

石邢越 16337208

饶洋辉 教授

人工智能实验

2018 年 9 月 20 日

实验 2: 决策树

1 算法原理

1.1 整体思路

用决策树解决有监督的分类问题主要分为建立决策树模型和根据得到的决策时获得预测值两步，这次实验主要工作量在第一步。

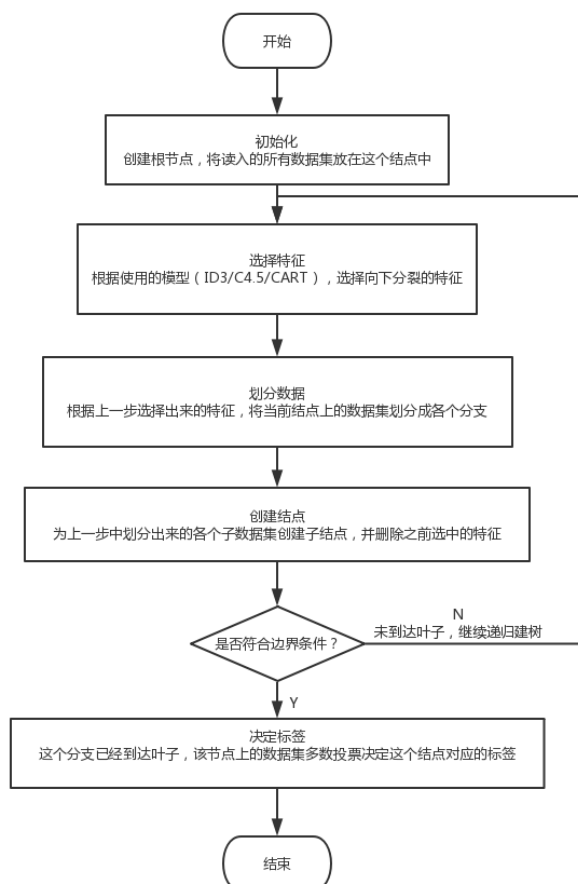
建立决策树的模型使用递归思想。对于决策树中的每一个节点，根据不同的决策树模型（ID3, C4.5 和 CART 决策树模型）以不同的指标（信息增益，信息增益比和 gini 指数）选择树向下分裂的属性，并将此分裂的过程递归进行到符合边界条件时，这时产生了叶子节点。对于每个叶子节点，用符合这条分支的所有数据集样本的 label 采用多数投票得分的方法得到其对应的预测 label。

决策树建立完成后，读取测试集中各样本各特征上的取值，遍历决策树找到对应的叶子节点（同样使用递归的算法），叶子节点的 label 为这个测试集样本的预测 label。

理论上，在建树的过程中，生成的叶子节点可能不能包括验证集/测试集中所有样本，所以我定义了变量 `test_cnt_of_arriving_leaf` 和 `test_cnt_of_not_arriving_leaf`，并检测了实际运行过程中是否会出现这种情况。从实验结果来看，确实会出现这样的情况，这时待预测的样本走不到叶子结点（分裂属性中没有它对应的值的分支），那么在这个结点（非叶子结点）用多数投票决定预测的 label。

1.2 建树步骤

递归建树的步骤用以下流程图表示：



边界条件为以下 3 种之一：

- 1) 当前结点数据集所有样本标签相同；
- 2) 当前结点数据集可选的特征集为空或者所有样本在所有可选特征上取值相同；
- 3) 当前结点数据集为空。

1.3 三种决策树模型——ID3, C4.5 和 CART

三种决策树模型的不同在于选择向下分裂的属性时采用的指标不同。ID3 模型考虑信息增益，最大者作为向下分裂的属性（同一属性在一条分支中不重复作为分裂属性，下两种模型同）；C4.5 考虑信息增益比，最大者作为向下分裂的属性；CART 模型考虑 gini 指数，最小者作为向下分裂的属性。

我的程序中首先从用户处接收 1, 2, 3 中的一个数字选择使用哪种模型建立决策树。另外我从测试集中分出一部分（结果中以 1/5 为例）作为验证集，以此比较 3 种模型的决策树预测的准确率。

2 伪代码

```
1 //generate the root of decision tree
2 //all train data put in root
3 initial();
4 //read train data set
```

```

5         //seperate a part of train set as validation set
6         ReadTrain();
7
8         int select_model;
9         get select_model to choose from ID3/C4.5/CART
10        //after initializing, build tree recursively
11        //build tree with the train set
12        recursive(root);
13
14
15        for each sample in validation set
16            traverse built tree to find its predicted label
17        analyze accuracy of this tree with validation set
18
19        //post-prune
20        prune(root);
21
22
23        //read test set
24        //get predicted label by traversing tree
25        //output predicted labels to a file
26        Resttest();
        output_test_result();

