_y==1) TP++;
        else if(vali_y[row_of_vali]==1 && predict_y==-1) FN++;
        else if(vali_y[row_of_vali]==-1 && predict_y==-1) TN++;
        else if(vali_y[row_of_vali]==-1 && predict_y==1) FP++;
    }
}
```

《5》 计算四个评测指标准确率、精确率、召回率、F 值:

```
void standard()
{
    Accuracy=1.0*(TP+TN)/(TP+FP+TN+FN);
    Recall=1.0*TP/(TP+FN);
    Precision=1.0*TP/(TP+FP);
    F1=1.0*(2*Precision*Recall)/(Precision+Recall);
    cout << "Accuracy=" << Accuracy << endl;
    cout << "Recall=" << Recall << endl;
    cout << "Precision=" << Precision << endl;
    cout << "F1=" << F1 << endl;
}
```

（2）PLA 口袋算法:

总体思路：

首先给每个样本前加一个常数项 1，初始化权重向量 w 以及一个全局最优向量 best_w 为 1。然后遍历所有样本，每当找到一个预测错误的样本，即$sign(w_t^T x_n) \neq y_n$，就更新 w $w_{t+1} \leftarrow w_t + y_n x_n$。如果更新的 w 的错误率小于全局最优向量 best_w，则将 w 赋给 best_w。（在这次实验中，我用 F1 值为标准来判断是否更新 best_w，即如果更新的 w 的 F 值更高，我就将 w 赋给 best_w）,然后又从头开始遍历样本，直到达到指定的迭代次数。

《1》 前面的切割文本和 PLA 原始算法相同，这里就不再重复。

《2》 用 w 值来预测 train 每一行文本的结果 label。

```
// 用w预测train的每一行的结果，cal_y= xi[n]*w[n]
for(int column_of_train=0;column_of_train<train_column;column_of_train++){
    cal_y+=1.0*train[row_of_train][column_of_train]*w[column_of_train];
}

if(cal_y<0) cal_y=-1;
else if(cal_y>0) cal_y=1;
```

如果预测错误，则更新 w.

```
//如果预测的结果和真实的结果不同的更新w
if(cal_y!=train_y[row_of_train]){
    for(int pos_w=0;pos_w<train_column;pos_w++){
        w[pos_w]+=1.0*train[row_of_train][pos_w]*train_y[row_of_train];
    }
    num_of_predict--;
    row_of_train=-1;
}
```

用更新的 w 去预测 train 的结果，得到 F1 值。

```
/*作用：用更新的W来预测train的结果*/
double predict_train(vector<double> w1)
{
    TP=0,FN=0,TN=0,FP=0;
    int train_row=train.size(),train_column=train[0].size();
    for(int row_of_train=0;row_of_train<train_row;row_of_train++){
        double predict_y=0;
        // predict_y= xi[n]*w[n]
        for(int column_of_train=0;column_of_train<train_column;column_of_train++){
            predict_y+=1.0*train[row_of_train][column_of_train]*w1[column_of_train];
        }
        if(predict_y<0) predict_y=-1;
        else if (predict_y>0) predict_y=1;
        if(train_y[row_of_train]==1 && predict_y==1) TP++;
        else if(train_y[row_of_train]==1 && predict_y==-1) FN++;
        else if(train_y[row_of_train]==-1 && predict_y==-1) TN++;
        else if(train_y[row_of_train]==-1 && predict_y==1) FP++;
    }
    double recall=1.0*TP/(TP+FN);
    double precision=1.0*TP/(TP+FP);
    double best_F1 =1.0*(2*precision*recall)/(precision+recall);
    return best_F1;
}
```

将通过更新的 w 得到的 F1 的值来和之前记录的全局最优向量 best_w 的 F1 来比较，如果更新的 w 的 F1 值较大，则将更新的 w 赋给 best_w。

```
/*比较正确性*/
double w_F1=predict(w);
if(w_F1 > best_F1){
    best_w.assign(w.begin(),w.end());
    best_F1=w_F1;
}
```

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）

（1）PLA 算法

<1> 用小数据测试自己的模型是否正确：（训练集-测试集-更新的 w）

①

-4	-1	1
0	3	-1

-2	3

```
2 -3 0
1 -3 -3
```

②

1	-4	-1	1
0	3	-1	1
2	5	8	-1
1	9	6	-1

-4	-3	9

```
2 2 -3 0
3 2 0 -1
```

<2>（目前找到的）最优解：

```
迭代次数= 7900
Accuracy=0.816
Recall=0.61875
Precision=0.445946
F1=0.518325
```

（2）PLA 口袋算法：

<1> 用小数据测试自己的模型是否正确：（训练集-测试集-更新的 w）

7.9	2.4	6.4	1.265	-1
4.6	3.7	13.3	1	1
6.2	2.1	54	1	-1
4.6	3	51.11	5	1

4.8	2.5	0	5.5
7	0.3	47	1

```
初始best_F1=0.666667
迭代次数=10
0 -6.9 -1.4 -5.4 -0.265
1 -2.3 2.3 7.9 0.735
0 -10.2 -0.1 1.5 -0.53
1 -5.6 3.6 14.8 0.47
0 -13.5 1.2 8.4 -0.795
更新best_F1=0.8
-1 -19.7 -0.9 -45.6 -1.795
0 -15.1 2.8 -32.3 -0.795
1 -10.5 6.5 -19 0.205
2 -5.9 10.2 -5.7 1.205
3 -1.3 13.9 7.6 2.205
```

<2> 最优解：

