实验2：决策树

> 教学班级：计科2班
>
> 专业：计算机科学与技术
>
> 学号：19335262
>
> 姓名：张航悦

一、 实验内容

1. 算法原理

决策树

决策树是一种利用树形结构来实现分类问题的方法。其中每个内部节点表示一个属性上的判断，每个分支代表一个判断结果的输出，最后每个叶节点代表一种分类结果。

建树过程

- 根据指标选择最佳属性作为根，递归地划分训练样本建树
- 停止划分的条件
  - 被分到同一个节点内的所有数据样本都属于同一个标签
  - 属性集为空集，则将当前节点记为叶子节点，标记为出现最多的类

- 数据集为空集，标记为父节点中出现最多的类

决策树算法

- ID3
  - 使用信息增益作为指标来选择属性。首先计算出数据集D的经验熵，后遍历所有可选属性，分别计算每个属性对数据集的经验熵，将属性选取前后的信息熵作差便得到了信息增益。信息增益越大，表明选择该属性后熵越小，最大程度地减少了混乱，选择性越好。
  - 但同时以信息增益作为指标的方法，会使整个模型倾向于选择子类别多的属性进行划分。因为分得越细的数据集确定性更高,也就是条件熵越小,信息增益越大，从而易造成过拟合的问题。
  - 步骤
    - 计算数据集D的经验熵

      $H(D) = -\sum_{d \in D} p(d) log p(d)$
    - 计算特征A对数据集D的条件熵

      $H(D|A) = -\sum_{a \in A} p(a) H(D|A = a)$
    - 计算信息增益

      $g(D, A) = H(D) - H(D|A)$
    - 选择信息增益最大的特征作为决策点

- C4.5
  - 使用信息增益率作为指标来选择属性。信息增益率即计算出信息增益后除以每个属性的熵。改进了ID3算法，这样就降低了子类别多的属性的权重，避免了倾向于选择子类别多的属性进行划分。
  - 步骤
    - 计算每个特征的信息增益
    - 计算数据集D关于特征A的值的熵

      $SplitInfo(D, A) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} * log(\frac{|D_j|}{|D|})$
    - 计算信息增益率

      $gRatio(D, A) = (H(D) - H(D|A))/SplitInfo(D, A)$
    - 选择信息增益率最大的特征作为决策点

- CART
  - 使用GINI系数作为指标来选择属性。
  - ID3，C4.5进行熵运算时，涉及了大量的对数操作，较为复杂，于是我们思考能不能简化但又不完全丢失熵模型的优点，于是引入了GINI系数的概念。CART算法用GINI系数基尼系数来代替信息增益比，基尼系数代表了模型的不纯度，基尼系数越小，则不纯度越低，特征越好。
  - 步骤
    - 计算特征A的条件下，数据集D的GINI系数

      v表示属性A的取值个数，n表示类别个数

      $gini(D,A)=\sum_{j=1}^{v}p(A_j)*gini(D_j|A=A_j)$

      $gini(D_j|A=A_j)=\sum_{i=1}^{n}p_i*(1-p_i)=1-\sum_{i=1}^{n}p_i^2$
    - 选择GINI系数最小的特征作为决策点

