1 朴素贝叶斯

[1]:
```
import numpy as np
import pandas as pd
from scipy import stats
```

1.1 正确地读取数据

注意原始数据文件的格式，对其进行正确地处理后读入两个 DataFrame：adult_data_df 是训练集，adult_test_df 是测试集。DataFrame 中名为 "50K" 的列为标签（即分类）。

读取数据的方法与上个实验（决策树算法）完全相同。

[2]:
```
col_names = ['age', 'workclass', 'fnlwgt', 'education', 'education-num',␣
 ↪'marital-status', 'occupation', 'relationship', 'race', 'sex',␣
 ↪'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', '50K']
adult_data_df = pd.read_csv('dataset/adult.data', index_col=False, header=None,␣
 ↪names=col_names, sep=', ', engine='python')

adult_test_df = pd.read_csv('dataset/adult.test', skiprows=[0],␣
 ↪index_col=False, header=None, names=col_names, sep=', ', engine='python')
adult_test_df['50K'] = adult_test_df['50K'].map(lambda x: x[:-1])  # 去除行末的
点
```

1.2 补充缺失值

通过对数据的基本观察得知，缺失值所在的列均为离散属性，因此只需要对离散缺失值进行补全即可，本例数据集上无需考虑连续型数据的补全。我采用的方法是使用该列出现次数最多的值（即众数）代替缺失值。

补充缺失值的方法与上个实验（决策树算法）完全相同。

[3]:
```
# 补充缺失值，
print('[adult.data]')
mode_df = adult_data_df.mode()  # 众数
for col in adult_data_df:
    if '?' in adult_data_df[col].tolist():
        missing_count = adult_data_df[col].value_counts()['?']    # 缺失值的个数
        adult_data_df[col] = adult_data_df[col].replace('?', mode_df[col][0])
        print('{}: {} missing values are replaced with "{}"'.format(col,␣
 ↪missing_count, mode_df[col][0]))

print('--------------------------------')
print('[adult.test]')
mode_df = adult_test_df.mode()  # 众数
for col in adult_test_df:
    if '?' in adult_test_df[col].tolist():
        missing_count = adult_test_df[col].value_counts()['?']    # 缺失值的个数
        adult_test_df[col] = adult_test_df[col].replace('?', mode_df[col][0])
```

```
        print('{}: {} missing values are replaced with "{}"'.format(col,␣
↪missing_count, mode_df[col][0]))
```

```
[adult.data]
workclass: 1836 missing values are replaced with "Private"
occupation: 1843 missing values are replaced with "Prof-specialty"
native-country: 583 missing values are replaced with "United-States"
-------------------------------
[adult.test]
workclass: 963 missing values are replaced with "Private"
occupation: 966 missing values are replaced with "Prof-specialty"
native-country: 274 missing values are replaced with "United-States"
```

1.3 预测和测试

对于测试集中的每个样本，使用朴素贝叶斯方法进行预测，然后与标签比对，并统计准确率。

[4]:
```
# 连续型属性
continuous_attrs = {'age', 'fnlwgt', 'education-num', 'capital-gain',␣
↪'capital-loss', 'hours-per-week'}

# 计算概率
def probability(df, attr, value):
    """
    计算数据集中某属性为某值的概率。
    Params:
        df: 数据集。
        attr_: 属性名。
        value: 属性值。
    Return:
        对于离散型属性，返回给定属性中值等于给定值的比例；
        对于连续型属性，返回对应高斯分布的概率密度函数值。
    """
    attr_series = df[attr]
    if attr in continuous_attrs:  # 连续型属性
        mean = attr_series.mean()  # 期望
        var = attr_series.var()    # 方差
        return stats.norm.pdf(value, loc=mean, scale=np.sqrt(var))  # 高斯分布的
概率密度
    else:  # 离散型属性
        return list(attr_series).count(value) / len(df)
```

[5]:
```
def predict(sample):
    """
    对一个样本进行预测。
    Params:
        sample: 待测样本。
    Returns:
```

```
        预测分类结果。
    """
    class_list = ['<=50K', '>50K']  # 所有类别
    max_prob = 0
    max_class = ''

    # 遍历所有可能的分类（本例中只有两种分类）
    for class_ in class_list:
        class_df = adult_data_df[adult_data_df['50K']==class_]  # 按类划分数据集
        prob = adult_data_df['50K'].value_counts().get('<=50K', 0) /␣
↪len(adult_data_df)  # 初始化为类的先验概率

        for attr in sample.index:
            if attr == '50K':  # 标签列不是属性，要跳过
                continue
            prob *= probability(class_df, attr, sample[attr])  # 累乘每个属性在数
据集中出现的概率

        if prob >= max_prob:
            max_prob = prob
            max_class = class_

    return max_class  # 返回概率最大的类作为预测结果
```

[6]:
```
correct_count = 0
for i in range(len(adult_test_df)):
    sample = adult_test_df.iloc[i]
    if predict(sample) == sample['50K']:
        correct_count += 1
```

[7]:
```
print('准确率：{:.3%}'.format(correct_count / len(adult_test_df)))
```

准确率：83.269%