



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	1511	专业（方向）	软件工程（移动信息）
学号	15352237	姓名	刘顺宇

一、 实验题目

感知机学习算法

二、 实验内容

1. 算法原理

1.1. 感知机学习算法：

1.1.1. 简述：感知机就是二类分类的线性分类模型，其输入为样本的特征向量，输出为样本的类别，取 +1 和 -1 二值（自己可以随便定义这个离散值，只要能分开两类的就行的），即通过某样本的特征，就可以准确判断该样本属于哪一类。顾名思义，感知机能够解决的问题首先要求特征空间是线性可分的，再者是二类分类，即将样本分为{+1, -1}两类。

1.1.2. 适用问题：线性可分问题。

1.1.3. 影响因素：迭代终止条件、迭代更新评测指标、数据是否线性可分

1.1.4. 迭代终止条件：

1.1.4.1. 设置迭代次数，到一定程度就返回此时的 w ，不管它到底满不满足所有训练集。

1.1.4.2. 找一个 w ，使得在训练集里以此 w 来划分后，分类错误的样本最少。即相当于有一个口袋放着一个 w ，把算到的 w 跟口袋里的 w 比对，放入比较好的一个 w ，这种算法又被称为口袋（pocket）算法。

1.2. 模拟退火（优化）：

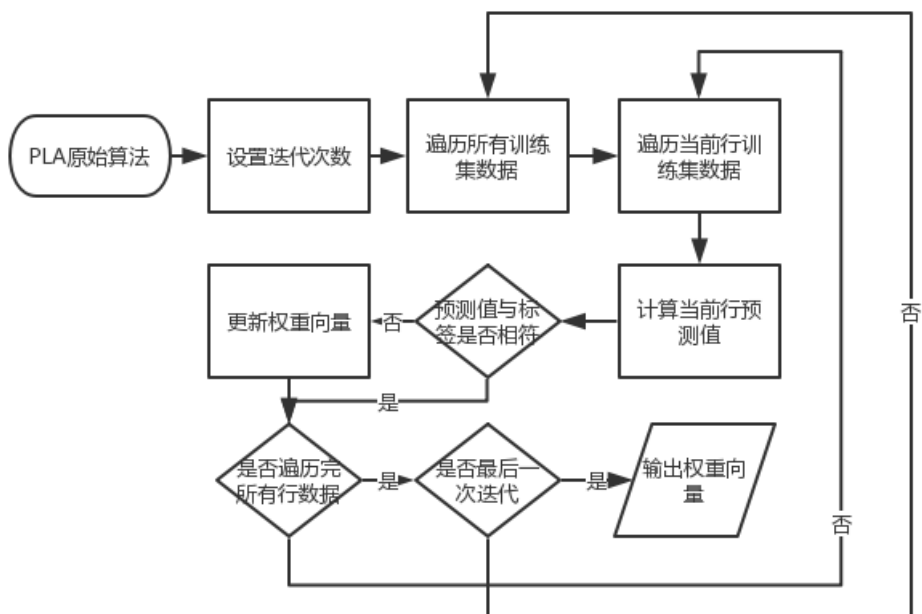
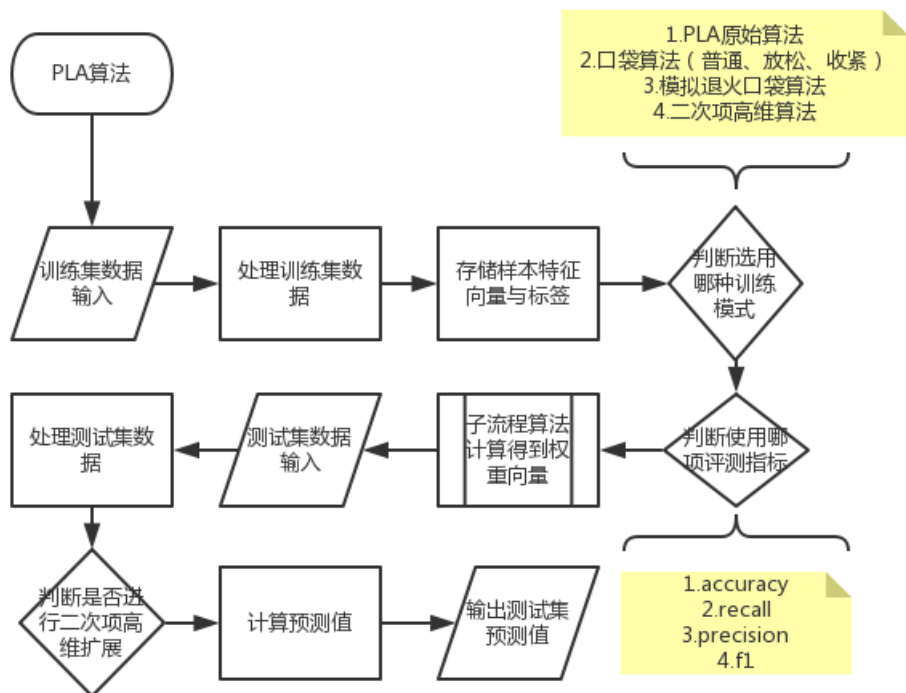
1.2.1. 简述：爬山算法是一个用来求解最优化问题的算法，每次都向着当前上升最快的方向往上爬，但是初始化不同可能会得到不同的局部最优值，模拟退火算法就可能跳出这种局部最优解的限制。模拟退火算法也是贪心算法，但是在其过程中引入了随机因素，以一定的概率接受一个比当前解要差的解，并且这个概率随着时间的推移而逐渐降低。

