

# 第一章 绪论

## 1.表1.1中若只包含编号为1， 4的两个样例， 试给出相应的版本空间。

假设数据集有n种属性， 第i个属性可能的取值有 $t_i$ 种， 加上该属性的泛化取值(\*)， 所以可能的假设有 $\prod (t_i + 1)$ 。再用空集表示没有正例， 假设空间中一共 $\prod (t_i + 1) + 1$ 种假设。

现实问题中常面临很大的假设空间， 我们可以寻找一个与训练集一致的假设集合， 称之为版本空间。版本空间从假设空间剔除了与正例不一致和与反例一致的假设， 它可以看成是对正例的最大泛化。

版本空间的可以通过搜索假设空间来得到， 这样需要遍历完整的假设空间。如果数据集中有正例， 则可以先对一个正例进行最大泛化， 得到 $2^n$ 个假设， 然后再对这些假设进行剔除操作， 可以适当精简计算量。

西瓜数据集（精简）

编号	色泽	根蒂	敲声	好瓜
1	青绿	蜷缩	浊响	是
2	乌黑	稍蜷	沉闷	否

数据集有3个属性， 每个属性2种取值， 一共  $3*3*3+1=28$ 种假设， 分别为

- 1.色泽=青绿 根蒂=蜷缩 敲声=浊响
- 2.色泽=青绿 根蒂=蜷缩 敲声=沉闷
- 3.色泽=青绿 根蒂=稍蜷 敲声=浊响
- 4.色泽=青绿 根蒂=稍蜷 敲声=沉闷
- 5.色泽=乌黑 根蒂=蜷缩 敲声=浊响
- 6.色泽=乌黑 根蒂=蜷缩 敲声=沉闷
- 7.色泽=乌黑 根蒂=稍蜷 敲声=浊响
- 8.色泽=乌黑 根蒂=稍蜷 敲声=沉闷
- 9.色泽=青绿 根蒂=蜷缩 敲声=\*
- 10.色泽=青绿 根蒂=稍蜷 敲声=\*
- 11.色泽=乌黑 根蒂=蜷缩 敲声=\*
- 12.色泽=乌黑 根蒂=稍蜷 敲声=\*
- 13.色泽=青绿 根蒂=\* 敲声=浊响
- 14.色泽=青绿 根蒂=\* 敲声=沉闷
- 15.色泽=乌黑 根蒂=\* 敲声=浊响
- 16.色泽=乌黑 根蒂=\* 敲声=沉闷
- 17.色泽=\* 根蒂=蜷缩 敲声=浊响
- 18.色泽=\* 根蒂=蜷缩 敲声=沉闷
- 19.色泽=\* 根蒂=稍蜷 敲声=浊响
- 20.色泽=\* 根蒂=稍蜷 敲声=沉闷
- 21.色泽=青绿 根蒂=\* 敲声=\*
- 22.色泽=乌黑 根蒂=\* 敲声=\*
- 23.色泽=\* 根蒂=蜷缩 敲声=\*
- 24.色泽=\* 根蒂=稍蜷 敲声=\*
- 25.色泽=\* 根蒂=\* 敲声=浊响

26.色泽=\* 根蒂=\* 敲声=沉闷

27.色泽=\* 根蒂=\* 敲声=\*

28.空集 $\emptyset$

编号1的数据可以删除 2-8, 10-12, 14-16, 18-20, 22, 24, 26, 28(不包含数据1)

编号1的数据可以删除 27(包含了数据2)

所以版本空间为:

1.色泽=青绿 根蒂=蜷缩 敲声=浊响

9.色泽=青绿 根蒂=蜷缩 敲声=\*

13.色泽=青绿 根蒂=\* 敲声=浊响

17.色泽=\* 根蒂=蜷缩 敲声=浊响

21.色泽=青绿 根蒂=\* 敲声=\*

23.色泽=\* 根蒂=蜷缩 敲声=\*

25.色泽=\* 根蒂=\* 敲声=浊响

一般情况下版本空间是正例的泛化,但由于数据集中只有1个正例,所以在版本空间中依然包含了这个样本的假设(假设1)。

## 2.与使用单个合取式来进行假设表示相比,使用“析合范式”将使得假设空间具有更强的表示能力。若使用最多包含k个合取式的析合范式来表达1.1的西瓜分类问题的假设空间,试估算有多少种可能的假设。

表1.1包含4个样例,3种属性,假设空间中有 $3*4*4+1=49$ 种假设。在不考虑冗余的情况下,最多包含k个合取式来表达假设空间,显然k的最大值是49,每次从中选出k个来组成析合式,共 $\sum C_k^{49} = 249$ 种可能。但是其中包含了很多冗余的情况(至少存在一个合取式被剩余的析合式完全包含<空集除外>)。

如果考虑冗余的情况

在这里忽略空集,一个原因是并不是太明白空集是否应该加入析合式,另外就算需要加入,求出了前面48种假设的组合,可以很容易求出加入空集后的组合数(每种可能都可以加上空集,再加上1种空集单独的情况)。

48种假设中:

具体假设:  $2*3*3=18$ 种

一个属性泛化假设:  $2*3+3*3+2*3=21$ 种

两个属性泛化假设:  $2+3+3=8$ 种

三属性泛化: 1种

当k=1时,任选一种假设都可以作为一种没有冗余的假设,共48种。

k的最大值是18,当k等于18时,就是18种具体属性假设的析取式,共1种。

当k取中间值时,就不好分析了。

一种可行的算法:

由于属性泛化后,一个泛化的假设可以对应多个具体假设。

把所有假设按三属性泛化,二属性泛化,一属性泛化,具体属性排序(这样可以保证排在后面的假设不会包含前面的任何一个假设,所以省略了一些包含判断),进行循环枚举,按顺序遍历所有假设组合248种可能(当然绝大部分都提前结束了,不会是那么夸张的量级,虽然也不低):

使用栈来实现非递归,如果当前假设还有没被析合式所包含的具体假设,则认为可以入栈,并当前栈大小的长度计数加1,并继续扫描。

如果当前扫描已经到了最后一个假设,或者所有具体假设已经被全部包含,则退栈。

循环结束条件:当最后一个假设作为第一个压入栈的元素时,认为已经遍历结束。

由于一共有18种具体假设,可以用一个32位整型(变量为hypos\_cur)的后18位来表示每一个具体假设。

用1表示具体假设没被包含,用0表示具体假设已经被析合式包含。初始的析合式为 $\emptyset$ ,可以设初试值为0X3FFFF。每个假设也对应一个32位整型(假设变量为hypo\_const),代表着它所对应了哪些具体假设,如果它包含了某种具体假设,则该位为1。

判断析合式是否包含了全部的具体假设: hypos\_cur=00。

判断该假设是否已经被析合范式包含:用hypo\_const与hypos\_cur做与运算(结果用hypo\_tmp表示), 如果为0表示已经被包含(判断该假设是否包含了当前的析合式:用hypo\_const与hypos\_cur做或运算, 如果为0X3FFFFFF, 则认为该假设包含了当前析合式, 但由于前面对所有假设做了排序, 不可能出现这种情况, 所以可以省略该判断)。

当某个假设加入析合范式后(入栈)用hypos\_cur与hypo\_tmp做异或运算, 来更改析合式所包含的具体假设。

出栈时再次用hypos\_cur与hypo\_tmp做异或, 回到加入该假设前的情况。

```
#include <vector>
#include <stack>
using namespace std;

//按泛化程度排序, 保证排在后面的假设不会包含前面的任何一个假设
static const char list[] = {
    0,0,0,
    0,0,1,0,0,2,0,0,3,0,1,0,0,2,0,0,3,0,1,0,0,2,0,0,
    0,1,1,0,1,2,0,1,3,0,2,1,0,2,2,0,2,3,0,3,1,0,3,2,0,3,3,
    1,0,1,1,0,2,1,0,3,2,0,1,2,0,2,2,0,3,
    1,1,0,1,2,0,1,3,0,2,1,0,2,2,0,2,3,0,
    1,1,1,1,1,2,1,1,3,1,2,1,1,2,2,1,2,3,1,3,1,1,3,2,1,3,3,
    2,1,1,2,1,2,2,1,3,2,2,1,2,2,2,2,3,2,3,1,2,3,2,2,3,3
};

//用来派生的抽象类
class hypos {
public:
    virtual int insert(int cur) = 0;
};

//单个的假设类
/*
hypo_const 假设对应的具体假设集合
*/
class hypo :public hypos {
public:
    hypo(int a, int b, int c) {
        hypo_const = 0;
        vector<char> p[3];
        if (a == 0) {
            p[0].push_back(1);
            p[0].push_back(2);
        }
        else
            p[0].push_back(a);
        if (b == 0) {
            p[1].push_back(1);
            p[1].push_back(2);
            p[1].push_back(3);
        }
        else
            p[1].push_back(b);
        if (c == 0) {
            p[2].push_back(1);
            p[2].push_back(2);
        }
    }
};
```

```

        p[2].push_back(3);
    }
    else
        p[2].push_back(c);
    for (unsigned int i = 0; i < p[0].size(); i++)
        for (unsigned int j = 0; j < p[1].size(); j++)
            for (unsigned int k = 0; k < p[2].size(); k++)
                hypo_const |= (1 << (p[0][i] * 9 + p[1][j] * 3 + p[2][k] -
13));
    }

    //判断是否要加入到析合式 如果还有具体假设没被包含, 则加入
    int insert(int cur) {
        return (hypo_const & cur);
    };

private:
    int hypo_const;
};

//用于压入栈的派生类 用来实现非递归
/*
hypo_tmp    记录这个假设入栈时, 带入了哪些具体假设, 出栈时要还原
ptr         记录入栈时的位置
*/
class hypo_ss : public hypos {
public:
    hypo_ss(int _ptr, int tmp) {
        hypo_tmp = tmp;
        ptr = _ptr;
    }
    int insert(int cur) {
        return 0;
    };
    int hypo_tmp;
    int ptr;
};

//用来循环遍历的类
/*
sum         各个长度的析合式各有多少种可能
ss          用来实现非递归的栈
hypos_cur   当前没被包含的具体假设 初始值为0x3FFFF
hyposs      48个假设集合
*/
class Traversal : public hypos {
public:
    Traversal() {
        hypos_cur = 0x3ffff;
        for(int i=0; i<48; i++)
            hyposs.push_back(hypo(list[3*i], list[3*i+1], list[3*i+2]));
    }

    //循环顺序遍历的主体
    //cur 初试的位置 设为0
    int insert(int cur) {
        //当前指向的位置
        int ptr = cur;

```

```

while (1) {
    //退出条件 当最后一个假设作为第一个入栈的元素 表示遍历完成
    if (ptr > 47 && !ss.size()) break;
    //回退条件 扫描到最后或者所有具体假设都被包含
    if (hypos_cur == 0 || ptr > 47) {
        hypo_ss hypo_tmp = ss.top();
        hypos_cur ^= hypo_tmp.hypo_tmp;
        ptr = hypo_tmp.ptr + 1;
        ss.pop();
        continue;
    }

    //入栈条件 如果该假设还有未被包含的具体假设 则入栈，并当前栈大小的计数加1
    if (int tmp = hyposs[ptr].insert(hypos_cur)) {
        hypos_cur ^= tmp;
        ss.push(hypo_ss(ptr, tmp));
        if (sum.size() < ss.size())
            sum.push_back(0);
        sum[ss.size() - 1]++;
    }
    ptr++;
}
return 1;
};
//输出各个长度的可能数
void print() {
    for (unsigned int i = 0; i < sum.size(); i++)
        printf("length %d : %d\n", i + 1, sum[i]);
}

private:
    vector<int> sum;
    stack<hypo_ss> ss;
    int hypos_cur;
    vector<hypo> hyposs;
};

int main()
{
    Traversal traversal;
    traversal.insert(0);
    traversal.print();
    system("pause");
    return 0;
}

/*
最终输出：
length 1 : 48
length 2 : 931
length 3 : 10332
length 4 : 72358
length 5 : 342057
length 6 : 1141603
length 7 : 2773332
length 8 : 4971915
length 9 : 6543060
length 10 : 6175660
length 11 : 4003914

```

```
length 12 : 1676233
length 13 : 422676
length 14 : 61884
length 15 : 5346
length 16 : 435
length 17 : 27
length 18 : 1
*/
```

### 3.若数据包含噪声，则假设空间中可能不存在与所有训练样本都一致的假设。在此情形下，试设计一种归纳偏好用于假设选择

通常认为两个数据的属性越相近，则更倾向于将他们分为同一类。若相同属性出现了两种不同的分类，则认为它属于与他最临近几个数据的属性。也可以考虑同时去掉所有具有相同属性而不同分类的数据，留下的数据就是没误差的数据，但是可能会丢失部分信息。

### 4.本章1.4节在论述“没有免费的午餐”定理时，默认使用了“分类错误率”作为性能度量来对分类器进行评估。若换用其他性能度量 $l$ ，试证明没有免费的午餐”定理仍成立

还是考虑二分类问题，NFL首先要保证真是目标函数 $f$ 均匀分布，对于有 $X$ 个样本的二分类问题，显然 $f$ 共 $2^X$ 种情况。其中一半是与假设一致的，也就  $P(f(x) = h(x)) = 0.5$   
此时，  $\sum f l(h(x), f(x)) = 0.5 * 2^X * (l(h(x) = f(x)) + l(h(x) \neq f(x)))$   
 $l(h(x) = f(x)) + l(h(x) \neq f(x))$ 应该是个常数，隐含的条件就应该是(一个比较合理的充分条件)  
 $l(0, 0) = l(1, 1), l(1, 0) = l(0, 1)$ 。如果不满足，NFL 应该就不成立了(或者不那么容易证明)。

### 5.试述机器学习在互联网搜索的哪些环节起什么作用

- 1.最常见的，消息推送，比如某东经常说某些商品我可能会感兴趣，然而并没有。
- 2.网站相关度排行，通过点击量，网页内容进行综合分析。
- 3.图片搜索，现在大部分还是通过标签来搜索，不过基于像素的搜索也总会有的吧。

## 第二章 模型评估与选择

1.数据集包含1000个样本，其中500个正例，500个反例，将其划分为包含70%样本的训练集和30%样本的测试集用于留出法评估，试估算共有多少种划分方式。

一个组合问题，从500500正反例中分别选出150150正反例用于留出法评估，所以可能取法应该是 $(C_{150}^{500})^2$ 种。

## 2.数据集包含100个样本，其中正反例各一半，假定学习算法所产生的模型是将新样本预测为训练样本数较多的类别（训练样本数相同时进行随机猜测），试给出用10折交叉验证法和留一法分别对错误率进行评估所得的结果。

10折交叉检验：由于每次训练样本中正反例数目一样，所以讲结果判断为正反例的概率也是一样的，所以错误率的期望是50%。

留一法：如果留下的是正例，训练样本中反例的数目比正例多一个，所以留出的样本会被判断是反例；同理，留出的是反例，则会被判断成正例，所以错误率是100%。

## 3.若学习器A的F1值比学习器B高，试析A的BEP值是否也比B高。

两个分类器的F1值得大小与他们的BEP值大小并没有明确的关系(没去找)

这道题这里用反推，设计两个BEP值相同的分类器，如果他们的F1值不一样，那么这道题的结论就是否定的

再加点我看了评论后的疑惑：

BEP值就是F1值吗？

BEP值是在 $P=R$ 时取到的，也就是 $BEP=P=R$ 。如果在计算F1时也要定义 $P=R$ ，那么F1和 $F\beta$ 将会恒等于BEP，那么 $P,R,F$ 在这里有什么意义呢？

这里分两种情况：

第一就是我的理解，在计算F1时就是按照分类器真实的分类结果来计算 $P,R$ ，再根据 $PR$ 计算F1。当这个分类器正好 $P=R$ 时，有 $P=R=BEP=F1$ 。否则BEP的计算不能用当前的 $PR$ ，而是通过一步一步尝试到查准率=查全率时， $P'=R'=BEP$ 。

第二种就是不存在我下面假设的分类器，分类器始终会在 $P=R$ 的位置进行截断(截断指的是分类器将所有样本按分为正例的可能性排序后，选择某个位置。这个位置前面分类为正，后面分类为负)。但是这个可能吗？这种情况下 $F1=F\beta=BEP$ 恒成立，分类器的评价本质将会变成了样本的正例可能性排序，而不是最终的样本划分结果。

分类器将所有训练样本按自己认为是正例的概率排序，排在越前面分类器更可能将它判断为正例。按顺序逐个把样本标记为正，当查准率与查全率相等时， $BEP=查准率=查全率$ 。当然分类器的真实输出是在这个序列中的选择一个位置，前面的标记为正，后面的标记为负，这时的查准率与查全率用来计算F1值。可以看出有同样的BEP值的两个分类器在不同位置截断可能有不同的F1值，所以F1值高不一定BEP值也高。

比如：

1/+	2/+	3/+	4/+	5/+	6/-	7/-	8/-	9/-	10/-
1/+	2/+	3/+	4/+	6/-	5/-	7/-	8/-	9/-	10/-
1/+	2/+	3/+	4/+	6/+	5/-	7/-	8/-	9/-	10/-

