决策树——基于蘑菇可食性的数据分析

李厚霖 520020910007

一、 问题描述

在决策树模型中，我采用了在 UCI 的数据集 Mushroom Data，其包含了根据奥杜邦协会北美蘑菇实地指南（1981 年）得出的蘑菇和麻风菌科 23 种有鳃蘑菇的假设样本的描述，其中每一种蘑菇都被确定为绝对可食用(definitely edible)，绝对有毒(definitely poisonous)，或未知食用性，不推荐(unknown edibility and not recommended)，后一类样本在本数据集中是和有毒的一类样本结合在一起的，他们的可食用性统称为 poisonous，即不可食用。本次作业我将利用决策树的模型，来对蘑菇的一系列特征与其可食用性间的关系进行建模，以此来判断未知特征蘑菇的实用性。

根据网站中所给数据的描述，所选数据集的蘑菇共有 22 个特征，他们分别表示为：

1. cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
2. cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s
3. cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r,
pink=p,purple=u,red=e,white=w,yellow=y
4. bruises?: bruises=t,no=f
5. odor: almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s
6. gill-attachment: attached=a,descending=d,free=f,notched=n
7. gill-spacing: close=c,crowded=w,distant=d
8. gill-size: broad=b,narrow=n
9. gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g,
green=r,orange=o,pink=p,purple=u,red=e, white=w,yellow=y
10. stalk-shape: enlarging=e,tapering=t
11. stalk-root: bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z,rooted=r,missing=?
12. stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s
13. stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s
14. stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,
pink=p,red=e,white=w,yellow=y
15. stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,
pink=p,red=e,white=w,yellow=y
16. veil-type: partial=p,universal=u
17. veil-color: brown=n,orange=o,white=w,yellow=y
18. ring-number: none=n,one=o,two=t
19. ring-type: cobwebby=c,evanescent=e,flaring=f,large=l,
none=n,pendant=p,sheathing=s,zone=z
20. spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r,
orange=o,purple=u,white=w,yellow=y
21. population: abundant=a,clustered=c,numerous=n, scattered=s,several=v,solitary=y
22. habitat: grasses=g,leaves=l,meadows=m,paths=p, urban=u,waste=w,woods=d

数据集链接为：https://archive.ics.uci.edu/ml/datasets/mushroom/

因此，对于多特征问题，采用决策树的模型能够较好的分析、判读蘑菇特征与其是

否可食用间的关系。

二、 模型理论

1、决策树简介

决策树（decision tree）是一种基本的分类与回归方法，主要是用来对数据进行预测分类。它是最符合直觉的分类器之一，且容易理解和构建、性能通常非常出色。用决策树分类基本上是以下流程：从根节点开始，对实例的某一特征进行测试，根据测试结果将实例分配到其子节点，此时每个子节点对应着该特征的一个取值，如此递归的对实例进行测试并分配，直到到达叶节点，最后将实例分到叶节点的类中。

2、信息熵和信息增益

在训练过程中，为了建立一棵决策树，我们需要在建树过程中为树的每一个结点挑选一个特征，为了使得所挑选出的特征能够最好的分割数据，我们需要引入“信息熵”和“信息增益”的概念。信息熵描述了数据的混乱程度，信息熵越大代表数据越混乱，即数据类别的不确定性越大。因此，在没有先验的条件下，信息熵为：

$$\mathrm{H(p)} = -\sum_{i=1}^{n} p_i \log_2 p_i$$

但是如果我们已知某些先验知识，例如样本的某些特征，在这种条件下，数据的信息熵会发生改变，改变后的熵称之为条件熵，条件熵的计算公式为:

$$\mathrm{H(Y|X)} = \sum p(\mathrm{X=i})\mathrm{H\,(Y|X=i)}$$

其中 Y 为样本的类别，X 为我们已知的样本的特征，该特征包含 n 种属性。

从而我们导出信息增益的概念，它表示为得知特征 X 的信息而使得类 Y 的信息的不确定性减少的程度，其公式为：

$$\mathrm{G(Y|X)} = \mathrm{H(Y)} - \mathrm{H(Y|X)}$$

3、决策树的构建

构建决策树的算法有很多，比如 C4.5、ID3 和 CART，本次实验报告中采用较为简单的 ID3 算法来构建决策树，ID3 相当于用极大似然法进行概率模型的选择。