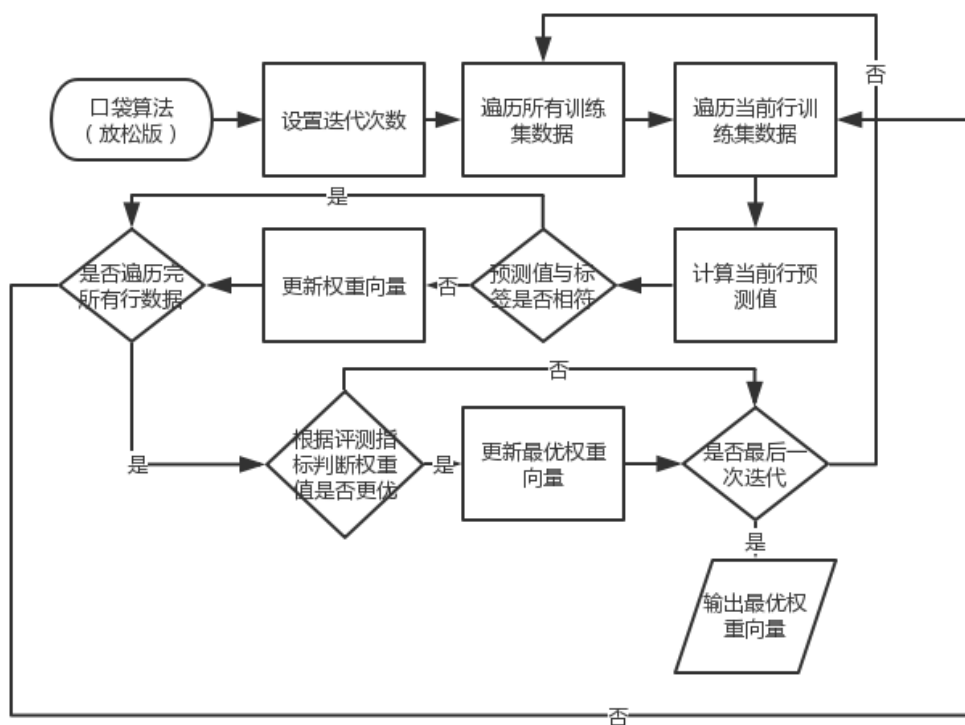
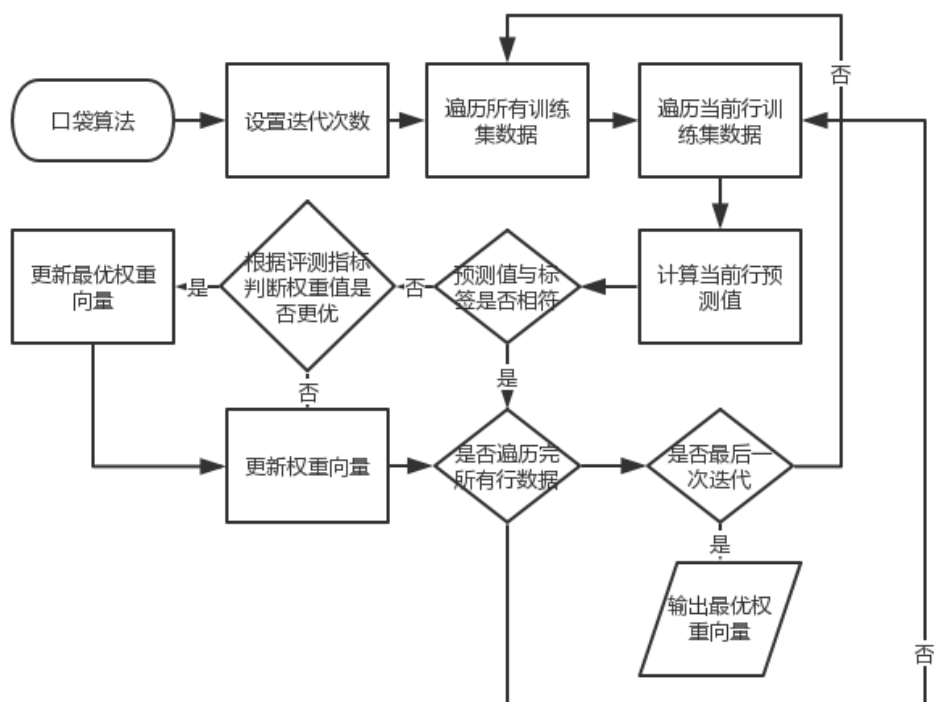
1.2.2. 影响因素：初始温度、结束温度、降温系数、随机因子

