

湖南城市学院

实验名称 实验一 python 基础练习

姓 名 李灵慧

学 号 2202501-18

学 院 市政与测绘工程学院

专 业 地理空间信息工程

指导教师 汤淼

2025 年 3 月 14 日

1 生成问题

1.1 矩阵的初步运用

实现矩阵的创建、变形、索引、基本运算，掌握广播机制。

关于python中numpy矩阵的初步运用生成问题

已深度思考 (用时 25 秒)

以下是一些针对NumPy矩阵初步运用的练习题，涵盖基础操作和常见应用场景：

基础操作篇

1. 矩阵创建

用NumPy实现以下操作：

创建一个3×3的全零浮点矩阵

创建一个从10到49的5×8整数矩阵（包含49）

生成5×5的单位矩阵

python

```
# 你的代码
import numpy as np
zero_matrix = ..... # 你的代码
```

复制

2. 矩阵变形

给定一维数组 `arr = np.arange(12)`，将其转换为：

3×4的二维矩阵

2×3×2的三维张量

保持元素总数不变的6×2矩阵（自动计算缺失维度）

3. 索引技巧

对于矩阵 `m = np.arange(25).reshape(5,5)`：

取出第2、4行的第1、3列

获取所有大于20的元素

将右下角2×2子矩阵替换为 `[[99,99],[99,99]]`

运算操作篇

4. 矩阵运算

给定两个矩阵：

python

```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
```

复制

计算：

元素级乘法（对应位置相乘）

矩阵乘法（点积）

计算a矩阵的迹（trace）

计算a矩阵的L2范数（Frobenius范数）

5. 广播机制

实现以下操作：

将3×4矩阵的每一行都加上[1,2,3,4]

将4×3矩阵的每一列都乘以[1,2,3,4]^T

计算5×3矩阵与3×1向量的欧式距离（无需循环）

图 1.1 矩阵的初步运用

1.2 图像滤波处理

实现均值滤波、中值滤波两个平滑滤波，以及 Roberts 和 Sobel 两个边缘增强滤波。

对于图像滤波处理，举几个常用滤波

已深度思考 (用时 24 秒)

在图像处理中，滤波是常用的去噪、平滑或增强特征的方法。以下是几种常见的图像滤波器及其特点：

1. 均值滤波（Mean Filter）

原理：用邻域内像素的平均值替换中心像素值。

作用：去除高斯噪声或均匀分布的噪声，但会导致图像模糊。

适用场景：简单快速去噪，但对边缘保护较差。

示例：3×3 或 5×5 的滑动窗口。

2. 高斯滤波（Gaussian Filter）

原理：对邻域像素进行加权平均，权重由高斯函数（钟形曲线）决定，距离中心越近权重越大。

作用：有效抑制高斯噪声，保留较多边缘信息。

适用场景：平滑图像同时减少边缘模糊（如人脸去噪）。

关键参数：高斯核大小和方差（ σ ），方差越大越模糊。

3. 中值滤波（Median Filter）

原理：用邻域内像素的中值替换中心像素值。

作用：高效去除椒盐噪声（随机黑白点）且保护边缘。

适用场景：处理脉冲噪声，适合医学图像或老旧照片修复。

特点：非线性滤波，不依赖于极值点。

交叉算子边缘检测

在边缘检测中，交叉算子（Cross Operator）通常指 Roberts 交叉算子（Roberts Cross Operator），它是早期的边缘检测算子之一。通过计算图像相邻方向的差分来检测边缘。以下是其原理、实现及与其他算子的对比：

1. Roberts 交叉算子（Roberts Cross）

原理

核心思想：利用对角线像素的差分来检测边缘，适合检测 45° 和 135° 方向的边缘。

模板（卷积核）：

水平模板（Gx）：

垂直模板（Gy）：

强度计算：

$G = \sqrt{G_x^2 + G_y^2}$ 或简化为 $G = |G_x| + |G_y|$

特点

优点：

计算简单，适合快速实现。

对噪声敏感，适合检测强边缘。

缺点：

对噪声敏感（2x2 小模板的弱性能），容易漏掉弱边缘。

适用场景

检测较小的图像边缘快速检测。

需要检测对角线方向边缘的场景。

2. 其他常用边缘检测算子

(1) Sobel 算子

模板（3x3 核）：

水平模板（Gx）：

垂直模板（Gy）：

特点：

图 1.2 图像滤波处理

1.3 遥感图像分类

1

实现 K-means 算法和肘部图的绘制。

已读深思 (用时 29 秒)

