机器学习实验报告

题　　目：	决策树
学生姓名：	张芮熙
学　　号：	22354188
指导教师：	马倩
专业班级：	22智科3班

2021年5月

本科生院制

目 录

第一部分——编写函数

1. 实验目的、思路：

导入数据集，编写函数计算数据集的信息熵，同时利用ID3,C4.5,GINI三种不同的算法对数据进行选择，求出最佳的信息增益和维度，求解最佳的决策树。

2.核心代码与实现

（1）数据导入

该数据集(train_titanic.csv)为分类数据集，为泰坦尼克号的部分乘客信息以及最后是否生还。包括了四个属性值以及一个标记属性(即为Survived类型,代表是否生还),属性信息分别为Sex(性别)，sibsp(堂兄弟妹个数)，Parch(父母与小孩的个数)，Pclass(乘客等级)。
其中该数据集无缺失值和异常值，且所有连续变量已自动转换为离散变量,标记(Survived类型)也自动转变为离散变量0和1，所以你无需进行数据预处理，可以直接使用该数据集。

In [21]:
```
from collections import Counter
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import pandas as pd
import numpy as np
import math
from collections import Counter
```

1) 使用pandas库将训练数据集'train_titanic.csv'载入到Dataframe对象中

In [22]:
```
# 加载数据集
data = pd.read_csv('train_titanic.csv')
print(data)
```

```
      Sex  sibsp  Parch  Pclass  Survived
0       0      1      0       3         0
1       1      1      0       1         1
2       1      0      0       3         1
3       1      1      0       1         1
4       0      0      0       3         0
...   ...    ...    ...     ...       ...
1004    0      0      0       3         0
1005    1      0      0       1         0
1006    0      0      0       3         0
1007    0      0      0       3         0
1008    0      1      1       3         0

[1009 rows x 5 columns]
```

运行结果：

输出了数据集Titanic，共有1009行，5列。

	Sex	sibsp	Parch	Pclass	Survived
0	0	1	0	3	0
1	1	1	0	1	1
2	1	0	0	3	1
3	1	1	0	1	1
4	0	0	0	3	0
...	...	...	...	...	...
1004	0	0	0	3	0
1005	1	0	0	1	0
1006	0	0	0	3	0
1007	0	0	0	3	0
1008	0	1	1	3	0

[1009 rows x 5 columns]

（2）计算信息熵

核心思路：利用counter类，取出数据的不同取值和这些不同取值出现的次数，出现比例=出现次数/样本总个数，这样我们求得了pk,遍历所有的不同取值求和，得到信息熵entropy.

2) 编写函数，给定任何标记数组计算其信息熵
输入：标记数组
输出：该数组对应的信息熵
计算信息熵公式: 某数组包含K个不同的取值，样本为第k(k=1,2,...,K)个值的数量所占比例为p_k,则其信息熵为

$$Ent = -\sum_{k=1}^{K} p_k log_2 p_k$$

例:
输入:[[0],[1]]
输出:(-$\frac{1}{2}$ log_2 $\frac{1}{2}$)+(-$\frac{1}{2}$ log_2 $\frac{1}{2}$)=1

```
1.
#定义计算信息熵函数
def calculate_entropy(labels):
    label_counts = Counter(labels)
    entropy = 0
    for count in label_counts.values():
        probability = count / len(labels)
        entropy -= probability * math.log2(probability)
    return entropy
```

(3)定义划分不同数据集函数

数据集一共有四个feature值，构造一个函数，可以将数据集按照不同的维数进行划分，得到被分裂的数据集。

3) 编写函数，函数功能为将所给的数据集按照指定维度dimension进行划分为若干个不同的数据集
【输入】：属性集合，标记集合，维度索引
【输出】：划分后所得到的子树属性集合，子树标记集合

例子:
【输入】:feature(属性值集合):[[0,0,0],[0,0,1],[1,0,2]]
label(标记集合):[[0],[1],[2]]
划分维度:0