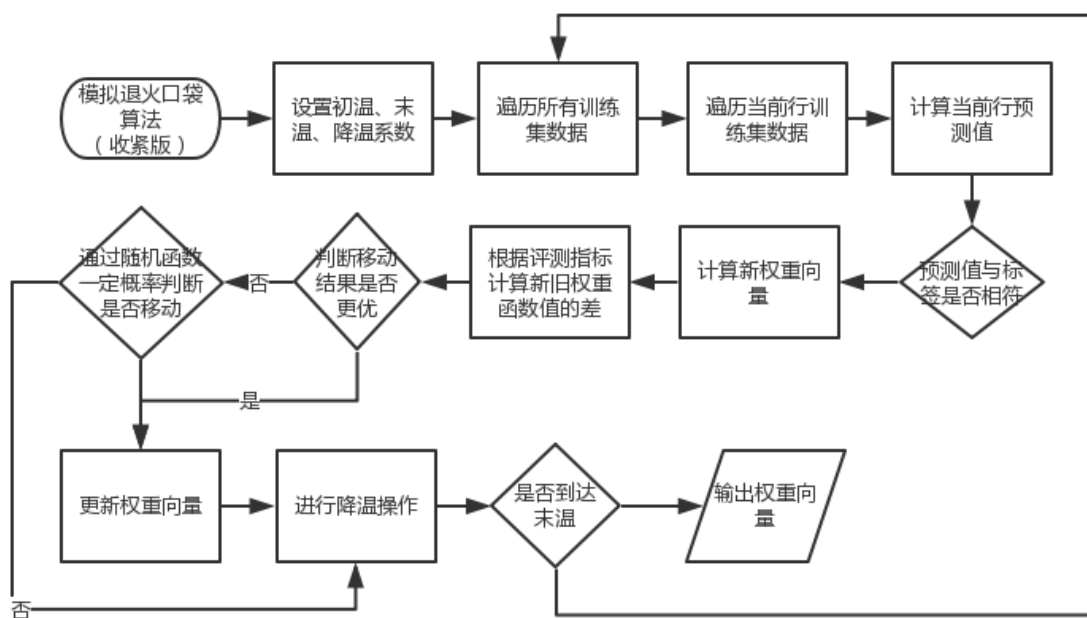
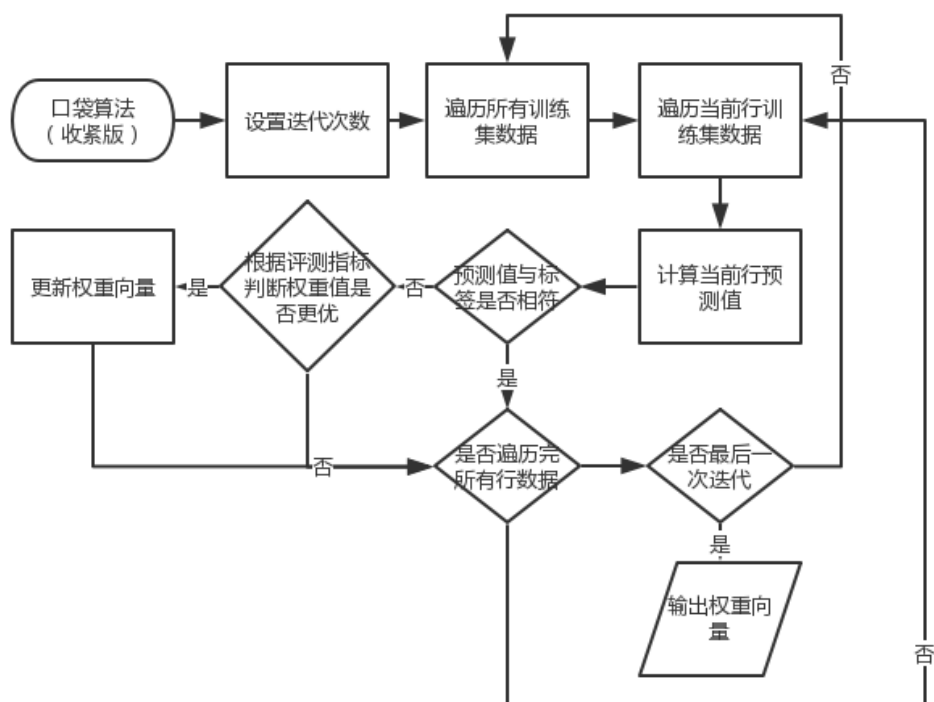
2. 伪代码流程图

