贝叶斯分类器实验报告

姓名：_____ 张韫译萱_____　学号：_____08023214_____

一、 实验题目

使用贝叶斯分类器实现 LIBSVM Data/dna 数据集分类。

1 实验任务

1. 使用朴素贝叶斯分类器实现分类。
2. 使用朴素贝叶斯分类器的拉普拉斯修正实现分类。

2 实验要求

1. Python 实现。
2. 不使用现成贝叶斯分类器库函数。

二、 实验原理

1 朴素贝叶斯分类器

朴素贝叶斯分类器是一种基于贝叶斯定理的概率分类方法，其核心思想是利用已知的先验概率和条件概率来计算后验概率，从而实现分类决策。

1.1 贝叶斯定理

贝叶斯定理描述了条件概率之间的关系，其数学表达式为：

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)} \tag{1}$$

其中：

- $P(C|X)$：后验概率，表示在观测到特征 X 的条件下，样本属于类别 C 的概率
- $P(X|C)$：似然概率，表示在类别 C 的条件下，观测到特征 X 的概率
- $P(C)$：先验概率，表示类别 C 出现的概率
- $P(X)$：证据因子，表示特征 X 出现的概率

1.2 条件独立性假设

朴素贝叶斯分类器的" 朴素" 之处在于假设各特征之间相互独立。设样本 X 由 n 个特征组成，即 $X = (x_1, x_2, \ldots, x_n)$，则条件独立性假设可表示为：

$$P(X|C) = P(x_1, x_2, \ldots, x_n|C) = \prod_{i=1}^{n} P(x_i|C) \tag{2}$$

1.3 分类决策

在分类时，我需要找到使后验概率 $P(C|X)$ 最大的类别。由于 $P(X)$ 对所有类别相同，可以忽略该项，因此分类决策规则为：

$$\hat{C} = \arg\max_{C} P(C) \prod_{i=1}^{n} P(x_i|C) \tag{3}$$

为避免概率连乘导致的数值下溢问题，实际计算中通常采用对数形式：

$$\hat{C} = \arg\max_{C} \left[\log P(C) + \sum_{i=1}^{n} \log P(x_i|C) \right] \tag{4}$$

1.4 概率估计

对于离散特征，先验概率和条件概率可通过极大似然估计得到：

先验概率：

$$P(C = c) = \frac{|D_c|}{|D|} \tag{5}$$

其中 $|D_c|$ 表示训练集中类别 c 的样本数，$|D|$ 表示训练集的总样本数。

条件概率：

$$P(x_i = v|C = c) = \frac{|D_{c,x_i=v}|}{|D_c|} \tag{6}$$

其中 $|D_{c,x_i=v}|$ 表示类别 c 中第 i 个特征取值为 v 的样本数。

2 朴素贝叶斯分类器的拉普拉斯修正

2.1 零概率问题

在实际应用中，极大似然估计可能导致某些条件概率为零的情况。例如，当训练集中某个类别下的某个特征取值从未出现过时，其条件概率估计值为零。这将导致整个后验概率为零，从而影响分类效果。

2.2 拉普拉斯修正

为解决零概率问题，我引入拉普拉斯修正。其基本思想是在每个特征取值的计数上加上一个平滑参数 α。

修正后的先验概率：

$$P(C = c) = \frac{|D_c| + 1}{|D| + N \cdot \alpha} \tag{7}$$

其中 K 表示类别的总数。

修正后的条件概率：

$$P(x_i = v|C = c) = \frac{|D_{c,x_i=v}| + \alpha}{|D_c| + N_i \cdot \alpha} \tag{8}$$

其中 N_i 表示第 i 个特征的可能取值数量。对于 DNA 数据集，每个特征是二值的，因此 $N_i = 2$。

拉普拉斯修正具有以下优点：

- 避免了零概率问题，确保所有条件概率估计值均为正
- 当训练样本充足时，拉普拉斯修正的影响较小，估计值接近极大似然估计

三、 过程与结果

1 实验流程

本实验的具体实现流程如下：

1.1 数据加载与预处理

我首先实现了 LIBSVM 格式数据的加载函数，将数据解析为特征矩阵和标签向量。由于 DNA 数据集的特征值为 -1 和 1，我将其二值化为 0 和 1 以便于概率计算。

