南　　开　　大　　学

计　算　机　学　院

实验报告

基于 kNN 的手写数字识别实验

姓名：申健强

学号：2313119

专业：计算机科学与技术

2025 年 9 月 25 日

一、 实验目的

1. **初级要求：**

   (i) 理解 k 近邻（kNN）算法的核心原理，掌握手动实现方法（禁用第三方机器学习库）。

   (ii) 掌握留一法（Leave-One-Out）交叉验证的流程，理解其评估模型泛化能力的意义。

2. **中级要求：**对比手动实现 kNN 与 Weka 工具的性能差异，分析精度（ACC）等指标的差距来源。[1]

3. **高级要求：**探索数据增强（如旋转）对模型性能的影响，尝试使用 CNN 等深度学习方法提升识别精度。

二、 实验原理

(一) kNN 算法核心原理

- **基本思想**：对于测试样本，计算其与所有训练样本的距离，选取距离最近的 k 个样本（“邻居”），通过多数投票法确定测试样本的类别。
- **距离度量**：采用欧氏距离（Euclidean Distance）：

$$d(\mathbf{x},\mathbf{y})=\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

  其中 $\mathbf{x},\mathbf{y}$ 为两个样本的特征向量，n 为特征维度（本实验中 $n=256$）。

(二) 留一法交叉验证（LOO-CV）

留一法是交叉验证的特殊形式：将数据集划分为 N 份（N 为样本总数），每次取 1 份作为测试集，其余 $N-1$ 份作为训练集，重复 N 次实验，最终精度为 N 次实验的平均精度。其优点是评估结果稳定（无随机划分误差），适合小样本数据集（如 semeion 数据集 $N=1593$）。

(三) 性能评价指标

- **精度（Accuracy, ACC）**：分类正确的样本数占总样本数的比例：

$$\text{ACC}=\frac{\text{正确分类样本数}}{\text{总样本数}}$$

- **归一化互信息（Normalized Mutual Information, NMI）**：衡量预测标签与真实标签的相似度，取值范围 [0,1]，值越大表示一致性越高：

$$\text{NMI}(Y,\hat{Y})=\frac{I(Y,\hat{Y})}{\sqrt{H(Y)H(\hat{Y})}}$$

  其中：

  - $I(Y,\hat{Y})$ 为互信息，$I(Y,\hat{Y})=H(Y)-H(Y|\hat{Y})$；

[1]因为一些原因，本人电脑上的 java 环境处于崩溃状态，所以中级要求我们选用 sklearn 实现的 KNN 算法与我们自己实现的 KNN 算法进行对比

- $H(Y)$ 为真实标签的熵，$H(Y) = -\sum_{i=1}^{C} P(y_i) \log P(y_i)$；
- $H(\hat{Y})$ 为预测标签的熵，$H(\hat{Y}) = -\sum_{j=1}^{C} P(\hat{y}_j) \log P(\hat{y}_j)$；
- C 为类别数（本实验中 $C = 10$）。

- **混淆熵（Confusion Entropy, CEN）**：衡量分类结果的不确定性，值越小表示分类越明确：

$$\mathrm{CEN} = -\sum_{i=1}^{C}\sum_{j=1}^{C} \frac{n_{ij}}{N} \log\left(\frac{n_{ij}}{N}\right)$$

