南开大学
Nankai University

南　　开　　大　　学

计　算　机　学　院

机器学习实验报告

Lab4 基于朴素贝叶斯的手写数字分类实验

马溟怡

学号：2311061

专业：计算机科学与技术

2025 年 11 月 21 日

目录

一、 实验内容

(一) 基础任务：分层采样 + 自实现朴素贝叶斯分类器

1. 代码

基础任务

```
import numpy as np
from sklearn.model_selection import train_test_split
# ==========================================================
# 1. 数据加载与预处理 (semeion.data)
# ==========================================================
try:
    raw_data = np.loadtxt('semeion.data')
except FileNotFoundError:
    print("错误: 未找到 'semeion.data' 文件。请确保文件与代码在同一目录下。")

# 特征 X (前 256 列, 16x16 像素)
X = raw_data[:, :256]
# 标签 y (后 10 列, 独热编码)
y_onehot = raw_data[:, 256:]

# 将独热编码标签转换为类别标签 (0-9)
# np.argmax(axis=1) 找到每行中 '1' 出现的位置, 即为类别索引
y = np.argmax(y_onehot, axis=1)

# ==========================================================
# 2. 自写代码实现分层采样 (7:3 划分)
# ==========================================================

# test_size=0.3 表示测试集占 30% (即 7:3 划分)
# stratify=y 确保训练集和测试集中的类别比例与原始数据集保持一致
# random_state 用于固定随机种子, 保证结果可复现
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    stratify=y,
    random_state=42
)

print(f"原始数据集样本数: {len(X)}")
print(f"训练集样本数 (70%): {len(X_train)}")
print(f"测试集样本数 (30%): {len(X_test)}")


# ==========================================================
# 3. 自写代码实现朴素贝叶斯分类器 (高斯朴素贝叶斯)
```

```python
# ==========

class NaiveBayesClassifier:
    """自实现高斯朴素贝叶斯分类器 (Gaussian Naive Bayes)"""

    def fit(self, X, y):
        """训练模型: 计算先验概率 P(C_k) 和高斯分布参数 ( , ²)"""
        n_samples, n_features = X.shape
        self._classes = np.unique(y)  # 获取所有类别 (0, 1, ..., 9)
        n_classes = len(self._classes)

        # 存储均值、方差和先验概率
        self._mean = np.zeros((n_classes, n_features), dtype=np.float64)
        self._var = np.zeros((n_classes, n_features), dtype=np.float64)
        self._priors = np.zeros(n_classes, dtype=np.float64)

        for idx, c in enumerate(self._classes):
            X_c = X[y == c]  # 提取当前类别 c 的样本

            # 计算均值 ( )
            self._mean[idx, :] = X_c.mean(axis=0)

            # 计算方差 ( ²) 并添加平滑项 (1e-6) 避免方差为零导致计算错误
            self._var[idx, :] = X_c.var(axis=0) + 1e-6

            # 计算先验概率 P(C_k)
            self._priors[idx] = X_c.shape[0] / float(n_samples)

    def _pdf(self, class_idx, x):
        """高斯概率密度函数 (PDF) 计算 P(x_i | C_k)"""
        mean = self._mean[class_idx]
        var = self._var[class_idx]

        # 高斯 PDF 计算
        numerator = np.exp(- (x - mean) ** 2 / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        prob = numerator / denominator

        # === 关键修正: 确保概率值不会为 0, 避免 log(0) ===
        # 使用 np.maximum 将所有小于 1e-9 的值替换为 1e-9 (拉普拉斯平滑思想)
        # 这样 log(prob) 结果最大不会低于 log(1e-9) = -20.72
        return np.maximum(prob, 1e-9)

    def _predict(self, x):
        """对单个样本 x 进行预测, 返回预测类别"""
        # 计算每个类别的后验概率的对数, 以保证数值稳定性
        log_posteriors = []
        for idx in range(len(self._classes)):
```

```
            # 1. 计算先验概率的对数: log(P(C_k))
            log_prior = np.log(self._priors[idx])

            # 2. 计算类条件概率的对数和: Sum(log(P(x_i | C_k)))
            # P(x_i | C_k) 来自高斯PDF
            log_likelihood = np.sum(np.log(self._pdf(idx, x)))

            # 3. 计算后验概率的对数: log(P(C_k|x)) log(P(C_k)) + log(P(x|
                C_k))
            log_posterior = log_prior + log_likelihood
            log_posteriors.append(log_posterior)

        # 选择对数后验概率最大的类别索引, 并映射回类别值
        return self._classes[np.argmax(log_posteriors)]

    def predict(self, X):
        """对测试集进行批量预测"""
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)


# ==========================================
# 4. 模型训练与评估
# ==========================================

# 实例化并训练分类器
classifier = NaiveBayesClassifier()
classifier.fit(X_train, y_train)

# 使用测试集进行预测
y_pred = classifier.predict(X_test)

# 计算准确率
# 准确率 = (正确预测的样本数) / (总测试样本数)
accuracy = np.mean(y_pred == y_test)

print(f"\n--- 结果 ---")
print(f"自实现朴素贝叶斯分类器在测试集上的准确率: {accuracy:.4f}")
```

