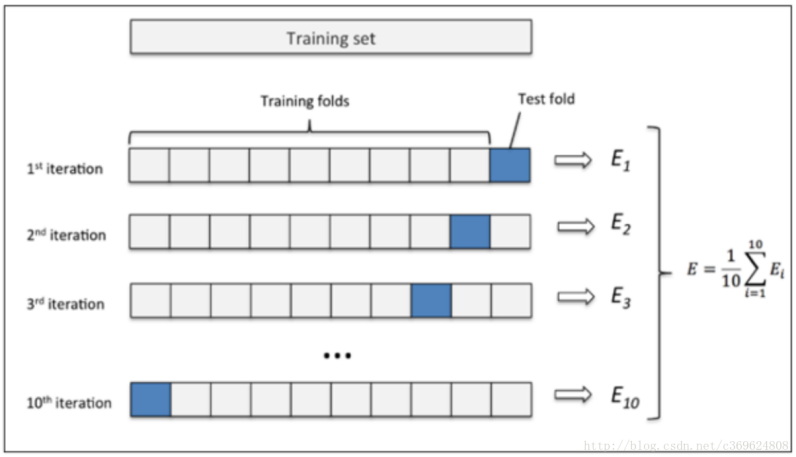
人工智能lab2 实验报告

> 学号：
>
> 姓名：TRY
>
> 专业：计算机科学与技术

算法原理

- 本次实验要求实现ID3, C4.5, CART三种决策树
- 建树的步骤主要有6步：初始化（划分训练集和验证集）、**选择特征**（ID3\C4.5\CART三种方法）、**划分数据**（根据特征，得到子数据集）、**创建节点**（创建子节点）、**递归建树**（对于每个节点，回到第二步"选择特征"，直到到达**边界条件**回溯）、完成建树（取众数）
- **其中，边界条件有3种：**
  - 假设数据集为D，特征集为A
  - **条件1**：D中的样本属于同一label C， 则将当前结点标记为C类叶结点。
  - **条件2**：A为空集，或D中所有样本在A中所有特征上取值相同， 此时无法划分。 将当前结点标记为叶结点， 类别为D中出现最多的类（取众数）。
  - **条件3**：D为空集，则将当前结点标记为叶结点， 类别为**父结点**中出现最多的类。
- **特征选择方法有3种：**
  - **方法一：ID3算法，利用信息增益来选择特征属性**
    - 信息增益： $g(D,A)=H(D)-H(D|A)$
      - 经验熵= $H(D)=-\sum_{d\in D}p(d)logp(d)$
      - 条件熵= $H(D|A)=\sum_{a\in A}p(a)H(D|A=a)$
    - 选择**信息增益最大**的特征作为当前决策点
  - **方法二：C4.5算法，利用信息增益率来选择特征属性**
    - 信息增益率：$gainRatio(D,A)=(H(D)-H(D|A))/SplitInfo(D,A)$
      - 信息增益：$g(D,A)=H(D)-H(D|A)$
      - 数据集D关于特征A的熵$SplitInfo(D,A)=-\sum_{j=1}^{v}\frac{|D_j|}{|D|}*log(\frac{|D_j|}{|D|})$
        - 实际就是H(A)
    - 选择**信息增益率最大**的特征作为当前决策点
  - **方法三：CART算法，利用GINI指数来选择特征属性**
    - GINI系数：值越小代表不确定性越小
    - 特征A的条件下，数据集D的GINI系数：
      $gini(D,A)=\sum_{j=1}^{v}p(A_j)*gini(D_j|A=A_j)$
    - 其中，$gini(D_j|A=A_j)=\sum_{i=1}^{n}p_i(1-p_i)=1-\sum_{i=1}^{n}p_i^2$
      - v表示属性A的取值个数，n表示label类别个数
    - 选择**GINI系数最小**的特征作为当前决策点

- **如何划分训练集和验证集?**
  - 关于这个问题，一开始我思考的是直接选取一定比例的数据集作为验证集，也就是**“留出法Hold-out”**
    - 但由于“一定比例”不好掌控，且选取不随机，可能导致结果误差较大。
  - 后来，经过查找资料，了解到了**“交叉验证法Cross validation"**的原理， 先将数据集D划分为k个大小相似的互斥子集，然后遍历k个子集，每次用k-1个子集的并集作为训练集，余下的1个子集作为验证集；这样就可以获得k组训练/验证集，从而可以进行k次训练和验证，最终返回的是k个验证结果的均值，作为这1个k值对应的准确率。
    - 如当K=10时，示意图如下：



