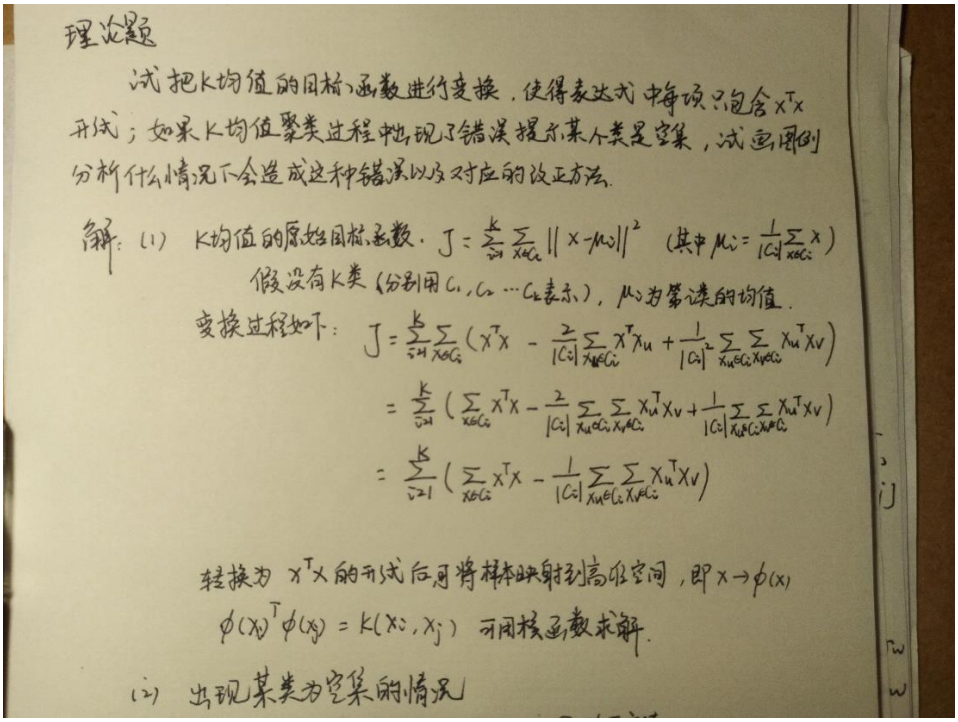
Homework3's Answer

第三次作业

一、理论题

题目：试把 k 均值的目标函数进行变换，使得表达式中每项只包含 x^Tx 形式；如果 k 均值聚类过程中出现了错误提示某个类是空集，试画图例分析什么情况下会造成这种错误以及对应的改正方法。

解答：



理论题

试把K均值的目标函数进行变换，使得表达式中每项只包含 x^Tx 形式；如果K均值聚类过程中出现了错误提示某个类是空集，试画图例分析什么情况下会造成这种错误以及对应的改正方法.

解：(1) K均值的原始目标函数 $J=\sum_{i=1}^{k}\sum_{x\in C_i}\|x-\mu_i\|^2$ (其中 $\mu_i=\frac{1}{|C_i|}\sum_{x\in C_i}x$)

假设有K类(分别用 $C_1, C_2 \cdots C_k$ 表示)，μ_i 为第i类的均值.

变换过程如下：

$$J=\sum_{i=1}^{k}\sum_{x\in C_i}\left(x^Tx-\frac{2}{|C_i|}\sum_{x_u\in C_i}x^Tx_u+\frac{1}{|C_i|^2}\sum_{x_u\in C_i}\sum_{x_v\in C_i}x_u^Tx_v\right)$$