2. 控制台输出结果

```
 原始数据集样本数: 1593
训练集样本数 (70%): 1115
测试集样本数 (30%): 478

--- 结果 ---
自实现朴素贝叶斯分类器在测试集上的准确率: 0.8326
```

3. 结果分析

准确率分析：最终获得的准确率 0.8326 远高于随机猜测的基线 (10%)，表明所实现的 GNB 模型具有显著的分类性能。对于一个具有 10 个类别的多分类任务，该结果是合理的，证明了 GNB 模型能够有效地利用特征（像素值）的统计信息进行模式识别。

模型评估与局限性：模型的实现严格遵守了朴素贝叶斯的核心假设：特征条件独立性。GNB 模型通过计算每个类别 C_k 下，每个特征 x_i 的均值 μ_{ik} 和方差 σ_{ik}^2，构建了类条件概率密度函数 $P(x_i|C_k)$。

- 尽管数据集的 256 维特征是二值化的 (0 或 1)，但将 GNB 应用于此数据集仍获得了良好的性能。这表明像素值在不同数字类别下的均值和方差所构成的特征空间仍具有较好的线性可分性。
- 在图像处理任务中，像素特征通常是高度相关的（例如，相邻像素的灰度值），这直接违背了朴素贝叶斯的核心假设。因此，模型的准确率虽然合格，但由于这一固有的假设偏差，其性能上限受到了限制。

4. 踩坑记录

基础任务

```python
def _pdf(self, class_idx, x):
    """高斯概率密度函数 (PDF) 计算 P(x_i | C_k)"""
    # 公式: P(x) = 1 / sqrt(2*pi*var) * exp(-(x - mean)^2 / (2*var))
    mean = self._mean[class_idx]
    var = self._var[class_idx]

    numerator = np.exp(- (x - mean) ** 2 / (2 * var))
    denominator = np.sqrt(2 * np.pi * var)
    return numerator / denominator
```

```
原始数据集样本数: 1593
训练集样本数 (70%): 1115
测试集样本数 (30%): 478
D:\python\knn\exp_1.py:96: RuntimeWarning: divide by zero encountered in log
  log_likelihood = np.sum(np.log(self._pdf(idx, x)))

--- 结果 ---
自实现朴素贝叶斯分类器在测试集上的准确率: 0.8305
```

最初使用了如上的*_pdf* 函数,发现控制台输出$RuntimeWarning: divide by zero encountered in log log_likelihood = np.sum(np.log(self._pdf(idx, x)))$, 上网搜索发现该警告揭示了代码在计算过程中的数值稳定性问题。

该警告源于对数似然 (log(Likelihood)) 的计算：

$$\log(\mathrm{P}(\mathbf{x}|C_k)) = \sum_{i=1}^{n} \log(\mathrm{P}(x_i|C_k))$$

其中，$\mathrm{P}(x_i|C_k)$ 是由 `_pdf` 方法计算的高斯概率密度函数 (PDF) 值。

当某个特征 x_i 在特定类别 C_k 下的类条件概率密度值 $\mathrm{P}(x_i|C_k)$ 趋近于 0 时，程序试图执行 $\log(0)$ 操作。