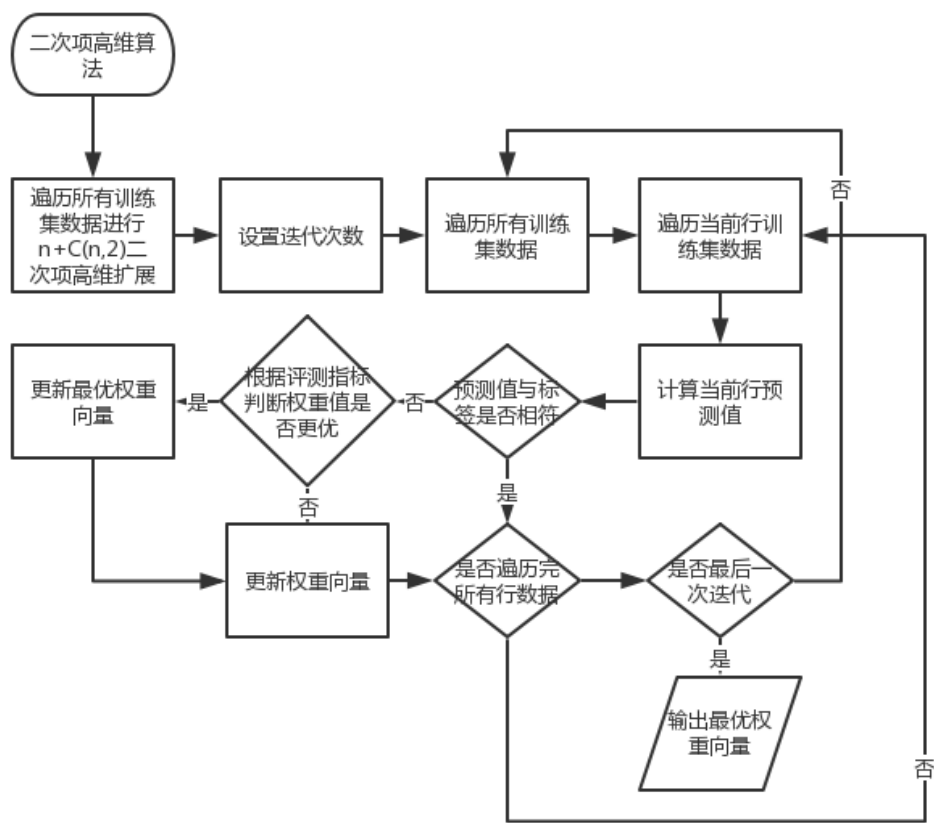
2.1. PLA 算法代码流程图（具体代码思想在关键代码解析中讲述）

首先读取训练集进行处理，然后选用各种不同的训练模式以及不同的评测指标得到权重向量，最后根据权重向量来预测测试集的标签。











3. 关键代码截图（重复代码不再贴图解释，具体代码可看 cpp 文件，有详尽注释）

3.1. 评测指标计算

通过算得的 w 值遍历数据计算预测值，将预测值与标签进行比较，记录 tp 、 tn 、 fn 、 fp 值，最后计算出四种评测指标值，返回需要用到的值。

```
//普通训练集数据评测指标计算
double train_pingce_col(double w[]){
    int tp = 0, fn = 0, tn = 0, fp = 0;
    for(int i = 0; i < cnt_row; i++){
        if(sign(vector_mul(w, Train_set[i].data, cnt_col)) == Train_set[i].value){
            if(Train_set[i].value == true) tp++;
            else tn++;
        }else{
            if(Train_set[i].value == true) fn++;
            else fp++;
        }
    }
    double accuracy = double(tp+tn) / double(tp+fp+tn+fn);
    double recall = double(tp) / double(tp+fn);
    double precision = double(tp) / double(tp+fp);
    double f1 = double(2*precision*recall) / double(precision+recall);
    return f1;
}
```

3.2. PLA 原始算法

设置迭代次数然后开始遍历所有的数据集，计算每行数据集的预测值是否准确，如果不准确便更新权重向量。

```
//原始PLA算法
if(method == 1){
    int iter = 8000; //设置迭代次数
    while(iter--){
        //遍历全部训练集数据
        for(int i = 0; i < cnt_row; i++){
            //判断预测值是否准确
            if(sign(vector_mul(w0, Train_set[i].data, cnt_col)) != Train_set[i].value){
                //更新权重向量
                for(int j = 0; j < cnt_col; j++){
                    w0[j] = w0[j] + Train_set[i].data[j] * value_bool_to_double(Train_set[i].value);
                }
            }
        }
    }
}
```

3.3. 口袋算法

设置迭代次数然后开始遍历所有的数据集，计算每行数据集的预测值是否准确，如果不准确便更新权重向量，在每次更新权重向量后都去计算比较当前权重向量是否优于最优权重向量，若是则记录替换最优权重向量。



```
// 口袋算法
}else if(method == 2){
    int iter = 20; // 设置迭代次数
    double old_train_accuracy_col = train_pingce_col(w0);
    while(iter--){
        // 遍历全部训练集数据
        for(int i = 0; i < cnt_row; i++){
            // 判断预测值是否准确
            if(sign(vector_mul(w1, Train_set[i].data, cnt_col)) != Train_set[i].value){
                // 更新权重向量
                for(int j = 0; j < cnt_col; j++){
                    w1[j] = w1[j] + Train_set[i].data[j] * value_bool_to_double(Train_set[i].value);
                }
                // 如果当前权重向量效果优于最优权重向量则更新最优权重向量
                double new_train_accuracy_col = train_pingce_col(w1);
                if(new_train_accuracy_col > old_train_accuracy_col){
                    for(int j = 0; j < cnt_col; j++){
                        w0[j] = w1[j];
                    }
                    old_train_accuracy_col = new_train_accuracy_col;
                }
            }
        }
    }
}
```