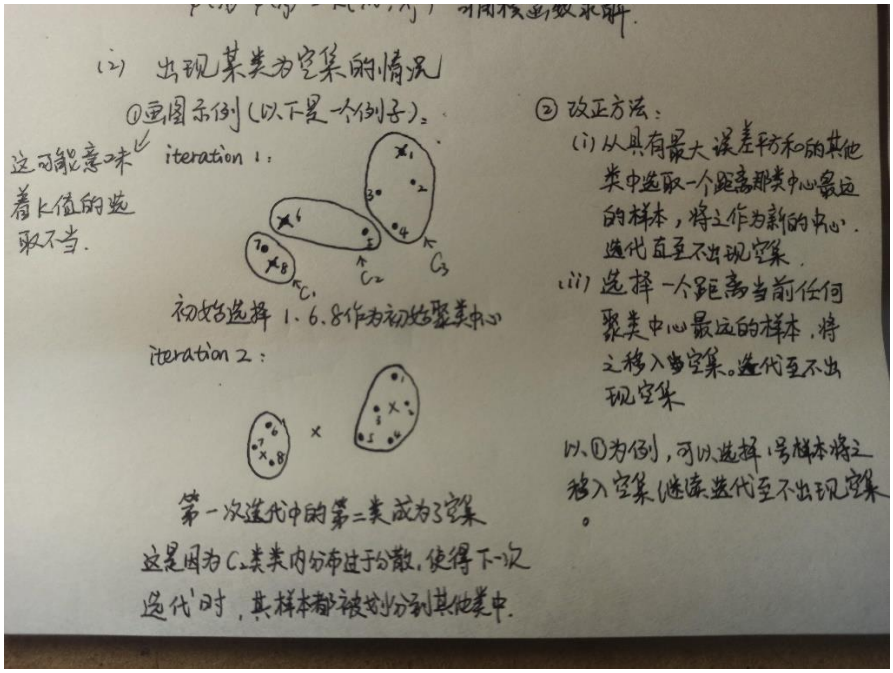
$$=\sum_{i=1}^{k}\left(\sum_{x\in C_i}x^Tx-\frac{2}{|C_i|}\sum_{x_u\in C_i}\sum_{x_v\in C_i}x_u^Tx_v+\frac{1}{|C_i|}\sum_{x_u\in C_i}\sum_{x_v\in C_i}x_u^Tx_v\right)$$

$$=\sum_{i=1}^{k}\left(\sum_{x\in C_i}x^Tx-\frac{1}{|C_i|}\sum_{x_u\in C_i}\sum_{x_v\in C_i}x_u^Tx_v\right)$$

转换为 x^Tx 的形式后可将样本映射到高维空间，即 $x\to\phi(x)$

$\phi(x_i)^T\phi(x_j)=k(x_i,x_j)$ 可用核函数求解.

(2) 出现某类为空集的情况



二、实践题

1、编程实现 MDS“求解方法二”的算法，并分别对 MNIST12 数据集做二维和三维降维。

解：

（对应的代码文件夹为“1 MDS”）

读取数据：读入数据后，筛选类别标记为 1 和 2 的样本，打乱样本。为了方便进行降维，对数据进行归一化，该部分代码如下图所示。

```
# 加载入数据
def load():
    data = pd.read_csv(path+'/mnist_train.csv',names=range(785))
    data1 = data[data[0] == 1]
    data2 = data[data[0] == 2]
    data12 = pd.concat([data1,data2],axis=0)
    data12 = data12.sample(frac=1).reset_index(drop=True)
    sample = data12.iloc[:,1:]/255.0
    label = data12.iloc[:,0]
    sample = np.array(sample)
    label = np.array(label)

    return sample,label
```

