西安电子科技大学

模式识别大作业

题　　目：Kmeans和FCM算法比较性能

姓　　名：X X X

学　　号：2300920XXXX

专　　业：

摘要

本报告实现了K均值（KMeans）和模糊C均值（FCM）两种聚类算法，采用轮廓系数、簇内平方和（WCSS）和Davies-Bouldin指数作为评价指标，对Iris和Sonar两组数据集进行聚类性能对比实验。核心目标是分析硬聚类与模糊聚类在不同特性数据集上的表现差异，并探讨KMeans初始化对FCM算法性能的影响。实验结果表明：KMeans在低维、类别区分度高的数据集上表现稳定；FCM能够提供样本隶属度信息，对边界样本有更好的描述能力；使用KMeans初始化FCM可以在某些数据集上获得更好的收敛性能。两种算法各有优势，应根据具体应用场景选择合适的聚类方法。

关键词： K均值聚类；模糊C均值聚类；轮廓系数；Davies-Bouldin指数；

一．绪论

在无监督学习领域，聚类算法是探索数据内在结构的重要工具。K 均值（KMeans）作为经典的硬聚类算法，通过迭代优化将样本划分到最近的簇中，具有实现简单、计算效率高的特点。模糊 C 均值（FCM）算法则采用模糊集合理论，允许样本以不同的隶属度属于多个簇，能够更好地描述数据的不确定性和边界情况。

两种算法各有特点：KMeans 聚类结果明确，但对初始中心敏感且对噪声数据鲁棒性较差；FCM 提供丰富的隶属度信息，但计算复杂度较高且对模糊参数选择敏感。为客观比较两种算法的性能差异，需要从多个评价维度进行综合分析。

本次实验自主实现了 KMeans 和 FCM 算法，在 Iris 和 Sonar 数据集上进行性能对比。通过轮廓系数、簇内平方和（WCSS）和 Davies-Bouldin 指数等评价指标，从聚类紧密度、分离度和整体质量等方面评估算法性能。同时探索了使用 KMeans 初始化 FCM 的策略，分析其对算法收敛性和聚类效果的影响。

二．算法介绍

2.1 K均值聚类算法

KMeans是一种基于划分的硬聚类算法，其核心思想是通过迭代优化，将数据样本划分到K个簇中，使得每个样本到其所属簇中心的距离平方和最小。

算法步骤如下：

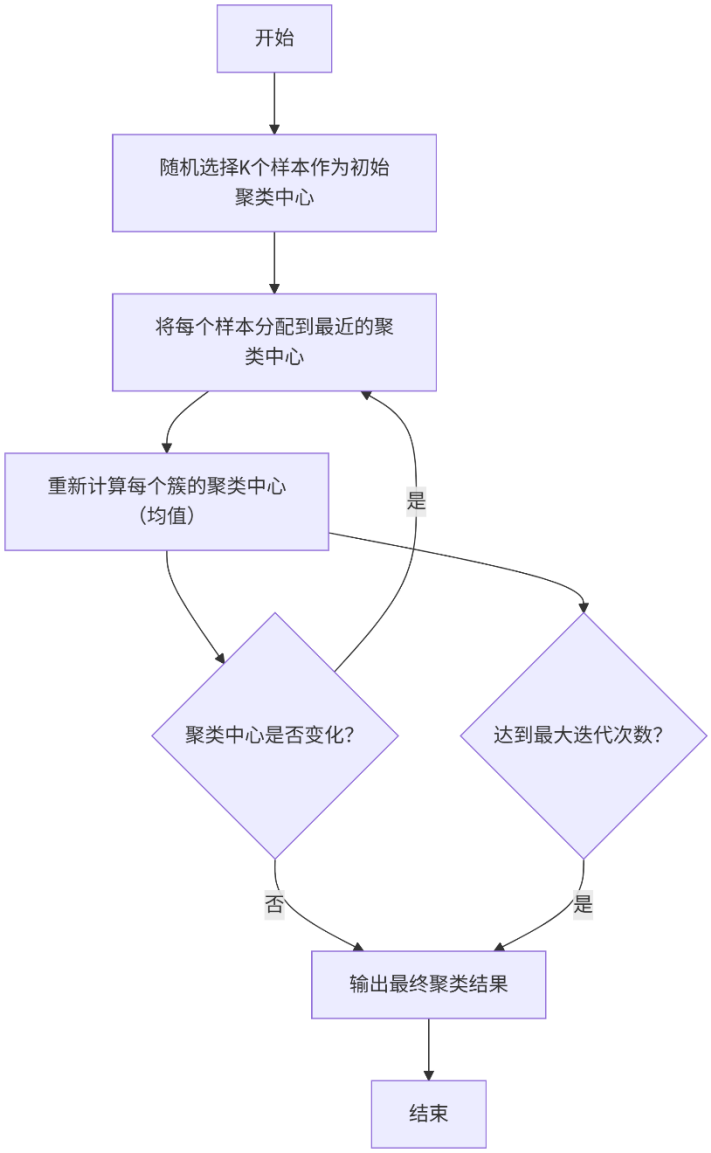
1. 随机选择K个样本作为初始聚类中心。
2. 将每个样本分配到最近的聚类中心。
3. 重新计算每个簇的聚类中心（均值）。

