中山大學 SUN YAT-SEN UNIVERSITY **实验报告**

学院：数据科学与计算机学院　　专业：计算机科学与技术　　科目：人工智能
学号：17341213　　姓名：郑康泽

决策树

一． 算法原理

1. 决策树主要是用来分类的一种方法，它利用训练数据建立一棵树，然后将这棵树应用到测试数据中，来预测测试数据的类别。
2. 建树的过程如下：
   1）遍历所有特征并选择一个特征作为结点（如果是第一次就是根结点）；
   2）根据该特征的值将该结点的数据集分为若干个子数据集；
   3）为每个子数据集创建子结点，并删去选中的特征；
   4）对每个子结点，回到第一步，直到递归边界；
   递归边界以及相关操作如下：
   1）当前结点数据集全部属于同一类别，则把该结点标为该类别的叶结点；
   2）当前结点的特征集为空，或者数据集在特征集上取值相同，此时无法划分数据集，则将当前结点标为类别为多数数据集属于的类别的叶结点；
   3）当前数据集为空，则将当前结点标为父结点多数数据集属于的类别的叶结点；
3. 预测的时候，只需要根据树的结点表示的特征与测试集的特征比对，直至走到叶结点，就得到预测的类别。
4. 特征选取有三种方法，如下：
   设数据集为 D，特征为 A
   1） 选择信息增益最大的特征为决策点：

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

$$= -\sum_{d\in D} p(d) \times \log p(d) + \sum_{a\in A} p(a) \times log\, p(a)$$

   2） 选择信息增益率最大的特征为决策点：

$$gRatio(D,A) = \frac{H(D) - H(D|A)}{SplitInfo(D,A)},$$

$$其中，SplitInfo(D,A) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times log\, \frac{|D_j|}{|D|}$$