计算距离矩阵：根据 PPT 公式，MDS 降维需要先计算距离矩阵 D^2,计算距离矩阵代码如下图所示。

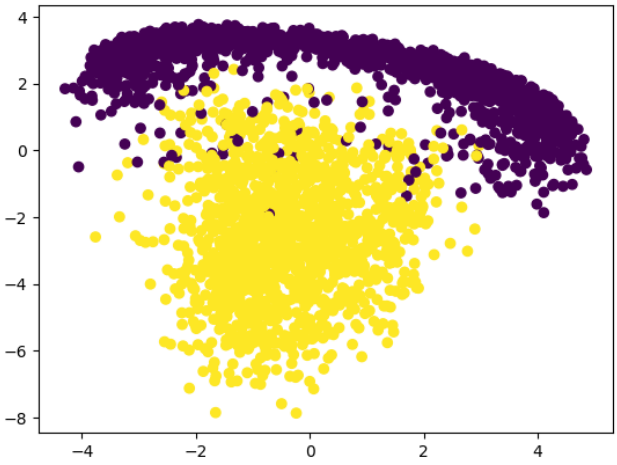
```
def calculate_dis(sample):
    n, m = sample.shape
    Dij = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            Dij[i][j] = np.sum(np.square(sample[i,:] - sample[j,:]))
    return Dij
```

MDS 主函数：在主函数中，通过调用 calculate_dis 函数计算得到 D 矩阵，然后根据公式计算 Di_* 、Dj_* 、D_{**} 矩阵，由此计算 B 矩阵。最后通过特征值分解，并选取最大的 m 个特征值和对应特征向量，可以计算得到降维后的矩阵。即将样本点从高维嵌入到了低维当中。

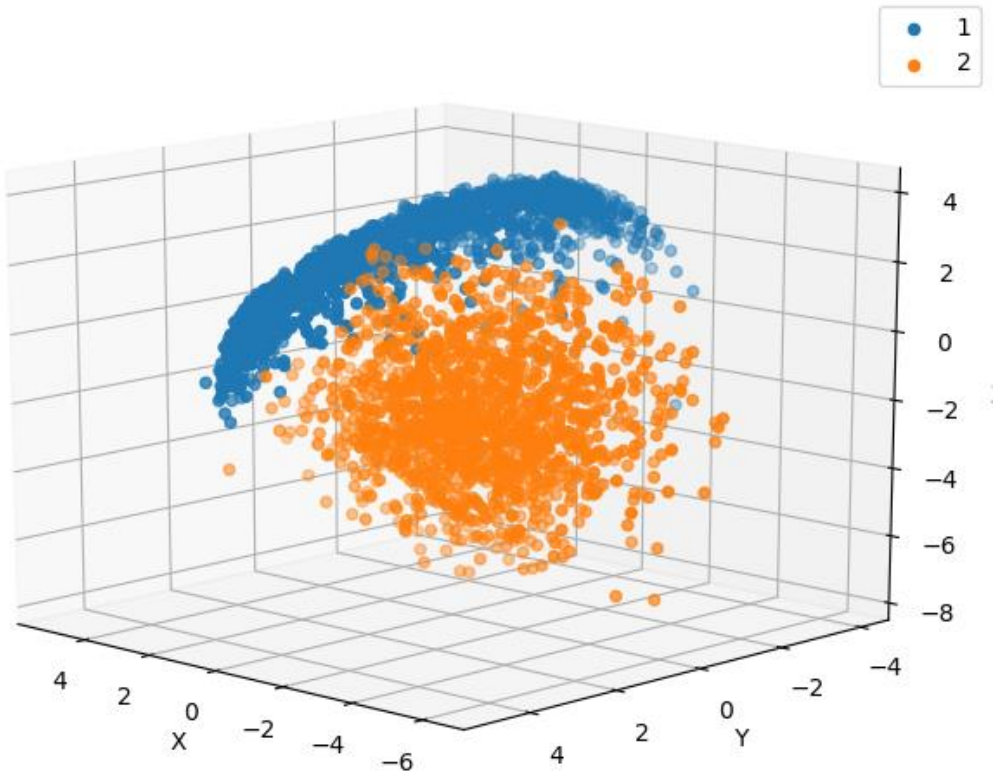
```
def MDS(sample,dim):
    n, m = sample.shape
    Dij = calculate_dis(sample)

    Di = np.sum(Dij,axis=1,keepdims=True)
    Dj = np.sum(Dij,axis=0,keepdims=True)
    D = np.ones((n,n)) * np.sum(Dij)

    Bij = 1/2 * (Di/n + Dj/n - Dij - D/n**2)

    eval , evec = np.linalg.eig(Bij)
    ind = np.argsort(eval)
    ind = ind.astype(np.int32)
    ind_select = ind[-dim:]
    V = evec[:,ind_select]
    A_ = eval[ind_select].real
    A = np.diag(A_)
    Output = np.dot(V,A**(0.5))
    return Output
```

