全变差折半剪枝和熵折半剪枝的实验对比

机器学习第四周作业

樊泽羲 2200010816

问题定义

考虑如下定义的监督学习任务：

$$\begin{aligned}&Dataset: D=\{x_i,y_i\}_{i=1}^N, \quad where\ y_i\in\{1,2,\ldots,M\}, 1\le i\le N\\&Goal: find\ classifier\ f(x_i)=y_i\end{aligned}$$

在决策树模型中,这个映射函数f表示为一个树形结构，在树中的内部节点检查若干特征的取值，分支表示这些检查的结果。当结果满足一定阈值条件时，中止扩展，所得结构成为叶子节点，在其中以多数表决制生成节点数据的类标签

$$\begin{aligned}&Category: C=\{C_1,C_2,\cdots,C_K\}\\&Prediction: \tilde{y}_i=argmax_{1\le m\le M}|D_m|/|D|, \quad where\ D\ is\ the\ node\ containing\ x_i\end{aligned}$$

其中，特征集合定义如下：

$$\begin{aligned}&Feature: \mathscr{A}=\{A_1,A_2,\cdots,A_p\}\\&Each\ Feature: A_g=\{a_1^g,a_2^g,\cdots,a_{n_g}^g\}, 1\le g\le p\end{aligned}$$

从具体过程来看，该分类任务须完成如下目标：

1. 在训练决策树T的每一步,应当对特征空间进行进一步细分，使得每个划分区域内某个特征C_j的纯度尽可能最大化(纯度的度量依赖于所选的损失函数)
2. 在测试阶段，每给定一个无标签的实例x_i，决策树沿根节点向下访问每层的节点，应用该节点的分类规则，直至访问至叶子节点，给出实例的一条分类路径，我们希望这条分类路径可以正确地给出$f(x_i)=y_i$
3. 在同等精确度下，采用结构风险最小化准则，尽可能地选取复杂度小的模型。考虑到树形结构，可以对其采取剪枝策略
4. 特别地，要求决策树在对最优特征下分出的不同子节点进行剪枝时，不是全部舍弃或者全部保留，而是采取更为灵活的剪枝策略

算法设计

为尽可能最小化结构风险，我们采取折半剪枝的策略，即：每次预剪枝从最优特征A(根据信息增益$g(D,A)$选取)中选择一半(向上取整)的a_j保留作为子节点。由于特征取值数量大大减少，留下的节点必须充分体现最优特征A对分类目标提供的后验知识。为此提出两个方案：

1. C4.5:采用信息增益率$g_R(D,a_j)$作为预剪枝的选择标准（熵折半剪枝）

   理由：与最优特征A的选取方式保持一致。而且在选取节点的步骤采取信息增益率而非单纯的信息增益，在一定程度上起到 了正则化的作用，可以防止决策树总倾向于选择取值较多的特征，导致结构风险增大

2. TV_ID3:采用全变差增益$\tilde{g}_R(D,a_j)$作为预剪枝的选择标准（全变差折半剪枝）

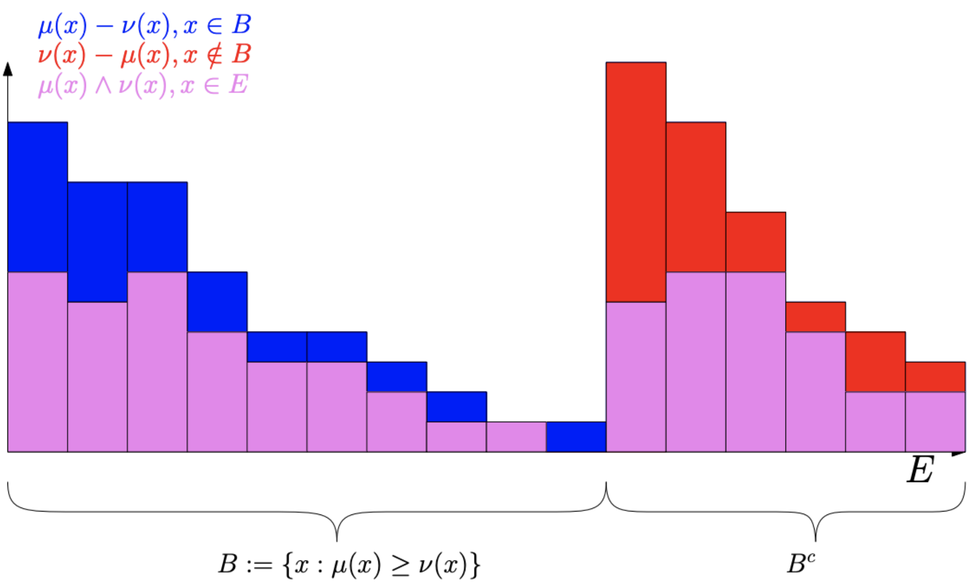
   理由：全变差是一种L_1范数，而L_1范数总体上对单个样本点更加敏感。如果我们可以假定，最优特征同时也是样本噪声最少的特征(否则，相关性可能会减弱，从而导致其增益小于其它特征)，那么当我们度量每个a_j的重要性时，就应当尽可能地保存每个样本点的贡献，此时就需要全变差这样更加敏感的度量

下面具体说明如何用全变差距离度量不确定性。全变差的定义如下：

$\mathcal{Definition}$ Given distributions μ_A, μ_B, the **total variation** between μ_A, μ_B is defined as:

$$||\mu_A - \mu_B|| = sup_{S\subset\mathbb{E}}|\mu_A(S) - \mu_B(S)| = \frac{1}{2}\sum_{x\in\mathbb{E}}|\mu_A(x) - \mu_B(x)|$$

这里$\mathbb{E}$代表所关心的随机实验 [1]



$$||\mu - \nu||_{TV} = \tfrac{1}{2}\Sigma_{x\in E}\,|\mu(x) - \nu(x)| = \Sigma_{x\in B}(\mu(x) - \nu(x)) = 1 - \Sigma_{x\in E}(\mu(x) \wedge \nu(x)).$$

考虑到信息熵度量的实际是某种分布与均匀分布(随机性最大)的相似程度，类比地，我们可以定义：

$$\mathcal{Definition}\ \textbf{TV information}:\tilde{I}(D|a_j) = \sum_{i=1}^{K}|p(c_i|a_j) - \frac{1}{K}|$$

$$\textbf{TV loss}:\tilde{H}(D|a_j) = -\tilde{I}(D|a_j)$$

在上式中，$TV\ information$度量的同样是已知a_j下的后验分布与先验分布（即：$\mathbb{P}(c=c_i)=\frac{1}{K}$，相当于盲猜）的距离。当最优特征$A$取定时，对于每个取值$a_j$, $TV\ information$越大，说明"是否为a_j"这一知识提供的后验信息越多，子节点a_j越适合保留。这样$\tilde{H}(D|a_j)$就起到了和$H(D|a_j)$(条件熵)类似的功能