1.2 朴素贝叶斯分类器实现

我实现了 `NaiveBayesClassifier` 类，包含以下主要方法：

1. `fit(X, y)`：训练方法，计算先验概率 $P(C)$ 和条件概率 $P(x_i|C)$
2. `predict_proba(X)`：预测样本属于各类别的概率
3. `predict(X)`：返回预测的类别标签
4. `score(X, y)`：计算分类准确率

分类器通过参数 `laplace_smoothing` 控制是否启用拉普拉斯修正，通过调整参数 α 设置平滑系数。

1.3 训练过程

在训练阶段，分类器执行以下步骤：

1. 统计各类别的样本数量，计算先验概率
2. 对于每个类别，统计每个特征各取值的出现次数
3. 根据是否启用拉普拉斯修正，计算条件概率

1.4 预测过程

在预测阶段，分类器对每个待预测样本：

1. 计算该样本属于各类别的对数后验概率
2. 选择对数后验概率最大的类别作为预测结果

2 朴素贝叶斯分类器的正确率

我首先使用未加拉普拉斯修正的朴素贝叶斯分类器进行实验，实验结果如表2所示。

数据集	准确率
训练集	94.14%
验证集	95.00%
测试集	93.34%

表 1: 朴素贝叶斯分类器（无拉普拉斯修正）实验结果

实验结果表明，朴素贝叶斯分类器在 DNA 数据集上取得了较好的分类效果，测试集准确率达到 93.34%。

3 朴素贝叶斯分类器的拉普拉斯修正的正确率

我使用拉普拉斯修正（$\alpha = 1$）的朴素贝叶斯分类器进行实验，结果如表3所示。

数据集	准确率
训练集	94.14%
验证集	94.83%
测试集	92.92%

表 2: 朴素贝叶斯分类器（拉普拉斯修正，$\alpha = 1$）实验结果

4 不同平滑参数的影响

我进一步探索了不同平滑参数 α 对分类性能的影响，结果如表4所示。

α	验证集准确率	测试集准确率
0.1	95.17%	93.42%
0.5	95.17%	93.09%
1.0	94.83%	92.92%
2.0	94.83%	92.92%
5.0	93.83%	92.66%

