《人工智能实验》

实验报告

Lab 2

决策树

学 院 名 称：	数据科学与计算机学院
专业（班级）：	17 级计算机科学与技术
学 生 姓 名：	薛伟豪
学　　　 号：	17341178
联 系 方 式：	15013041671

Lab 2：决策树

1. 算法原理

- **决策树简述**

决策树是一种有监督的分类预测模型。直观上来看，决策树是一种基于贪心策略的树形结构。树中包含了内部节点和叶子节点：内部节点表示某一种属性，每次都选择最优的属性进行分裂，该内部节点的每个分支代表了在该属性上所有可能取值的输出；而叶子节点则表示预测的标签结果。

在已经建好的决策树上进行分类，输入为样本的特征向量（即在各个属性上的取值），输出为所预测的标签。从根节点开始，经过内部节点时，根据分裂的属性和该特征向量在该属性上的取值进入对应的分支，直到该特征向量到达叶子节点。该叶子节点的标签取值即为该文本特征向量的预测标签结果。

- **分裂属性的选取规则**

从简述中可以看出，在每个节点上如何选择分裂属性是很重要的。我们希望决策树的内部节点在分裂之后形成的各个分支，能够尽可能属于同一个标签，如果分裂属性选取得不好，那么预测的准确率会大大降低。因此决策树构建中十分关键的一点，便是需要在决策树每个内部节点上选择最优的分裂属性进行分裂。

那么哪一种属性才能算是最优呢？在这里，我们需要使用相关指标来对分裂属性的选取进行评判。常用的指标有：信息增益、信息增益率、Gini系数。

- **基于信息增益的算法：ID3模型**

ID3模型以信息增益作为衡量指标。

在信息论中，熵是表示随机变量不确定性的度量。变量的不确定性越大，对应的信息熵也就越大。以频率作为概率，这里令$p(d)=\frac{|d|}{|D|}$为类别d在数据集D中出现的概率，信息熵的计算公式如下：

$$H(D)=-\sum_{d\in D}p(d)\cdot log\,p(d)$$

条件熵衡量的是在某个随机变量已知的条件下，另一个随机变量的不确定性。在这里，我们需要计算某个属性划分已知的情况下，数据集D的条件熵。假设已知划分属性为A，那么A对数据集D的条件熵的计算公式如下：

$$H(D|A)=-\sum_{a\in A}p(a)\cdot H(D|A=a)$$

其中

$$H(D|A=a) = -\sum_{d \in D} p(d|A=a) \cdot log\, p(d|A=a)$$

信息增益也称为互信息，指的是在某个随机变量已知的情况下，对另一个随机变量的不确定性的减少程度。在这里，我们需要计算在属性A对数据集D进行划分之后，数据集D分类不确定性的减少程度。计算属性A对数据集D的信息增益的公式如下：

$$g(D,A) = H(D) - H(D|A)$$

根据信息增益的定义，我们容易看出，在构建决策树时，我们需要选择信息增益最大的属性作为分裂属性。

- **基于信息增益率的算法：C4.5模型**

C4.5模型对ID3模型进行了改进，以信息增益率作为衡量指标。

我们不难发现，在使用信息增益作为评判指标的时候是有一定缺陷的。在分裂属性的选择上，ID3模型会更倾向于选择那些属性取值较多的属性，而这往往会导致过拟合的情况。C4.5模型克服了这个问题，对各个属性计算得到的信息增益进行了正则化。假设划分属性为A，计算属性A对数据集D的信息增益率的公式如下：

$$gRatio(D,A) = \frac{g(D,A)}{SplitInfo(D,A)}$$

其中$g(D,A)$为属性A对数据集D的信息增益，$SplitInfo(D,A)$为数据集D关于属性A的值的熵。具体地，假设属性A的可能取值数为N，那么有：

$$SplitInfo(D,A) = -\sum_{i=1}^{N} \frac{|D_i|}{|D|} \cdot log \frac{|D_i|}{|D|}$$