- **如何表示树?**
  - 由于python中，没有“指针”这个概念，所以需要通过list列表**branch**来储存子树。
  - 而由于在边界条件中，需要用到父节点的类别，所以需要变量**label**来储存节点的label。
    - **注意：**实际上，对于叶结点这个label才是真正有意义的，表示叶结点的类别。对于非叶结点来说，只是为了提供给子节点才记录而已。
  - 当然，也要记录此时该节点所选择用来划分的特征**attribute**。
  - 因此，决策树节点的数据结构如下：

```
class DecisionNode(object):
    def __init__(self, label=None, attribute=None, branch=None):
        self.attribute = attribute  # 节点的属性标签
        self.label = label  # 保存当前分支的label，当是叶结点的时候值才有真正意义
        self.branch = branch
```

伪代码

- 以下只展示本实验中几个关键函数的伪代码：

```
Function cal_entropy(subdataset, index)
/*可以用来计算经验熵（index=-1)或者某个属性的熵，或者条件熵（dataset经过处理），用于ID3或C4.5*/
    size := subdataset的长度
    count := {}
    for line in subdataset
```

```
        label := line的第index个属性的取值
        count中label对应的个数+1
    end for
    result := 0
    for i in count的值集合
        p := i / size
        if p!=0 then
            result -= p*log2(p)
        end if
    end for
    return result
```

```
Function cal_gini(dataset, attribute)
    sub_attribute_count := {}    /*key为子属性，value为子属性的个数*/
    sub_attribute_label := {}    /*key为子属性，value类型为{label:label数量}，储存子属性所含的label及其数量*/
    total := dataset总长度
    for line in dataset
        sub_attribute := line第attribute列对应的取值
        sub_attribute_count的sub_attribute单元+1
        sub_attribute_label单元添加line对应的label，并更新label对应的数量
    end for
    gini := 0
    for i1 in sub_attribute_count.keys()
        size := 子属性为sub_attribute的个数
        gini_temp := 1
        for value in sub_attribute_label[i1]对应的value中
            gini_temp -= value/size的平方
        end for
        gini += size/total * gini_temp
    return gini
```

```
Function choose_best_attribute(dataset, attribute_dic, available_attribute, method)
/*attribute_dic为每个属性对应的子属性的集合字典，available_attribute为当前可以选择的属性，储存的是属性对应的下标，method表示使用什么策略*/

    if 选择ID3策略 then
        经验熵 := call cal_entropy(dataset, -1)/*-1表示最后一列*/
        info_gain_list := []    /*信息增益的数组*/
        for attribute in available_attribute:
            条件熵 := 0
            遍历attribute对应的子属性，得到由子属性划分形成的子数据集，根据子数据集求出条件熵
            将“经验熵-条件熵”的结果添加到info_gain_list末尾
        end for
        通过np.argmax函数求出info_gain_list中的最大元素的下标
        return available_attribute[max_index]/*该下标对应的属性的下标*/

    else if 选择C4.5策略 then
        经验熵 := call cal_entropy(dataset, -1)/*-1表示最后一列*/
        info_gain_ratio_list := []  /*信息增益率的数组*/
        for attribute in available_attribute:
            条件熵 := 0
            遍历attribute对应的子属性，得到由子属性划分形成的子数据集，根据子数据集求出条件熵
```

```
            属性attribute的熵split_info := call cal_entropy(dataset, attribtue)
            if split_info = 0 then
                continue
            end if
            将“(经验熵-条件熵)/split_info”的结果添加到info_gain_ratio_list末尾
        end for
        通过np.argmax函数求出info_gain_ratio_list中的最大元素的下标
        return available_attribute[max_index]

    else if 选择CART策略 then
        gini_list := []
        for attribute in available_attribute
            gini_temp := call cal_gini(dataset, attribute)
            将gini_temp加入到gini_list末尾
        end for
        通过np.argmin函数求出gini_list中的最小元素的下标
        return available_attribute[min_index]

    end if
```

```
Function create_tree(dataset, available_attribute, attribute_dic, parent_label,
method)
/*建树函数*/
    label_list := dataset中的最后一列label
    if dataset为空集 then
        return DecisionNode(parent_label)
    end if
    if dataset里面的样本都取同一label then
        return DecisionNode(label=共同的label)
    end if
    if 当没有属性可选时，即available_atttribute为空 then
        lable := 众数标签
        return DecisionNode(label=label)
    end if

    best_attribute := call choose_best_attribute(dataset, attribute_dic,
available_attribute, strategy)
    更新available_attribute,即把best_attribute从available_attribute中移除
    branch := {}     /*子节点初始化为空*/
    parent_label := dataset中的众数标签
    branch := 遍历best_attribute对应的子属性，得到子数据集，并根据子数据集和更新后的
available_attribute建树
    return DecisionNode(label=parent_label, attribute=best_attribute,
branch=branch)
```

```
Function validation(valid_set, root)     /*计算准确率*/
    cnt := 0
    for line in valid_set
        cur := root
        while cur.branch非空 do
            cur := cur.branch[line在该节点属性的取值]
        end
        if cur.label = line[-1]
            cnt += 1
        end if
    end for
    return cnt/valid_set的长度      /*准确率*/
```

