贝叶斯实践

助教：李兵

5222023330043@smail.nju.edu

作业介绍

1.手动实现朴素贝叶斯分类器，在基础的西瓜数据集上进行实验。

2.探索使用拉普拉斯平滑对模型性能的影响。

3.尝试使用更多的复杂的数据集。

导入相关包

In [1]:
```
import numpy as np
import pandas as pd
from collections import defaultdict
from scipy.stats import norm
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.preprocessing import LabelEncoder
```

贝叶斯分类器的手动实现

在类中实现了朴素贝叶斯分类器的训练和预测函数以及拉普拉斯平滑的选项

In [2]:
```
class NaiveBayesClassifier:
    def __init__(self, laplace_smoothing=True):
        """
        朴素贝叶斯分类器
        :param laplace_smoothing: 是否使用拉普拉斯平滑
        """
        self.class_priors = {}  # 先验概率 P(C)
        self.likelihoods = {}   # 似然 P(x|C)
        self.feature_types = {} # 记录每个特征是离散还是连续
        self.laplace_smoothing = laplace_smoothing  # 是否使用拉普拉斯修正

    def fit(self, X, y):
        """
        训练朴素贝叶斯分类器
        X: 特征数据 (DataFrame)
        y: 目标标签 (Series)
        """
        self.classes = np.unique(y)
        self.feature_types = {col: 'categorical' if X[col].dtype == 'object' else 'continuous' for col in X.columns}

        # 计算先验概率 P(C)
        class_counts = y.value_counts().to_dict()
        total_samples = len(y)
        self.class_priors = {c: class_counts[c] / total_samples for c in self.classes}

        # 计算条件概率 P(x|C)
        self.likelihoods = {c: {} for c in self.classes}

        for c in self.classes:
            subset = X[y == c]
            for col in X.columns:
                if self.feature_types[col] == 'categorical':
                    # 计算 P(x|C) 使用拉普拉斯平滑
                    value_counts = subset[col].value_counts().to_dict()
                    total_count = len(subset) + (len(X[col].unique()) if self.laplace_smoothing else 0)
                    self.likelihoods[c][col] = {val: (value_counts.get(val, 0) + (1 if self.laplace_smoothing else 0)) / total_count for val in X[col].unique()}
                else:
                    # 计算正态分布参数（均值和标准差）
                    mean = subset[col].mean()
                    std = subset[col].std() if subset[col].std() > 0 else 1e-6  # 避免除零错误
                    self.likelihoods[c][col] = (mean, std)

    def predict(self, X):
        """
        对新数据进行预测
        X: 测试数据 (DataFrame)
        """
        predictions = []
        for _, row in X.iterrows():
            class_probs = {}
            for c in self.classes:
                # 计算后验概率 P(C|x) ∝ P(C) * P(x|C)
                prob = np.log(self.class_priors[c])  # 取对数防止数值下溢
                for col in X.columns:
                    if self.feature_types[col] == 'categorical':
                        prob += np.log(self.likelihoods[c][col].get(row[col], 1e-6))  # 避免 log(0)
                    else:
                        mean, std = self.likelihoods[c][col]
                        prob += np.log(norm.pdf(row[col], mean, std))
                class_probs[c] = prob
            predictions.append(max(class_probs, key=class_probs.get))  # 选择概率最大的类别
        return predictions
```