遥感图像的非监督分类不需要先验样本，直接通过数据本身的特征进行分类划分。以下是常见的非监督分类算法及其特点：

1. K-means 聚类

- 原理：通过迭代将数据划分为 K 个簇，最小化样本到簇中心的距离平方和。
- 遥感应用：适用于光谱特征明显分离的场景（如水体、植被、裸土等）。
- 缺点：需预设类别数 K ，对初始中心敏感，无法处理非球形簇。

关于遥感图像非监督分类有几种算法

K-means 聚类确定最佳K值的肘法

已读深思 (用时 22 秒)

确定 K-means 聚类的最佳 K 值是聚类分析的关键步骤。以下是常用的 自动或半自动选择最佳 K 值 的算法和方法，结合数学原理与应用场景进行说明：

1. 肘部法 (Elbow Method)

- 原理：通过计算不同 K 值下的 总类内平方和 (SSE, Sum of Squared Errors)，找到 SSE 下降速度突然变缓的“拐点”（形如肘部），对应的 K 即为最佳值。
- 公式：
$$SSE = \sum_{k=1}^K \sum_{x \in C_k} \|x - \mu_k\|^2$$
其中 C_k 是第 k 个簇， μ_k 是簇中心。
- 步骤：
 - 计算 K 从 1 到最大候选值时的 SSE。
 - 绘制 K-SSE 曲线，选择拐点对应的 K 。
- 优点：简单直观。
- 缺点：拐点可能不明显（尤其是高维数据），依赖主观判断。

图 1.3 遥感图像分类

2 解题逻辑

2.1 矩阵的初步运用

创建有 `numpy.array()`、`numpy.zeros()`、`numpy.ones()`、`numpy.eye()`函数

索引有行列号索引、单列索引、单行索引、布尔索引。

变形有 `numpy.resize()`、`array.reshape()`、`array.flatten()`、`array.ravel()`函数

运算 `numpy.multiply()`、`numpy.dot()`、`numpy.trace()`、`numpy.linalg.norm()`等函数

广播机制是将较小数组自动扩张的机制，便于数组计算

2.2 图像滤波处理

首先，知道如何读取图像和图像读取后的数据结构

然后，根据数据结构，进行卷积运算，得到结果。

最后，知道如何显示处理前后的图像。

2.3 遥感图像分类

根据图像滤波处理得到的经验读取和输出图像

K-means 分类原理即：

1. 选取初始点：在多维空间中随机选取一个中心点，然后进行循环，选取距离已选的 i 个中心点最远的中心点，直到选完 K 个中心点。
2. 标记数据：每个像素点找到最近的中心点，并打上中心点的类别标记。
3. 计算新点：算各个类别在多维空间的质心，并将质心作为新中心点。