代码截图

- 决策树节点数据结构：

```
class DecisionNode(object):
    # 类似于C里面的构造函数，初始化类实例的属性
    def __init__(self, label=None, attribute=None, branch=None):
        self.attribute = attribute  # 节点的属性标签
        self.label = label  # 保存当前分支的label，当是叶结点的时候值才有真正意义
        self.branch = branch
```

- 读文件readfile，使用了pandas库完成：

```
def readfile(filename):
    """
    将数据和label读出来，存入labelSet和dataSet中
    :return: labelSet & dataSet
    """
    dataset = pd.read_csv(filename)
    labelset = list(dataset.columns.values)      # 这样可以读出属性名
    dataset = list(dataset.values)      # 读出数据，存到列表中，这里去除了第一行的表头！！
    return dataset, labelset
```

- 交叉验证函数k_fold，将数据集划分为训练集+验证集：

```
def k_fold(dataset, k, i):
    """
    "交叉验证法“ 划分数据集为训练集+验证集
    :param dataset: 大数据集
    :param k:划分的总份数
    :param i:取第i份作为验证集
    :return:返回训练集和验证集
    """
    total = len(dataset)
    step = total // k    # 这样可以返回下取整: 步长
    start1 = i * step
    end1 = start1 + step
```

```
    train_set = dataset[:start1] + dataset[end1:]
    valid_set = dataset[start1:end1]
    return train_set, valid_set
```

- 获取属性字典，即统计各个属性所含有的子属性名称：

```
def get_attribute_dic(dataset, labelset):
    """
    计算各属性的可能取值
    :param dataset: 数据集
    :param labelset: 属性名称集合
    :return: 返回属性字典：键值key为labelset中的下标，值value为对应属性可能的取值
    """
    attribute_num = len(labelset) - 1   # 属性个数：减去最后的标签
    attribute_dic = {}
    for i in range(attribute_num):
        temp_set = set()
        for line in dataset:
            temp_set.add(line[i])     # 获取当前特征所有可能的取值
        attribute_dic[i] = temp_set
    return attribute_dic
```

- 计算经验熵或某个属性的熵，用于C4.5 & ID3：

```
def cal_entropy(subdataset, index):
    """
    可以用来计算经验熵（index=-1)或者某个属性的熵，或者条件熵（dataset经过处理），用于ID3或C4.5
    :param subdataset: 数据集
    :param index: 目标的列号
    :return: 返回熵值
    """
    size = len(subdataset)
    count = {}  # 存储label或attribute的取值及其数量
    for line in subdataset:
        label = line[index]
        count[label] = count.get(label, 0) + 1
    result = 0.0
    for i in count.values():
        p = float(i) / size
        if p != 0:
            result -= p * math.log2(p)
            # result -= p * np.log2(p)
    return result
```

- 计算GINI指数：设置两个字典

```
def cal_gini(dataset, attribute):
    """
    计算CART方法的指标gini。只计算对应属性的gini值
    :param dataset: 数据集
    :param attribute: 属性下标
    :return: 返回gini值
    """
    sub_attribute_count = {}    # 储存每个子属性的个数
    sub_attribute_label = {}    # 储存每个子属性所含的label及数量
```

```
    total = len(dataset)
    for line in dataset:
        sub_attribute = line[attribute]
        sub_attribute_count[sub_attribute] =
sub_attribute_count.get(sub_attribute, 0)
        sub_attribute_count[sub_attribute] += 1     # 次数+1
        sub_attribute_label[sub_attribute] =
sub_attribute_label.get(sub_attribute, {})     # get默认返回空的dictionary，然
后赋值创建
        if line[-1] not in sub_attribute_label[sub_attribute]:
            sub_attribute_label[sub_attribute][line[-1]] = 0    # 将对应的
label的次数赋值为0
        sub_attribute_label[sub_attribute][line[-1]] += 1

    gini = 0
    for i1 in sub_attribute_count.keys():
        size = sub_attribute_count[i1]   # sub_attribute的个数
        gini_temp = 1
        for value in sub_attribute_label[i1].values():
            gini_temp -= np.square(value / size)
        gini += size / total * gini_temp
    return gini
```

