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秋季学期)

姓名	学号	教学班级	课程名称	专业方向
张子豪	15352427	15M1	Artificial Intelligence	移动互联网

一、实验题目

感知机学习算法(Perception Learning Algorithm)

二、实验内容

1.算法原理

感知机是一种二分类的线性分类模型，每个实例都由其特征向量表示并有着对应的标签。在n维空间上，将实例根据标签划分为两块区域，而感知机算法则旨在求出分离它们的超平面。

初始化一个n维的权重向量$\boldsymbol{w} = \{w_1, w_2, \ldots, w_d\}$，样本$\boldsymbol{x} = \{x_1, x_2, \ldots, x_d\}$，设定一个阈值$\theta$，当

$\sum_{i=1}^{d} w_i x_i >= \theta$时，该样本则预测为+1，否则预测为-1。

PLA原始算法为，遍历训练集中每个文本，使用该权值向量$\boldsymbol{w}$进行预测，当预测错误时就更新$\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{label} * \boldsymbol{x_n}$，直到所有样本都预测正确。

对于PLA的口袋算法，进行一定次数的迭代，每次迭代都遍历一次训练集，对每个预测错误的文本更新权值向量$\boldsymbol{w}$，然后将这个$\boldsymbol{w}$放入口袋中继续进行迭代，迭代完毕后从口袋中取出最优解作为我们要求的权值向量$\boldsymbol{w}$，也就是一种贪心算法。口袋算法的优点在于一定可以取到这个迭代次数内的最优解，而PLA原始算法则不一定保证这个迭代次数下的解是最优解。

2.伪代码

PLA原始算法：

Input：训练集$\{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$

Output：权值向量$\boldsymbol{w} = \{w_1, w_2, \ldots, w_n\}$

- step1：读取数据
- step2：初始化:$w=\{1,1,\ldots,1\}$，阈值$\theta=0$
- step3：在训练集中选取(x_i,y_i)
- step4：$if\ sign(\sum_{i=1}^{d}w_ix_i-\theta)!=y_i\ \ then\ w=w+label*x_i$
- step5：转至step3，直至所有样本预测正确或超过指定迭代次数

PLA口袋算法：

Input：训练集$\{(x_1,y_1),(x_2,y_2),\ldots(x_n,y_n)\}$

Output：权值向量$w=\{w_1,w_2,\ldots,w_n\}$

- step1：读取数据
- step2：$init:w=\{1,1,\ldots,1\}$，阈值$\theta=0$
- step3：遍历训练集，$if\ sign(\sum_{i=1}^{d}w_ix_i-\theta)!=y_i\ \ then\ w=w+label*x_i$
- step4：将当前w放入口袋
- step5：$if(iterate\ is\ ending)\ \ excute\ next\ step$

  $else\ go\ to\ step3$
- step6：从口袋中取出最优解

3.关键代码截图(带注释)

```
//处理数据
for (int i = 0; i<data.size(); i++){//遍历原始数据每一行
    Data tmp;
    tmp.data.push_back(1);//每个样本前添加常数项1
    str.clear();
    for (int j = 0; j < data[i].size(); j++){
        if (data[i][j] != ','){//当前字符不是逗号
            str += data[i][j];
        }
        else{
            tmp.data.push_back(atof(str.c_str()));//string转double
            str.clear();
        }
    }
    if (str.size() != 0) tmp.label = atof(str.c_str());//存储标签
    if (data_name == "train.csv") train.push_back(tmp);
    else  if (data_name == "val.csv" || data_name == "test.csv") test.push_back(tmp);
}
```

```
//传入两个向量预测结果
int calc(const vector<double> v1, const vector<double> v2){
    double sum = 0;
    for (int i = 0; i < v1.size(); i++) sum += v1[i] * v2[i];
    if (sum > threshold) return 1;
    else return -1;
}
```

PLA原始算法代码：

```
void Get_w0(int times){//传入迭代次数
    for (int i = 0; i < times; i++){
        bool flag = true;
        for (int j = 0; j < train.size(); j++){//遍历训练集
            if (calc(train[j].data, w0) != train[j].label){//预测错误，更新w0
                flag = false;
                for (int k = 0; k < train[j].data.size(); k++) w0[k] = w0[k] +
train[j].data[k] * train[j].label;
                break;
            }
        }
        if (flag) break;//如果全部预测正确，退出迭代
    }
}
```

PLA口袋算法代码:

```
void Get_w0(int times){//传入迭代次数
    for (int i = 0; i < times; i++){
        Weight tmp;
        for (int j = 0; j < train.size(); j++){//遍历训练集
            if (calc(train[j].data, w0) != train[j].label){//预测错误，更新w0
                for (int k = 0; k < train[j].data.size(); k++) w0[k] = w0[k] +
train[j].data[k] * train[j].label;
            }
        }
        tmp.w = w0;
        /*遍历训练集统计数据，由于此部分代码较为简单，因此不占用篇幅*/
        if (Evaluate(best_sol) < Evaluate(tmp)) best_sol = tmp;//若当前解优于最优解，则更新最优解
    }
}
```