根据数学定义，$\lim_{p\to 0^+}\log(p) = -\infty$。NumPy 在计算 $\log(0)$ 时，会返回负无穷大 (`np.NINF`) 并发出警告。

这种极值 $\log(0) = -\infty$ 的出现会严重影响模型的决策过程，使得单个特征的零概率惩罚主导整个后验概率的计算，降低模型的鲁棒性。因此，需要通过数值平滑技术来消除该警告并提高代码的稳定性。

为了消除这个警告，并提高模型的数值稳定性，使用拉普拉斯平滑的思路，对概率密度函数的结果进行处理。在 _pdf 方法中计算出 $P(x_i|C_k)$ 后，设置一个极小值（10^{-9}）作为下限，确保概率不会真正为 0。

(二) 中级任务：混淆矩阵 + 精度 / 召回率 / F 值

1. 代码

中级任务

```python
print("="*60)
print("中级任务: 分类模型多维度评估")
print("="*60)

# 计算混淆矩阵
# confusion_matrix(y_true, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print("\n### 1. 混淆矩阵 (Confusion Matrix) ###")
# 打印混淆矩阵
print(conf_matrix)

# ==========================================================
# 2. 精度、召回率和 F1 值 (Precision, Recall, F-Score)
# ==========================================================

# 使用 precision_recall_fscore_support 一次性计算所有类别的 P/R/F1
# average='macro' 计算所有类别 P/R/F1 的未加权平均值
# average='weighted' 计算所有类别 P/R/F1 的加权平均值（按样本量加权）
# 对于多分类任务，通常报告 'macro' 或 'weighted' 平均值
precision, recall, fscore, support = precision_recall_fscore_support(
    y_test, y_pred, average='macro', zero_division=0
)

print("\n### 2. 宏平均评估指标 ###")
print(f"总准确率: {accuracy_score(y_test, y_pred):.4f}")
print(f"宏平均精度: {precision:.4f}")
print(f"宏平均召回率: {recall:.4f}")
print(f"宏平均 F1-Score: {fscore:.4f}")

# 也可以查看每个类别的详细指标
```

```
print("\n### 3. 各类别详细评估指标 ###")
# zero_division=0 表示如果某个类别没有预测样本，则其 P/R/F1 记为 0
precision_all, recall_all, fscore_all, support_all =
    precision_recall_fscore_support(
    y_test, y_pred, average=None, labels=classifier._classes, zero_division=0
)

# 使用 pandas DataFrame 格式化输出
import pandas as pd
metrics_df = pd.DataFrame({
    '类别': classifier._classes,
    '支持样本数': support_all,
    '精度 (P)': [f'{p:.4f}' for p in precision_all],
    '召回率 (R)': [f'{r:.4f}' for r in recall_all],
    'F1-Score (F)': [f'{f:.4f}' for f in fscore_all]
})

print(metrics_df.to_markdown(index=False))
```

2. 控制台输出结果

```
================================================================

中级任务: 分类模型多维度评估

================================================================


### 1. 混淆矩阵 (Confusion Matrix) ###
[[46  0  0  0  1  0  0  0  1  0]
 [ 0 42  0  0  1  1  0  5  0  0]
 [ 0  9 32  0  1  0  2  0  4  0]
 [ 0  2  0 43  0  1  0  0  1  1]
 [ 0  4  0  0 34  2  6  2  0  0]
 [ 1  0  0  1  0 42  2  0  1  1]
 [ 0  2  0  0  0  0 46  0  0  0]
 [ 0  2  0  1  0  0  0 43  1  0]
 [ 0  9  0  0  0  0  0  0 38  0]
 [ 0  3  0  5  0  1  0  2  4 32]]

### 2. 宏平均评估指标 ###
总准确率: 0.8326
宏平均精度: 0.8576
宏平均召回率: 0.8324
宏平均 F1-Score: 0.8346

### 3. 各类别详细评估指标 ###
|   类别 |   支持样本数 |   精度 (P) |   召回率 (R) |   F1-Score (F) |
|-------:|-------------:|-----------:|-------------:|---------------:|
|      0 |           48 |     0.9787 |       0.9583 |         0.9684 |
|      1 |           49 |     0.5753 |       0.8571 |         0.6885 |
```