4. 循环选取：重复 2、3 步直到循环次数超出限定次数，或者旧中心点和新中心点的距离小于限差。

肘部图即计算不同 K 值下的 总类内平方和，即计算所有数据点到对应聚类中心的平均距离，然后除以样本数。

3 代码

3.1 矩阵的初步运用

```
import numpy as np

# ===== 矩阵创建 =====
# 从列表创建
A = np.array([[1,2], [3,4]]) # 2x2矩阵

# 特殊矩阵
zeros = np.zeros((3,2)) # 3x2零矩阵
ones = np.ones((2,3)) # 2x3单位矩阵
identity = np.eye(3) # 3x3单位矩阵
random_mat = np.random.rand(2,2) # 2x2随机矩阵

# ===== 矩阵索引 =====
print(A[0,1]) # 输出 2 (第0行第1列)
print(A[:,1]) # 输出 [2,4] (第1列)
print(A[1,:]) # 输出 [3,4] (第1行)
print(A[A > 2]) # 输出 [3,4] (布尔索引)

# ===== 矩阵变形 =====
B = A.reshape(4,1) # 变形为4x1矩阵 (不修改原数据)
C = np.resize(A, new_shape=(3,3)) # 自动填充重复元素

# 展平操作
flattened = A.flatten() # [1,2,3,4] (返回拷贝)
raveled = A.ravel() # 原数据视图

# ===== 元素级乘法 =====
D = A * A # 对应元素相乘
E = np.multiply(A, A)

# ===== 矩阵点积 =====
F = np.dot(A, A) # 矩阵乘法
G = A @ A # Python 3.5+ 运算符

# ===== 矩阵的迹 =====
trace = np.trace(A) # 1+4=5

# ===== L2范数 =====
l2_norm = np.linalg.norm(A) # sqrt(1+4+9+16)=5.477

# ===== 广播机制 =====
vec = np.array([10, 20])
H = A + vec # 自动广播为:
# [[1+10, 2+20],
# [3+10, 4+20]]
```

图 3.1 矩阵的初步运用（代码）

3.2 图像滤波处理

```
import numpy as np # 导入 numpy 模块

def normalize(image, new_min=0, new_max=1, ifInt=False):...

# 一级滤波模式字典，包含基本统计量计算方式
FilterMode1={
    'mean': lambda image: np.mean(image), # 计算局部均值
    'median': lambda image: np.median(image) # 计算局部中值
}

# 二级滤波模式字典，包含需要特殊处理的边缘检测算子
FilterMode2={
    "sobel": lambda image,stretch: sobel_filter(image,stretch), # Sobel算子边缘检测
    "roberts": lambda image,stretch: roberts_filter(image,stretch) # Roberts算子边缘检测
}
```

图 3.2.1 图像滤波处理（代码）

```

17 def f_mode(image, mode='mean', size=3, stretch=1): 3 用法
18 """
19 通用滤波处理函数，支持多种滤波模式
20
21 参数:
22     image (ndarray): 输入的单通道灰度图像
23     mode (str): 滤波模式，可选 'mean', 'median', 'sobel', 'roberts'
24     size (int): 滤波窗口尺寸 (仅一维滤波模式有效)
25     stretch (float): 结果拉伸系数 (仅二维滤波模式有效)
26
27 返回:
28     ndarray/str: 滤波后的图像数组，或错误提示字符串
29
30 实现说明:
31     - 一维滤波使用滑动窗口进行局部统计计算
32     - 二维滤波调用专用边缘检测算子处理
33 """
34
35 if mode in FilterMode1:
36     # 一维滤波处理流程
37     Func = FilterMode1[mode]
38     pad_width = size // 2
39     padded_image = np.pad(image, pad_width, mode='symmetric') # 对称填充边界
40     filtered_image = np.zeros_like(image)
41
42     # 滑动窗口遍历每个像素
43     for i in range(image.shape[0]):
44         for j in range(image.shape[1]):
45             localImg = padded_image[i:i+size, j:j+size]
46             filtered_image[i, j] = Func(localImg)
47     return filtered_image
48 elif mode in FilterMode2:
49     # 二维滤波直接调用对应算子
50     Func = FilterMode2[mode]
51     return Func(image, stretch)
52 else:
53     return "不支持的计算模式"
54
55 76 def roberts_filter(image, stretch=1): 1 用法
56 """
57 Roberts交叉算子边缘检测
58
59 参数:
60     image (ndarray): 输入的单通道灰度图像
61     stretch (float): 结果增强系数
62
63 返回:
64     ndarray: 边缘检测结果图像
65 """
66 filterX = np.array([[ -1, 0], [0, 1]]) # Roberts水平模板
67 filterY = np.array([[0, -1], [1, 0]]) # Roberts垂直模板
68 return custom_filter2(image, filterX, filterY, mode='absSum', stretch=stretch)
69
70 92 def sobel_filter(image, stretch=1): 2 用法
71 """
72 Sobel算子边缘检测
73
74 参数:
75     image (ndarray): 输入的单通道灰度图像
76     stretch (float): 结果增强系数
77
78 返回:
79     ndarray: 边缘检测结果图像
80 """
81 filterX = np.array([[ -1, 0, 1], [-2, 0, 2], [-1, 0, 1]]) # Sobel水平模板
82 filterY = np.array([[ -1, -2, -1], [0, 0, 0], [1, 2, 1]]) # Sobel垂直模板
83 return custom_filter2(image, filterX, filterY, mode='absSum', stretch=stretch)
84
85 107