第一行是真实的测试样本编号与分类，第二三行是两个分类器对所有样本按为正例可能性的排序，以及判断的结果。显然两个分类器有相同的BEP值，但是他们的F1值一个是0.89，一个是0.8。

#### 4.试述真正例率（TPR）、假正例率（FPR）与查准率（P）、查全率（R）之间的联系。

查全率: 真实正例被预测为正例的比例

真正例率: 真实正例被预测为正例的比例

显然查全率与真正例率是相等的。

查准率: 预测为正例的实例中真实正例的比例

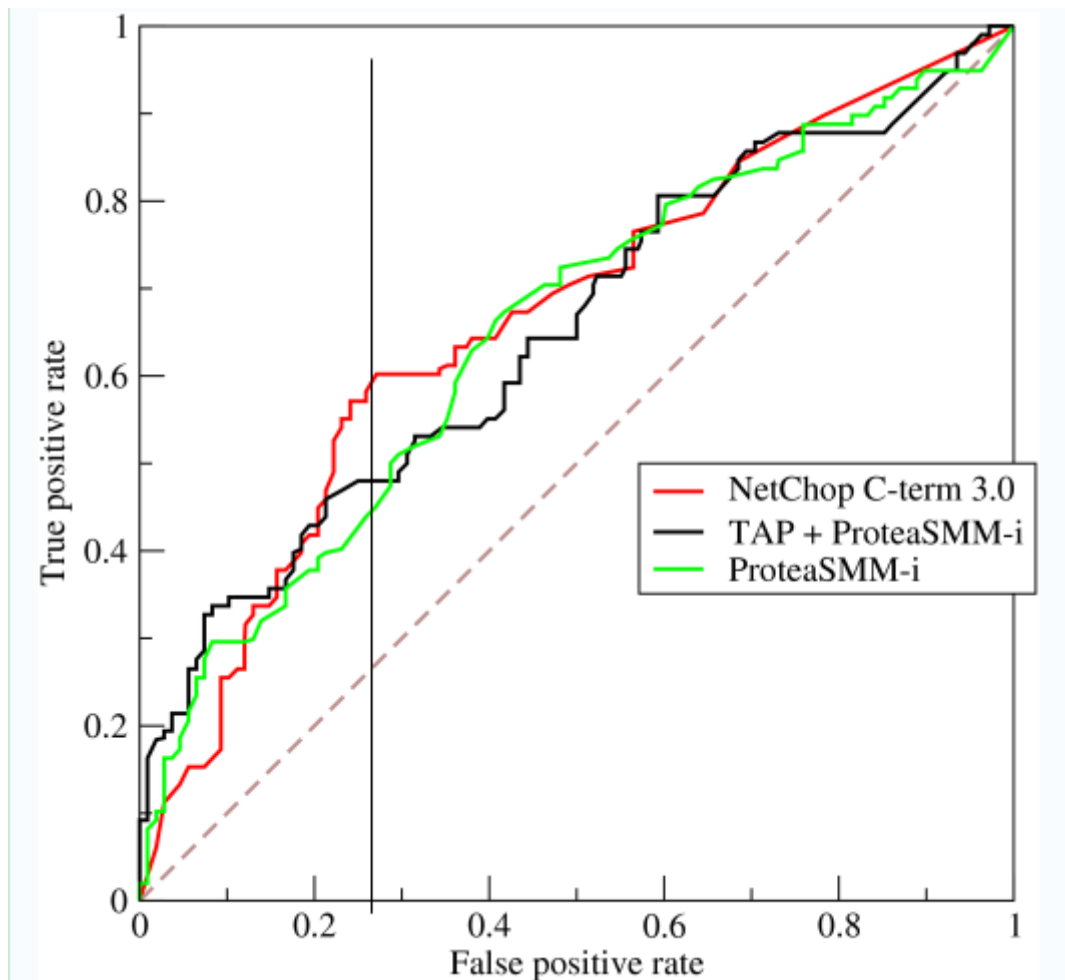
假正例率: 真实反例被预测为正例的比例

两者并没有直接的数值关系。

#### 5.试证明(2.22) $AUC = 1 - \text{lrank}$

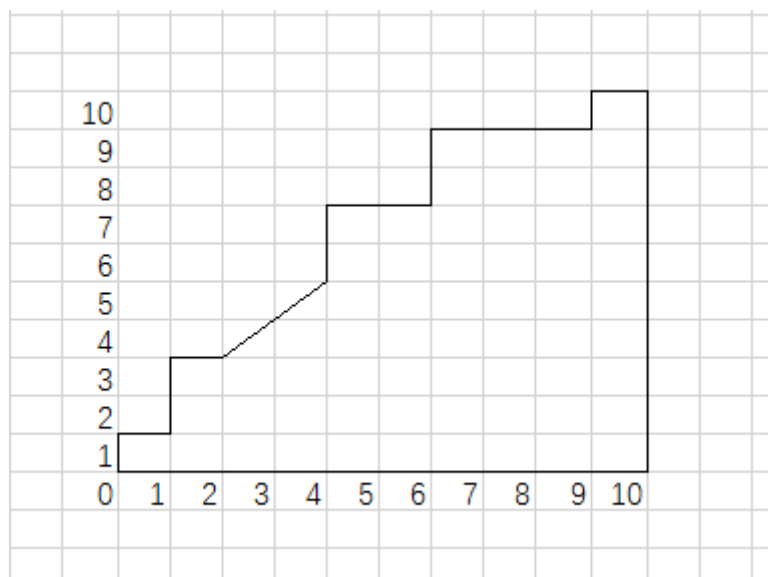
从书34页b图看来，AUC的公式不应该写的这么复杂，后来才发现原来这个图并没有正例反例预测值相等的情况。当出现这种情况时，ROC曲线会呈斜线上升，而不是这种只有水平和垂直两种情况。

由于一开始做题时并没有想过ROC曲线不可以是斜线，所以画了这张图，如果不存在正例反例预测值相等的情况，那么斜线也没必要存在。



与BEP一样，学习器先将所有测试样本按预测概率排序，越可能是正的排在越前面。然后依次遍历，每扫描到一个位置，里面如果只有正例，则ROC曲线垂直向上，如果只有反例，曲线水平往右，如果既有正例也有反例，则斜向上。如图所示





由于 $TPR$ 与 $FPR$ 的分母是常数，所以这里按比例扩大了坐标(分别是真实正例和真实反例的数目倍)，可以更好看出曲线走势。

可以看出一共有20个测试样本，10个正，10个反。学习器排序的结果是  
 $+, -, (+, +), (+, -), (+, -), (+, +), (-, -), (+, +), (-, -, -), +, -$  其中括号内的样本排在相同的位置。  
 $<(+, +, -, -)$ 与 $(+, -), (+, -)$ 是同样的效果

公式2.21累加了所有不在正例的反例数目，其中同样的位置标记为0.5，在正例前面标记为1。从图中可以看出，折线每次向右(右上)延伸，表示扫描到了反例，折线上方对应的面积，就是该反例后面有多少个正例，每个正例是一个正方形，对应的面积是1。同位置上的正例是个三角形，对应的面积是0.5。计算出总面积后，由于 $ROC$ 图的坐标是归一化的，所以总面积要除以一开始放大的倍数，也就是 $m+m-$ 。

## 6. 试述错误率与ROC曲线之间的关系

$ROC$ 曲线每个点对应了一个 $TPR$ 与 $FPR$ ，此时对应了一个错误率。

$$E_{cost} = (m + *(1 - TPR) * cost_{01} + m - *FPR * cost_{10}) / (m + +m -)$$

学习器会选择错误率最小的位置作为截断点。

## 7. 试证明任意一条ROC曲线都有一条代价曲线与之对应，反之亦然。

由定义可以知道 $TPR$ 与 $FPR$ 都是由0上升到1，那么 $FNR$ 则是由1下降到0。

每条 $ROC$ 曲线都会对应一条代价曲线，由于第一条代价线段的是 $(0, 0)$ ， $(1, 1)$ ，最后是 $(0, 1)$ ， $(1, 0)$ ，所有代价线段总会有一块公共区域，这个区域就是期望总体代价，而这块区域的边界就是代价曲线，且肯定从 $(0, 0)$ 到 $(1, 0)$ ， $(1, 0)$ 。

在有限个样本情况下 $ROC$ 是一条折线，此时根据代价曲线无法还原 $ROC$ 曲线。但若是理论上有无限个样本， $ROC$ 是一条连续的折线，代价曲线也是连续的折线，每个点的切线可以求出 $TPR$ 与 $FNR$ ，从而得到唯一的 $ROC$ 曲线。

## 8. Min-Max规范化与z-score规范化如下所示。试析二者的优缺点。

Min-max规范化方法简单，而且保证规范化后所有元素都是正的，每当有新的元素进来，只有在该元素大于最大值或者小于最小值时才会重新计算全部元素。但是若存在一个极大(小)的元素，会导致其他元素变的非常小(大)。

z-score标准化对个别极端元素不敏感，且把所有元素分布在0的周围，一般情况下元素越多，0周围区

间会分布大部分的元素，每当有新的元素进来，都要重新计算方差与均值。

## 9.试述卡方检验过程。

- 1) 分均值已知与均值未知两种情况，求得卡方检验统计量
- 2) 根据备选假设以及 $\alpha$ ，求得所选假设对应的拒绝域
- 3) 根据1)中求得的卡方统计量与2)中求得的拒绝域，判断假设成立与否。

卡方检验，或称 $\chi^2$ 检验，被誉为二十世纪科学技术所有分支中的20大发明之一，它的发明者卡尔·皮尔逊是一位历史上罕见的百科全书式的学者，研究领域涵盖了生物、历史、宗教、哲学、法律。之前做文本分类项目用过卡方值做特征选择（降维），后来听内部培训，另一个部门说他们有用卡方检验做异常用户的检测，于是就想把卡方检验再温习一次，同时把卡方检验和特征选择串起来理解。

## 10.试述在使用Friedman检验中使用式(2.34)与(2.35)的区别

书上说Friedman检验，在 $Nk$ 比较大时，平均序值 $r_i$ 近似于正态分布，均值为 $k+1/2$ ，方差为 $k^2 - 1/12$

$$\text{即: } r_i \sim N\left(\frac{k+1}{2}, \frac{k^2-1}{12}\right)$$

$$\text{所以 } \frac{12N}{k^2-1} \left(r_i - \frac{k+1}{2}\right)^2 \sim \chi^2(1)$$

统计量 $\frac{12N}{k^2-1} \sum_k \left(r_i - \frac{k+1}{2}\right)^2$ 由于 $k$ 个算法的平均序值 $r_i$ 是有关联的，知道其中 $k-1$ 个就能推出最后一个，所以自由度为 $k-1$ ，在前面乘上 $\frac{k-1}{k}$ ，最终得到Friedman统计量为 $fri = \frac{k-1}{k} * \frac{12N}{k^2-1} \sum_k \left(r_i - \frac{k+1}{2}\right)^2$

猜测:由于Friedman统计量只考虑了不同算法间的影响，而没去考虑不同数据集(其他方差)所带来的影响，所以书上说这个Friedman统计量太保守。

对序值表做方差分析:

$$\text{总方差 } SST = N * (E(X^2) - (EX)^2) = N * k * (k^2 - 1)/12 \text{ 自由度 } N * (k - 1)$$

$$\text{算法间方差 } SSA = N * \sum_k \left(r_i - \frac{k+1}{2}\right)^2 \text{ 自由度 } k - 1$$

$$\text{其他方差 } SSE = SST - SSA \text{ 自由度 } (N - 1) * (k - 1)$$

$$\text{做统计量 } f = \frac{SSA/(k-1)}{SSE/((N-1)*(k-1))} = \frac{(N-1)fri}{N(k-1)-fri}, f \text{ 服从 } (k-1) \text{ 和 } (N-1)*(k-1) \text{ 的 } F \text{ 分布}$$

## 第三章 线性模型

### 1.试分析在什么情况下，在以下式子中不比考虑偏置项b。

线性模型,两个实例相减得到,以此消除了。所以可以对训练集每个样本都减去第一个样本，然后对新的样本做线性回归，只需要用模型。

## 2.试证明，对于参数w，对率回归（logistics回归）的目标函数（式1）是非凸的，但其对数似然函数（式2）是凸的。

如果一个多元函数是凸的，那么它的Hessian矩阵是半正定的。

$$y = \frac{1}{1+e^{-(w^T x + b)}}$$
$$\frac{dy}{dw} = \frac{x e^{-(w^T x + b)}}{(1+e^{-(w^T x + b)})^2} = x(y - y^2)$$
$$\frac{d}{dw^T} \left( \frac{dy}{dw} \right) = x(1 - 2y) \left( \frac{dy}{dw} \right)^T = x x^T y(y - 1)(1 - 2y)$$

$xx^T$  合同于单位矩阵，所以  $xx^T$  是半正定矩阵

$y$  的值域为  $(0, 1)$ ，当  $y \in (0.5, 1)$  时， $y(y - 1)(1 - 2y) < 0$ ，导致  $\frac{d}{dw^T} \left( \frac{dy}{dw} \right)$  半负定，所以

$y = \frac{1}{1+e^{-(w^T x + b)}}$  是非凸的。

$$l(\beta) = \sum_{i=1}^m (-y_i \beta^T x_i + \ln(1 + e^{\beta^T x_i}))$$
$$\frac{d}{d\beta^T} \left( \frac{dl}{d\beta} \right) = x x^T p1(x; \beta)(1 - p1(x; \beta))$$

显然概率  $p1 \in (0, 1)$ ，则  $p1(x; \beta)(1 - p1(x; \beta)) \geq 0$ ，所以

$l(\beta) = \sum_{i=1}^m (-y_i \beta^T x_i + \ln(1 + e^{\beta^T x_i}))$  是凸函数。

## 3.编程实现对率回归，并给出西瓜数据集3.0a上的结果

```
%对率回归 西瓜数据集3.0a
old_l=0; %记录上次计算的l
n=0; %计算迭代次数
b=[0;0;1]; %初始参数（自定义）

x = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.0.xlsx', 'sheet1', 'A1:Q3');
y = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.0.xlsx', 'sheet1', 'A4:Q4');

while(1)
    cur_l=0;
    bx=zeros(17,1);
    %计算当前参数下的l
    for i=1:17
        bx(i) = b.'*x(:,i);
        cur_l = cur_l + ((-y(i)*bx(i)))+log(1+exp(bx(i)));
    end

    %迭代终止条件
    if abs(cur_l-old_l)<0.001
        break;
    end

    %更新参数(牛顿迭代法)以及保存当前l
    n=n+1;
    old_l = cur_l;
    p1=zeros(17,1);
    d1=0;
    d2l=0;
```

```

for i=1:17
    p1(i) = 1 - 1/(1+exp(bx(i)));
    d1 = d1 - x(:,i)*(y(i)-p1(i));
    d21 = d21 + x(:,i) * x(:,i) .* p1(i)*(1-p1(i));
end
b = b - d21\d1;
end

%画出散点图以及计算出的直线
%逐点画 分别表示是否好瓜
for i=1:17
    if y(i)==1
        plot(x(1,i),x(2,i),'+r');
        hold on;
    else if y(i)==0
        plot(x(1,i),x(2,i),'og');
        hold on;
    end
end
end

%计算出直线边界点 并绘制直线
ply=-(0.1*b(1)+b(3))/b(2);
pry=-(0.9*b(1)+b(3))/b(2);
line([0.1 0.9],[ply pry]);

xlabel('密度');
ylabel('含糖率');
title('对率回归');

```

#### 4.选择两个UCI数据集，比较10折交叉验证法和留一法所估计出的对率回归的错误率。

```

%对率回归 iris数据集
%数据集3类数据 A-AX AY-CV CW-ET 各50组
%为了程序读取方便对数据集做了处理1-6 1,2类数据 7-12 1,3类数据 13-18 2,3类数据 A-CV

x = xlsread('C:\Users\icefire\Desktop\ml\iris.xlsx', 'sheet1', 'A7:CV11');
y = xlsread('C:\Users\icefire\Desktop\ml\iris.xlsx', 'sheet1', 'A12:CV12');
%数据集分类用了1, 2, 3来表示 使用不同的类别对y的处理也不同(总之保证y的取值为 {0,1})
y = y-1;
k=100;

%10折交叉与留1法主体部分一样，只是在过滤用来验证样本的地方有区别 为了简单复制了大部分代码

%10折交叉验证
err0=0;%记录错误分类的次数
for tn=0:9
    old_l=0; %记录上次计算的l
    b=[0;0;0;0;1]; %初始参数 (自定义)
    while(1)
        cur_l=0;
        bx=zeros(k,1);
        %计算当前参数下的l