结果：运行程序，因为全样本特征值分解较为费时间，在此随机选取了 3000+个样本进行降维，因为 3000+个样本的低维嵌入基本可以代表全样本的空间分布了。二维和三维的嵌入分布如下图所示：

（1）嵌入到二维（紫色样本为标签为 1 的样本，黄色样本为标签为 2 的样本）：



（2）嵌入到三维（蓝色样本为标签为 1 的样本，橙色样本为标签为 2 的样本）：



（3）对比与结论：可以看出对于 MDS 降维，无论是降到二维还是三维，

基本都可以实现将标签为“1”的样本和标签为“2”的样本分离，并且二维嵌入的空间分布与三维嵌入的空间分布非常类似。

这说明了 MDS 可以较好地解决 mnist 样本降维的问题，而降维后更容易对样本进行下一步的分析。

2、实现自顶向下或自下往上层级聚类，分别使用平均欧氏距离和 Ncut 值作为两类 Ci 和 Cj 的距离/相似性度量，并对西瓜 3.0 做聚类分析。

解：

（对应的代码文件夹为“2 Cluster”）

预处理：在计算样本之间的距离的时候，对于离散变量，采用了 onehot 编码方式，使得对于某离散变量的不同取值，它们之间的距离是一样的。因此，在计算距离（相似度）之前，现将样本转换成为 19 维的样本。

计算最近的两个簇：得到 M 矩阵后，将当前所有簇的列表和 M 矩阵传入 find_closest 函数，可以返回得到当前所有簇中距离最近的两个簇对应的索引。判断最近时，必须满足正在判断的两个簇不是同一个簇。该部分代码如下图所示。

```python
def find_closest(C,M):
    min_dist = 100000
    i_ = j_ = -1
    m = len(C)
    for i in range(m):
        for j in range(m):
            if M[i,j] < min_dist and i != j:
                min_dist = M[i,j]
                i_ = i
                j_ = j

    return i_ , j_
```

主函数：采用了自下向上的层次聚类方法，在主函数中，通过计算距离的函数（在下面讲到），可以构造出 M 矩阵，层次聚类的过程就是不断选择当前所有簇中最近（M 矩阵对应值最小）的两个簇，将它们合并，然后在 M 矩阵中删去被合并的簇，并重新计算其他簇到合并后的簇之间的距离。如此重复，直到满足要求簇的个数。迭代过程的代码如下图所示。

```python
while q > k :
    print('For ',q,'\'s cluster, the answer: ',)
    for ci in C:
        print(ci)
    i_, j_ = find_closest(C,M)
    C[i_].extend(C[j_])
    C.remove(C[j_])
    M = np.delete(M,j_,axis=0)
    M = np.delete(M,j_,axis=1)
    for j in range(q-1):
        if dist_func == 'ave_elu':
            M[i_,j] = dist_ave(C[i_],C[j],feature)
        if dist_func == 'Ncut':
            M[i_,j] = dist_ncut(C[i_],C[j],feature)
        M[j,i_] = M[i_,j]
    q -= 1
```

（1）采用平均欧式距离

①平均欧式距离计算：在计算平均欧式距离时，首先对两个簇每两个样本计算其欧式距离，然后对其进行求和，求和后除以每个簇的样本个数，就

可以计算得两个簇之间的平均欧式距离。该部分代码如下图所示。

```
def dist_ave(C1,C2,feature):
    dist = 0
    for i in C1:
        for j in C2:
            dist += np.sqrt(np.sum(np.square(feature[i]-feature[j])))
    dist = dist / (len(C1)*len(C2))
    return dist
```