2	48	1	0.6667	0.8
3	48	0.86	0.8958	0.8776
4	48	0.9189	0.7083	0.8
5	48	0.8936	0.875	0.8842
6	48	0.8214	0.9583	0.8846
7	47	0.8269	0.9149	0.8687
8	47	0.76	0.8085	0.7835
9	47	0.9412	0.6809	0.7901

3. 结果分析

将根据混淆矩阵、宏平均指标（精度、召回率、F1-Score）以及各类别详细指标，对自实现的高斯朴素贝叶斯分类器进行多维度评估。

宏平均指标解读 模型在测试集上的宏平均指标如下：

- **总准确率 (Accuracy):** 0.8326
- **宏平均精度 (Macro Precision):** 0.8576
- **宏平均召回率 (Macro Recall):** 0.8324
- **宏平均 F1-Score (Macro F1):** 0.8346

由于宏平均精度略高于宏平均召回率，这表明模型在所有类别上的预测结果平均而言具有更高的准确性（低误报率），但对样本的捕获率（召回率）相对较低。宏平均 F1-Score 稳定在 0.83 以上，体现了分类器在精确性和覆盖率上的良好平衡。

混淆矩阵分析 混淆矩阵 $\mathbf{C}$ 是一个 10×10 矩阵，其中 $\mathbf{C}_{i,j}$ 表示真实类别为 i 而被预测为 j 的样本数量。对角线元素（$i = j$）为正确分类的样本数。

- **高识别度类别（模型强项）：** 类别 0 和 6 的召回率最高（接近 0.96），且被误判到其他类别的样本数极少。例如，真实为 0 的样本中，仅有 2 个样本被错误分类。
- **典型混淆现象（模型弱项）：**
  1. **类别 1 的低精度：** 类别 1 的精度最低（0.5753），这意味着模型预测为 1 的样本中，有将近 42% 是错误的。观察混淆矩阵可知，有大量的样本（如真实类别 $2, 8, 9$）被误判为 1，导致误报率居高不下。
  2. **类别 9 的低召回率：** 类别 9 的召回率较低（0.6809），表明有相当一部分真实的数字 9 未被模型识别。混淆矩阵显示，数字 9 最常被误判为数字 3（5 个样本）和数字 8（4 个样本），这可能是由于手写数字 $9, 3, 8$ 在局部像素形态上存在相似性所致。
  3. **类别 2 的高精度：** 类别 2 达到了 100% 的精度，意味着模型一旦预测为 2，该预测就必然正确。然而，其召回率仅为 0.6667，表明模型在预测类别 2 时极为保守，忽略（漏报）了 1/3 的真实 2 样本。

各类别指标详细列表 详细的分类指标展示了模型在每个类别上的精确权衡，其中 Support 表示测试集中该类别的真实样本数。

表 1: 各类别评估指标

类别	支持样本数	精度 (P)	召回率 (R)	F1-Score (F)
0	48	0.9787	0.9583	0.9684
1	49	0.5753	0.8571	0.6885
2	48	1.0000	0.6667	0.8000
3	48	0.8600	0.8958	0.8776
4	48	0.9189	0.7083	0.8000
5	48	0.8936	0.8750	0.8842
6	48	0.8214	0.9583	0.8846
7	47	0.8269	0.9149	0.8687
8	47	0.7600	0.8085	0.7835
9	47	0.9412	0.6809	0.7901