在构建决策树时，我们需要选择信息增益率最大的属性作为分裂属性。

- **基于Gini系数的算法：CART模型**

我们很容易可以看到，不管是ID3模型还是C4.5模型，都是采用基于熵的衡量指标。与这不同的是，CART模型以Gini系数作为衡量指标。

假设数据集D包含来自n个类的样本，p_i是类i在D中的相对频率，Gini系数的定义如下：

$$Gini(D) = \sum_{i=1}^{n} p_i(1-p_i) = 1 - \sum_{i=1}^{n} {p_i}^2$$

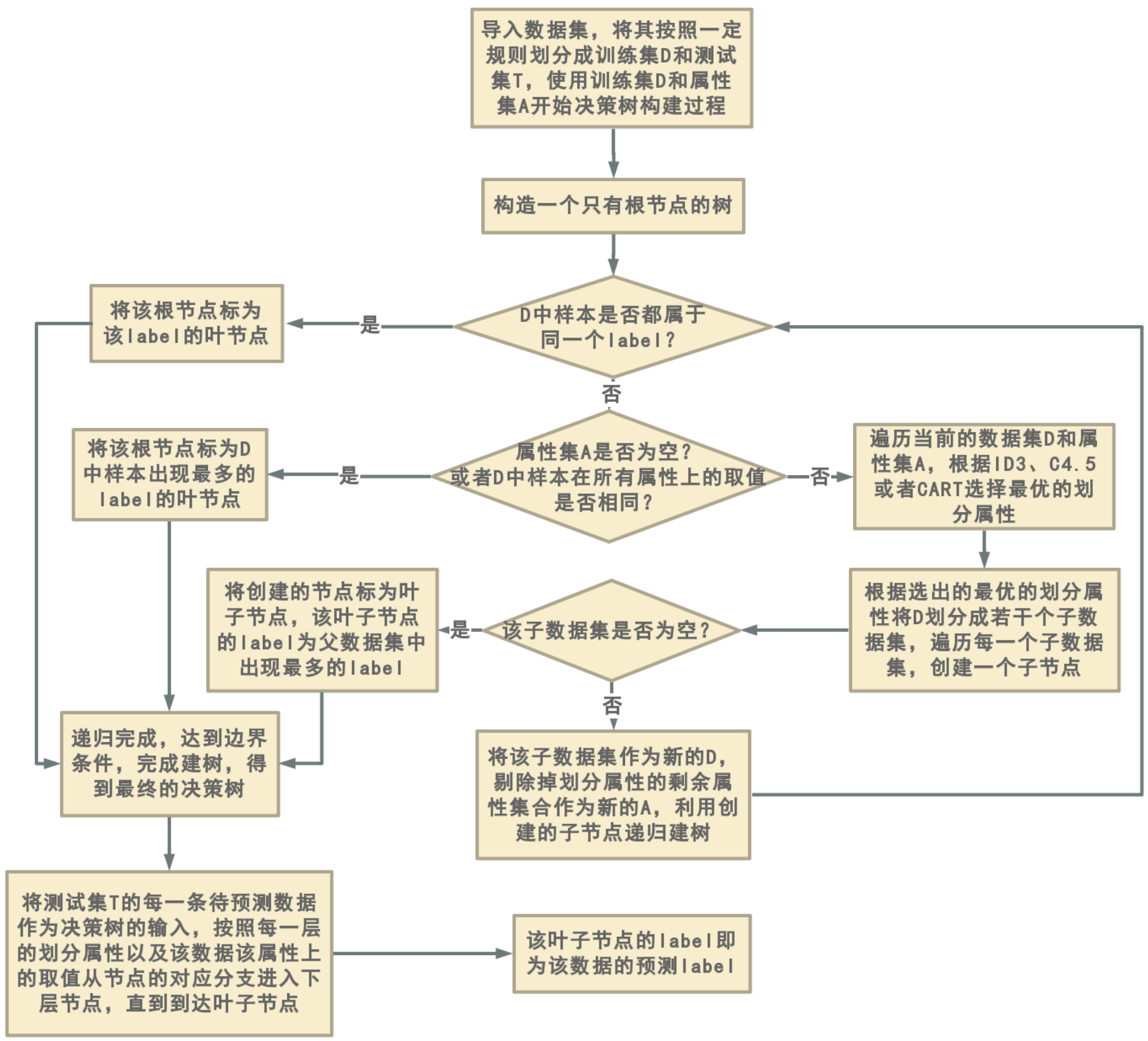
而在决策树分裂属性的选取上，我们需要计算当我们选择属性A作为分裂属性时，数据集D的Gini系数。假定属性A的取值数为v，计算公式如下：

$$Gini(D,A)=\sum_{i=1}^{v}p(A_i)\cdot Gini(D_i|A=A_i)$$

我们不难看出，当类别越少，类别集中度越高的时候，基尼系数越低；当类别越多，类别集中度越低的时候，基尼系数越高。换句话来说，基尼系数越小，则样本的不确定性越低。因此，在分裂属性的选择上，我们使用Gini系数最小的属性来对数据集D进行划分。

