中山大學
SUN YAT-SEN UNIVERSITY

《人工智能实验》

实验报告

（实验二）

学 院 名 称 ：	数据科学与计算机学院
专业（班级）：	17 计算机科学与技术
学 生 姓 名 ：	仲逊
学　　号	16327143
时　　间 ：	2019　年　9　月　18　日

实 验 2 ：决策树

一. 实验目的

1. 掌握如何使用决策树模型对离散特征数据集进行分类。
2. 在给定数据集分别使用 ID3，C4.5 和 CART 模型选择特征递归建立决策树，并验证其准确率。

二. 算法原理

1. 决策树

决策树是一种直观上非常容易理解的监督学习算法，也是非常符合人的决策思维的一种算法，用于对离散特征的数据集进行分类。决策树中的每一个内部节点都表示数据集中的一个特征，该节点的分支则表示该特征的不同取值，而叶子节点则表示该样本的类别。决策树训练时需要根据选择的属性，递归地划分训练样本。决策树建立完毕后，对于一个新的样本，只需从根节点开始自顶而下进入对于特征值的分支，最后走到的叶子节点就是预测的类别。

影响决策树的性能和准确度的因素主要是分裂属性的选择，直观上来说自然是树高越低，每个叶子节点的平均深度越小，分类越少，决策树的性能就越好。而另一方面，分的类越详细，准确度就越高。这二者的平衡如何把握就是一个值得思考的问题，除此以外还应考虑分类过于精细会导致过拟合泛化能力差，而分类过粗略又会导致欠拟合等等问题，所以如何选择分裂属性是决策树中一个非常重要的问题。

2. 基于信息增益的ID3模型：

ID3模型以各个属性的信息增益作为指标来衡量一个属性的分裂优先级。

首先需要清楚的是一些信息论的背景知识，在信息论中，一个离散变量X的不确定性用熵$H(X)$来衡量，数据集D的经验熵公式如下：

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

其中$p(d)$表示类标签d在数据集D中出现的概率。条件熵表示的是在一个随机变量已知的情况下另一个随机变量的熵，比如假设特征A已经确定被作为分裂属性，则特征A对数据集D的条件熵为：

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

而数据集D的熵减去特征A对D的条件熵就是信息增益，在信息论中这被称为互信息，其含义也不难理解，因为熵是衡量离散变量X的不确定性的指标，原本是熵是$H(D)$，现在确定了一个属性得到的条件熵是$H(D|A)$，那么信息增益就是确定属性A后数据集D的不确定性减少的部分，故信息增益越大，则该属性就越好。

3. 基于增益率的C4.5模型：

根据条件熵的公式，我们不难发现ID3模型的缺陷，信息增益的衡量偏向于那些属性值非常多的属性，比如如果数据中有一个属性为ID，由于各个样本的ID都不同，知道ID就可以直接得到该样本的标签，它的信息增益就是最大的，ID3模型就一定会选择它作为分裂属性，那么我们建立的决策树就可以想象：内部节点只有一个根节点ID，其他都是叶节点。这就会导致过拟合的问题。

C4.5就是为了解决这一问题，它使用的指标是信息增益率来对信息增益进行正则化，信息增益率的公式如下：

$$GainRatio_A(D) = \frac{Gain_A(D)}{SplitInfo_A(D)}$$

它的分子$GainRatio_A(D)$为信息增益，分母$SplitInfo_A(D)$为该属性的熵，分支数越多则属性熵越大，以起到减少分支数增强泛化能力的作用，它选取的最佳分裂属性是信息增益率最大的属性。

4. 基于Gini指数的CART模型：

上面两种模型的衡量属性分裂优劣的指标都是信息论中的熵相关，而CART模型使用的是经济学中常见Gini指数，Gini指数的定义如下：

$$gini(D) = \sum_{j=1}^{n} p_j(1 - p_j) = 1 - \sum_{j=1}^{n} {p_j}^2$$

其中p_j是类j在D中的相对频率，n则是数据集D包含样本的类的个数。如果一个数据集D被分成两个子集D1和D2，大小分别为N1和N2，数据包含来自n个类的样本，则基尼指数 $gini_{split}(D)$ 定义如下：

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

从公式可以看出基尼指数越小，该类的不确定性就越小，比如其中一个类占据绝大多数，那么该类的p值就很接近1，得到的基尼指数就越小，因此我们应当选择基尼指数最小的特征作为分裂属性。