其中 n_{ij} 为第 i 类被预测为第 j 类的样本数，N 为总样本数。

1. 代码实现关键步骤

(1) 数据加载：读取 semeion 数据集，解析特征（前 256 列）和标签（后 10 列 one-hot 编码转数字）。

(2) 距离计算：利用 numpy 快速进行批量 distance 计算。

(3) 留一法验证：循环将每个样本作为测试集，其余作为训练集，计算 k 近邻并投票。

核心代码片段（KNN 留一法验证）[2]：

```python
#KNN算法，使用numpy直接进行距离计算nmi与cen为自封装函数，在代码仓库中可以找到
def knn(vectors,labels,k):
    n=vectors.shape[0]
    correct=0
    preds=[]
    for i in range(n):
        test_vec=vectors[i]
        test_label=labels[i]

        train_vecs=np.delete(vectors,i,axis=0)
        train_labels=np.delete(labels,i,axis=0)

        distances = np.linalg.norm(train_vecs - test_vec, axis=1)
        sorted_idx = np.argsort(distances)
        k_nearest = train_labels[sorted_idx[:k]]

        pred=Counter(k_nearest).most_common(1)[0][0]
        preds.append(pred)
        if pred==test_label:
            correct+=1

    acc=correct/n
    nmi_val=nmi(labels,preds)
    cen_val=cen(labels,preds)

    return acc,nmi_val,cen_val
```

[2]出于报告版面设计考虑，报告中我们不会显示所有的代码，从本次实验开始的所有实验的所有源代码与实验结果我都将开源在 GitHub 仓库：https://github.com/sjq0098/Machine_Learning.git

2. 不同 k 值下的识别精度

k 值	留一法精度（LOO Accuracy）
5	91.71 %
9	92.47 %
13	91.59 %

表 1: 不同 k 值下手动实现 kNN 的识别精度

3. 结果截图与可视化

我们将实验结果存放于 result.txt 文件中，将实验结果列表如下：

表 2: KNN 在不同 k 下的性能比较（LOOCV）

k	ACC	NMI	CEN
5	0.9171	0.8405	0.3534
7	0.9247	0.8521	0.3220
9	0.9247	0.8503	0.3170
13	0.9159	0.8358	0.3653

同时我们，对数据进行可视化便于进行直观分析：

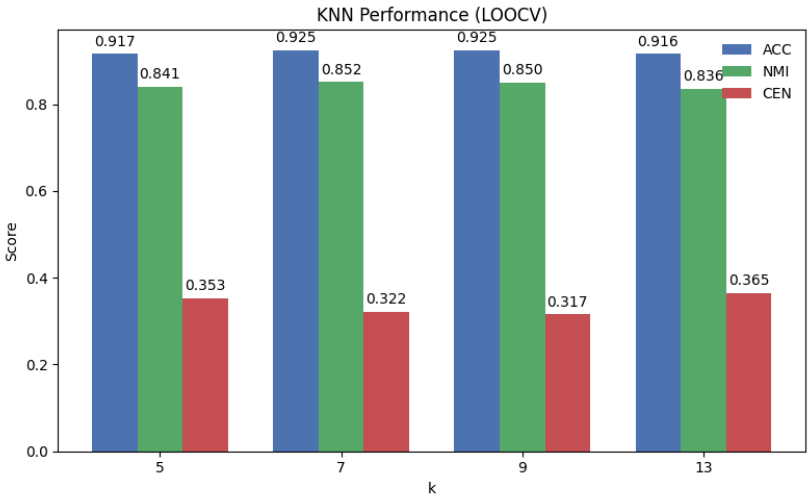

图 1: KNN performance

结果解释 从表 1 以及表格结果可以看出：

- 当 $k = 7$ 与 $k = 9$ 时，识别精度（ACC）均达到最高值 92.47%，优于 $k = 5$ 与 $k = 13$ 的情况。这说明在本实验数据集上，中等规模的邻居数能够在避免过拟合与欠拟合之间取得较好平衡。
- NMI 指标同样在 $k = 7$ 时达到最大值 0.8521，表明预测结果与真实类别之间的信息一致性最强。随着 k 继续增大到 13，NMI 出现下降趋势，说明分类边界逐渐模糊。

- CEN 指标反映了分类混淆度，其值越小越好。在 $k = 9$ 时达到最优（0.3170），表明此时错误分类的熵最低。
- 综合 ACC、NMI 和 CEN 三个指标，可以认为 $k = 7 \sim 9$ 是较优选择，其中 $k = 9$ 在整体表现上最为稳定。

