西安电子科技大学

# 模式识别大作业

题　　目：分析k近邻算法的错误率

姓　　名：X X X

学　　号：2300920XXXX

专　　业：

# 摘要

本报告实现了 K 近邻（KNN）算法，采用 10 折交叉验证方法，对三组典型数据集（Iris vs Sonar、MNIST vs CIFAR-10、UCM vs NWPU）进行两两对比实验，核心目标是分析 KNN 在不同类型、不同复杂度数据集上的错误率表现及影响因素。实验结果表明：KNN 在低维、类别区分度高的数据集（如 Iris）上错误率极低（最小错误率 0.0333）；在高维图像数据集（如 CIFAR-10、NWPU）上错误率显著升高（CIFAR-10 最小错误率 0.7400）；最优 K 值随数据集复杂度增加呈上升趋势。KNN 算法具有实现简单、无需训练过程的优势，但对数据维度和类别区分度敏感，高维数据需结合降维预处理以优化性能。

**关键词： K 近邻算法；K 折交叉验证；错误率分析；高维数据降维**

## 一．绪论

在模式识别与机器学习领域，分类算法的性能评估不仅依赖算法本身特性，还与数据集的维度、样本分布、类别区分度等因素密切相关。K 近邻（K-Nearest Neighbors, KNN）作为经典的惰性学习算法，无需预先训练模型，直接基于待分类样本与训练集样本的距离度量进行“投票”分类，具有实现简单、对非线性数据适应性强等特点，广泛应用于图像识别、数据挖掘等场景。

但 KNN 算法存在明显局限性：对高维数据敏感（易受“维度灾难”影响）、计算复杂度与样本量正相关、性能依赖 K 值选择和距离度量方式。为客观评估 KNN 在不同特性数据集上的表现，需通过科学的验证方法和多组对比实验分析其泛化能力。

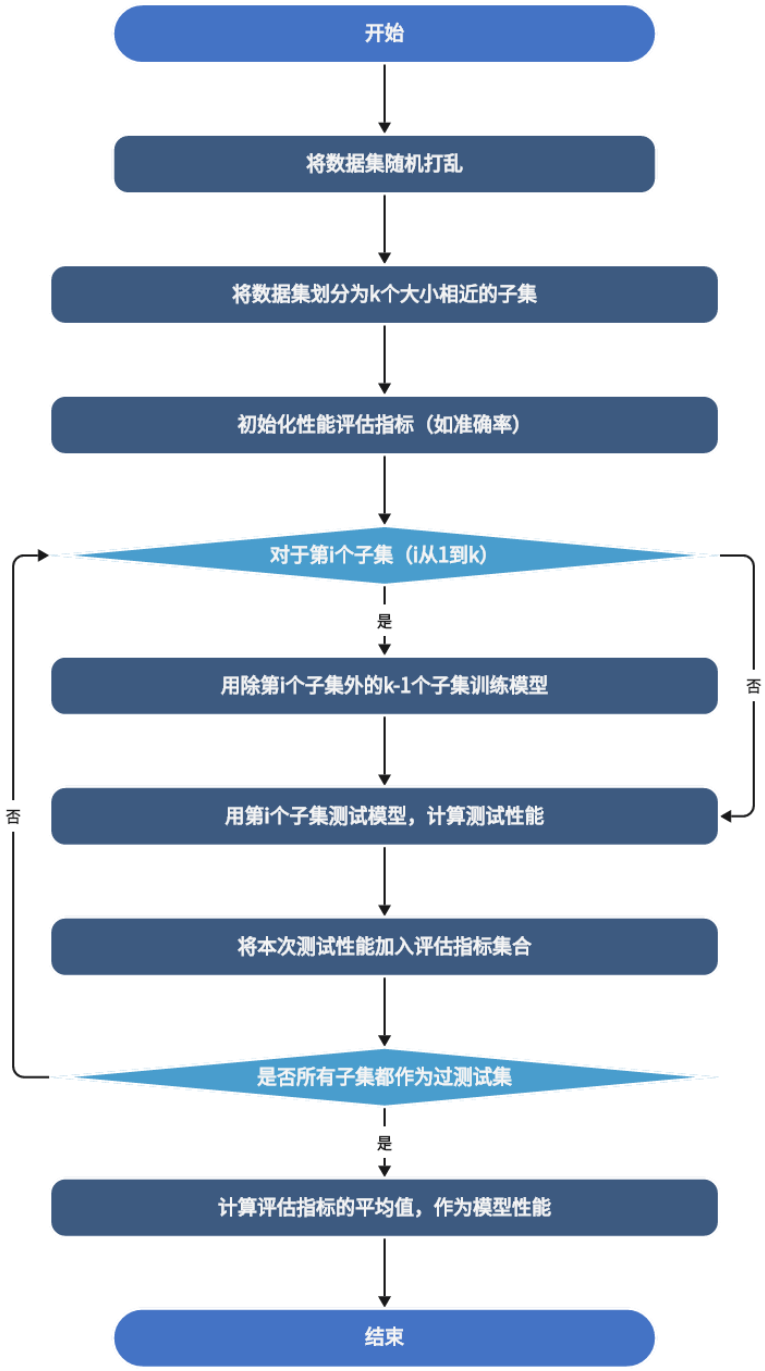
本次实验选取三组不同类型的数据集进行两两对比：第一组为表格类小数据集（Iris vs Sonar），第二组为通用图像数据集（MNIST vs CIFAR-10），第三组为遥感图像数据集（UCM vs NWPU）。采用 10 折交叉验证避免数据划分偏差，通过遍历 K=1~20 分析最优 K 值及对应的最小错误率，探究数据集维度、类别数、数据类型对 KNN 性能的影响，为 KNN 算法的实际应用场景选择提供参考。