```

3 关键代码

3.1 初始化

```

1    //initialize root node
2    void initial(){
3        //read train set, some of train set is used as validation
4        set
5        ReadTrain();
6        root->datasize = traincnt;
7        root->attrsiz = Length;
8        //not prune at first
9        root->prune_or_not = 0;
10       //place all train data into root node
11       for (int i = 0; i<traincnt; i++){
12           for (int j = 0; j < Length; j++){
13               root->Data[i][j] = train[i][j];
14           }
15           root->Label[i] = label[i];
16       }
17       //all attributes are available to choose for root node
18       for (int i = 0; i < Length; i++) {
19           root->Attr[i] = i;
20       }
21       cout << "finish initializing!" << endl; //test
22       return;
23   }

```

3.2 选择向下分裂的属性

在递归建树的函数 `recursive ()` 中使用参数 `select_model` 选择决策树模型。

`Recursive` 函数的子函数 `Choose_attr` 函数根据参数 `select_model` 调用 3 中模型中的一种选择分裂属性。

```

1  //choose attribute to split on
2  int choose_attr(Node *p, int select_model)
3  {
4      int attr;//attributes chosen
5      if (select_model==0) attr=ID3(p);
6      if (select_model == 1) attr=C4_5(p);
7      if (select_model == 2) attr = CART(p);
8      return attr;
9  }
```

3 种模型选择分裂属性的函数分别为 `ID3`, `C4_5` 和 `CART`, 如下:

```

1  int ID3(Node *p){
2      //previous entropy
3      double HD = Empirical_entropy(p);
4      //entropy after introducing this attribute
5      double HDA[7];
6      double gain[7], maxGain = -100;
7      for (int i = 0; i < p->attrsize; i++) {
8          HDA[i] = Conditional_entropy(i, p);
9      }
10     int max_index = 0;
11     for (int i = 0; i < p->attrsize; i++){
12         gain[i] = HD - HDA[i];
13         if (gain[i]>maxGain) {
14             maxGain = gain[i];           max_index = i;
15         }
16     }
17     return p->Attr[max_index];
18 }
```

```

1  int C4_5(Node* p){
2      double HD = Empirical_entropy(p);
3      double gainRatio[7];
4      double maxGainRatio = 0;
5      for (int i = 0; i<p->attrsize; i++) {
6          gainRatio[i] = (HD - Conditional_entropy(i, p)) /
7      SplitInfo(i, p);
8      }
9      int max_index = 0;
10     for (int i = 0; i<p->attrsize; i++){
11         if (gainRatio[i]>maxGainRatio){
12             maxGainRatio = gainRatio[i];
13             max_index = i;
14         }
15     }
16     return p->Attr[max_index];
17 }

```

```

1  int CART(Node* p)
2  {
3      double Gini[7];
4      for (int i = 0; i < p->attrsize; i++)
5      {
6          Gini[i] = gini(i, p);
7      }
8      double minnum = 1000;
9      int min_index = 0;
10     for (int i = 0; i<p->attrsize; i++){
11         if (Gini[i]<minnum){
12             minnum = Gini[i];
13             min_index = i;
14         }
15     }
16     return p->Attr[min_index];
17 }

```

三种模型中用于计算熵和 gini 指数的子函数不在此赘述，详细实现见头文件

models.h。

3.3 判断边界

```

1  bool meet_with_bound(Node* p){
2      //boundary 1: all labels in train set are identical
3      bool bound_1 = true;
4      int firstone = p->Label[0];
5      for (int i = 0; i<p->datasize; i++) {
6          if (p->Label[i] != p->Label[0]) {
7              bound_1 = false;
8              break;
9          }
10     }
11     //boundary 2: attrsize=0

```

```

12      //or all samples have identical value on all attributes
13      //former situation can be regared as
14      //a special instance of the later one
15      bool bound_2 = true;
16      for (int j = 0; j<p->attrsiz; j++) {
17          int maxnum = 0, minnum = 1000;
18          for (int i = 0; i<p->datasize; i++) {
19              if (p->Data[i][j]<minnum)
20                  minnum = p->Data[i][j];
21              if (p->Data[i][j]>maxnum)
22                  maxnum = p->Data[i][j];
23          }
24          if (maxnum != minnum) {
25              bound_2 = false;
26              break;
27          }
28      }
29      //boundary 3: datasize=0
30      bool bound_3 = true;
31      if (p->datasize != 0) {
32          bound_3 = false;
33      }
34      //meet with bound when at least one kind of boundary is meet
35      return (bound_1 || bound_1 || bound_3);
36  }