需要注意的是，CART算法构建的决策树最为常见的是二叉树。而在本次实验中，我采取了多叉树模型而非二叉树模型。

2. 流程图



3. 核心代码展示

- **属性字典（从序号映射到该属性所有取值的列表）**

```
attribution_dict={ 0 : ['vhigh', 'high', 'med', 'low'],      #buying
                  1 : ['vhigh', 'high', 'med', 'low'],      #maint
                  2 : ['2', '3', '4', '5more'],             #dorrs
                  3 : ['2', '4', 'more'],                   #persons
                  4 : ['big', 'med', 'small'],              #lug_boot
                  5 : ['high', 'med', 'low']                #safety
                }
```

- **计算数据集的信息熵**

```
def get_empirical_entropy(data_set, label_col):
    #获取数据集 data_set 的信息熵
    label_dict = {}
    data_num = len(data_set)
    for row in data_set:
        if row[label_col] not in label_dict:
            label_dict[row[label_col]] = 1
        else:
            label_dict[row[label_col]] += 1
    empirical_entropy = 0
    for value in label_dict.values():
        empirical_entropy -= value/data_num*np.log(value/data_num)
    return empirical_entropy
```

- **计算属性A对数据集的信息熵**

```
def get_conditional_entropy(data_set, label_col, A):
    #计算属性 A 对数据集 data_set 的条件熵
    A_kind_dict={}
    for row in data_set:
        if row[A] not in A_kind_dict:
            A_kind_dict[row[A]] = [row]
        else:
            A_kind_dict[row[A]].append(row)
    conditional_entropy = 0
```

```
    for key in A_kind_dict.keys():
        A_kind_dict[key] = np.array(A_kind_dict[key])
        empirical_entroy = get_empirical_entropy(A_kind_dict[key], label_col)
        conditional_entropy += len(A_kind_dict[key])/len(data_set)*empirical_entroy
    return conditional_entropy
```

- **计算信息增益**

```
def get_information_gain(data_set, label_col, A):
    #信息增益=经验熵-条件熵
    return get_empirical_entropy(data_set, label_col) - get_conditional_entropy(data_set, 
label_col, A)
```

- **计算信息增益率**

```
def get_infor_gain_ratio(data_set, label_col, A):
    #计算信息增益率
    return get_information_gain(data_set, label_col, A) / get_empirical_entropy(data_set, 
label_col)
```

- **计算基尼系数**

```
def get_gini_index(data_set, label_col, A):
    #计算 gini 系数(二分类)
    #函数返回计算所得的 gini 系数，和该属性中能使 gini 系数最小的二分类属性取值
    A_kind_dict = {}
    A_kind_dict_one = {}
    for row in data_set:
        if row[A] not in A_kind_dict.keys():
            A_kind_dict[row[A]] = 1
            if row[label_col] == '1':
                A_kind_dict_one[row[A]] = 1
            else:
                A_kind_dict_one[row[A]] = 0
        else:
            A_kind_dict[row[A]] += 1
            if row[label_col] == '1':
                A_kind_dict_one[row[A]] += 1
    gini_index=0
```

```
    for kind in A_kind_dict.keys():
        one_pro=A_kind_dict_one[kind]/A_kind_dict[kind]
        gini_index+=A_kind_dict[kind]/len(data_set)*(1-one_pro**2-(1-one_pro)**2)
    return gini_index
```

- **获取最优划分属性**

```
def Select_Attribution(data_set, mood, attribution_list):
     #按照给定的mood对划分属性进行选择
     #attribution_list 为当前可以进行划分的属性列表
    #找到最优的划分属性并返回
     label_col = len(data_set[0]) - 1
    #使用ID3模型
    if mood == 'ID3':
        max_infor_gain = float('-inf')
        max_attribution = None
        for attribution in attribution_list:
            if get_infor_gain(data_set, label_col, attribution) > max_infor_gain:
                max_infor_gain = get_infor_gain(data_set, label_col, attribution)
                max_attribution = attribution
        return max_attribution
    #使用C4.5模型
    elif mood == 'C4.5':
        max_ratio = float('-inf')
        max_attribution = None
        for attribution in attribution_list:
            if get_infor_gain_ratio(data_set, label_col, attribution) > max_ratio:
                max_ratio = get_infor_gain_ratio(data_set, label_col, attribution)
                max_attribution = attribution
        return max_attribution
    #使用CART模型（二分类）
    elif mood == 'CART':
        min_gini_index = float('inf')
        min_attribution = None
        for attribution in attribution_list:
            if get_gini_index(data_set, label_col, attribution) < min_gini_index:
                min_gini_index = get_gini_index(data_set, label_col, attribution)
                min_attribution = attribution
        return min_attribution
```