②结果：采用平均欧式距离，自下向上层级聚类的分步结果和最后结果如下图所示：

0 3 1 2 4 8 16 12 13 5 14 7 6													9 10 11 15		
0 3 1 2 4 8 16 12 13 5 14 7 6													9	10 11 15	
0 3 1 2 4 8 16 12 13								5 14 7 6					9	10 11 15	
0 3 1 2 4				8 16 12 13				5 14 7 6					9	10 11 15	
0 3 1 2 4				8 16		12 13		5 14 7 6					9	10 11 15	
0 3 1 2 4				8 16		12 13		5 14 7 6					9	10	11 15
0 3 1 2 4				8	16	12 13		5 14 7 6					9	10	11 15
0 3 1 2 4				8	16	12	13	5 14 7 6					9	10	11 15
0 3 1 2 4				8	16	12	13	5 14 7		6			9	10	11 15
0 3 1			2 4	8	16	12	13	5 14 7		6			9	10	11 15
0 3		1	2 4	8	16	12	13	5 14 7		6			9	10	11 15
0 3		1	2 4	8	16	12	13	5 14	7	6			9	10	11 15
0 3		1	2 4	8	16	12	13	5 14	7	6			9	10	11	15
0	3	1	2 4	8	16	12	13	5 14	7	6			9	10	11	15
0	3	1	2 4	8	16	12	13	5	14	7	6		9	10	11	15
0	3	1	2	4	8	16	12	13	5	14	7	6	9	10	11	15

```
If using average Euclidean distance, the final clustering result is :
For 2's cluster, the answer:
[0, 3, 1, 2, 4, 8, 16, 12, 13, 5, 14, 7, 6]
[9, 10, 11, 15]
```

（2）采用 Ncut 值

①Ncut 值计算：在计算 Ncut 值时，运用公式 Ncut = CUT/VOL1 + CUT/VOL2 进行计算。为此，首先需要计算 W 矩阵、D 矩阵和 y1、y2 矩阵。（其中 y1、y2 分别是两簇的指示向量。）在计算 W 矩阵时，仅仅对讨论的两个簇（C1、C2）进行计算 $w_{ij} = \exp\left(-\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2 / 2\sigma^2\right)$，这是因为在对于两个选取的簇讨论类内紧实性的时候，忽视其他簇对于它们的影响会比较合适。该部分代码如下图所示。（在提交的代码中，我仍保留了注释掉的用全图构建的 W 矩阵计算 Ncut 值的函数，这或许也是一种可行的方法。）

```python
def dist_ncut(C1,C2,feature):
    dist = 0
    sigma = 1.0

    n , m = feature.shape
    W = np.zeros((n,n))
    C = []
    C.extend(C1)
    C.extend(C2)
    for i in C:
        for j in C:
            W[i][j] = np.exp(-(np.sum(np.square(feature[i]-feature[j])))/(2*sigma**2))
    y1 = np.zeros((n,1))
    y2 = np.zeros((n,1))
    for i in C1:
        y1[i] = 1
    for j in C2:
        y2[j] = 1
    D = np.diag(np.sum(W,1))

    CUT = np.dot(np.dot(y1.T,W),y2)
    VOL1 = np.dot(np.dot(y1.T,D),y1)
    VOL2 = np.dot(np.dot(y2.T,D),y2)

    if len(C1) == 1:
        VOL1 = 1
    if len(C2) == 1:
        VOL2 = 1
    dist = CUT/VOL1 + CUT/VOL2
    return dist
```

