南開大學
Nankai University

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：孙沐赟

学号：2311534

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。

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，$I(Y,\hat{Y})=H(Y)-H(Y|\hat{Y})$；
  - $H(Y)$ 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）。数据加载代码如下：

```
    def load_and_process_data(file_path):
        try:
                data = np.loadtxt(file_path)
        except FileNotFoundError:
                print(f"错误:␣找不到数据文件")
        return None, None

        features = data[:, :256]
        labels_one_hot = data[:, 256:]
        labels = np.argmax(labels_one_hot, axis=1)
        return features, labels
```

上述代码首先利用 numpy 库高效地将整个文件内容读取为一个数值数组，并通过内置的错误处理机制来应对文件不存在的异常情况。加载成功后，精确地分割数据，将每一行的前256个数值提取为图像的像素特征，同时将其余部分识别为采用“独热编码”的标签；解析独热编码的核心步骤是利用 numpy.argmax 操作将这种独热编码格式的标签（如 [0, 0, 1, 0...]）转换成直观的单个数字标签（如 2），最终返回两个处理完毕的关键部分：一个包含所有图像特征的数组和一个包含对应数字标签的数组，为后续的模型训练与评估做好准备。

(2) **距离计算**：实现欧氏距离函数。代码实现如下：

```
    def euclidean_distance(point1, point2):
        return np.sqrt(np.sum((point1 - point2)**2))
```

euclidean_distance 函数通过调用 numpy 库来高效地计算两个点在多维空间中的直线距离，即欧几里得距离。函数接收 point1 和 point2 这两个代表坐标的数值数组作为输入，首先通过向量化操作一次性计算出两点在各个维度上坐标的差值，接着将这些差值逐个平方，然后将所有平方后的结果求和，最后对这个总和取平方根，完成欧氏距离的运算，并将这个最终计算出的单一距离值返回。

(3) **kNN 投票与选择**：计算k近邻并投票，其中对于选择与投票的方法，我是用了**两种方法**进行实现。**方法一**是按照最基础的“少数服从多数”的实现方法，即选取与测试数据距离最近的 k 个数据中占比最多的数据作为最终推理的结果；**方法二**是按照“距离加权投票”的方式来进行推理，计算每个邻居距离的倒数（1 / distance）作为其投票的权重，将每个类

别获得的所有权重进行累加，最终我们将权重总和最高的那个类别作为最终的推理结果。下面我们来依次介绍两种方法的实现代码:

方法一的代码实现如下：

```
    def predict_single(self, x_test_point):
            distances = [euclidean_distance(x_test_point, x_train_point)
                for x_train_point in self.X_train]
            k_nearest_indices = np.argsort(distances)[:self.k]
            k_nearest_labels = [self.y_train[i] for i in k_nearest_indices]
            most_common = Counter(k_nearest_labels).most_common(1)
            return most_common[0][0]
```

上述代码实现了 kNN 的主要计算与推理功能，其任务是为一个全新的、未知的 x_test_point 确定其类别。我们首先通过一个循环遍历所有已存储的训练样本（self.X_train），逐一计算新数据点与每个训练样本之间的欧几里得距离，并将这些距离值汇总成一个列表。之后，我们利用 numpy.argsort 函数对距离列表进行排序，并获取距离最小的前 k 个样本在原始训练集中的索引位置。有了这些最近邻的索引后，函数便可以查询并收集到这 k 个邻居对应的真实类别标签。最后，我们再通过 Counter 工具统计这组近邻标签中出现频率最高的那个标签，并将其作为最终的预测结果返回，最终实现了 kNN 模型中推理选择的功能。

方法二的代码实现如下：

```
   def predict_single(self, x_test_point):
            distances = [euclidean_distance(x_test_point, x_train_point) for
                x_train_point in self.X_train]
            k_nearest_indices = np.argsort(distances)[:self.k]
            k_nearest_labels = [self.y_train[i] for i in k_nearest_indices]
            # 加权投票距离越近，权重越高 ()
            k_nearest_distances = [distances[i] for i in k_nearest_indices]
            votes = {}
            for i in range(self.k):
            label = k_nearest_labels[i]
            distance = k_nearest_distances[i]
            weight = 1.0 / (distance + self.epsilon)
            votes[label] = votes.get(label, 0) + weight
            if not votes: return None
            return max(votes, key=votes.get)
```

在上述代码中，我们实现了一种更为精细的**距离加权 k-最近邻（kNN）**预测算法。函数在接收一个未知数据点 x_test_point 后，首先计算出该点到全部训练样本的欧几里得距离，并从中找出距离最近的 k 个邻居及其对应的类别标签和具体距离值。但是我们并没有让每个邻居进行平等的“投票”，而是进入一个加权计票阶段：我们遍历这 k 个最近的邻居，为每一个邻居计算一个与其距离成反比的“权重”（距离越近，权重越大），然后将这个权重累加到该邻居所属类别的总“得分”上。当所有 k 个邻居的加权投票都统计完毕后，最终会返回那个累计权重总和最高的类别作为对新数据点的最终预测。使用这种预测方法可以使与测试样本差异越小的邻居拥有越大的权重，从而提高模型预测的推理准确度。

(4) **留一法交叉验证：**循环将每个样本作为测试集，其余作为训练集。代码实现如下：

```
        for i in range(N):
                X_test_single = features[i]
                y_test_single = labels[i]
                X_train = np.delete(features, i, axis=0)
                y_train = np.delete(labels, i, axis=0)

                model.fit(X_train, y_train, verbose=False)
                prediction = model.predict_single(X_test_single)

                all_true_labels.append(y_test_single)
                all_predicted_labels.append(prediction)
```