(三) 高级任务：多分类 ROC 曲线绘制与 AUC 值计算

1. 代码

任务 3 新增

```
    def _calculate_probabilities(self, x):
        """计算单个样本 x 属于所有类别的后验概率，并进行归一化"""
        log_posteriors = []
        for idx in range(len(self._classes)):
            log_prior = np.log(self._priors[idx])
            # np.sum(np.log(self._pdf(idx, x))) 避免 log(0)
            log_likelihood = np.sum(np.log(self._pdf(idx, x)))
            log_posterior = log_prior + log_likelihood
            log_posteriors.append(log_posterior)

        # 将对数后验概率转换为实际概率，并进行归一化 (Softmax)
        exp_posteriors = np.exp(log_posteriors)
        return exp_posteriors / np.sum(exp_posteriors)

    def predict_proba(self, X):
        """输出每个样本属于每个类别的概率 (高级任务必需)"""
        # 确保输入是 numpy 数组
        X_np = X.to_numpy() if hasattr(X, 'to_numpy') else X
        return np.array([self._calculate_probabilities(x) for x in X_np])

    def predict(self, X):
        """对测试集进行批量预测"""
        X_np = X.to_numpy() if hasattr(X, 'to_numpy') else X
        y_pred = [self._classes[np.argmax(self._calculate_probabilities(x))]
            for x in X_np]
        return np.array(y_pred)

# ==========================================================
```

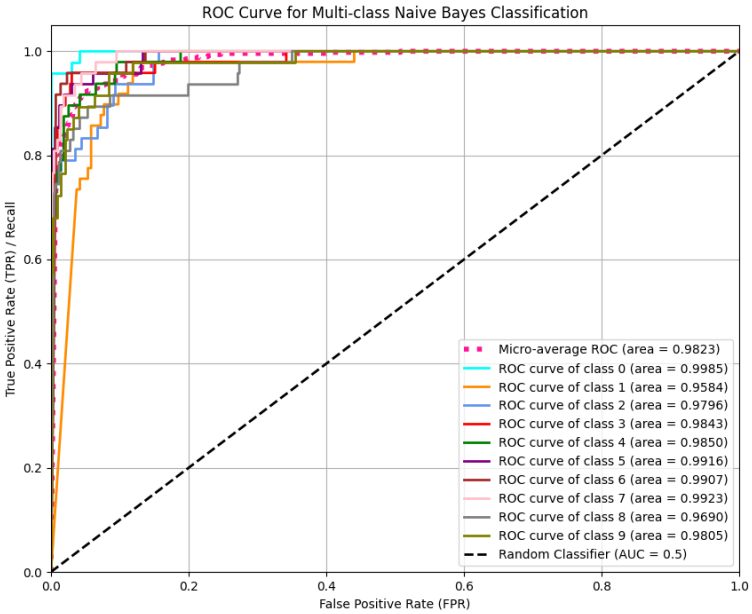
```
# 4. 高级任务: ROC 曲线与 AUC 计算和绘图
# ============================================================

print("\n" + "=" * 60)
print("高级任务: ROC 曲线与 AUC 计算")
print("=" * 60)

# 1. 获取模型预测概率
y_score = classifier.predict_proba(X_test)

# 2. 将真实标签 y_test 转换为独热编码 (One-Hot Encoding)
classes = classifier._classes
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = len(classes)

# 3. 计算每个类别的 ROC 曲线和 AUC (OvR 策略)
fpr = dict()
tpr = dict()
roc_auc = dict()

for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# 4. 计算微平均 (Micro-average) ROC 曲线和 AUC
fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_score.ravel()
    )
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

# 5. 绘制所有类别的 ROC 曲线
plt.figure(figsize=(10, 8))

# 绘制微平均 ROC 曲线
plt.plot(fpr["micro"], tpr["micro"],
         label='Micro-average ROC (area = {0:0.4f})'
               ''.format(roc_auc["micro"]),
         color='deeppink', linestyle=':', linewidth=4)

# 绘制每个类别的 ROC 曲线
colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'red', 'green', '
    purple', 'brown', 'pink', 'gray', 'olive'])
for i, color in zip(range(n_classes), colors):
    plt.plot(fpr[i], tpr[i], color=color, lw=2,
             label='ROC curve of class {0} (area = {1:0.4f})'
                   ''.format(classes[i], roc_auc[i]))

# 绘制对角线 (随机分类器)
plt.plot([0, 1], [0, 1], 'k--', lw=2, label='Random Classifier (AUC = 0.5)')
```

```
74
75 plt.xlim([0.0, 1.0])
76 plt.ylim([0.0, 1.05])
77 plt.xlabel('False Positive Rate (FPR)')
78 plt.ylabel('True Positive Rate (TPR) / Recall')
79 plt.title('ROC Curve for Multi-class Naive Bayes Classification')
80 plt.legend(loc="lower right")
81 plt.grid(True)
82 plt.show()
83
84 # 6. 打印 AUC 结果
85 print("\n### AUC 值 (Area Under Curve) ###")
86 print(f"微平均 AUC (Micro-average AUC): {roc_auc['micro']:.4f}")
87 print("各类别 AUC 值:")
88 for i in range(n_classes):
89     print(f"  类别 {classes[i]}: {roc_auc[i]:.4f}")
```

