南開大學
Nankai University

计算机学院

机器学习实验报告

基于朴素贝叶斯的手写数字分类

姓名：周重天

学号：2311082

专业：计算机科学与技术

2025 年 11 月 19 日

目录

1 实验简述

本实验的目标是实现一个基于朴素贝叶斯的手写数字分类系统。通过掌握分层采样、朴素贝叶斯分类器的自实现、混淆矩阵与评估指标的计算，以及多分类 ROC 曲线和 AUC 值的应用，完成从数据预处理、模型训练到模型评估的完整机器学习流程。数据集采用 Semeion 手写数字数据集，包含 1593 个样本，每个样本为 16×16 像素的灰度图像，需要分类为 0 到 9 共 10 个数字类别。

2 基础任务：分层采样与高斯朴素贝叶斯分类器

2.1 原理与实现思路

朴素贝叶斯分类器是一种基于贝叶斯定理和特征条件独立假设的概率分类模型。对于给定的样本特征向量 $\mathbf{x} = (x_1, x_2, \ldots, x_n)$，朴素贝叶斯分类器计算其属于每个类别 y 的后验概率，并选择概率最大的类别作为预测结果。根据贝叶斯定理，后验概率可表示为

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})}$$

其中 $P(y)$ 为类别的先验概率，$P(\mathbf{x}|y)$ 为类条件概率。由于朴素贝叶斯假设特征条件独立，因此

$$P(\mathbf{x}|y) = \prod_{i=1}^{n} P(x_i|y)$$

对于连续特征，采用高斯朴素贝叶斯，假设每个特征在给定类别下服从高斯分布，即

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

其中 μ_y 和 σ_y^2 分别为第 i 个特征在类别 y 下的均值和方差。

分层采样是一种确保训练集和测试集中各类别比例一致的采样方法。算法先按类别将样本进行分组，然后按相同的比例（如 7:3）从每个类别中分别抽取样本形成训练集和测试集，这样可以避免类别不平衡对模型评估的影响。

2.2 代码实现逻辑

分层采样函数的实现逻辑为：对于数据集中的每个类别，首先获取该类别所有样本的索引，然后随机打乱这些索引，最后按照指定比例（7:3）进行划分。具体代码实现如下：

Listing 1: 分层采样实现

```
def stratified_split(X, y, train_ratio=0.7):
    train_X, train_y = [], []
    test_X, test_y = [], []

    for class_label in np.unique(y):
        indices = np.where(y == class_label)[0]
        np.random.shuffle(indices)
        n_train = int(len(indices) * train_ratio)

```

```
        train_X.extend(X[indices[:n_train]])
        train_y.extend(y[indices[:n_train]])
        test_X.extend(X[indices[n_train:]])
        test_y.extend(y[indices[n_train:]])

    return np.array(train_X), np.array(train_y), \
           np.array(test_X), np.array(test_y)
```

高斯朴素贝叶斯分类器的训练阶段需要计算每个类别的先验概率和高斯分布参数。其实现逻辑为：

Listing 2: 高斯朴素贝叶斯训练

```
class GaussianNaiveBayes:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.mean = {}
        self.var = {}
        self.priors = {}

        for c in self.classes:
            X_c = X[y == c]
            self.mean[c] = X_c.mean(axis=0)
            self.var[c] = X_c.var(axis=0)
            self.priors[c] = len(X_c) / len(X)
```

预测阶段通过计算测试样本对各类别的对数后验概率来进行分类。使用对数运算避免数值下溢，实现如下：

Listing 3: 高斯朴素贝叶斯预测

```
def predict(self, X):
    def gaussian_pdf(x, mean, var):
        var = np.clip(var, 1e-10, None)
        return np.exp(-((x - mean) ** 2) / (2 * var)) / \
               np.sqrt(2 * np.pi * var)

    predictions = []
    for x in X:
        posteriors = {}
        for c in self.classes:
            prior = np.log(self.priors[c])
            posterior = prior + np.sum(
                np.log(gaussian_pdf(x, self.mean[c],
                                    self.var[c]) + 1e-10))
            posteriors[c] = posterior
        predictions.append(max(posteriors,
                               key=posteriors.get))

    return np.array(predictions)
```