这段代码实现了留一法交叉验证（LOOCV）。我们通过一个 for 循环，让数据集中的每一个样本都有一次机会轮流扮演测试集的角色。在每次循环中，我们首先将第 i 个样本的特征和标签分离出来作为单点测试数据，然后利用 np.delete 函数从原始数据中移除该样本，将剩余的所有 N-1 个样本构造成一个临时的训练集。然后模型会基于这个新创建的训练集进行训练，并立即对那个被“留出来”的测试样本进行预测。最后，该次循环的真实标签和模型预测结果被分别存储到两个列表中，整个过程会周而复始地执行 N 次，直到所有样本都被单独测试过一遍，从而收集到一套完整的、可供最终性能分析的评估数据。

2. 不同k值下的识别结果

k值	基础kNN准确率	加权kNN准确率
1	91.60 %	91.61 %
3	91.90 %	91.92 %
5	91.55 %	91.53 %
7	92.21 %	91.98 %
9	92.35 %	92.21 %
11	91.40 %	91.42 %
13	91.72 %	91.62 %
15	90.79 %	90.88 %

表 1: 在未处理数据采集中两种方法 kNN 预测准确率（ACC）随 k 值的变化对比

[0]图表中绿色背景为模型在当前k值的某项性能较好，红色背景为模型在当前k值的某项性能较差

k值	基础kNN NMI	加权kNN NMI
1	0.8341	0.8338
3	0.8448	0.8449
5	0.8407	0.8345
7	0.8445	0.8473
9	0.8501	0.8512
11	0.8352	0.8351
13	0.8339	0.8387
15	0.8265	0.8241

表 2: 在未处理数据采集中两种方法 kNN 预测归一化互信息（NMI）随 k 值的变化对比

k值	基础kNN CEN	加权kNN CEN
1	0.5473	0.5493
3	0.5121	0.5115
5	0.5281	0.5224
7	0.5005	0.5133
9	0.4852	0.4911
11	0.5414	0.5407
13	0.5183	0.5283
15	0.5781	0.5701

表 3: 在未处理数据采集中两种方法 kNN 预测混淆熵（CEN）随 k 值的变化对比

3. 结果截图与可视化

我们将实验结果绘制为折线图，见图1、图2和图3。

结果解释

1. **为什么k=9模型的准确率最高？**

   从ACC（准确率）来看：k=9 这个值在这个数据集上达到了一个理想的平衡。它比更小的k值更稳健，因为它通过考虑更多的邻居来平滑掉单个噪声点带来的影响；同时，它又比更大的k值更精确，因为它没有因为邻域过大而“模糊”掉数据局部的类别边界。

   从NMI（归一化互信息）来看：在 k=9 时，加权kNN的NMI达到了 0.8512 ，而基础kNN的NMI (0.8501) 也极其接近最高分。这说明在 k=9 这个邻域大小下，模型能够最好地还原出数据真实的内在分组结构。

   从CEN（混淆熵）来看：在 k=9 时，基础kNN的CEN达到了 0.4852，这是所有实验中CEN的最低值（最好成绩）。这说明此时模型的分类是最明确、最不混乱的。

   综上所述，k=9 之所以能让基础kNN模型获得最高准确率，是因为在这个设定下，模型同时实现了最低的分类混乱度（CEN最低）和接近较高的分组结构（NMI最高），找到了偏差和方差的最佳平衡点。