```

图 3.2.2 图像滤波处理（代码）

```

108 def custom_filter(image, matrix, stretch=1):
109     """
110     单矩阵自定义卷积滤波器
111
112     参数:
113         image (ndarray): 输入的单通道灰度图像
114         matrix (ndarray): 二维卷积核矩阵
115         stretch (float): 结果增强系数
116
117     返回:
118         ndarray/str: 滤波结果数组或错误提示
119
120     实现说明:
121         - 自动计算所需填充宽度
122         - 使用对称填充处理边界
123     """
124     if matrix.shape[0] != matrix.shape[1]: # 检查矩阵是否为正方形
125         return "滤波器矩阵必须为正方形"
126     if matrix.shape[0] != 2:
127         pad_width = matrix.shape[0] // 2
128     else:
129         pad_width = ((0, 1), (0, 1)) # 2x2矩阵特殊填充处理
130     padded_image = np.pad(image, pad_width, mode='symmetric')
131     filtered_image = np.zeros_like(image)
132     for i in range(image.shape[0]):
133         for j in range(image.shape[1]):
134             localImg = padded_image[i:i+matrix.shape[0], j:j+matrix.shape[1]]
135             filtered_image[i, j] = np.sum(localImg * matrix) * stretch # 卷积
136     return filtered_image
137
138 # 双矩阵滤波结果合并方式字典
139 CustomF2Mode = {
140     'mean': lambda MA, MB: (MA+MB)/2, # 均值合并
141     'max': lambda MA, MB: np.max([MA, MB]), # 取最大值
142     'min': lambda MA, MB: np.min([MA, MB]), # 取最小值
143     'sum': lambda MA, MB: MA+MB, # 直接求和
144     'avg': lambda MA, MB: (MA+MB)/2, # 平均值 (同mean)
145     'absSum': lambda MA, MB: np.sqrt(MA**2+MB**2) # 平方和开根号
146 }
147
148
149 def custom_filter2(image, matrixA, matrixB, mode='mean', stretch=1): 2 用法
150     """
151     双矩阵自定义卷积滤波器
152
153     参数:
154         image (ndarray): 输入的单通道灰度图像
155         matrixA (ndarray): 第一个二维卷积核
156         matrixB (ndarray): 第二个二维卷积核
157         mode (str): 双通道结果合并方式
158         stretch (float): 结果增强系数
159
160     返回:
161         ndarray/str: 滤波结果数组或错误提示
162
163     实现说明:
164         - 要求两个矩阵为相同尺寸的正方形
165         - 支持多种双通道计算结果合并方式
166     """
167     if matrixA.shape[0] != matrixA.shape[1] or matrixB.shape[0] != matrixB.shape[1]:
168         return "滤波器矩阵必须为正方形"
169     if matrixA.shape[0] != matrixB.shape[0]:
170         return "两个矩阵大小要相等"
171     if matrixA.shape[0] != 2:
172         pad_width = matrixA.shape[0] // 2
173     else:
174         pad_width = ((0, 1), (0, 1)) # 2x2矩阵特殊填充处理
175     padded_image = np.pad(image, pad_width, mode='symmetric')
176     filtered_image = np.zeros_like(image)
177     step = matrixA.shape[0]
178     if mode in CustomF2Mode:
179         Func = CustomF2Mode[mode]
180     else:
181         return "不支持的计算模式"
182
183 # 双矩阵卷积计算循环
184 for i in range(image.shape[0]):
185     for j in range(image.shape[1]):
186         localImg = padded_image[i:i+step, j:j+step]
187         MA = np.sum(localImg * matrixA) # 矩阵A的卷积结果
188         MB = np.sum(localImg * matrixB) # 矩阵B的卷积结果
189         filtered_image[i, j] = Func(MA, MB) * stretch # 合并计算结果并增强
190
191 return filtered_image