- 获取子数据集：

```
def get_sub_dataset(dataset, index, attribute):
    """
    从数据集中提取出某个属性取值相同的部分
    :param dataset: 数据集
    :param index: which 属性
    :param attribute: 属性取值
    :return: 返回子数据集
    """
    sub_dataset = []
    for record in dataset:
        if record[index] == attribute:
            sub_dataset.append(record)
    return sub_dataset
```

- 选择最好的属性choose_best_attribute：可以利用3种策略

```
def choose_best_attribute(dataset, attribute_dic, available_attribute,
strategy):
    """
    根据某一个指标（ID3,C4.5,CART）来选择最优的属性作为当前子树的划分标准
    :param dataset: 数据集
    :param attribute_dic: 属性字典
    :param available_attribute: 当前可以选择的属性集合，存的是属性在attribute_dic
中对应的下标
    :param method: 指标
    :return:
    """
    if strategy == "ID3":
        data_size = len(dataset)    # 数据集的总长度
        empirical_entropy = cal_entropy(dataset, -1)  # 经验熵 empirical
entropy
```

```
        info_gain_list = []     # 信息增益的数组
        for attribute in available_attribute:   # 这里存储的是下标，表示属性
            conditional_entropy = 0.0   # 条件熵
            for sub_attribute in attribute_dic[attribute]:  # 遍历这个属性对应
的取值sub_attribute
                sub_dataset = get_sub_dataset(dataset, attribute,
sub_attribute)    # 获得对应属性的子数据集
                p = len(sub_dataset) / data_size    # p(a)
                conditional_entropy += p * cal_entropy(sub_dataset, -1)
# 计算条件熵
            # 这里不用insert（insert要指明下标），而是用append加到list最后，因为
attribute不是连续的
            info_gain_list.append(empirical_entropy - conditional_entropy)
        max_index = np.argmax(info_gain_list)  # 返回的是信息增益最大的下标
        # print(available_attribute[max_index])
        return available_attribute[max_index]   # 返回的是信息增益最大的属性的下
标

    elif strategy == "C4.5":
        data_size = len(dataset)  # 数据集的总长度
        empirical_entropy = cal_entropy(dataset, -1)  # 经验熵 empirical
entropy
        info_gain_ratio_list = []  # 信息增益率的数组
        for attribute in available_attribute:  # 这里存储的是下标，表示属性
            conditional_entropy = 0.0  # 条件熵
            for sub_attribute in attribute_dic[attribute]:  # 遍历这个属性对应
的取值sub_attribute
                sub_dataset = get_sub_dataset(dataset, attribute,
sub_attribute)  # 获得对应属性的子数据集
                p = len(sub_dataset) / data_size  # p(a)
                conditional_entropy += p * cal_entropy(sub_dataset, -1)  #
            split_info = cal_entropy(dataset, attribute)    # 计算特征
attribute的信息熵
            if split_info == 0:     # 证明这个attribute对决策没贡献（取得都是相同
的sub_attribute）
                continue
            # 这里不用insert（insert要指明下标），而是用append加到list最后，因为
attribute不是连续的
            info_gain_ratio_list.append((empirical_entropy -
conditional_entropy)/split_info)
        max_index = np.argmax(info_gain_ratio_list)  # 返回的是信息增益率最大的
下标
        return available_attribute[max_index]

    elif strategy == "CART":
        gini_list = []
        for attribute in available_attribute:
            gini_temp = cal_gini(dataset, attribute)
            gini_list.append(gini_temp)
        min_index = np.argmin(gini_list)
        return available_attribute[min_index]
```

- 建树函数creat_tree，返回的是节点：需要考虑**边界条件**和**递归条件**
    - **注意：**在这里修改了available_attribute之后，要传入值（副本），而不是默认的直接传引用！！否则准确率会比较低，且会变化！

```
def create_tree(dataset, available_attribute, attribute_dic,
parent_label, strategy):
    """
    构建决策树
    :param dataset: 数据集
    :param available_attribute: 可选属性
    :param attribute_dic: 属性字典
    :param parent_label: 父节点的label
    :param strategy: 选择属性时候的方法&指标
    :return: 根节点
    """
    label_list = [record[-1] for record in dataset]
    # 3个边界条件：
    # 条件3：dataset为空集,则取父节点的属性：这个要放在最前面！！cause有可能越
界！！
    if len(dataset) == 0:
        return DecisionNode(label=parent_label)
    # 条件1：当dataset里面的样本都取同一label
    if label_list.count(label_list[0]) == len(label_list):
        return DecisionNode(label=label_list[0])
    # 条件2：当没有属性可选时，即available_atttribute为空时
    if len(available_attribute) == 0:
        label = max(label_list, key=label_list.count)  # 找出众数标签
        return DecisionNode(label=label)
    # 选择出最好的特征,返回的是attribute中对应的下标
    best_attribute = choose_best_attribute(dataset, attribute_dic,
available_attribute, strategy)
    available_attribute.remove(best_attribute)
    branch = {}
    parent_label = max(label_list, key=label_list.count)     # 传给下一轮
递归，实际不一定有意义
    for sub_attribute in attribute_dic[best_attribute]:     # 利用最好的
属性对dataset进行划分，并构建子树
        sub_dataset = get_sub_dataset(dataset, best_attribute,
sub_attribute)
        # available_attribute[:]表示传值！不加的话是传引用！
        # 如果函数收到的是一个可变对象(比如字典或者列表)的引用,就能修改对象的原始
值--相当于通过“传引用”来传递对象。
        branch[sub_attribute] = create_tree(sub_dataset,
available_attribute[:], attribute_dic, parent_label, strategy)
    return DecisionNode(label=parent_label, attribute=best_attribute,
branch=branch)
```