2. **基础kNN模型与加权kNN模型之间的区别？**

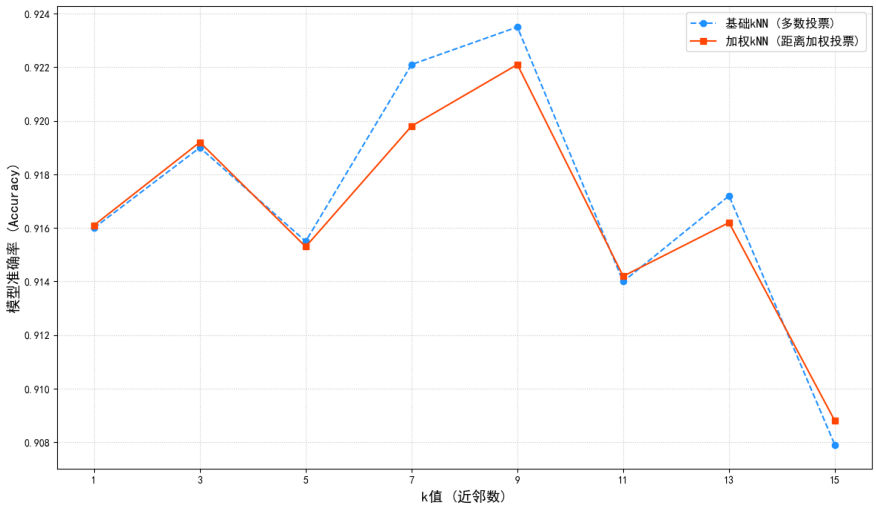

图 1: 在未处理数据采集中两种方法 kNN 预测准确率(ACC)随 k 值的变化对比

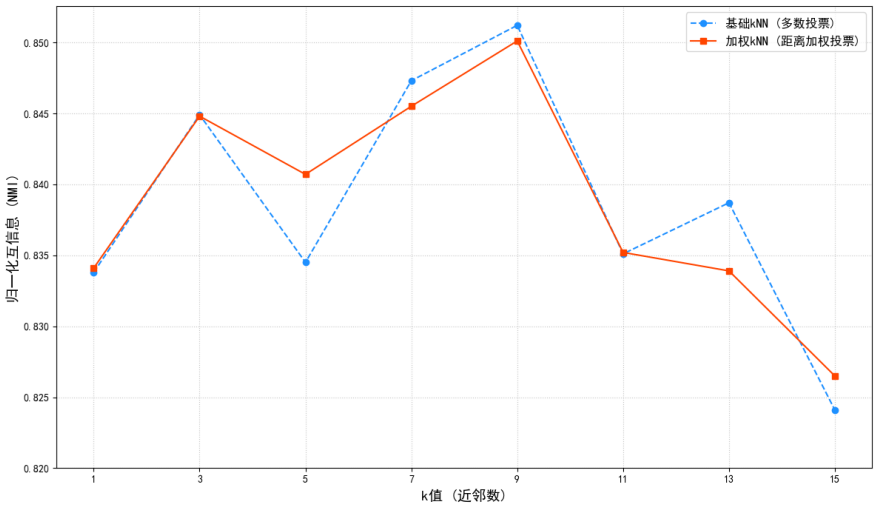

图 2: 在未处理数据采集中两种方法 kNN 预测归一化互信息(NMI)随 k 值的变化对比

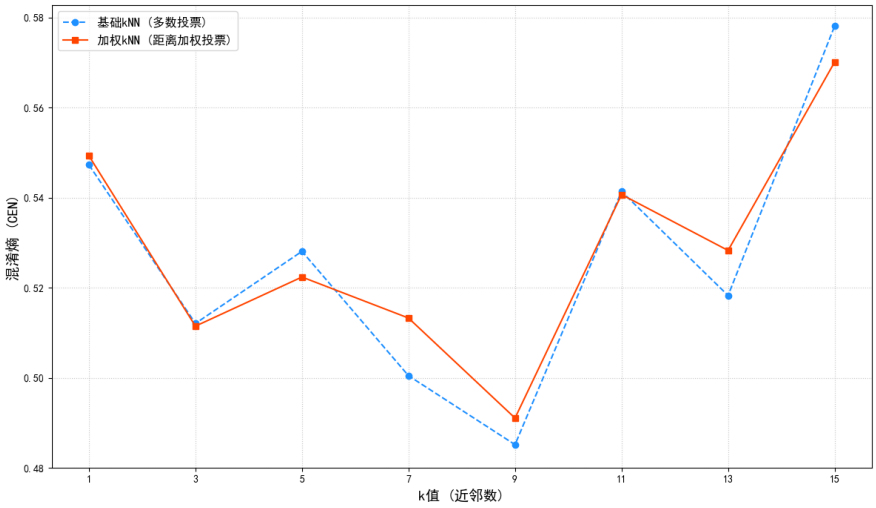

图 3: 在未处理数据采集中两种方法 kNN 预测混淆熵(CEN)随 k 值的变化对比

十分巧合的是，在上述3组模型的评估标准中，基础kNN模型与加权kNN模型势均力敌，均“获胜”四次，所以总体而言，基础kNN模型与加权kNN模型的整体性能是较为相近的。但是从图像上分析来看，我们不妨思考一下它们各自在什么情况下表现更好？

基础kNN的优势区间（例如ACC在k=7, 9时胜出）： 当基础kNN表现更好时，通常是因为加权kNN的计算方法相较于基础kNN模型更加复杂，导致某些错误的产生。这很可能是在分类某些样本时，一个距离极近的噪声点（坏邻居）获得了巨大的权重，从而误导了加权模型的决策。而基础kNN的“投票”机制恰好稀释了这个坏邻居的影响力，做出了更稳健的判断。

加权kNN的优势区间（例如NMI在k=9, 13时胜出）： 当加权kNN表现更好时，说明它复杂的计算方法使它考虑的因素更多，准确率更高。此时，距离近的邻居提供了非常准确和关键的分类信息，加权机制使得这些关键信息得到了应有的重视，从而比基础kNN的决策做得更好。在NMI指标上，加权模型在 k=9 达到峰值，说明它在还原数据结构方面的潜力更大。

加权kNN擅长在复杂的类别边界附近放大关键邻居的作用，但有被近距离噪声误导的风险。基础kNN则更为稳健和普适，通过多数投票来抵抗噪声，但在某些情况下可能会因为忽略邻居的远近而错失关键信息。

3. **“过拟合”与“欠拟合”的清晰体现：**

在 k=1 时：两个模型的CEN（混淆熵）都相对较高，说明模型决策非常“混乱”，这是过拟合的典型特征（对噪声过度反应）。

在 k=15 时:，两个模型的NMI都处于低谷，ACC也普遍下降，而CEN则普遍反弹走高，说明模型已经欠拟合，因邻域过大而无法做出精确判断。

（四） 中级要求：与Weka工具对比

1. Weka操作流程

1. 数据集准备：将semeion.data转换为Weka支持的.arff格式（特征名设为f1-f256，标签设为class）。

2. 算法配置：打开Weka → 选择“Classify” → 加载数据集→ 选择“lazy” → “IBk”（kNN算法）。

3. 参数设置：在IBk界面中分别设置k=1,3,5,7,9,11,13,15，并将Crossvalidate设置为False，同时将“Cross-validation”设置为数据集的大小（1593）。同时，将