```

图 3.2.3 图像滤波处理（代码）

```

1 import Filter
2 import rasterio
3 import matplotlib.pyplot as plt
4 import os # 导入 os 模块
5
6 # 配置 matplotlib 支持中文
7 plt.rcParams['font.sans-serif'] = ['SimHei'] # 使用黑体或其他支持中文的字体
8 plt.rcParams['axes.unicode_minus'] = False # 解负号显示问题
9
10 # 获取当前项目路径
11 project_path = os.path.dirname(os.path.abspath(__file__))
12
13 # 定义文件路径变量
14 file_path = os.path.join(project_path, 'RES', 'test-data') # 使用 os.path.join 构建路径
15
16 # 打开文件并读取数据
17 with rasterio.open(file_path) as src: # 指定 nodata 值
18     band1 = src.read(1) # 读取第1波段
19     band5 = src.read(5)
20
21
22 # 对 band1 和 band5 进行中值滤波
23 band1_mean = Filter.f_mode(band1, mode='mean', size=3) # size 可以根据需要调整
24 band1_roberts = Filter.f_mode(band1_mean, mode='roberts', stretch=50)
25 band5_median = Filter.f_mode(band5, mode='median', size=3) # size 可以根据需要调整
26 band5_sobel = Filter.sobel_filter(band5_median)
27
28
29 fig, axes = plt.subplots(nrows=2, ncols=3) # 创建2行3列布局
30 axes[0,0].imshow(Filter.normalize(band5), cmap='gray') # 在第一个子图显示
31 axes[0,0].set_title('原始图像')
32 axes[0,1].imshow(Filter.normalize(band5_median), cmap='gray') # 在第二个子图显示
33 axes[0,1].set_title('中值滤波')
34 axes[0,2].imshow(Filter.normalize(band5_sobel), cmap='gray')
35 axes[0,2].set_title('Sobel滤波')
36 axes[1,0].imshow(Filter.normalize(band1), cmap='gray')
37 axes[1,0].set_title('原始图像')
38 axes[1,1].imshow(Filter.normalize(band1_mean), cmap='gray')
39 axes[1,1].set_title('均值滤波')
40 axes[1,2].imshow(Filter.normalize(band1_roberts), cmap='gray')
41 axes[1,2].set_title('Roberts滤波')
42 plt.tight_layout()
43 plt.show()

```

图 3.2.4 图像读取和输出（代码）

2. 使用 Rasterio

那么如何显示遥感图像

Rasterio是基于GDAL的简化库，接口更友好。

安装

```

bash
conda install -c conda-forge rasterio # 推荐
# 或
pip install rasterio

```

代码示例

```

python
import rasterio

# 使用上下文管理器自动处理资源
with rasterio.open('path/to/file.tif') as src:
    # 读取所有波段（形状为（波段数，行，列））
    data = src.read()
    # 读取单个波段（如第1波段）
    band1 = src.read(1)
    # 获取元数据
    transform = src.transform # 地理变换参数
    crs = src.crs # 坐标参考系统
    bounds = src.bounds # 地理范围

# 数据自动关闭，无需手动释放

```

优势

- 自动资源管理（通过 with 语句）。
- 支持分块读取大文件：src.read(window=...)。

您已浏览思考（用时 22 秒）

在 Python 中显示遥感图像，通常需要将栅格数据转换为可视化的图像格式（如灰度图、伪彩色图或 RGB 合成图）。以下是几种常见的方法和库的详细说明：

1. 使用 Matplotlib 显示单波段灰度图

适用于快速查看单个波段（如 DEM、单光谱波段）的分布。

代码示例（配合 Rasterio）：

```

python
import rasterio
import matplotlib.pyplot as plt

# 打开文件并读取数据
with rasterio.open('path/to/image.tif') as src:
    band = src.read(1) # 读取第1波段

# 显示图像
plt.figure(figsize=(10, 8))
plt.imshow(band, cmap='gray') # 使用灰度色带
plt.colorbar(label='像素值')
plt.title('单波段灰度图')
plt.xlabel('X 坐标')
plt.ylabel('Y 坐标')
plt.show()

