南開大學
Nankai University

南　　开　　大　　学

计　算　机　学　院

机器学习实验报告

Lab1 基于 kNN 的手写数字识别实验

马溟怡

学号：2311061

专业：计算机科学与技术

2025 年 9 月 28 日

目录

一、 实验内容

(一) 实验名称

基于 kNN 的手写数字识别实验

(二) 实验目的

1. 实现 k 近邻（k-Nearest Neighbor, kNN）分类器;
2. 掌握留一法（Leave-One-Out, LOO）交叉验证流程;
3. 对比自实现结果与 Weka 内置 kNN 精度;
4. 学会用图表展示实验过程与结论。

代码

逐列访问平凡算法

```
void ord()
{
    double head,tail,freq,head1,tail1,timess=0; // timers
    init(N);
    QueryPerformanceFrequency((LARGE_INTEGER *)&freq );
    QueryPerformanceCounter((LARGE_INTEGER *)&head);
    for (int i=0; i<NN; i++)
        for (int j=0; j<NN; j++)
            col_sum[i] += (b[j][i]*a[j]);
    QueryPerformanceCounter ((LARGE_INTEGER *)& tail) ;
    cout << "\nordCol:" <<(tail-head)*1000.0 / freq<< "ms" << endl;
}
```

(三) 实验环境

编程语言：Python 3.12（虚拟环境.venv）

IDE：PyCharm Community Edition

主要依赖库：

numpy 2.3.3

matplotlib 3.10.6

scikit-learn 1.7.2

数据集：semeion.data（UCI 手写数字识别数据集，共 1593 个样本，256 维特征 + 1 个标签）

机器学习工具：Weka 3.8.6 ————————-

NKU

(四) 基础任务

1. 算法思路

1. **距离计算**
   对 x 与训练集中每一个样本 y，计算欧式距离：

$$d(x,y)=\sqrt{\sum_{i=1}^{256}(x_i-y_i)^2}$$

2. **排序与选择邻居**
   将所有训练样本按距离从小到大排序，选择前 k 个最近的邻居。

3. **投票机制**
   统计这 k 个邻居的标签出现次数，票数最多的标签作为最终分类结果。若出现平局，则选择编号最小的标签。

4. **验证方法**
   采用留一法交叉验证（Leave-One-Out, LOO）：每次取一个样本作为测试集，其余作为训练集，重复 N 次（$N=1593$），统计分类正确的比例作为准确率。

2. 代码

阶 knn 算法

```python
import numpy as np
from sklearn.neighbors import KNeighborsClassifier

#  数据加载
def Img2Mat0(filename):
    """
    读取 semeion.data 文件
    每行：256 个像素值 +10 个 one-hot 标签
    返回：
    X : (N,256) 样本矩阵
    y : (N,) 标签数组 (0-9)
    """
    raw = np.loadtxt(filename)
    X = raw[:, :256].astype(float)
    Y_onehot = raw[:, 256:]
    y = np.argmax(Y_onehot, axis=1).astype(int)
    return X, y


# 自写版 KNN
def classify0(inX, dataSet, labels, k):
    """
    inX: 待分类样本
    dataSet: 训练样本 (N, d)
    labels: 训练标签 (N,)
```

NKU

```
    k: 邻居数
    """
    diff = dataSet - inX
    dist = np.sqrt(np.sum(diff ** 2, axis=1))  # 欧式距离
    idx = np.argsort(dist)  # 排序
    votes = {}
    for i in range(k):
        lab = labels[idx[i]]
        votes[lab] = votes.get(lab, 0) + 1
    # 返回票数最多的类别（若平局，取最小类别编号）
    max_votes = max(votes.values())
    winners = [lab for lab, v in votes.items() if v == max_votes]
    return int(min(winners))


def loo_eval_handmade(X, y, k):
    """手写版 LOO 交叉验证"""
    n = X.shape[0]
    correct = 0
    for i in range(n):
        X_train = np.vstack((X[:i], X[i + 1:]))
        y_train = np.hstack((y[:i], y[i + 1:]))
        pred = classify0(X[i], X_train, y_train, k)
        if pred == y[i]:
            correct += 1
    return correct / n


# 主函数
if __name__ == "__main__":
    X, y = Img2Mat0("semeion.data")

    print("\n===== 自写版 KNN LOO 结果 =====")
    for k in [1, 3, 5]:
        acc = loo_eval_handmade(X, y, k)
        print(f"k={k}, 准确率={acc:.4f}")
```