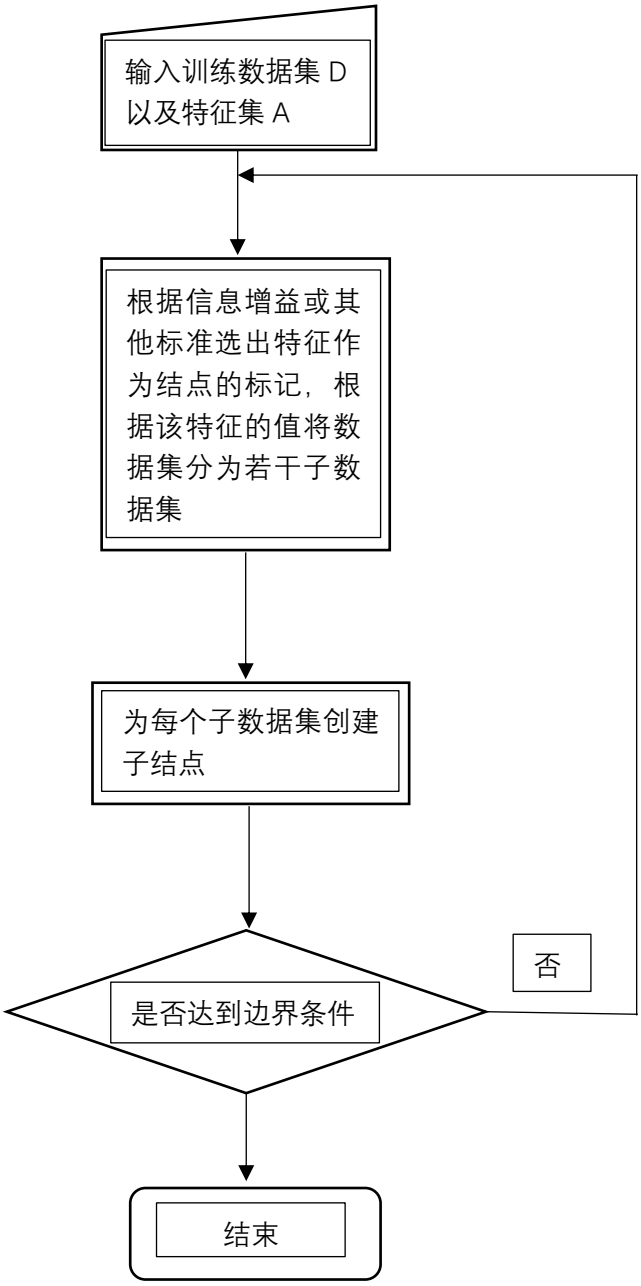
   3） 选择基尼系数最小的特征为决策点：

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

其中，$gini(D_j|A = A_j) = \sum_{i=1}^{n} p_i(1-p_i)$

二． 流程图

建立决策树的流程图



三. 代码解释

1. 将数据集随机划分为训练集和测试集，其中 3/4 为训练集，1/4 为测试集：

```
def division(read_path, train_path, test_path):
    '''
    :param read_path: 数据集路径
    :param train_path: 训练集路径
    :param test_path: 测试集路径
    :return: 无
    将数据集划分为训练集和测试集
    '''
    train_data = []
    test_data = []

    df = pd.read_csv(read_path)                 # 读文件
    columns = df.columns.values                 # 获取属性名
    df = np.array(df)
    np.random.shuffle(df)                       # 打乱顺序
    num = len(df)                               # 数据集大小
    for i in range(int(num / 4)):               # 1/4为测试集
        test_data.append(df[i])
    for i in range(int(num / 4), num):          # 3/4为训练集
        train_data.append(df[i])
```

```
    # 写入文件
    train_data = pd.DataFrame(train_data, columns=columns)
    test_data = pd.DataFrame(test_data, columns=columns)
    train_data.to_csv(train_path, index=0, line_terminator='\n')
    test_data.to_csv(test_path, index=0, line_terminator='\n')
```

2. 读出 csv 文件中的数据集和特征集：

```
def get_data(path):
    '''
    :param path: 要读的文件的路径
    :return: 数据集和特征集
    读出文件中的数据集以及特征集
    '''
    train_data = pd.read_csv(path)                  # 读文件
    features = list(train_data.columns.values)      # 特征
    train_data = np.array(train_data)               # 数据集
    return train_data, features
```

3. 计算 feature 对应的信息增益、信息增益率、基尼系数：

   1）首先要算出数据集的经验熵，只需要记录出现的标签种类以及出现的次数即可算出；然后算出数据集在特征 feature 上的条件熵，需要算出在每种 feature 取值上的数据集的熵，然后乘上一个权重，然后全部加起来，权重是该取值数量占整个数据集数量的比重。这两个算出来后，利用经验熵减去条件熵就是信息增益。

```
def cal_IG(data, features, feature):
    '''
    :param data: 数据集
    :param features: 特征集
    :param feature: 要计算信息增益的特征
    :return: 该特征的信息增益
    计算feature在data中的信息增益
    '''
    dic = {}                                    # 记录（特征值，标签）的次数
    feature_val = {}                            # 记录出现的特征值以及出现的次数
    label_val = []                              # 记录出现的标签
    num = len(data)
    IG = 0.0
    index = features.index(feature)             # 第index列是对应的特征值
    # 计算条件熵
    for row in data:
        tup = (row[index], row[-1])              # (特征值，标签)
```

```
        if tup[0] not in feature_val.keys():            # [特征值：特征值出现的次数]
            feature_val[tup[0]] = 1
        else:
            feature_val[tup[0]] += 1

        if tup[1] not in label_val:              # 标签值
            label_val.append(tup[1])

        if tup not in dic.keys():               # (特征值，标签值)：[特征值次数，对应标签次数]
            dic[tup] = [1, 1]
        else:
            dic[tup][0] += 1
            dic[tup][1] += 1
    for a, b in feature_val.items():            # a是特征值，b是出现的次数
        sum = 0.0
        for c in label_val:                     # c是标签
            if (a, c) in dic.keys():
                tmp = float(dic[(a, c)][1]) / float(b)
                if tmp != 0:
                    sum -= tmp * math.log(tmp)
        IG += float(b) / float(num) * float(sum)
```

```
    # 经验熵
    num_of_0_1 = [0.0, 0.0]
    for row in data:
        if row[-1] == 1:
            num_of_0_1[1] += 1                  # 1标签出现的次数
        elif row[-1] == 0:
            num_of_0_1[0] += 1                  # 0标签出现的次数
    num_of_0_1[0] /= float(num)
    num_of_0_1[1] /= float(num)
    IG = -IG
    for digit in num_of_0_1:
        if digit != 0:
            IG -= digit * math.log(digit)
    return IG
```