```

```

for i=1:k
    if fix(mod(i-1,50)/5) == tn %跳过用来检验的样本
        continue;
    end
    bx(i) = b.'*x(:,i);
    if bx(i)>30 %防止exp函数的参数过大导致无穷大 手动计算
        tlog=bx(i);
    else
        tlog=log(1+exp(bx(i)));
    end
    cur_l = cur_l - y(i)*bx(i) +tlog;
end

%迭代终止条件
if abs(cur_l-old_l)<0.001
    break;
end
%更新参数(牛顿迭代法)以及保存当前l
old_l = cur_l;
p1=zeros(k,1);
d1=0;
d2l=0;

for i=1:k
    if fix(mod(i-1,50)/5) == tn %跳过用来检验的样本
        continue;
    end
    p1(i) = 1 - 1/(1+exp(bx(i)));
    d1 = d1 - x(:,i)*(y(i)-p1(i));
    d2l = d2l + x(:,i) * x(:,i).'*p1(i)*(1-p1(i));
end
b = b - d2l\d1;
end

for i=0:1 %用来检验的样本 如果出错则计数+1 每组5个1类5个2类
    for j=1:5
        tmp = 1/(1+exp(-b.'*x(:,i*50+5*tn+j)));
        tmp = (tmp>=0.5);
        err0 = err0 + (tmp ~= y(i*50+5*tn+j));
    end
end
end

%留1法
err1=0;%记录错误分类的次数
for tn=84:k
    n=0;
    old_l=0; %记录上次计算的l
    b=[0;0;0;0;1]; %初始参数 (自定义)
    while(1)
        cur_l=0;
        bx=zeros(k,1);
        %计算当前参数下的l
        for i=1:k
            if i == tn %跳过用来检验的样本
                continue;
            end
            bx(i) = b.'*x(:,i); %防止exp函数的参数过大导致无穷大 手动计算

```

```

        if bx(i)>30
            tlog=bx(i);
        else
            tlog=log(1+exp(bx(i)));
        end
        cur_l = cur_l - y(i)*bx(i) +tlog;
    end

    %迭代终止条件
    if abs(cur_l-old_l)<0.001
        break;
    end
    n=n+1;
    %更新参数(牛顿迭代法)以及保存当前l
    old_l = cur_l;
    p1=zeros(k,1);
    d1=0;
    d2l=0;

    for i=1:k
        if i == tn        %跳过用来检验的样本
            continue;
        end
        p1(i) = 1 - 1/(1+exp(bx(i)));
        d1 = d1 - x(:,i)*(y(i)-p1(i));
        d2l = d2l + x(:,i) * x(:,i) .* p1(i)*(1-p1(i));
    end
    b = b - d2l\d1;
end

    %用来检验的样本 如果出错则计数+1 每组1个
    tmp = 1/(1+exp(-b.*x(:,tn)));
    tmp = (tmp>=0.5);
    err1 = err1 + (tmp ~= y(tn));
end

```

## 5.编程实现线性判别分析，并给出西瓜数据集3.0a上的结果。

```

%线性判别分析（LDA） 西瓜数据集3.0a

x = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.0.xlsx', 'sheet1', 'A1:Q2');
y = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.0.xlsx', 'sheet1', 'A4:Q4');
%更改y的值用来适合程序 1好瓜 2坏瓜
y=2-y;
u= zeros(2);

%计算均值
for i=1:17
    u(:,y(i)) = u(:,y(i)) + x(:,i);
end
u(:,1) = u(:,1) / 8;
u(:,2) = u(:,2) / 9;

```

```

%计算两类协方差矩阵和
Sw=zeros(2);
for i=1:17
    Sw=Sw+(x(:,i)-u(:,y(i)))*(x(:,i)-u(:,y(i))).';
end

%使用奇异值分解计算w
[U,S,V] = svd (Sw);
w=V/S*U.'*(u(:,1) - u(:,2));

%画出散点图以及计算出的直线
%逐点画 分别表示是否好瓜
for i=1:17
    if y(i)==1
        plot(x(1,i),x(2,i),'+r');
        hold on;
    else if y(i)==2
        plot(x(1,i),x(2,i),'og');
        hold on;
    end
end
end

%计算出直线边界点 并绘制直线
ply=-(0.1*w(1)-0.01)/w(2);
pry=-(0.9*w(1)-0.01)/w(2);
line([0.1 0.9],[ply pry]);

xlabel('密度');
ylabel('含糖率');
title('线性判别分析 (LDA) ');

```

**6. LDA仅在线性可分数据上能获得理想结果，试设计一个改进方法，使其能较好地用于非线性可分数据。**

在当前维度线性不可分，可以使用适当的映射方法，使其在更高一维上可分，典型的方法有KLDA，可以很好的划分数据。

**7.令码长为9，类别数为4，试给出海明距离意义下理论最优的EOOC二进制码并证明之**

对于 $ECOC$ 二进制码，当码长为 $2^n$ 时，至少可以使 $2n$ 个类别达到最优间隔，他们的海明距离为 $2^{(n-1)}$ 。比如长度为8时，可以的序列为

1	1	1	1	-1	-1	-1	-1
1	1	-1	-1	1	1	-1	-1
1	-1	1	-1	1	-1	1	-1
-1	-1	-1	-1	1	1	1	1
-1	-1	1	1	-1	-1	1	1
-1	1	-1	1	-1	1	-1	1

其中4, 5, 6行是对1, 2, 3行的取反。若分类数为4，一共可能的分类器共有 $2^4 - 2$ 种(排除了全1和全0)，在码长为8的最优分类器后添加一列没有出现过的分类器，就是码长为9的最优分类器。

## 8. $ECOC$ 编码能起到理想纠错作用的重要条件是：在每一位编码上出错的概率相当且独立。试析多分类任务经 $ECOC$ 编码后产生的二分类器满足该条件的可能性及由此产生的影响。

理论上的 $ECOC$ 码能理想纠错的重要条件是每个码位出错的概率相当，因为如果某个码位的错误率很高，会导致这位始终保持相同的结果，不再有分类作用，这就相当于全0或者全1的分类器，这点和NFL的前提很像。但由于事实的样本并不一定满足这些条件，所以书中提到了有多种问题依赖的 $ECOC$ 被提出。

## 9. 使用OvR和MvM将多分类任务分解为二分类任务求解时，试述为何无需专门针对类别不平衡性进行处理。

书中提到，对于OvR, MvM来说，由于对每个类进行了相同的处理，其拆解出的二分类任务中类别不平衡的影响会相互抵消，因此通常不需要专门处理。以 $ECOC$ 编码为例，每个生成的二分类器会将所有样本分成较为均衡的二类，使类别不平衡的影响减小。当然拆解后仍然可能出现明显的类别不平衡现象，比如一个超级大类和一群小类。



### 10.试推出多分类代价敏感学习(仅考虑基于类别的错误分类代价)使用“再缩放”能获得理论最优解的条件。

题目提到仅考虑类别分类的误分类代价，那么就默认正确分类的代价为0。  
于是得到分类表,(假设为3类)

0	$c_{12}$	$c_{13}$
$c_{21}$	0	$c_{23}$
$c_{31}$	$c_{32}$	0

对于二分类而言，将样本为正例的后验概率设为是 $p$ ,那么预测为正的代价是 $(1 - p) * c_{12}$ ，预测为负的代价是 $p * c_{21}$ 。当 $(1 - p) * c_{12} \leq p * c_{21}$ 样本会被预测成正例，因为他的代价更小。  
当不等式取等号时，得到了最优划分，这个阈值 $p_r = \frac{c_{12}}{c_{12}+c_{21}}$ ，这表示正例与反例的划分比例应该是初始的 $\frac{c_{12}}{c_{21}}$ 倍。假设分类器预设的阈值是 $p_o$ ,不考虑代价敏感时，当 $\frac{y}{1-y} > \frac{p_o}{1-p_o}$ 时取正例。当考虑代价敏感，则应该是 $\frac{y}{1-y} > \frac{1-p_r}{p_r} * \frac{p_o}{1-p_o} = \frac{c_{21}}{c_{12}} * \frac{p_o}{1-p_o}$ 。  
推广到对于多分类，任意两类的最优再缩放系数 $t_{ij} = c_{ij}/c_{ji}$  ,然而所有类别的最优缩放系数并不一定能同时满足。当代价表满足下面条件时，能通过再缩放得到最优解。  
设 $t_{ij} = w_i/w_j$ ，则 $w_i/w_j = c_{ij}/c_{ji}$ 对所有 $i, j$ 成立，假设有 $k$ 类，共 $C_k^2$ 个等式，此时代价表中 $k * (k - 1)$ 个数，最少只要知道 $2 * (k - 1)$ 就能推出整张表。

### 3.(这是第四章决策树第三题)试编程实现基于信息熵进行划分选择的决策树算法，并为表4.3中数据生成一棵决策树。

## 输入示例

```
1 编号,色泽,根蒂,敲声,纹理,脐部,触感,密度,含糖率,好坏
2 1,青绿,蜷缩,浊响,清晰,凹陷,硬滑,0.697,0.46,好瓜
3 2,乌黑,蜷缩,沉闷,清晰,凹陷,硬滑,0.744,0.376,好瓜
4 3,乌黑,蜷缩,浊响,清晰,凹陷,硬滑,0.634,0.264,好瓜
5 4,青绿,蜷缩,沉闷,清晰,凹陷,硬滑,0.608,0.318,好瓜
6 5,浅白,蜷缩,浊响,清晰,凹陷,硬滑,0.556,0.215,好瓜
7 6,青绿,稍蜷,浊响,清晰,稍凹,软粘,0.403,0.237,好瓜
8 7,乌黑,稍蜷,浊响,稍糊,稍凹,软粘,0.481,0.149,好瓜
9 8,乌黑,稍蜷,浊响,清晰,稍凹,硬滑,0.437,0.211,好瓜
10 9,乌黑,稍蜷,沉闷,稍糊,稍凹,硬滑,0.666,0.091,坏瓜
11 10,青绿,硬挺,清脆,清晰,平坦,软粘,0.243,0.267,坏瓜
12 11,浅白,硬挺,清脆,模糊,平坦,硬滑,0.245,0.057,坏瓜
13 12,浅白,蜷缩,浊响,模糊,平坦,软粘,0.343,0.099,坏瓜
14 13,青绿,稍蜷,浊响,稍糊,凹陷,硬滑,0.639,0.161,坏瓜
15 14,浅白,稍蜷,沉闷,稍糊,凹陷,硬滑,0.657,0.198,坏瓜
16 15,乌黑,稍蜷,浊响,清晰,稍凹,软粘,0.36,0.37,坏瓜
17 16,浅白,蜷缩,浊响,模糊,平坦,硬滑,0.593,0.042,坏瓜
18 17,青绿,蜷缩,沉闷,稍糊,稍凹,硬滑,0.719,0.103,坏瓜
```

决策树输出树的数组结构，节点按树的先根遍历排序，每个节点4个属性值，依次为

1. 父节点的index，规定根节点该位置的index指向自己
2. 如果节点是叶节点，记录分类；如果是非叶节点，记录当前位置最佳的划分属性标签
3. 记录所有的非根节点连接父节点的具体属性值，没有则为空
4. 如果该属性是连续属性，则记录阈值，没有则为空。

决策树输出示例

```
1 [[0, '纹理', [], []],
2 [0, '密度', '清晰', []],
3 [1, '坏瓜', '小于', 0.3815],
4 [1, '好瓜', '大于', 0.3815],
5 [0, '触感', '稍糊', []],
6 [4, '坏瓜', '硬滑', []],
7 [4, '好瓜', '软粘', []],
8 [0, '坏瓜', '模糊', []]]
```

```
"""
```

```
Created on Mon Jan 16 12:01:08 2017
```

```
@author: icefire
```

```
"""
```

```
from dtreepplot import dtreepplot
import math
```

```
#属性类
```

```
class property:
    def __init__(self, idnum, attribute):
        self.is_continuity=False      #连续型属性标记
        self.attribute=attribute      #属性标签
        self.subattributes=[]         #属性子标签
        self.id=idnum                #属性排在输入文本的第几位
        self.index={}                #属性子标签的索引值
```

```
#决策树生成类
```

```
class dtree():
    '''
    构造函数
    filename:输入文件名
    haveID:输入是否带序号
    property_set: 为空则计算全部属性，否则记录set中的属性
    '''
    def __init__(self, filename, haveID, property_set):

        self.data=[]
        self.data_property=[]
        #读入数据
        self.__dataread(filename, haveID)
        #判断选择的属性集合
        if len(property_set)>0:
            tmp_data_property=[]
            for i in property_set:
                tmp_data_property.append(self.data_property[i])
            tmp_data_property.append(self.data_property[-1])
        else:
            tmp_data_property=self.data_property

        #决策树树形数组结构
        self.treelink=[]

        #决策树主递归
        self.__TreeGenerate(range(0, len(self.data[-1])), tmp_data_property, 0, [],
[[])

        #决策树绘制
        dtreepplot(self.treelink, 6, 1, -6)

    '''
    决策树主递归
    data_set:当前样本集合
    property_set: 当前熟悉集合
    father:父节点索引值
    attribute:父节点连接当前节点的子属性值
    threshold:如果是连续参数就是阈值，否则为空
```

```

def __TreeGenerate(self, data_set, property_set, father, attribute, threshold):
    #新增一个节点
    self.treelink.append([])
    #新节点的位置
    curnode=len(self.treelink)-1
    #记录新节点的父亲节点
    self.treelink[curnode].append(father)

    #结束条件1: 所有样本同一分类
    current_data_class=self.__count(data_set,property_set[-1])
    if(len(current_data_class)==1):
        self.treelink[curnode].append(self.data[-1][data_set[0]])
        self.treelink[curnode].append(attribute)
        self.treelink[curnode].append(threshold)
        return

    #结束条件2: 所有样本相同属性, 选择分类数多的一类作为分类
    if all(len(self.__count(data_set,property_set[i]))==1 for i in
range(0,len(property_set)-1)):
        max_count=-1;
        for dataclass in property_set[-1].subattributes:
            if current_data_class[dataclass]>max_count:
                max_attribute=dataclass
                max_count=current_data_class[dataclass]
        self.treelink[curnode].append(max_attribute)
        self.treelink[curnode].append(attribute)
        self.treelink[curnode].append(threshold)
        return

    #信息增益选择最优属性与阈值
    prop,threshold = self.__entropy_parselect(data_set,property_set)

    #记录当前节点的最优属性标签与父节点连接当前节点的子属性值
    self.treelink[curnode].append(prop.attribute)
    self.treelink[curnode].append(attribute)

    #从属性集合中移除当前属性
    property_set.remove(prop)

    #判断是否是连续属性
    if(prop.is_continuity):
        #连续属性分为2子属性, 大于和小于
        tmp_data_set=[[], []]
        for i in data_set:
            tmp_data_set[self.data[prop.id][i]>threshold].append(i)
        for i in [0,1]:
            self.__TreeGenerate(tmp_data_set[i],property_set[:],curnode,prop.subattributes[
i],threshold)
        else:
            #离散属性有多子属性
            tmp_data_set=[[] for i in range(0,len(prop.subattributes))]
            for i in data_set:
                tmp_data_set[prop.index[self.data[prop.id][i]]].append(i)

            for i in range(0,len(prop.subattributes)):
                if len(tmp_data_set[i])>0:

```

```

self.__TreeGenerate(tmp_data_set[i],property_set[:,],curnode,prop.subattributes[
i],[])

    else:
        #如果某一个子属性不存没有对应的样本，则选择父节点分类更多的一项作为分类
        self.treelink.append([])
        max_count=-1;
        tnode=len(self.treelink)-1
        for dataclass in property_set[-1].subattributes:
            if current_data_class[dataclass]>max_count:
                max_attribute=dataclass
                max_count=current_data_class[dataclass]
        self.treelink[tnode].append(curnode)
        self.treelink[tnode].append(max_attribute)
        self.treelink[tnode].append(prop.subattributes[i])
        self.treelink[tnode].append(threshold)

        #为没有4个值得节点用空列表补齐4个值
        for i in range(len(self.treelink[curnode]),4):
            self.treelink[curnode].append([])

'''
信息增益算则最佳属性
data_set:当前样本集合
property_set:当前属性集合
'''

def __entropy_paraselect(self,data_set,property_set):
    #分离散和连续型分别计算信息增益，选择最大的一个
    max_ent=-10000
    for i in range(0,len(property_set)-1):
        prop_id=property_set[i].id
        if(property_set[i].is_continuity):
            tmax_ent=-10000
            xlist=self.data[prop_id][:]
            xlist.sort()
            #连续型求出相邻大小值的平局值作为待选的最佳阈值
            for j in range(0,len(xlist)-1):
                xlist[j]=(xlist[j+1]+xlist[j])/2
            for j in range(0,len(xlist)-1):
                if(i>0 and xlist[j]==xlist[j-1]):
                    continue
            cur_ent = 0
            nums=[[0,0],[0,0]]
            for k in data_set:
                nums[self.data[prop_id][k]>xlist[j]]
[property_set[-1].index(self.data[-1][k])] += 1
            for k in [0,1]:
                subattribute_sum=nums[k][0]+nums[k][1]
                if(subattribute_sum > 0):
                    p=nums[k][0]/subattribute_sum
                    cur_ent +=(p*math.log(p+0.00001,2)+(1-
p)*math.log(1-p+0.00001,2))*subattribute_sum/len(data_set)
            if(cur_ent>tmax_ent):
                tmax_ent = cur_ent
                tmp_threshold = xlist[j]
        if(tmax_ent > max_ent):
            max_ent=tmax_ent;
            bestprop = property_set[i];