3. 操作过程

首先安装相关库，将代码文件与 semeion.data 放在同一文件夹下

运行结果

NKU

```
===== 自写版 KNN LOO 结果 =====
k=1, 准确率=0.9190
k=3, 准确率=0.9046
k=5, 准确率=0.9046
```

（五） 中级任务

1. 目的

在 Weka 这个机器学习软件里，使用自带的 KNN 算法（z 在 Weka 里叫 IBk），对同一份数据集做同样的留一法交叉验证，然后把结果和自写的 KNN 程序对比。

2. 操作过程

1. 打开 **Weka** → Explorer → Preprocess → Open file → 选择 `semeion.data`。但是此处 Weka 提示要打开 Arff 格式的数据集，所以使用下面的脚本来将 semeion.data 转化为 Arff 格式。

转化脚本

```python
import numpy as np

def convert_to_arff(input_file, output_file):
    # 读原始数据
    raw = np.loadtxt(input_file)
    X = raw[:, :256].astype(int)        # 特征 256 维
    Y_onehot = raw[:, 256:]
    y = np.argmax(Y_onehot, axis=1)    # 标签 0–9

    with open(output_file, "w") as f:
        # 写 Header
        f.write("@RELATION semeion\n\n")
        # 写 256 个像素特征
        for i in range(256):
            f.write(f"@ATTRIBUTE pixel{i} NUMERIC\n")
        # 写 class 标签
        f.write("@ATTRIBUTE class {0,1,2,3,4,5,6,7,8,9}\n\n")
        f.write("@DATA\n")

        # 写数据部分
        for i in range(len(X)):
            features = ",".join(map(str, X[i]))
            f.write(f"{features},{y[i]}\n")

```

NKU

```
    print(f"转换完成:␣{output_file}")

if __name__ == "__main__":
    convert_to_arff("semeion.data", "semeion.arff")
```

2. 切换至 **Classify** 面板：

   (a) 选择 classifier: `lazy → IBk`。

   (b) 点击 IBk 行，设置参数 `KNN = 1`，然后点击 `OK`。

   (c) 在 `Test options` 中选择 **Leave-One-Out** （注意修改为“Cross-validation folds = 1593”）。

