**链家小区爬虫及文本分析代码**

目录

[一、 数据爬取 1](#_Toc4575)

[（一） 链家北京小区数据爬虫 1](#_Toc32422)

[（二） 链家北京小区介绍爬虫 5](#_Toc8913)

[二、 数据清洗与另存 9](#_Toc28415)

[三、 文本分析 10](#_Toc14294)

[（一） LDA语料向量化表示 10](#_Toc3803)

[（二） 词云图（小区介绍&周边配套） 12](#_Toc2084)

[（三） 词云图（小区命名） 14](#_Toc10351)

[（四） K-means聚类 16](#_Toc19848)

[（五） TF-IDF关键词提取 18](#_Toc3331)

[（六） TEXtrank关键词提取 19](#_Toc16429)

[四、 分类算法 21](#_Toc6156)

[（一） KNN 21](#_Toc10699)

[（二） 朴素贝叶斯 23](#_Toc8239)

[（三） 随机森林 26](#_Toc4878)

# 数据爬取

## 链家北京小区数据爬虫

import urllib # 用于处理URL  
import openpyxl # 用于Excel文件操作  
import parsel as parsel # 网页解析库  
from selenium import webdriver # 浏览器自动化工具  
from selenium.webdriver.common.by import By # 定位元素方法  
import time # 时间处理模块  
import requests # 发送HTTP请求模块  
import pandas as pd # 数据处理库  
  
# 使用Chrome浏览器作为webdriver  
driver = webdriver.Chrome()  
  
# 打开链家网站首页  
url = 'https://bj.lianjia.com/'  
driver.get(url)  
driver.maximize\_window()  
time.sleep(5) # 等待页面加载  
  
# 点击“小区”对应的链接  
xiaoqu\_link = driver.find\_element(By.XPATH, '//li[@data-click-evtid="20599" and contains(@data-action, "click\_name=小区")]')  
xiaoqu\_link.click()  
time.sleep(10)  
  
# 切换到新打开的窗口  
driver.switch\_to.window(driver.window\_handles[1])  
current\_url = driver.current\_url  
  
# 基于xpath定位，获取页面中的小区链接列表  
links = driver.find\_elements(By.XPATH, '//div[@data-role="ershoufang"]//a')  
  
# 创建一个新的 Excel 文件  
wb = openpyxl.Workbook()  
sheet = wb.active  
sheet.title = 'Sheet1'  
  
# 设置 Excel 表头  
sheet['A1'] = '区划'  
sheet['B1'] = '链接'  
  
# 逐个点击链接，并将 href 和 text 写入 Excel  
row = 2 # 从第二行开始写入数据  
for link in links:  
 text = link.get\_attribute("text") # 获取链接文本  
 href = link.get\_attribute("href") # 获取链接地址  
  
 # 检查 text 和 href 是否存在  
 if text and href:  
 # 写入数据到 Excel 文件中  
 sheet[f'A{row}'] = text  
 sheet[f'B{row}'] = href  
 row += 1  
  
# 保存 Excel 文件  
wb.save('1-各区划链接.xlsx')  
  
time.sleep(10)  
  
# 加载保存的Excel文件  
wb = openpyxl.load\_workbook('1-各区划链接.xlsx')  
sheet = wb['Sheet1']  
result\_list = []  
  
# 遍历 Excel 表中的链接  
for row in range(2, sheet.max\_row + 1):  
 href = sheet[f'B{row}'].value # 获取Excel中的链接  
 if href:  
 driver.get(href) # 访问链接  
 print(href)  
 time.sleep(20)  
 headers = {  
 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.5112.102 Safari/537.36'  
 }  
 # 获取当前页面的完整URL  
 current\_url = driver.current\_url  
 # 如果href是相对链接，将其与基本URL拼接  
 if not href.startswith('http'):  
 href = urllib.parse.urljoin(current\_url, href)  
 # 发送请求并解析页面内容  
 response = requests.get(url=href, headers=headers)  
 content = response.content.decode('utf-8')  
 selector = parsel.Selector(response.text)  
 time.sleep(20)  
 lis = selector.css('.listContent li')  
 # 提取小区信息并保存到列表  
 for li in lis:  
 # 小区名称  
 community\_name = li.css('.title a::text').get()  
 # 销售情况  
 sale = li.css('.houseInfo a::text').getall()  
 # 地址  
 positionInfo = li.css('.positionInfo a::text').getall()  
 # 区划  
 division = positionInfo[0]  
 # 区域  
 address = positionInfo[1]  
 # 材质及年份  
 material = li.css('.positionInfo::text').getall()  
 material\_str = ''.join(material).replace('\n', '').replace(' ', '').replace('\xa0', '')  
 material\_list = material\_str.split('/')  
 building\_type = '/'.join(material\_list[:-1]) # 获取除最后一个元素外的所有元素  
 year\_built = material\_list[-1] # 获取最后一个元素  
 # 周边地铁  
 subway = li.css('.tagList span::text').get()  
 # 房价  
 Price = li.css('.totalPrice span::text').get() + '万'  
 # 在售房数  
 housecount = li.css('.xiaoquListItemSellCount a span::text').get() + '套'  
 dit = {  
 '小区名称': community\_name,  
 '销售情况': sale,  
 '区划': division,  
 '区域': address,  
 '周边地铁': subway,  
 '建筑材质': building\_type,  
 '建造年份': year\_built,  
 '房价': Price,  
 '在售房数': housecount,  
 }  
 result\_list.append(dit)  
 print(dit)  
  