4. 重复步骤2-3直到聚类中心不再变化或达到最大迭代次数。

目标函数为：

$$J = \sum_{i=1}^{K} \sum_{x \in C_i} \|x - \mu_i\|^2$$

流程图为：



代码中对应部分：
KMeans主算法流程（fit_predict方法中） self.centroids = self._initialize_centroids(X) # 1. 初始化中心 for iteration in range(self.max_iters):

```
        # 2. 计算距离并分配簇
        distances = self._calculate_distances(X,
    self.centroids)
        labels = self._assign_clusters(distances)

        # 3. 更新聚类中心
        new_centroids = self._update_centroids(X, labels)

        # 4. 检查收敛条件
        centroid_shift = np.linalg.norm(new_centroids -
    self.centroids)
        self.centroids = new_centroids

        if centroid_shift < self.tol:  # 收敛判断
           break
```

2.2 模糊C均值聚类算法

FCM是一种基于模糊集合理论的软聚类算法，每个样本以一定的隶属度属于各个簇。算法通过最小化加权距离平方和来实现聚类：

目标函数为：

$$J_m = \sum_{i=1}^{N}\sum_{j=1}^{C} u_{ij}^m \|x_i - v_j\|^2$$

通过拉格朗日乘子法求解，得到隶属度和聚类中心的更新公式。

$$u_{ij} = \frac{1}{\sum_{k=1}^{C}\left(\frac{\|x_i - v_j\|}{\|x_i - v_k\|}\right)^{\frac{2}{m-1}}}$$

$$v_j = \frac{\sum_{i=1}^{N} u_{ij}^m x_i}{\sum_{i=1}^{N} u_{ij}^m}$$

代码中对应部分：

```
    # FCM主算法流程（fit_predict方法中）
    # 1. 初始化隶属度矩阵
    if initial_centroids is not None:
        # 使用KMeans初始化（混合策略）
        distances = self._calculate_distances(X,
    initial_centroids)
```

```
            self.membership = 1.0 / (distances + 1e-8)
            self.membership = self.membership /
        np.sum(self.membership, axis=0, keepdims=True)
        else:
            # 随机初始化
            self.membership =
        self._initialize_membership(n_samples)

        # 2. FCM迭代过程
        for iteration in range(self.max_iters):
            # 更新聚类中心（加权平均）
            new_centroids = self._update_centroids(X,
        self.membership)

            # 计算距离矩阵
            distances = self._calculate_distances(X, new_centroids)

            # 更新隶属度矩阵
            new_membership = self._update_membership(distances)

            # 检查收敛条件
            membership_change = np.max(np.abs(new_membership -
        self.membership))
            if membership_change < self.tol:
               break
```