两种距离可以通过$\mathbf{Fannes–Audenaert}$不等式[2]联系起来：

$$||\mu_A-\mu_B||\leq\epsilon\leq\frac{1}{2}\Rightarrow|H(A)-H(B)|\leq H(\epsilon)+\epsilon\cdot\log{(N-1)}$$

其中N代表随机变量的维数

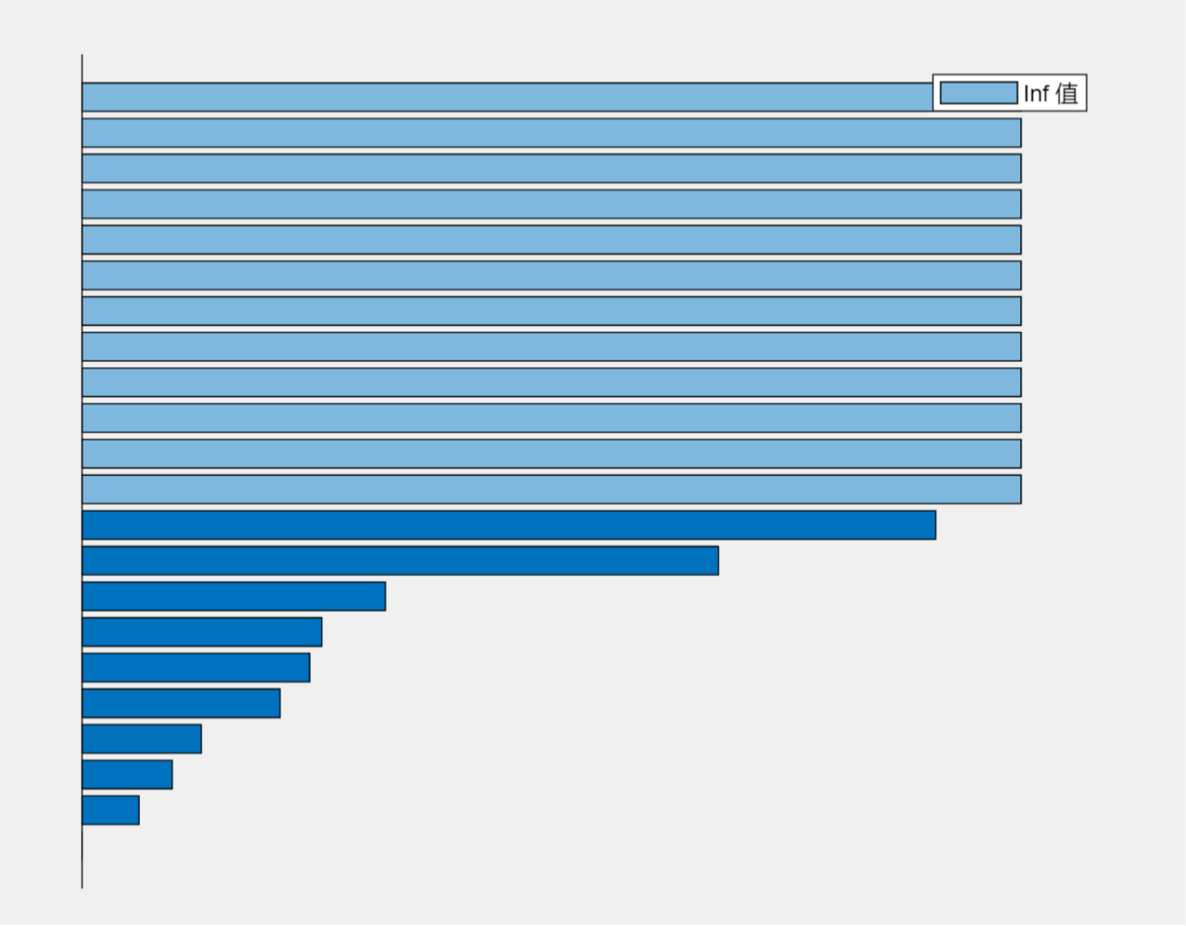
在我们的情况中，可以视$\mu_B, H(B)$为固定值。从而，当全变差充分小时，熵也会被很好地控制，但是反之不然，从而前者确实是比后者更为精细的距离刻画

同时由上式可见，如果从特征集$\mathscr{A}$中选择了与类标签c_i低度相关的特征A,那么全变差很可能先失效，而熵仍然可以比较好地刻画各个子节点a_j的保留价值

有关算法的具体实现，见"实验设置"

数据集处理

由于折半剪枝可能会迅速减少特征的取值数量，我们需要寻找特征信息尽量多的数据集。为此，采用Mushroom数据集（8124个数据，22个特征，2个类别）[3],利用χ^2检验计算因子权重如下：



可见，诸特征与类标签的关联程度差异较为显著，在22个特征中，有大约14种与类标签高度相关，其余8种均低度相关。因此在决策树的扩展过程中，仍然有$\frac{4}{11}$的特征可能会削弱全变差的度量效果。基于此，我们推测，在Mushroom数据集上，C4.5可能比TV_ID3取得更好的效果

由于matlab中数值矩阵的操作更为丰富，实验前先将.data文件中各列的不同类别用正整数进行标号，并存储在my_data数值矩阵中（见代码）

实验设置

对照组：CART（rough tree,无剪枝,通过matlab分类学习器实现）

实验组1：TV_ID3（折半预剪枝）

Algorithm 1: 选择最优特征（利用信息增益）

输入：数据集D,特征集$\mathscr{A}$

输出：最优特征A

1.计算经验熵：$H(D)=-\sum_{k=1}^{K}\frac{|C_k|}{|D|}\log_2\frac{|C_k|}{|D|}$

2.计算各个特征A_g的对数据D的条件熵：$H(D|A_g)=-\sum_{i=1}^{n}\frac{|D_{ik}^g|}{|D_i^g|}\sum_{k=1}^{K}\frac{|D_{ik}^g|}{|D_i^g|}$