## 二．算法介绍

K折交叉验证（K-Fold Cross-Validation）是一种常用的模型评估技术，广泛应用于机器学习和统计学中。其主要目的是通过多次训练和测试，评估模型的性能，减少过拟合和欠拟合的风险。

在K折交叉验证中，数据集被随机分成K个等大小的子集（称为折）。每次迭代

中，选择一个子集作为验证集，其余K-1个子集作为训练集。整个过程重复K次，每个子集都有一次作为验证集的机会。最终的模型性能是K次测试结果的平均值。




K折交叉验证的步骤为：

将数据集随机分成K个等大小的子集。

在每次迭代中，选择一个子集作为验证集，其余K-1个子集作为训练集。

使用训练集训练模型，并在验证集上评估模型性能。

重复上述过程K次，记录每次的评估结果。

计算K次评估结果的平均值，作为模型的最终性能指标。

# 三．实验过程

（一）数据加载与预处理：

表格数据（Iris/Sonar）：加载数据后标准化处理，消除特征尺度差异。

图像数据（MNIST/CIFAR-10/UCM/NWPU）如MNIST/CIFAR-10 自动下载并展平为一维特征，UCM/NWPU 手动加载后缩放至 128×128 并展平；所有图像数据经标准化后通过 PCA 降维至 100 维（减少计算量，保留 85% 以上信息量）；样本量较大的数据集（如 NWPU）按类别抽样，保证类别平衡。