综上，手动实现的 KNN 在本数据集上的最佳参数约为 $k = 9$，此时能够获得较高的准确率与较低的分类混淆度，体现了 KNN 在模式识别中的有效性。

另外，对于原实验指导书中提到的加权投票方式我们也尝试了自己实现，但是发现实现效果可能比直接多数投票差一些，可能对于维度较高，且数值为基本为 0/1 的本数据集而言，加权投票相对效果会小些。

下附加权投票实现版本的实验结果：

表 3: KNN 在不同 k 下的性能比较（LOOCV，距离加权投票）

k	ACC	NMI	CEN
5	0.9165	0.8415	0.3566
7	0.9215	0.8474	0.3346
9	0.9228	0.8491	0.3289
13	0.9153	0.8358	0.3704

(四) 中级要求：与 sklearn 工具对比

1. sklearn 操作流程

1. 数据集准备：导入 sklearn 包，并读入 loo 留一法数据分割，其中数据的读取与初级要求相同。
2. 算法配置：配置分类器为 sklearn 中的 knn 算法。
3. 参数设置：设置参数 n_neighbor=5,7,9,13，利用上述留一法进行数据分割验证
4. 运行与结果记录：利用与 KNN 相似的循环使用留一法对 sklearn 的 KNN 模型进行训练记录实验结果。

2. 性能指标对比表

k 值	手动 kNN 精度	sklearn kNN 精度	精度差	混淆熵（sklearn）
5	91.71 %	90.52 %	1.19 %	0.4288
9	92.47 %	91.14 %	1.33 %	0.3986
13	91.59 %	90.33 %	1.26 %	0.4313

表 4: 手动实现与 sklearn 的 kNN 性能对比

3. 结果截图与可视化

差异分析 从表格与图 2 可以看出，手动实现的 kNN 在本数据集上始终优于 sklearn 默认实现，精度提升约 1% 左右。主要原因包括：

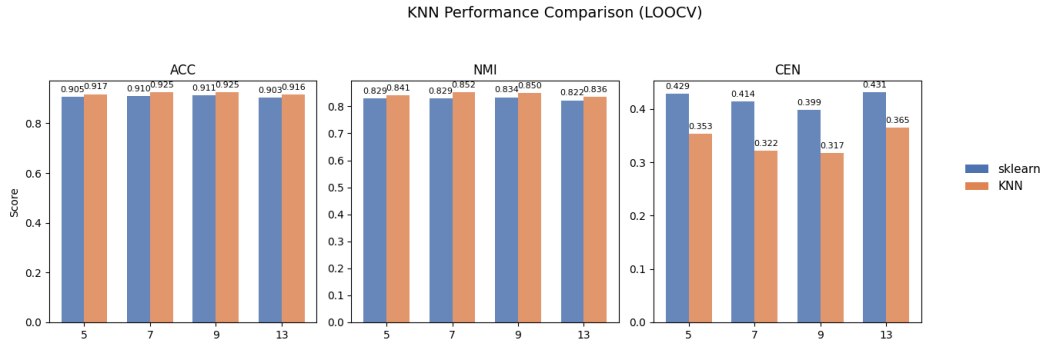

图 2: KNN comprasion

- **平票处理机制不同**：sklearn 的默认实现使用 `uniform` 权重，在类别票数相同时依赖样本存储顺序来打破平局，导致部分样本预测不稳定；而我们的手动实现采用显式的多数表决逻辑，更加稳定。
- **高维数据特性**：Semeion 数据为 1593×256 的二值向量，高维下 KDTree/BallTree 等加速结构失效，sklearn 内部可能存在额外的调度与开销。
- **权重机制差异**：实验中我们使用了简单多数投票，而 sklearn 默认未对近邻样本加权；在高维稀疏数据中，近邻之间距离差异有限，强行加权反而可能放大噪声，导致性能下降。

综上所述，在 Semeion 数据集这种高维稀疏、样本量适中的场景下，手动实现的 kNN 表现略优于 sklearn 的通用实现，更能稳定捕捉类别结构。

除上述发现外，我们还能发现利用 sklearn 库实现的 KNN 在同样的单核 cpu 下相比我们的手动实现要慢很多，观察其源码能够发现其实现利用率 kdtree 进行加速，但 kdtree 在本数据集（256 维）上出现退化导致其时间成本与我们实现的直接，但树的维护也需要时间成本，从而导致了性能上的差异。