- 计算验证集预测的准确率：

```
def validation(valid_set, root):
    """
    返回预测准确率
    :param valid_set: 验证集
    :param root: 根节点
    :return: 准确率
    """
    cnt = 0     # 记录预测正确的个数
    for line in valid_set:
        cur = root
        while cur.branch is not None:
            cur = cur.branch[line[cur.attribute]]
```

```
            if cur.label == line[-1]:
                cnt += 1
        return cnt/len(valid_set)
```

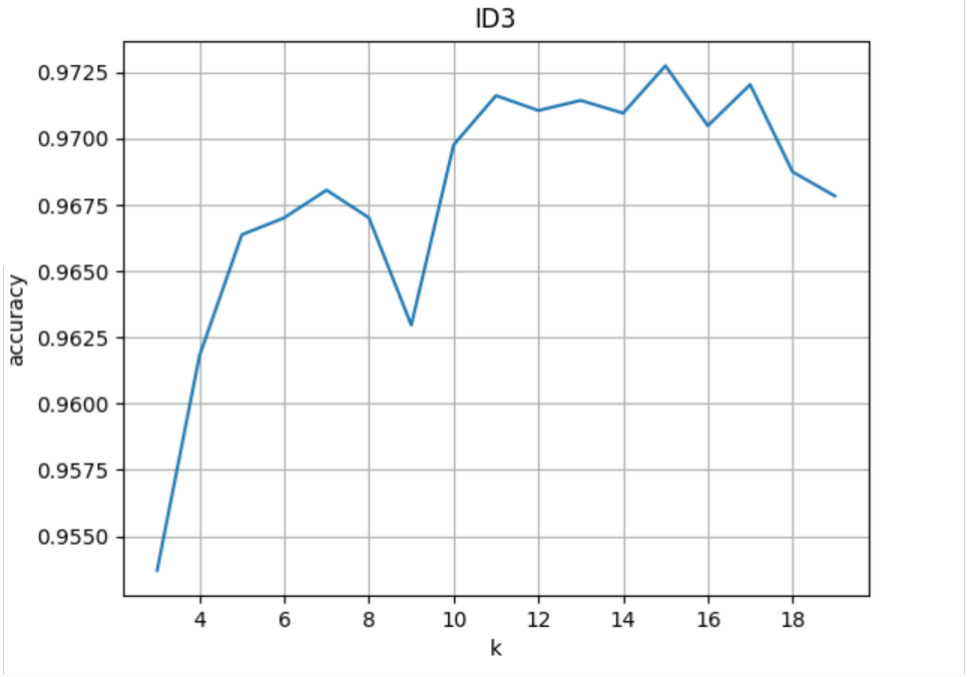
- 预测验证集，并输出结果，画图（matplotlib）：

```
def valid_predict():
    dataset, label_set = readfile('car_train.csv')
    x = []
    y = []
    strategy = "C4.5"
    for k in range(3, 20):  # 交叉验证
        temp = 0
        for i in range(k):  # 对不同的验证集取均值作为一个k的结果
            train_set, valid_set = k_fold(dataset, k, i)
            attribute_dic = get_attribute_dic(dataset, label_set)
            available_attribute = list(range(0, len(label_set) - 1))
            root = create_tree(train_set, available_attribute,
attribute_dic, -1, strategy)
            temp += validation(valid_set, root)
        print("利用%s方法，对数据集进行%s折划分后的准确率为%s" % (strategy, k,
temp / k))
        x.append(k)
        y.append(temp / k)
    plt.plot(x, y)
    plt.grid(True)
    plt.xlabel('k')
    plt.ylabel('accuracy')
    plt.title('%s' % strategy)
    plt.show()
```