3.计算各个特征A_g的信息增益：$g(D,A_g)=H(D)-H(D|A_g)$

4.选择增益最大者作为最优特征：$A=argmax_{A_g\in\mathscr{A}}g(D,A_g)$

Algorithm 2: 选择预剪枝中保留的子节点（通过全变差增益）

输入：数据集D,最优特征A

输出：保留的子节点$\{a_{i_1},a_{i_2},\cdots,a_{i_h}\}$

1.计算保留节点数：$h=\lceil\frac{n}{2}\rceil$

2.计算经验$TV\ loss$：$\tilde{H}(D)=-\sum_{k=1}^{K}|\frac{|C_k|}{|D|}-\frac{1}{K}|$

3.计算各个取值a_j的条件$TV\ loss$：

$\tilde{H}(D|a_j)=-\frac{|D_{a_j}|}{|D|}\sum_{k=1}^{K}|\frac{|D_{a_j,k}|}{|D_{a_j}|}-\frac{1}{K}|-\frac{|\widehat{D_{a_j}}|}{|D|}\sum_{k=1}^{K}|\frac{|\widehat{D_{a_j,k}}|}{|\widehat{D_{a_j}}|}-\frac{1}{K}|$

其中$\widehat{D}$代表取值不是a_j的组，D代表取值是a_j的组

4.计算各个取值a_j的全变差增益：$\tilde{g}(D,a_j)=\tilde{H}(D)-\tilde{H}(D|a_j)$

5.选择增益最大的h个子节点保留，其余剪去：$\{a_{i_1},a_{i_2},\cdots,a_{i_h}\}=argmaxk_{a_j\in A,k=h}\tilde{g}(D,a_j)$

其中$maxk$代表选取最大的k个

Algorithm 3: ID3扩展 (折半预剪枝)