②结果：采用 Ncut 值，自下向上层级聚类的分步结果和最后结果如下图所示：

<table>
<tr><td colspan="7">0 13 7 3 6 5 15</td><td colspan="10">1 11 12 4 8 9 2 10 14 16</td></tr>
<tr><td colspan="7">0 13 7 3 6 5 15</td><td colspan="6">1 11 12 4 8 9</td><td colspan="4">2 10 14 16</td></tr>
<tr><td colspan="3">0 13 7</td><td colspan="4">3 6 5 15</td><td colspan="6">1 11 12 4 8 9</td><td colspan="4">2 10 14 16</td></tr>
<tr><td colspan="3">0 13 7</td><td colspan="4">3 6 5 15</td><td colspan="3">1 11 12</td><td colspan="3">4 8 9</td><td colspan="4">2 10 14 16</td></tr>
<tr><td colspan="3">0 13 7</td><td colspan="2">3 6</td><td colspan="2">5 15</td><td colspan="3">1 11 12</td><td colspan="3">4 8 9</td><td colspan="4">2 10 14 16</td></tr>
<tr><td colspan="2">0 13</td><td>7</td><td colspan="2">3 6</td><td colspan="2">5 15</td><td colspan="3">1 11 12</td><td colspan="3">4 8 9</td><td colspan="4">2 10 14 16</td></tr>
<tr><td colspan="2">0 13</td><td>7</td><td colspan="2">3 6</td><td colspan="2">5 15</td><td colspan="3">1 11 12</td><td colspan="3">4 8 9</td><td>2</td><td colspan="3">10 14 16</td></tr>
<tr><td colspan="2">0 13</td><td>7</td><td colspan="2">3 6</td><td colspan="2">5 15</td><td colspan="2">1 11</td><td>12</td><td colspan="3">4 8 9</td><td>2</td><td colspan="3">10 14 16</td></tr>
<tr><td>0</td><td>13</td><td>7</td><td colspan="2">3 6</td><td colspan="2">5 15</td><td colspan="2">1 11</td><td>12</td><td colspan="3">4 8 9</td><td>2</td><td colspan="3">10 14 16</td></tr>
<tr><td>0</td><td>13</td><td>7</td><td colspan="2">3 6</td><td colspan="2">5 15</td><td colspan="2">1 11</td><td>12</td><td>4</td><td colspan="2">8 9</td><td>2</td><td colspan="3">10 14 16</td></tr>
<tr><td>0</td><td>13</td><td>7</td><td colspan="2">3 6</td><td colspan="2">5 15</td><td colspan="2">1 11</td><td>12</td><td>4</td><td colspan="2">8 9</td><td>2</td><td colspan="2">10 14</td><td>16</td></tr>
<tr><td>0</td><td>13</td><td>7</td><td colspan="2">3 6</td><td>5</td><td>15</td><td colspan="2">1 11</td><td>12</td><td>4</td><td colspan="2">8 9</td><td>2</td><td colspan="2">10 14</td><td>16</td></tr>
<tr><td>0</td><td>13</td><td>7</td><td colspan="2">3 6</td><td>5</td><td>15</td><td>1</td><td>11</td><td>12</td><td>4</td><td colspan="2">8 9</td><td>2</td><td colspan="2">10 14</td><td>16</td></tr>
<tr><td>0</td><td>13</td><td>7</td><td>3</td><td>6</td><td>5</td><td>15</td><td>1</td><td>11</td><td>12</td><td>4</td><td colspan="2">8 9</td><td>2</td><td colspan="2">10 14</td><td>16</td></tr>
<tr><td>0</td><td>13</td><td>7</td><td>3</td><td>6</td><td>5</td><td>15</td><td>1</td><td>11</td><td>12</td><td>4</td><td colspan="2">8 9</td><td>2</td><td>10</td><td>14</td><td>16</td></tr>
<tr><td>0</td><td>13</td><td>7</td><td>3</td><td>6</td><td>5</td><td>15</td><td>1</td><td>11</td><td>12</td><td>4</td><td>8</td><td>9</td><td>2</td><td>10</td><td>14</td><td>16</td></tr>
</table>

```
If using NCUT distance, the final clustering result is :
For 2's cluster, the answer:
[0, 13, 7, 3, 6, 5, 15]
[1, 11, 12, 4, 8, 9, 2, 10, 14, 16]
```