具体方法是：

- 计算当前数据集的信息熵 H(Y)
- 计算在分别得知不同特征 X 的情况下，当前数据集的条件熵 H(Y|X)
- 分别计算在当前数据集下各自的信息增益$\mathrm{IG(Y|X)}$，找到特征 X 使得 IG 最大，挑选该特征作为当前结点
- 分别对 X 的每一项属性进行研究(每一项属性均生成一个子结点)，若该属性下所有样本的分类均为同一类，则该子结点返回该类别；若样本包含不同种类，则只关注与该属性相关的样本，剔除其他样本，对该子结点重复 1-4 号步骤。
- 当所有子结点均返回类别或特征列表为空，结束建树过程。

在决策树建立完成之后，我们可以通过 matplotlib 将构建的决策树进行可视化，帮助我们理解决策树是如何生成的。

其伪代码如下图所示：

输入：训练集 $D=\{(x_1,y_1),(x_2,y_2),\ldots,(x_m,y_m)\}$；
　　属性集 $A=\{a_1,a_2,\ldots,a_d\}$.
过程：函数 TreeGenerate(D, A)
1: 生成结点 node;
2: **if** D 中样本全属于同一类别 C **then**
3:　将 node 标记为 C 类叶结点; **return**　　递归返回，情形(1).
4: **end if**
5: **if** $A=\varnothing$ **OR** D 中样本在 A 上取值相同 **then**
6:　将 node 标记为叶结点，其类别标记为 D 中样本数最多的类; **return**　　递归返回，情形(2).
7: **end if**
8: 从 A 中选择最优划分属性 a_*;　　我们将在下一节讨论如何获得最优划分属性.
9: **for** a_* 的每一个值 a_*^v **do**
10:　为 node 生成一个分支；令 D_v 表示 D 中在 a_* 上取值为 a_*^v 的样本子集;
11:　**if** D_v 为空 **then**
12:　　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**　　递归返回，情形(3).
13:　**else**
14:　　以 TreeGenerate(D_v, $A\setminus\{a_*\}$)为分支结点　　从 A 中去掉 a_*.
15:　**end if**
16: **end for**
输出：以 node 为根结点的一棵决策树

三、 代码实现

1、 调用库

```
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
```

2、 计算熵函数和条件熵

```
##信息熵
def Entropy(data):
    labellist = {} #记录yes和no的个数
    for tmp in data:
        if tmp[-1] not in labellist:
            labellist[tmp[-1]] = 1
        else:
            labellist[tmp[-1]]+=1

    dataEntropy = 0
    for tmp in labellist:
        dataEntropy-=(labellist[tmp]/len(data)) * math.log2(labellist[tmp]/len(data))
    return dataEntropy
```

```
def condEntropy(data,i):
    #获取data第i列的各个值
    allValue = [example[i] for example in data]
    allValue=set(allValue)
    conditionEntropy=0
    for key in allValue:
        newdata=spliteData(data,key,i)
        conditionEntropy=conditionEntropy+len(newdata)/len(data) *Entropy(newdata)
    return conditionEntropy
```

```python
def spliteData(data,value,i):
    newdata=[]
    for row in data:
        if row[i]==value:
            newVec = row[:i]
            newVec.extend(row[i + 1:])
            newdata.append(newVec)
    return newdata
```