 for page in range(2, 30):  
 # 更新URL以包含页码  
 page\_url = f"{href}pg{page}/"  
 driver.get(page\_url)  
 print(page\_url)  
 time.sleep(20)  
 headers = {  
 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.5112.102 Safari/537.36'  
 }  
 # 获取当前页面的完整URL  
 current\_url = driver.current\_url  
 # 如果href是相对链接，将其与基本URL拼接  
 if not page\_url.startswith('http'):  
 page\_url = urllib.parse.urljoin(current\_url, page\_url)  
 response = requests.get(url=page\_url, headers=headers)  
 content = response.content.decode('utf-8')  
 selector = parsel.Selector(response.text)  
 lis = selector.css('.listContent li')  
 for li in lis:  
 # 标题  
 community\_name = li.css('.title a::text').get()  
 # 出售情况  
 sale = li.css('.houseInfo a::text').getall()  
 # 地址  
 positionInfo = li.css('.positionInfo a::text').getall()  
 # 区划  
 division = positionInfo[0]  
 # 地名  
 address = positionInfo[1]  
 # 材质及年份  
 material = li.css('.positionInfo::text').getall()  
 material\_str = ''.join(material).replace('\n', '').replace(' ', '').replace('\xa0', '')  
 material\_list = material\_str.split('/')  
 building\_type = '/'.join(material\_list[:-1]) # 获取除最后一个元素外的所有元素  
 year\_built = material\_list[-1] # 获取最后一个元素  
 # 周边地铁  
 subway = li.css('.tagList span::text').get()  
 # 房价  
 Price = li.css('.totalPrice span::text').get() + '万'  
 # 房子套数  
 housecount = li.css('.xiaoquListItemSellCount a span::text').get() + '套'  
 dit = {  
 '小区名称': community\_name,  
 '销售情况': sale,  
 '区划': division,  
 '区域': address,  
 '周边地铁': subway,  
 '建筑材质': building\_type,  
 '建造年份': year\_built,  
 '房价': Price,  
 '在售房数': housecount,  
 }  
 result\_list.append(dit)  
 print(dit)  
 # 将结果列表转换为DataFrame并保存到Excel文件中  
 df = pd.DataFrame(result\_list)  
 df.to\_excel('1-北京小区数据.xlsx', index=False)  
 time.sleep(50)

## 链家北京小区介绍爬虫

# 导入所需的库  
import openpyxl # 用于处理 Excel 文件  
from selenium import webdriver # 用于模拟浏览器操作  
from selenium.webdriver.common.by import By # 用于定位网页元素  
import time # 用于添加延迟等待  
import requests # 用于发送 HTTP 请求  
import parsel # 用于解析 HTML 数据  
import pandas as pd # 用于处理数据和生成 Excel 文件  
  
# 使用 Selenium webdriver 打开 Chrome 浏览器  
driver = webdriver.Chrome()  
  
# 打开链家网站首页  
url = 'https://bj.lianjia.com/'  
driver.get(url)  
driver.maximize\_window()  
time.sleep(5)  
  
# 点击“二手房”对应的链接  
xiaoqu\_link = driver.find\_element(By.XPATH,'//li[@data-click-evtid="20599" and contains(@data-action, "click\_name=二手房")]')  
xiaoqu\_link.click()  
time.sleep(10)  
  
# 切换到新打开的窗口  
driver.switch\_to.window(driver.window\_handles[1])  
current\_url = driver.current\_url  
  
# 创建一个新的 Excel 文件  
wb = openpyxl.Workbook()  
sheet = wb.active  
sheet.title = 'Sheet1'  
  
# 设置 Excel 表头  
sheet['A1'] = '二手房'  
sheet['B1'] = '链接'  
  
# 逐个点击链接，并将 href 和 text 写入 Excel  
row = 2 # 从第二行开始写入数据  
  
# 获取当前页面中的小区链接列表  
links = driver.find\_elements(By.XPATH, '//div[@class="info clear"]//a')  
  
# 遍历链接列表，获取文本和链接，并写入 Excel 文件  
for link in links:  
 text = link.get\_attribute("text")  
 href = link.get\_attribute("href")  
 print(text)  
 print(href)  
 # 检查 text 和 href 是否存在  
 if text and href:  
 # 写入数据到 Excel 文件中  
 sheet[f'A{row}'] = text  
 sheet[f'B{row}'] = href  
 row += 1  
  
time.sleep(40)  
  