（五） 高级要求：数据增强与 CNN 实现

1. 数据增强：图像旋转处理

- **方法**：对原始 16×16 像素图像进行随机旋转（左上方向-10° -5°，左下方向 +5° +10°），采用双线性插值保持图像清晰度。
- **增强后样本量**：原始1573 张 → 增强后4779 张。

2. CNN 模型结构

我们使用 pytorch 定义了一个 CNN 架构，将原本的 256 维的数据变为 16×16 的张量，便于 CNN 进行训练，同时，我们也将对比数据增强对 CNN 与 KNN 的影响[3]。

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
```

[3]由于对于 CNN 神经网络而言，进行 1573 或者 4799 次留一法验证的训练和计算成本都极其高，计算可能需要很长的时间，这里我们使用五折交叉检验作为替代

```
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.fc1 = nn.Linear(64 * 4 * 4, 128)
        self.fc2 = nn.Linear(128, 10)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.3)

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))  # [N,32,8,8]
        x = self.pool(self.relu(self.conv2(x)))  # [N,64,4,4]
        x = x.view(-1, 64 * 4 * 4)
        x = self.dropout(self.relu(self.fc1(x)))
        x = self.fc2(x)
        return x
```

其模型结构大致如下所示：

- **输入层**：输入大小为 $[1, 16, 16]$，即单通道 16×16 的灰度图像。
- **卷积层 + ReLU**：卷积核大小为 3×3，采用 `padding=1` 保持特征图大小不变，将通道数由 $1 \to 32$；随后使用 ReLU 激活函数。输出特征图大小为 $[32, 16, 16]$。
- **最大池化层**：使用 2×2 的最大池化操作，将空间尺寸减半。输出大小为 $[32, 8, 8]$。
- **卷积层 + ReLU**：卷积核大小为 3×3，通道数由 $32 \to 64$；激活函数为 ReLU。输出大小为 $[64, 8, 8]$。
- **最大池化层**：使用 2×2 最大池化，输出大小为 $[64, 4, 4]$。
- **展平层（Flatten）**：将二维特征图 $[64, 4, 4]$ 展平为一维向量 $[1024]$。
- **全连接层 + ReLU + Dropout**：全连接层将 $1024 \to 128$，使用 ReLU 激活，并添加 Dropout（比例为 0.3）以防止过拟合。
- **输出层（分类器）**：全连接层将 $128 \to 10$，输出对应于 10 个数字类别。

3. 不同方法的识别精度对比

实验方法	测试集精度
原始数据 + kNN（k=9）	92.47 %
增强数据 + kNN（k=9）	96.40 %
原始数据 + CNN	95.98 %