```

3.4 递归建树

```

1  void recursive(Node *p, int select_model){
2      //meet with boundary, generate a leaf
3      if (meet_with_bound(p)){
4          cnt_of_leave++;
5          decide_final_label(p);
6          return;
7      }
8      //not leaf, continue splitting
9      //choose 1 model from 3
10     int attr_chosen = choose_attr(p,select_model);
11     p->attr_chosen = attr_chosen;
12
13     //test ouput: see the process of buiding decison tree
14     //cout << "split attribute chosen: " << attr_chosen << endl;//test
15
16
17     int index_chosen;
18     for (int i = 0; i<p->attrsize; i++){if (p->Attr[i] == attr_chosen){
19         index_chosen = i; break;
20     }
21     int maxnum = 0, minnum = 1000;
22     for (int i = 0; i<p->datasize; i++){
23         if (p->Data[i][index_chosen]<minnum)
24             minnum = p->Data[i][index_chosen];
25         if (p->Data[i][index_chosen]>maxnum)
26             maxnum = p->Data[i][index_chosen];
27     }
28     for (int i = minnum; i <= maxnum; i++){
29         Node *tem = new Node;
30         //a son to split from current node
31         tem->datasize = 0;
32         tem->attrsize = 0;
33         tem->attr_num = i;
34         //not prune at first
35         tem->prune_or_not = 0;
36         //1 less attribute than father node available to choose
37         tem->attrsize = p->attrsize - 1;
38         int sample_counter = 0;
39         for (int j = 0; j<p->datasize; j++){
40             if (p->Data[j][index_chosen] == i){
41                 tem->Label[sample_counter] = p->Label[j];
42                 int attr_counter = 0;
43                 for (int k = 0; k<p->attrsize; k++){
44                     if (k != index_chosen){
45                         tem->Data[sample_counter]
46 [attr_counter] = p->Data[j][k];
47                         attr_counter++;
48                     }
49                 }
50                 sample_counter++;
51             }
52         }
53         tem->datasize = sample_counter;
54         if (sample_counter != 0)
55             p->children.push_back(tem);
56         else
57             free(tem);
58     }
59     for (int i = 0; i<p->children.size(); i++)
60         recursive(p->children[i],select_model);
61 }

```


3.5 后剪枝

基于验证集准确率的后剪枝。

若分支后验证集准确率可以提高，则分支，否则认为分支没有必要，剪掉。

```

1 void prune(Node* p){
2     //return when meet with boundary
3     if (p->children.size() == 0)
4         return;
5     //post traverse
6     for (int i = 0; i<p->children.size(); i++){
7         //prune recursively
8         prune(p->children[i]);
9         p->prune_or_not = 1;
10        decide_final_label(p);
11        for (int i = 0; i<valicnt; i++) traverse_tree(i, root,
12 "vali");
13        double right = 0;
14        for (int i = 0; i<valicnt; i++){
15            if (vali_label_result[i] == valilabel[i]) right++;
16        }
17        double temac = right / valicnt;
18        //if this branch is unnecessary (accuracy doesn't increase),
19        prune branch
20        if (temac >= accuracy){
21            cnt_of_pruned_leaves++;
22            p->children.clear();
23            accuracy = temac;
24        }
25        else
26            p->prune_or_not = 0; //choose not to prune
    }
}

```

4 创新点与优化

实现后剪枝

我目前实现的剪枝是基于验证集准确率的后剪枝。在使用未剪枝的决策树验证

完验证集后，得到未剪枝时的准确率。之后递归剪枝，对已有决策树的每个结

点，尝试将它剪枝成叶子结点，若这种尝试可以提高或保持验证集准确率，说明没有必要在这个结点上继续分支，则剪枝将此结点设为叶子结点。

这种剪枝的判断依据仅为验证集上的准确率，比较粗糙。另外可能的判断标准还有权衡准确率提高和分支的代价，以决定是否要分支，这时即使准确率可以提高，如果分支代价过高也可能选择不分枝（此次实验我还没有深入了解并实现这种剪枝。）