输入：数据集D,特征集$\mathscr{A}$,阈值ϵ

输出：决策树T

1.若D中所有实例属于同一类C_k，则T为单节点树，并将类C_k作为该节点的类标记，返回T

2.若$\mathscr{A}=\emptyset$,则T为单节点树，并将D中实例数最大的类C_k作为该节点的类标记，返回T

3.否则，按Algorithm 1选择最优特征A

4.若A的增益小于阈值ϵ,则置T为单节点树，并将D中实例数最大的类C_k作为该节点的类标记，返回T

5.否则，调用Algorithm 2选择A中一半的子节点进行保留，其余实行剪枝

6.对于第j个子节点a_j，以D_{a_j}作为训练数据集，以$\mathscr{A}-A$为特征集，递归地调用步骤1到5，得到子树T_i，返回T_i

实验组2： C4.5（折半预剪枝）

主体框架与TV_ID3相同，但是在预剪枝时采用信息增益率$g_R(D,a_j)$作为选择标准，此处从略

各组的测试方法：选择全部的22个特征进行训练，采用10折交叉验证法，最终准确率取在10折测试数据集上的平均得分

实验结果

检测指标：

1.混淆矩阵：表示机器对p,e两类的混淆情况，主对角线颜色越深，分类器越理想

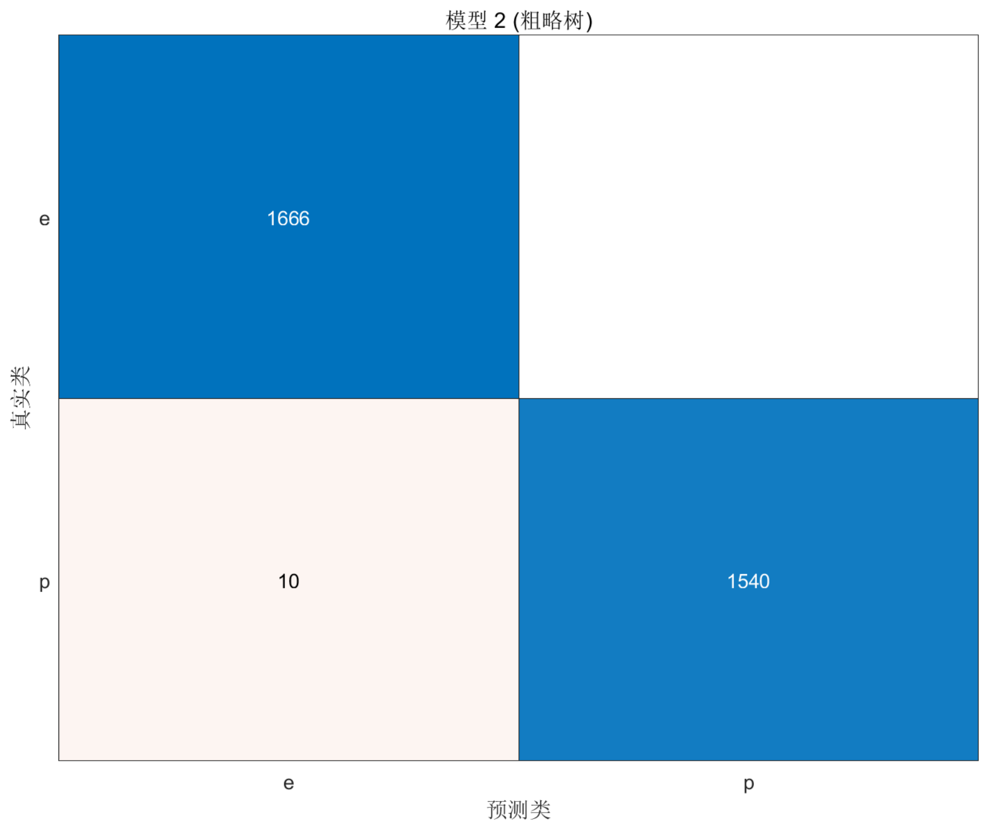
2.十折准确率：蓝色代表在训练集上验证的结果，红色代表在测试集上验证的结果

3.平均准确率：测试集准确率的平均，作为模型的总准确率

所得结果如下：

1. CART（无剪枝）

   混淆矩阵：

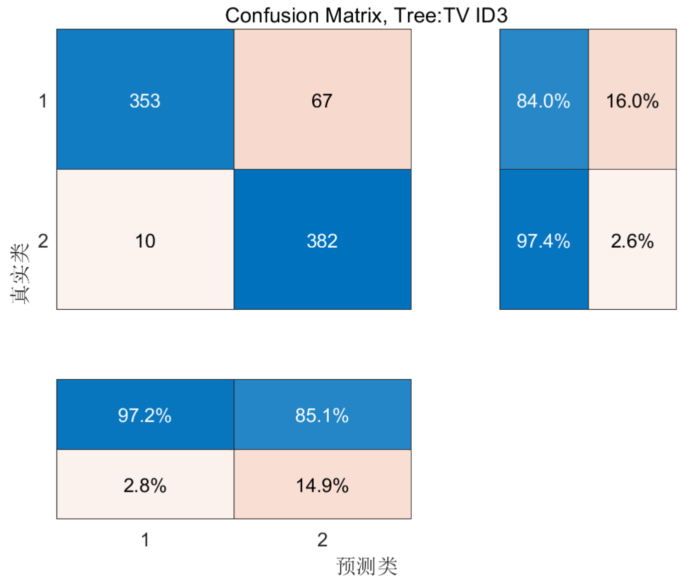


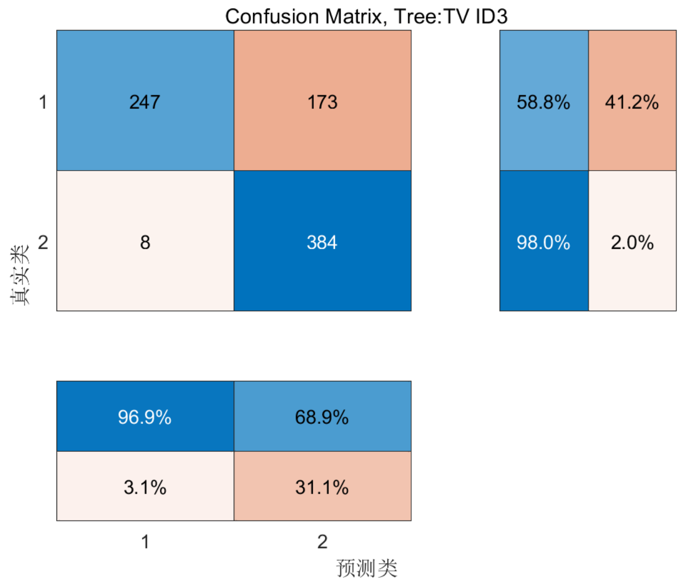
平均准确率：99.7%

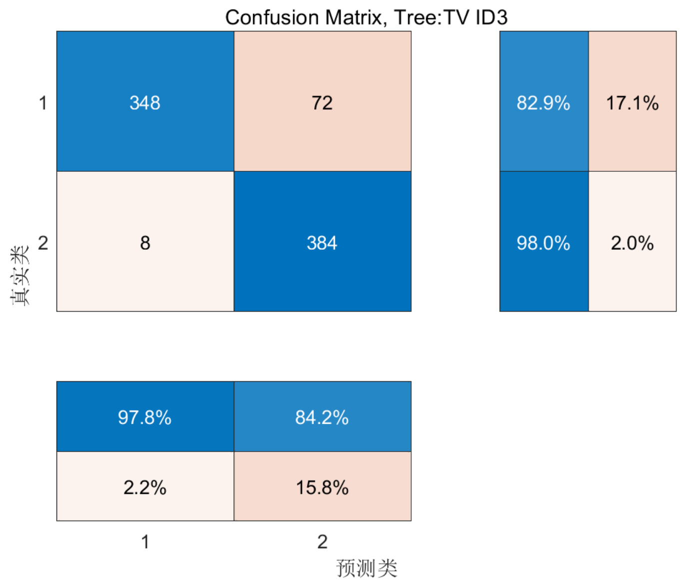
(因为CART采用的是matlab中的默认参数，并且表现十分稳定，因此这里展示的是在十折叠加在一起的结果)