k-fold k折交叉认证

本次实验利用老师上课所讲的k-fold法来划分训练验证集。

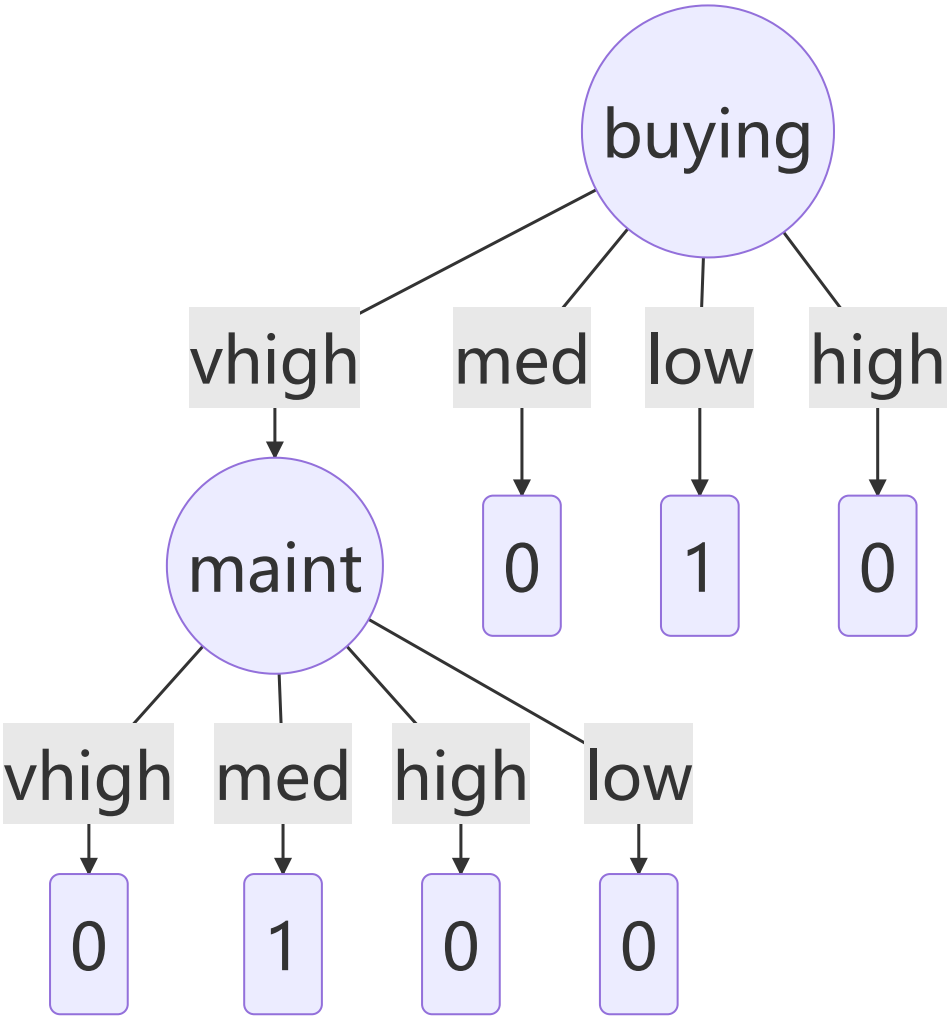
将原始数据集分为k个大小相似互斥的子集，每次选择1个子集作为验证集，剩余的k-1个子集作为训练集。遍历所有k个子集，这样就会得到k个模型，这k个模型分别在验证集评估准确率，最后将准确率加和平均的结果作为这个k值对应的准确率。

python建树

在本次实验中，我利用多级字典嵌套实现树的数据结构。

例如一下多级字典可以表示出如下的一棵树。

```
{'buying': {'vhigh': {'maint': {'vhigh': 0, 'med': 1, 'high': 0, 'low': 0}}, 'med': 0, 'low': 1, 'high': 0}}
```



2. 伪代码

- 选择最佳属性

```
Procedure choose_attribute(dataset,method)
//input:dataset:数据集
//      attri_dict:属性取值集合
//      avail_index:可选属性的下标值集合
//      method:建树算法
//output:最佳属性的下标值

    //采用ID3算法
    if method=="ID3" then
        empirical_entropy=cal_entropy //计算经验熵
        info_gain:=[]
        for i in avail_index
```

```
            conditional_entopy = 0.0
            for sub_attri in attri_dixt[i] do
                conditional_entropy:=cal_entropy //计算条件熵
            end for
            info_gain.append(empirical_entropy-
conditional_entropy) //计算信息增益
        end for
        best_index:=max(info_gain)
        return best_index
    end if

    //采用C4.5算法啊
    else if methood=="C4.5" then
        info_gain_ratio:=[]
        for i in avail_index do
            split_info:=cal_entropy //计算特征i的信息熵
            info_gain_ratio.append(info_gain/split_info)
        end for
        best_index:=max(info_gain_ratio)
        return best_index
    end if

    //采用CART算法
    else if method=="CART" then
        gini_list:=[]
        for i in avail_index do
            gini_temp:=cal_gini
            gini_list.append(gini_temp)
        end for
        min_index:=min(gini_list)
        return min_index
    end if
```