```

参数调整：

- cmap: 色带（如 'viridis'、'jet'、'terrain' 等）。
- vmin/vmax: 设置显示范围（例如 plt.imshow(band, cmap='gray', vmin=0, vmax=1000)）。

图 3.2.5 图像读取和输出（AI 问答）

3.3 遥感图像分类

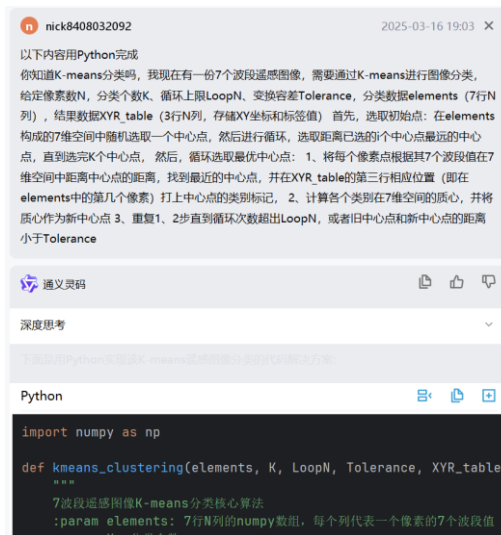


图 3.3.1 K-Means 分类（AI 问答）

```
1 import numpy as np # 导入 numpy 模块
2
3 def kmeans_clustering(elements, K=2, LoopN=100, Tolerance=1e-4, width=0, height=0): 6 两法
4     """
5     多波段遥感图像K-means分类核心算法
6     :param height:
7     :param width:
8     :param elements: N行M列的numpy数组，每个列代表一个像素的M个波段值
9     :param K: 分类个数
10    :param LoopN: 最大迭代次数
11    :param Tolerance: 中心点变化容差
12    """
13
14    assert elements.ndim == 2, "输入的elements必须是二维数组"
15    assert width * height == 0 or elements.shape[1] == width * height, "输入的elements与输入的高度宽度不匹配"
16
17    # 创建结果数组 (10行 x (height*width)列)
18    XYR_table = []
19    if width * height != 0:
20        XYR_table = np.zeros((3, height * width))
21        XYR_table[0] = np.tile(np.arange(width), height) # 列号按行重复
22        XYR_table[1] = np.repeat(np.arange(height), width) # 行号逐行展开
23
24    # 初始化中心点 (K-means++优化版)
25    centers = initialize_centers(elements, K)
26    labels = np.zeros(height * width)
27
28    # 主循环
29    for i in range(LoopN):
30        print("第" + str(i) + "次K-means循环")
31        # 步骤1: 分配标签
32        labels = assign_labels(elements, centers)
33        if width * height != 0:
34            XYR_table[2] = labels # 更新结果表的第三行
35
36        # 步骤2: 计算新质心
37        new_centers = calculate_centroids(elements, labels, K)
38
39        # 步骤3: 检查终止条件
40        if np.max(np.linalg.norm(new_centers - centers, axis=1)) < Tolerance:
41            break
42        centers = new_centers
43
44    print("分类完成")
45    # 将结果转换为二维数组
46    resultImg = []
47    if width * height != 0:
48        resultImg = convert_to_2d(XYR_table)
49        resultImg = np.flipud(resultImg)
50        resultImg = [list(row) for row in zip(*resultImg[::-1])]
51    return resultImg, centers, labels
```