2. TV_ID3（折半剪枝）

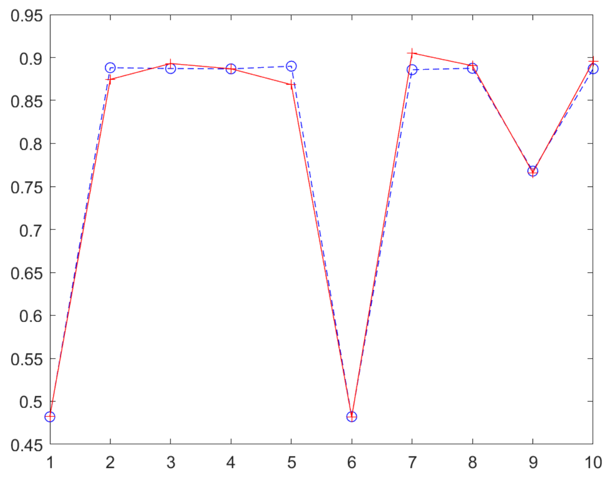
混淆矩阵：



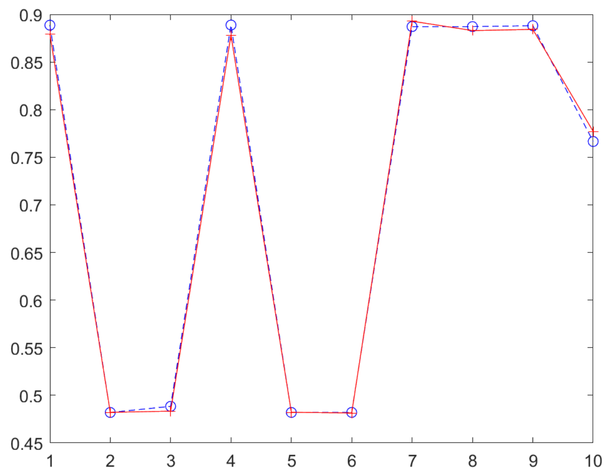




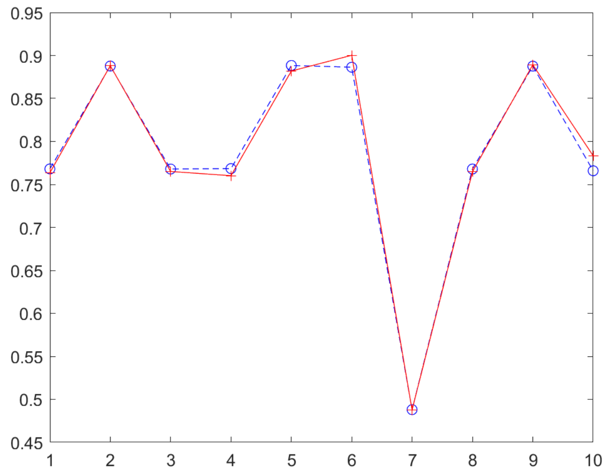
十折准确率：



0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
1
2
3
4
5
6
7
8
9
10



0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
1
2
3
4
5
6
7
8
9
10

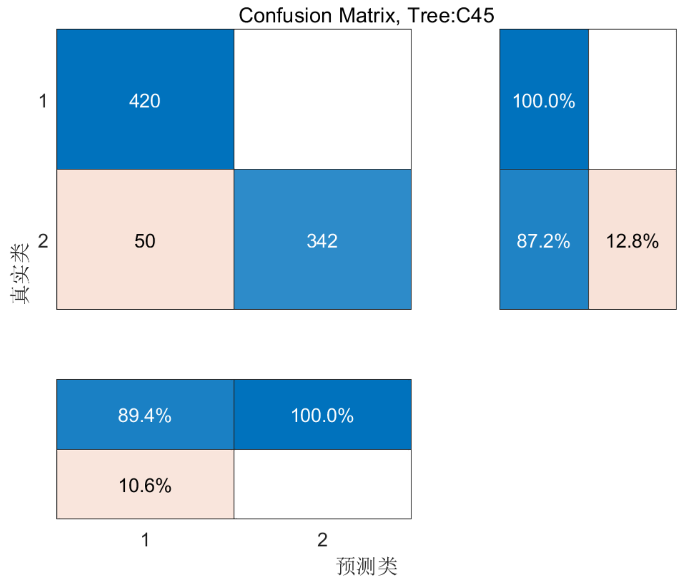


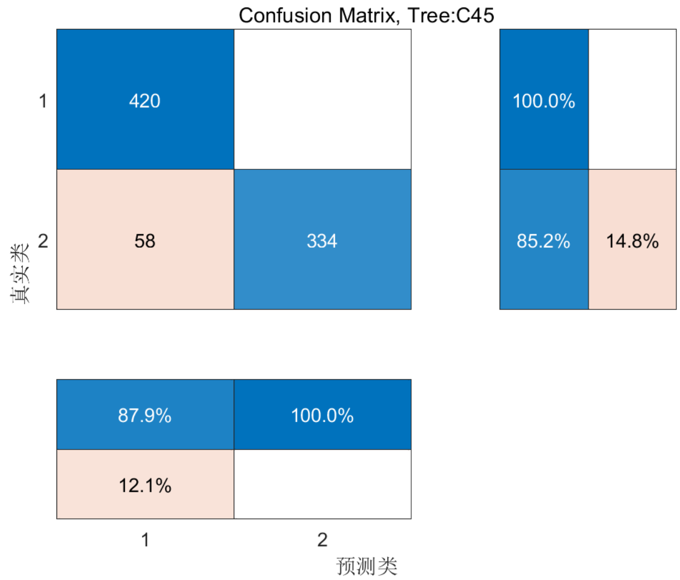
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
1
2
3
4
5
6
7
8
9
10

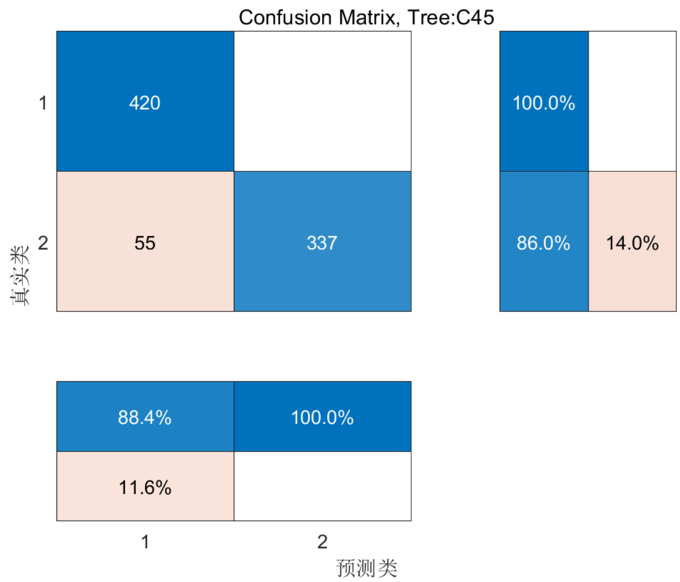
平均准确率：81.22%

3. C4.5（折半剪枝）

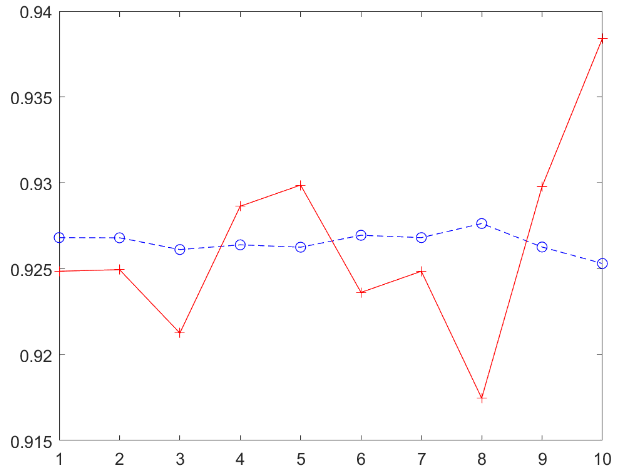
混淆矩阵：

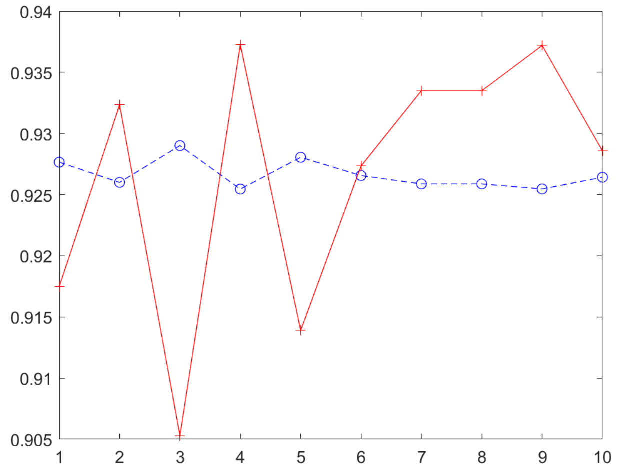


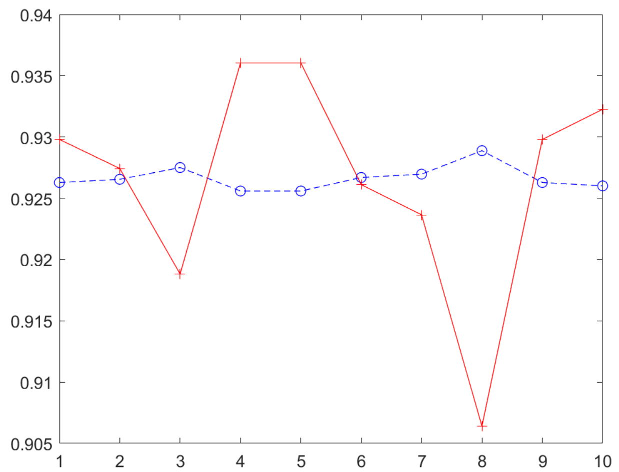




十折准确率：


0.94
0.935
0.93
0.925
0.92
0.915
1
2
3
4
5
6
7
8
9
10


0.94
0.935
0.93
0.925
0.92
0.915
0.91
0.905
1
2
3
4
5
6
7
8
9
10


0.94
0.935
0.93
0.925
0.92
0.915
0.91
0.905
1
2
3
4
5
6
7
8
9
10

平均准确率：92.66%

结果分析：

1.对比无剪枝网络和折半剪枝网络：

由于折半剪枝删除了大量特征信息，因此无剪枝CART的表现总体上优于折半剪枝的TV_ID3和C4.5。但是，即使是在折半这样稀疏连接的情况下，两个网络仍然都保持了80%以上的准确率，其中C4.5更是高达92.66%，说明当决策树规模较大或者成本需要纳入考虑的情况下，我们的两种剪枝策略确实有一定的可取之处