3、 选取最佳划分属性

```python
#选择最优划分
def optimalPartition(data, attribute):
    originalEntroy=Entropy(data)
    bestGainEnt=0
    bestopt=-1
    attributeLen=len(attribute)
    for i in range(attributeLen):
        conditionEntropy = condEntropy(data, i)
        gainEntropy = originalEntroy-conditionEntropy
        if gainEntropy > bestGainEnt:
            bestGainEnt = gainEntropy
            bestopt = i
    return bestopt
```

4、 决策树的构建

```python
#构建决策树
def createTree(data, attribute):
    attribute_copy = attribute[:]
    #获取yes和no的标签
    labels = [tmp[-1] for tmp in data]
    #全为一类
    if labels.count(labels[0]) == len(labels):
        return labels[0]
    #只剩一个属性
    if len(data[0]) == 1:
        return majorityCnt(labels)
    #其他情况
    bestopt = optimalPartition(data, attribute)
    bestattribute = attribute[bestopt]
    myTree={bestattribute:{}}
    allvalues = [tmp[bestopt] for tmp in data]
    allvalues = set(allvalues)

    del(attribute_copy[bestopt])
    for value in allvalues:
        #modify1
        myTree[bestattribute][value]=createTree(spliteData(data,value,bestopt),attribute_copy)
    #print(myTree)
    return myTree
```

```python
##计算哪个类别最多
def majorityCnt(datalabel):
    labelCnt = {}
    for key in datalabel:
        if key not in labelCnt:
            labelCnt[key] = 1
        else :
            labelCnt[key] +=1

    labelCnt = sorted(labelCnt.items(), key= lambda x :x[1], reverse=True)
    return labelCnt[0][0]
```

5、 模型检验

```python
#测试算法
def classify(decisionTree, testVec, attr_list):
    feature = list(decisionTree.keys())[0]          # feature为决策树的根节点
    feature_dict = decisionTree[feature]            # feature_dict为根节点下的子树
    feature_index = attr_list.index(feature)        # feature_index为feature对应的属性名索引
    feature_value = testVec[feature_index]          # feature_value为测试集中对应属性的值
    label = None
    if feature_value in feature_dict.keys():
        # 如果没有结果就继续向下找
        if type(feature_dict[feature_value]) == dict:
            label = classify(feature_dict[feature_value], testVec, attr_list)
        else:
            label = feature_dict[feature_value]
    return label

def testAccuracy(decisionTree, test_sample_list, test_label_list, attr_list):
    rightNum = 0
    predict_label_list = []
    for i in range(len(test_sample_list)):
        predict_label = classify(decisionTree, test_sample_list[i], attr_list)
        predict_label_list.append(predict_label)
        if predict_label == test_label_list[i][0]:
                rightNum += 1
    accuracy = rightNum/len(test_sample_list)
    return accuracy*100
```

6、 主函数

```python
##数据处理
data = pd.read_csv("./dataset1.csv", sep=',')
data = np.array(data)
#print(data)
train_data_x = data[0:6200,1:24]
print(train_data_x.shape)
train_data_y = data[0:6200,-1]
test_data_x = data[6200:8000,1:23]
#print(test_data_x)
test_data_y = data[6200:8000,23:24]
attribute = ['cap-shape','cap-surface','cap-color','bruises','odor','gill-attachment','gill-spacing','gill-size','gil
train_data_x_list = train_data_x.tolist()

Tree = createTree(train_data_x_list, attribute)
print(" Mytree:")
print(Tree)
createPlot(Tree)
test_data_x_list = test_data_x.tolist()
#print(test_data_x_list)
test_data_y_list = test_data_y.tolist()
#print(test_data_y_list)
print("Accuracy :", testAccuracy(Tree, test_data_x_list, test_data_y_list, attribute), "%")
```