表 3: 不同平滑参数 α 的实验结果

实验结果表明，较小的平滑参数（如 $\alpha = 0.1$）在本数据集上表现更好。

5 结果可视化

图1展示了两种分类器在不同数据集上的准确率对比。

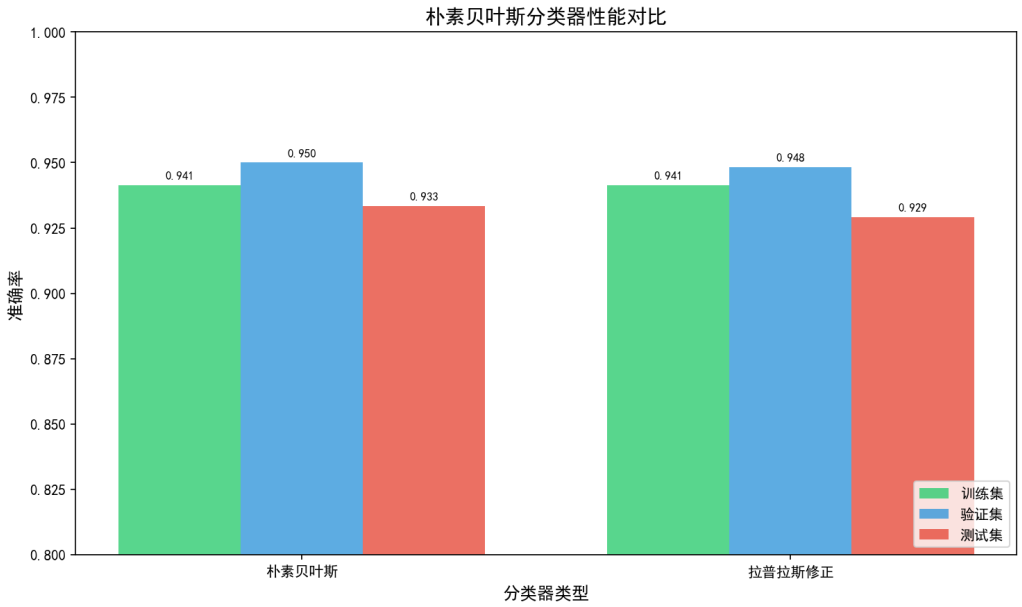

图 1: 朴素贝叶斯分类器性能对比

图2展示了平滑参数 α 对分类准确率的影响。

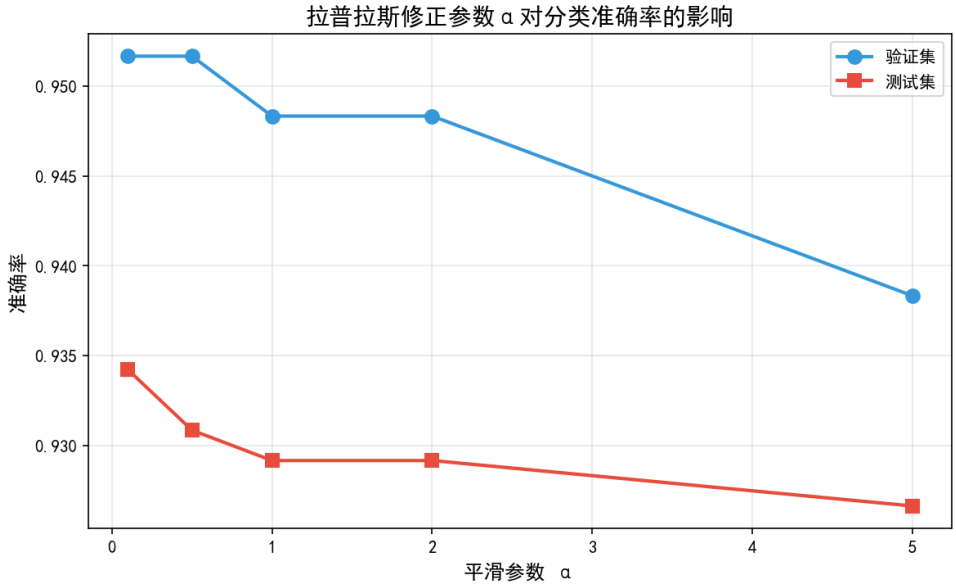

图 2: 拉普拉斯修正参数 α 对分类准确率的影响

6 结果分析

通过实验结果，我可以得出以下结论：

1. **朴素贝叶斯分类器的有效性**：在 DNA 数据集上，朴素贝叶斯分类器取得了超过 93% 的测试准确率，证明了该方法在高维二值特征分类问题上的有效性。

2. **拉普拉斯修正的影响**：在本实验中，拉普拉斯修正对分类性能的提升并不明显，甚至略有下降。这可能是因为 DNA 数据集的训练样本较为充足（1400 个样本），各特征取值的条件概率估计较为准确，零概率问题并不严重。与此同时，实验结果表明较小的平滑参数（$\alpha = 0.1$）能取得更好的效果。这是因为较大的平滑参数会过度平滑概率分布，降低模型对数据特征的敏感度。

四、 实验总结

通过本次实验，我系统地理解并实践了朴素贝叶斯分类器的原理与实现过程：在建模过程中，利用贝叶斯定理将分类问题转化为后验概率的计算，并据此进行分类决策；同时认识到朴素贝叶斯所依赖的特征条件独立性假设虽然在实际问题中往往难以完全满足，但在许多应用场景下仍能取得较好的分类效果。实验还表明，拉普拉斯修正能够有效缓解零概率问题，但其实际影响与数据规模和分布密切相关，在样本充足时作用可能不明显；为提高数值计算的稳定性，我采用对数概率形式避免了概率连乘引起的下溢问题。此外，平滑参数 α 作为重要超参数，对模型性能具有一定影响，需要借助验证集进行合理调优；最后，通过结合训练集、验证集和测试集对模型进行评估，我加深了对过拟合问题的理解，并能够更可靠地评价模型的泛化能力。本实验加深了我对概率分类方法的理解。