2）信息增益率是在信息增益的基础上，加上了对分类数量的惩罚，如果分类数量多，惩罚就越大，反之则越小。信息增益率的公式是信息增益除以特征的熵。

```
def cal_GR(data, features, feature):
    '''
    :param data: 数据集
    :param features: 特征集
    :param feature: 要计算信息增益率的特征
    :return: 该特征的信息增益率
    计算feature在data上的信息增益率
    '''
    IG = cal_IG(data, features, feature)                    # 获得信息增益
    feature_val = list(Attribute_VALUE[feature])            # 该特征所有的取值
    index = features.index(feature)                         # 该特征所在列
    num = len(data)                                         # 数据集的数量
    num_feature_val = [0] * len(feature_val)                # 该特征值出现的次数
    splitinfo = 0.0
```

```
    # 记录出现的次数
    for row in data:
        num_feature_val[feature_val.index(row[index])] += 1
    # 特征的熵
    for i in range(len(feature_val)):
        t = float(num_feature_val[i]) / float(num)
        if t != 0:
            splitinfo -= t * math.log(t)
    return IG / (splitinfo + 0.1)                           # 会出现除以0的情况
```

3）计算 gini 指数和计算条件熵的方法，要计算数据集在特征的各个特征值上的熵，然后乘上权重再加起来，权重是特征值数量占数据集数量的比重。

```
def cal_GN(data, features, feature):
    '''
    :param data: 数据集
    :param features: 特征集
    :param feature: 要计算gini指数的特征
    :return: 该特征的gini指数
    计算feature在data上的gini指数
    '''
    dic = {}                                                # 记录（特征值，标签）的次数
    feature_val = {}                                        # 记录出现的特征值以及出现的次数
    label_val = []                                          # 记录出现的标签值
    num = len(data)                                         # 数据集的数量
    GN = 0.0
    index = features.index(feature)                         # 该特征值在第index列
```

```
    # gini系数
    for row in data:
        tup = (row[index], row[-1])                         # (特征值，标签)

        if tup[0] not in feature_val:                       # [特征值：特征值出现的次数]
            feature_val[tup[0]] = 1
        else:
            feature_val[tup[0]] += 1

        if tup[1] not in label_val:                         # 标签值
            label_val.append(tup[1])

        if tup not in dic.keys():                           # (特征值，标签值)：[特征值次数，对应标签次数]
            dic[tup] = [1, 1]
        else:
            dic[tup][0] += 1
            dic[tup][1] += 1
```

```
    for a, b in feature_val.items():
        sum = 1.0
        for c in label_val:
            if (a, c) in dic.keys():
                tmp = float(dic[(a, c)][1]) / float(b)
                sum -= tmp**2
        GN += float(b) / float(num) * float(sum)
    return GN
```

4. 根据信息增益、信息增益率、gini 指数的选择函数：

1）信息增益是选择最大的，因为数据集的熵是一个定值，如果条件熵越小，说明在这个特征下，数据集的不确定性小，那么这个特征可以更好将数据集分类。

```
def choose_vertex_v1(data, features):
    '''
    :param data: 数据集
    :param features: 特征集
    :return: 适合做结点的特征
    for information gain
    '''
    IGs = []
    for feature in features[:-1]:
        IGs.append(cal_IG(data, features, feature))
    vertex = np.argsort(-np.array(IGs))[0]
    vertex = features[vertex]
    return vertex
```

2）信息增益率是信息增益除以特征的熵，如果信息增益大，特征的熵小，说明该特征可以通过更少的结点获得更好的分类效果，所以信息增益率越大越好。

```
def choose_vertex_v1(data, features):
    '''
    :param data: 数据集
    :param features: 特征集
    :return: 适合做结点的特征
    for information gain
    '''
    IGs = []
    for feature in features[:-1]:
        IGs.append(cal_IG(data, features, feature))
    vertex = np.argsort(-np.array(IGs))[0]
    vertex = features[vertex]
    return vertex
```

3）根据 gini 指数的定义，我们选择最小的 gini 指数的特征作为结点：

```python
def choose_vertex_v3(data, features):
    '''
    :param data: 数据集
    :param features: 特征集
    :return: 适合做结点的特征
    for gini
    '''
    GNs = []
    for feature in features[:-1]:
        GNs.append(cal_GN(data, features, feature))
    vertex = np.argsort(np.array(GNs))[0]
    vertex = features[vertex]
    return vertex
```