- 建树

```
Procedure build_tree(dataset,attri_dict,avail_index,method)
//input:dataset:数据集
//      attri_dict:属性取值集合
//      avail_index:可选属性的下标值集合
//      method:建树算法
//output:决策树

  //边界条件
```

```
  if 数据集为空 then
    return 父节点的label
  end if

  if 所有样本的label都相同 then
    return label
  end if

  if 属性集为空 then
    return 当前label的众数
  end if

  best_atrri:=choose_attribute(dataset，method) //选择最佳属性
  avail_index.remove(best_attri) //从可选属性集合中删除该属性

  //递归建树
  my_tree={best_attri:{}}
  for sub_attri in atrri_dict[best_attri] do
    sub_dataset=get_sub_dataset(dataset,best_attri,sub_attri)
    my_tree[best_attri]
[sub_attri]=build_tree(sub_dataset,attri_dict,avail_index,met
hod)
  end for

  return my_tree
```

- 预测标签

```
Function predict(line,input_tree,labelset)
//input:line:待预测的一条数据
//      input_tree:决策树，字典构成
//      label_set:label标签集合
//output:该条数据预测的label

    first_attri:=根节点字典键值     //获取根节点属性名
    key:=line[first_attri]  //获取数据在该属性的取值
    value:=input_tree[first_attri][key] //value为根节点在该取值下
对应的子树

    //递归分类预测label
    if value仍为字典结构 then //说明value还为中间节点 继续递归
        label=predict(line，input_tree,labelset)
    end if
```

```
    else label=value

    return label
```

3. 关键代码展示

- k-fold 划分训练集验证集

```
def k_fold(dataset, k, i):
    """
    划分训练集和验证集
    :param dataset: 数据集
    :param k: 将数据集分为k个子集
    :param i: 选取第i个子集作为验证集
    :return: 划分出的训练集和验证集
    """
    total = len(dataset)
    step = total // k  # 每一步的步长 向下取整
    start = i * step
    end = start + step
    train_set = dataset[:start] + dataset[end:]
    valid_set = dataset[start:end]
    return train_set, valid_set
```

- 计算熵，用于ID3，C4.5算法

```
def cal_entropy(subdataset, index):
    """
    用于计算当前数据集的经验熵(index=-1时，相当于传入label)、条件熵
    :param subdataset:前六列为属性值 最后一列为标签值
    :param index: 目标列号
    :return: 熵值
    """
    size = len(subdataset)
    count = {}  # 统计各个属性的数量
    for line in subdataset:
        temp = line[index]
        count[temp] = count.get(temp, 0) + 1

    result = 0.0
```

```
    for i in count.values():
        p = float(i) / size
        if p != 0:
            result -= p * math.log2(p)

    return result
```

- 计算GINI系数

```
def cal_gini(dataset, attri):
    """
    用于计算GINI系数
    :param dataset:数据集
    :param attri:要计算GINI系数的属性的下标
    :return:
    """
    subattri_count = {}  # 记录该特征 的 子属性的 个数
    subatrri_label = {}  # 记录子属性 对应的label的 个数
    total = len(dataset)
    for line in dataset:
        temp = line[attri]
        # 子属性取值的个数统计
        subattri_count[temp] = subattri_count.get(temp, 0) +
1
        # 子属性的标签个数统计
        subatrri_label[temp] = subatrri_label.get(temp, {})
        if line[-1] not in subatrri_label[temp]:
            subatrri_label[temp][line[-1]] = 0
        subatrri_label[temp][line[-1]] += 1

    gini = 0
    for i in subattri_count.keys():
        num = subattri_count[i]  # 该子属性的个数
        gini_temp = 1
        for value in subatrri_label[i].values():
            gini_temp -= np.square(value / num)
        gini += num / total * gini_temp
    return gini