实验结果以及分析

- **结果展示和分析**
    - **ID3算法：**

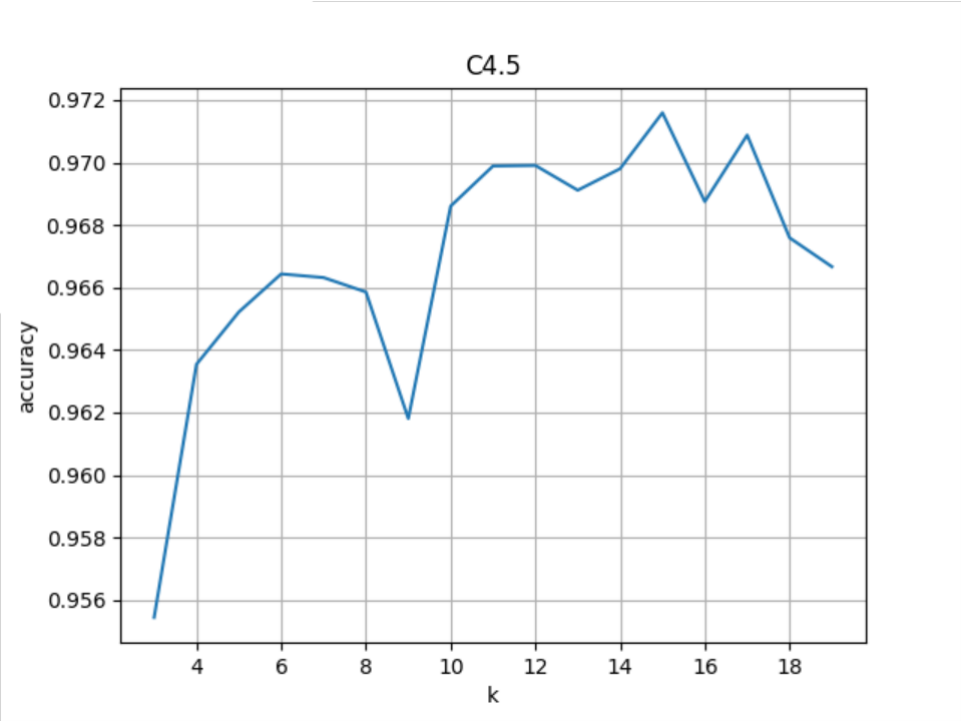
```
利用ID3方法，对数据集进行3折划分后的准确率为0.9537037037037037
利用ID3方法，对数据集进行4折划分后的准确率为0.9618055555555556
利用ID3方法，对数据集进行5折划分后的准确率为0.9663768115942029
利用ID3方法，对数据集进行6折划分后的准确率为0.967013888888889
利用ID3方法，对数据集进行7折划分后的准确率为0.9680603948896632
利用ID3方法，对数据集进行8折划分后的准确率为0.9670138888888888
利用ID3方法，对数据集进行9折划分后的准确率为0.9629629629629629
利用ID3方法，对数据集进行10折划分后的准确率为0.969767441860465
利用ID3方法，对数据集进行11折划分后的准确率为0.9716270990156339
利用ID3方法，对数据集进行12折划分后的准确率为0.9710648148148148
利用ID3方法，对数据集进行13折划分后的准确率为0.9714452214452215
利用ID3方法，对数据集进行14折划分后的准确率为0.9709639953542392
利用ID3方法，对数据集进行15折划分后的准确率为0.9727536231884056
利用ID3方法，对数据集进行16折划分后的准确率为0.9704861111111113
利用ID3方法，对数据集进行17折划分后的准确率为0.9720442632498544
利用ID3方法，对数据集进行18折划分后的准确率为0.96875
利用ID3方法，对数据集进行19折划分后的准确率为0.9678362573099417
```

- 

- 根据输出结果和折线图可知，利用ID3算法的信息增益指标进行特征选择时，准确率整体随着k的增大先上升后降低，但由于纵坐标的差值精确度很高，实际上准确率的差值也在2%以内。（由于k-fold方法的随机性，可知实验结果正确，且效果非常好）