```

```

        best_threshold = tmp_threshold;
    else:
        #直接统计并计算
        cur_ent=0
        nums=[[0,0] for i in
range(0,len(property_set[i].subattributes))]
        for j in data_set:
            nums[property_set[i].index[self.data[prop_id][j]]]
[property_set[-1].index[self.data[-1][j]]]+=1
        for j in range(0,len(property_set[i].subattributes)):
            subattribute_sum=nums[j][0]+nums[j][1]
            if(subattribute_sum>0):
                p=nums[j][0]/subattribute_sum
                cur_ent += (p*math.log(p+0.00001,2)+(1-p)*math.log(1-
p+0.00001,2))*subattribute_sum/len(data_set)
            if(cur_ent > max_ent):
                max_ent=cur_ent;
                bestprop = property_set[i];
                best_threshold = [];

    return bestprop,best_threshold

'''
计算当前样本在某个属性下的分类情况
'''
def __count(self,data_set,prop):
    out={}

    rowdata=self.data[prop.id]
    for i in data_set:
        if rowdata[i] in out:
            out[rowdata[i]]+=1
        else:
            out[rowdata[i]]=1;

    return out

'''
输入数据处理
'''
def __dataread(self,filename,haveID):
    file = open(filename, 'r')
    lineLen=0
    first=1;
    while 1:
        #按行读
        line = file.readline()

        if not line:
            break

        line=line.strip('\n')
        rowdata = line.split(',')
        #如果有编号就去掉第一列
        if haveID:
            del rowdata[0]

        if(lineLen==0):

```

```

#处理第一行，初始化属性类对象，记录属性的标签
for i in range(0,len(rowdata)):
    self.data.append([])
    self.data_property.append(property(i,rowdata[i]))
    self.data_property[i].attribute=rowdata[i]
lineLen=len(rowdata)
elif(lineLen==len(rowdata)):
    if(first==1):
        #处理第二行，记录属性是否是连续型和子属性
        for i in range(0,len(rowdata)):
            if(isnumeric(rowdata[i])):
                self.data_property[i].is_continuity=True
                self.data[i].append(float(rowdata[i]))
                self.data_property[i].subattributes.append("小于")
                self.data_property[i].index["小于"]=0
                self.data_property[i].subattributes.append("大于")
                self.data_property[i].index["大于"]=1
            else:
                self.data[i].append(rowdata[i])
        else:
            #处理后面行，记录子属性
            for i in range(0,len(rowdata)):
                if(self.data_property[i].is_continuity):
                    self.data[i].append(float(rowdata[i]))
                else:
                    self.data[i].append(rowdata[i])
                    if rowdata[i] not in
self.data_property[i].subattributes:

self.data_property[i].subattributes.append(rowdata[i])

self.data_property[i].index[rowdata[i]]=len(self.data_property[i].subattributes
)-1

        first=0
    else:
        continue
'''
判断是否是数字
'''
def isnumeric(s):
    return all(c in "0123456789.-" for c in s)

filename="西瓜3.data"
property_set=range(0,6)
link=dtree(filename,True,property_set)

```

下面是决策树的绘制类，步骤是：

处理决策树生成的数组结构，生成树形结构与层次结构

根据层次结构从下往上绘制

对于每层，首先求非叶节点的初始位置，值为它最边缘两个子节点中间；然后计算非叶节点

对于每层第一个非叶节点前和最后一个非叶节点后的节点，直接用最小距离绘制。

对于两个非叶节点中间的叶节点，如果两个非叶节点中间足够大，则中间的叶节点均匀分布，否则按最小距离绘制，并记录非叶节点的新位置，计算出相对于初始位置的偏移。

```

# -*- coding: utf-8 -*-
"""
Created on Sun Jan 15 10:19:20 2017

@author: icefire
"""

import numpy as np
from matplotlib import pyplot as plt

'''
树的节点类
data:树的数组结构的一项，4值
height:节点的高
'''
class treenode:
    def __init__(self,data,height):
        self.father=data[0]      #父节点
        self.children=[]        #子节点列表
        self.data=data[1]       #节点标签
        self.height=height
        self.pos=0;              #节点计算时最终位置，计算时只保存相对位置
        self.offset=0;          #节点最终位置与初始位置的相对值
        self.data_to_father=data[2]    #链接父节点的属性值
        #如果有阈值，则加入阈值
        if type(data[3])!=list:
            self.data_to_father=self.data_to_father+str(data[3]);

'''
树的绘制类
link:树的数组结构
minspace:节点间的距离
r:节点的绘制半径
lh:层高
'''
class dtreeplot:
    def __init__(self,link,minspace,r,lh):

        s=len(link)
        #所有节点的列表，第一项为根节点
        treenodelist=[]
        #节点的层次结构
        treelevel=[]

        #处理树的数组结构
        for i in range(0,s):
            #根节点的index与其父节点的index相同
            if link[i][0]==i:
                treenodelist.append(treenode(link[i],0))
            else:
                treenodelist.append(treenode(link[i],treenodelist[link[i]
[0]].height+1))
                treenodelist[link[i][0]].children.append(treenodelist[i]);
                treenodelist[i].father=treenodelist[link[i][0]];
            #如果有新一层的节点则新建一层
            if len(treelevel)==treenodelist[i].height:
                treelevel.append([])

```



```

        treelevel[treenodelist[i].height].append(treenodelist[i])

    #控制绘制图像的坐标轴
    self.right=0
    self.left=0
    #反转层次，从底往上画
    treelevel.reverse()
    #计算每个节点的位置
    self.__calpos(treelevel,minspace)
    #绘制树形
    self.__drawtree(treenodelist[0],r,lh,0)
    plt.xlim(xmin=self.left,xmax=self.right+minspace)
    plt.ylim(ymin=len(treelevel)*lh+lh/2,ymax=lh/2)
    plt.show()

'''
逐一绘制计算每个节点的位置
nodes: 节点集合
l,r: 左右区间
start: 当前层的初始绘制位置
minspace: 使用的最小间距
'''
def __calonebyone(self,nodes,l,r,start,minspace):
    for i in range(l,r):
        nodes[i].pos=max(nodes[i].pos,start)
        start=nodes[i].pos+minspace;
    return start;

'''
计算每个节点的位置与相对偏移
treelevel: 树的层次结构
minspace: 使用的最小间距
'''
def __calpos(self,treelevel,minspace):
    #按层次画
    for nodes in treelevel:
        #记录非叶节点
        noleaf=[]
        num=0;
        for node in nodes:
            if len(node.children)>0:
                noleaf.append(num)
                node.pos=(node.children[0].pos+node.children[-1].pos)/2
                num=num+1

        start=minspace

        #如果全是非叶节点，直接绘制
        if(len(noleaf))==0:
            self.__calonebyone(nodes,0,len(nodes),0,minspace)
        else:
            start=nodes[noleaf[0]].pos-noleaf[0]*minspace
            self.left=min(self.left,start-minspace)
            start=self.__calonebyone(nodes,0,noleaf[0],start,minspace)
            for i in range(0,len(noleaf)):
                nodes[noleaf[i]].offset=max(nodes[noleaf[i]].pos,start)-
nodes[noleaf[i]].pos
                nodes[noleaf[i]].pos=max(nodes[noleaf[i]].pos,start)

```

```

        if(i<len(noleaf)-1):
            #计算两个非叶节点中间的间隔，如果足够大就均匀绘制
            dis=(nodes[noleaf[i+1]].pos-
nodes[noleaf[i]].pos)/(noleaf[i+1]-noleaf[i])
            start=nodes[noleaf[i]].pos+max(minspace,dis)

start=self.__calonebyone(nodes,noleaf[i]+1,noleaf[i+1],start,max(minspace,dis))
        else:
            start=nodes[noleaf[i]].pos+minspace

start=self.__calonebyone(nodes,noleaf[i]+1,len(nodes),start,minspace)

'''
采用先根遍历绘制树
treenode:当前遍历的节点
r:半径
lh:层高
curoffset:每层节点的累计偏移
'''
def __drawtree(self,treenode,r,lh,curoffset):
    #加上当前的累计偏差得到最终位置
    treenode.pos=treenode.pos+curoffset

    if(treenode.pos>self.right):
        self.right=treenode.pos

    #如果是叶节点则画圈，非叶节点画方框
    if(len(treenode.children)>0):
        drawrect(treenode.pos,(treenode.height+1)*lh,r)
        plt.text(treenode.pos, (treenode.height+1)*lh, treenode.data+'=?',
color=(0,0,1),ha='center', va='center')
    else:
        drawcircle(treenode.pos,(treenode.height+1)*lh,r)
        plt.text(treenode.pos, (treenode.height+1)*lh, treenode.data,
color=(1,0,0),ha='center', va='center')

    num=0;
    #先根遍历
    for node in treenode.children:
        self.__drawtree(node,r,lh,curoffset+treenode.offset)

    #绘制父节点到子节点的连线
    num=num+1

    px=(treenode.pos-r)+2*r*num/(len(treenode.children)+1)
    py=(treenode.height+1)*lh-r-0.02

    #按到圆到方框分开画
    if(len(node.children)>0):
        px1=node.pos
        py1=(node.height+1)*lh+r
        off=np.array([px-px1,py-py1])
        off=off*r/np.linalg.norm(off)

    else:
        off=np.array([px-node.pos,-lh+1])
        off=off*r/np.linalg.norm(off)

```

```

        px1=node.pos+off[0]
        py1=(node.height+1)*lh+off[1]

        #计算父节点与子节点连线的方向与角度
        plt.plot([px,px1],[py,py1],color=(0,0,0))
        pmx=(px1+px)/2-(1-2*(px<px1))*0.4
        pmy=(py1+py)/2+0.4
        arc=np.arctan(off[1]/(off[0]+0.0000001))
        #绘制文本以及旋转
        plt.text(pmx,pmy, node.data_to_father, color=(1,0,1),ha='center',
va='center', rotation=arc/np.pi*180)

'''
画圆
'''
def drawcircle(x,y,r):
    theta = np.arange(0, 2 * np.pi, 2 * np.pi / 1000)
    theta = np.append(theta, [2 * np.pi])
    x1=[]
    y1=[]
    for tha in theta:
        x1.append(x + r * np.cos(tha))
        y1.append(y + r * np.sin(tha))
    plt.plot(x1, y1,color=(0,0,0))

'''
画矩形
'''
def drawrect(x,y,r):
    x1=[x-r,x+r,x+r,x-r,x-r]
    y1=[y-r,y-r,y+r,y+r,y-r]
    plt.plot(x1, y1,color=(0,0,0))

```

## 第四章 决策树

---

## 1.试证明对于不含冲突数据（即特征向量完全相同但标记不同）的训练集，必存在与训练集一致（即训练误差为0）的决策树。

因为决策树是通过属性来划分，相同属性的样本最终肯定会进入相同的叶节点。一个叶节点只有一个分类，如果样本属性相同而分类不同，必然产生训练误差。反之，决策树只会在当前样本集合是同一类或者所有属性相同时才会停止划分，最终得到训练误差为0的决策树。

## 2.试析使用“最小训练误差”作为决策树划分选择的缺陷。

从机器学习最开始就讲起，最小训练误差并不可靠，由于过度学习样本特性最终导致严重的过拟合，而没有泛化能力。

# 4

```
%{
    x: 训练样本属性    连续变量直接用数值,离散变量用1, 2, 3区别
    y: 训练样本分类值  1-好 2-不好
    x: 测试样本属性    连续变量直接用数值,离散变量用1, 2, 3区别
    y: 测试样本分类值  1-好 2-不好
    tree: 生成的树形结构, 用数组表示 按先根遍历编号 值为父节点编号 1为根节点
    treeleaf: 标记是否为叶子节点
    treeres: 每组3个变量
            1: 如果是叶子节点则记录分类 如果是非叶节点记录当前节点的分类属性
            2: 离散属性直接记录父节点分类的具体属性 连续属性1-小于 2-大于
            3: 如果是连续属性, 记录阈值, 离散属性为0
    ptr: 节点数目累加变量
}%
global x y x_test y_test fenlei fenlei1 xigua;
global tree treeleaf treeres ptr;

%其实这样多次读取文件很慢
x = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A1:K6');
y = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A9:K9');
x_test = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'L1:Q6');
y_test = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'L9:Q9');

%西瓜属性的中文标识
fenlei1={'色泽', '根蒂', '敲声', '纹理', '脐部', '触感'};
fenlei={'青绿','蜷缩', '浊响', '清晰', '凹陷', '硬滑',;
'乌黑', '稍蜷', '沉闷', '稍糊', '稍凹', '软粘',;
'浅白', '硬挺', '清脆', '模糊', '平坦', '无',;};
xigua = {'好瓜','坏瓜'};

[m,n]=size(y);
[tm,tn]=size(y_test);
%为set集合提前分配空间 集合中存放所有样本的编号
for i=n:-1:1
    set(i) = i;
end
```

```

for i=tn:-1:1
    test_set(i) = i;
end

tree=zeros(1,100);
treeleaf=zeros(1,100);
treeres=cell(3,100);
ptr = 0;
%{
手动设置的变量:修改输入数据时要收到修改
    pf: 属性的编号, 按顺序编号
    pu: 属性是连续还是离散的0-离散 1-连续
    pt: 属性对应的分类数, 连续属性用不着(可以设为0)
%}
pf=[1 2 3 4 5 6];
pu=[0 0 0 0 0 0];
pt=[3 3 3 3 3 2];

TreeGenerate_pre(0,set,test_set,pf,pu,pt);
treepplot(tree(1:ptr));

```

使用了一个新的函数TreeGenerate\_pre和前面的TreeGenerate区分, 当然主体还是一样, 只是加了剪枝的计算与判断。当然还有一个TreeGenerate\_aft,不过由于和TreeGenerate\_pre几乎完全一样, 就不贴了, 会在TreeGenerate\_pre里面注释

```

%{
    parent:父节点编号
    curset:当前的训练样本编号集合
    test_set:当前的测试样本编号集合
    pf:当前的属性编号集合
%}
function TreeGenerate_pre(parent,curset,test_set,pf,pu,pt)
global x y x_test y_test fenlei fenlei1 xigua;
global tree treeleaf treeres ptr;
%由于加入了测试样本 测试样本随着当前训练样本一起划分到子节点
%但属性划分时只考虑训练样本
%剪枝判断时只考虑测试样本
%大体和TreeGenerate一样 仅增加剪枝判断
ptr=ptr+1;
tree(ptr)=parent;
cur = ptr;

%递归返回情况1:当前所有样本属于同一类
n = size(curset);
n = n(2);

cury=y(curset);
y_data=unique(cury);
y_nums=histc(cury,y_data);

if(y_nums(1)==n)
    treeres{1,ptr} = xigua{y_data};
    treeleaf(cur) = n;
    return;

```

end

%计算当前y的取值分类及各有多少个 如果只有一类表示属于同一类

```
pfn=size(pf);
tmp_para = x(pf,curset(1));
f = 1;
classy = y(curset(1));
sum=zeros(1,2);
for i=1:n
    if isequal(tmp_para , x(pf,curset(i)))
        t = (classy == y(curset(i)));
        sum(t) = sum(t)+1;
    else
        f=0;
        break;
    end
end
if(f == 1 || pfn(2) == 0)
    treeres{1,cur} = xigua{(sum(2)>=sum(1))+1};
    treeleaf(cur) = 1;
    return;
end
```

%主递归

%在最优参数选择上与TreeGenerate没有区别

```
[k, threshold] = entropy_paraselect(curset,pf,pu);
curx = x(pf,:);
p_num=pf(k);
treeres{1,cur} = fenlei1{p_num};
if(pu(k))%偷懒没实现对连续属性的剪枝
    num = [1 1];
    tmp_set = zeros(2,100);
    for i=1:n
        if(curx(k,curset(i))>=threshold)
            tmp_set(1,num(1))=curset(i);
            num(1) = num(1)+1;
        else
            tmp_set(2,num(2))=curset(i);
            num(2) = num(2)+1;
        end
    end

    for i=1:2
        treeres{2,ptr+1} = fenlei{i,p_num};
        treeres{3,ptr+1} = threshold;
        ttmp_set = intersect(tmp_set(i,:),curset);
        TreeGenerate_pre(cur,ttmp_set,pf,pu,pt);
    end
else
    %离散属性
    tpt=pt(k);
    curx_test = x_test(pf,:);
    pf(:,k)=[];
    pu(:,k)=[];
    pt(:,k)=[];
    %计算当前训练样本未分类时各属性取值对应的正反例数
    num = ones(1,tpt);
```

```

tmp_set = zeros(tpt,100);
n=size(curset);
for i=1:n(2)
    tmp_set(curx(k,curset(i)),num(curx(k,curset(i))))=curset(i);
    num(curx(k,curset(i))) = num(curx(k,curset(i)))+1;
end

%计算当前测试样本未分类时各属性取值对应的正反例数
num = ones(1,tpt);
tmp_test_set = zeros(tpt,100);
n=size(test_set);
for i=1:n(2)

tmp_test_set(curx_test(k,test_set(i)),num(curx_test(k,test_set(i))))=test_set(i)
;

    num(curx_test(k,test_set(i))) = num(curx_test(k,test_set(i)))+1;
end

%计算分类后训练样本分类错误个数
cury_test=y_test(test_set);
y_test_data=unique(cury_test);
y_test_nums=histc(cury_test,y_test_data);
err_pre=y_test_nums((y_nums(1)>y_nums(2))+1); %分类前的错误数
err_aft=0; %分类后的错误计数
for i=1:tpt %每个具体取值对应的错误数
    ttmp_test_set = intersect(tmp_test_set(i,:),test_set);
    ttmp_set = intersect(tmp_set(i,:),curset);
    ttmp_set_nums=histc(y(ttmp_set),y_test_data);
    ttmp_test_set_nums=histc(y_test(ttmp_test_set),y_test_data);
    err_aft= err_aft +
ttmp_test_set_nums((ttmp_set_nums(1)>=ttmp_set_nums(2))+1);
end

%预剪枝主要判断 如果错误率没降低 就进行剪枝
%在划分递归前进行剪枝判断 如果是后剪枝 就把这个判断放在划分递归的后面(有点像树的遍历)

if(err_pre <= err_aft)
    treeres{1,cur} = xigua{(y_nums(1)>y_nums(2))+1};
    treeleaf(cur) = 0;
    return;
end

% 按每种取值递归
for i=1:tpt
    ttmp_test_set = intersect(tmp_test_set(i,:),test_set);
    ttmp_set = intersect(tmp_set(i,:),curset);
    n = size(ttmp_set);
    %如果该取值下没有样本, 不进行递归, 标记为当前样本下最多的一个分类
    if(n(2)==0)
        ptr=ptr+1;
        tree(ptr)=cur;
        treeres{1,ptr} = xigua{(y_nums(2)>y_nums(1))+1};
        treeres{2,ptr} = fenlei{i,p_num};
        treeleaf(cur) = 0;
    else
        treeres{2,ptr+1} = fenlei{i,p_num};
        TreeGenerate_pre(cur,ttmp_set,ttmp_test_set,pf,pu,pt);
    end
end
end