```

- 选择最佳属性

```python
def choose_attribute(dataset, attribute_dict, avail_index, method):
    """

    :param dataset:数据集
    :param attribute_dict:属性取值集合
    :param avail_index: 当前可选的属性的下标 下标号与attribut_dict存放的下标对应
    :param method:建树算法
    :return:最佳属性的下标
    """
    if method == "ID3":
        empirical_entropy = cal_entropy(dataset, -1)  # 计算经验熵
        info_gain = []  # 信息增益列表

        # 遍历可选属性，计算每个属性的条件熵
        for i in avail_index:
            conditional_entopy = 0.0
            for sub_attribute in attribute_dict[i]:  # 遍历该属性的所有可能取值
                sub_dataset = get_sub_dataset(dataset, i, sub_attribute)
                p = len(sub_dataset) / len(dataset)  # 该属性在数据集中的比例
                conditional_entopy += p * cal_entropy(sub_dataset, -1)  # 条件熵计算
            info_gain.append(empirical_entropy - conditional_entopy)
        # 找出信息增益最大的属性
        best_index = np.argmax(info_gain)
        return avail_index[best_index]  # 返回信息增益最大的属性的下标

    elif method == "C4.5":
        empirical_entropy = cal_entropy(dataset, -1)
        info_gain_ratio = []
        for i in avail_index:
            conditional_entopy = 0.0
            for sub_attribute in attribute_dict[i]:  # 遍历该属性的所有可能取值
                sub_dataset = get_sub_dataset(dataset, i, sub_attribute)
                p = len(sub_dataset) / len(dataset)  # 该属性在数据集中的比例
```

```
                conditional_entopy += p * cal_entropy(sub_dataset, -1)  # 条件熵计算
            split_info = cal_entropy(dataset, i)
            if split_info == 0:  # 说明这个属性所有取值相同，对决策分裂没有意义
                continue
            info_gain_ratio.append(((empirical_entropy - conditional_entopy) / split_info, i))
        best_index = max(info_gain_ratio, key=lambda x: x[0])
        return best_index[1]  # 返回信息增益率最大的属性的下标

    elif method == "CART":
        gini_list = []
        for i in avail_index:
            gini_temp = cal_gini(dataset, i)
            gini_list.append(gini_temp)
        min_index = np.argmin(gini_list)
        return avail_index[min_index]  # 返回GINI系数最小的属性的下标

```

- 建决策树

```
def build_tree(dataset, attri_dict, avail_index, parent_lable, method, labelset):
    """
    建树
    :param dataset:数据集
    :param attri_dict:属性取值集合
    :param avail_index:可选属性下标集合
    :param parent_lable:父节点的label
    :param method:建树算法
    :param labelset:属性名称集合
    :return:根节点
    """
    label_list = [record[-1] for record in dataset]
    # 边界条件
    # 若dataset为空集，则取父节点的label
    if len(dataset) == 0:
        return parent_lable
    # 若所有样本的label都相同，直接取label
    if label_list.count(label_list[0]) == len(label_list):
        return label_list[0]
```

```
    # 若属性集为空
    if len(avail_index) == 0:
        temp = max(label_list, key=label_list.count)
        return temp

    best_attri = choose_attribute(dataset, attri_dict, avail_index, method)  # 选择最佳属性作为根节点
    avail_index.remove(best_attri)  # 从可选属性中去除该属性
    my_tree = {labelset[best_attri]: {}}  # 以该属性的 名称 为根创建树
    parent_lable = max(label_list, key=label_list.count)  # 获得父节点的label

    # 划分子属性 递归建树
    for sub_attri in attri_dict[best_attri]:
        sub_dataset = get_sub_dataset(dataset, best_attri, sub_attri)
        my_tree[labelset[best_attri]][sub_attri] = build_tree(sub_dataset, attri_dict, avail_index[:], parent_lable,

method, labelset)
    return my_tree
```

- 预测标签

```
def predict(line, input_tree, labelset):
    """
    预测数据的label
    :param line: 带预测的一行数据
    :param input_tree: 决策树
    :param labelset: 属性名称集合
    :return: 预测标签
    """
    first_attri = list(input_tree.keys())[0]  # 获得根节点的属性名称
    second_dict = input_tree[first_attri]  # 获得根节点对应子树
    index = labelset.index(first_attri)  # 获得该属性在属性集合中的对应下标
    key = line[index]  # 获得该数据在根属性的取值
    value = second_dict[key]  # 进入该取值对应的子树
    if isinstance(value, dict):  # 若非叶子结点，继续递归
        label = predict(line, value, labelset)
```