三. 伪代码

- 建树伪代码：

输入:训练集$D=\{(x_1,y_1),(x_2,y_2),\dots,(x_m,y_m)\}$;

属性集$A=\{a_1,a_2,\dots,a_d\}$;

函数$buildTree(D,A)$

生成结点$node$;

if D中样本全属于同$-$类别C **then**:

将$node$标记为C类叶结点; **return**;

end if

if $A=\emptyset$ **OR** D中样本在A上取值相同 **then**:

将$node$标记为叶结点,其类别标记为D中样本数最多的类; **return**;

end if

从A中选择最优划分属性a_*;

for a_*的每一个值a_*^v **do**:

为$node$生成一一个分支;令D_v表示D中在a_*上取值为a_*^v的样本子集;

if D_v为空**then**:

将分支结点标记为叶结点,其类别标记为D中样本最多的类; return;

else:

以$buildTree(D_v,\ A\backslash\{a_*\})$为子树;

end if

end for

输出:以$node$为根结点的-棵决策树

四. 代码截图

- 计算信息熵

```
def computeEnt(labels):
    """
    input: labels 数据集的标签
    return: 数据集的信息熵
    """
    sampleNum = len(labels)
    #统计各个 label 出现次数
    labelCounts = {}
    for i in range(sampleNum):
        labelCounts[labels[i]]=labelCounts.get(labels[i],0)+1
    #计算信息熵
    entropy = 0.0
    for label in labelCounts:
        prob = labelCounts[label]/sampleNum
        entropy -= prob * np.log2(prob)
    return entropy
```

- 计算gini指数

```
def computeGini(attribute, labels):
    """
    input: attribute 数据集的一列
           label 数据集的标签
    return: 选取该属性分裂时数据集的 Gini 指数
    """
    sampleNum = len(labels)
    #统计各个属性值的出现次数
    attriCounts = {}
    for i in range(sampleNum):
        attriCounts[attribute[i]]=attriCounts.get(attribute[i],0)+1
    #计算信息熵
    gini = 0.0
    for key in attriCounts:
        labelCounts = {}
        #统计一个属性值对应的不同 label 数量
        for i in range(sampleNum):
            if attribute[i] == key:
                labelCounts[labels[i]]=labelCounts.get(labels[i],0)+1
        #该属性值对应的权重
        weight = attriCounts[key]/sampleNum
        #使用公式计算该属性的 Gini 指数
```

```
        attrGini = (1-sum ([(labelCounts[label]/ attriCounts[key])
                                **2  for label in labelCounts]))
        gini +=  weight*attrGini
    return gini
```

■ ID3模型选取最佳分裂属性（C4.5只需稍作改动，见源码）：

```
def ID3SplitFeature(train,labels):
    """
    input: train 训练数据
           labels 训练标签
    return: 最佳分裂属性的下标(对应属性名列表中位置)
    """
    numFeatures = len(train[0])
    entropy = computeEnt(labels)
    bestmutualInfo = 0.0
    bestFeaIndex = -1
    for i in range(numFeatures):
        #取出一列(一个attribute)
        attribute = [sample[i] for sample in train]
        #获得特征的取值集合
        uniqueVals = set(attribute)
        #计算在选取第i个特征下的条件熵
        conditionalEnt = 0.0
        for value in uniqueVals:
            splitedData,splitedlabels = splitData(train, labels, i, value)
            prob = len(splitedData)/len(train)
            conditionalEnt += prob * computeEnt(splitedlabels)
        #计算互信息（信息增益）
        mutualInfo = entropy - conditionalEnt
        #选取最佳信息增益
        if (mutualInfo > bestmutualInfo):
            bestmutualInfo = mutualInfo
            bestFeaIndex = i
    return bestFeaIndex
```

■ CART模型选取最佳分裂属

```
def CARTSplitFeature(train,labels):
    """
    input: train 训练数据
           labels 训练标签
    return: 最佳分裂属性的下标(对应属性名列表中位置)
    """
```

```
    numFeatures = len(train[0])
    bestGini = float('inf')
    bestFeaIndex = -1
    for i in range(numFeatures):
        #取出一列(一个attribute)
        attribute = [sample[i] for sample in train]
        gini = computeGini(attribute, labels)
        #选取最大信息增益
        if gini < bestGini:
            bestGini = gini
            bestFeaIndex = i

    return bestFeaIndex
```