```

```

%
%后剪枝 在递归后判断
%if(err_pre < err_aft)
%    treeres{1,cur} = xigua{(y_nums(2)>y_nums(1))+1};
%    treeleaf(cur) = 0;
%    %比预剪枝多一步 把分配了的节点收回来 数目是当前属性取值可能数
%    ptr = ptr - tpt;
%    return;
%end
%
end

end
-----

```

基尼指数选择参数，其实和信息增益没什么区别，连生成的树都是一样的。。。信息增益率还能看到些不同的树结构。稍微改了下信息增益的代码,由于大部分一样，注释就少了点

```

%{
基尼指数选择
    curset: 当前样本集合
    pf: 当前属性的编号
    pu: 当前属性是连续还是离散的0-离散 1-连续
输出
    n: 最优属性
    threshold: 连续属性返回阈值
}%
function [n, threshold] = cart_paraselect(curset,pf,pu)
global x y;
%几乎和信息增益一样 有不明白的参考信息增益选择注释
curx = x(pf,curset);
cury = y(curset);
pn = size(pf);
all = size(cury);
min_cart = 100; %由于是取最小值 所以初值设为很大的正数
minn=0;
threshold = 0;
for i=1:pn(2)
    if(pu(i) == 1)
        con_min_cart = 100;
        con_threshold = 0;
        xlist = sort(curx(i,:),2);
        for j=all(2):-1:2
            xlist(j) = (xlist(j)+xlist(j-1))/2;
        end
        for j=2:all(2)
            cur_cart = 0;
            nums = zeros(2,2);
            for k=1:all(2)
                nums((curx(i,k)>=xlist(j))+1,cury(k)) =
nums((curx(i,k)>=xlist(j))+1,cury(k)) + 1;
            end
            for k=1:2
                if(nums(k,1)+nums(k,2) > 0)

```



```

        p=nums(k,1)/(nums(k,1)+nums(k,2));
        cur_cart = cur_cart + (1-p^2-(1-p)^2)*
(nums(k,1)+nums(k,2))/all(2);
    end
end
if(cur_cart<con_min_cart)
    con_min_cart = cur_cart;
    con_threshold = xlist(j);
end
end
if(con_min_cart < min_cart)
    min_cart=con_min_cart;
    minn = i;
    threshold = con_threshold;
end
else
    cur_cart = 0;
    set_data=unique(curx(i,:));
    setn=size(set_data);
    nums = zeros(10,2);
    for j=1:all(2)
        nums(curx(i,j),cury(j))=nums(curx(i,j),cury(j))+1;
    end
    for j=1:setn(2)
        if((nums(set_data(j),1)+nums(set_data(j),2))>0)

p=nums(set_data(j),1)/(nums(set_data(j),1)+nums(set_data(j),2));
        cur_cart = cur_cart + (1-p^2-(p-1)^2)*
(nums(set_data(j),1)+nums(set_data(j),2))/all(2);
    end
    end
    if(cur_cart < min_cart)
        minn = i;
        min_cart = cur_cart;
    end
end
end
n = minn;
threshold = threshold * pu(n);

end

```

程序主体也稍微修改下，就是对数据的读入，以及需要手动设置的属性参数上(pu,pt,pf)。TreeGenerate递归不用该，只需把entropy\_paraselect改成logre\_paraselect。

程序主体需要改动的地方

```
1 x = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A7:Q8');
2 y = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A9:Q9');
3
4 pf=[1 2];
5 pu=[1 1];
6 pt=[3 3];
```

%对率回归用不着pf和pu

%{

对率回归选择（只能处理连续变量）

curset: 当前样本集合

pf: 当前属性的编号

pu: 当前属性是连续还是离散的0-离散 1-连续

输出

n: 最优属性

threshold: 连续属性返回阈值

%}

function [n, threshold] = logre\_paraselect(curset,pf,pu)

global x y;

curx = x(:,curset);

cury = y(curset);

cury=cury-1; %对率回归处理0,1

[km, kn] = size(curx);

err=kn;

%每次只对一个属性做对率回归，求出最佳划分

for j=1:km

%对率回归代码 稍微改动

old\_l=0; %记录上次计算的l

n=0; %计算迭代次数

b=[0;1]; %初始参数（自定义）

while(1)

cur\_l=0;

bx=zeros(kn,1);

tx=[curx(j,:) ;ones(1,kn)];

%计算当前参数下的l

for i=1:kn

bx(i) = b.'\*tx(:,i);

cur\_l = cur\_l + ((-cury(i)\*bx(i)))+log(1+exp(bx(i))));

end

%迭代终止条件

if abs(cur\_l-old\_l)<0.001

break;

end

%更新参数(牛顿迭代法)以及保存当前l

n=n+1;

old\_l = cur\_l;

```

p1=zeros(kn,1);
d1=0;
d2l=0;

for i=1:kn
    p1(i) = 1 - 1/(1+exp(bx(i)));
    d1 = d1 - tx(:,i)*(cury(i)-p1(i));
    d2l = d2l + tx(:,i) * tx(:,i) .* p1(i)*(1-p1(i));
end
b = b - d2l\d1;
end
%对每个划分计算分类错误率 选择错误率低的作为最优属性 wx+b最接近0的两个x的均值作为
阈值
min_res=1000;
smin_res=1000;
cur_threshold=0;
err0=0;
for i=1:kn
    res = tx(1,i)*b(1)+b(2);
    if((res>=0)~=cury(i))
        err0 = err0 +1;
    end
    if(abs(res)<abs(min_res))
        cur_threshold2=cur_threshold;
        cur_threshold=tx(1,i);
        smin_res = min_res;
        min_res = res;
    elseif (abs(res)<abs(smin_res))
        cur_threshold2=tx(1,i);
        smin_res =res;
    end
end
if(err0 < err)
    n=j;
    threshold=(cur_threshold+cur_threshold2)/2;
end
end

end

```

## 6.试选择4个UCI数据集，对上述3种算法所产生的未剪枝、预剪枝、后剪枝决策树进行实验比较，并进行适当的统计显著性检验。

这里要对上面三种实现的算法进行未剪枝，预剪枝，后剪枝做比较，对率回归划分就算了，都不知道是个什么情况，信息增益和基尼指数的差别并不大，其实就是为了比较未剪枝，预剪枝，后剪枝对测试样本的输出结果。显著性分析，对2种算法，3种剪枝方式的错误数做方差分析，信息增益和基尼指数有显著区别是拒绝的，未剪枝，预剪枝，后剪枝有显著区别是接受的。

**7.图4.2是一个递归算法，若面临巨量数据，则决策树的层数会很深，使用递归方法易导致“栈”溢出，试使用“队列”数据结构，以参数maxDepth控制数的最大深度，写出与图4.2等价、但不使用递归的决策树生成算法。**

直接用递归会导致大量的临时变量被保存，当层数过深时会导致“栈”溢出。

用队列对决策树进行层次遍历来生成，用Max\_Depth来控制树的最大层数。队列中每个元素代表着决策树的每个节点，它必要的属性有：样本集合、剩余属性集合，当前层数指示，父节点序号。队列一开始里面只有一个元素，就是最初初始化，带着所有样本的根节点。然后当队列不为空的时候开始循环，每次取出一个元素，判断是否需要划分，如果不要，就是一个叶节点，出队列就不用管了；如果需要划分，那么找出最好的划分属性，然后划分成n个子区间，依次送入队列，继续循环，直到队列为空。

是否需要划分有3个依据：

- 当前所有样本属于一类
- 当前所有样本属性完全相同
- 达到了Max\_Depth的深度

这样就完成了层次遍历(广度优先搜索)对决策树的构建。

显然由于每次出队的元素要先完全划分，那么如果是进行预剪枝算法的决策树，用队列结构是非常方便的。

如果是后剪枝，那必须要等到最终整棵树完全生成，才能进行。

**8.试将决策树生成的深度优先搜索过程修改为广度优先搜索，以参数MaxNode控制树的最大结点数，将题4.7中基于队列的决策树算法进行改写。对比题4.7中的算法，试分析哪种方式更易于控制决策树所需储存不超过内存。**

队列结构看着不错，但是极端情况下，比如层树很低，但是分叉多，会导致队列内有大量元素。另外不能在决策树生成过程中很好的进行后剪枝，全树生成后再次统计会浪费大量时间。

用栈结构把递归转换为非递归是很好的方法，设置最大节点数MaxNode来保证节点数不会爆炸。栈结构内的元素组成与队列的基本相同：样本集合、剩余属性集合，当前层数指示，父节点序号。初始化也是一样，将一个包含所有样本的集合压入栈中，当栈不为空时，每次拿出栈顶的元素进行操作，如果不可以划分，则不做操作；如果可以划分，则划分出1个子集，先将剩余子集压回栈，再将划分出的子集压回栈。然后再取出一个元素，知道栈内没有元素为止。

是否需要划分有3个依据：

- 当前所有样本属于一类
- 当前所有样本属性完全相同
- 达到了Max\_node上限

栈也有他的问题，极端情况下会生成一棵畸形的二叉树，但就算这样内部元素也只是队列结构极端情况的一半。栈可以很好的进行后剪枝操作，当非叶节点所有叶节点生成后可以做为后剪枝，由于划分子集的时候就需要计算各个属性样本数，所以这些操作代价并不高。对于一棵完整的树，后剪枝要对所有只拥有叶节点的非叶节点进行遍历，如果某个非叶节点可以剪枝，还需要进一步考虑它的父节点。虽然在某些剪枝需求不大的情况下生成树后剪枝有不错的效果，但是我觉得生成中判断更加方便，而且不需要太多额外的代码。

**9.试将4.4.2节对缺失值的处理机制推广到基尼指数的计算中去。**

只需要把信息增益的公式换成基尼指数就行，包括扩展到连续参数，缺失参数，都是很直观的方法

**10.从网上下载或自己编程实现任意一种多变量决策树算法，并观察其在西瓜数据集3.0上产生的结果。**

```

%{
    x:训练样本属性 连续变量直接用数值,离散变量用1, 2, 3区别
    y:训练样本分类值 1-好 2-不好
    tree:生成的树形结构,用数组表示 按先根遍历编号 值为父节点编号 1为根节点
    treeleaf:标记是否为叶子节点
    treeres:每组3个变量
        1:如果是叶子节点则记录分类 如果是非叶节点记录当前节点的分类属性
        2:离散属性直接记录父节点分类的具体属性 连续属性1-小于 2-大于
        3:如果是连续属性,记录阈值,离散属性为0
    ptr:节点数目累加变量
    记录每次对率回归的参数 存入lw,lb
%}

global x y ;
global tree treeleaf treeres ptr;
global lw lb ;

x = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A7:Q8');
y = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A9:Q9');

%为变量预分配空间
[m,n]=size(y);
[tm,tn]=size(y_test);
for i=n:-1:1
    set(i) = i;
end
for i=tn:-1:1
    test_set(i) = i;
end
lw = zeros(2,100);
lb = zeros(2,100);
tree=zeros(1,100);
treeleaf=zeros(1,100);
treeres=zeros(1,100);
tf = zeros(1,100);
ptr = 0;

TreeGenerate_mul(0,set);

%treepplot(tree(1:ptr));

%绘制点
for i=1:17
    if y(i)==1
        plot(x(1,i),x(2,i),'+r');
        hold on;
    else if y(i)==2
        plot(x(1,i),x(2,i),'og');
        hold on;
    end
end
end

%绘制直线

%第一条划分线单独画出
w1=lw(:,1);

```

```

b1=lb(1);
y1=-(0.1*w1(1)+b1)/w1(2);
y2=-(0.9*w1(1)+b1)/w1(2);
line([0.1 0.9],[y1 y2]);
hold on;

%通过树结构查询父节点直线参数，求出交点画出直线
for i=2:ptr
    if(treeleaf(i)>0)
        continue;
    end
    w1=lw(:,i);
    w2=lw(:,tree(i));
    b1=lb(i);
    b2=lb(tree(i));

    x1=0.1;
    y1=-(x1*w1(1)+b1)/w1(2);
    if(y1<0)
        y1=0.5;
        x1=-(y1*w1(2)+b1)/w1(1);
    end
    y2=(w2(1)*b1-w1(1)*b2)/(w1(1)*w2(2)-w2(1)*w1(2));
    if(y2>0.5)
        y2=0.5;
    end
    if(y2<0.1)
        y2=0.1;
    end
    x2=-(y2*w1(2)+b1)/w1(1);
    line([x1 x2],[y1 y2]);
    hold on;
end

xlabel('密度');
ylabel('含糖率');
title('对率回归-多变量划分决策树');

```

```

function TreeGenerate_mul(parent,curset)
global x y;
global tree treeleaf treeres ptr;
global lw lb;

    ptr=ptr+1;
    tree(ptr)=parent;
    cur = ptr;
    %returne case 1
    n = size(curset);
    n = n(2);

    cury=y(curset);
    y_data=unique(cury);
    y_nums=histc(cury,y_data);

    if(y_nums(1)==n)

```

```

        treeres(cur) = y_data(1);
        treeleaf(cur) = n;
        return;
    end

    %{
        多参数对率回归(不用再选取最优属性)
        每次选取最优的划分，将数据集分为2类
        递归直到训练误差到0(没有冲突的样本)
        或者到递归最大深度
    %}

    [w,b] = logre_paraselect_mul(curset);
    lw(:,cur)=w;
    lb(cur) =b;
    curx = x(:,curset);
    num = [1 1];
    tmp_set = zeros(2,100);
    %将样本分为2类 在支线上面和下面
    for i=1:n
        if(w'*curx(:,i)+b>0)
            tmp_set(1,num(1))=curset(i);
            num(1) = num(1)+1;
        else
            tmp_set(2,num(2))=curset(i);
            num(2) = num(2)+1;
        end
    end

    for i=1:2
        tmp_set = intersect(tmp_set(i,:),curset);
        TreeGenerate_mul(cur,tmp_set);
    end
end
end

```

%其实就是对率回归 返回参数

```

function [lw,lb] = logre_paraselect_mul(curset)
global x y;
curx = x(:,curset);
cury = y(curset);
cury=cury-1;
old_l=0; %记录上次计算的l
n=0; %计算迭代次数
b=[0;0;1]; %初始参数 (自定义)
[km,kn]=size(curx);
while(1)
    cur_l=0;
    bx=zeros(kn,1);
    tx=[curx ;ones(1,kn)];
    %计算当前参数下的l
    for i=1:kn
        bx(i) = b.'*tx(:,i);
        cur_l = cur_l + ((-cury(i)*bx(i)))+log(1+exp(bx(i)));
    end
end

```

```

%迭代终止条件
if abs(cur_l-old_l)<0.001
    break;
end

%更新参数(牛顿迭代法)以及保存当前l
n=n+1;
old_l = cur_l;
p1=zeros(kn,1);
d1=0;
d2l=0;

for i=1:kn
    p1(i) = 1 - 1/(1+exp(bx(i)));
    d1 = d1 - tx(:,i)*(cury(i)-p1(i));
    d2l = d2l + tx(:,i) * tx(:,i).'*p1(i)*(1-p1(i));
end
b = b - d2l\d1;
end
lw = b(1:km,:);
lb = b(km+1);
end

```

## 第五章 神经网络

### 1.试述将线性函数 $f(x) = wtx$ 用作神经元激活函数的缺陷。

必须要强调的是，神经网络中必须要有非线性的激活函数，无论是在隐层，还是输出层，或者全部都是。如果单用  $wtx$  作为激活函数，无论多少层的神经网络会退化成一个线性回归，只不过是把他复杂化了。