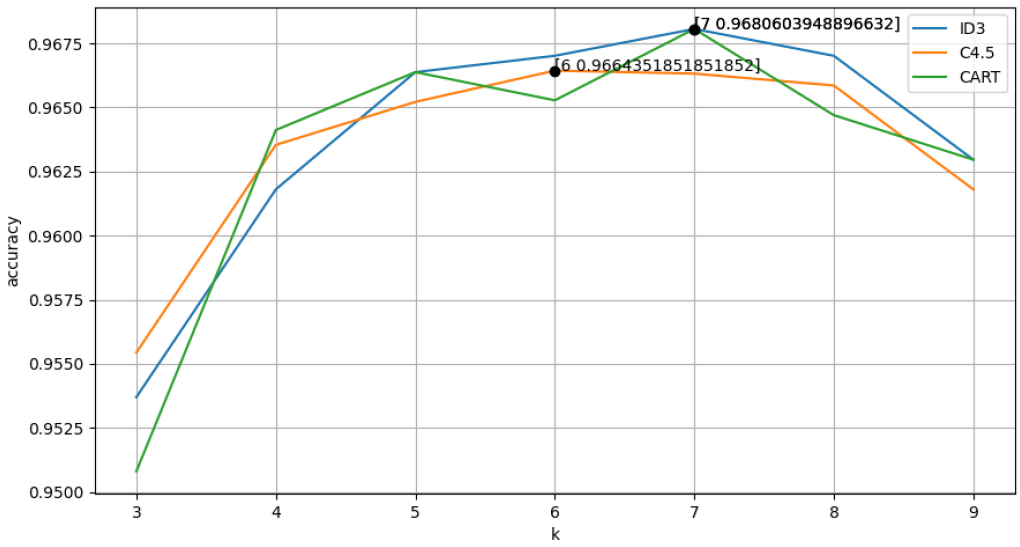
```
    else:
        label = value

    return label
```

二、 实验结果及分析

- 实验结果展示

调整k_fold验证的k值，划分出不同数量的训练集和验证集进行实验。得到如下图所示的结果。

整体三种算法的准确率都随着k值增大先增后降，ID3和CART在k=7时取得最高准确率0.96806，C4.5在k=6时取得最高准确率0.96644。



- 性能分析

在本次实验提供的数据集上，三种算法的性能相差不多，准确率都较高。

对比ID3和C4.5算法，两者折线图整体趋势相同，但是让我意外的是C4.5的性能并没有优于ID3，反而在大部分k值都略差于ID3。我觉得原因可能为，本次实验给出的数据集每个属性的子特征选项都不是很多，且数量较为平均，因此导致归一化的方法的优化效果并不是十分的明显，甚至导致了准确率的下降。

当k较小时，由于划分出的训练集样本较小，导致模型训练效果不佳，在验证集上准确率不高。但当k较大时，虽然模型可以得到较好的训练，但由于测试集样本较小，得到的验证集的准确率的随机性也会比较大，从而导致准确率的下降。

三、 思考题

- 决策树有哪些避免过拟合的方法?
  - 若样本中的噪音数据干扰过大，或者样本抽取错误等问题，会导致决策树过拟合于当前样本。针对这种问题，解决方法为有效地抽样，用相对能够反映业务逻辑的训练集去产生决策树。
  - 更主要的避免过拟合的方法为剪枝。分为预剪枝和后剪枝两种。
    - 预剪枝：在决策树生成过程中进行。对于当前结点，判断是否应当继续划分。如果无需划分，则直接将当前结点设置为叶子结点。
    - 后剪枝：先生成完整的决策树，再自底向上地对非叶结点进行考察。

- C4.5相比于ID3的优点是什么，C4.5有可能有什么缺点?
  - 优点

    ID3以信息增益作为指标的方法，会使整个模型倾向于选择子类别多的属性进行划分。因为分得越细的数据集确定性更高,也就是条件熵越小,信息增益越大。C4.5采用信息增益率作为指标，信息增益率计算出信息增益后除以每个属性的熵，这样就降低了子类别多的属性的权重，避免了倾向于选择子类别多的属性进行划分。此外通过将连续型的属性进行离散化处理，克服ID3算法不能处理连续型数据缺陷
  - 缺点

    再构造树的过程中，需要对数据集进行多次的顺序扫描和排序，因而导致算法的低效。针对含有连续属性值的训练样本时，算法计算效率较低。此外只适合于能够驻留于内存的数据集，当训练集大得无法在内存容纳时程序无法运行。

- 如何用决策树来进行特征选择（判断特征的重要性）？

  ID3算法采用信息增益作为指标，信息增益越大的特征越重要。C4.5算法采用信息增益率作为指标，信息增益率越大的特征越重要。CART算法采用GINI系数作为指标，GINI系数越小的特征越重要。