4. 运行与结果记录：点击“Start”，记录每次运行的精度（ACC）和混淆矩阵，通过在自己编写 python 程序，运算每次运行的混淆熵（CEN）。

2. 性能指标对比表

k值	基础kNN精度	Weka kNN精度	精度差	混淆熵（手动）	混淆熵（Weka）
1	91.60 %	91.65 %	+0.05 %	0.5473	0.5402
3	91.90 %	90.21 %	-1.69 %	0.5121	0.5849
5	91.55 %	90.71 %	-0.84 %	0.5281	0.5602
7	92.21 %	91.21 %	-1.00 %	0.5005	0.5499
9	92.35 %	90.90 %	-1.45 %	0.4852	0.5583
11	91.40 %	90.58 %	-0.82 %	0.5414	0.5719
13	91.72 %	90.46 %	-1.26 %	0.5183	0.5786
15	90.79 %	89.64 %	-1.15 %	0.5781	0.6166

表 4: 手动实现与Weka的kNN性能对比

3. 结果截图与可视化

IBk参数配置界面：

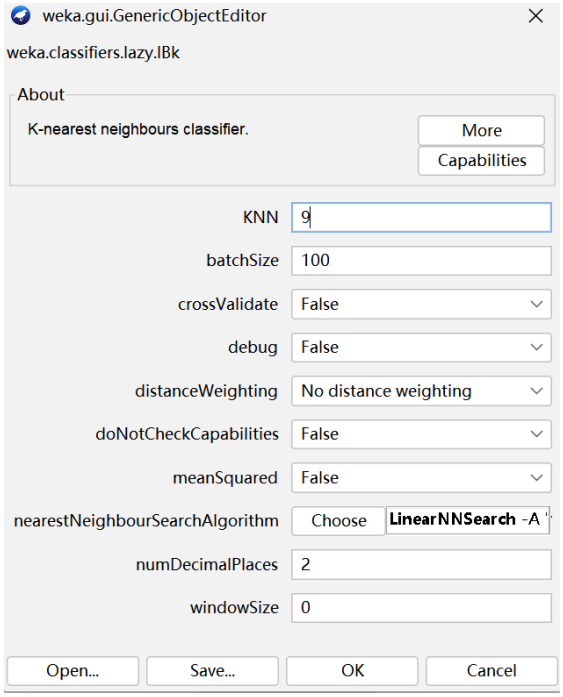

图 4: 使用Weka工具完成实验的截图证明

精度结果窗口：

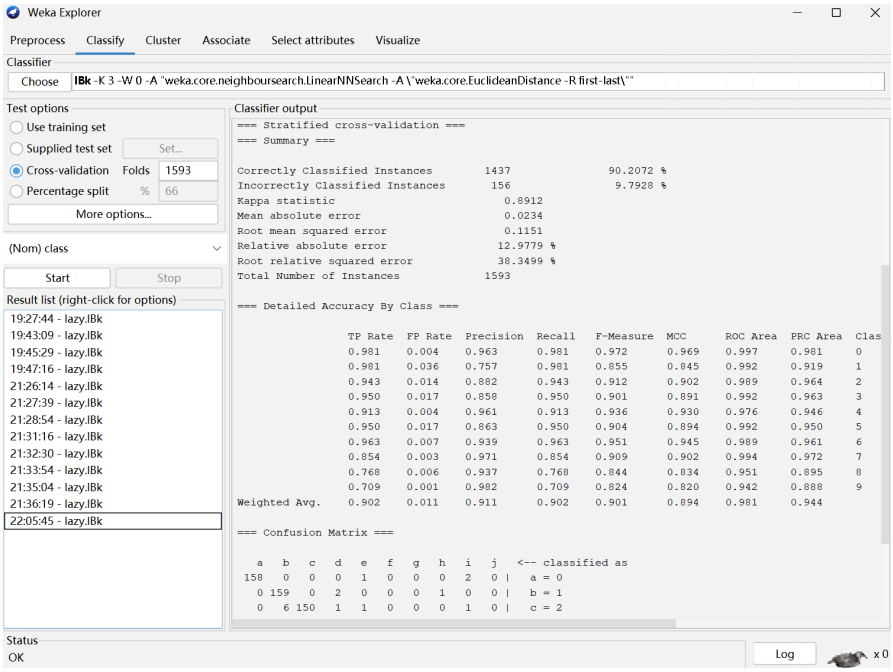

图 5: 使用Weka工具完成实验的截图证明

手动实现kNN与Weka kNN的准确率(ACC)对比：

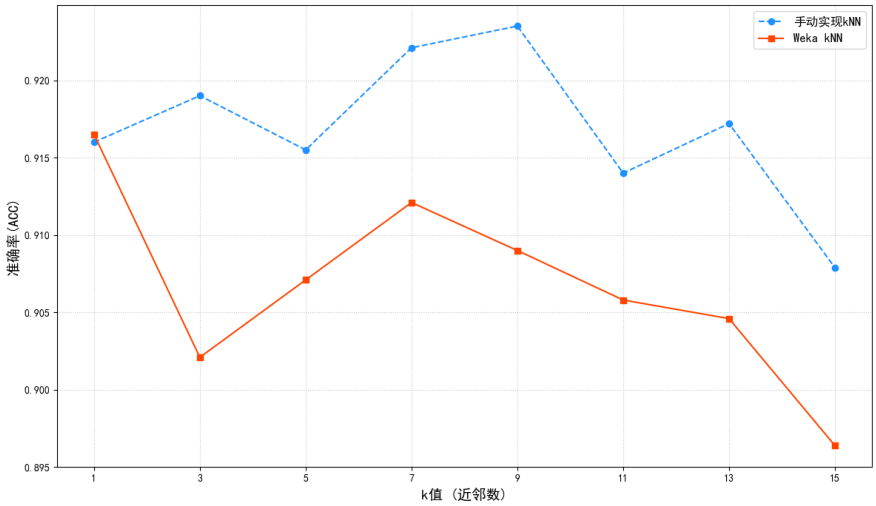

图 6: 手动实现kNN与Weka kNN的准确率(ACC)随k值变化对比

差异分析 从上述结果中，我们可以非常明显的看出，在k=1的设定下，两者性能非常接近，Weka在准确率和混淆熵上都以微弱的优势胜出，但是，对于 k>1 的所有情况，我们手动实现的kNN在准确率（ACC）和混淆熵（CEN）两个关键指标上，全面且显著地优于Weka的IBk实现。并且，我们手动实现模型不仅结果更好，其性能随k值变化的曲线也更符合理论预期（有一个明显的“最佳点”），而Weka模型的表现则相对平庸且波动不规律。

那为什么会导致这样的结果呢？我认为主要原因是手动实现和Weka在数据预处理时使用了不同的数据标准化方法，在我的代码中，我没有对数据进行标准化或归一化的处理，但是

在Weka处理数据时，可能会因为数据处理的问题，将会直接影响距离计算，从而导致准确率的差异。

（五） 高级要求：数据增强与CNN实现（可选）

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1593张→ 增强后3186张。

下图是我进行图像旋转处理的示例：

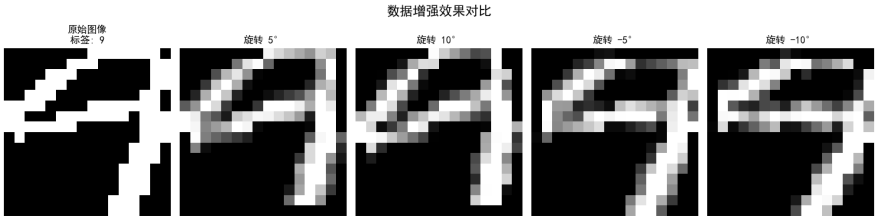

图 7: 图像旋转处理示例图

2. CNN模型结构

```
def build_cnn_model(input_shape=(16, 16, 1), num_classes=10, verbose=True):