### 2.试述使用图5.2(b)激活函数的神经元与对率回归的联系。

两者都是希望将连续值映射到  $\{0,1\}$  上，但由于阶跃函数不光滑，不连续的性质，所以才选择了 sigmoid 作为映射函数。不同之处在于激活函数不一定要使用 sigmoid，只要是非线性的可导函数都可以使用。



### 3.对于图5.7中 $v_{ih}$ ,试推导出BP算法中的更新公式。

$$-\frac{\partial E_k}{\partial v_{ih}} = -\frac{\partial E_k}{\partial b_h} \frac{\partial b_h}{\partial a_h} \frac{\partial a_h}{\partial v_{ih}}$$

$$\frac{\partial a_h}{\partial v_{ih}} = x_i$$

$$e_h = -\frac{\partial E_k}{\partial b_h} \frac{\partial b_h}{\partial a_h} \text{ 在书中5.15已经证明}$$

所以得到更新公式 $\Delta v_{ih} = \eta e_h x_i$

### 4.试述学习率的取值对神经网络训练的影响。

如果学习率太低，每次下降的很慢，使得迭代次数非常多。

如果学习率太高，在后面迭代时会出现震荡现象，在最小值附近来回波动。

### 5.试编程实现标准BP算法与累积BP算法，在西瓜数据集3.0上分别用这个算法训练一个单隐层网络，并进行比较。

假设一个单隐层BP网络中

d个输入节点

隐层有q个神经元

输出层l个神经元

BP算法要训练的参数有

输入层与隐层全连接的权值 $v_{ij}$   $d \times q$ 个

隐层神经元阈值 $\theta_i$   $q$ 个

隐层与输出层全连接的权值 $w_{ij}$   $q \times l$ 个

输出层神经元阈值 $\gamma_i$   $l$ 个

BP算法每次迭代依次计算每一个样本，最小化该样本输出值与真实值的差距，然后将修改过参数传给下一个样本，直到达到收敛条件。这样做参数更新频繁，也可能出现参数更改相互抵销的情况，于是便有了ABP。

ABP算法每次迭代会先算出所有样本的输出，然后最小化整个样本输出与真实值的最小平方和，修改参数后进行下一次迭代。ABP参数更新次数比BP算法少的多，但是当累计误差降到一定程度时，进一步下降会非常缓慢。

迭代终止条件：这里设置的终止条件是相邻一百次迭代的累计误差的差值不超过0.001。

BP算法结果：

在西瓜数据集3上迭代1596次迭代，使得累计误差达到0.0013，此时对比表为

输出值	真实值
0.000574239	0
0.003128893	0
0.001880475	0
0.016725407	0
0.019083811	0
0.993134077	1
0.993662799	1
0.977969458	1
0.990724507	1
0.987087467	1
0.005125069	0
0.008542622	0
0.023722006	0
0.99006993	1
0.99689923	1
0.989569681	1
0.986590243	1

```

x = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A1:Q8');
y = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A9:Q9');
x=x';
y=y';
%将y设为0, 1两类
y=y-1;
%获取输入参数的样本数与参数数
[m,n]=size(x);

t=1;    %输出层神经元

v=rand(n,n+1); %输入层与隐层的权值
w=rand(n+1,t); %隐层与输出层的权值
thy=rand(n+1); %隐层阈值
thj=rand(t);   %输出层阈值
ty=zeros(m,t); %输出层输出
b=zeros(n+1);  %隐层输出
gj=zeros(t);   %累计误差对w, thj求导的参数
eh=zeros(n+1); %累计误差对v, thy求导的参数
xk=1;          %学习率

kn=0;          %迭代次数
sn=0;          %同样的累计误差值累积次数
old_ey=0;      %前一次迭代的累计误差
while(1)
    kn=kn+1;
    ey=0;       %当前迭代的累计误差
    for i=1:m
        %计算隐层输出
        for j=1:n+1
            ca=0;

```

```

        for k=1:n
            ca=ca+v(k,j)*x(i,k);
        end
        b(j)=1/(1+exp(-ca+thy(j)));
    end
    %计算输出层输出
    for j=1:t
        cb=0;
        for k=1:n+1
            cb=cb+w(k,j)*b(k);
        end
        ty(i,j)=1/(1+exp(-cb+thj(j)));
    end
    %计算当前迭代累计误差
    for j=1:t
        ey=ey+((y(i)-ty(i,j))^2)/2;
    end
    %计算w, thj 导数参数
    for j=1:t
        gj(j)=ty(i,j)*(1-ty(i,j))*(y(i)-ty(i,j));
    end
    %计算v, thy 导数参数
    for j=1:n+1
        teh=0;
        for k=1:t
            teh=teh+w(j,k)*gj(k);
        end
        eh(j)=teh*b(j)*(1-b(j));
    end
    %更新v, thy
    for j=1:n+1
        thy(j)=thy(j)+(-xk)*eh(j);
        for k=1:n
            v(k,j)=v(k,j)+k*eh(j)*x(i,k);
        end
    end
    %更新thj, w
    for j=1:t
        thj(j)=thj(j)+(-xk)*gj(j);
        for k=1:n+1
            w(k,j)=w(k,j)+xk*gj(j)*b(k);
        end
    end
    %迭代终止判断
    if(abs(old_ey-ey)<0.0001)
        sn=sn+1;
        if(sn==100)
            break;
        end
    else
        old_ey=ey;
        sn=0;
    end
end

```

ABP：与BP算法最大的不同是参数在计算完全部样本才更改，由于ABP后期下降很慢，所以ABP的终止条件是50次相同的累计误差。经过1660次迭代，累计误差达到0.0015。

输出值	真实值
0.00337534523762459	0
0.00197885866471479	0
0.00379711463016395	0
0.0205696800567297	0
0.0208229173910228	0
0.994965085512552	1
0.989507102641039	1
0.966100739459433	1
0.993137548630017	1
0.993568720687235	1
0.00820831499581366	0
0.0106159051815879	0
0.0199653750774157	0
0.995719300147538	1
0.993319830282617	1
0.992253246322361	1
0.986823143409678	1

```

x = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A1:Q8');
y = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A9:Q9');
x=x';
y=y';
y=y-1;
[m,n]=size(x);
t=1;      %输出层神经元

v=rand(n,n+1); %输入层与隐层的权值
w=rand(n+1,t); %隐层与输出层的权值
thy=rand(n+1); %隐层阈值
thj=rand(t);   %输出层阈值
ty=zeros(m,t); %输出层输出
b=zeros(n+1);  %隐层输出
gj=zeros(t);   %累计误差对w,thj求导的参数
eh=zeros(n+1); %累计误差对v,thy求导的参数
tk=1;          %学习率

kn=0;          %迭代次数
sn=0;          %同样的累计误差值累积次数
old_ey=0;      %前一次迭代的累计误差
while(1)
    kn=kn+1;
    ey=0;%当前迭代的累计误差
    %计算全部样本输出层输出

```

```

for i=1:m
    %计算隐层输出
    for j=1:n+1
        ca=0;
        for k=1:n
            ca=ca+v(k,j)*x(i,k);
        end
        b(i,j)=1/(1+exp(-ca+thy(j)));
    end
    %计算输出层输出
    for j=1:t
        cb=0;
        for k=1:n+1
            cb=cb+w(k,j)*b(i,k);
        end
        ty(i,j)=1/(1+exp(-cb+thj(j)));
    end
end
%用来存累计误差对四个变量的下降方向
tv=zeros(n,n+1);
tw=zeros(n+1,t);
tthy=zeros(n+1);
tthj=zeros(t);
%计算累计误差
for i=1:m
    for j=1:t
        ey=ey+((y(i)-ty(i,j))^2)/2;
    end
    %计算w, thj 导数参数
    for j=1:t
        gj(j)=ty(i,j)*(1-ty(i,j))*(y(i)-ty(i,j));
    end
    %计算v, thy 导数参数
    for j=1:n+1
        teh=0;
        for k=1:t
            teh=teh+w(j,k)*gj(k);
        end
        eh(j)=teh*b(i,j)*(1-b(i,j));
    end

    %计算w, thj 导数
    for j=1:n+1
        tthy(j)=tthy(j)+(-1)*eh(j);
        for k=1:n
            tv(k,j)=tv(k,j)+k*eh(j)*x(i,k);
        end
    end
    %计算v, thy 导数
    for j=1:t
        tthj(j)=tthj(j)+(-1)*gj(j);
        for k=1:n+1
            tw(k,j)=tw(k,j)+gj(j)*b(i,k);
        end
    end
end
%更新参数
v=v+tk*tv;

```

```

w=w+tk*tw;
thy=thy+tk*tthy;
tthj=thj+tk*tthj;
%迭代终止条件
if(abs(old_ey-ey)<0.0001)
    sn=sn+1;
    if(sn==50)
        break;
    end
else
    old_ey=ey;
    sn=0;
end

end

```

## 6. 试设计一个BP改进算法，能通过动态学习率显著提升收敛速度。

1 这真是一个蛋疼的题，本来以为方法很多，结果没有一个能用的。

固定的学习率要么很慢要么在后期震荡，设计一种自适应的动态学习率算法是有必要的。

- 对四种参数的最速方向做一位搜索  
这是很直观的一种方法，已知 $f(x)$ 在 $x_0$ 的导数为 $d$ ，那么下降方向就是 $-d$ 。一位搜索就是求试 $f(x + td)$ 最小的 $t$ ，也就是当前的学习率。  
然而这方法的 $t$ 用解析法并不好求， $f'_t(x + td) = 0$ 也是无解的。  
使用近似方法尝试了下收敛速度并没有显著提升
- 对四种参数做牛顿迭代  
虽然不符合题目改学习率的要求，但是牛顿法肯定能大大提高收敛速度，只是没有了学习率这个概念。  
然而Hessian矩阵求的我想吐血，最后也没去继续了。

书中给出了两篇文献供参考

- Neural Networks: Tricks of the Trade
- Neural Smothing:Supervised learning in Feedforward Artificial Neural Networks

暂时不解这个题，等以后遇到好的方法再来更新。

7.根据式5.18和5.19，试构造一个能解决异或问题的RBF神经网络。

*RBF*是一种单隐层的前馈网络，它使用径向基函数作为隐层神经元激活函数，输出层则是对隐层的一个线性组合。

RBF隐层相当于把输入映射到了一个更高维空间，使得不可分的数据线性可分，所以隐层神经元至少比输入层多一个

式5.18:  $\varphi(x) = \sum w_i p(x, c_i)$

式5.19:  $p(x, c_i) = w^{-\beta_i \|x - c_i\|^2}$  (径向基函数)

这里以题目的要求来举例:

由于是异或问题，所以构造数据集

$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0

由此设计RBF网络

- 输入层：由于有2个输入，所以输入层2个神经元
- 隐层：隐层神经元越多拟合的越好，设为可变的t个，但至少要比输入层多1个。
- 输出层：1个神经元。

该网络的参数有

- $x, y$ : 样本参数
- $w_i$ : 隐层第i个神经元与输出神经元的权值
- $c_i$ : 隐层第i个神经元的中心
- $\beta_i$ : 样本与第i个神经元的中心的距离的缩放系数

$c_i$ 通过对x聚类或者随即采样获得。

下面使用ABP算法来确定参数 $w_i, \beta_i$ (由于是基于ABP来改的)

定义累计误差函数  $E = \sum_j \frac{1}{2} (\varphi(x_j) - y_j)^2$

那么:

$$\frac{\partial E}{\partial w_i} = \sum_j (\varphi(x_j) - y_j) p(x_j, c_i)$$

$$\frac{\partial E}{\partial \beta_i} = - \sum_j w_i (\varphi(x_j) - y_j) p(x_j, c_i) \|x_j - c_i\|^2$$

使用固定的学习率1。

然后通过ABP算法迭代。

经过144次迭代，累计误差达到0.000088，迭代终止，结果为：

$x_1$	$x_2$	$y$	输出
0	0	0	0.0035
0	1	1	0.9936
1	0	1	0.9972
1	1	0	0.0053

```
clear
x=[0 0;0 1;1 0;1 1];
y=[0;1;1;0];

t=10;    %隐层神经元数目 大于输入层
p=rand(4,t);    %径向基函数的值
ty=rand(4,1);    %输出值
w=rand(1,t);    %隐层第i个神经元与输出神经元的权值
b=rand(1,t);    %样本与第i个神经元的中心的距离的缩放系数
```

```

tk=0.5;

[id,c]=kmeans(x,2);
c = rand(t,2); %隐层第i个神经元的中心

kn=0; %来及迭代次数
sn=0; %同样的累计误差值累积次数
old_ey=0; %前一次迭代的累计误差

while(1)
    kn=kn+1;
    %计算每个样本的径向基函数值
    for i=1:4
        for j=1:t
            p(i,j)=exp(-b(j)*(x(i,:)-c(j,:))*(x(i,:)-c(j,:))') ;
        end
        ty(i)=w* p(i,:);
    end
    %计算累计误差
    ey = (ty-y)'*(ty-y);

    %更新w,b
    dw=zeros(1,t);
    db=zeros(1,t);
    for i=1:4
        for j=1:t
            dw(j)=dw(j)+(ty(i)-y(i))* p(i,j);
            db(j)=db(j)-(ty(i)-y(i))*w(j)*(x(i,:)-c(j,:))*(x(i,:)-
c(j,:))'*p(i,j);
        end
    end

    w = w - tk * dw / 4;
    b = b - tk * db / 4;
    %迭代终止条件
    if(abs(old_ey-ey)<0.0001)
        sn=sn+1;
        if(sn==10)
            break;
        end
    else
        old_ey=ey;
        sn=0;
    end
end

end

```



## 8.从网上下载或自己编程实现一个SOM网络，并观察在西瓜数据集3.0a上产生的结果。

SOM网络上真的就是草草介绍了下，顺便配了个看不懂的图。我去翻了下《MATLAB神经网络43个案例分析》这本书，里面对SOM讲的还是很明白的，包括原理与训练方法。

简单的说，SOM网络在平面放 $M \times N$ 个神经元( $M, N$ 自定义)，神经元按8邻接来连接。网络将输入层映射到平面上 $M \times N$ 个神经元上( $M, N$ 自定义)，每个输入与每个神经元都有一个初始权值，然后计算各个神经元的权值向量与输入向量的距离，选择距离最小的神经元更新他以及他8邻接神经元的权值向量，循环这个过程直到收敛。

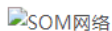
如果神经元比样本数多，足够多次迭代后所有样本会映射到了不同的神经元上。

网络生成后就通过输入算出离他最近的神经元，将它标记为该神经元的分类。

matlab中有现成的SOM网络，我就没自己编程了。

```
1 clear
2 simpleclassInputs = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A7:Q8');
3 net = newsom(simpleclassInputs,[8 8]); %64个神经元
4 %迭代次数1000次
5 net.trainParam.epochs=1000;
6 net = train(net,simpleclassInputs);
7 plotsompos(net,simpleclassInputs);
8 %获取每个样本映射到哪个神经元
9 sim(net,simpleclassInputs);
```

最终得到SOM网络为

SOM网络

64个神经元最终的位置，并可以查看样本映射到了哪个神经元上。

## 9.试推导用于Elman网络的BP算法。

Elman比正常网络多了个反馈，把前一次的 $b_h$ 作为隐层的输入来调节隐层。

假设用 $u_{ih}$ 来表示反馈输入与隐层连接的参数，由于前一次计算的 $b_h$ 作为常数输入， $u_{ij}$ 与 $v_{ij}$ 的计算方法一样， $\Delta u_{ih} = \eta e_h b_h$ ，其中 $e_h$ 书上5.15给出。就是相当于多了几个输入会变的输入层神经元。

## 10.实现一个卷积神经网络。

CNN作为深度学习一个重要工具，Stanford专门有个CS231n来讲CNN。等我把西瓜书弄完再回头解决这个问题，我感觉一时半会搞不定，还是先把机器学习基础的做完再说。

# 第六章 支持向量机

## 1.试证明样本空间中任意点 $x$ 到超平面 $(w, b)$ 的距离为式(6.2)。

超平面 $(w, b)$ 的平面法向量为 $w$ ，任取平面上一点 $x_0$ ，有 $w^T x_0 + b = 0$ 。 $x$ 到平面的距离就是 $x$ 到 $x_0$ 的距离往 $w$ 方向的投影，就是 $\frac{|w^T(x-x_0)|}{|w|} = \frac{|w^T x + b|}{|w|}$ 。

## 2.使用libsvm,在西瓜数据集3.0 $\alpha$ 上分别用线性核和高斯核训练一个SVM,并比较其支持向量的差别。

1 由于电脑的vs是2015的，而matlab上最高只支持2013来编译libsvm，所以只能在vs上用libsvm了

训练的结果是线性核与高斯核得到了完全一样的支持向量，由于没去分析libsvm内部是如何计算的，这里只贴结果。

第一列是支持向量的权值，后面则是支持向量对应的属性

$a_i$	$x_1$	$x_2$
1	0.697	0.46
1	0.744	0.376
1	0.634	0.264
1	0.608	0.318
1	0.556	0.215
1	0.403	0.237
1	0.481	0.149
1	0.437	0.211
-1	0.666	0.091
-1	0.243	0.267
-1	0.343	0.099
-1	0.639	0.161
-1	0.657	0.198
-1	0.36	0.37
-1	0.593	0.042
-1	0.719	0.103

