山东大学 计算机科学与技术 学院

机器学习（双语） 课程实验报告

学号：	姓名：	班级：
实验题目：Experiment 4: Naïve Bayes		
实验学时：4	实验日期： 2022/11/2	

实验目的：

1. 实现实验指导书中朴素贝叶斯的相关内容；
2. 学习使用 MATLAB、Python 等工具进行实验；
3. 通过选取拥有不同样本数量的训练集，尝试预测在测试集上得到的结果，并对结果进行分析。

硬件环境：

Inter（R） Core（TM） i7-8750H

RAM：16.0 GB

软件环境：

Visual Studio Code

版本：1.67.2（user setup）

OS：Windows_NT x64 10.0.19044

Python 3.9.7

numpy 1.20.3

matplotlib 3.4.3

实验步骤与内容：

1. 载入数据，记录训练集中各种数据可能出现的结果个数：

```
num = []
for i in range(training_data.shape[1]):
    num.append(training_data[:, i].max())
num
```

```
[2, 4, 3, 3, 2, 1, 2, 2, 4]
```

2. 对于每种最终的预测结果，计算其在所有预测结果中出现的概率，并计算在该种预测结果下，前方 n-1 种影响因素出现的概率：

```python
def cal_prob(data):
    m, n = data.shape
    # print(training_data.shape)

    p_1, p_2 = [], []
    pred = data[:, -1]
    for i in range(num[-1]+1):    # 每种预测结果
        p_1.append((pred == i).sum() / m) # 该种预测结果的概率
        temp_1 = []
        for j in range(n):    # n-1种影响因素
            temp_2 = []
            temp_data = np.array([data[:, j][k] for k in range(m) if pred[k] == i])
            for k in range(num[j]+1):
                if ((pred == i).sum() != 0):
                    temp_2.append((temp_data == k).sum() / (pred == i).sum())
                else:
                    temp_2.append(0)
            temp_1.append(temp_2)
        p_2.append(temp_1)
    return p_1, p_2
```

若有某种预测结果下，并未出现该种影响因素，则直接将对应概率设为 0；

3. 给定训练集、测试集，在通过上图 cal_prob()方法算得相应概率后，根据测试集中对应数据，对其结果进行预测：

```python
def pred(train_data, test_data):
    p_1, p_2 = cal_prob(train_data)
    # print(p_1, p_2)
    m, n = test_data.shape
    cnt = 0
    for i in range(m):
        temp = []
        for j in range(test_data[:, -1].min(), test_data[:, -1].max()+1):
            pred_p = p_1[j]
            for k in range(n):
                pred_p *= p_2[j][k][test_data[i][k]]
            temp.append(pred_p)
        if (temp.index(max(temp)) == test_data[i][-1]):
            cnt += 1
    return cnt / m
```

该方法最终返回在给定训练集测试集下，对训练集中预测结果正确的比例。

4. 由此可由训练集中数据得到预测正确率：

```python
res = pred(training_data, test_data)
```

5. 实验第二部分为随机选取测试集中部分数据，再次进行预测，比较实验结果，因此，编写 get_train_data()方法来随机选取相应比例的训练数据：

```python
def get_train_data(training_data, prob=1):
    temp = training_data.copy()
    np.random.shuffle(temp)
    return temp[:int(prob*temp.shape[0])]
```

6. 分别选取测试集中比例为 50%、1%、0.5%、0.1%的数据进行预测：

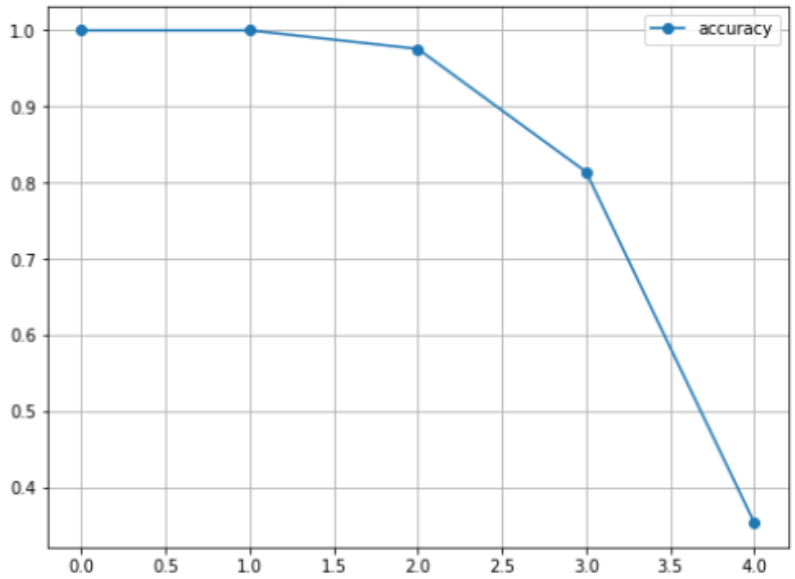
```
train_1 = get_train_data(training_data, prob=0.5)
train_2 = get_train_data(training_data, prob=0.01)
train_3 = get_train_data(training_data, prob=0.005)
train_4 = get_train_data(training_data, prob=0.001)
```

```
res_1 = pred(train_1, test_data)
res_2 = pred(train_2, test_data)
res_3 = pred(train_3, test_data)
res_4 = pred(train_4, test_data)
```