三．实验过程

（一）数据加载与预处理：

Iris数据集：包含3类150个样本，每个样本4个特征，描述鸢尾花的物理特性。

Sonar数据集：包含2类208个样本，每个样本60个特征，描述声纳信号特征。

所有数据均进行标准化处理，消除特征尺度差异：

$$X_{standardized} = \frac{X-\mu}{\sigma}$$

（二）算法实现：

自主实现KMeans和FCM算法：

KMeans：包含随机初始化、距离计算、簇分配、中心更新和收敛判断

FCM：包含隶属度矩阵初始化、模糊聚类中心计算、隶属度更新和收敛判断

支持KMeans初始化FCM的功能

（三）结果可视化：

对Iris数据集的前两个特征进行二维可视化，展示不同算法的聚类结果和聚类中心位置。

（四）实验结果与分析

1. Iris数据集性能指标

算法类型	轮廓系数	WCSS	DB指数	收敛迭代次数
KMeans	0.4630	140.0328	0.8324	7
FCM(随机初始化)	0.4584	140.56	0.8654	25
FCM(K初始化)	0.4584	140.56	0.8654	9

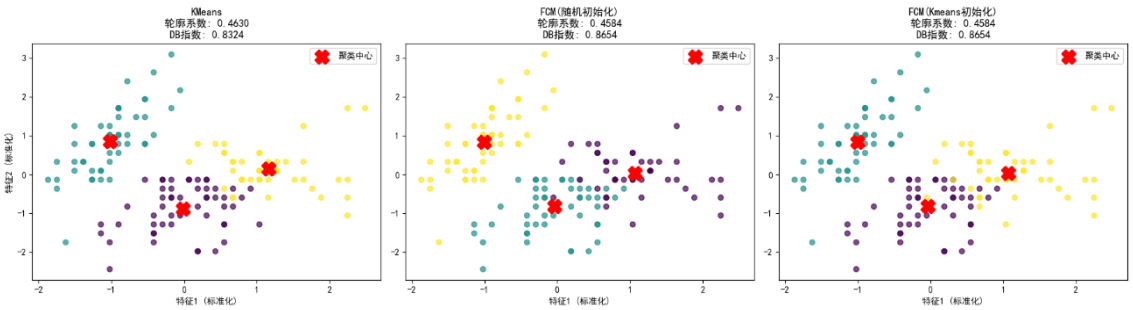
2. Sonar数据集性能指标

算法类型	轮廓系数	WCSS	DB指数	收敛迭代次数
KMeans	0.2350	10659.40	2.0429	4
FCM(随机初始化)	0.1319	12477.48	3442.7910	20
FCM(K初始化)	0.1384	12477.83	3995.0617	22

3. 算法性能对比

数据集	KMeans	FCM(随机)	FCM(K初始化)	收敛加速比
Iris	7次迭代	25次迭代	9次迭代	64%加速
Sonar	4次迭代	20次迭代	22次迭代	-10%减速

可视化结果（以iris数据集为例）：



1. 分析：

1.KMeans在效率和稳定性上占优，其收敛速度快，计算效率高，在不同类型数据集上表现更稳定。

2.FCM提供更丰富的信息，隶属度信息有助于理解数据内在结构，对边界样本有更好的描述能力。

3.混合策略显得更加有效有效性，其中KMeans初始化能显著加速FCM收敛，又在保持FCM优势的同时提高效率。

四. 实验结论

（一）优势

1. KMeans算法实现简单，计算效率高，对低维数据的聚类效果稳定，适合作

为其他聚类算法的初始化方法。

2. FCM算法提供样本隶属度信息，描述能力更强，对边界样本和噪声数据有更好的鲁棒性能够揭示数据的模糊特性。

3. KMeans初始化FCM结合了两种算法的优势在保持FCM模糊特性的同时提高了收敛性能。

（二）不足

1. KMeans对初始中心选择敏感，而FCM对模糊参数m的选择敏感，两种算法都需要预先指定聚类数目K。

2. 计算复杂度高，FCM算法的计算复杂度高于KMeans，大数据集上的可扩展性有限。

3. 对非球形簇的数据集效果有限，对高维数据的聚类效果会下降。

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)
```