2.对比两个折半剪枝网络：

首先，从总体准确率上来看，采用熵选择节点的C4.5在稳定性（即十折准确率图线的波动程度，注意纵坐标的尺度的差别）和平均准确率方面均优于TV_ID3,这可能是由于在Mushroom数据集中存在与分类标签相关度较低的特征，而决策树在后期扩展过程中选择了8个相关度较低的特征，导致全变差方法更先失效。这个结果与我们在"特征选择"部分的最后提出的猜想是相符的

其次，从实用性上来看，对Mushroom数据集分类的目的是区分蘑菇是否可食用，因此假阴性实际上比假阳性更加致命，因此我们倾向于选择更为保守的分类器。由混淆矩阵左下角的一块可见，TV_ID3的假阴性率显著地小于C4.5，假阴性样本个数保持在10个及以下，和无剪枝的CART相近。因此在Mushroom分类这一任务中，从人的安全出发，TV_ID3可能是折半剪枝情况下对CART更好的替代

总结

实验描述:我们在Mushroom数据集上进行了10折交叉验证，比较CART (no pruning)、TV_ID3 (half pruning using total variance gain)和C4.5 (half pruning using information gain ratio)的性能。实验结果包括平均准确率、混淆矩阵和十折准确率

特征分析:Mushroom数据集包含8124个样本和22个特征。卡方检验表明，约有14个特征与目标类高度相关，其余8个特征弱相关。我们猜测，当在构建决策树时选择弱相关的特征时，全变差可能会在早期失去作用

实验分析:通过比较结果，我们可以看到不进行剪枝的CART表现最好，平均准确率为99.7%。这是因为剪枝会删除大量的特征信息。然而，TV_ID3和C4.5在半剪枝下的平均准确率仍在80%以上，其中C4.5达到92%以上，表明了剪枝策略的经济性

C4.5具有更好的稳定性，平均准确率为92.66%，优于TV_ID3的81.22%。这验证了最初的假说，即当选择弱相关特征时，总方差会更早失去作用，而在这种情况下，熵仍然可以更好地表征节点价值

回归特点:从实用角度来看，对蘑菇分类问题，假阴性比假阳性的更加危险。基于混淆矩阵的TV_ID3算法的漏报率远低于C4.5算法，接近无剪枝的CART算法。因此，考虑到人身安全，TV_ID3可能是成本有限情况下对CART更好的替代

总之，本文利用全变差增益进行折半剪枝，开发了两种新的决策树TV_ID3(half)和C4.5(half)。在Mushroom数据集上的实验结果表明了剪枝策略的经济性，并揭示了不同技术的优缺点。这实现了问题中概述的目标。

附录

计算信息增益：info_gain.m

```
function info_gain=info_gain(train_data,feature)
    %input: train_data:array,with last column labels
           %feature:double,corresponding to the index of feature in dat
    %output:information gain of train_data on feature
    classes=unique(train_data(:,end));
    K=length(classes);
    D=size(train_data,1);
    k=0;
```

```
    empirical_entropy_array=zeros(K,1);
    for class=classes'
        k=k+1;
        C_k=sum(train_data(:,end)==class);
        empirical_entropy_array(k)=-(C_k/D)*log2(C_k/D);
    end
    empirical_entropy=sum(empirical_entropy_array);
    feature_classes=unique(train_data(:,feature));
    n=length(feature_classes);
    conditional_entropy_array=zeros(n,1);
    i=0;
    for feature_class=feature_classes'
        i=i+1;
        K_array=zeros(K,1);
        D_i_data=train_data(find(train_data(:,feature)==feature_class),:);
        D_i=size(D_i_data,1);
        for k=1:K
            D_ik=sum(D_i_data(:,end)==k);
            if D_ik~=0
                K_array(k,1)=-(D_ik/D_i)*log2(D_ik/D_i)*(D_i/D);
            end
        end
        conditional_entropy_array(i)=sum(K_array);
    end
    conditional_entropy=sum(conditional_entropy_array);
    info_gain=empirical_entropy-conditional_entropy;
end
```

计算信息增益率：info_gain_ratio.m

```
function info_gain_ratio=info_gain_ratio(train_data,feature)
    %input: train_data:array,with last column labels
           %feature:double,corresponding to the index of feature in dat
    %output:information gain ratio of train_data on feature
    info_gain_j=info_gain(train_data,feature);
    D=size(train_data,1);
    feature_classes=unique(train_data(:,feature));
    H_A_D=zeros(length(feature_classes),1);
    i=0;
    for feature_class=feature_classes'
        i=i+1;
        D_i=size(train_data(find(train_data(:,feature)==feature_class)),1);
        H_A_D(i)=-(D_i/D)*log2(D_i/D);
    end
    deno=sum(H_A_D);
    info_gain_ratio=info_gain_j/deno;
end
```

计算全变差增益：TV_gain.m

```
function TV_gain=TV_gain(train_data,feature)
    %input: train_data:array,with last column labels
           %feature:double,corresponding to the index of feature in dat
    %output:TV gain of train_data on feature
    classes=unique(train_data(:,end));
    K=length(classes);
```

```
    D=size(train_data,1);
    empirical_TV_array=zeros(K,1);
    k=0;
    for class=classes'
        k=k+1;
        C_k=sum(train_data(:,end)==class);
        empirical_TV_array(k)=-abs((C_k/D)-(1/K));
    end
    empirical_TV=sum(empirical_TV_array);
    feature_classes=unique(train_data(:,feature));
    n=length(feature_classes);
    conditional_TV_array=zeros(n,1);
    i=0;
    for feature_class=feature_classes'
        i=i+1;
        K_array=zeros(K,1);
        D_i_data=train_data(find(train_data(:,feature)==feature_class),:);
        D_i=length(D_i_data);
        for k=1:K
            D_ik=sum(D_i_data(:,end)==k);
            K_array(k)=-(D_i/D)*abs((D_ik/D_i)-(1/K));
        end
        conditional_TV_array(i)=sum(K_array);
    end
    conditional_TV=sum(conditional_TV_array);
    TV_gain=empirical_TV-conditional_TV;
end
```