3.4. 口袋算法（放松版）

设置迭代次数然后开始遍历所有的数据集，计算每行数据集的预测值是否准确，如果不准确便更新权重向量，在迭代完一次之后，即遍历完一次全部数据集后，才去计算比较当前权重向量是否优于最优权重向量，若是则记录替换最优权重向量。

```
// 口袋算法（放松版）
}else if(method == 3){
    int iter = 1000; // 设置迭代次数
    double old_train_accuracy_col = train_pingce_col(w0);
    while(iter--){
        // 遍历全部训练集数据
        for(int i = 0; i < cnt_row; i++){
            // 判断预测值是否准确
            if(sign(vector_mul(w1, Train_set[i].data, cnt_col)) != Train_set[i].value){
                // 更新权重向量
                for(int j = 0; j < cnt_col; j++){
                    w1[j] = w1[j] + Train_set[i].data[j] * value_bool_to_double(Train_set[i].value);
                }
            }
        }
        // 如果当前权重向量效果优于最优权重向量则更新最优权重向量
        double new_train_accuracy_col = train_pingce_col(w1);
        if(new_train_accuracy_col > old_train_accuracy_col){
            for(int j = 0; j < cnt_col; j++){
                w0[j] = w1[j];
            }
            old_train_accuracy_col = new_train_accuracy_col;
        }
    }
}
```

3.5. 口袋算法（收紧版）

设置迭代次数然后开始遍历所有的数据集，计算每行数据集的预测值是否准确，如果不准确，计算比较如果更新权重向量的评测指标是否优于当前权重向量，若是则才更新权重向量，若不是则不更新权重向量（注意在这里没有引用最优权重向量的概念了）



```
//口袋算法（收紧版）
}else if(method == 4){
    int iter = 3; //设置迭代次数
    double old_train_accuracy_col = train_pingce_col(w0);
    while(iter--){
        //遍历全部训练集数据
        for(int i = 0; i < cnt_row; i++){
            //判断预测值是否准确
            if(sign(vector_mul(w0, Train_set[i].data, cnt_col)) != Train_set[i].value){
                //记录更新后的权重向量
                for(int j = 0; j < cnt_col; j++){
                    w1[j] = w0[j] + Train_set[i].data[j] * value_bool_to_double(Train_set[i].value);
                }
                //如果更新后的权重向量效果优于当前权重向量则更新当前权重向量
                double new_train_accuracy_col = train_pingce_col(w1);
                if(new_train_accuracy_col > old_train_accuracy_col){
                    for(int j = 0; j < cnt_col; j++){
                        w0[j] = w1[j];
                    }
                    old_train_accuracy_col = new_train_accuracy_col;
                }
            }
        }
    }
}
```

3.6. 针对口袋算法（收紧版）的模拟退火算法

设置模拟退火参数，即初温、末温、降温系数，然后开始遍历所有的数据集，计算每行数据集的预测值是否准确，如果不准确，计算比较如果更新权重向量的评测指标是否优于当前权重向量，若是则才更新权重向量，若不是则根据概率函数一定概率更新权重向量，最后进行降温。（注意在这里没有引用最优权重向量的概念了）

```
//模拟退火口袋算法（收紧版）
}else if(method == 5){
    double T=100000; //系统的温度，系统初始应该要处于一个高温的状态
    double T_min=1; //温度的下限，若温度T达到T_min，则停止搜索
    double dE;
    double r=0.99999; //用于控制降温的快慢
    srand(time(0));
    srand(1508848158); //r=0.999 0.508453
    cout<<"种子: " << time(0) << endl;
    double old_train_accuracy_col = train_pingce_col(w0);
    while(T > T_min){
        //遍历全部训练集数据
        for(int i = 0; i < cnt_row; i++){
            //判断预测值是否准确
            if(sign(vector_mul(w0, Train_set[i].data, cnt_col)) != Train_set[i].value){
                //记录更新后的权重向量
                for(int j = 0; j < cnt_col; j++){
                    w1[j] = w0[j] + Train_set[i].data[j] * value_bool_to_double(Train_set[i].value);
                }
                double new_train_accuracy_col = train_pingce_col(w1);
```




```
dE = (new_train_accuracy_col - old_train_accuracy_col); //记录新旧权重函数值的差
//表达移动后得到更优解，则总是接受移动
if(dE > 0){
    for(int j = 0; j < cnt_col; j++){
        w0[j] = w1[j];
    }
    old_train_accuracy_col = new_train_accuracy_col;
//表达移动不一定到更优解，则通过随机函数一定概率判断是否移动
}else if(exp(dE/T) > ((rand())/11)/10.000){
//函数exp(dE/T)的取值范围是(0,1)，dE/T越大，则exp(dE/T)也越大
    for(int j = 0; j < cnt_col; j++){
        w0[j] = w1[j];
    }
}
T=r*T; //降温退火，0<r<1。r越大，降温越慢；r越小，降温越快
/*
若r过大，则搜索到全局最优解的可能会较高，但搜索的过程也就较长。
若r过小，则搜索的过程会很快，但最终可能会达到一个局部最优值
*/
}
```