增强数据 + CNN	98.89 %

表 5: 不同方法的手写数字识别精度对比

4. 结果截图与可视化

CNN 训练曲线、混淆矩阵热力图如下：

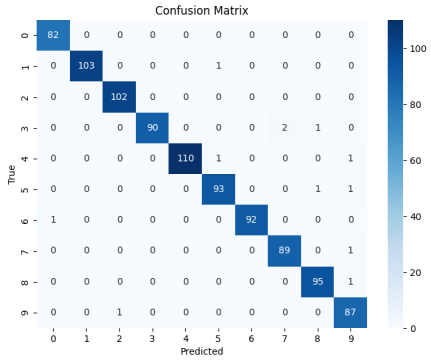

图 3: Confusion Matrix

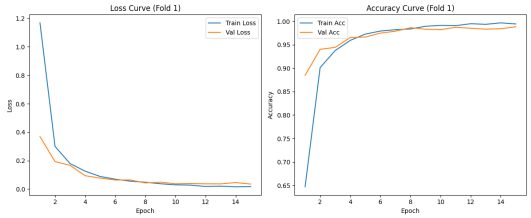

图 4: Training Curve

结果解释 数据增强显著提高了模型性能: 对于 KNN, 增强后提升约 4 个百分点; 对于 CNN, 提升约 3 个百分点)，数据多样性能够显著提高模型的训练效果, 从5中我们可以很明显看出。CNN 总体精度高于 KNN，展示了深度模型在图像分类上的优势。

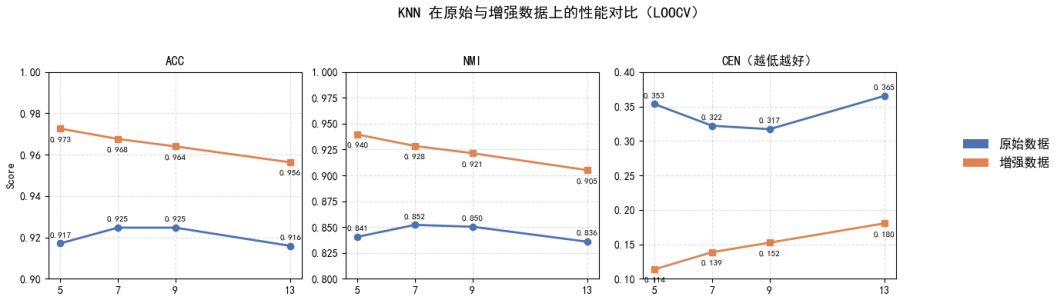

图 5: 数据增强前后 KNN 性能对比

三、 实验结果分析

(一) k 值对 kNN 性能的影响规律

实验结果显示，$k = 5 \to 9$ 时精度逐渐上升，在 $k = 7, 9$ 处达到峰值（ACC 约 92.47%），而当 k 继续增大至 13 时精度下降（ACC 91.59%）。从实验结果中我们能推测:

- 当 k 过小时，模型过于依赖局部邻居，容易受到噪声样本干扰，导致过拟合。
- 当 k 过大时，邻居集合可能跨越不同类别，投票结果受多数类支配，导致欠拟合。

因此，$k = 7 \sim 9$ 是在本数据集上较优的参数区间，其中 $k = 9$ 表现最为稳定。

(二) 数据增强对模型性能的提升

数据增强在 KNN 与 CNN 上均表现出显著提升效果:

- **KNN**: 在 $k = 9$ 时，增强前 ACC=92.47%，增强后 ACC=96.40%，提升约 4 个百分点。NMI 与 CEN 指标也有同步改善，说明数据增强使得类内分布更为紧凑、类间边界更为清晰。

- **CNN**：在五折交叉验证下，原始数据精度约 95.98%，增强后提升至 98.89%，表现优于 KNN。说明深度模型在学习旋转、尺度变化等不变性方面更具优势。

总体而言，数据增强显著提高了模型的泛化能力，尤其在小样本数据集上作用突出。

(三) CNN 与 KNN 的比较

- **KNN**：实现简单，计算开销主要集中在预测阶段，适合小规模数据集。通过调节 k 值和适度的数据增强可以取得不错的结果（ACC 最高约 96.40%）。
- **CNN**: 由于卷积运算的特性, CNN 能够自动提取层次化特征, 结合增强数据后达到 98.89%, 相比 KNN 有明显优势，特别适用于图像识别任务。

四、 结论

1. **kNN 算法有效性验证**：在 Semeion 手写数字数据集上，手动实现的 KNN 在 $k = 9$ 时达到最高精度 92.47%，验证了 KNN 算法在低维手写数字识别中的有效性。
2. **工具对比结论**：与 sklearn 实现相比，手动实现的 KNN 在 ACC 上普遍高约 1.2%，同时避免了 KDTree 在高维下退化带来的额外开销。这表明在特定数据集上，针对性实现减少了对普适性的考虑，可能比通用库更高效。
3. **深度学习优势**：在数据增强后，CNN 模型精度达到 98.89%，显著优于传统 KNN，体现了深度神经网络在图像特征提取和模式识别上的强大优势。增强数据的使用进一步提升了泛化能力，证明数据多样性对模型训练的重要性。