调用决策树，分类单个样本：tree_classify_single.m

```
function class=tree_classify_single(tree,data)
%input:tree:structure,a trained tree, C45 or TV_ID3
      %data:array,one piece of data to classify
%ouput:class:double,classification result
        piece=data;
        class=0;
        if strcmp(tree.type,'leaf')
            class=tree.label;
        else
            for j=1:numel(tree.children)
                a_child=tree.children{j};
                if ~isempty(a_child)
                    feature=a_child.criterion(1,1);
                    region=a_child.criterion(1,2);
                    if j<=numel(tree.children)-1
                        if piece(1,feature)==region
                            class=tree_classify_single(a_child,piece);
                        end
                    else
                        if class==0
                            class=tree_classify_single(a_child,piece);
                        end
                    end
                else
                    continue;
                end
            end
```

```
        end
end
```

调用决策树分类多个样本：tree_classify.m

```
function classes=tree_classify(tree,data)
%input:tree:structure,a trained tree, C45 or TV_ID3
      %data:array,one piece of data to classify
%ouput:class:array,classification result
    classes=zeros(size(data,1),1);
    for k=1:length(classes)
        datum=data(k,:);
        class=tree_classify_single(tree,datum);
        classes(k,1)=class;
    end
end
```

绘制混淆矩阵：confu.m

```
function confu(tree,test_data,name)
%input:tree:structure,a trained tree
       %test_data:array,with last column labels
       %name:string,tree's name
%output:NaN
    predict=tree_classify(tree,test_data);
    true_ans=test_data(:,end);
    figure;
    cm=confusionchart(true_ans,predict);
    conf_name=sprintf('Confusion Matrix, Tree:%s',name);
    cm.Title=conf_name;
    cm.RowSummary='row-normalized';
    cm.ColumnSummary='column-normalized';
end
```

TV_ID3主程序：TV_ID3_main.m

```
clear all;
% Load the agaricuslepiota table
load('agaricuslepiota.mat');

% Reorder the columns
reordered_table = agaricuslepiota(:, [2:end, 1]);

for col = 1:size(reordered_table, 2)
    [unique_classes{col}, ~, class_indices] = unique(reordered_table(:, col));
    my_data(:, col) = class_indices;
end
% Delete rows with NaN values
my_data(any(isnan(my_data), 2), :) = [];

%assume we have imported a numerical data array named my_data,with last
%column labels
split_data=cvpartition(my_data(:,end),"KFold",10);
%cross validation
train_array=zeros(10,1);
test_array=zeros(10,1);
```

```
for test_index=1:10
    train_logic=training(split_data,test_index);
    test_logic=test(split_data,test_index);
    train_data=my_data(train_logic,:);
    test_data=my_data(test_logic,:);
    tree=ID3(train_data,1:size(train_data,2)-1,1e-8);
    training_results=tree_classify(tree,train_data);
    test_results=tree_classify(tree,test_data);
    train_key=my_data(train_logic,end);
    test_key=my_data(test_logic,end);
    train_score=sum(training_results==train_key)/length(train_key);
    test_score=sum(test_results==test_key)/length(test_key);
    train_array(test_index)=train_score;
    test_array(test_index)=test_score;
end
plot(train_array','--bo');
hold on
plot(test_array','-r+')
hold off
confu(tree,test_data,'TV ID3');
train_fin_term=mean(train_array,1)
test_fin_term=mean(test_array,1)
function tree=ID3(D,A,epsilon)


%input:D:array,dataset,with last column labels
      %A:array:indices of all features selected
      %epsilon: threshold in ID3
%output:tree:sturcture,a trained decision tree
    epsilon=epsilon/exp(0.5);
    %we use exponetially decaying threshold
    if length(unique(D(:,end)))==1
        tree.type='leaf';
        tree.label=D(1,end);
    elseif isempty(A)
        tree.type='leaf';
        tree.label=mode(D(:,end));
    else
        info_gains=zeros(length(A),1);
        j=0;
        for feature=A
            j=j+1;
            info_gain_j=info_gain(D,feature);
            info_gains(j)=info_gain_j;
        end
        [max_info_gain,max_id]=max(info_gains);
        if max_info_gain<epsilon
            tree.type='leaf';
            tree.label=mode(D(:,end));
        else
            max_feature_array=D(:,max_id);
            max_feature_classes=unique(max_feature_array);
            TV_gain_classes=zeros(length(max_feature_classes),1);
            u=0;
            for class=max_feature_classes'
                u=u+1;
                temp_D=D;
                temp=(D(:,max_id)==class);
```

```
                temp_D(:,max_id)=temp;
                TV_gain_class=TV_gain(temp_D,max_id);
                TV_gain_classes(u)=TV_gain_class;
            end
            num_of_expand=ceil(length(max_feature_classes)/2);
            %choose half of nodes to expand
            tree_cell={};
            [~,classes_id]=maxk(TV_gain_classes,num_of_expand);
            count=0;
            for id=classes_id'
                count=count+1;
                id_data=D(find(D(:,max_id)==id),:);
                if size(id_data,1)>0
                    A_bar=A(A~=max_id);
                    subtree=ID3(id_data,A_bar,epsilon);
                    subtree.criterion=[max_id,id];
                    tree_cell{count}=subtree;
                end
            end
            tree.children=tree_cell;
            tree.type='branch';
        end
    end
end
```