- **C4.5算法：**

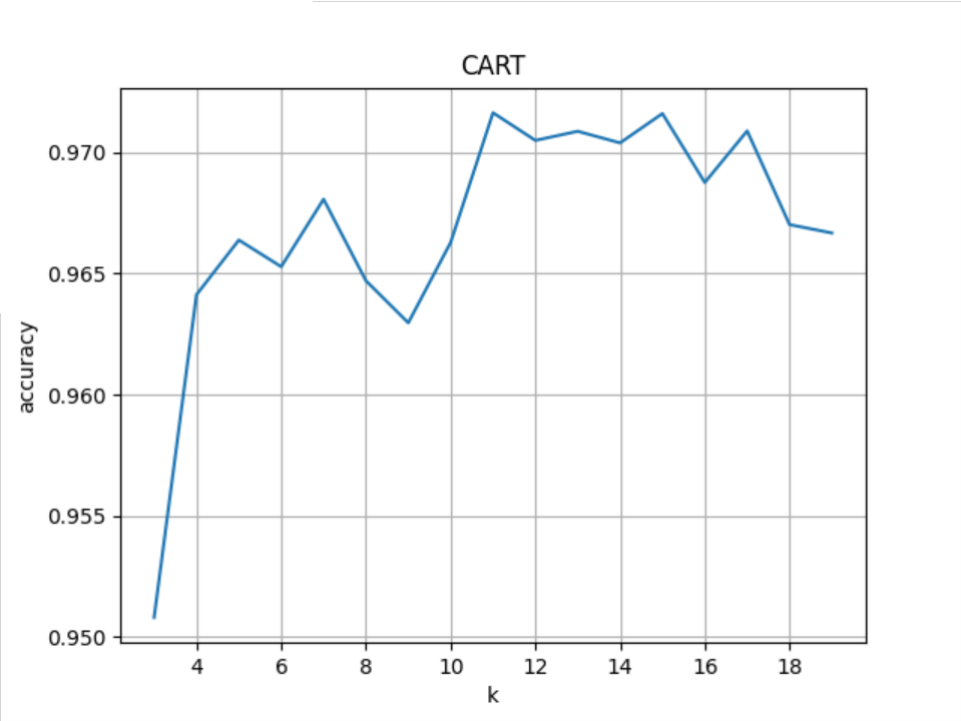
```
利用C4.5方法，对数据集进行3折划分后的准确率为0.9554398148148149
利用C4.5方法，对数据集进行4折划分后的准确率为0.9635416666666667
利用C4.5方法，对数据集进行5折划分后的准确率为0.9652173913043478
利用C4.5方法，对数据集进行6折划分后的准确率为0.9664351851851852
利用C4.5方法，对数据集进行7折划分后的准确率为0.9663182346109176
利用C4.5方法，对数据集进行8折划分后的准确率为0.9658564814814814
利用C4.5方法，对数据集进行9折划分后的准确率为0.9618055555555556
利用C4.5方法，对数据集进行10折划分后的准确率为0.9686046511627907
利用C4.5方法，对数据集进行11折划分后的准确率为0.9698899826288362
利用C4.5方法，对数据集进行12折划分后的准确率为0.9699074074074074
利用C4.5方法，对数据集进行13折划分后的准确率为0.9691142191142192
利用C4.5方法，对数据集进行14折划分后的准确率为0.9698025551684087
利用C4.5方法，对数据集进行15折划分后的准确率为0.9715942028985506
利用C4.5方法，对数据集进行16折划分后的准确率为0.9687500000000001
利用C4.5方法，对数据集进行17折划分后的准确率为0.9708794408852651
利用C4.5方法，对数据集进行18折划分后的准确率为0.9675925925925927
利用C4.5方法，对数据集进行19折划分后的准确率为0.9666666666666667
```

- 

  - 根据输出结果和折线图可知，利用C4.5算法的信息增益率指标进行特征选择时，准确率整体随着k的增大先上升后降低，但由于纵坐标的差值精确度很高，实际上准确率的差值也在1.5%以内。（由于k-fold方法的随机性，可知实验结果正确，且效果非常好）
  - 且C4.5和ID3算法两者的准确率误差较小。

- **CART算法：**

```
利用CART方法，对数据集进行3折划分后的准确率为0.9508101851851851
利用CART方法，对数据集进行4折划分后的准确率为0.9641203703703703
利用CART方法，对数据集进行5折划分后的准确率为0.9663768115942029
利用CART方法，对数据集进行6折划分后的准确率为0.9652777777777778
利用CART方法，对数据集进行7折划分后的准确率为0.9680603948896632
利用CART方法，对数据集进行8折划分后的准确率为0.9646990740740741
利用CART方法，对数据集进行9折划分后的准确率为0.9629629629629629
利用CART方法，对数据集进行10折划分后的准确率为0.9662790697674419
利用CART方法，对数据集进行11折划分后的准确率为0.9716270990156339
利用CART方法，对数据集进行12折划分后的准确率为0.970486111111111
利用CART方法，对数据集进行13折划分后的准确率为0.9708624708624709
利用CART方法，对数据集进行14折划分后的准确率为0.970383275261324
利用CART方法，对数据集进行15折划分后的准确率为0.9715942028985507
利用CART方法，对数据集进行16折划分后的准确率为0.9687500000000001
利用CART方法，对数据集进行17折划分后的准确率为0.9708794408852651
利用CART方法，对数据集进行18折划分后的准确率为0.9670138888888888
利用CART方法，对数据集进行19折划分后的准确率为0.9666666666666667
```