结论与分析：基于平均欧式距离的层次聚类和基于 Ncut 值的层次聚类可以得到不同的结果。除此之外，层次聚类的过程也可以有助于我们的分析和运用。

三、附加题

实现基于信息增益率的决策树，并对西瓜 3.0 数据集进行 70%训练-30%测试。

解：

（对应的代码文件夹为“3 DecisionTree”）

数据获得与预处理：

与上面不同，决策树不需要计算两个样本之间的距离，所以并没有将原来的样本转化为 onehot 编码，也保留了原来的中文名称，仅仅对标签进行了 0/1 的转化。

建立决策树：

将前 70%的样本（共 11 个）作为训练集建立决策树。

建立树时，需要注意这么几个停止条件：1.D^v 中的样本标签都相同，那么直接标记为该类 2.待划分属性集为空或者所有样本的特征都相同，那么直接标记为 D^v 中标签数最多的那类 3.D^v 为空集，那么直接标记为父类（D）中标签数最多的那类。

除此之外，都先根据信息增益率选择最优划分属性，对于该属性的不

同取值进行划分，不断迭代。该部分代码如下图所示。

```
def CreateTree(feature,label,attr):
    label_tmp = list(label)
    feature_copy = feature.copy()
    if len(set(label_tmp)) == 1:
        return label_tmp[0]
    if len(attr) == 0 or len(feature_copy.drop_duplicates(subset=list(attr))) == 1:
        return MajorityCount(label)

    a_ = ChooseFeature(feature,label,attr)
    Tree = {a_:{}}
    attr.remove(a_)
    attr_dic = {'色泽':['青绿', '乌黑', '浅白'], '根蒂':['蜷缩', '稍蜷', '硬挺'], '敲声':['浊响', '沉闷', '清脆'], \
        '纹理':['清晰', '稍糊', '模糊'], '脐部':['凹陷', '稍凹', '平坦'], '触感':['硬滑', '软粘']}
    values = attr_dic[a_]

    for value in values:
        sub_feature, sub_label = GetSubData(feature,label,a_,value)
        if sub_feature.empty:
            Tree[a_][value] = MajorityCount(label)
            continue
        Tree[a_][value] = CreateTree(sub_feature,sub_label,attr)

    return Tree
```

选取最优属性：

在选取最优属性的时候，分别计算待选择属性在当前样本集中对应的信息增益率，选择信息增益率最大对应的属性作为划分属性。选取最优属性部分的代码如下图所示。

```
def ChooseFeature(feature,label,attr):
    max_Gain_ratio = -1
    best_attr = None
    for a in attr:
        Gain_ratio = CalcGainRatio(feature,label,a)
        if Gain_ratio > max_Gain_ratio:
            max_Gain_ratio = Gain_ratio
            best_attr = a
    return  best_attr
```

计算信息增益率：

在计算信息增益率的时候，按照书本的公式分别计算 Gain 和 IV，在调试过程中，发现 Gain 和 IV 有时同时为 0 导致计算出的 Gain_ratio 为 Nan。这是因为当前样本集在属性 a 上的取值都是一样的，从实际角度出发，不会选取该属性作为划分，因此将信息增益率设置为 0。该部分代码如下图所示。

```
def Ent(feature,label):
    label_list = list(label)
    ent = 0.0
    dic = {}
    for l in set(label_list):
        ct = label_list.count(l)
        dic[l] = ct
    for l in set(label_list):
        pk = float(dic[l])/float(len(label_list))
        ent -= pk * np.log2(pk)
    return ent

def CalcGainRatio(feature,label,a):
    Gain = 0.0
    IV = 0.0
    Gain += Ent(feature,label)

    values = feature.loc[:,a]
    values = set(values)
    for value in values:
        subfeature, sublabel = GetSubData(feature,label,a,value)
        pro = float(len(subfeature))/float(len(feature))
        Gain -= pro * Ent(subfeature,sublabel)
        IV -= pro * np.log2(pro)
    if IV == 0 and Gain == 0:
        Gain_ratio = 0.0
    else:
        Gain_ratio = Gain / IV
    return Gain_ratio
```