   (d) 点击 `Start`，记录 `Correctly Classified Instances` 的百分比。

3. 运行结果截图

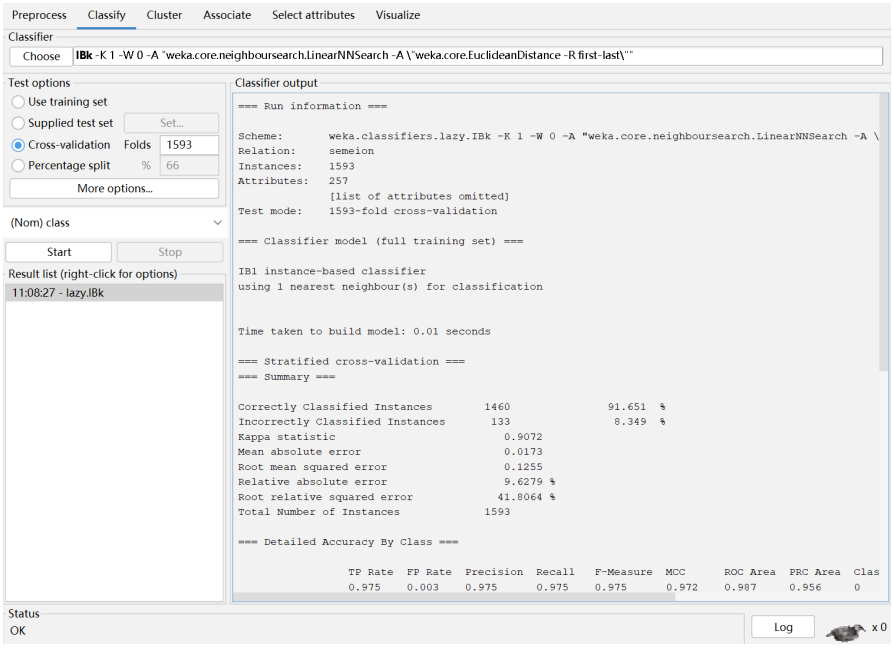

图 1: knn=1

```
Correctly Classified Instances        1437               90.2072 %
Incorrectly Classified Instances       156                9.7928 %
Kappa statistic                          0.8912
Mean absolute error                      0.0234
Root mean squared error                  0.1151
Relative absolute error                 12.9779 %
Root relative squared error             38.3499 %
Total Number of Instances             1593

=== Detailed Accuracy By Class ===

                 TP Rate   FP Rate   Precision   Recall   F-Measure   MCC      ROC
                 0.981     0.004     0.963       0.981    0.972       0.969    0.99
                 0.981     0.036     0.757       0.981    0.855       0.845    0.99
                 0.943     0.014     0.882       0.943    0.912       0.902    0.98
                 0.950     0.017     0.858       0.950    0.901       0.891    0.99
                 0.913     0.004     0.961       0.913    0.936       0.930    0.97
                 0.950     0.017     0.863       0.950    0.904       0.894    0.99
                 0.963     0.007     0.939       0.963    0.951       0.945    0.98
                 0.854     0.003     0.971       0.854    0.909       0.902    0.99
                 0.768     0.006     0.937       0.768    0.844       0.834    0.95
                 0.709     0.001     0.982       0.709    0.824       0.820    0.94
Weighted Avg.    0.902     0.011     0.911       0.902    0.901       0.894    0.98
```

图 2: knn=3

```
=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances        1445               90.7094 %
Incorrectly Classified Instances       148                9.2906 %
Kappa statistic                          0.8968
Mean absolute error                      0.0278
Root mean squared error                  0.1157
Relative absolute error                 15.4309 %
Root relative squared error             38.5536 %
Total Number of Instances             1593

=== Detailed Accuracy By Class ===

                 TP Rate   FP Rate   Precision   Recall   F-Measure   MCC      ROC
                 0.988     0.005     0.958       0.988    0.972       0.969    0.99
                 0.969     0.034     0.762       0.969    0.853       0.842    0.99
                 0.937     0.010     0.914       0.937    0.925       0.917    0.98
                 0.962     0.019     0.850       0.962    0.903       0.893    0.99
                 0.919     0.005     0.955       0.919    0.937       0.930    0.98
                 0.956     0.014     0.884       0.956    0.918       0.910    0.99
                 0.957     0.006     0.945       0.957    0.951       0.945    0.98
                 0.873     0.003     0.972       0.873    0.920       0.913    0.99
                 0.781     0.005     0.945       0.781    0.855       0.846    0.96
```

图 3: knn=5

3. 结果

表 1: 自写 KNN 与 Weka IBk 在 LOO 下的精度对比

k	自写 LOO 精度	Weka LOO 精度	差异（绝对值）
1	0.9190	0.91651	0.00249
3	0.9046	0.902072	0.00253
5	0.9046	0.907094	0.00249

二、 实验总结

本次实验通过对 `semeion` 手写数字数据集进行处理与分析，分别采用自编写的 KNN 算法与 Weka 工具中的 IBk 实现进行了对比实验。在基础任务中，自写 KNN 在留一交叉验证（LOO）下的精度与 Weka 结果高度一致，差异在 0.5% 以内，说明算法实现正确。

通过实验进一步熟悉了 KNN 算法的原理与实现过程，掌握了数据预处理、交叉验证以及分类器评估等关键步骤。同时，利用 Weka 与 sklearn 等工具，有助于对比不同实现方式的结果，从而加深对机器学习算法性能的理解。

总体而言，本实验不仅锻炼了编程实现与调试能力，也培养了利用工具进行验证与分析的习惯，为后续更复杂的模式识别与机器学习实验打下了良好基础。

NKU