由于口袋算法本质上是一种贪心算法，因此需要对每次迭代得到的w0进行评估是否更优，这里个人使用了加权的方法，F1与Acc所占权重较高

```
double Evaluate(Weight tmp)
{
    return 10*tmp.F1+8*tmp.Accuracy+5*tmp.Precision+5*tmp.Recall;
}
```

除此之外，本次实验没有进行较大创新or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）

小数据集展示：

这里使用的小数据集只有两个向量——{ (-4,-1) , +1 }与{ (0,3) , -1 }，将w0初始化为{1,1,1}的话，在第一个向量就会预测错误，此时进行迭代，得到的结果应该为{2,-3,0}，继续预测时在第二个向量会预测错误 ，此时再进行迭代，得到的结果应为{1,-3,-3}，此时预测即全部正确。

程序运行结果如下 ，迭代次数设为 1，得到的结果为

```
iterate_times:1
TP:1
FN:0
TN:0
FP:1
Accuracy:0.5   Recall:1
Precision:0.5   F1:0.666667
w0= 2 -3 0
```

迭代次数设为2，得到的结果为

```
iterate_times:2
TP:1
FN:0
TN:1
FP:0
Accuracy:1   Recall:1
Precision:1   F1:1
w0= 1 -3 -3
```

均符合理论计算。

2.评测指标展示即分析（以下指标均是在验证集上测试）

PLA原始算法：

选定迭代次数为200 300 500，在验证集上进行指标评估 ，结果如下

```
iterate_times:200
TP:96
FN:64
TN:359
FP:481
Accuracy:0.455   Recall:0.6
Precision:0.166378   F1:0.260516
```

```
iterate_times:300
TP:40
FN:120
TN:774
FP:66
Accuracy:0.814   Recall:0.25
Precision:0.377358   F1:0.300752
```

```
iterate_times:500
TP:84
FN:76
TN:676
FP:164
Accuracy:0.76   Recall:0.525
Precision:0.33871   F1:0.411765
```

我认为出现这种波动与训练集中文本的排列顺序、+1 -1的分布与数量等属性均有关，假如迭代次数不够多时，模型只会根据训练集前面几组文本进行更新迭代，而训练集中靠后的文本则完全没有被模型学习到。由于这次我们使用的训练集中，label为-1的文本要远远多于label为1的文本，这点体现在指标上就是在验证集上预测出的结果中TN的数量远远大于TP的数量。

另外，假如该数据集收敛，但收敛的迭代时间较长，那么设置迭代次数就可能错过最优解；若数据集不收敛，设置迭代次数就可以解决这个问题，但是无法保证在这个迭代次数时获得的解是最优解，因此这里就需要口袋算法了。

PLA口袋算法：

进行5000次迭代，得到的最优解的评测指标如下：

```
iterate_times:5000
TP:102
FN:58
TN:704
FP:136
Accuracy:0.806   Recall:0.6375
Precision:0.428571   F1:0.512563
```

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：

- 删去数据集中的非线性部分训练样本。但是这个方法实现起来较为困难，因为要辨认出哪些部分是非线性部分。
- 升高维度，将训练样本从低维变成高维，例如在二维平面上非线性的训练集，变成三维空间后再进行学习即可。

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：$Accuracy = \frac{TP+TN}{All\ Simples}$ $Recall = \frac{TP}{TP+FN}$

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

Accuracy为准确率，指的是分类器预测正确的比例

Precision为精确率，指的是所有为+1的文本被分类器正确找出来的比例，即分类器预测出的+1中有多少是真的+1

Recall为召回率，也叫灵敏度，指的是本来是+1的文本中有多少被分类器正确预测

精确率与召回率的区别在于，精确率的分母是**分类器预测出的+1的数量**，而召回率的分母是**本来是+1的文本的数量**

F1是综合评价指标，F1实际上由$\frac{1}{F1} = \frac{1}{2} * (\frac{1}{Precision} + \frac{1}{Recall})$得到，即精确率和召回率的调和均值。由于精确率和召回率单独来看的话都不能很好的区分出分类器的好坏，因此使用了调和均值作为综合评价指标来平衡精确率和召回率。

因为在不同的场景下，对分类器的考量标准也不同，比如过滤垃圾邮件的时候，精确率就远比正确率、召回率重要的多，这个场景下，召回率低的话只是代表有较多垃圾邮件没有被过滤掉，但精确率低的话就代表分类器将很多原本不是垃圾邮件的重要邮件标记为垃圾邮件过滤掉了。因此，需要使用四种指标来综合考量分类器的好坏。