【输出】:[[0,0,0],[0,0,1]]和[[1,0,2]]
[[0],[1]]和[[2]]
tips:即将feature按其第0维度进行划分，由于feature的0维有0和1两个不同的数值，所以feature划分为[[0,0,0],[0,0,1]]和[[1,0,2]]，label划分为[[0],[1]]和[[2]]

```
2.
#定义划分不同数据集函数
def split_dataset(features, labels, dimension):
    unique_values = np.unique([row[dimension] for row in features])
    split_subsets = []
    for value in unique_values:
        subset = [[], []]  # 第一个列表存放属性，第二个列表存放标记
        for i, row in enumerate(features):
            if row[dimension] == value:
                subset[0].append(row)
                subset[1].append(labels[i])
        split_subsets.append(subset)
    return split_subsets
```

（4）ID3算法

根据信息增益计算公式，遍历feature，把best_gain（最大信息增益）和info_gain（当前信息增益）进行比较，得到最大的信息增益和划分的dimension.

4) 编写函数，函数功能为进行【一次】决策树的结点划分，遍历找出该属性集合中信息增益(使用ID3算法中的公式计算)【最大】的属性
输入：属性集合，标记集合
输出：该次划分的最佳信息增益，最佳划分维度
计算信息增益公式:
某数据集D有若干属性值以及对应的标记值，其总样本大小为|D|,这里取其中一个属性类型feature,该特征包含V个不同的取值，属性值为第v(v=1,2,...,V)个值的数量为$|D^v|$($\sum_{v=1}^{V}|D^v|=|D|$),则该属性值对应的信息增益为为

$$Gain(D, feature) = Ent(D) - \sum_{v=1}^{V}\frac{|D^v|}{|D|}Ent(D^v)$$

所以该函数中你需要计算出每个属性值的信息增益并输出，然后返回最大的信息增益并返回该特征的维数以及最大的信息增益值

```
3.
#定义信息增益最大的决策划分（ID3算法）
def find_best_split_id3(features, labels):
    base_entropy = calculate_entropy(labels)
    best_gain = 0
    best_dimension = -1
    num_features = len(features[0])
    #遍历
    for i in range(num_features):
        subsets = split_dataset(features, labels, i)
        new_entropy = 0
        for subset in subsets:
            subset_entropy = calculate_entropy(subset[1])
            new_entropy += (len(subset[1]) / len(labels)) * subset_entropy
        info_gain = base_entropy - new_entropy
        if info_gain > best_gain:
            best_gain = info_gain
            best_dimension = i

    return best_gain, best_dimension
```

（5）C4.5算法

在ID3的基础上，增加计算信息增益率功能，信息增益率=（最大信息增益）/(信息熵)，遍历数据集，将gain_ratio和best_gain_ratio进行比较，从而获得最佳的信息增益率和其对应的dimension。

5) 编写函数，函数功能为进行【一次】决策树的结点划分，遍历找出该属性集合中信息增益率(使用C4.5算法中的公式计算)【最大】的属性
输入：属性集合，标记集合
输出：该次划分的信息增益率，最佳维度
计算信息增益率公式:
某数据集D有若干属性值以及对应的标记值，其总样本大小为|D|,这里取其中一个属性类型feature,该属性包含V个不同的取值，属性值为第v(v=1,2,...,V)个值的数量为$|D^v|(\sum_{v=1}^{V}|D^v|=|D|)$,该属性本身的信息熵为Ent(feature),则该属性值对应的信息增益率为

$$Gain\_ratio(D, feature) = \frac{Gain(D, feature)}{Ent(feature)}$$

所以该函数中你需要输出每个特征的信息增益率，之后返回最大的信息增益率对应的属性维数以及最大的信息增益率

```
4.
#定义信息增益率最大决策划分（C4.5算法）
#需要结合前面的ID3算法和计算信息熵的函数相除
def find_best_split_c45(features, labels):
    base_entropy = calculate_entropy(labels)
    best_gain_ratio = 0
    best_dimension = -1
    num_features = len(features[0])

    for i in range(num_features):
        subsets = split_dataset(features, labels, i)
        new_entropy = 0
        split_info = 0
        for subset in subsets:
            subset_entropy = calculate_entropy(subset[1])
            proportion = len(subset[1]) / len(labels)
            new_entropy += proportion * subset_entropy
            split_info -= proportion * math.log2(proportion)
        info_gain = base_entropy - new_entropy
        if split_info != 0:
            gain_ratio = info_gain / split_info
            if gain_ratio > best_gain_ratio:
                best_gain_ratio = gain_ratio
                best_dimension = i
    return best_gain_ratio, best_dimension
```