## 3.选择两个UCI数据集，分别用线性核和高斯核训练一个SVM，并与BP神经网络和C4.5决策树进行实验比较。

使用的是iris数据集，选取其中分类为1，2的样本，各50个，4属性。

每类选前40个样本训练，后10个样本作为测试

线性核：找出3个支持向量

$a_i$	$x_1$	$x_2$	$x_3$	$x_4$
0.04	5.1	3.3	1.7	0.5
0.16	4.8	3.4	1.9	0.2
-0.20	4.9	2.5	4.5	1.7

偏置为1.50709

高斯核：找出9个支持向量

$a_i$	$x_1$	$x_2$	$x_3$	$x_4$
0.48	4.4	2.9	1.4	0.2
0.45	4.3	3	1.1	0.1
0.90	5.7	4.4	1.5	0.4
-0.17	6.3	3.3	6	2.5
-0.66	4.9	2.5	4.5	1.7
-0.03	6.5	3.2	5.1	2
-0.40	7.7	2.6	6.9	2.3
-0.15	6	2.2	5	1.5
-0.41	7.9	3.8	6.4	2

偏置为-0.212437

编写一个验证的matlab程序

读取数据

```
w1 = xlsread('C:\Users\icefire\Desktop\ml\tmp.xlsx', 'sheet1', 'A1:C1');  
w2 = xlsread('C:\Users\icefire\Desktop\ml\tmp.xlsx', 'sheet1', 'D1:L1');  
x1 = xlsread('C:\Users\icefire\Desktop\ml\tmp.xlsx', 'sheet1', 'A2:C5');  
x2 = xlsread('C:\Users\icefire\Desktop\ml\tmp.xlsx', 'sheet1', 'D2:L5');  
x = xlsread('C:\Users\icefire\Desktop\ml\tmp.xlsx', 'sheet1', 'A7:T10');
```

```
y1=zeros(20,1);
```

```
y2=zeros(20,1);
```

```
%验证线性核
```

```
for i=1:20
```

```
    for j=1:3
```

```
        y1(i)=y1(i)+w1(j)*(x(:,i)'+x1(:,j));
```

```
    end
```

```
end
```

```
y1=y1+1.50709;
```

```
%验证高斯核
```

```
for i=1:20
```

```
    for j=1:9
```

```
        y2(i)=y2(i)+w2(j)*exp(-0.25*(x2(:,j)-x(:,i))'+(x2(:,j)-x(:,i))));
```

```
    end
```

```
end
```

```
y2=y2-0.212437;
```

y1	y2
1.155205016	1.164557907
0.997242309	0.780919564
1.148298718	1.068992278
0.906038093	1.085988234
0.842638654	1.042299416
1.035492502	1.062906963
1.062585631	1.141980465
1.09304642	1.100071803
1.101546038	1.141056647
1.103671899	1.13405161
-1.839224592	-1.049900524
-1.542242776	-0.95432202
-1.471406411	-1.085122464
-1.958761346	-1.084832335
-1.895362864	-1.029274823
-1.608120552	-1.002438466
-1.448029593	-1.0262247
-1.519044109	-1.03794725
-1.658415695	-0.995288562
-1.402518712	-1.058306748

#### 4.讨论线性判别分析与线性核支持向量机在何种情况下等价。

在线性可分的情况下,LDA求出的 $w_l$ 与线性核支持向量机求出的 $w_s$ 有 $w_l \cdot w_s = 0$ ，即垂直，此时两者是等价的。

当初在做这个题的时候也没细想，就想当然的认为在线性可分时两者求出来的 $w$ 会垂直，现在看来并不一定。

首先，如果可以使用软间隔的线性SVM，其实线性可分这个条件是不必要的，如果是硬间隔线性SVM，那么线性可分是必要条件。这个题只说了是线性SVM，就没必要关心数据是不是可分，毕竟LDA是都可以处理的。

第二，假如当前样本线性可分，且SVM与LDA求出的结果相互垂直。当SVM的支持向量固定时，再加入新的样本，并不会改变求出的 $w$ ，但是新加入的样本会改变原类型数据的协方差和均值，从而导致LDA求出的结果发生改变。这个时候两者的 $w$ 就不垂直了，但是数据依然是可分的。所以我上面说的垂直是有问题的。

我认为这个题的答案应该就是，当线性SVM和LDA求出的 $w$ 互相垂直时，两者是等价的，SVM这个时候也就比LDA多了个偏移 $b$ 而已。

## 5.试述高斯核SVM与RBF神经网络的联系

RBF网络的径向基函数与SVM都可以采用高斯核，也就分别得到了高斯核RBF网络与高斯核SVM。神经网络是最小化累计误差，将参数作为惩罚项，而SVM相反，主要是最小化参数，将误差作为惩罚项。

在二分类问题中，如果将RBF中隐层数为样本个数，且每个样本中心就是样本参数，得出的RBF网络与核SVM基本等价，非支持向量将得到很小的 $w$ 。

使用LIBSVM对异或问题训练一个高斯核SVM得到 $\alpha$ ，修改第5章RBF网络的代码，固定 $\beta$ 参数为高斯核SVM的参数，修改每个隐层神经元的中心为各个输入参数，得到结果 $w, w$ 与 $\alpha$ 各项成正比例。

---

## 6.试析SVM对噪声敏感的原因。

SVM的目的是求出与支持向量有最大化距离的直线，以每个样本为圆心，该距离为半径做圆，可以近似认为圆内的点与该样本属于相同分类。如果出现了噪声，那么这个噪声所带来的错误分类也将最大化，所以SVM对噪声是很敏感的。

## 7.试给出式(6.52)的完整KT条件。

非等式约束写成拉格朗日乘子式，取最优解要满足两个条件

- 拉格朗日乘子式对所有非拉格朗日参数的一阶偏导为0
- 非等式约束对应的拉格朗日项，要么非等式的等号成立，要么对应的拉格朗日参数为0

所以得到完整KT条件

$$w = \sum_i (\alpha'_i - \alpha_i) x_i$$

$$0 = \sum_i (\alpha'_i - \alpha_i)$$

对所有的 $i$

$$C = \alpha_i + u_i$$

$$C = \alpha'_i + u'_i$$

$$\alpha_i (f(x_i) - y_i - \varepsilon - \xi_i) = 0$$

$$\alpha'_i (y_i - f(x_i) - \varepsilon - \xi'_i) = 0$$

$$(C - \alpha_i) \xi_i = 0$$

$$(C - \alpha'_i) \xi'_i = 0$$

## 8.以西瓜数据集3.0a的“密度”属性为输入，“含糖率”为输出，使用LIBSVM训练一个SVR。

含糖率和密度有什么必然联系吗？训练后得到的支持向量为

$\alpha_i$	密度 $x_i$
1	0.697
1	0.744
0.798	0.608
-1	0.666
0.452	0.243
-1	0.245
-0.25	0.343
1	0.36
-1	0.593
-1	0.719

偏置为 0.213589

得到含糖率与密度的关系

假设密度为 $x$

$$\text{含糖率}(x) = \sum_i \alpha_i e^{-(x-x_i)^2} + 0.213589$$

## 9

### 9.试使用技巧和推广对率回归，产生“核对率回归”。

这题相当于将第三章3.3题进行推广，思路基本和对率回归相当。

由表示定理可知，一般优化问题的解可 $h(x)$ 以写成核函数的线性组合。

$$\text{即: } h(x) = \sum_i w_i * k(x, x_i)$$

$$\text{可以推出 } w = \sum_i \alpha_i * \varphi(x_i)$$

其中 $\varphi(x)$ 是 $x$ 在更高维的映射

$$\text{由此可将3.22式中 } w^T x + b \text{ 改写为 } \sum_i \alpha_i * \varphi(x_i) * \varphi(x) + b = \sum_i \alpha_i * k(x, x_i) + b$$

令 $\beta = (\alpha; b)$ ,  $t'_i = (k_{:,i}; 1)$ , 其中 $k_{:,i}$ 表示核矩阵 $k$ 的第 $i$ 列

得到

$$l(\beta) = \sum_i (-y_i \beta^T t_i + \ln(1 + e^{\beta^T t_i}))$$

下面解法与对率回归完全一样。

这是使用高斯核产生的回归结果(图1 $\sigma = 0.5$ ,图2 $\sigma = 0.8$ )

```
%核对率回归 西瓜数据集3.0a
old_l=0; %记录上次计算的l
n=0; %计算迭代次数
b=zeros(18,1); %初始参数（自定义）
b(18)=1;
x = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.0.xlsx', 'sheet1', 'A1:Q2');
y = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.0.xlsx', 'sheet1', 'A4:Q4');
```

```

%保存核矩阵
k=ones(18);
%保存高斯核矩阵
for i=1:17
    for j=1:17
        k(i,j)=exp(-0.5*(x(:,i)-x(:,j))'*(x(:,i)-x(:,j)));
    end
end

while(1)
    cur_l=0;
    bx=zeros(17,1);
    %计算当前参数下的l
    for i=1:17
        bx(i) = b.'*k(:,i);
        cur_l = cur_l + ((-y(i)*bx(i)))+log(1+exp(bx(i)));
    end

    %迭代终止条件
    if abs(cur_l-old_l)<0.001
        break;
    end

    %更新参数(牛顿迭代法)以及保存当前l
    n=n+1;
    old_l = cur_l;
    p1=zeros(17,1);
    d1=0;
    d2l=0;

    for i=1:17
        p1(i) = 1 - 1/(1+exp(bx(i)));
        d1 = d1 - k(:,i)*(y(i)-p1(i));
        d2l = d2l + k(:,i) * k(:,i).'*p1(i)*(1-p1(i));
    end

    b = b - d2l\d1;
end

%绘制核速率回归的曲线
syms px py
fxy(px,py)=px-py+b(18);

for i=1:17
    fxy=fxy+ b(i)*exp(-0.5*((px-x(1,i))^2+(py-x(2,i))^2));
end
ezplot(fxy,[0 1]);
hold on;

%逐点画 分别表示是否好瓜
for i=1:17
    if y(i)==1
        plot(x(1,i),x(2,i),'+r');
        hold on;
    else if y(i)==0
        plot(x(1,i),x(2,i),'og');
        hold on;
    end
end

```

```

        end
    end
end
%计算出直线边界点 并绘制直线

xlabel('密度');
ylabel('含糖率');
title('核对率回归  $\sigma=0.5$ ');

```

## 10.设计一个显著减少SVM中支持向量数目而不显著降低泛化性能的方法。

对于线性的SVM，三个属性不完全一样的支持向量就能确定这个SVM，而其他的落在边缘上的点都可以舍弃。

还没太想明白，以后有机会更新。

## 第七章 贝叶斯分类器

### 1.试使用极大似然法估算西瓜数据集3.0中前3个属性的类条件概率。

极大似然法要先假定一种概率分布形式。

色泽：

对于好瓜，假设

$$P(\text{色泽}=\text{青绿}|\text{好瓜})=\sigma_1$$

$$P(\text{色泽}=\text{乌黑}|\text{好瓜})=\sigma_2$$

$$P(\text{色泽}=\text{浅白}|\text{好瓜})=\sigma_3=1-\sigma_1-\sigma_2$$

$$L(\sigma)=\prod_i P(\text{色泽}=x_i|\text{好瓜})=\sigma_1^3\sigma_2^4(1-\sigma_1-\sigma_2)$$

$$L'(\sigma_1)=\sigma_2^4\sigma_1^2(3-4\sigma_1-3\sigma_2)$$

$$L'(\sigma_2)=\sigma_1^3\sigma_2^3(4-4\sigma_1-5\sigma_2)$$

$$\text{令 } L'(\sigma_1)=0, L'(\sigma_2)=0 \text{ 得 } \sigma_1=\frac{3}{8}, \sigma_2=\frac{1}{2}, \sigma_3=\frac{1}{8}$$

可以看出 $\sigma_1, \sigma_2, \sigma_3$ 分别对应他们在样本中出现的频率。

对于坏瓜以及另外两种属性计算方式相同，得出类似的结果。



## 2.试证明：条件独立性假设不成立时，朴素贝叶斯分类器任有可能产生最优分类器。

朴素贝叶斯分类器就是建立在条件独立性假设上的。当有不独立的属性时，假如所有样本不独立的属性取值相同时分类也是相同的，那么此时朴素贝叶斯分类器也将产生最优分类器。

---

## 3.试编程实现拉普拉斯修正的朴素贝叶斯分类器，并以西瓜数据集3.0为训练集，并对“测1”样本进行分类。

```
x = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A1:Q8');
y = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A9:Q9');
%测试用例
test=x(:,1);
%各参数取值
pn=[3; 3; 3; 3; 3; 3; 2];

pc=0;    %为正的的概率
nc=0;    %为正的的概率
%对6种离散参数遍历
for i=1:6
    c=zeros(2,1);
    %累积次数，计算p(xi|c)
    for j=1:17
        if(x(i,j)==test(i))
            c(y(j))=c(y(j))+1;
        end
    end
    %取对数连乘变连加
    pc=pc+log((c(1)+1)/(8+pn(i)));
    nc=nc+log((c(2)+1)/(9+pn(i)));
end
%对2种连续参数遍历
p=[1.959,1.203;0.788,0.066];

for i=1:2
    pc=pc+log(p(i,1));
    nc=nc+log(p(i,2));
end
```

#### 4.实践中用式(7.15)决定分类类别时，若数据的维度非常高，则连乘的概率结果会非常接近0并导致下溢。试述防止下溢的可能方案。

若连乘的式子太多，导致乘积接近0。由于属性个数是已知的，可以对每个乘式做适当的开方处理，可以保证结果不会为0。另外也可以对各项取对数，当累加太多时，可能导致和接近负无穷。可以对每个加数除以属性的个数，来防止溢出。

#### 5.试证明:二分类任务中两类数据满足高斯分布且方差相同时，线性判别分析产生最优贝叶斯分类器。

假设1类样本均值为 $u_1$ ，2类样本均值为 $u_2$

由于数据满足同方差的高斯分布，当样本足够大时，可以认为

线性判别分析公式  $J = \frac{|w^T(u_1 - u_2)|^2}{w^T(\Sigma_1 + \Sigma_2)w}$  求最大值

对  $\frac{1}{J} = \frac{w^T(\Sigma_1 + \Sigma_2)w}{|w^T(u_1 - u_2)|^2} = \sum_i \frac{(1 - y_i)|w^T(x_i - u_1)|^2 + y_i|w^T(x_i - u_2)|^2}{|w^T(u_1 - u_2)|^2}$  求最小值

最优贝叶斯分类器使每个训练样本的后验概率  $P(c|x)$  最大，对应线性判别分析中，即离对应分类的中心距离(平方)除以两个分类中心的距离(平方)越小。

即求  $\sum_i \frac{(1 - y_i)|w^T(x_i - u_1)|^2 + y_i|w^T(x_i - u_2)|^2}{|w^T(u_1 - u_2)|^2}$  的最小值

两个式子相同，所以线性判别分析产生最优贝叶斯分类器。

#### 6.试编程实现AODE分类器，并以西瓜数据集3.0为训练集，并对“测试”样本进行分类。

```
x = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A1:Q6');
y = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A9:Q9');
```

%测试用例

```
test=x(:,1);
```

%各参数取值

```
pn=[3; 3; 3; 3; 3; 2];
```

```
pc=0; %为正的率
```

```
nc=0; %为正的率
```

%对6种离散参数遍历

```
for i=1:6
```

```
    c=zeros(2,1);
```

```
    %累积次数，计算p(xi|c)
```

```
    for j=1:17
```

```
        if(x(i,j)==test(i))
```

```
            c(y(j))=c(y(j))+1;
```

```
        end
```

```
    end
```

```

tpc=1;
tnc=1;
%累积次数, 计算p(xj|c,xi)
for j=1:6
    ct=zeros(2,1);
    for k=1:17
        if(x(i,k)==test(i) && x(j,k)==test(j))
            ct(y(j))=ct(y(j))+1;
        end
    end
    tpc = tpc * (ct(1)+1)/(c(1)+pn(j));
    tnc = tnc * (ct(2)+1)/(c(2)+pn(j));
end
pc=pc+(c(1)+1)/(8+pn(i))*tpc;
nc=nc+(c(2)+1)/(9+pn(i))*tnc;
end

```

## 7. 给定d个二值属性的分类任务，假设对于任何先验概率的估算需要30个样本。试估计AODE中估算先验概率 $p(c, x_i)$ 所需要的样本数。

显然对于正负样本，各属性对应的取值 $x_i$ 需要出现30次。

最好的情况下，只需要60个样本就能估算概率。其中30个 $x_i$ 属性的样本取值为1，30个 $x_i$ 属性的样本取值为0。尽管这不符合实际情况(相同属性取值不同)。