- 

  - 根据输出结果和折线图可知，利用CART算法的GINI指数进行特征选择时，准确率整体随着k的增大先上升后降低（且大概可看出两个平台值），但由于纵坐标的差值精确度很高，实际上准确率的差值也在2%以内。（由于k-fold方法的随机性，可知实验结果正确，且效果非常好）
  - 可以看出，CART算法的准确率比前两者的准确率要稍微偏低一点，大概低0.5%左右。

- **模型性能展示和分析**

  以下比较基于**“k-fold交叉验证法”**划分数据集得到：

	ID3	C4.5	CART	准确率	k值
初始	1	0	0	95.37%	3
优化1	1	0	0	97.28%	15
优化2	0	1	0	97.16%	15
优化3	0	0	1	97.16%	11
最优效果	1	0	0	97.28%	15

  调参的变量为k值(k-fold)和特征选择的方法（ID3\C4.5\CART）。

思考题

1. **决策树有哪些避免过拟合的方法?**

- 过拟合（overfitting）**概念**:

> 一个假设在训练数据上能够获得比其他假设更好的拟合,但是在训练数据外的数据集上却不能很好的拟合数据.此时我们就叫这个假设出现了overfitting的现象.

- **产生原因**：
  - 原因1：样本问题
    - 样本里的噪音数据干扰过大，大到模型过分记住了噪音特征，反而忽略了真实的输入输出间的关系；
    - 样本抽取错误，包括（但不限于）样本数量太少，抽样方法错误，抽样时没有足够正确考虑业务场景或业务特点，等等导致抽出的样本数据不能有效足够代表业务逻辑或业务场景；
    - 建模时使用了样本中太多无关的输入变量。
  - 原因2： 构建决策树的方法问题
    - 在决策树模型搭建中，我们使用的算法对于决策树的生长没有合理的**限制**和**修剪**的话，决策树的自由生长有可能每片叶子里**只包含单纯的事件数据或非事件数据**，可以想象，这种决策树当然可以完美匹配（拟合）训练数据，但是一旦应用到新的业务真实数据时，效果是一塌糊涂。
- **解决方法：**
  - 针对原因1： 合理、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；
  - 针对原因2： 剪枝：提前停止树的增长或者对已经生成的树按照一定的规则进行后剪枝。
    - a)先剪枝：在决策树生成过程中进行。对于当前的结点，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。判断方法：假设基于ID3， 选择了某个特征进行划分。 如果划分后，决策树在验证集上的准确率不提高，则无需划分。
    - b)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。后序遍历。对于某个非叶结点， 假如将它变成叶子结点， 决策树在验证集上的准确率不降低， 则将它变成叶子结点。

2. C4.5相比于ID3的优点是什么， C4.5又可能有什么缺点?

- C4.5继承了ID3算法的优点，并进行了以下的改进。所以，C4.5的**优点**如下：
  - ID3算法是通过信息增益选择属性，当某特征种的特征值很多时，信息增益会很大，也就是ID3算法在选择属性的时候会**偏向多个属性值的属性**作为分裂属性的不足；
  - ID3只能对离散型的数据进行处理，改进后的C4.5能够将连续型数据离散化
  - 能够进行剪枝操作，避免树的高度无节制的增长，避免过度拟合数据
  - 能够对空缺值进行处理
- C4.5的**缺点**：
  - 在构造树的过程中，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**。此外，C4.5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。

3. 如何用决策树来进行特征选择（判断特征的重要性）？

- 特征选择的方法有本次实验中的3种：ID3、C4.5、CART算法
- 除此之外，还有**“随机森林”**：
  - 随机森林由多个决策树构成。决策树中的每一个节点都是关于某个特征的条件，为的是将数据集按照不同的响应变量一分为二。
  - 从直观角度来解释，每棵决策树都是一个分类器（假设现在针对的是分类问题），那么对于一个输入样本，N棵树会有N个分类结果。而随机森林集成了所有的分类投票结果，将投票次数最多的类别指定为最终的输出，这就是一种最简单的 Bagging 思想。
  - 随机森林在训练之后可以产生一个各个特征重要性的数据集，利用这个数据集，确定一个阈值，选出来对模型训练帮助最大的一些特征，筛选出重要变量后可以再训练模型。