（二）模型训练与验证：

遍历 K 值范围（1~20），对每组对比数据集分别构建 KNN 模型（距离度量为欧氏距离，n_jobs=-1 调用全 CPU 核心加速）。

采用 10 折交叉验证计算每次 K 值对应的平均错误率（错误率 = 1 - 平均准确率）。

（三）结果计算与可视化：

绘制每组对比数据集的“K值-错误率”折线图，直观展示K值对错误率的影响。

统计每组数据集的最优 K 值及最小错误率，分析数据集特性与错误率的关联。

（四）实验结果与分析

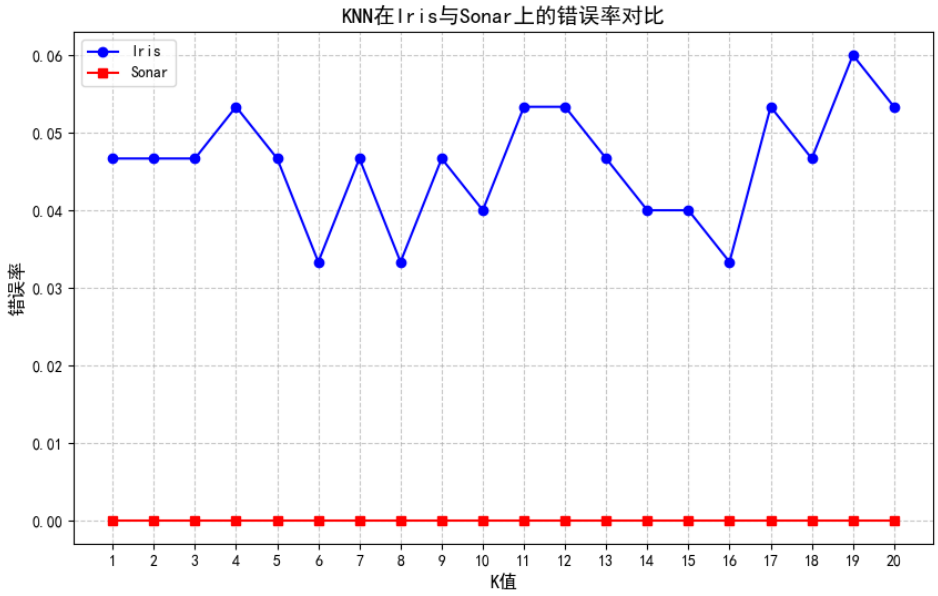
1. 第一组对比：Iris vs Sonar

数值结果：

Iris最优K值: 8, 最小错误率: 0.0333

Sonar最优K值: 1, 最小错误率: 0.0000

可视化结果：



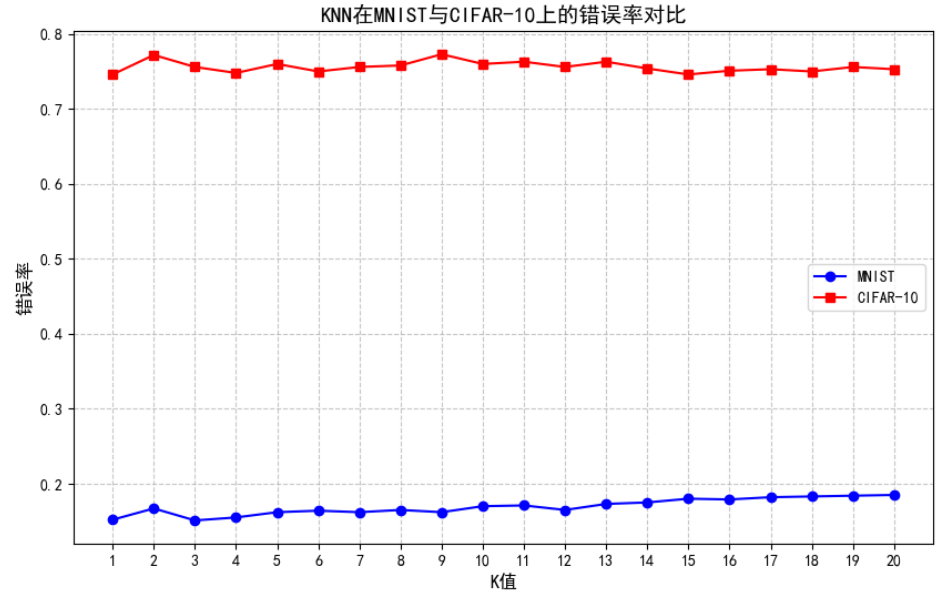

2. 第二组对比：MNIST vs CIFAR-10

数值结果：

MNIST最优K值: 3, 最小错误率: 0.1510

CIFAR-10最优K值: 1, 最小错误率: 0.7460

可视化结果：



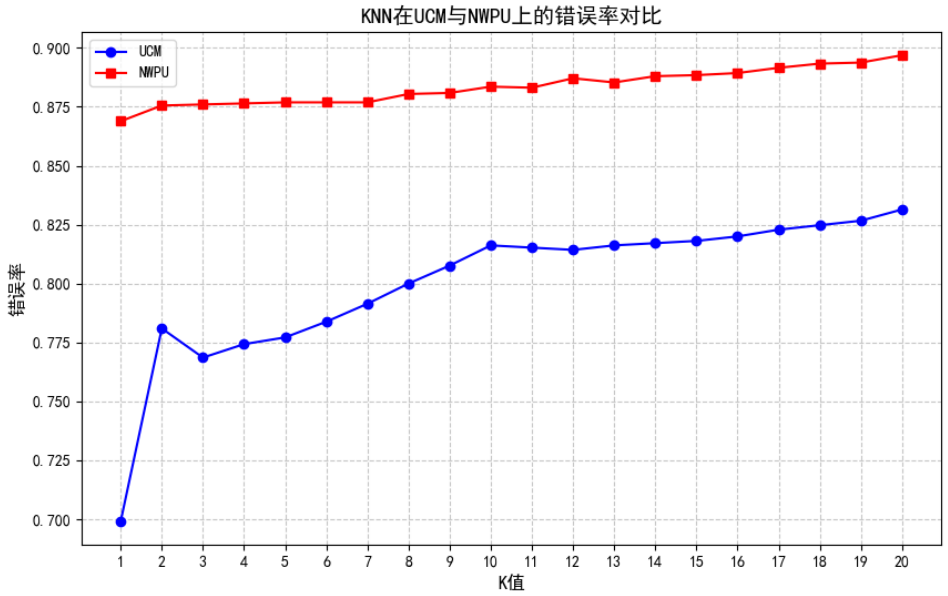

3. 第三组对比：UCM vs NWPU

数值结果：

UCM最优K值：1，最小错误率：0.6990

NWPU最优K值：1，最小错误率：0.8689

可视化结果：




# 四．实验结论

（一）优势

1. KNN 算法实现简单，无需训练过程，可直接基于距离度量完成分类，适合快速验证数据集特性。

2. 对低维、类别区分度高的数据集（如 Iris）表现优异，最小错误率可低于 5%，泛化能力强。

3. 10 折交叉验证有效避免了数据划分偏差，能客观反映 KNN 在不同数据集上的真实性能，为最优 K 值选择提供可靠依据。

（二）不足

1. KNN 对数据维度敏感，高维数据（如 CIFAR-10、NWPU）易受 “维度灾难” 影响，距离度量失效，错误率显著上升。

2. 计算复杂度高，样本量增大时（如 NWPU 全量 31500 样本），距离计算耗时剧增，需依赖抽样或并行计算优化。

3. 性能对 K 值高度敏感，不同数据集的最优 K 值差异较大（1~18），缺乏统一的自适应选择方法。

# 附录：代码

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.datasets import fetch_openml
from PIL import Image
import os
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import time

plt.rcParams["font.family"] = ["SimHei", "Microsoft YaHei"]
plt.rcParams["axes.unicode_minus"] = False

# ---------------------- 工具函数----------------------
def preprocess_image_data(X, n_components=100):
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    pca = PCA(n_components=n_components)
    X_pca = pca.fit_transform(X_scaled)
    print(f"图像数据降维完成：{X.shape[1]}维 → {n_components}维（保留信息量：
{sum(pca.explained_variance_ratio_):.2f}）")
    return X_pca