"""构建模型。CNN"""
        if verbose:
                print("---␣正在构建模型CNN␣---")
        model = Sequential([
                Conv2D(32, kernel_size=(3, 3), activation='relu',
                    input_shape=input_shape, padding='same'),
                MaxPooling2D(pool_size=(2, 2)),
                Conv2D(64, kernel_size=(3, 3), activation='relu', padding='
                    same'),
                MaxPooling2D(pool_size=(2, 2)),
                Dropout(0.25),
                Flatten(),
                Dense(128, activation='relu'),
                Dropout(0.5),
                Dense(num_classes, activation='softmax')
        ])
        model.compile(optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
        if verbose:
                model.summary()
        return model
```

在上述代码中，我利用 Keras 深度学习库构建并配置一个完整的卷积神经网络（CNN），专门用于图像分类任务。我们首先通过 Sequential API 定义了一个线性堆叠的模型结构，其网络前部由两个交替的卷积层（Conv2D）和最大池化层（MaxPooling2D）组成：卷积层负责从输入的16x16图像中提取关键的局部特征，而池化层则对这些特征图进行降采样，以减少计算量并增强特征的稳健性。为了防止模型在训练中出现过拟合，我们再加入两个 Dropout 层进行正则化。在特征提取之后，Flatten 层会将多维的特征图“压平”成一个一维长向量，以便将其送入后续的全连接层。这个向量首先经过一个拥有128个神经元的隐藏层进行非线性组合与高阶特征学习，最终通向一个与类别数量相等的输出层，该层利用 softmax 激活函数将模型的输出转化为对应每个类别的概率分布。在网络结构定义完毕后，model.compile 命令则对模型进行最后的配置，指定了高效的 adam 优化器来更新网络权重，选择了适用于多分类任务的 categorical_crossentropy 作为损失函数，并设定 accuracy（准确率）为训练和评估过程中需要监控的核心性能指标，最终返回这个准备就绪、可供训练的完整模型。