```
迭代次数=3000
Accuracy=0.798
Recall=0.6875
Precision=0.419847
F1=0.521327
```

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）

（1）PLA 算法：

不同的迭代次数：

```
迭代次数= 8000
Accuracy=0.826
Recall=0.575
Precision=0.464646
F1=0.513966
```

```
迭代次数= 7000
Accuracy=0.828
Recall=0.48125
Precision=0.463855
F1=0.472393
```

```
迭代次数= 6000
Accuracy=0.833
Recall=0.29375
Precision=0.465347
F1=0.360153
```

```
迭代次数= 4000
Accuracy=0.775
Recall=0.7125
Precision=0.389078
F1=0.503311
```

```
迭代次数= 2000
Accuracy=0.775
Recall=0.6625
Precision=0.382671
F1=0.485126
```

```
迭代次数= 1000
Accuracy=0.359
Recall=0.9125
Precision=0.188875
F1=0.312969
```

```
迭代次数= 800
Accuracy=0.726
Recall=0.675
Precision=0.327273
F1=0.440816
```

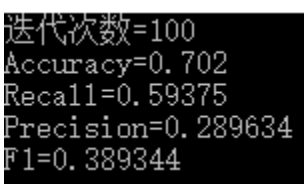
```
迭代次数= 400
Accuracy=0.81
Recall=0.2625
Precision=0.368421
F1=0.306569
```

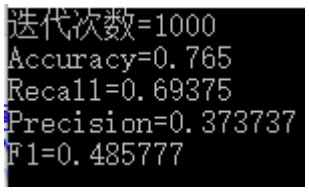
```
迭代次数= 100
Accuracy=0.791
Recall=0.25625
Precision=0.312977
F1=0.281787
```

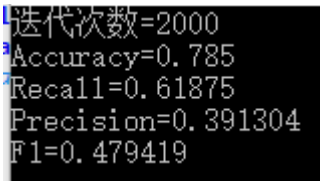
因为该验证集线性不可分，感知学习算法不收敛，迭代结果会发生震荡。

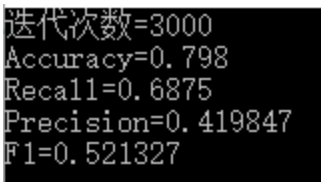
（2）PLA 口袋算法：

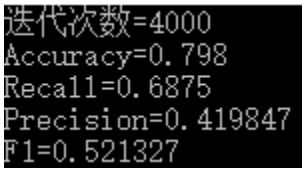
不同迭代次数：

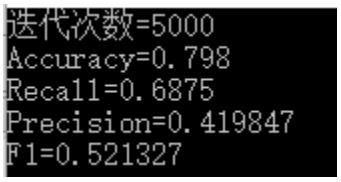










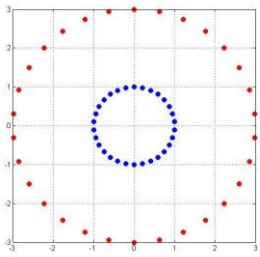


四、 思考题

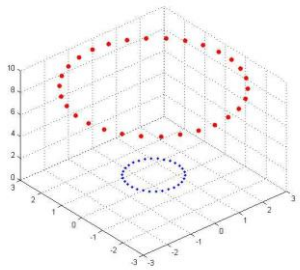
1. 有什么其他方法可以解决数据集非线性可分的问题？

答：我们可以通过升维来解决数据非线性可分的问题。因为在低维空间内线性不可分的数据，在高维空间可以线性可分。

例如：假设在二维平面 x-y 上存在若干点，其中点集 A 服从$\{x,y|x^2+y^2=1\}$,点集 B 服从$\{x,y|x^2+y^2=9\}$，那么这些点在二维平面上的分布如下：



蓝色是点集 A，红色是点集 B，它们在 xy 平面是不能线性可分的，即不能用一条直线分割。采用映射$(x,y)\rightarrow(x,y,x^2+y^2)$后，在三维空间的点的分布为：



由图可知红色和蓝色的点被映射到了不同的平面，在三维空间中就是线性可分的（用一个平面去分割）。

升维的方法：我们可以使用核函数来升维，即通过核函数把原始样本映射到高维空间中。

不同核函数的介绍见：http://www.cnblogs.com/pinard/p/6103615.html

参考：https://www.zhihu.com/question/27210162

http://www.cnblogs.com/pinard/p/6103615.html

2. 请查询相关资料，解释为什么要用准确率（Accuracy）、精确率(Precision)、召回率(Recall)、F 值这四种评测指标，各自的意义是什么？

答：准确率是指分类正确的样本数占总样本数的比重。精确率是指分类为正的样本数中分类正确的样本的比重。召回率是指在正样本数中分类正确的样本数的比重。F1 是召回率和精确率调和平均值。我们不能单一地靠某一个指标来评价分类器的性能。准确率在某种意义上能够用来判断一个分类器是否有效。但是在正负样本不平衡的情况下，准确率这个评价指标就有很大缺陷。准确率和召回率反映了分类器性能的两个方面，它们是相互影响的。F 值是综合两者指标的评估指标，用于综合反映整体的指标。我们要根据不同的需求来选择不同的指标。