- **建立决策树：**

```
def buildTree(train,labels,attributes,attrValSet,chooseFeatFun):
    """
    input: train 训练数据
           labels 训练标签
           attributes 属性名列表
           attrValSet 列表，每个元素代表每个属性的取值集合
           chooseFeatFun 选择最佳分裂属性的函数
           使用方法为 chooseFeatFun(train,labels)
           得到的是最佳分裂属性的下标(对应属性名列表中位置)
    return: 一棵用嵌套字典表示的决策树
            如{a:{'值1':{b:{'值1':1,'值2':0}},'值2':0} 表示的就是:
               |值2--分类0
            a--|      |值1--分类1
               |值1--b|
                      |值2--分类0
    """
    #如果数据集为空则表明该属性已经在之前被过滤掉，返回父节点标签的众数
    #此处返回空字典 用于接下来递归处理时的判断
    if len(train)==0:
        return {}
    #所有样本属于同一类别，将当前结点标记为该类别，返回该label
    if labels.count(labels[0]) == len(labels):
        return labels[0]
    #用完了所有特征仍然不能将数据集划分成包含唯一类别的分组，此时返回label 众数
    if len(train[0]) == 0:
        return pd.Series(labels).mode()[0]

    #获取最佳分裂属性
```

```
    bestFeatIndex = chooseFeatFun(train,labels)
    bestFeatLabel = attributes[bestFeatIndex]
    #创建决策树，使用嵌套字典表示
    decisionTree = {bestFeatLabel:{}}
    del(attributes[bestFeatIndex])
    #获取最佳分裂属性中的属性值集合
    attrVals = attrValSet[bestFeatIndex]
    del attrValSet[bestFeatIndex]
    for value in attrVals:
        copyAttr = attributes.copy()
        copyattrValSet=attrValSet.copy()
        #以最佳分裂属性为根节点划分数据集
        splitedData,splitedlabels = splitData(train,labels,bestFeatIndex,value)
        #最佳分裂属性的一个值即为分支，分支再通过递归建树
        subTree = buildTree(splitedData,splitedlabels,copyAttr,
                                copyattrValSet,chooseFeatFun)
        #如果子树为空字典，表明该值在此前已被过滤，返回父节点标签的众数
        if (not isinstance(subTree, dict)) or len(subTree)>0:
            decisionTree[bestFeatLabel][value] = subTree
        else:
            decisionTree[bestFeatLabel][value] = pd.Series(labels).mode()[0]
    return decisionTree
```

- **使用决策树进行预测分类**

```
def classify(decisionTree,attributes,testVec):
    """
    input: decisionTree 建成的决策树，由嵌套字典表示
           attributes 属性名列表
           test 一个测试数据
    return: 预测结果
    """
    #字典的第一个键，对应一个属性
    firstAttr = list(decisionTree.keys())[0]
    #首键对应的值是个字典，字典对应的键就是属性的各个取值，即分支
    firstAttrBranch = decisionTree[firstAttr]
    featIndex = attributes.index(firstAttr)

    predictLabel=None
    #顺着分支递归查找，内部节点为字典，叶节点为预测标签
    for key in list(firstAttrBranch.keys()):
        if testVec[featIndex]==key:
            if isinstance(firstAttrBranch[key], dict):
                predictLabel = classify(firstAttrBranch[key], attributes, testVec)
```

```
            else:
                predictLabel = firstAttrBranch[key]
    return predictLabel
```

- K折交叉验证划分数据集

```
def KFold(dataSet,k):
    """
    input: dataSet 含标签的数据集，类型为 np.array
           k 将数据集划分为 k 个部分
    return: 列表 每个元素代表数据集的一部分
    """
    #获取打乱后的数据集下标
    rowIndex = np.arange(dataSet.shape[0])
    np.random.shuffle(rowIndex)
    foldSize = int(len(dataSet)/k)
    foldList = []
    counts=k
    #每次取 int(len(dataSet)/k) 个 sample 作为一个 fold
    while(counts>1):
        foldList.append(dataSet[rowIndex[(k-counts)*foldSize:
                                         (k-counts+1)*foldSize]])
        counts-=1
    foldList.append(dataSet[rowIndex[(k-1)*foldSize:]])
```