数据集的准备与处理

In [3]:
```
def load_dataset(name):
    """
    根据名称加载不同的数据集
    :param name: 数据集名称 ['watermelon', 'iris', 'adult', 'spam', 'wine']
    :return: X, y (DataFrame, Series)
    """
    if name == 'watermelon':  # 经典西瓜数据集
        data = pd.DataFrame([
            ['青绿', '蜷缩', '浊响', '清晰', '好瓜'],
            ['乌黑', '蜷缩', '沉闷', '清晰', '好瓜'],
            ['乌黑', '蜷缩', '浊响', '清晰', '好瓜'],
            ['青绿', '蜷缩', '沉闷', '清晰', '好瓜'],
```

```
            ['浅白', '蜷缩', '浊响', '清晰', '好瓜'],
            ['青绿', '稍蜷', '浊响', '清晰', '好瓜'],
            ['乌黑', '稍蜷', '浊响', '稍糊', '好瓜'],
            ['乌黑', '稍蜷', '浊响', '清晰', '好瓜'],
            ['乌黑', '稍蜷', '沉闷', '稍糊', '坏瓜'],
            ['青绿', '硬挺', '清脆', '清晰', '坏瓜'],
            ['浅白', '硬挺', '清脆', '模糊', '坏瓜']
        ], columns=['色泽', '根蒂', '敲声', '纹理', '好瓜'])
        X, y = data.iloc[:, :-1], data.iloc[:, -1]

    elif name == 'iris':  # 鸢尾花数据集
        iris = datasets.load_iris()
        X = pd.DataFrame(iris.data, columns=iris.feature_names)
        y = pd.Series(iris.target).astype(str)  # 转换为字符串

    elif name == 'wine':  # 葡萄酒质量数据集
        wine = datasets.load_wine()
        X = pd.DataFrame(wine.data, columns=wine.feature_names)
        y = pd.Series(wine.target).astype(str)

    elif name == 'adult':  # Adult (收入预测)
        url = "https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data"
        columns = ['age', 'workclass', 'fnlwgt', 'education', 'education-num', 'marital-status',
                   'occupation', 'relationship', 'race', 'sex', 'capital-gain', 'capital-loss',
                   'hours-per-week', 'native-country', 'income']
        data = pd.read_csv(url, names=columns, na_values=" ?", skipinitialspace=True)
        data.dropna(inplace=True)
        X, y = data.iloc[:, :-1], data.iloc[:, -1]

    elif name == 'spam':  # Spam (垃圾邮件分类)
        from sklearn.datasets import fetch_openml
        spam = fetch_openml(name="spambase", version=1)
        X = pd.DataFrame(spam.data)
        y = spam.target.astype(str)

    else:
        raise ValueError("未找到数据集，请选择 ['watermelon', 'iris', 'adult', 'spam', 'wine']")#可以尝试添加更多的数据集

    return X, y
```

数据集选择，参数选择与数据的预处理

In [4]:
```
# 选择数据集 & 是否使用拉普拉斯修正
dataset_name = 'wine'  # 'watermelon', 'adult', 'spam', 'wine'
use_laplace = True  # 是否使用拉普拉斯修正

# 加载数据集
X, y = load_dataset(dataset_name)

# 预处理
if X.select_dtypes(include=['object']).shape[1] > 0:
    X = X.apply(LabelEncoder().fit_transform)  # 对离散特征编码

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

训练，预测与评估

In [5]:
```
nb = NaiveBayesClassifier(laplace_smoothing=use_laplace)
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test)
print(f"=== {dataset_name} 数据集 ===")
print(f"分类准确率: {accuracy_score(y_test, y_pred):.4f}")
print("混淆矩阵:\n", confusion_matrix(y_test, y_pred))
print("分类报告:\n", classification_report(y_test, y_pred))
```

```
=== wine 数据集 ===
分类准确率: 1.0000
混淆矩阵:
 [[19  0  0]
 [ 0 21  0]
 [ 0  0 14]]
分类报告:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        21
           2       1.00      1.00      1.00        14

    accuracy                           1.00        54
   macro avg       1.00      1.00      1.00        54
weighted avg       1.00      1.00      1.00        54
```

替换不同的数据集进行测试（spam）

In [6]:
```
dataset_name = 'spam'  # 'watermelon', 'adult', 'spam', 'wine'
use_laplace = True  # 是否使用拉普拉斯修正
X, y = load_dataset(dataset_name)
if X.select_dtypes(include=['object']).shape[1] > 0:
    X = X.apply(LabelEncoder().fit_transform)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
nb = NaiveBayesClassifier(laplace_smoothing=use_laplace)
nb.fit(X_train, y_train)
y_pred = nb.predict(X_test)
print(f"=== {dataset_name} 数据集 ===")
print(f"分类准确率: {accuracy_score(y_test, y_pred):.4f}")
print("混淆矩阵:\n", confusion_matrix(y_test, y_pred))
print("分类报告:\n", classification_report(y_test, y_pred))
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_28748\1884281875.py:59: RuntimeWarning: divide by zero encountered in log
  prob += np.log(norm.pdf(row[col], mean, std))
```

```
=== spam 数据集 ===
分类准确率: 0.8146
混淆矩阵:
 [[583 221]
 [ 35 542]]
分类报告:
              precision    recall  f1-score   support

           0       0.94      0.73      0.82       804
           1       0.71      0.94      0.81       577

    accuracy                           0.81      1381
   macro avg       0.83      0.83      0.81      1381
weighted avg       0.85      0.81      0.82      1381
```

思考

朴素贝叶斯基于独立性假设，并没有考虑特征之间的依赖关系。请尝试任选（1）（2）中的一个进行实现，并回答（3）。

（1）TAN：用决策树找到特征之间的依赖关系，优化贝叶斯分类器的结构。

（2）SPODE：选择一个超级特征，让所有的子特征依赖于它，尝试实现，观察是否能提高模型性能。

（3）分析一下半朴素贝叶斯是否还能使用拉普拉斯平滑。