3.7. 二次项高维算法

首先遍历全部数据数据将其转换为二次项高维数据 $n+C(n, 2)$ ，其中 n 个为一次项， $C(n, 2)$ 个为二次项，升维操作完后设置迭代次数然后开始遍历所有的数据集，计算每行数据集的预测值是否准确，如果不准确便更新权重向量，在每次更新权重向量后都去计算比较当前权重向量是否优于最优权重向量，若是则记录替换最优权重向量。

```
//二次项高维算法
}else if(method == 6){
//遍历全部训练集数据将其转换为二次项高维数据 n+C(n, 2)
for(int i = 0; i < cnt_row; i++){
    int dim = 0;
    Train_set_high_dim[i].value=Train_set[i].value;
    Train_set_high_dim[i].data[0] = 1; //向量扩展
    dim++;
//n个为一次项
    for(int j = 1; j < cnt_col; j++){
        Train_set_high_dim[i].data[dim] = Train_set[i].data[j];
        dim++;
    }
//C(n, 2)个为二次项
    for(int j = 1; j < cnt_col; j++){
        for(int k = j; k < cnt_col; k++){
            Train_set_high_dim[i].data[dim] = Train_set[i].data[j] * Train_set[i].data[k];
            dim++;
        }
    }
    cnt_col_high_dim = dim;
}
```



```
int iter = 5; // 设置迭代次数
double old_train_accuracy_col_high_dim=train_pingce_col_high_dim(w0);
while(iter--){
    // 遍历全部训练集数据
    for(int i = 0; i < cnt_row; i++){
        // 判断预测值是否准确
        if(sign(vector_mul(w1, Train_set_high_dim[i].data, cnt_col_high_dim)) != Train_set_high_dim[i].value){
            // 更新权重向量
            for(int j = 0; j < cnt_col_high_dim; j++){
                w1[j] = w1[j] + Train_set_high_dim[i].data[j] * value_bool_to_double(Train_set_high_dim[i].value);
            }
            // 如果当前权重向量效果优于最优权重向量则更新最优权重向量
            double new_train_accuracy_col_high_dim=train_pingce_col_high_dim(w1);
            if(new_train_accuracy_col_high_dim > old_train_accuracy_col_high_dim){
                for(int j = 0; j < cnt_col_high_dim; j++){
                    w0[j] = w1[j];
                }
                old_train_accuracy_col_high_dim = new_train_accuracy_col_high_dim;
            }
        }
    }
}
```

4. 创新点&优化（全部创新点算法已在关键代码截图中提及，在这里不再截图，只解释关键思路）

- 4.1. 口袋算法（放松版）：与普通口袋算法不同之处在于不再每次更新完权重向量便马上比较是否替换最优权重向量，而是遍历完一次全部数据集后才去计算比较是否替换最优权重向量。

这种做法可以极大的加速口袋算法，去除原本口袋算法受速度影响不能迭代太多次，以至于 w 被受限于一小范围内。

- 4.2. 口袋算法（收紧版）：与普通口袋算法不同之处在于不是每次预测失败都更新权重向量，而是先计算是否更优，如果更优才更新。

这种做法类似于较为简单的爬山法，基本思路是如果我们拥有一个较优的解，那么一个更优的解与当前解相差不远，这种办法有利于较准确的找到一个优解。但很容易陷入局部最优解中，并且也受限于一速度影响不能迭代太多次，以至于 w 被受限于一小范围内。

- 4.3. 模拟退火口袋算法（收紧版）：针对口袋算法（收紧版）陷入局部最优解的缺陷，使用模拟退火的办法使其有一定概率接受一个差解并更新权重向量，从而跳出局部最优解找到一个全局最优解，但仍受限于一速度影响不能迭代太多次。

- 4.4. 二次项高维算法：先对数据集进行预处理，将其转换为二次项高维数据 $n+C(n, 2)$ ，其中 n 个为一次项， $C(n, 2)$ 个为二次项，然后再进行口袋算法。

根据 Cover 定理可知将复杂的模式分类问题非线性地投射到高维空间将比投射到低维空间更可能是线性可分的，所以可能可以更容易找到线性可分的那个超平面。但是这个算法也局限在我们无法找到一个正确的将数据转换为更高维的映射函数。