最坏的情况下，要60d个样本才能估算。其中每个样本只有一个属性和测试样本 $x_i$ 相同，其余都是另一个取值。

## 8. 考虑图7.3，证明：在同父结构中，若 $x_1$ 的取值未知，则 $x_3 \perp x_4$ 不成立。在顺序结构中， $y \perp z|x$ 成立，但 $y \perp z$ 不成立。

①.  $x_1$ 已知时,  $p(x_1, x_3, x_4) = p(x_1)p(x_3|x_1)p(x_4|x_1)$

$$p(x_3, x_4|x_1) = \frac{p(x_1, x_3, x_4)}{p(x_1)} = p(x_3|x_1)p(x_4|x_1)$$

所以 $x_3 \perp x_4|x_1$ 。

$x_1$ 未知时,  $p(x_1, x_3, x_4) = p(x_1)p(x_3|x_1)p(x_4|x_1)$

$$p(x_3, x_4) = \sum_{x_1} p(x_1)p(x_3|x_1)p(x_4|x_1) = \sum_{x_1} p(x_1)p(x_3|x_1)p(x_4|x_1)$$

由于不知道 $p(x_3|x_1)p(x_4|x_1)$ ，所以无法得出 $p(x_3, x_4) = p(x_3)p(x_4)$ 。

②.  $x$ 已知时,  $p(x, y, z) = p(z)p(x|z)p(y|x)$

$$p(y, z|x) = \frac{p(x, y, z)}{p(x)} = \frac{p(z)p(x|z)p(y|x)}{p(x)} = p(z|x)p(y|x)$$

所以 $y \perp z|x$

$x$ 未知时,  $p(x, y, z) = p(z)p(x|z)p(y|x)$

$$p(y, z) = \sum_x p(x, y, z) = p(z) \sum_x p(x|z)p(y|x)$$

无法得出 $p(y, z) = p(y)p(z)$

## 第八章 集成学习

1.假设硬币正面朝上的概率为 $p$ ，反面朝上的概率为 $1-p$ 。令 $H(n)$ 代表抛 $n$ 次硬币所得正面朝上的次数，则最多 $k$ 次正面朝上的概率为 $P(H(n) \leq k) = \sum_{i=0}^k \binom{n}{i} p^i (1-p)^{n-i}$ 。对 $\delta > 0$ ， $k = (p - \delta)n \leq e^{-2\delta^2 n}$ 试推导出(8.3)。

取 $p - \delta = \frac{1}{2}$ ，则 $\delta = p - \frac{1}{2} = \frac{1}{2} - \epsilon$

$$P(H(n) \leq \frac{n}{2}) = \sum_{i=0}^{\frac{n}{2}} \binom{n}{i} p^i (1-p)^{n-i} \leq e^{-2(\frac{1}{2}-\epsilon)^2 n} = e^{-\frac{1}{2}(1-2\epsilon)^2 n}$$

2.对于0/1损失函数来说，指数损失函数并非仅有的一致替代函数。考虑式(8.5)，试证明：任意随机函数 $l(-H(x)f(x))$ ，若对于 $H(X)$ 在区间 $[-\infty, \delta]$  ( $\delta > 0$ )上单调递减，则 $l$ 是0/1损失函数的一致替代函数。

$$\begin{aligned} \text{总损失 } L &= l(-H(x)f(x))P(f(x)|x) + l(-H(x))P(f(x) = 1|x) \\ &\quad + l(H(x))P(f(x) = 0|x) \end{aligned}$$

$H(x) \in [-1, 1]$ ，要使 $L$ 最小，当 $P(f(x) = 1|x) > P(f(x) = 0|x)$ 时，会希望 $l(-H(x)) < l(H(x))$ ，由于 $l$ 是递减的，得 $H(x) > -H(x)$ ，的 $H(x) = 1$ 。同理当 $P(f(x) = 1|x) < P(f(x) = 0|x)$ 时， $H(x) = -1$ 。

$l(-H(x)f(x))$ 是对 $H(X)$ 的单调递减函数，那么可以认为 $l(-H(x)f(x))$ 是对 $(-H(X))$ 的单调递增函数

此时 $H(x) = \operatorname{argmax}_{y \in \{0,1\}} P(f(x) = y|x)$ ，即达到了贝叶斯最优错误率，说明 $l$ 是0/1损失函数的一致替代函数。

### 3.自己编程实现一个AdaBoost,以不剪枝决策树为基学习器，在西瓜数据集3.0a上训练一个AdaBoost集成，并与图8.4比较。

AdaBoost通过改变样本的权重来生成下一个分类器。由于样本的权值不同，在决策树最优属性选择中，计算信息熵增益的时候，以及当树高达到2，对该节点进行分类时，不能直接计数，而是应该累积他们的权值。对第四章的决策树代码稍微修改，得到以下表格。由于没有很好的画图办法，所以以表格表示。

编号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	分类属性	阈值
样本	好	好	好	好	好	坏	坏	坏	坏	坏	好	好	好	坏	坏	坏	坏	无	无
1	好	好	好	好	好	好	好	好	坏	坏	好	好	好	坏	坏	坏	好	含糖率	0.126
2	好	好	好	好	好	好	好	好	坏	坏	好	好	好	坏	坏	坏	好	密度	0.3815
3	好	好	好	好	好	坏	好	坏	坏	坏	好	好	好	坏	坏	坏	好	密度	0.3815
4	好	好	好	好	坏	好	坏	好	坏	坏	好	好	好	坏	坏	坏	坏	含糖率	0.2045
5	好	好	好	好	好	好	好	好	坏	坏	好	好	好	坏	坏	坏	好	密度	0.3815
6	好	好	好	好	坏	坏	坏	坏	坏	坏	好	好	好	坏	坏	坏	坏	密度	0.3815
7	好	好	好	好	好	坏	好	坏	坏	坏	好	好	好	坏	坏	坏	好	密度	0.3815
8	好	好	好	好	好	坏	坏	坏	坏	坏	好	好	好	坏	坏	坏	坏	密度	0.6365
9	好	好	好	好	坏	坏	坏	坏	坏	坏	好	好	好	坏	坏	坏	坏	含糖率	0.3730
10	好	好	好	好	好	坏	坏	坏	坏	坏	好	好	好	坏	坏	坏	坏	密度	0.3815
11	好	好	好	好	好	坏	坏	坏	坏	坏	好	好	好	坏	坏	坏	坏	密度	0.5745
12	好	好	好	好	好	坏	坏	坏	坏	坏	好	好	好	坏	坏	坏	坏	密度	0.3815

每一行代表当前的累积分类器，一共设置了12个分类器。

每一列代表一个样本在某累积分类器下的取值。最后两列是每个分类器分类属性与阈值(信息增益决策树)

由于不能使用颜色，用红字的表示当前分类器分类错误的项。

可以看出在8个分类器时，累计分类器已经能0误差的分类，但是有9个时候，累计分类器在第5个样本产生了错误的分类，可以看出AdaBoost在分类器较少时的波动性。当分类器很多事，分类会趋于稳定。

下面是参考代码

```

global x y py res;
global tree ptr;

x = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A7:Q8');
y = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A9:Q9');
%将y映射到1,-1
y = -2*y+3;

[m,n]=size(y);
set=1:1:n;
%记录累积分类器的分类
sy=zeros(1,17);
%样本的权值，初始相同
st=ones(1,17)/17;
fenlei={'好','坏'};
shuxing={'密度','含糖率'};
%记录每次累积分类器的分类
res=cell(12,19);
%产生12个分类器
for i=1:12
    tree=zeros(1,100);
    ptr = 0;
    py=zeros(1,17);
    %生成决策树，返回根节点的最优属性与阈值
    [minn, threshold]=TreeGenerate(0,set,st,1);

```

```

%根据公式计算误差与当前分类器的权值，并更改样本权值
er=sum((py~=y).*st);
if(er>=0.5)
    break;
end
a=0.5*log((1-er)/er);
st=st.*exp(a*((py~=y)*2-1));
st =st /sum(st);
sy=sy+a*py;
for j=1:17
    res{i,j}=fenlei{(3-sign(sy(j)))/2};
end
res{i,18}=shuxing{minn};
res{i,19}=threshold;
end
end

```

下面是精简的决策树生成代码

```

%{
    parent:父节点编号
    curset:当前的样本编号集合
    st:样本权值
    height:最高高度
%}
function [k, threshold]=TreeGenerate(parent,curset,st,height)
global x y py;
global tree ptr;
%新增一个节点，并记录它的父节点
ptr=ptr+1;
tree(ptr)=parent;
cur = ptr;

%递归返回情况1:当前所有样本属于同一类
n = size(curset);
n = n(2);

%计算当前y的取值分类及各有多少个 如果只有一类表示属于同一类
cury=y(curset); %通过当前样本编号从所有样本中选出当前样本
y_data=unique(cury); %集合去重
y_nums=histc(cury,y_data); %统计各个取值各多少个

if(y_nums(1)==n)
    for i=1:n
        py(curset(i))=y_data;
    end
    return;
end

%递归返回情况2:属性已经全部划分还有不同分类的样本，或者所有样本属性一致但是分类不同(这时就是有错误的样本)

if(height==2)
    cst=st(curset);

```

```

        tans=(sum(cst.*cury)>0)*2-1;
        for i=1:n
            py(curset(i))=tans;
        end
        return;
    end

%主递归
%实现了4个最优属性选择方法,使用了相同的参数与输出:信息增益, 增益率, 基尼指数, 对率回归
%具体参数介绍间函数具体实现
%因为是相同的参数与输出, 直接该函数名就能换方法
[k, threshold] = entropy_paraselect(curset,st);
%连续属性只会分为两类 大于阈值一类 小于阈值一类 分类后继续递归
num = [1 1];
tmp_set = zeros(2,100);
for i=1:n
    if(x(k,curset(i))>=threshold)
        tmp_set(1,num(1))=curset(i);
        num(1) = num(1)+1;
    else
        tmp_set(2,num(2))=curset(i);
        num(2) = num(2)+1;
    end
end
for i=1:2
    tmp_set = intersect(tmp_set(i,:),curset); %由于用数组存编号,
    %会有0存在, 将样本分开后与当前的样本求交集 把0去掉
    TreeGenerate(cur,tmp_set,st,height+1);
end

end
m

```

和最优分类属性代码

```

%{
信息增益选择
    curset: 当前样本集合
    st: 样本权值
输出
    n: 最优属性
    threshold: 连续属性返回阈值
%}
function [n, threshold] = entropy_paraselect(curset,st)
global x y;
%通过样本编号与属性获取当前需要的样本
curx = x(:,curset);
cury = y(curset);
cst = st(curset);
all = size(cury); %当前样本总数
max_ent = -100; %为ent设初值, 要最大值, 所以设为一个很小的数

```

```

minn=0; %记录最优的属性编号
threshold = 0;
for i=1:2
    con_max_ent = -100;
    con_threshold = 0;
    xlist = sort(curx(i,:),2);
    %计算n-1个阈值
    for j=all(2):-1:2
        xlist(j) = (xlist(j)+xlist(j-1))/2;
    end
    %从n-1个阈值中选最优 (循环过程中ent先递减后递增 其实只要后面的ent比前面的大
    %就可以停止循环)
    for j=2:all(2)
        cur_ent = 0;
        %计算各类正负例数
        nums = zeros(2,2);
        for k=1:all(2)
            nums((curx(i,k)>=xlist(j))+1,(3-cury(k))/2) =
nums((curx(i,k)>=xlist(j))+1,(3-cury(k))/2) + cst(k);
        end
        %计算ent 连续属性只分两种情况
        for k=1:2
            if(nums(k,1)+nums(k,2) > 0)
                p=nums(k,1)/(nums(k,1)+nums(k,2));
                cur_ent = cur_ent + (p*log2(p+0.00001)+(1-p)*log2(1-p+0.00001))*
(nums(k,1)+nums(k,2))/all(2);
            end
        end
        %记录该分类最优取值
        if(cur_ent>con_max_ent)
            con_max_ent = cur_ent;
            con_threshold = xlist(j);
        end
    end
    %如果该最优取值比当前总的最优还要优，记录
    if(con_max_ent > max_ent)
        max_ent=con_max_ent;
        minn = i;
        threshold = con_threshold;
    end
end
n = minn;
end

```

#### 4.GradientBoosting是一种常用的Boosting算法，是分析其与AdaBoost的异同。

GradientBoosting与AdaBoost相同的地方在于要生成多个分类器以及每个分类器都有一个权值，最后将所有分类器加权累加起来

不同在于：

AdaBoost通过每个分类器的分类结果改变每个样本的权值用于新的分类器和生成权值，但不改变每个样本不会改变。

GradientBoosting将每个分类器对样本的预测值与真实值的差值传入下一个分类器来生成新的分类器和权值(这个差值就是下降方向)，而每个样本的权值不变。



## 5.试编程实现Bagging,以决策树桩为学习器，在西瓜数据集3.0a上训练一个Bagging集成，并与8.6进行比较。

Bagging采用可重复抽样的方式来生成子样本，然后通过子样本训练一个分类器，最终将所有分类器对样本进行预测，选择分类较多的一个。

这里用随机数的方式来模拟抽样，试了几种分布的随机数，泊松分布还算比较好，所以采用参数为1的泊松分布，然后将所有权值缩小到和为1。

由于每次都是随机数，所以每次结果不相同。当用12个分类器时，可能会出现0，1，2个分类误差。这是某次结果列表

编号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	分类属性	阈值
样本	好	好	好	好	好	坏	坏	坏	坏	坏	好	好	好	坏	坏	坏	坏	无	无
1	好	好	好	好	坏	好	坏	好	坏	好	好	好	好	坏	好	好	坏	含糖率	0.2045
2	好	好	好	好	坏	好	坏	好	坏	坏	好	好	好	坏	坏	好	坏	含糖率	0.1795
3	好	好	好	好	坏	好	坏	好	坏	坏	好	好	好	坏	坏	坏	坏	含糖率	0.2045
4	好	好	好	好	好	好	坏	好	坏	坏	好	好	好	坏	坏	坏	坏	密度	0.3815
5	好	好	好	好	好	好	坏	好	坏	坏	好	好	好	坏	坏	坏	坏	含糖率	0.2045
6	好	好	好	好	好	好	坏	好	坏	坏	好	好	好	坏	坏	坏	坏	含糖率	0.101
7	好	好	好	好	好	好	坏	好	坏	坏	好	好	好	坏	坏	坏	坏	密度	0.3815
8	好	好	好	好	好	好	坏	好	坏	坏	好	好	好	坏	坏	坏	坏	含糖率	0.2045
9	好	好	好	好	好	好	坏	坏	坏	坏	好	好	好	坏	坏	坏	坏	密度	0.3815
10	好	好	好	好	好	好	坏	坏	坏	坏	好	好	好	坏	坏	坏	坏	含糖率	0.2045
11	好	好	好	好	好	好	坏	坏	坏	坏	好	好	好	坏	坏	坏	坏	密度	0.3515
12	好	好	好	好	好	坏	好	坏	坏	坏	好	好	好	坏	坏	坏	坏	密度	0.3815

但从训练误差来说Bagging并不如AdaBoost。Bagging可以很大程度上降低方差，但却不降低训练误差。AdaBoost同时降低训练误差与方差，但没有Bagging那么好的多样性。

下面是Bagging的代码，决策树与参数选择的代码没变。参考8.3

```
global x y py;
global tree ptr;

x = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A7:Q8');
y = xlsread('C:\Users\icefire\Desktop\ml\西瓜3.xlsx', 'sheet1', 'A9:Q9');
y = -2*y+3;
```

```

[m,n]=size(y);
set=1:1:n;
%记录累积分类器的分类
sy=zeros(1,17);
%样本的权值，初始相同
st=ones(1,17)/17;
fenlei={'好','坏'};
shuxing={'密度','含糖率'};
%记录每次累积分类器的分类
res=cell(12,19);
%产生12个分类器
for i=1:12
    %随机权值并缩小
    st=abs(poissrnd(1,1,17));
    st=st/sum(st);

    tree=zeros(1,100);
    ptr = 0;
    py=zeros(1,17);
    %生成决策树，返回根节点的最优属性与阈值
    [minn,threshold]=TreeGenerate(0,set,st,1);
    sy=sy+py.*st;
    for j=1:17
        res{i,j}=fenlei{fix((3-sign(sy(j)))/2)};
    end
    res{i,18}=shuxing{minn};
    res{i,19}=threshold;
end

```

## 6.试述为什么Bagging难以提升朴素贝叶斯分类器的性能。

Bagging主要是降低分类器的方差，而朴素贝叶斯分类器没有方差可以减小。对全训练样本生成的朴素贝叶斯分类器是最优的分类器，不能用随机抽样来提高泛化性能。

## 7.试述随即森林为什么比决策树Bagging集成的训练速度快。

随机森林不仅会随机样本，还会在所有样本属性中随机几种出来计算。这样每次生成分类器时都是对部分属性计算最优，速度会比Bagging计算全属性要快。

## 8.MultiBoosting算法与Iterative Bagging的优缺点。

MultiBoosting由于集合了Bagging, Wagging, AdaBoost, 可以有效的降低误差和方差，特别是误差。但是训练成本和预测成本都会显著增加。

Iterative Bagging相比Bagging会降低误差，但是方差上升。由于Bagging本身就是一种降低方差的算法，所以Iterative Bagging相当于Bagging与单分类器的折中。

## 9.试设计一种可视化多样性度量，并与k-误差图比较。

暂无

---

## 10.试设计一种能提升k近邻分类器性能的集成学习算法。

可以使用Bagging来提升k近邻分类器的性能，每次随机抽样出一个子样本，并训练一个k近邻分类器，对测试样本进行分类。最终取最多的一种分类。