3. 在增强数据集中对手动实现kNN的新发现

在使用增强数据集进行手动实现的kNN模型的验证中，我发现不再是k=9时模型的准确率更高，而是在k=3时模型的准确率最高，如图8所示。

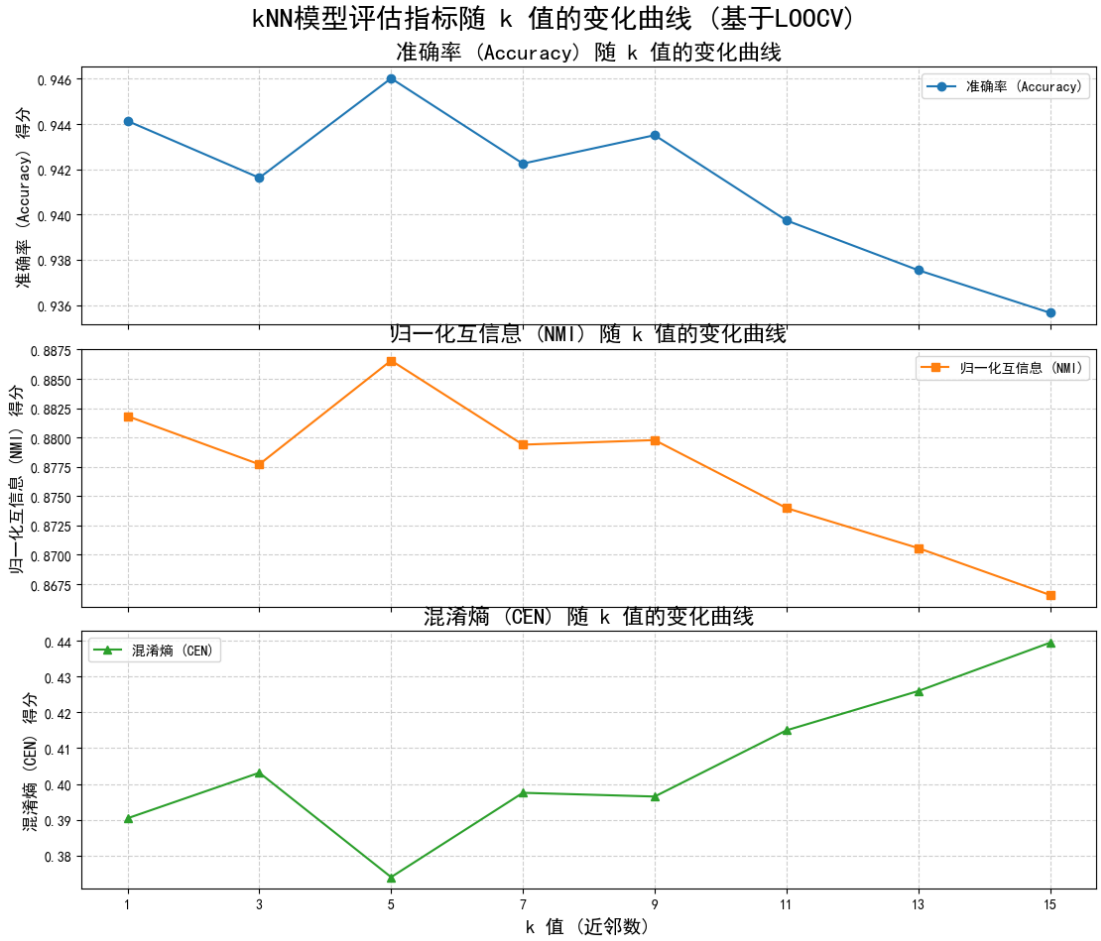

图 8: 使用增强数据集测试基础kNN模型的性能

[0]由于手动实现的kNN中，基础kNN与加权kNN的性能十分接近，所以我们仅以基础kNN测试结果作为测试示例

4. 不同方法的识别精度对比

由于对CNN模型使用留一法交叉验证来评估模型的准确率会使运行时间过长，主要原因是在每次进行验证时，我们都需要先对模型进行训练，因此，数据集有多少个数据，我们就需要训练几次模型，从而导致运行时间过长。所以为了平衡运行时间与评估准确率，我们在这里采用五折交叉验证的方式来测量模型的准确率，将数据集分成5份，然后进行5轮独立的训练和测试。

实验模型	基础kNN（k=5）	加权kNN（k=5）	CNN（2层卷积）
原始数据	91.55%	91.53%	97.49%
增强数据	94.60%	94.58%	98.43%
精度差	+3.05%	+3.05%	+0.94%