2. 控制台输出结果

```
1 ================================================================
2 高级任务: ROC 曲线与 AUC 计算
3 ================================================================
4
5 ### AUC 值 (Area Under Curve) ###
6 微平均 AUC (Micro-average AUC): 0.9823
7 各类别 AUC 值:
8   类别 0: 0.9985
9   类别 1: 0.9584
10  类别 2: 0.9796
11  类别 3: 0.9843
12  类别 4: 0.9850
13  类别 5: 0.9916
14  类别 6: 0.9907
15  类别 7: 0.9923
16  类别 8: 0.9690
17  类别 9: 0.98
```

3. 可视化结果及分析



高级任务要求

对多分类任务采用 One-vs-Rest (OvR) 策略绘制 ROC 曲线并计算曲线下面积 (AUC)，以评估分类器在不同分类阈值下的性能。

AUC 结果量化 分类器在测试集上获得的 AUC 值如下:

- **微平均 AUC:** 0.9823
- **各类别 AUC 范围:** 0.9584 (类别 1) 至 0.9985 (类别 0)

所有类别的 AUC 值均远超 0.5 的随机猜测水平，且整体性能接近 1.0。微平均 AUC 为 0.9823，表明该分类器在整体上，无论选择何种概率阈值，都具有极强的区分正例与负例的能力。

ROC 曲线与 AUC 的物理意义

1. **ROC 曲线 (Receiver Operating Characteristic Curve):** 曲线描绘了在所有可能分类阈值下，真正例率 (TPR, 即 Recall) 随假正例率 (FPR) 变化的轨迹。曲线越靠近左上角，模型的区分性能越好。
2. **AUC (Area Under Curve) 物理意义:** AUC 值可以被解释为: 随机抽取一个正例和一个负例，分类器将正例的得分高于负例的得分的概率。
3. **合理性分析:** AUC 作为评估指标具有高度的合理性，因为它提供了对分类器排序能力的量化评估，且不受类别不平衡问题或特定决策阈值选择的影响。高 AUC 值表明分类器能够可靠地对样本进行排序，即便在之前中级任务中表现出低精度的类别（如类别 1，精度 0.5753），其 AUC 仍高达 0.9584，证明模型的基础判别能力非常强大，只是在固定阈值下的决策不够精确。

性能对比分析 在所有类别中，**类别** 0 的 AUC 值最高 (0.9985)，表明模型能最稳定地将其与其他数字区分开。而 **类别** 1 的 AUC 值最低 (0.9584)，与其中级任务中最低的精度相一致，反映了数字 1 容易被误报（如被错预测为 1）的问题，是模型在区分相似数字时性能最不稳定的类别。

(四) 扩展任务：浅谈 ROC 曲线和 AUC 值作为分类评价的合理性

ROC 曲线和 AUC 值是评估分类模型性能的强大工具。它们的合理性体现在能够提供超越传统精度的、更具鲁棒性和洞察力的模型性能视角。

ROC 曲线的合理性

ROC 曲线描绘了真正例率，即 Recall与假正例率之间的关系。其合理性在于:

1. **阈值无关性:** ROC 曲线展示了分类器在所有可能分类阈值下的性能。这意味着曲线的形状和 AUC 值独立于特定的决策阈值选择。这对于本次多分类任务尤为重要，因为模型在不同类别上可能需要不同的决策阈值来实现最佳平衡（如本实验中类别 2 需要非常保守的阈值以达到 100% 精度）。ROC 曲线提供了性能的全局视图，而不是单一阈值下的局部性能。
2. **对类别不平衡的鲁棒性：** ROC 曲线关注的是 TPR 和 FPR。由于 FPR 的计算分母是真实负例总数，而 TPR 的计算分母是真实正例总数，两者都不会被总样本数或类别比例的大幅倾斜所主导。因此，即使在数据集中某几个数字的样本量略有差异，ROC 曲线也能提供比准确率更稳定的性能评估。