图 3.3.2 K-Means 分类（代码）

```

53  ~ def initialize_centers(elements, K): 1个用法
54      """K-means++初始化中心点"""
55      N = elements.shape[1]
56      centers = np.zeros((K, elements.shape[0]))
57
58      # 随机选择第一个中心点
59      first_idx = np.random.randint(N)
60      centers[0] = elements[:, first_idx]
61
62      # 依次选择后续中心点
63  ~ for i in range(1, K):
64      # 计算每个点到最近现有中心的距离
65      distances = np.min(np.linalg.norm(
66          elements.T[:, np.newaxis, :] - centers[:i],
67          axis=2
68      ), axis=1)
69      # 选择距离最远的点作为新中心
70      new_center_idx = np.argmax(distances)
71      centers[i] = elements[:, new_center_idx]
72      print("中心点初始化完成")
73      return centers
74
75
76  ~ def assign_labels(elements, centers): 1个用法
77      """分配样本到最近的中心点"""
78      # 向量化计算所有距离
79      distances = np.linalg.norm(
80          elements.T[:, np.newaxis, :] - centers,
81          axis=2
82      )
83      return np.argmin(distances, axis=1)
84
85
86  ~ def calculate_centroids(elements, labels, K): 1个用法
87      """计算各簇的质心"""
88      new_centers = np.zeros((K, elements.shape[0]))
89  ~ for k in range(K):
90      # 获取当前簇的所有样本
91      cluster_samples = elements[:, labels == k]
92  ~ if cluster_samples.size == 0: # 处理空簇
93      new_centers[k] = elements[:, np.random.choice(elements.shape[1])]
94  ~ else:
95      new_centers[k] = np.mean(cluster_samples, axis=1)
96      return new_centers
97
98

```

图 3.3.3 K-Means 分类（代码）


```

123  def getDistortions(data, krange): 2用法
124  """计算不同k值下k-means聚类结果的distortion指标
125
126  参数:
127  data : numpy.ndarray
128  | 输入数据矩阵, 形状为(dimensions, samples), 每列代表一个样本数据点
129  krange : iterable
130  | 需要测试的k值组成的可迭代对象(如列表/范围)
131
132  返回值:
133  list[float]
134  | 每个k值对应的平均distortion值, 与krange顺序一致
135  """
136  dists = []
137  for k in krange:
138      print("正在计算k=" + str(k) + "的 distortion")
139
140      # 执行k-means聚类算法, 获取聚类中心和标签
141      result_img, centers, labels = kmeans_clustering(
142          elements=data,
143          K=k,
144          LoopN=100, # 适当增大迭代次数
145          Tolerance=1e-4, # 容差阈值
146      )
147
148      # 手动计算所有数据点到对应聚类中心的平均距离
149      total_dist = 0.0
150      for i in range(data.shape[1]):
151          cluster_center = centers[int(labels[i])]
152          dist = np.linalg.norm(data[:,i] - cluster_center)
153          total_dist += dist
154      print("k=" + str(k) + " distortion=" + str(total_dist / data.shape[1]))
155      dists.append(total_dist / data.shape[1])
156
157  return dists