5. 根据输入的特征划分数据集，以及从该结点的特征集中删去该特征：

```python
def split(data, features, feature):
    '''
    :param data: 数据集
    :param features: 特征集
    :param feature: 根据此特征划分数据集
    :return: 根据feature的值划分得到的若干子数据集，feature的所有值，删去feature后的特征集
    '''
    _features = copy.deepcopy(features)                         # 避免影响其他同一层的特征集
    index = _features.index(feature)                            # 该特征所在的列
    feature_val = list(Attribute_VALUE[feature])                # 该特征所有的取值
    split_data = [[] for i in range(len(feature_val))]          # 分裂的子数据集
    for row in data:
        split_data[feature_val.index(row[index])].append(row)
    # 删去feature对应的列数据
    for i in range(len(feature_val)):
        if len(split_data[i]) != 0:
            split_data[i] = np.delete(split_data[i], index, axis=1)
            split_data[i] = np.array(split_data[i])
    _features.remove(feature)                                   # 从特征集删去该特征
    return np.array(split_data), feature_val, _features
```

6. 递归建树：

先选择一个特征作为树的结点，然后根据这个特征的值来划分数据集，并从特征集中删去该特征。之后，依次检查子数据集是否满足递归边界，如果满足，则按照上述原理操作，否则继续递归建树。

```python
def create_tree(data, features, choice):
    '''
    :param data: 数据集
    :param features: 特征集
    :param choice: 选择信息增益、信息增益率还是gini指数来选择结点
    :return: 树的根结点
    '''
    tree = {}                                   # 特征：子字典
    sub_tree = {}                               # 特征值：子字典
    if choice == 1:                             # 信息增益作为标准
        vertex = choose_vertex_v1(data, features)
    elif choice == 2:                           # 信息增益率作为标准
        vertex = choose_vertex_v2(data, features)
    else:                                       # gini指数作为标准
        vertex = choose_vertex_v3(data, features)
    split_data, feature_val, features = split(data, features, vertex)       # 划分数据集并从当前特征集中删去特征
```

```
    for i in range(len(feature_val)):
        if len(split_data[i]) == 0:                 # 子数据集为空，递归停止
            res = majority(data[:, -1])             # 选择父结点的数据集的多数标签作为叶结点的标签
            sub_tree[feature_val[i]] = res
        elif check_1(split_data[i]):                # 子数据集的标签都一样，递归停止
            sub_tree[feature_val[i]] = split_data[i][0][-1]     # 选择该标签作为叶结点的标签
        elif check_2(split_data[i], features):      # 子数据集在特征集上的取值都一样，递归停止
            res = majority(split_data[i][:, -1])    # 选择多数标签作为叶结点的标签
            sub_tree[feature_val[i]] = res
        else:                                       # 递归建树
            sub_tree[feature_val[i]] = create_tree(split_data[i], features, choice)

    tree[vertex] = sub_tree
    return tree
```

7. 预测以及计算准确率：
   根据建树的办法可以很容易写出预测的过程。

```
def predict(data, features, decision_tree):
    '''
    :param data: 数据集
    :param features: 特征集
    :param decision_tree: 决策树
    :return: 在测试集上的准确率
    '''
    total_num = len(data)                               # 数据集的数量
    correct_num = 0                                     # 预测正确的数量

    for row in data:
        tree = decision_tree
        while type(tree) != int:                        # 到达叶结点
            feature = list(tree.keys())[0]
            vertex = row[features.index(feature)]       # 在该特征上的取值
            tree = tree[feature]
            tree = tree[vertex]                         # 子树
        if tree == row[-1]:                             # 预测正确
            correct_num += 1

    return float(correct_num) / float(total_num)
```

8. 一些辅助函数：
   1）检查数据集的标签是否都一样：

```
def check_1(data):
    '''
    :param data: 数据集
    :return: 标签全部一样返回True，否则返回False
    '''
    last = data[0][-1]
    for row in data[1:]:
        if last != row[-1]:
            return False
    return True
```

   2）检查数据集在特征集上的取值是否都一样：

```python
def check_2(data, features):
    '''
    :param data: 数据集
    :param features: 特征集
    :return: 数据集在特征集上的取值都一样返回True，否则返回False
    '''
    num = len(features)
    for i in range(len(features[: -1])):
        last = data[0][i]
        for row in data[1:]:
            if row[i] != last:
                return False
    return True
```

3）选择多数标签：

```python
def majority(column):
    '''
    :param column: 数据集的标签
    :return: 多数标签的种类
    '''
    num = [0, 0]
    for each in column:
        num[each] += 1
    if num[0] > num[1]:
        return 0
    return 1
```