剪枝的函数 `prune` 见上方关键代码部分。

5 实验结果

5.1 测试集预测结果

运行完成的界面如下图，预测结果见 `output.csv` 文件。这是用 4/5 做训练集，再用 1/5 做验证集剪枝得到的结果。

```
testcnt=100
test_cnt_of_arriving_leaf=100
test_cnt_of_not_arriving_leaf=0
testacceptable=24
testunacceptable=76
请按任意键继续. . .
```

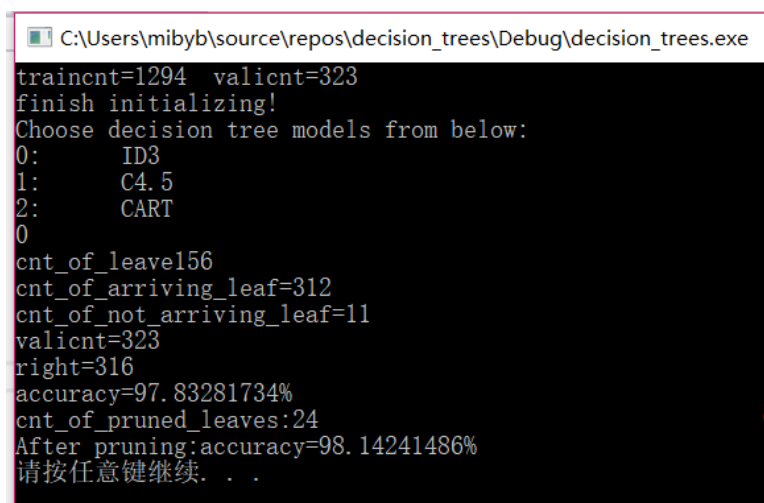
5.2 3 种模型建树对比

得到验证集的方法

读取 train set 时, 序号为 5 的倍数的样本作为验证集, 其他作为训练集。

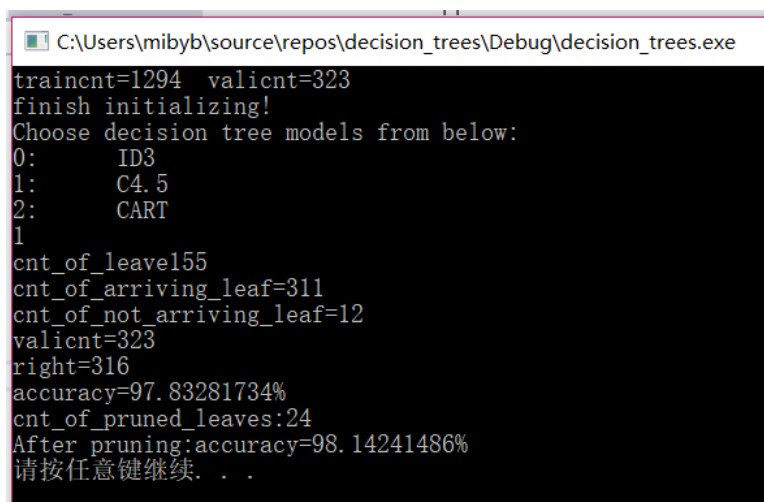
运行结果对比

图 1, 2, 3 分别是使用 ID3, C4.5 和 CART 模型建立决策树的运行结果, 三种模型的对比见表 1。三种模型使用的训练集、验证集相同, 大小分别为 1294, 323。



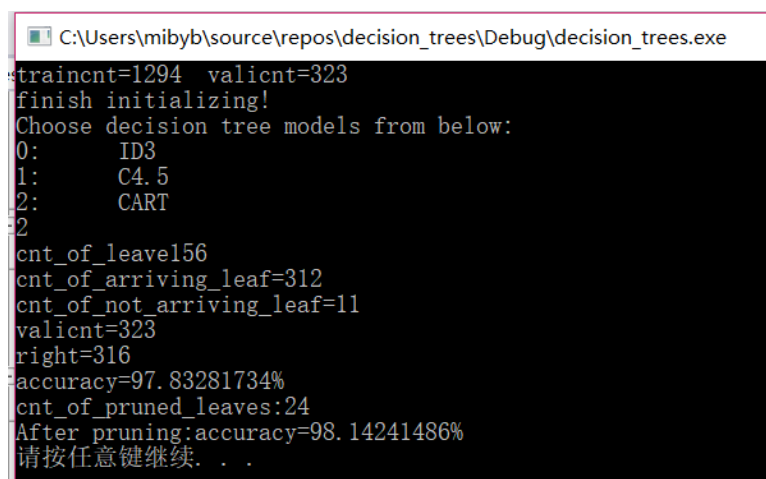
```
C:\Users\mibyby\source\repos\decision_trees\Debug\decision_trees.exe
traincnt=1294 valicnt=323
finish initializing!
Choose decision tree models from below:
0: ID3
1: C4.5
2: CART
0
cnt_of_leavel56
cnt_of_arriving_leaf=312
cnt_of_not_arriving_leaf=11
valicnt=323
right=316
accuracy=97.83281734%
cnt_of_pruned_leaves:24
After pruning:accuracy=98.14241486%
请按任意键继续. . .
```