五. 实验结果及分析

1. 使用整个数据集作为训练集建树，再计算在整个训练集上的预测准确率:

```
#######################全部训练全部测试############################
path="D:/AI Lab/lab2/car_train.csv"
train,labels,attributes=loadData(path)
attrValSet=[set([sample[i] for sample in train]) for i in range(len(train[0]))]
ans=buildTree(train.copy(),labels.copy(),attributes.copy(),attrValSet,
              CARTSplitFeature)
correct=0
sampleNum=len(train)
for i in range(sampleNum):
    if(classify(ans,attributes,train[i])==labels[i]):
        correct+=1
print("accuracy: ", correct/sampleNum)
```

只需分别修改buildTree函数中最后一个参数为ID3SplitFeature，C45SplitFeature

和CARTSplitFeature再执行上述代码，即可得到下表的结果：

模型	结果
ID3	In [5]: runfile('D:/AI Lab/lab2/decisionTree.py', accuracy: 1.0
C4.5	In [6]: runfile('D:/AI Lab/lab2/decisionTree.py', accuracy: 1.0
CART	In [7]: runfile('D:/AI Lab/lab2/decisionTree.py', accuracy: 1.0

在训练集上三种模型的准确率都是100%，其一说明该数据集可以完美建树，其二也可以验证建树过程的正确性。

2. 分析决策树的结构并使用其预测样例的标签：

我们可以输出这棵树的字典来大致了解数的结构：

```
In [111]: ans
Out[111]:
{'safety': {'low': 0,
  'med': {'persons': {'more': {'lug_boot': {'med': {'buying': {'vhigh':
{'maint': {'vhigh': 0,
           'low': {'dorrs': {'3': 1, '4': 1, '2': 0, '5more': 1}},
           'med': {'dorrs': {'3': 1, '4': 1, '2': 0, '5more': 1}},
           'high': 0}},
         'low': {'maint': {'vhigh': {'dorrs': {'3': 1,
             '4': 1,
             '2': 0,
             '5more': 1}},
           'low': 1,
           'med': 1,
           'high': 1}},
         'med': {'dorrs': {'3': 1,
           '4': 1,
           '2': {'maint': {'vhigh': 0, 'low': 1, 'med': 1, 'high': 0}},
           '5more': 1}},
         'high': {'maint': {'vhigh': 0,
           'low': {'dorrs': {'3': 1, '4': 1, '2': 0, '5more': 1}},
```

通过大括号的匹配，可以看出根节点属性为safety，safety属性的取值有low，med和high三种，当safety为low时，直接得出预测标签为0，如果safety为med或者high，则还需继续看persons属性的值，然后一直查找到叶节点得出预测结果。

我们以数据集的前三行为例来验证这棵树的正确性：

buying	maint	dorrs	persons	lug_boot	safety	Label
high	vhigh	3	2	small	high	0
high	med	3	2	small	low	0
vhigh	low	2	4	big	low	0

数据集前三个样例

首先分析第一组数据，safety为high，找到根节点safety的high分支：

```
'high': {'persons': {'more': {'buying': {'vhigh':
      'low': {'dorrs': {'3': 1,
        '4': 1,
        '2': {'lug_boot': {'med': 1, 'small': 0,
        '5more': 1}},
      'med': {'dorrs': {'3': 1,
```

然后查看persons属性，为2，找到persons属性值为2的分支，得到预测标签0。

```
'4': {'buying': {'vhigh': {'maint': {'vhigh': 0,
    'low': 1,
    'med': 1,
    'high': 0}},
  'low': 1,
  'med': 1,
  'high': {'maint': {'vhigh': 0, 'low': 1, 'med': 1, 'high': 1}}}},
'2': 0}}}}
```