四． 实验结果以及分析

1. 结果展示和分析：

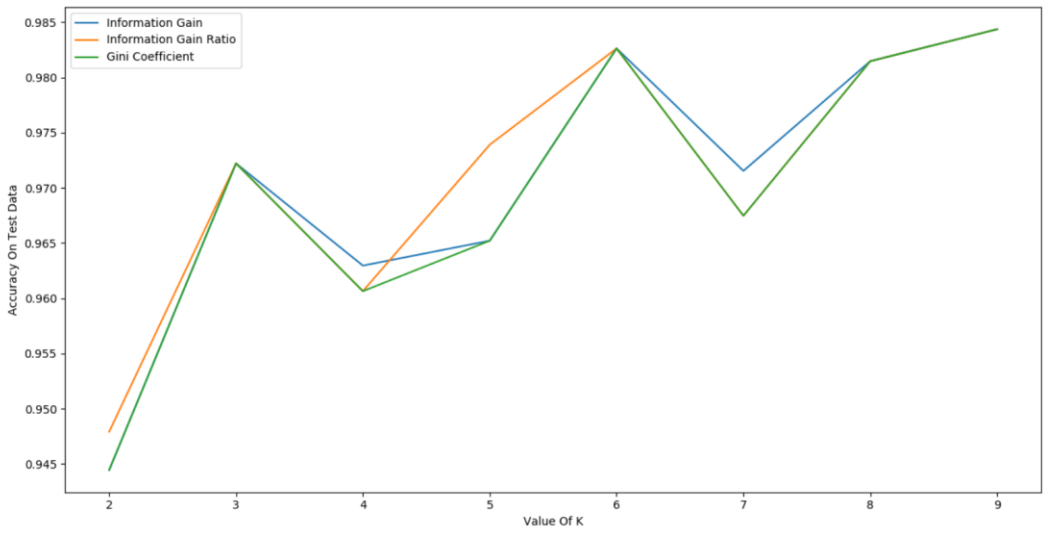
```
选择信息增益作为标准
真正的标签0所占的比例： 0.7222222222222222
真正的标签1所占的比例： 0.2777777777777778
预测的标签1所占的比例： 0.7222222222222222
预测的标签2所占的比例： 0.2777777777777778
准确率： 0.9722222222222222

选择信息增益率作为标准
真正的标签0所占的比例： 0.7222222222222222
真正的标签1所占的比例： 0.2777777777777778
预测的标签1所占的比例： 0.7106481481481481
预测的标签2所占的比例： 0.28935185185185186
准确率： 0.9837962962962963

选择gini指数作为标准
真正的标签0所占的比例： 0.7222222222222222
真正的标签1所占的比例： 0.2777777777777778
预测的标签1所占的比例： 0.7175925925925926
预测的标签2所占的比例： 0.2824074074074074
准确率： 0.9722222222222222
```

根据上面的结果显示，决策树对于二分类问题还是有很高的准确率的，并且可以看到三种模型的准确率十分相近，看不出哪种模型是对哪种模型的改进，毕竟只有遇到极端情况才能看出；
观察三种模型的预测的标签的分布，都是不一样的，说明建出来的树都是不一样的，但与真实的标签的分布都很靠近，这也能猜测出准确率应该不低。
观察到模型 1 预测的标签的分布与真实的标签的分布完全一致，但准确率却没有模型 2 的准确率高，尽管模型 2 预测的标签的分布与真实的标签的分布还是有点区别，这说明预测的标签的分布与真实的标签的分布的差距与准确率没有必然的关系，但还是有参考的意义。

2. 模型性能展示和分析：



K 值是将原始数据集划分为 K 份，(K-1)/K 为训练集，1/K 为测试集，由图可见，随着 K 值的增大，即训练集增加，测试集减少，准确率曲折式上升，而且三条曲线经常重叠在一起，说明三个模型的准确率基本一致。

五. 思考题

1. 决策树有哪些避免过拟合的方法？
预剪枝、后剪枝、随机森林等方法都可以避免过拟合。
剪枝通过删除一些结点来避免过拟合；
随机森林通过多次随机选择一部分样本以及一部分特征建多棵树来避免过拟合；
融入模型复杂度通过计算加入对结点个数的惩罚的泛化错误率来选择。
2. C4.5 相比于 ID3 的优点是什么，C4.5 又有什么缺点？
优点是考虑了特征的值的个数，对具有多的值的特征进行了惩罚计算；
缺点是比 ID3 计算更加复杂，更加耗时，无法处理非常大的数据。
3. 如何用决策树来进行特征选择（判断特征的重要性）？
距离根结点的距离越近，特征的重要性就越强，距离越远，特征的重要性就越弱。