实　　验　　报　　告

学号	21050005076	姓名	田晶怡	专业班级	2021 级智能科学与技术
课程名称	机器学习			学期	年 季学期
任课教师	仲国强	完成日期	2024/5/14	上机课时间	2024/5/13
实 验 名 称	特征选择与分类——隐形眼镜数据集				

一、 实验目的

1. 将数据集拆分成训练集（前 20 个）和测试集（后 4 个）。
2. 用前向搜索算法来选择最优特征集合。
3. 使用最优特征集合训练模型并在测试集上进行测试，输出测试集的准确率。

二、 实验原理：

特征选择

定义：我们能用很多属性描述一个西瓜，例如色泽、根蒂、敲声、纹理、触感等，但有经验的人往往只需看看根蒂、听听敲声就知道是否好瓜。换言之，对一个学习任务来说，给定属性集，其中有些属性可能很关键、很有用，另一些属性则可能没什么用。

对此，我们将属性称为“特征”(feature)，对当前学习任务有用的属性称为“**相关特征**”(relevant feature)、没什么用的属性称为“**无关特征**”(irrelevantfeature)。从给定的特征集合中选择出相关特征子集的过程，称为“**特征选择**”(feature selection)。

1.如何根据评价结果获取下一个候选特征子集?

2.如何评价候选特征子集的好坏?

第一个环节是“子集搜索" (subset search)问题。给定特征集合{a1,a2...,ad}，我们可将每个特征看作一个候选子集，对这d个候选单特征子集进行评价，假定{a2}最优，于是将{a2}作为第一轮的选定集；然后，在上一轮的选定集中加入一个特征，构成包含两个特征的候选子集，假定在这d-1个候选两特征子集中{a2，a4} 最优，且优于{a2},于是将{a2,a4}作为本轮的选定集；...假定在第k+1轮时，最优的候选(k+1)特征子集不如上一轮的选定集，则停止生成候选子集，并将上一轮选定的k特征集合作为特征选择结果。这样逐渐增加相关特征的策略称为**“前向”(forward)搜索**.

第二个环节是“子集评价”(subset evaluation)问题。给定数据集D，假定D中第i类样本所占的比例为pi (i= 1,2....1)。为便于讨论，假定样本属性均为离散型。对属性子集A，假定根据其取值将D分成了V个子集 $\{D^1, D^2 ... D^V\}$，每个子集中的样本在A上取值相同，于是我们可计算属性子集A的**信息增益**:

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

其中信息熵定义为:

$$Ent(D) = - \sum_{i=1}^{|y|} p_k \log_2 p_k,$$

信息增益Gain(A)越大，意味着特征子集A包含的有助于分类的信息越多。于是,对每个候选特征子集，我们可基于训练数据集D来计算其信息增益，以此作为评价准则。

三、 实验内容：（关键代码展示等）

关键代码可以分为三个部分，分别是信息增益计算，前向搜索算法和朴素贝叶斯模型：

1.信息增益的计算：

```
#计算信息熵
def getEntropy(datacol):
    p = pd.value_counts(datacol) / len(datacol)
    e = sum((-1) * p * np.log2(p))
    return e

#计算信息增益
def getGain(subset,dataset):
    dataA = dataset.groupby(subset)  #对属性子集A，假定根据其取值将D分成了V个子集
    EntDv = dataA.apply(lambda x:getEntropy(x[4])) #去除编号后第5列为目标变量
    p = dataA.size() / len(dataset[subset])
    GainA = getEntropy(dataset[4]) - sum(p*EntDv)
    return GainA
```

这两个函数根据公式 $Ent(D) = - \sum_{i=1}^{|y|} p_k \log_2 p_k,$ 和 $Gain(A) = Ent(D) - \sum_{v-1}^{V} \frac{|D^v|}{|D|} \mathrm{Ent}(D^v),$ 编写即可。但需要注意的是，计算属性子集的信息增益，但是函数的输入参数为整个数据集，所以需要我们根据取值将数据集划分为几个子集，这里使用groupby()函数实现；然后要实现每个子集中的样本在 A 上取值相同的话，则使用 apply()函数将我们自设的匿名函数（实际上就是求信息熵），对每个子集使用，即可得到 EntDv。这里一直使用[4]即数据集中的第 5 列，是因为这一列是输出变量，即目标变量。