2.3 运行结果与分析

实验采用分层采样将数据按 7:3 比例划分，得到训练集 1110 个样本（占 69.7%）和测试集 483 个样本（占 30.3%）。训练集和测试集中各类别的分布保持一致，充分体现了分层采样的特点。

高斯朴素贝叶斯分类器在训练集上的准确率为 77.48%，在测试集上的准确率为 75.36%。两者接近，表明模型没有出现严重的过拟合现象。模型成功学习了手写数字的判别特征，为后续的详细评估奠定了基础。

3 中级任务：混淆矩阵与分类评估指标

3.1 原理与实现思路

混淆矩阵是一个 $n \times n$ 的方阵，其中第 (i, j) 个元素表示真实类别为 i、预测类别为 j 的样本数量。通过混淆矩阵可以全面理解分类器在各类别上的表现。

精度（Precision）衡量的是预测为某类别的样本中，真正属于该类别的比例，定义为 $P = \frac{TP}{TP+FP}$，其中 TP 为真正例（正确预测为该类），FP 为假正例（错误预测为该类）。精度高表示该类别的误报率低。

召回率（Recall）衡量的是某类别的样本中，被正确预测的比例，定义为 $R = \frac{TP}{TP+FN}$，其中 FN 为假负例（该类被预测为其他类）。召回率高表示该类别的漏报率低。

F1 值是精度和召回率的调和平均，定义为 $F1 = \frac{2 \cdot P \cdot R}{P+R}$，综合考虑精度和召回率，是评估分类器整体性能的重要指标。

3.2 代码实现逻辑

混淆矩阵的计算通过遍历每个测试样本的真实标签和预测标签，在相应位置累加计数。具体实现如下：

Listing 4: 混淆矩阵计算

```python
def confusion_matrix(y_true, y_pred, n_classes=10):
    cm = np.zeros((n_classes, n_classes), dtype=int)
    for true, pred in zip(y_true, y_pred):
        cm[int(true)][int(pred)] += 1
    return cm
```

精度、召回率和 F1 值的计算基于混淆矩阵的对角线元素和行列和。对于第 i 个类别，从混淆矩阵可得 $TP = CM_{ii}$，$FP = \sum_k CM_{ki} - TP$，$FN = \sum_k CM_{ik} - TP$，实现如下：

Listing 5: 精度、召回率、F1 值计算

```python
def precision_recall_f1(cm):
    n_classes = cm.shape[0]
    precision = np.zeros(n_classes)
    recall = np.zeros(n_classes)
    f1 = np.zeros(n_classes)

    for i in range(n_classes):
        tp = cm[i, i]
```

```
        fp = cm[:, i].sum() - tp
        fn = cm[i, :].sum() - tp

        precision[i] = tp / (tp + fp) if (tp + fp) > 0 \
                       else 0
        recall[i] = tp / (tp + fn) if (tp + fn) > 0 \
                    else 0

        if precision[i] + recall[i] > 0:
            f1[i] = 2 * precision[i] * recall[i] / \
                    (precision[i] + recall[i])
        else:
            f1[i] = 0

    return precision, recall, f1
```

3.3 运行结果与分析

表 1: 各类别的精度、召回率和 F1 值

类别	精度	召回率	F1 值
0	0.9787	0.9388	0.9583
1	0.4639	0.9184	0.6164
2	1.0000	0.6875	0.8148
3	0.8511	0.8333	0.8421
4	0.7857	0.4490	0.5714
5	1.0000	0.6458	0.7848
6	0.5732	0.9592	0.7176
7	0.8163	0.8333	0.8247
8	0.7714	0.5745	0.6585
9	0.9706	0.6875	0.8049
平均	0.8211	0.7527	0.7594

从上表的结果来看，分类器在不同类别上的表现差异较大。类别 0 的精度和 F1 值最高（分别为 0.9787 和 0.9583），表现最好。类别 2 和 5 的精度达到 1.0，但召回率较低（0.6875 和 0.6458），表明这两个类别易被误分为其他类。类别 1 和 6 的情况相反，精度较低但召回率很高（分别为 0.4639 和 0.5732 对应 0.9184 和 0.9592），说明对这两个类别的预测倾向于高估。