（6）利用GINI指数进行判断

基尼指数=（1-遍历样本中重复的概率的平方求和），基尼指数越大，说明信息熵越大，样本的类型越多，基尼指数越小，说明信息熵越小，样本的类型越少，大多数集中在同一个类型，越接近最佳，在这个算法中，遍历样本，将best_gini_index进行迭代，求解最小的基尼指数和对应的dimension，对应的分类值。

6) 编写函数，进行【一次】决策树的结点划分，遍历找出该属性集合中基尼系数(使用CART算法中的公式计算)最小的属性以及最佳的划分值
输入：属性集合，标记集合
输出：该次划分的最佳的基尼系数，最佳维度，最佳划分值
计算基尼系数公式:
某数据集D有若干属性值以及对应的标记值，其总样本大小为|D|,该集合中样本标记类别总共有K类，第k类样本所占比例为p_k(k=1,2,...,K)则该数据集对应的基尼系数为

$$Gini(D) = 1 - \sum_{k=1}^{K} p_k^2$$

而取其中一个属性类型feature，选定该类型的一个值value，根据feature这个属性是否为value将数据集分为两个子集D_1和D_2,则该属性feature对应的基尼系数为

$$Gini\_index(D, feature) = \frac{|D_1|}{|D|}Gini(D_1) + \frac{|D_2|}{|D|}Gini(D_2)$$

所以该函数中你需要首先找出每列中的非重复值，之后根据该列的非重复值进行分类，计算出该列中以该值进行分类的基尼系数，计算出每个属性的每个非重复值的基尼系数，之后找到最小的基尼系数对应的属性维数以及对应的分类值

```
5.
#利用基尼系数进行划分，
def find_best_split_cart(features, labels):
    best_gini_index = float('inf')
    best_dimension = -1
    best_split_value = None
    num_features = len(features[0])

    for i in range(num_features):
        unique_values = np.unique([row[i] for row in features])
        for value in unique_values:
            subsets = [[], []]  # 分别存储两个子集的标记
            for j, row in enumerate(features):
                if row[i] == value:
                    subsets[0].append(labels[j])
                else:
                    subsets[1].append(labels[j])
            gini_index = sum(
                [(len(subset) / len(labels)) * (1 - sum([((subset.count(k) / len(subset)) ** 2) for k in np.unique(subset)]))
                 for subset in subsets])
            if gini_index < best_gini_index:
                best_gini_index = gini_index
                best_dimension = i
                best_split_value = value

    return best_gini_index, best_dimension, best_split_value
```

（7）将ID3,C4.5,GINI三种判断算法综合利用，求解最优信息增益和其对应的dimension

7) 应用之前你在第4、5、6个部分编写的三个函数，在训练数据集'train_titanic.csv'上依次使用这些函数进行【一次】结点划分，并输出对应的最佳属性维数以及相应的信息增益值/信息增益率/(基尼系数和分类值)

```
7.
#实现一次划分
features = data.iloc[:, :-1].values  # 特征值
labels= data.iloc[:, -1].values      # 标签值
print("信息熵是：",calculate_entropy(labels)) # 调用函数计算信息熵
print("经过数据集函数划分后：")
print(split_dataset(features, labels, 2))  #划分不同数据集函数


#使用ID3算法进行信息划分：
best_gain, best_dimension=find_best_split_id3(features, labels)       #ID3算法
print("best_gain:",best_gain)
print("best_dimension:",best_dimension)


#使用C4.5算法进行信息划分：
best_gain_ratio, best_dimension2=find_best_split_c45(features, labels)
print("best_gain:",best_gain)
print("best_dimension:",best_dimension)


#使用gini函数进行信息划分：
best_gini_index, best_dimension3, best_split=find_best_split_cart(features, labels)
print("best_gini_index:",best_gini_index)
print("best_dimension:",best_dimension)
print("best_split:",best_split)
```

运行结果：

```
信息熵是： 0.7709131174627031
ID3:
best_gain: 0.10750711887455178
best_dimension: 0
C4.5:
best_gain: 0.10750711887455178
best_dimension: 0
GINI:
best_gini_index: 0.2964915724641573
best_dimension: 0
best_split: 0
```