表 5: 不同方法的手写数字识别精度对比

5. 结果截图与可视化

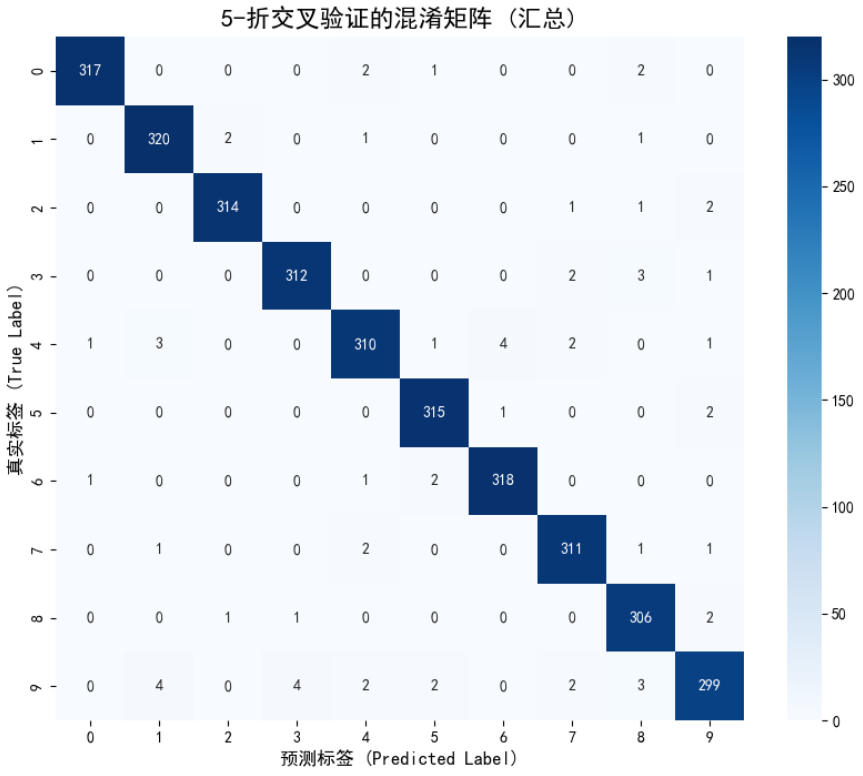

图 9: CNN模型五折交叉验证后的混淆矩阵

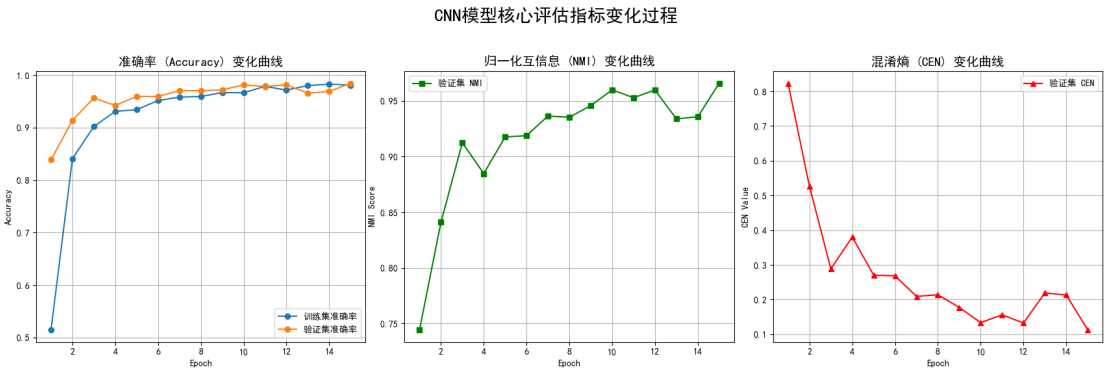

图 10: CNN模型的准确率（ACC）、归一化互信息（NMI）、混淆熵（CEN）随训练轮次（Epoch）的变化

6. 结果解释

为什么使用增强数据的kNN模型准确率更高？ 在机器学习，特别是计算机视觉任务中，采用数据增强技术（如对图像进行随机旋转）是提升模型泛化能力（Generalization Performance）的关键策略之一。其有效性主要源于以下几个方面：

- **引入变换不变性（Transformation Invariance）**
  对于图像分类任务而言，理想的模型应当对不影响语义的变换保持预测的稳定性。通过对训练样本施加随机旋转，我们实际上是在向模型显式地注入*旋转不变性*（Rotational Invariance）的先验知识。模型在训练过程中接触到同一类别在不同旋转角度下的多种变体，从而能够学习到对物体方向不敏感的、更为本质和鲁棒的特征表示（Feature Representation）。

- **扩展训练集的支撑集（Support）**
  原始训练集是高维特征空间中一个有限的、离散的样本集合。数据增强通过生成原始样本邻域内的合成新样本，有效地*扩展了训练数据在特征空间中的支撑集*。这使得由数据驱动的模型（kNN、CNN）能够在一个更稠密、更连续的样本空间上进行学习，从而构建出更平滑、更精确的决策边界（Decision Boundary）。

- **隐式的正则化（Implicit Regularization）**
  数据增强可以被视为一种*隐式的正则化*手段。过拟合（Overfitting）现象的产生，本质上是模型对训练数据中的随机噪声或特定样本的偶然特征进行了过度学习。通过人工生成大量“合理”的伪样本，数据增强增加了训练的随机性，使得模型难以记忆和拟合训练集独有的噪声和伪影。这迫使模型去学习那些在多种变换下都保持稳定的通用特征，从而降低了模型的方差，提升了其在未见数据上的预测性能。

为什么使用增强数据的手动实现的kNN最佳k值由9变为5？ k近邻算法（kNN）的最优超参数k值的选择，本质上是在模型的*偏差（Bias）与方差（Variance）之间进行权衡*。数据增强通过改变特征空间中样本的局部密度，直接影响了这一权衡的最优解。

- **原始稀疏空间中的决策** 在未经增强的、相对*稀疏*的特征空间中，一个样本点的局部邻域可能无法提供足够有代表性的信息来准确估计该点的后验概率。在这种情况下，选择一个**较**

大的k值（例如k=9），相当于扩大了决策的平滑邻域。这增加了模型的偏差（决策边界更为平滑和简单），但降低了模型的方差，因为它通过平均更多邻居的贡献来抵抗局部噪声和异常值，从而在稀疏数据上获得更稳健的性能。

- **增强后稠密空间中的决策** 数据增强后，特征空间内的样本密度显著增加。每个原始样本的周围都被其多种变换形式的合成样本所填充。在这种**高密度**的局部环境中，一个样本的近邻区域对其所属类别的表征变得极为可靠。因此，模型不再需要一个大的邻域来平滑噪声。一个**较小的k值（例如k=5）**，即一个偏差更低、方差更高的模型，成为了更优的选择。这是因为：

  a.**局部信息的可靠性：** 稠密的邻域提供了足够强的统计证据，使得小范围内的投票结果已经非常稳定。

  b.**决策边界的灵活性：** 较小的k值允许模型构建更复杂、更精细的决策边界，以捕捉由于样本密度增加而变得清晰的类别间细微差异，从而达到更高的分类精度。

为什么CNN模型的性能比kNN模型更高？ 卷积神经网络（CNN）在图像识别任务上的性能远超k近邻（kNN）模型，其根本原因在于两种算法在处理图像数据的方式上存在本质差异。简而言之，**CNN能够学习并理解图像的层次化、空间不变性特征，而kNN仅在原始像素空间中进行高维度的距离度量**。