- 4.5. 本来权重向量初始化权重为 1，后来改用 rand() 随机附值初始化权重向量

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）

1.1. 训练集为

	A	B	C
1	-4	-1	1
2	0	3	-1

1.2. 验证集为

	A	B	C
1	-2	3	-1

1.3. 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 $w = \{1, 1, 1\}$

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

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

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

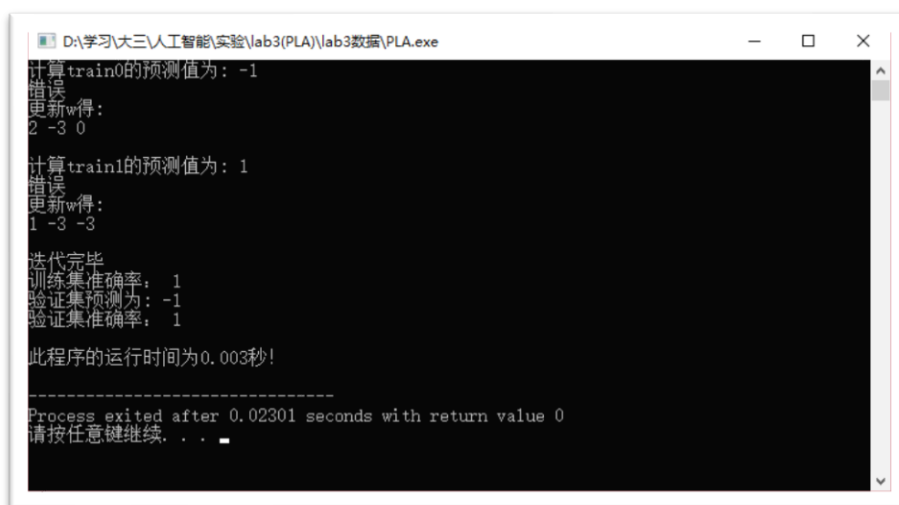
更新 w 得 $w = w + y_2 x_2 = \{1, -3, -3\}$

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

预测全正确，停止迭代

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

1.5. 程序运行结果符合计算结果与预测结果，程序正确执行。



```
D:\学习\大三人工智能实验\lab3(PLA)\lab3数据\PLA.exe
计算train0的预测值为: -1
错误
更新w得:
2 -3 0

计算train1的预测值为: 1
错误
更新w得:
1 -3 -3

迭代完毕
训练集准确率: 1
验证集预测为: -1
验证集准确率: 1

此程序的运行时间为0.003秒!

-----
Process exited after 0.02301 seconds with return value 0
请按任意键继续. . .
```

注：以上结果展示使用得是 PLA 原始模型，由于其他模型都是基于 PLA 思想所设计得，所以不再重复展示实验结果。

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）

2.1. 最优结果（针对验证集）

问题	最优准确率	最优精准率	最优召回率	最优 F1
PLA 原始算法	0.843	0.52459	0.6375	0.5
PLA 口袋算法	0.85475	0.509091	1	0.509524

2.2. 优化结果分析（针对验证集）

注：由于这次实验数据集分类不够平均，所以采用 F1 作为评测标准。

由于优化较多，展示时使用控制变量法，即只改变某优化看结果，所以只有同一个优化分析中的结果有可比性，在不同的优化分析中由于没有很好控制变量，没有可比性。本次实验中的**所有优化结果都没有比原始结果有很大的显著提升，甚至有所下降，主要问题在于口袋算法的时间复杂度**，我们无法有效的提升速度得到更多次迭代从而验证我们优化办法的好坏。

当然值得一提的是，我们的口袋算法（放松版）可以比普通的口袋算法在极短的时间内进行上千次迭代得到一个与普通口袋算法相近的解，大大提高了算法的时间复杂度。而且我们也可以看出对于口袋算法（收紧版）使用模拟退火之后，F1 有显著提高，实现了跳出局部最优解的功能。

效果最差的当属最复杂的二次项高维算法，事实证明我没有找到一个正确的映射函数来实现这个算法，导致了算法的失败。

优化前办法	优化前 F1	优化后办法	优化后 F1
口袋算法	0.509524	口袋算法（放松版）	0.50495
口袋算法	0.509524	口袋算法（收紧版）	0.423983
口袋算法（收紧版）	0.423983	模拟退火口袋算法	0.497487
口袋算法	0.509524	权重向量随机初始化	0.517333
口袋算法	0.509524	二次项高维算法	0.0020141

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？

1.1. Cover 定理可以定性地描述为：将复杂的模式分类问题非线性地投射到高维空间将比投射到低维空间更可能是线性可分的。

所以我们要解决解决数据集非线性可分的问题可以首先找到一个把数据从低维投射到高维空间的映射函数，如果找到这个映射函数并映射成功便可以通过原来的算法对这个更高维度的数据进行线性划分。

- 1.2. 进行多条线性划分，即同时计算多条不同的线性划分的超平面，然后通过多数投票原则进行划分。
- 1.3. 扩展容忍边界，在线性划分时扩展线性划分的容忍边界，允许线性划分之后超平面附近存在一定错误的点。