可见，未进行处理的原数据的信息熵较大，说明样本的取值较为分散，用三种算法进行处理，都得到了第一个维度的信息增益最大，也就是对应样本中性别的信息熵最大，说明男女比例可能差距较大（经过打开数据集的Excel表格发现，在1009个样本中存在两类样本0,1代表性别，平均值只有0.36，说明性别比例较大）

3. 心得体会

信息增益最大的特征值所对应的维度具有以下特点：样本之间差异大，对分类结果影响显著，具有代表性。但是需要注意的是，信息增益最大的特征值所对应的维度并不一定与label值最相关。

第二部分——构建决策树

1.实验目的、思路：

利用第一部分中的计算熵entropy函数，以及划分数据集的split函数，还要定义信息增益最大的最优划分函数，由于实验要求是根据ID3算法进行决策树构建并求出预测准确率，在这里，就只使用ID3算法而不是利用GINI指数和C4.5算法进行求解。

2.核心代码及实现：

（1）定义计算熵的公式，定义划分不同数据集的函数，定义信息增益最大的最优划分（和第一部分一致）

定义计算熵的公式，定义划分不同数据集的函数，定义信息增益最大的最优划分

```
#定义计算信息熵函数
def entropy(label):
    counter = Counter(label)
    ent = 0.0
    for i in counter.values():
        ent = ent - i / len(label) * math.log2(i / len(label))
    return ent
#定义划分不同数据集函数
def split(feature, label, dimension):
    unFeature = np.unique(feature[:,dimension])
    unFeatureNum = len(unFeature)

    split_feature = list()
    split_label = list()
    splitElementF = list()
    splitElementL = list()

    for i in range(unFeatureNum):
        splitElementF.append([])
        splitElementL.append([])

    for i in range(len(label)):
        for j in range(unFeatureNum):
            if feature[i, dimension] == unFeature[j]:
                splitElementF[j].append(feature[i,:])
                splitElementL[j].append(label[i])
                break;

    for i in range(unFeatureNum):
        split_feature.append(np.array(splitElementF[i]))
        split_label.append(np.array(splitElementL[i]))

    return split_feature,split_label
```

```
#定义信息增益最大的最优划分
def one_split_ID3(feature, label):

    gain = list()
    entD = entropy(label)

    for i in range(feature.shape[1]):
        split_feature,split_label = split(feature, label, i)
        s = 0.0
        for j in range(len(split_label)):
            entDv = entropy(split_label[j])
            s = s + entDv * len(split_label[j]) / feature.shape[0]
        gain.append(entD - s)

    best_entropy = -1.0
    best_dimension = -1

    for i in range(feature.shape[1]):
        if gain[i] > best_entropy:
            best_entropy = gain[i]
            best_dimension = i

    return best_entropy, best_dimension
def best_split(D, A):

    label = D[:,-1]
    feature = []
    for i in range(D.shape[1]-1):
        if A[i] == 1:
            feature.append(D[:,i])
    f = np.array(feature)
    best_entropy, best_dimension = one_split_ID3(f.T, label)
    b = best_dimension
    for i in range(D.shape[1]-1):
        if A[i] == 1:
            b = b - 1
            if b == -1:
                best_dimension = i

    return best_dimension
```

（2）建立树的结点（参考PPT上的伪代码）

建立树的结点

```
# 树结点类
class Node:
    def __init__(self, isLeaf=True, label=-1, index=-1):
        self.isLeaf = isLeaf # isLeaf表示该结点是否是叶结点
        self.label = label # label表示该叶结点的label（当结点为叶结点时有用）
        self.index = index # index表示该分支结点的划分属性的序号（当结点为分支结点时有用）
        self.children = {} # children表示该结点的所有孩子结点，dict类型，方便进行决策树的搜索

    def addNode(self, val, node):
        self.children[val] = node #为当前结点增加一个划分属性的值为val的孩子结点
```