计算标签中的多数：

在停止条件时，需要计算当前样本集标签的多数为什么标签，并将决策树该结点标记为该标签。该部分的代码如下图所示。

```
def MajorityCount(label):
    label_list = list(label)
    dic = {}
    for l in set(label_list):
        ct = label_list.count(l)
        dic[l] = ct
    max_label = max(dic,key=dic.get)
    return max_label
```

分割样本集：

在迭代生成决策树时，需要分割样本集进行下一步的迭代运算。该部分的代码如下图所示。

```
def GetSubData(feature,label,a,value):
    idx = feature[a].isin([value])
    new_feature = feature[idx]
    new_label = label[idx]
    return new_feature, new_label
```

预测分类：

训练完成后返回了一棵决策树，就可以将测试集输入决策树得到其预测标签。首先将所有测试样本一一输入到决策树中，然后根据字典（决策树）的键值对不断迭代进行预测。该部分的代码如下图所示。

```
def SingleJudge(single_feature,DecisionTree):
    a = list(DecisionTree.keys())[0]
    dic = DecisionTree[a]
    value = single_feature.loc[a]
    SubTree = dic[value]
    if type(SubTree).__name__ == 'dict':
        label = SingleJudge(single_feature,SubTree)
    else:
        label = SubTree
    return label

def Judge(test_feature,DecisionTree):
    predict_label = np.zeros(len(test_feature))
    for i in range(len(test_feature)):
        single_feature = test_feature.iloc[i]
        single_label = SingleJudge(single_feature,DecisionTree)
        predict_label[i] = single_label
    return predict_label
```

结果与分析：

在打乱样本顺序后按 70%训练集，30%测试集进行测试，可以得到决策树的精度在 1/2 ～ 5/6 不等，这是因为数据集偏小的原因。除此之外，倘若考虑剪枝的话，或许也可以使精度有所提高。

```
The Decision Tree is :  {'纹理': {'清晰': {'触感': {'硬滑': 1, '软粘': 0}}, '稍糊': {'敲声': {'浊响': 1, '沉闷': 0, '清脆': 0}}, '模糊': 0}}
The accuracy is :  0.6666666666666666

The Decision Tree is :  {'脐部': {'凹陷': {'根蒂': {'蜷缩': 1, '稍蜷': 0, '硬挺': 1}}, '稍凹': {'色泽': {'青绿': 1, '乌黑': {'触感': {'硬滑':
, '软粘': {'纹理': {'清晰': 0, '稍糊': 1, '模糊': 0}}}}, '浅白': 0}}, '平坦': 0}}
The accuracy is :  0.6666666666666666

The Decision Tree is :  {'纹理': {'清晰': {'触感': {'硬滑': 1, '软粘': {'色泽': {'青绿': {'敲声': {'浊响': 1, '沉闷': 0, '清脆': 0}}, '乌黑':
, '浅白': 0}}}}, '稍糊': 0, '模糊': 0}}
The accuracy is :  0.8333333333333334

The Decision Tree is :  {'纹理': {'清晰': {'触感': {'硬滑': 1, '软粘': {'色泽': {'青绿': 1, '乌黑': 0, '浅白': 0}}}}, '稍糊': {'敲声': {'浊响'
 {'脐部': {'凹陷': 0, '稍凹': 1, '平坦': 0}}, '沉闷': 0, '清脆': 0}}, '模糊': 0}}
The accuracy is :  0.8333333333333334

The Decision Tree is :  {'纹理': {'清晰': {'触感': {'硬滑': 1, '软粘': {'色泽': {'青绿': {'敲声': {'浊响': 1, '沉闷': 0, '清脆': 0}}, '乌黑':
, '浅白': 0}}}}, '稍糊': 0, '模糊': 0}}
The accuracy is :  0.8333333333333334
```