- **节点Node类定义**

```
class Node:
    def __init__(self, split_attribution = None, attribution_kind = None, label = None):
        #split_attribution 用于标记该 Node 的分裂属性
        #attribution_kind 用于标记父节点分裂后该结点对应分裂属性的取值
        #label 用于标记 Node 的预测分类，非叶子节点则为 None
        #child_nodes 用于存放 Node 的子节点
        #is_leaf 用于标记该结点是否为叶子节点
        self.split_attribution = split_attribution
        self.attribution_kind = attribution_kind
        self.label = label
        self.child_nodes = []
        self.is_leaf = False
```

- **决策树构建函数（DecisionTree类成员函数）**

```
    def build_tree(self, data_set, root, attribution_list):
        #如果 data_set 都属于一个标签，则将其标为该标签的叶节点
        label_col = len(data_set[0]) - 1
        if self.is_same_in_label(data_set):
            root.label = data_set[0][label_col]
            root.is_leaf = True
            return

        #如果属性集为空或者在所有属性上取值相同，将当前节点标记为叶节点，类别为 D 出现最多的类
        if len(attribution_list) == 0 or self.is_same_in_label(data_set):
            root.label = self.get_max_label(data_set)
            root.is_leaf = True
            return

        attribution = Select_Attribution(data_set, self.mood, attribution_list)
        root.split_attribution = attribution
        for kind in attribution_dict[attribution]:
            sub_data_set = []
            for row in data_set:
                if row[attribution] == kind:
                    sub_data_set.append(row)
            sub_data_set = np.array(sub_data_set)
            #如果数据集为空，则将当前节点标记为叶节点，类别为父类中出现最多的类
            if len(sub_data_set) == 0:
                sub_tree = Node(-1, kind, self.get_max_label(data_set))
```

```
                sub_tree.is_leaf = True
                root.child_nodes.append(sub_tree)
            else:
                new_attribution_list = []
                for item in attribution_list:
                    if item != attribution:
                        new_attribution_list.append(item)
                sub_tree = Node(-1, kind, self.get_max_label(sub_data_set))
                self.build_tree(sub_data_set, sub_tree, new_attribution_list)
                root.child_nodes.append(sub_tree)
```

- **利用构建好的决策树对测试集数据进行分类**

```
def classification(root, test_sample):
    if root.is_leaf:
        if int(root.label) == int(test_sample[6]):
            return True
        else:
            return False
    split_attribution = root.split_attribution
    for child in root.child_nodes:
        attribution_kind = child.attribution_kind
        if test_sample[split_attribution] == attribution_kind:
            return classification(child, test_sample)
```

- **五折交叉验证计算平均预测准确率（以ID3模型为例）**

```
def Five_Cross_Validation(data_set):
    #将数据集打乱
    row_rand_array = np.arange(data_set.shape[0])
    np.random.shuffle(row_rand_array)
    set_size = int(0.2*len(data_set))
    set_list = []
    set_list.append(data_set[row_rand_array[0:set_size]])
    set_list.append(data_set[row_rand_array[set_size:2*set_size]])
    set_list.append(data_set[row_rand_array[2*set_size:3*set_size]])
    set_list.append(data_set[row_rand_array[3*set_size:4*set_size]])
    set_list.append(data_set[row_rand_array[4*set_size:]])
    avg_accuracy_ID3 = 0
    #五折交叉验证计算平均准确率
    for index in range(len(set_list)):
```

```
        test_set = set_list[index]
        train_set_list = [set_list[i] for i in range(len(set_list)) if i != index]
        train_set = train_set_list[0]
        for i in range(1,len(train_set_list)):
            train_set = np.vstack((train_set,train_set_list[i]))
        ID3_tree=DecisionTree('ID3')
        ID3_root = Node()
        ID3_tree.build_tree(train_set, ID3_root, [0,1,2,3,4,5])
        accuracy_ID3 = cal_accuracy(ID3_root, test_set)
        avg_accuracy_ID3 += accuracy_ID3
    avg_accuracy_ID3 /= 5
    print('ID3:', avg_accuracy_ID3)
    return
```