（3）构建决策树

在DTree类中定义一系列函数：生成结点函数，选择信息增益最大函数，将信息增益最大的作为划分决策树的结点。

1. 决策树的构建（参考PPT的伪代码）

在DTree中定义一个生成结点的函数Treegenerate，把数据集的标签值导出到label中，利用unique函数，如果label_unique的长度是1，那么就说明样本属于同一个类型，可以生成结点，叶子结点标记为true。反之，如果label_unique的长度不是1，说明样本不属于同一个类型，那么就遍寻样本，找到样本最多的类型，并构建结点。

```
# 决策树类
class DTree:
    def __init__(self):
        self.tree_root = None #决策树的根结点
        self.possible_value = {} # 用于存储每个属性可能的取值
    def TreeGenerate(self, D, A):
        # 生成结点 node
        node = Node()
        #获得标签值（最后一列）
        label = D[:,-1]
        label_unique = np.unique(label)
        if len(label_unique) == 1:# if D中样本全属于同一类别C then
            node.isLeaf = True     #将node标记为C类叶结点并返回
            node.label = label_unique[0]
            return node     # end if
        # if A = Ø OR D中样本在A上取值相同 then
        # end if
        t1 = 0
        t2 = 0
        for i in range(len(A)):
            if A[i] == 1:
                t1 = t1 + 1
                feature_unique = np.unique(D[:,i])
                if len(feature_unique) != 1:
                    t2 = t2 + 1
        if t1 == 0 or t2 == 0:
            node.isLeaf = True
            counter = Counter(label)
            counter_values = list(counter.values())
            counter_label = list(counter)
            max = counter_values[0]
            node.label = counter_label[0]
            for i in range(len(counter_values)):
                if counter_values[i] > max:
                    # 将node标记叶结点，其类别标记为D中样本数最多的类并返回
                    max = counter_values[i]
                    node.label = counter_label[i]
            return node
```

2. 通过最大信息增益划分决策树

```
# 从A中选择最优划分属性a_star
# （选择信息增益最大的属性，用到上面实现的best_split函数）
best_dimension = best_split(D, A)
#     为node 生成每一个分支；令D_v表示D中在a_star上取值为a_star_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
#     else
#         以TreeGenerate(D_v,A-{a_star}) 为分支结点
#     end if
# end for
feature_a_unique = self.possible_value[best_dimension]
# for a_star 的每一个值a_star_v do
for i in range(len(feature_a_unique)):
    newNode = Node()
    node.index = best_dimension
    node.isLeaf = False
    Dv = []
    for j in range(D.shape[0]):
        if D[j, best_dimension] == feature_a_unique[i]:
            Dv.append(D[j])
    D_v = np.array(Dv)
    if len(D_v) == 0:
        newNode.isLeaf = True
        counter = Counter(label)
        counter_values = list(counter.values())
        counter_label = list(counter)
        max = counter_values[0]
        newNode.label = counter_label[0]
        for i in range(len(counter_values)):
            if counter_values[i] > max:
                max = counter_values[i]
                newNode.label = counter_label[i]
        node.addNode(feature_a_unique[i], newNode)
    else:
        copyA = A[:]
        copyA[best_dimension] = 0
        node.addNode(feature_a_unique[i], self.TreeGenerate(D_v, copyA))


return node
```

3.train函数和predict函数
决策树的构建已经完成，接下来构造train函数，先生成根节点，再通过递归调用生成决策树。
Predict函数则将预测的样本类型和真实值进行对比，如果预测正确，count+=1,最后用count/len(D)得到准确率。

```
#train函数可以做一些数据预处理（比如Dataframe到numpy矩阵的转换，提取属性集等），并调用TreeGenerate函数来递归地生成决策树
    def train(self, D):
        D = np.array(D)
        A = []
        for i in range(D.shape[1]-1):
            A.append(1)
        for i in range(len(A)):
            self.possible_value[i] = np.unique(D[:,i])
        self.tree_root = self.TreeGenerate(D, A)

        pass
#predict函数对测试集D进行预测， 并输出预测准确率（预测正确的个数 / 总数据数量）
    def predict(self, D):
#         D = np.array(D) # 将Dataframe对象转换为numpy矩阵（也可以不转，自行决定做法）
#         #对于D中的每一行数据d，从当前结点x=self.tree_root开始，当当前结点x为分支结点时，
#         #则搜索x的划分属性为该行数据相应的属性值的孩子结点（即x=x.children[d[x.index]]），不断重复，
#         #直至搜索到叶结点，该叶结点的label就是数据d的预测label
        D = np.array(D)
        pre = []
        count = 0
        for i in range(D.shape[0]):
            d = D[i]
            x = self.tree_root
            while x.isLeaf == False:
                x = x.children[d[x.index]]
            pre.append(x.label)
            if x.label == d[-1]:
                count = count + 1
        accuracy = count/D.shape[0]
        print("accuracy:", accuracy)
        pass
```