C4.5主程序：C45_main.m

```
clear all;
% Load the agaricuslepiota table
load('agaricuslepiota.mat');

% Reorder the columns
reordered_table = agaricuslepiota(:, [2:end, 1]);

for col = 1:size(reordered_table, 2)
    [unique_classes{col}, ~, class_indices] = unique(reordered_table(:, col));
    my_data(:, col) = class_indices;
end
% Delete rows with NaN values
my_data(any(isnan(my_data), 2), :) = [];

%assume we have imported a numerical data array named my_data,with last
%column labels
split_data=cvpartition(my_data(:,end),"KFold",10);
%cross validation
train_array=zeros(10,1);
test_array=zeros(10,1);
for test_index=1:10
    train_logic=training(split_data,test_index);
    test_logic=test(split_data,test_index);
    train_data=my_data(train_logic,:);
    test_data=my_data(test_logic,:);
    tree=ID3(train_data,1:size(train_data,2)-1,1e-8);
    training_results=tree_classify(tree,train_data);
    test_results=tree_classify(tree,test_data);
    train_key=my_data(train_logic,end);
    test_key=my_data(test_logic,end);
```

```
    train_score=sum(training_results==train_key)/length(train_key);
    test_score=sum(test_results==test_key)/length(test_key);
    train_array(test_index)=train_score;
    test_array(test_index)=test_score;
end
plot(train_array','--bo');
hold on
plot(test_array','-r+')
hold off
confu(tree,test_data,'C45');
train_fin_term=mean(train_array,1)
test_fin_term=mean(test_array,1)
function tree=ID3(D,A,epsilon)


%input:D:array,dataset,with last column labels
      %A:array:indices of all features selected
      %epsilon: threshold in ID3
%output:tree:sturcture,a trained decision tree
    epsilon=epsilon/exp(0.5);
    %we use exponetially decaying threshold
    if length(unique(D(:,end)))==1
        tree.type='leaf';
        tree.label=D(1,end);
    elseif isempty(A)
        tree.type='leaf';
        tree.label=mode(D(:,end));
    else
        info_gains=zeros(length(A),1);
        j=0;
        for feature=A
            j=j+1;
            info_gain_j=info_gain(D,feature);
            info_gains(j)=info_gain_j;
        end
        [max_info_gain,max_id]=max(info_gains);
        if max_info_gain<epsilon
            tree.type='leaf';
            tree.label=mode(D(:,end));
        else
            max_feature_array=D(:,max_id);
            max_feature_classes=unique(max_feature_array);
            info_gain_classes=zeros(length(max_feature_classes),1);
            u=0;
            for class=max_feature_classes'
                u=u+1;
                temp_D=D;
                temp=(D(:,max_id)==class);
                temp_D(:,max_id)=temp;
                info_gain_class=info_gain_ratio(temp_D,max_id);
                info_gain_classes(u)=info_gain_class;
            end
            num_of_expand=ceil(length(max_feature_classes)/2);
            %choose half of nodes to expand
            tree_cell={};
            [~,classes_id]=maxk(info_gain_classes,num_of_expand);
            count=0;
            for id=classes_id'
```

```
                count=count+1;
                id_data=D(find(D(:,max_id)==id),:);
                if size(id_data,1)>0
                    A_bar=A(A~=max_id);
                    subtree=ID3(id_data,A_bar,epsilon);
                    subtree.criterion=[max_id,id];
                    tree_cell{count}=subtree;
                end
            end
            tree.children=tree_cell;
            tree.type='branch';
        end
    end
end
```

CART模型函数：CART.m

```
function [trainedClassifier, validationAccuracy] = trainClassifier(trainingData)
inputTable = trainingData;
predictorNames = {'x', 's', 'n', 't', 'p1', 'f', 'c', 'n1', 'k', 'e', 'e1',
's1', 's2', 'w', 'w1', 'p2', 'w2', 'o', 'p3', 'k1', 's3', 'u'};
predictors = inputTable(:, predictorNames);
response = inputTable.p;
isCategoricalPredictor = [true, true, true, true, true, true, true, true, true,
true, true, true, true, true, true, true, true, true, true, true, true, true];
classNames = categorical({'e'; 'p'});

% 训练分类器
classificationTree = fitctree(...
    predictors, ...
    response, ...
    'SplitCriterion', 'gdi', ...
    'MaxNumSplits', 4, ...
    'Surrogate', 'off', ...
    'ClassNames', classNames);

% 使用预测函数创建结果结构体
predictorExtractionFcn = @(t) t(:, predictorNames);
treePredictFcn = @(x) predict(classificationTree, x);
trainedClassifier.predictFcn = @(x) treePredictFcn(predictorExtractionFcn(x));

% 向结果结构体中添加字段
trainedClassifier.RequiredVariables = {'c', 'e', 'e1', 'f', 'k', 'k1', 'n',
'n1', 'o', 'p1', 'p2', 'p3', 's', 's1', 's2', 's3', 't', 'u', 'w', 'w1', 'w2',
'x'};
trainedClassifier.ClassificationTree = classificationTree;
trainedClassifier.About = '此结构体是从分类学习器 R2023a 导出的训练模型。';
trainedClassifier.HowToPredict = sprintf('要对新表 T 进行预测，请使用: \n
[yfit,scores] = c.predictFcn(T) \n将 ''c'' 替换为作为此结构体的变量的名称，例如
''trainedModel''。\n \n表 T 必须包含由以下内容返回的变量: \n c.RequiredVariables \n变量
格式(例如矩阵/向量、数据类型)必须与原始训练数据匹配。\n忽略其他变量。\n \n有关详细信息，请参阅
<a href="matlab:helpview(fullfile(docroot, ''stats'', ''stats.map''),
''appclassification_exportmodeltoworkspace'')">How to predict using an exported
model</a>。');

% 提取预测变量和响应
inputTable = trainingData;
```

```matlab
predictorNames = {'x', 's', 'n', 't', 'p1', 'f', 'c', 'n1', 'k', 'e', 'e1',
's1', 's2', 'w', 'w1', 'p2', 'w2', 'o', 'p3', 'k1', 's3', 'u'};
predictors = inputTable(:, predictorNames);
response = inputTable.p;
isCategoricalPredictor = [true, true, true, true, true, true, true, true, true,
true, true, true, true, true, true, true, true, true, true, true, true, true];
classNames = categorical({'e'; 'p'});

% 设置留出法验证
cvp = cvpartition(response, 'Holdout', 0.44);
trainingPredictors = predictors(cvp.training, :);
trainingResponse = response(cvp.training, :);
trainingIsCategoricalPredictor = isCategoricalPredictor;

% 训练分类器
classificationTree = fitctree(...
    trainingPredictors, ...
    trainingResponse, ...
    'SplitCriterion', 'gdi', ...
    'MaxNumSplits', 4, ...
    'Surrogate', 'off', ...
    'ClassNames', classNames);

% 使用预测函数创建结果结构体
treePredictFcn = @(x) predict(classificationTree, x);
validationPredictFcn = @(x) treePredictFcn(x);

% 计算验证预测
validationPredictors = predictors(cvp.test, :);
validationResponse = response(cvp.test, :);
[validationPredictions, validationScores] =
validationPredictFcn(validationPredictors);

% 计算验证准确度
correctPredictions = (validationPredictions == validationResponse);
isMissing = ismissing(validationResponse);
correctPredictions = correctPredictions(~isMissing);
validationAccuracy = sum(correctPredictions)/length(correctPredictions);
```

参考文献

1. Total Variation. https://zhuanlan.zhihu.com/p/352946799 ↩

2. Fannes–Audenaert inequality - Wikipedia ↩

3. Mushroom. (1987). UCI Machine Learning Repository. https://doi.org/10.24432/C5959T. ↩