4. 实验结果及分析

- **结果&模型性能展示与分析**

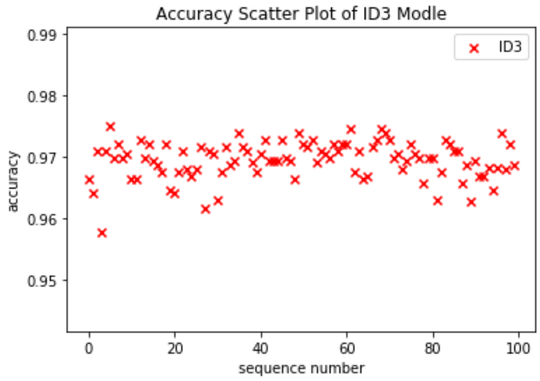
为了说明本次实验中所构建的三个模型的可靠性，我们将全部的数据集作为训练集和测试集，预测准确率如下所示：

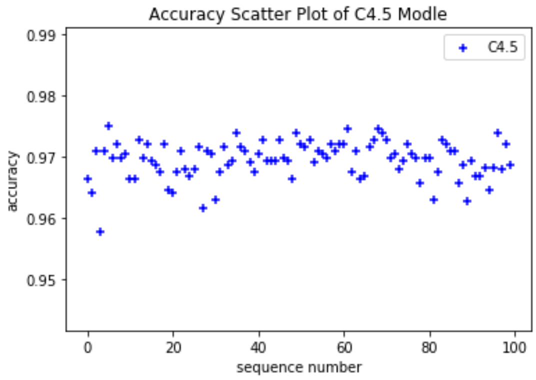
模型	代码&输出结果	预测准确率
ID3	In [156]: train_set = data_set[:] ...: test_set = data_set[:] ...: tree=DecisionTree('ID3') ...: root=Node() ...: tree.build_tree(train_set, root, [0,1,2,3,4,5]) ...: print('ID3:', cal_accuracy(root, test_set)) ID3: 1.0	100%
C4.5	In [155]: train_set = data_set[:] ...: test_set = data_set[:] ...: tree=DecisionTree('C4.5') ...: root=Node() ...: tree.build_tree(train_set, root, [0,1,2,3,4,5]) ...: print('C4.5:', cal_accuracy(root, test_set)) C4.5: 1.0	100%
CART	In [154]: train_set = data_set[:] ...: test_set = data_set[:] ...: tree=DecisionTree('CART') ...: root=Node() ...: tree.build_tree(train_set, root, [0,1,2,3,4,5]) ...: print('CART:', cal_accuracy(root, test_set)) CART: 1.0	100%

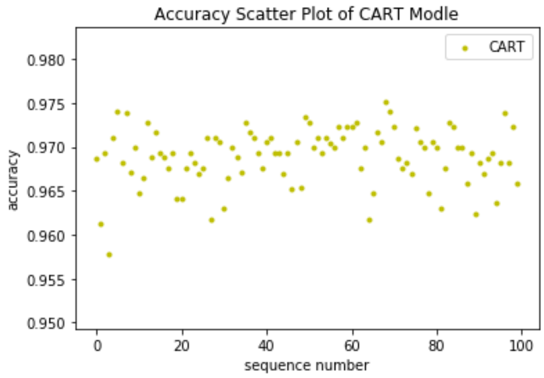
可以看到，当我们使用全部的数据集作为训练集和测试集时，三个模型计算得到的预测准确率都为100%，这可以很好地体现决策树构建的正确性。

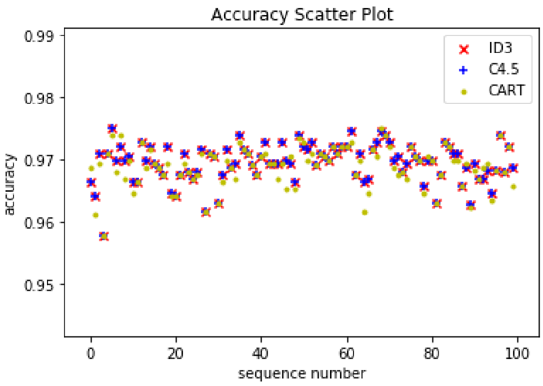
接着我们使用五折交叉验证（代码见代码展示部分），将数据集随机分成五份，从中选取四份作为训练集，一份作为训练集，进行决策树的构建与预测准确率计算，重复五次