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。

- TP: 本来为+1, 预测为+1
- FN: 本来为+1, 预测为-1
- TN: 本来为-1, 预测为-1
- FP: 本来为-1, 预测为+1

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN} \quad \text{Recall} = \frac{TP}{TP+FN}$$
$$\text{Precision} = \frac{TP}{TP+FP} \quad \text{F1} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

- 2.1. 准确率(Accuracy): 准确率是针对全部的样本而言的，分母是总样本数，它表示的是对于给定的测试数据集，分类器正确分类的样本数与总样本数之比
- 2.2. 精确率(Precision): 精确率是针对我们预测结果而言的，分母是预测为正的样本数，它表示的是预测为正的样本中有多少是真正的正样本。那么预测为正就有两种可能了，一种就是把正类预测为正类(TP)，另一种就是把负类预测为正类(FP)
精确率=查准率=检索出的相关信息量 / 检索出的信息总量
计算的是所有"正确被检索的 item(TP)"占有所有"实际被检索到的(TP+FP)"的比例。
- 2.3. 召回率(Recall): 召回率是针对我们原来的样本而言的，分母是原来样本中所有的正样本数，它表示的是样本中的正例有多少被预测正确了。那也有两种可能，一种是把原来的正类预测成正类(TP)，另一种就是把原来的正类预测为负类(FN)
召回率=查全率=检索出的相关信息量 / 系统中的相关信息总量
计算的是所有"正确被检索的 item(TP)"占有所有"应该检索到的 item(TP+FN)"的比例
- 2.4. F-Measure: 精确率和召回率的加权调和平均值，是综合这二者指标的评估指标，用于综合反映整体的指标。当参数权值为 1 时，就是 F1 值。
准确率和召回率是相互影响的，理想情况下肯定是两者都高，但是一般情况下准确率高，召回率就低；召回率高，准确率就低；如果两者都低，那肯定是什么环节有问题了。比如，在检索系统中，如果希望提高召回率，即希望更多的相关文档被检索到，就要放宽"检索策略"，便会在检索中伴随出现一些不相关的结果，从而影响到准确率。如果希望提高准确率，即希望去除检索结果中的不相关文档时，就需要严格"检索策略"，便会使一些相关文档不能被检索到，从而影响到召回率。
针对不同目的，如果是做搜索，那就是优先提高召回率，在保证召回率的情况下，提升准确率；如果做疾病监测、反垃圾，则是优先提高准确率。



当然我们希望精确率和召回率都越高越好，但因为精确率和召回率有时候会出现互相为负相关关系的时候，所以这时候需要使用 F-Measure 来综合评估。

2.5. 例子

假如某个火锅里有猪肉丸 80 个,牛肉丸 20 个,共计 100 个.目标是找出所有牛肉丸.现在某人挑选出 50 个肉丸,其中 20 个是牛肉丸,另外还错误的把 30 个猪肉丸也当作牛肉丸挑选出来了。作为评估者的你需要来评估(evaluation)下他的工作

通过下表,我们可以很容易得到这几个值：TP=20 FP=30 FN=0 TN=50

	相关(Relevant),正类	无关(NonRelevant),负类
被检索到 (Retrieved)	true positives(TP 正类判定为正类,例子中就是正确的判定"这个是牛肉丸")	false positives(FP 负类判定为正类,"存伪",例子中就是分明是猪肉丸却判断为牛肉丸)
未被检索到 (Not Retrieved)	false negatives(FN 正类判定为负类,"去真",例子中就是,分明是牛肉丸,却判断为猪肉丸)	true negatives(TN 负类判定为负类,也就是一个猪肉丸被判断为猪肉丸)

2.5.1. 准确率(Accuracy)：在得到的所有肉丸中，正确的肉丸与无关未被检索的肉丸占全部肉丸的比例，所以其 Accuracy 也就是 70% ((20 牛肉丸 + 50 未被判断的猪肉丸) / (全部肉丸数))

2.5.2. 精确率(precision)：在得到的所有肉丸中,正确的肉丸(也就是牛肉丸)占有的比例.所以其 precision 也就是 40% (20 牛肉丸 / (20 牛肉丸 + 30 误判为牛肉丸的猪肉丸))

2.5.3. 召回率(recall)：得到的牛肉丸占本班中所有牛肉丸的比例,所以其 recall 也就是 100% (20 牛肉丸 / (20 牛肉丸+ 0 误判为猪肉丸的牛肉丸))

2.5.4. F1 值就是精确值和召回率的调和均值,例子中 F1-measure 也就是约为 57.143% ((2 * recall * precision) / (recall + precision)).

参考文献：

[1] <https://www.zhihu.com/question/19645541> 如何解释召回率与准确率？

[2] <http://www.cnblogs.com/sddai/p/5696870.html> 准确率(Accuracy)，精确率(Precision)，召回率(Recall)和 F1-Measure