2.前向搜索函数：

```
#前向搜索
def forward(dataset):
    subset = [] #特征子集
    subsetGain = 0 #特征自己的增益
    for i in range(1,5): #选择特征个数，数据集共4个特征
        maxGain = 0
        maxSubset = []  #最优候选特征子集
        for j in range(0,4): #逐步加入特征
            if j in subset:
                continue
            else:
                newSubset = copy.deepcopy(subset)
                newSubset.append(j)
                Gain = getGain(newSubset,dataset)
                if Gain > maxGain:
                    maxGain = Gain
                    maxSubset = newSubset
        if maxGain > subsetGain: #最优的候选(k+1)特征子集不如上一轮的选定集，则停止生成候选子集
            subsetGain = maxGain
            subset = maxSubset
        else:
            break
    return subset,subsetGain
```

根据实验原理，前向搜索的思想就是特征的叠加。这里用 maxGain 和 maxSubet 记录最优候选，当最优候选结果不如上一轮子集的信息增益事停止迭代。

3.朴素贝叶斯分类器：

```
#使用朴素贝叶斯训练模型
# 计算高斯分布概率密度函数
def gaussian_pdf(x, mean, var):
    return 1 / np.sqrt(2 * np.pi * var) * np.exp(- (x - mean)**2 / (2 * var))

#训练模型
def bayesModelTrain(train_x,train_y):
    #计算先验概率
    prior_p = {}
    for i in np.unique(train_y):
        prior_p[i] = (np.sum(train_y == i) + 1) / (train_y.size + np.unique(train_y).size)

    likelihood = {}  # 存储该类别在每个特征上的似然概率
    for i in np.unique(train_y):
        likelihood[i] = {}
        data = train_x[train_y == i]
        for col in range(train_x.shape[1] - 1):
            for j in np.unique(train_x[:, col]):#添加拉普拉斯平滑
                likelihood[i][j] = ((data[data[:, col] == j]).shape[0] + 1) / (
                        data.shape[0] + np.unique(train_x[:, 0]).size)
    class_type = np.unique(train_y)
    return prior_p, likelihood, class_type

#模型分类测试
def bayestest(test_x,test_y,prior_p, likelihood, class_type):
    acc = 0
    predict = []
    for x, lable in zip(test_x, test_y):  # 测试样本
        max_p = 0
        y = 0
        for category in class_type:
            p = prior_p[category]
            for col,feature in enumerate(x):
                #p *= likelihood[category][feature]
                if col in likelihood[category].keys():  # 判断是否存在该特征
                        info = likelihood[category][col]
                        if isinstance(info, tuple) and len(info) == 2:  # 判断是否为合法的元组
                            mean, var = info
                            p *= gaussian_pdf(feature, mean, var)
            if max_p < p:
                max_p = p
                y = category
        predict.append(y)
        if y == lable:
            acc += 1
    return acc / test_x.shape[0], predict
```

根据实验 6 实现朴素贝叶斯分类器，先求先验概率 P(c)，这里使用布尔值索引 train_y== i 更加简便快捷。使用 np.unique(train_y)可以直接去重得出所有取值。同时考虑到有的属性在训练集

中未出现，导致求出的概率为 0，使用拉普拉斯修正进行数据平滑。然后求出条件概率，即不同属性在不同类别中出现的概率。需要保存两个键，键可能是负数或字符，使用嵌套的字典。三层循环，最外层枚举类型，第二层枚举属性列，最内层枚举该属性的所有取值。同样使用拉普拉斯修正进行数据平滑。