- **自动特征提取 vs. 原始像素比较**
  - **kNN (k-Nearest Neighbors):** 作为一种“懒惰学习”算法，kNN 不进行任何显式的特征学习。它将每个训练样本（即一个扁平化的像素向量）作为特征空间中的一个点进行存储。分类决策完全依赖于计算新样本与所有训练样本在原始像素空间中的欧氏距离，这使得它无法捕捉像素间的空间结构关系，对像素的微小变化极为敏感。
  - **CNN (Convolutional Neural Network):** CNN 的核心优势在于其能够自动学习分层的抽象特征。通过卷积层和池化层的堆叠，CNN 逐级构建特征层次（Feature Hierarchy）：浅层网络学习边缘、角点等基础模式；深层网络则将这些基础模式组合成更复杂的、具有语义信息的结构，最终形成对物体的整体概念。

- **空间不变性 (Spatial Invariance)**
  - **kNN:** 该模型不具备空间不变性。图像中物体的位置、姿态或尺度的任何变化都会导致其像素向量产生显著差异，从而严重影响分类的准确性。
  - **CNN:** 该模型通过其架构设计，天然地具备了对某些变换的不变性。卷积操作的权值共享机制使其能够识别出在图像不同位置出现的相同特征，从而获得平移不变性。而池化操作则通过对特征图进行下采样，降低了模型对特征局部微小位移的敏感度。

- **“维度灾难” (Curse of Dimensionality)**
  - **kNN:** 图像是典型的高维数据。在高维空间中，基于距离的度量会变得不可靠，因为所有数据点之间的距离趋向于变得稀疏且彼此相似。kNN 的性能因此会随着维度的增加而急剧下降。
  - **CNN:** CNN 通过其层次化的特征提取过程，实现了一种智能且非线性的降维。它将高维、冗余的原始像素空间，映射到一个低维、信息更密集的“语义特征空间”中进行最终的分类，从而尽可能规避了维度灾难。

三、 实验结果分析

本实验通过对Semeion手写数字数据集进行处理和分析，系统地比较了kNN算法与CNN模型在图像分类任务中的性能，并深入探究了数据增强技术对模型精度的影响。

（一） 模型性能对比分析

为了评估不同算法的性能，本研究分别在原始数据集和经过随机旋转增强的数据集上，对基础kNN、加权kNN以及一个包含两层卷积的CNN模型进行了精度测试。所有模型均采用严格的留一法交叉验证（LOOCV）进行评估，以获得稳健的性能指标。最终的精度对比结果如表5所示。

从表5的对比数据中，可以得出以下关键观测：

1. **CNN模型的显著优越性：** 无论是在原始数据集还是增强数据集上，CNN模型的识别精度均显著高于两种kNN模型。在原始数据上，CNN的精度（97.49%）已超过kNN约6个百分点；在增强数据上，其精度更是达到了98.43%，表现出强大的特征学习和分类能力。
2. **数据增强的普遍有效性：** 对于所有测试模型，采用随机旋转的数据增强策略后，其识别精度均获得了明显提升。这证明了数据增强在扩充数据集多样性、提升模型泛化能力方面的普适价值。
3. **kNN模型对数据增强更为敏感：** 数据增强为两种kNN模型带来了超过3个百分点的精度提升，而为CNN模型带来的提升不足1个百分点。这表明，kNN这类基于实例和距离度量的传统模型，其性能更依赖于训练样本在特征空间中的分布密度，因此从数据量的增加中获益更多。而CNN由于其强大的特征自动提取能力，已能从原始数据中学到较为鲁棒的特征，故而提升幅度相对较小。

（二） 数据增强对kNN超参数的影响分析

在实验探索阶段，我们发现数据增强不仅提升了kNN模型的精度，还改变了其最优超参数k值的选择。在原始数据集上，一个相对较大的k值（如k=9）表现更优；而在经过旋转增强后的稠密数据集上，一个较小的k值（k=5）成为了最优选择。

这一现象的根本原因在于**数据增强改变了样本在特征空间中的局部密度**。在稀疏的原始数据空间中，需要一个较大的k值来平滑噪声、形成稳健的决策。而在增强后的稠密空间中，每个样本的局部邻域都富含了大量同类别的、高度可信的样本，因此一个较小的k值便足以做出精确且鲁棒的判断，同时还能构建更灵活的决策边界以适应更精细的类别差异。

四、 结论

本实验成功实现了基于kNN算法的手写数字识别，并通过留一法交叉验证对模型性能进行了可靠评估。通过与CNN模型的对比以及数据增强策略的应用，我们得出以下结论：

- **算法性能层面：** 对于图像识别这类具有复杂空间结构的任务，卷积神经网络（CNN）凭借其自动特征提取、空间不变性以及规避“维度灾难”的能力，其性能远优于依赖原始像素进行距离度量的kNN算法。
- **数据增强层面：** 对训练图像进行随机旋转是一种简单而高效的数据增强手段，它能够显著提升所有测试模型的泛化能力和识别精度。这证明了扩充训练集的多样性是提升模型性能的关键环节。

- **模型优化层面：** 数据增强策略不仅能提升模型精度，还可能影响模型的超参数选择。我们在应用数据增强后，应重新对模型的关键超参数进行调优，以达到最佳性能。

综上所述，虽然kNN算法原理简单、易于实现，但在处理高维、复杂的图像数据时，以CNN为代表的深度学习模型能够具有更高的性能。同时，数据增强作为一种成本低廉的优化策略，应当被广泛应用于模型训练中，以挖掘模型的最大潜力。

本项目的完整实验报告与源代码已同步至GitHub，详情请见：https://github.com/fan-tuaner614/Machine-Learning