```

图 3.3.4 肘部图计算

4 结果图

4.1 图像滤波处理

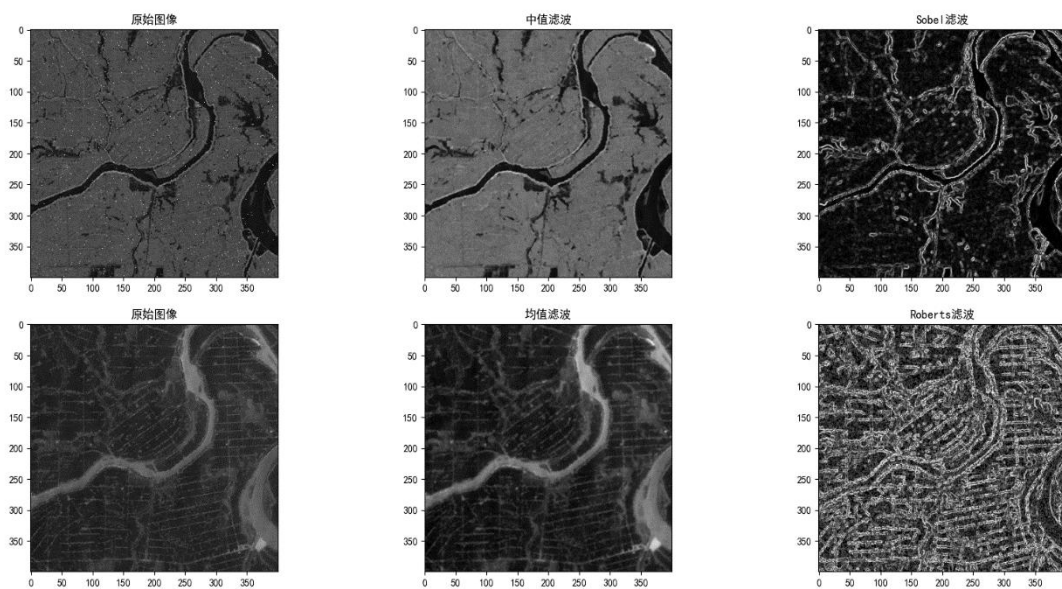


图 4.1 图像滤波处理结果图

4.2 遥感图像分类

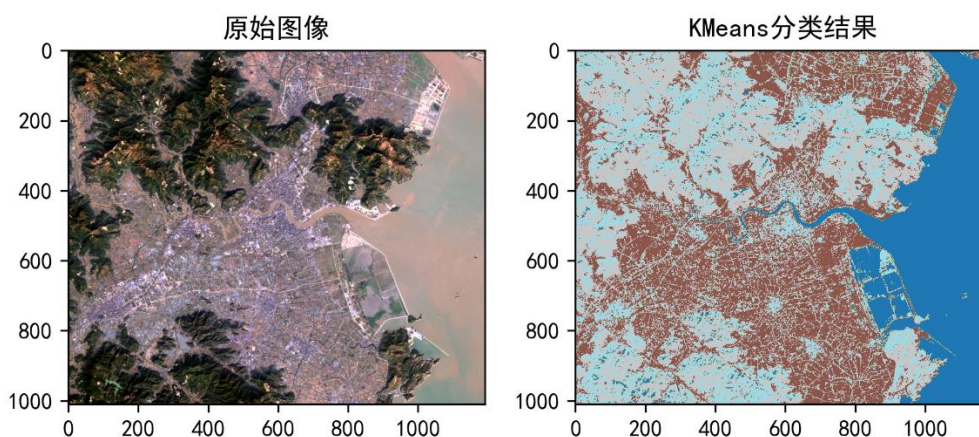


图 4.2.1 K-Means 分类前后对比 (K=5)

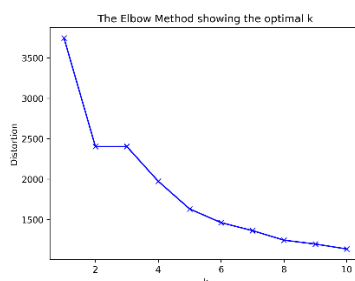


图 4.2.1 肘部图

5 思考题

问题：请设想一下现阶段 AI 能在学习、生活和工作中帮助到自己的地方。

AI 可以作为外置大脑，快速查询各个函数和算法，这样就需要记住关键。

AI 还可以作为免费劳力，编写简单代码，快速添加注释，不同语言转译。

AI 还可以作为私人导师，除了复杂的数学问题，其它基本可以问，讲的很详细。

6 心得体会

有了 AI 的帮助，大大减少了浪费在代码语法、函数查询调用上的时间，我们可以更专注于问题和算法本身，就算是不精通于代码，只要看得懂基本关键字，各个专业领域的人士也可以借用 AI 和大量的库，依靠他们自身的专业知识，来快速搭建一个固定流程处理程序，或者实现复杂的算法和图表，这不仅意味着大量重复科研工作的解放，而且使科研工作的表现力和拓展性大大提升。

在这样的形势下，不论未来是就业还是继续读研究生，效率和生产力的提升必须要学会用好 AI 和互联网资源，不过，代码的基础知识也还是必要的。