然后根据贝叶斯判定准则：$$h_{nb}(\boldsymbol{x}) = \arg\max_{c \in \mathcal{Y}} P(c) \prod_{i=1}^{d} P(x_i \mid c)$$，对测试集进行测试即可。这一部分我增加了一个高斯分布概率密度，然后在测试函数中来估计特征的似然概率，主要是为了提高模型的准确度，但实际效果并未提升，或许是因为数据集太简单只能达到 75%的精度，所以说这个部分可有可无。

四、 实验结果：（要有实验结果的截图）

```
================== RESTART: E:\桌面\机器学习\实验10-特征选择与分类\test10.py ==
===============
info_gain:  1.3709505944546687
['tear', 'astigmatic', 'age', 'spectacle']
[3, 2, 0, 1]
训练开始
贝叶斯分类器的先验概率为:
{1.0: 0.21739130434782608, 2.0: 0.21739130434782608, 3.0: 0.5652173913043478}
训练完成
模型评估开始
朴素贝叶斯分类器的准确度为75.00 %
结果是:  [3.0, 3.0, 3.0, 3.0]
模型评估结束
>>>
```

五、 心得体会：（实验遇到的问题，解决方法，心得体会）

问题：在向前搜索的过程中如何将上一轮的特征子集保存下来？

解决方法：使用 copy.deepcopy()函数,返回一个与原对象完全相同的新的对象，通过递归地调用自身的 deepcopy 方法来实现对内部数据的复制。这种深拷贝确保了即使原始对象内部的某些部分被修改，也不会影响到新对象。

问题：如何实现“根据取值将 D 分成 V 个子集”？

解决方法：这里想到的是直接使用 pandas 库的 groupby()函数，但容易忽略的一个问题是，数据读入时是调用的 numpy 库中的 loadtxt()函数，所以直接运行的话就会有如下报错：

```
================== RESTART: E:\桌面\机器学习\实验10-特征选择与分类\test10.py ==
===============
Traceback (most recent call last):
  File "E:\桌面\机器学习\实验10-特征选择与分类\test10.py", line 104, in <module>
    subset,gain = forward(traindata)
  File "E:\桌面\机器学习\实验10-特征选择与分类\test10.py", line 51, in forward
    Gain = getGain(newSubset,dataset)
  File "E:\桌面\机器学习\实验10-特征选择与分类\test10.py", line 32, in getGain
    dataA = dataset.groupby(subset)  #对属性子集A，假定根据其取值将D分成了V个子
集
AttributeError: 'numpy.ndarray' object has no attribute 'groupby'
>>>
```

所以需要增加一步，将数据集 dataset 转化成 pandas 库的数据结构：

```
traindata = pd.DataFrame(traindata)
```

问题：接续上个问题，修改了数据结构后，获取信息熵的函数报错。

```
Traceback (most recent call last):
  File "C:\Users\santa\AppData\Local\Programs\Python\Python310\lib\site-packages
\pandas\core\groupby\groupby.py", line 1353, in apply
    result = self._python_apply_general(f, self._selected_obj)
  File "C:\Users\santa\AppData\Local\Programs\Python\Python310\lib\site-packages
\pandas\core\groupby\groupby.py", line 1402, in _python_apply_general
    values, mutated = self.grouper.apply(f, data, self.axis)
  File "C:\Users\santa\AppData\Local\Programs\Python\Python310\lib\site-packages
\pandas\core\groupby\ops.py", line 767, in apply
    res = f(group)
  File "E:\桌面\机器学习\实验10-特征选择与分类\test10.py", line 33, in <lambda>
    EntDv = dataA.apply(lambda x:getEntropy(x[4])) #去除编号后第5列为目标变量
  File "E:\桌面\机器学习\实验10-特征选择与分类\test10.py", line 21, in getEntrop
y
    if i[-1] not in label_num.keys():
TypeError: 'float' object is not subscriptable
```

解决方法：因为前向搜索做的是特征的逐个添加，所以我们每次可以只计算单个特征列的信息增益作为评判标准，这里可以使用 pandas 库的 value_counts()函数，它可以用来处理单列 Series 对象，这样我们就可以在特征选择这一部分统一数据集为 pandas 数据结构。