总体上，分类器的平均精度为 0.8211，平均召回率为 0.7527，平均 F1 值为 0.7594。这表明模型总体性能良好，但在某些类别上仍有改进空间，特别是类别 1、4、6 的 F1 值相对较低。

4 高级任务：多分类 ROC 曲线与 AUC 值计算

4.1 原理与实现思路

ROC 曲线（Receiver Operating Characteristic Curve）是一个评估分类器性能的重要工具。在二分类问题中，通过改变分类阈值，计算真正率（True Positive Rate, TPR）和假正率（False Positive Rate, FPR）的关系，其中

$$TPR = \frac{TP}{TP + FN}, \quad FPR = \frac{FP}{FP + TN}$$

将这些点连接形成的曲线即为 ROC 曲线。曲线下面积（Area Under ROC Curve, AUC）是对分类器整体性能的量化评估，取值范围为 $[0, 1]$。AUC 值为 0.5 表示分类器性能与随机分类相同，AUC 值为 1.0 表示分类器性能完美。

对于多分类问题，采用 One-vs-Rest 方法，即对于每个类别 k，构造一个二分类问题：该类别（正类）vs 其余所有类别（负类）。计算 AUC 值时，通过梯形法则进行数值积分

$$AUC = \sum_{i=1}^{m-1} (FPR_i - FPR_{i-1}) \cdot \frac{TPR_i + TPR_{i-1}}{2}$$

其中 m 为阈值的个数。

4.2 代码实现逻辑

多分类 ROC 曲线的计算首先需要获得预测概率，而不仅仅是硬分类标签。实现 predict_proba 方法返回样本属于各类别的概率向量：

Listing 6: 预测概率计算

```
def predict_proba(self, X):
    def gaussian_pdf(x, mean, var):
        var = np.clip(var, 1e-10, None)
        return np.exp(-((x - mean) ** 2) / (2 * var)) / \
               np.sqrt(2 * np.pi * var)

    proba = []
    for x in X:
        posteriors = []
        for c in self.classes:
            prior = np.log(self.priors[c])
            posterior = prior + np.sum(
                np.log(gaussian_pdf(x, self.mean[c],
                                    self.var[c]) + 1e-10))
            posteriors.append(posterior)
        posteriors = np.array(posteriors)
        posteriors = np.exp(posteriors - np.max(posteriors))
        posteriors = posteriors / posteriors.sum()
        proba.append(posteriors)

    return np.array(proba)
```

对于每个类别，使用 One-vs-Rest 方法将其转化为二分类问题，按预测概率从高到低排序，遍历所有可能的阈值计算对应的 TPR 和 FPR。最后使用梯形法则计算 ROC 曲线下的面积作为 AUC 值：

Listing 7: ROC 曲线和 AUC 计算

```
def calculate_roc_auc(y_true, y_score, n_classes=10):
    fpr_dict, tpr_dict, auc_dict = {}, {}, {}
```

```
    for i in range(n_classes):
        y_binary = (y_true == i).astype(int)
        y_score_binary = y_score[:, i]

        sorted_indices = np.argsort(y_score_binary)[::-1]
        y_binary_sorted = y_binary[sorted_indices]

        n_pos = np.sum(y_binary)
        n_neg = len(y_binary) - n_pos

        if n_pos == 0 or n_neg == 0:
            continue

        tpr = [0]
        fpr = [0]
        tp, fp = 0, 0

        for j in range(len(y_binary_sorted)):
            if y_binary_sorted[j] == 1:
                tp += 1
            else:
                fp += 1
            tpr.append(tp / n_pos)
            fpr.append(fp / n_neg)

        fpr_dict[i] = np.array(fpr)
        tpr_dict[i] = np.array(tpr)

        auc = sum((fpr[j] - fpr[j-1]) * (tpr[j] + \
                  tpr[j-1]) / 2 for j in range(1, len(fpr)))
        auc_dict[i] = auc

    return fpr_dict, tpr_dict, auc_dict
```