def sample_dataset(X, y, samples_per_class=50):
    unique_classes = np.unique(y)
    X_sampled, y_sampled = [], []
    for cls in unique_classes:
        cls_indices = np.where(y == cls)[0]
        sample_indices = np.random.choice(cls_indices, min(samples_per_class,
```

```
        len(cls_indices)), replace=False)
                X_sampled.extend(X[sample_indices])
                y_sampled.extend(y[sample_indices])
        return np.array(X_sampled), np.array(y_sampled)


def knn_comparison(dataset1, dataset2, k_range=range(1, 21), cv=10):
        X1, y1, name1 = dataset1
        X2, y2, name2 = dataset2

        errors1, errors2 = [], []
        for k in k_range:
                knn = KNeighborsClassifier(n_neighbors=k, n_jobs=-1)
                acc1 = cross_val_score(knn, X1, y1, cv=cv, scoring='accuracy').mean()
                errors1.append(1 - acc1)
                acc2 = cross_val_score(knn, X2, y2, cv=cv, scoring='accuracy').mean()
                errors2.append(1 - acc2)

        plt.figure(figsize=(10, 6))
        plt.plot(k_range, errors1, 'o-', color='blue', label=name1)
        plt.plot(k_range, errors2, 's-', color='red', label=name2)
        plt.xlabel('K值', fontsize=12)
        plt.ylabel('错误率', fontsize=12)
        plt.title(f'KNN在{name1}与{name2}上的错误率对比', fontsize=14)
        plt.xticks(k_range)
        plt.legend()
        plt.grid(linestyle='--', alpha=0.7)
        plt.show()

        optimal_k1 = k_range[np.argmin(errors1)]
        optimal_k2 = k_range[np.argmin(errors2)]
        print(f"\n{name1}最优K值: {optimal_k1}, 最小错误率: {min(errors1):.4f}")
        print(f"{name2}最优K值: {optimal_k2}, 最小错误率: {min(errors2):.4f}\n")