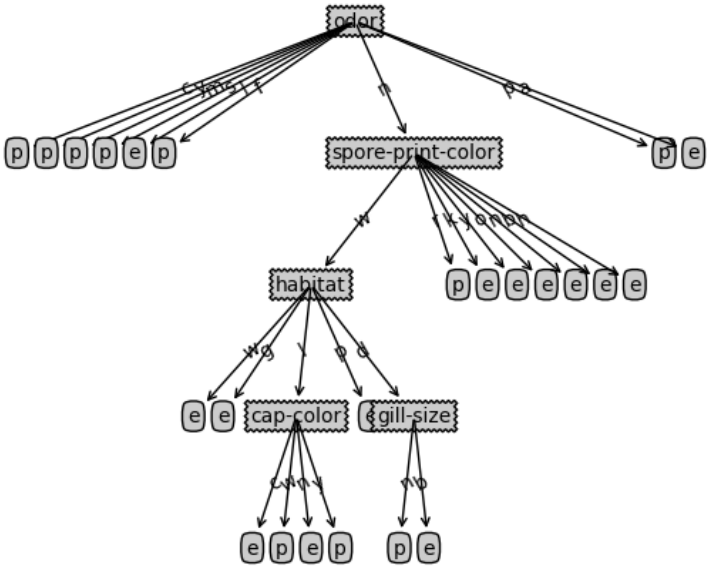
四、 结果分析

1、 运行结果

按照训练集与数据集 4:1 的最佳比例，我将 8000 条数据的前 6400 条作为训练集，剩下的 1600 条数据作为测试集，可以得到如下结果：

```
Mytree:
{'odor': {'c': 'p', 'y': 'p', 'm': 'p', 's': 'p', 'l': 'e', 'f': 'p', 'n': {'spore-print-color': {'w': {'habitat': {'w': 'e', 'g': 'e', 'l': {'cap-
color': {'c': 'e', 'w': 'p', 'n': 'e', 'y': 'p'}}, 'p': 'e', 'd': {'gill-size': {'n': 'p', 'b': 'e'}}}}, 'r': 'p', 'k': 'e', 'y': 'e', 'o': 'e',
'n': 'e', 'b': 'e', 'h': 'e'}}, 'p': 'p', 'a': 'e'}}
```

同时，我参照网上的画决策树的代码（具体在.ipynb 文件中展示），将其以可视化做出决策树的图如下：



此后，我将测试集输入进行预测分类，惊讶的发现得到的精确度结果居然为 100%，不可思议。

```
Accuracy : 100.0 %
```

2、 五倍交叉验证

1 中的数据是以 0-6400 组数据作为训练集，6400-8000 组作为测试集得到的结论，为验证该方法的正确性，我们分别将 0-1600、1600-3200、3200-4800、4800-6400、6400-8000 作为测试集，其余作为训练集，计算正确率，如下表

测试集	0-1600	1600-3200	3200-4800	4800-6400	6400-8000
正确率	**100%**	**100%**	**100%**	**100%**	**100%**

五倍交叉验证之后，平均准确率为 65.5%，任务完成得十分理想，超乎想象。也就是说我们所构建的决策树模型能够完美地依靠蘑菇的部分特征分辨出蘑菇的毒性，这对我们判断蘑菇是否有毒提供了极大的帮助，为研究蘑菇毒性的来源提供了数据支持。同时，通过决策树的可视化图像，我们可以发现蘑菇的 odor 特征最能分辨出蘑菇是否具有毒性，其次是 spore-print-color，接下来是 habitat，最后再通过 cap-color 和 gill-size 基本能够实现准确分类。

五、 总结与展望

1、 误差分析

经过五倍交叉验证后，我们发现预测得准确率基本就是 100%，这说明在当前数据的情况下，构建决策树进行分类时，准确率相当高。 探究其原因，应该是数据样本的数量较多，同时数据依照特征的分类效果好，数据集很适合用决策树来预测分类。

2 总结

本次用决策树的方式，根据 22 种特征数据，划分来判断蘑菇是否有毒，能否食用。通过训练集和测试集的划分，通过训练集构建决策树，用测试集测试预测效果，最终在五倍交叉验证下，平均正确率为 100%。总体预测效果较为理想，任务完成的较为圆满