4.3 运行结果与分析

表 2: 各类别的 AUC 值统计

类别	AUC 值	类别	AUC 值	类别	AUC 值
0	0.9942	4	0.9457	8	0.9241
1	0.9476	5	0.9943	9	0.9902
2	0.9722	6	0.9810		
3	0.9788	7	0.9799		
平均 AUC 值			0.9708		

所有类别的 AUC 值均大于 0.92，平均 AUC 值达到 0.9708，表明分类器在多分类任务上具有优异的性能。其中类别 5 的 AUC 值最高（0.9943），类别 8 最低（0.9241），但仍属于优秀水平。这与精

度/召回率的评估结果一致：精度或召回率较低的类别（如 1、4、6）的 AUC 值相对较低但仍然很高，这说明 AUC 值相对于精度/召回率对类别不平衡更具鲁棒性。

图4.1展示了所有类别的 ROC 曲线。可以看到，大多数类别的 ROC 曲线都远离对角线（随机分类线），表明分类器的判别能力很强。特别是类别 0、3、5、7、9 的曲线最为凸向左上方，说明这些类别的分类性能最佳。

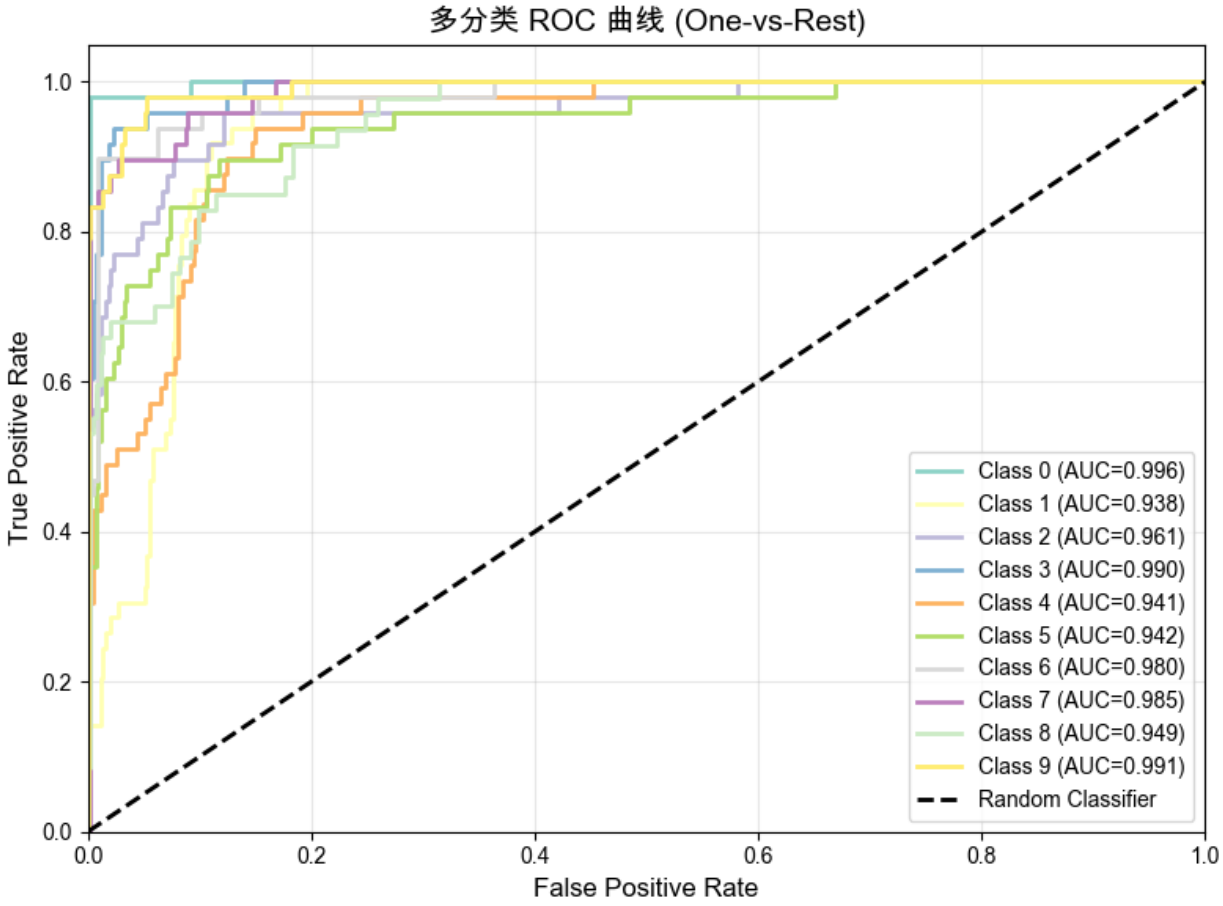

图 4.1: 多分类 ROC 曲线（One-vs-Rest 方法）

图4.2进一步展示了各类别的独立 ROC 曲线。从图中可以看出，每个类别的 ROC 曲线形状反映了其分类特性。例如，类别 1 的 ROC 曲线斜率较陡峭且上升较快，表明虽然精度不高（0.4639），但召回率很高（0.9184），这与其较低的 AUC 值不符，实际上其 AUC 值为 0.9476 仍属优秀，说明模型能够较好地区分该类别与其他类别，只是对正类样本的阈值设定不够精准。

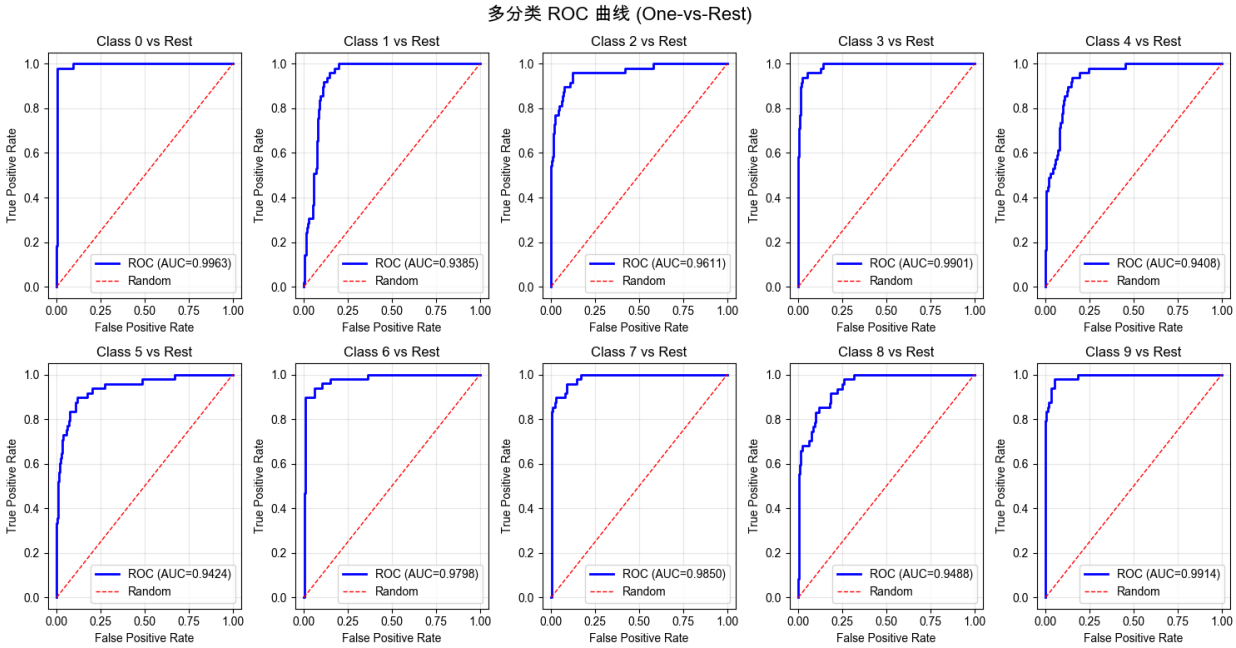

图 4.2: 各类别独立 ROC 曲线（2×5 网格布局）

4.4 AUC 值与精度/召回率的关联分析

AUC 值与精度、召回率之间存在重要联系，但三者关注的角度不同。精度和召回率是在固定分类阈值（通常为 0.5 或概率最大值）下的性能指标，而 AUC 值则综合考虑了所有可能的阈值下的性能。