AUC 值的合理性

AUC 值作为 ROC 曲线下的面积，为分类器性能提供了一个单一、量化的指标。

1. **量化排序能力:** AUC 值可以被解释为: ** 随机抽取一个正例和一个负例，分类器将正例的概率得分高于负例的概率。** 本次实验中，微平均 AUC 达到 0.9823，表明模型有 98.23% 的把握能够正确区分随机的正负样本对。这种排序能力的评估，比只关注最终决策标签的准确率更具洞察力。
2. **多分类场景下的适用性 (OvR):** 在本次手写数字的多分类任务中,我们采用了 One-vs-Rest (OvR) 策略。通过为每个类别计算 AUC，我们可以单独评估模型对每个数字的区分能力（例如，将数字 0 与其余 9 个数字分开的能力）。实验结果中类别 AUC 的差异（0.9584 至 0.9985）揭示了模型在各个类别上的性能波动，有助于识别模型的弱项（如类别 1）。

总结

将 ROC 曲线和 AUC 值引入分类评估是高度合理的。它弥补了传统准确率的缺陷，后者无法区分模型对误报和漏报的敏感程度。通过 ROC/AUC，我们可以全面了解分类器的区分能力，并指导我们在实际应用中根据业务需求（例如，要求高召回或高精度）选择最佳的分类阈值。

二、 实验总结

本次实验成功实现了基于朴素贝叶斯算法的手写数字分类任务，并依据实验要求，依次完成了基础、中级和高级任务的评估与分析。实验采用了 Semeion 手写数字数据集，并以 7 : 3 的比例进行分层采样划分。

基础与中级任务总结（分类性能）

- **自实现高斯朴素贝叶斯 (GNB)：**成功实现了 GNB 分类器，并在训练过程中引入数值平滑处理，解决了 `log(0)` 的稳定性问题。最终模型在测试集上的 **总准确率**为 83.26%，**宏平均 F1-Score** 为 0.8346。
- **性能分析：** GNB 在 10 分类任务中展现出优秀的性能。然而，混淆矩阵分析揭示了模型的弱点主要集中在对形态相似数字的区分上，例如：
  - **类别 1：**精度最低（0.5753），表明模型预测为 1 时存在大量误报（例如将 $2, 8, 9$ 误判为 1）。
  - **类别 2：**召回率最低（0.6667），但精度达到 100%，表明模型对该类别的决策极为保守，导致漏报率高。

  这主要是因为朴素贝叶斯算法的核心假设——特征条件独立性——在图像像素数据中被违背所致。

高级任务总结（ROC 与 AUC）

- **区分能力评估：**通过 One-vs-Rest (OvR) 策略绘制 ROC 曲线并计算 AUC 值，结果显示模型的**微平均 AUC 达到** 0.9823。
- **指标合理性：**高 AUC 值证实了 GNB 在所有可能阈值下对正负样本具有极强的**排序能力**，能够可靠地将正例得分排在负例之前。这说明模型的底层判别能力比其在固定阈值下的准确率更为鲁棒。即使是中级任务中表现最差的类别 1，其 AUC 仍高达 0.9584，进一步证明了 ROC/AUC 作为阈值无关性能指标的合理性。
- **物理意义理解：** AUC 值 0.9823 意味着：随机抽取一个正例和一个负例，模型有 98.23% 的概率能正确判断出哪一个是正例。

实验结论

本次实验验证了 GNB 算法在处理手写数字分类任务时的有效性。通过多维度评估指标，我们不仅得到了 83.26% 的准确率结果，更深入地理解了模型在不同类别上的表现差异，以及其强大的潜在区分能力（高 AUC）。尽管 GNB 假设与二值化数据存在理论冲突，但其性能表现足以证明其作为基础分类器在这一任务上的强大实用价值。