五、 代码

```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
matplotlib.rcParams['font.sans-serif'] = ['SimHei']
matplotlib.rcParams['axes.unicode_minus'] = False

def load_libsvm_data(filename, num_features=180):
    """
    加载LIBSVM格式的数据文件
    格式: label index1:value1 index2:value2 ...
    """
    labels = []
    features = []

    with open(filename, 'r') as f:
        for line in f:
            parts = line.strip().split()
            if not parts:
                continue

```

```
            # 第一个元素是标签
            label = int(parts[0])
            labels.append(label)

            # 初始化特征向量为0
            feature_vector = np.zeros(num_features)

            # 解析特征 index:value
            for item in parts[1:]:
                if ':' in item:
                    idx, val = item.split(':')
                    idx = int(idx) - 1  # LIBSVM索引从1开始
                    val = float(val)
                    if 0 <= idx < num_features:
                        feature_vector[idx] = val

            features.append(feature_vector)

    return np.array(labels), np.array(features)


class NaiveBayesClassifier:
    def __init__(self, laplace_smoothing=False, alpha=1.0):

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.classes = np.unique(y)
        n_classes = len(self.classes)

        # 将特征值二值化
        X_binary = (X > 0).astype(int)

        # 计算每个类别的先验概率
        for c in self.classes:
            class_count = np.sum(y == c)
            if self.laplace_smoothing:
                self.class_prior[c] = (class_count + self.alpha) / \
                    (n_samples + n_classes * self.alpha)
            else:
                self.class_prior[c] = class_count / n_samples

        # 计算每个特征在每个类别下的条件概率
        for c in self.classes:
            X_c = X_binary[y == c]
            n_c = X_c.shape[0]

            self.feature_prob[c] = {}
            for j in range(n_features):
```

```
                count_1 = np.sum(X_c[:, j] == 1)

                if self.laplace_smoothing:
                    prob_1 = (count_1 + self.alpha) / (n_c + 2 * self.alpha)
                else:
                    prob_1 = count_1 / n_c if n_c > 0 else 0.5

                self.feature_prob[c][j] = {1: prob_1, 0: 1 - prob_1}

        return self

    def predict_proba(self, X):
        """预测每个样本属于各类别的概率"""
        n_samples, n_features = X.shape
        X_binary = (X > 0).astype(int)

        log_probs = np.zeros((n_samples, len(self.classes)))

        for i, c in enumerate(self.classes):
            log_prior = np.log(self.class_prior[c] + 1e-10)

            log_likelihood = np.zeros(n_samples)
            for j in range(n_features):
                for k in range(n_samples):
                    feature_val = X_binary[k, j]
                    prob = self.feature_prob[c][j][feature_val]
                    log_likelihood[k] += np.log(prob + 1e-10)

            log_probs[:, i] = log_prior + log_likelihood

        # 转换为概率
        log_probs_max = np.max(log_probs, axis=1, keepdims=True)
        probs = np.exp(log_probs - log_probs_max)
        probs = probs / np.sum(probs, axis=1, keepdims=True)

        return probs

    def predict(self, X):
        """预测类别"""
        probs = self.predict_proba(X)
        return self.classes[np.argmax(probs, axis=1)]

    def score(self, X, y):
        """计算准确率"""
        predictions = self.predict(X)
        return np.mean(predictions == y)


```

```python
def main():
    """主函数"""
    # 加载数据
    y_train, X_train = load_libsvm_data('dna.scale.tr.txt')
    y_val, X_val = load_libsvm_data('dna.scale.val.txt')
    y_test, X_test = load_libsvm_data('dna.scale.txt')

    # 朴素贝叶斯分类器（无拉普拉斯修正）
    clf_nb = NaiveBayesClassifier(laplace_smoothing=False)
    clf_nb.fit(X_train, y_train)
    print(f"朴素贝叶斯测试集准确率: {clf_nb.score(X_test, y_test):.4f}")

    # 朴素贝叶斯分类器（拉普拉斯修正）
    clf_laplace = NaiveBayesClassifier(laplace_smoothing=True, alpha=1.0)
    clf_laplace.fit(X_train, y_train)
    print(f"拉普拉斯修正测试集准确率: {clf_laplace.score(X_test, y_test):.4f}")


if __name__ == "__main__":
    main()
```