使得每一份都被作为测试集，计算五次的平均准确率。在此基础之上，对ID3模型、C4.5模型以及CART模型使用五折交叉验证进行准确率计算100次，将结果绘制成散点图如下所示：









从四张散点图中，我们可以很直观地看出，在使用五折交叉验证对三个模型准确率进行多次计算，ID3模型和CART模型的预测准确率较为集中，且两个模型预测准确率高度重合，而CART模型预测准确率相比于另外两个模型较为分散。

具体地，我们计算出使用整个数据集作为训练集和测试集，以及上述这100次五折交叉验证的平均结果如下：

五折交叉验证预测准确率			
模型	最大值	最小值	平均值
ID3	0.9751124438	0.9577761119	0.9695340330
C4.5	0.9751124438	0.9577761119	0.9695340330
CART	0.9751274363	0.9577761119	0.9689091954
结果截图	Max ID3: 0.975112443778111 C4.5: 0.975112443778111 CART: 0.9751274362818592	Min ID3: 0.957776111944028 C4.5: 0.957776111944028 CART: 0.957776111944028	Average ID3: 0.9695340329835083 C4.5: 0.9695340329835083 CART: 0.9689091954022987

从表中我们可以看到，使用ID3模型和C4.5模型可以得到相同的预测准确率，这里我猜想这可能与所给的数据集有关。而CART模型的预测准确率不如ID3模型和C4.5模型高，使用ID3模型和C4.5模型进行决策树构建能得到更好的效果。

5. 思考题

5.1. 决策树有哪些避免过拟合的方法?

1. 预剪枝

a) 定义一个高度，当决策树达到该高度时，停止分裂。

b) 定义一个阈值，当达到某个节点的样本个数小于该阈值时，停止分裂。

c) 在决策树的构建过程中，选择某个特征进行分裂。如果分裂后，决策树在验证集上的准确率不提高，则无需分裂。

2. 后剪枝

a) 先构建一个完整的决策树，再自底向上地按后序遍历对非叶子节点进行考察。

b) 对于某个非叶子节点，假如使其成为叶子结点（该叶子节点的标签为该节点子树中标签的众数），决策树在验证集上的准确集不降低，则将它变成叶子节点。

5.2. C4.5相比于ID3的优点是什么？C4.5有可能有什么缺点?

C4.5优点：

- ID3算法使用信息增益作为分裂指标，偏向于选择属性取值个数多的特征。而C4.5算法使用信息增益率作为分裂指标，克服了这一不足。
- ID3无法处理连续型属性。而C4.5采用单点离散化的思想，用信息增益率来进行连续值特征的分裂属性值选择，可以处理连续型属性。

C4.5缺点：

- C4.5算法较为复杂，例如在处理连续值属性部分耗时较多。

5.3. 如何用决策树来进行特征选择（判断特征的重要性）?

决策树是一种基于贪心策略的树形结构。树中包含了内部节点和叶子节点：内部节点表示某一种属性，每次都选择最优的属性进行分裂，该内部节点的每个分支代表了在该属性上所有可能取值的输出；而叶子节点则表示预测的标签结果。

在已经建好的决策树上进行分类，输入为样本的特征向量（即在各个属性上的取值），输出为所预测的标签。从根节点开始，经过内部节点时，根据分裂的属性和该特征向量在该属性上的取值进入对应的分支，直到该特征向量到达叶子节点。该叶子节点的标签取值即为该文本特征向量的预测标签结果。

我们可以容易看出，在每个节点上如何选择分裂属性是很重要的。我们需要在决策树每个内部节点上选择最优的分裂属性进行分裂。

在ID3模型中，我们使用信息增益作为衡量指标。我们根据计算得到的各种特征对数据集的条件熵以及信息熵，找到信息增益最大的特征作为该节点的分裂特征。

在C4.5模型中，我们使用信息增益率作为衡量指标。ID3算法使用信息增益作为分裂指标，偏向于选择属性取值个数多的特征。而使用信息增益率则可以避免这个问题，我们选择信息增益率最大的特征作为该节点的分裂特征。

在CART模型中，我们使用Gini系数作为衡量指标。我们需要计算各个特征对应的Gini系数，然后选择Gini系数最小的特征作为该节点的分裂特征。