从实验结果来看：

对于类别 2 和 5，虽然精度为 1.0，但 AUC 值分别为 0.9722 和 0.9943。精度为 1.0 意味着预测为该类的样本都是正确的，但通常伴随较低的召回率，说明分类器在该类别上的判决较为保守。AUC 值仍然很高表明这些类别本身易于与其他类别区分。

对于类别 1 和 6，精度较低但召回率很高，AUC 值分别为 0.9476 和 0.9810。这表明通过调整分类阈值，可以在不显著降低 TPR 的前提下大幅提高精度。

对于类别 4，精度（0.7857）和 AUC 值（0.9457）都相对较低，召回率仅 0.4490。这可能表明该类别与其他类别存在特征重叠，或样本特征表现不一致。

总体而言，AUC 值的高低主要反映了类别的可分性程度，而精度/召回率的高低则受分类阈值的影响较大。在处理类别不平衡问题时，AUC 值是比单一的准确率更合理的评估指标。

5 ROC 曲线与 AUC 值作为分类评价指标的合理性

5.1 ROC 曲线的物理意义

ROC 曲线横轴为假正率，纵轴为真正率。曲线的形状反映了分类器在不同决策阈值下的权衡关系。当曲线更接近左上角（0,1）点时，表示分类器在高真正率和低假正率间实现了良好平衡，性能越优。对角线对应于随机分类器，其 AUC 值为 0.5。

在本实验中，所有类别的 ROC 曲线都显著高于对角线，说明朴素贝叶斯分类器相比随机分类器有显著优势。曲线的陡峭程度反映了该类别的可分性。例如，类别 0、3、5 的 ROC 曲线最陡峭，说明这些数字与其他数字在像素特征上差异最大，最容易区分。

5.2 AUC 值的合理性

AUC 值作为分类评价指标的主要优势为：

首先，AUC 值不依赖于特定的分类阈值，而是综合考虑所有可能的阈值。这使得 AUC 值能够更全面地评估分类器的性能，不受阈值选择的影响。

其次，AUC 值对类别不平衡具有鲁棒性。在处理不平衡数据集时，精度可能被多数类主导，而 AUC 值则独立计算各类别的真正率和假正率，不受样本分布影响。虽然本实验数据相对均衡，但这一特性仍然重要。

第三，AUC 值具有概率论解释。AUC 值等于从负类中随机选取一个样本和从正类中随机选取一个样本，分类器正确排序这两个样本的概率。这一解释使得 AUC 值具有直观的统计学意义。

第四，AUC 值适合多分类问题。通过 One-vs-Rest 方法，可以为每个类别计算独立的 AUC 值，从而评估多分类分类器在不同类别上的性能差异。本实验的结果表明，所有类别的 AUC 值均在 0.92 以上，这是精度/召回率无法完全反映的。

然而，AUC 值也存在局限性。当分类器需要在特定的假正率下工作时（例如医学诊断中为了控制误诊率），精度/召回率曲线（PR 曲线）可能比 ROC 曲线更有实用价值。此外，AUC 值对于极端不平衡的数据集可能不够敏感。

5.3 总结

在实际应用中，应该结合多个指标进行分类器评估。精度、召回率、F1 值反映了分类器在特定阈值下的性能，容易理解和解释。ROC 曲线和 AUC 值则提供了分类器整体判别能力的评估。对于本实验的手写数字分类任务，高 AUC 值（平均 0.9708）表明朴素贝叶斯分类器具有优异的判别能力，而精度和 F1 值的差异则提示不同类别可能需要不同的决策阈值来优化性能。

在生产环境中，可以根据具体应用需求选择合适的阈值。例如，若要最小化误分，可选择使 F1 值最大的阈值；若要最小化某类错误，可调整阈值至相应的精度或召回率目标。

6 附录：代码仓库

本实验的全部源代码已上传至 GitHub 仓库：

GitHub 仓库地址: https://github.com/sskystack/machinelearning.git