改前(报错代码)：

```
#计算信息熵
def getEntropy(dataset):
    #求总样本数
    num = len(dataset)
    label_num = {}
    #统计共多少个类，以及每个类出现的次数
    for i in dataset:
        i = list(i)
        if i[-1] not in label_num.keys():
            label_num[i[-1]] = 0
        label_num[i[-1]] += 1
    e = 0
    for key in label_num:
        p = float(label_num[key]) / num
        e -= p * log2(p)
    return e

#计算信息增益
def getGain(subset,dataset):
    data = pd.DataFrame(dataset)
    dataA = data.groupby(subset)  #对属性子集A，假定根据其取值将D分成了V个子集
    EntDv = dataA.apply(lambda x:getEntropy(x[4])) #去除编号后第5列为目标变量
    p = dataA.size() / len(dataset[subset])
    GainA = getEntropy(dataset[4]) - np.sum(p*EntDv)
    return GainA
```

改后：

```
#计算信息熵
def getEntropy(datacol):
    p = pd.value_counts(datacol) / len(datacol)
    e = sum((-1) * p * np.log2(p))
    return e

#计算信息增益
def getGain(subset,dataset):
    dataA = dataset.groupby(subset)  #对属性子集A，假定根据其取值将D分成了V个子集
    EntDv = dataA.apply(lambda x:getEntropy(x[4])) #去除编号后第5列为目标变量
    p = dataA.size() / len(dataset[subset])
    GainA = getEntropy(dataset[4]) - sum(p*EntDv)
    return GainA
```

可以看到，代码也简洁了很多，报错解决。

问题：实验结果的准确度好像比较低。

```
========== RESTART: E:\桌面\机器学习\实验10-特征选择与分类\test10.py ===
==========
info_gain:  1.3709505944546687
[3, 2, 0, 1]
训练开始
贝叶斯分类器的先验概率为:
{1.0: 0.21739130434782608, 2.0: 0.21739130434782608, 3.0: 0.5652173913043478}
训练完成
模型评估开始
朴素贝叶斯分类器的准确度为75.00 %
结果是:  [3.0, 3.0, 3.0, 3.0]
模型评估结束
>>>
```

关于可优化部分做以下分析：

1.特征选择：确保选择了与分类任务最相关的特征——已经使用了前向搜索算法进行特征选择。

2.模型训练：确保在训练模型时正确地处理了每个特征的概率——已经使用了拉普拉斯平滑来避免概率为零的情况。

3.模型测试：确保在测试模型时正确地计算了每个样本的类别概率，并选择了概率最大的类别作为预测。

好像在实验的要求下，已经没有可以优化的地方了。于是我在模型测试函数中使用了高斯分布概率密度函数来估计特征的似然概率：

```
# 计算高斯分布概率密度函数
def gaussian_pdf(x, mean, var):
    return 1 / np.sqrt(2 * np.pi * var) * np.exp(- (x - mean)**2 / (2 * var))

for col,feature in enumerate(x):
    #p *= likelihood[category][feature]
    if col in likelihood[category].keys():  # 判断是否存在该特征
            info = likelihood[category][col]
            if isinstance(info, tuple) and len(info) == 2:  # 判断是否为合法的元组
                mean, var = info
                p *= gaussian_pdf(feature, mean, var)
```

结果如下：

```
========== RESTART: E:\桌面\机器学习\实验10-特征选择与分类\test10.py ===
==========
info_gain:  1.3709505944546687
['tear', 'astigmatic', 'age', 'spectacle']
[3, 2, 0, 1]
训练开始
贝叶斯分类器的先验概率为:
{1.0: 0.21739130434782608, 2.0: 0.21739130434782608, 3.0: 0.5652173913043478}
训练完成
模型评估开始
朴素贝叶斯分类器的准确度为75.00 %
结果是:  [3.0, 3.0, 3.0, 3.0]
模型评估结束
>>>
```

从结果来看好像没什么改变，可能 75%是本实验要求模型下能达到的最好精度了。