图 1. ID3 模型



```
C:\Users\mibyby\source\repos\decision_trees\Debug\decision_trees.exe
traincnt=1294 valicnt=323
finish initializing!
Choose decision tree models from below:
0: ID3
1: C4.5
2: CART
1
cnt_of_leavel55
cnt_of_arriving_leaf=311
cnt_of_not_arriving_leaf=12
valicnt=323
right=316
accuracy=97.83281734%
cnt_of_pruned_leaves:24
After pruning:accuracy=98.14241486%
请按任意键继续. . .
```

图 2. C4.5 模型



```

C:\Users\mibyb\source\repos\decision_trees\Debug\decision_trees.exe
traincnt=1294 valcnt=323
finish initializing!
Choose decision tree models from below:
0: ID3
1: C4.5
2: CART
2
cnt_of_leavel56
cnt_of_arriving_leaf=312
cnt_of_not_arriving_leaf=11
valcnt=323
right=316
accuracy=97.83281734%
cnt_of_pruned_leaves:24
After pruning:accuracy=98.14241486%
请按任意键继续. . .

```

图 3. CART 模型

	可达叶子 样本数	剪枝前叶 子数	剪枝前准确率	剪枝数	剪枝后准确率
ID3	312	156	97.83281734%	24	98.14241486%
C4.5	311	155	97.83281734%	24	98.14241486%
CART	312	156	97.83281734%	24	98.14241486%

表 1. 三种决策树模型运行结果的对比

观察建树过程

由表 1 发现三种模型得到剪枝前、剪枝后的准确率都完全一样，为了验证三种模型在选择分裂属性时的不同，我用大小 55 的训练集，大小 13 的验证集观察了决策树的建立过程。有图可以看到三种模型建树时依次选择的分裂属性不同。

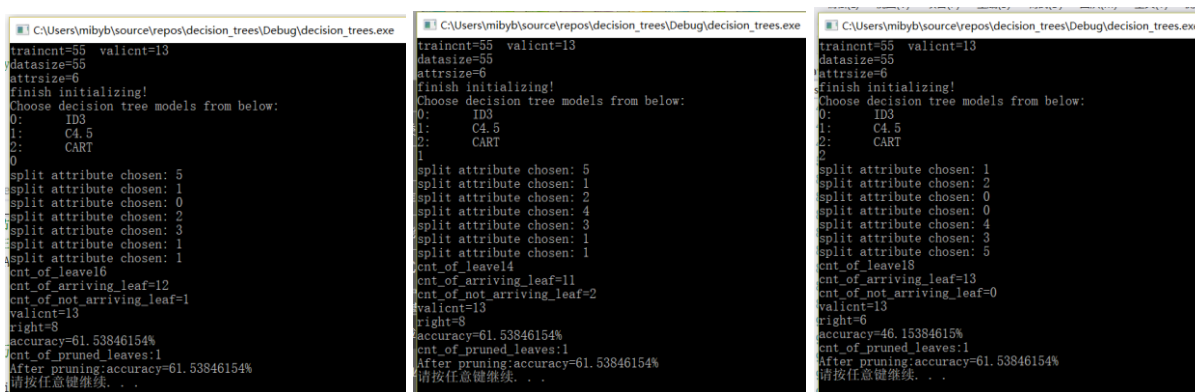


图 4~6. ID3 选择的分裂属性依次为 $5 \rightarrow 1 \rightarrow 0 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 1$

C4.5 选择的分裂属性依次为 $5 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 3 \rightarrow 1 \rightarrow 1$

CART 选择的分裂属性依次为 $1 \rightarrow 2 \rightarrow 0 \rightarrow 0 \rightarrow 4 \rightarrow 3 \rightarrow 5$

6 评测指标

评测指标：验证集上的准确率

使用不同大小的训练集（即划去做验证集的部分大小不同），得到的验证集上准确率如下图。Output.csv 是用 4/5 做训练集，1/5 做验证集剪枝得到的。