输出预测结果：

```
0.9996621621621622
0.9753378378378378
0.8138513513513513
0.3533783783783784
```

7. 预测结果如图所示：



可见，随着训练样本数目的减少，预测准确率也在减少，这与朴素贝叶斯通过结合先验概率和后验概率进行计算的特点相吻合。

结论分析与体会：

1. 在实验前，需要充分理解使用 matlab、python 等工具，才能更好地进行实验，实现实验中的各个步骤。
2. 在实验中，需要理解掌握朴素贝叶斯的实现原理，掌握其深层含义，结合实验指导书，才能更好地完成实验；
3. 朴素贝叶斯结合先验概率和后验概率进行计算，所需参数少，算法实现简单，有着较高的准确率。

附录：程序源代码

```python
# %%
import numpy as np
import matplotlib.pyplot as plt

# %%
training_data = np.loadtxt('data4/training_data.txt', dtype=np.int32)  # 载入数据
test_data = np.loadtxt('data4/test_data.txt', dtype=np.int32)

# %%
num = []
for i in range(training_data.shape[1]):
    num.append(training_data[:, i].max())
num

# %%
def cal_prob(data):
    m, n = data.shape
    # print(training_data.shape)

    p_1, p_2 = [], []
    pred = data[:, -1]
    for i in range(num[-1]+1):   # 每种预测结果
        p_1.append((pred == i).sum() / m) # 该种预测结果的概率
        temp_1 = []
        for j in range(n):   # n-1 种影响因素
            temp_2 = []
            temp_data = np.array([data[:, j][k] for k in range(m) if pred[k] == i])
            for k in range(num[j]+1):
                if (pred == i).sum() != 0:
                    temp_2.append((temp_data == k).sum() / (pred == i).sum())
                else:
                    temp_2.append(0)
            temp_1.append(temp_2)
        p_2.append(temp_1)
    return p_1, p_2

# %%
def pred(train_data, test_data):
    p_1, p_2 = cal_prob(train_data)
    # print(p_1, p_2)
    m, n = test_data.shape
```

```
    cnt = 0
    for i in range(m):
        temp = []
        for j in range(test_data[:, -1].min(), test_data[:, -1].max()+1):
            pred_p = p_1[j]
            for k in range(n):
                pred_p *= p_2[j][k][test_data[i][k]]
            temp.append(pred_p)
        if temp.index(max(temp)) == test_data[i][-1]:
            cnt += 1
    return cnt / m

# %%
res = pred(training_data, test_data)

# %%
def get_train_data(training_data, prob=1):
    temp = training_data.copy()
    np.random.shuffle(temp)
    return temp[:int(prob*temp.shape[0])]

# %%
train_1 = get_train_data(training_data, prob=0.5)
train_2 = get_train_data(training_data, prob=0.01)
train_3 = get_train_data(training_data, prob=0.005)
train_4 = get_train_data(training_data, prob=0.001)

# %%
res_1 = pred(train_1, test_data)
res_2 = pred(train_2, test_data)
res_3 = pred(train_3, test_data)
res_4 = pred(train_4, test_data)

# %%
print(res_1)
print(res_2)
print(res_3)
print(res_4)

# %%
x = np.arange(0, 5)
y = np.array([res, res_1, res_2, res_3, res_4])

# %%
```

```
plt.figure(figsize=(8, 6))
plt.plot(x, y, marker='o', label='accuracy')
plt.grid()
plt.legend()
plt.show()

# %%
```