验证：
利用train中的样本构建一颗决策树，再用test进行预测，得到预测准确率。

预测并输出结果

```
train_frame = pd.read_csv('train_titanic.csv')
dt = DTree()

# 构建决策树
dt.train(train_frame)

# 利用构建好的决策树对测试数据集进行预测，输出预测准确率（预测正确的个数 / 总数据数量）
test_frame = pd.read_csv('test_titanic.csv')
dt.predict(test_frame)
```

```
accuracy: 0.8316831683168316
```

心得体会：

在构建决策树的过程中，遇到了不少困难。

1. 第一次构建的时候，输出的准确率是0.78，以为已经成功地构建了决策树，后来检查的时候发现决策树根本没有叶子结点（label全是-1），也就说明了根本就没有成功地构建起一棵决策树，不得不和同学讨论，推倒重建，修改后的代码经过了验证，可以构建一棵决策树了。

2. 由于之前写过了C4.5和cart算法，我在网上查到决策树的预测准确率大概在70%-90%左右，出于好奇心于是我又利用C4.5和cart算法又实现了一次，结果如下：（两种算法结果基本一致）

预测并输出结果

```
train_frame = pd.read_csv('train_titanic.csv')
dt = DTree()

# 构建决策树
dt.train(train_frame)

# 利用构建好的决策树对测试数据集进行预测，输出预测准确率（预测正确的个数 / 总数据数量）
test_frame = pd.read_csv('test_titanic.csv')
dt.predict(test_frame)
```

```
accuracy: 0.8415841584158416
```

经过计算发现C4.5算法，cart算法和ID3算法在预测率上并没有明显的区别（只相差了0.01）经过一系列的探讨和上网查找，我从实验一函数的实现处得到了灵感，在最后输出最大信息增益时，ID3和C4.5算法并没有明显的差异，说明二者在处理该样本的时候可能并没有明显的区别。

```
信息熵是： 0.7709131174627031
ID3:
best_gain: 0.10750711887455178
best_dimension: 0
C4.5:
best_gain: 0.10750711887455178
best_dimension: 0
GINI:
best_gini_index: 0.2964915724641573
best_dimension: 0
best_split: 0
```

（两者的best_gain和best_dimension一样）

而Cart算法也没有较大区别的原因可能是样本数据比较便于划分的原因，没有极端情况，所以三种方法没有明显的区别。经过上网查找，我得到了这三种算法的差异和优缺点：**（总而言之，ID3容易造成过拟合，Cart计算量相对较小，ID3和C4.5适合小样本的计算，如果样本过多可能造成过拟合，而Cart适合大样本量，对于小样本量，泛化误差会比较大，各有利弊。）**

1.划分标准的差异：ID3 使用信息增益偏向特征值多的特征，C4.5 使用信息增益率克服信息增益的缺点，偏向于特征值小的特征，CART 使用基尼指数克服 C4.5 需要求 log 的巨大计算量，偏向于特征值较多的特征。

2.使用场景的差异：ID3 和 C4.5 都只能用于分类问题，CART 可以用于分类和回归问题；ID3 和 C4.5 是多叉树，速度较慢，CART 是二叉树，计算速度很快。

3.样本数据的差异：ID3 只能处理离散数据且缺失值敏感，C4.5 和 CART 可以处理连续性数据且有多种方式处理缺失值；从样本量考虑的话，小样本建议 C4.5、大样本建议 CART。C4.5 处理过程中需对数据集进行多次扫描排序，处理成本耗时较高，而 CART 本身是一种大样本的统计方法，小样本处理下泛化误差较大。

4.样本特征的差异：ID3 和 C4.5 层级之间只使用一次特征，CART 可多次重复使用特征。