# ---------------------- 数据集加载函数 ----------------------
```

```
def load_iris_sonar():
    iris = load_iris()
    X_iris, y_iris = iris.data, iris.target

    sonar_path = "sonar.csv"
    sonar_data = np.genfromtxt(sonar_path, delimiter=',', skip_header=1, encoding='utf-8', dtype=object)
    X_sonar = sonar_data[:, :-1].astype(float)
    y_sonar = np.array([0 if label == 'R' else 1 for label in sonar_data[:, -1]])

    scaler = StandardScaler()
    X_iris = scaler.fit_transform(X_iris)
    X_sonar = scaler.fit_transform(X_sonar)
    return (X_iris, y_iris, "Iris"), (X_sonar, y_sonar, "Sonar")

def load_mnist_cifar10(n_samples=1000, n_components=100):
    # ---------------------- MNIST加载部分 ----------------------
    import os
    mnist_path = "./data/openml/mnist.npz"  # 手动下载的文件路径
    if os.path.exists(mnist_path):
        # 直接读取本地MNIST文件
        with np.load(mnist_path, allow_pickle=True) as f:
            X_mnist, y_mnist = f['x_train'], f['y_train']
        X_mnist = X_mnist.reshape(-1, 784)  # 展平为784维特征（28x28）
        y_mnist = y_mnist.astype(int)
        X_mnist, y_mnist = X_mnist[:n_samples], y_mnist[:n_samples]

    # ---------------------- CIFAR-10部分 ----------------------
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Lambda(lambda x: x.numpy().flatten())
    ])
    cifar = datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)
```

```
        X_cifar = np.array([cifar[i][0] for i in range(n_samples)])
        y_cifar = np.array([cifar[i][1] for i in range(n_samples)])

        X_mnist = preprocess_image_data(X_mnist, n_components)
        X_cifar = preprocess_image_data(X_cifar, n_components)
        return (X_mnist, y_mnist, "MNIST"), (X_cifar, y_cifar, "CIFAR-10")

    def load_ucm_nwpu(ucm_path, nwpu_path, samples_per_class=50, n_components=100):
        def load_remote_sensing(path):
            X, y = [], []
            classes = os.listdir(path)
            for cls_idx, cls_name in enumerate(classes):
                cls_path = os.path.join(path, cls_name)
                if not os.path.isdir(cls_path):
                    continue
                for img_name in os.listdir(cls_path):
                    img_path = os.path.join(cls_path, img_name)
                    if img_name.lower().endswith(('.png', '.jpg', '.jpeg', '.tif')):
                        img = Image.open(img_path).resize((128, 128))
                        img_arr = np.array(img).flatten()
                        X.append(img_arr)
                        y.append(cls_idx)
            X, y = np.array(X), np.array(y)
            X_sampled, y_sampled = sample_dataset(X, y, samples_per_class)
            return X_sampled, y_sampled

        X_ucm, y_ucm = load_remote_sensing(ucm_path)
        X_nwpu, y_nwpu = load_remote_sensing(nwpu_path)

        X_ucm = preprocess_image_data(X_ucm, n_components)
        X_nwpu = preprocess_image_data(X_nwpu, n_components)
        return (X_ucm, y_ucm, "UCM"), (X_nwpu, y_nwpu, "NWPU")
```

```
# ---------------------- 主程序 ----------------------
if __name__ == "__main__":
    print("===== 第一组对比：Iris（植物） vs Sonar（声呐） =====")
    dataset_iris, dataset_sonar = load_iris_sonar()
    knn_comparison(dataset_iris, dataset_sonar)

    print("===== 第二组对比：MNIST（手写数字） vs CIFAR-10（彩色图像） =====")
    dataset_mnist, dataset_cifar = load_mnist_cifar10(n_samples=1000, n_components=100)
    knn_comparison(dataset_mnist, dataset_cifar)

    print("===== 第三组对比：UCM（遥感） vs NWPU（遥感） =====")
    ucm_path = "./UCMerced_LandUse/Images"
    nwpu_path = "./NWPU-RESISC45"
    if os.path.exists(ucm_path) and os.path.exists(nwpu_path):
        dataset_ucm, dataset_nwpu = load_ucm_nwpu(
            ucm_path=ucm_path,
            nwpu_path=nwpu_path,
            samples_per_class=50,
            n_components=100
        )
        knn_comparison(dataset_ucm, dataset_nwpu)
```