对于第二三行的数据，safety属性为low，直接得出预测标签0，均与真实值相符。

3. 使用K折交叉验证评估其准确率

模型	K的取值	结果
ID3	5	In [114]: runfile('D:/AI Lab/lab2/decisionTree.py', 0.9593023255813954 0.9796511627906976 0.9680232558139535 0.9709302325581395 0.962536023054755 5 Fold Average accuracy: 0.9680885999597884
	10	In [104]: runfile('D:/AI Lab/lab2/decisionTree.py', 0.9590643274853801 0.9649122807017544 0.9649122807017544 0.9532163742690059 0.9649122807017544 0.9415204678362573 0.9883040935672515 0.9766081871345029 0.9590643274853801 0.9776536312849162 10 Fold Average accuracy: 0.9650168251167957
C4.5	5	In [101]: runfile('D:/AI Lab/lab2/decisionTree.py', 0.9593023255813954 0.9651162790697675 0.9593023255813954 0.9651162790697675 0.9740634005763689 5 Fold Average accuracy: 0.964580121975739

	10	In [103]: runfile('D:/AI Lab/lab2/decisionTree.py', 0.9824561403508771 0.9532163742690059 0.9415204678362573 0.9415204678362573 0.9707602339181286 0.9707602339181286 0.9649122807017544 0.9590643274853801 0.9649122807017544 0.9664804469273743 10 Fold Average accuracy: 0.9615603253944919
CART	5	In [95]: runfile('D:/AI Lab/lab2/decisionTree.py', 0.9534883720930233 0.9505813953488372 0.9622093023255814 0.9709302325581395 0.9711815561959655 5 Fold Average accuracy: 0.9616781717043095
	10	In [94]: runfile('D:/AI Lab/lab2/decisionTree.py', 0.9532163742690059 0.9649122807017544 0.9649122807017544 0.9707602339181286 0.9649122807017544 0.9649122807017544 0.9532163742690059 0.9649122807017544 0.9766081871345029 0.9776536312849162 10 Fold Average accuracy: 0.9656016204384331

这里选取的K值是比较常用的5和10，经过多次试验对比可以看出在该数据集上，三种模型的预测准确度差距非常小，都在96%左右，ID3模型比C4.5和CART略好一些。

六. 思考题

1. 决策树有哪些避免过拟合的方法?

①提前终止：限制树高，可以利用交叉验证的准确率来衡量，如果下一次分裂没有降低误差，则停止分类限制树的节点个数。

②剪枝，分为预剪枝和后剪枝，预剪枝是对于当前的结点，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；后剪枝则是先生成完整的决策树，再对其后序遍历，对于遍历过程中的非叶结点，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。

③融入降低模型复杂度的惩罚项，给每个叶节点一个惩罚系数，再对决策树使用泛化错

误率e_c来衡量决策树的优劣。在此种情况下惩罚系数为1意味着只有当分裂一个节点能够使多于一个的训练样本被正确分类时，该节点才应该被分裂。

2. C4.5相比于ID3的优点是什么，C4.5又可能有什么缺点?

优点：ID3模型中信息增益的衡量容易偏向那些分支数众多的属性，导致过拟合问题。而C4.5使用了信息增益率作为衡量指标克服了上述问题，它对信息增益进行了正则化，提高了泛化能力。此外，ID3模型只能处理离散特征，而C4.5利用连续特征离散化的方法解决了连续特征的问题。

缺点：处理连续特征值时需要扫描排序，这使得C4.5的耗时大大增加。此外，C4.5虽然对ID3进行了拓展，但依旧不能解决回归问题。

3. 如何用决策树来进行特征选择（判断特征的重要性）?

决策树中特征的重要程度衡量模型主要有ID3，C4.5和CART三种：

①ID3模型中特征重要程度的衡量标准是信息增益，需要计算数据集的信息熵和选取特征后的条件熵，二者相减即得到信息增益，即确定特征后数据集的熵减少了多少，信息增益越大的特征即可认为它越重要。

②C4.5模型克服了ID3模型中信息增益的衡量容易偏向那些分支数众多的属性导致的过拟合问题，使用信息增益率作为衡量指标，信息增益率即信息增益除以该属性的熵，即选取信息增益高且取值集合比较小的属性，信息增益率越大即可认为其越重要。

③CART模型使用Gini系数作为衡量指标，先计算各个特征对应的Gini系数，然后选择Gini系数最小的属性分裂。Gini系数是经济学术语，Gini系数越小表明该国家收入分配越是趋向平等，即趋同性越好，因此属性的Gini系数则可认为它越重要。

七. 参考文献

1) https://docs.scipy.org/doc/numpy/

2) https://pandas.pydata.org/pandas-docs/stable/

3) 《机器学习实战》Peter Harrington page32-50 北京.人民邮电出版社
ISBN: 9787115317957