# 遍历多个页面  
for page in range(2, 100):  
 # 更新 URL 以包含页码  
 page\_url = f"{current\_url}pg{page}/"  
 driver.get(page\_url)  
 print(page\_url)  
 links = driver.find\_elements(By.XPATH, '//div[@class="info clear"]//a')  
 for link in links:  
 text = link.get\_attribute("text")  
 href = link.get\_attribute("href")  
  
 # 检查 text 和 href 是否存在  
 if text and href:  
 # 写入数据到 Excel 文件中  
 sheet[f'A{row}'] = text  
 sheet[f'B{row}'] = href  
 row += 1  
 time.sleep(40)  
  
# 保存 Excel 文件  
wb.save('2-二手房链 接.xlsx')  
time.sleep(10)  
  
# 读取之前保存的 Excel 文件  
wb = openpyxl.load\_workbook('2-二手房链接.xlsx')  
sheet = wb['Sheet1']  
result\_list = []  
  
# 遍历 Excel 文件中的链接  
for row in range(2, sheet.max\_row + 1, 3):  
 href = sheet[f'B{row}'].value  
 if href:  
 try:  
 # 获取链接内容并解析  
 driver.get(href)  
 time.sleep(10)  
 headers = {  
 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/104.0.5112.102 Safari/537.36'  
 }  
  
 response = requests.get(url=href, headers=headers)  
 content = response.content.decode('utf-8')  
 selector = parsel.Selector(response.text)  
 time.sleep(5)  
  
 # 提取相关信息  
 community1 = selector.css('.introContent.showbasemore div')  
 community = community1.css('.baseattribute.clear div::text').getall()  
 print(community)  
 division = community[1]  
 address = community[3]  
 address1 = community[5]  
  
 lis = selector.css('.aroundInfo div')  
 sale = lis.css('.communityName a::text').getall()  
 division1 = sale[0]  
  
 dit = {  
 '小区名称': division1,  
 '核心卖点': division,  
 '小区介绍': address,  
 '周边配套': address1,  
 }  
 result\_list.append(dit)  
 print(dit)  
  
 # 将数据存入 DataFrame 并保存到 Excel 文件中  
 df = pd.DataFrame(result\_list)  
 df.to\_excel('2-北京小区简介数据.xlsx', index=False)  
 except Exception as e:  
 print(f"Error processing URL {href}: {e}")  
 continue  
 time.sleep(5)

# 数据清洗与另存

**目的及功能：**

主要目的为笔记文本提取、处理与另存，具体步骤如下：

1. 从指定的 CSV 文件中读取数据，并将每个文件的最后一列保存到对应的 TXT 文件中。
2. 建立一个停用词集合，将从文件中读取的停用词添加到集合中。
3. 读取之前保存的 TXT 文件内容，并过滤掉其中的停用词，将处理后的内容重新写入相应的 TXT 文件中。

**代码：**

import pandas as pd # 导入Pandas库用于数据处理  
  
# 读取Excel文件  
excel\_file1 = pd.ExcelFile('2-北京小区简介数据.xlsx') # 读取包含小区简介数据的Excel文件  
excel\_file2 = pd.ExcelFile('1-北京小区数据.xlsx') # 读取包含北京小区数据的Excel文件  
  
# 读取第三列和第四列数据  
df1 = excel\_file1.parse(excel\_file1.sheet\_names[0]) # 解析第一个Excel文件的第一个表单数据  
column\_3\_data = df1.iloc[:, 2] # 获取第一个Excel文件中的第三列数据  
column\_4\_data = df1.iloc[:, 3] # 获取第一个Excel文件中的第四列数据  
df2 = excel\_file2.parse(excel\_file2.sheet\_names[0]) # 解析第二个Excel文件的第一个表单数据  
column\_5\_data = df2.iloc[:, 0] # 获取第二个Excel文件中的第一列数据  
  
# 将第三列数据保存为小区介绍.txt  
with open('3-小区介绍.txt', 'w', encoding='utf-8') as file:  
 for item in column\_3\_data:  
 file.write("%s\n" % item) # 将第三列数据逐行写入小区介绍文件  
  
# 将第四列数据保存为周边配套.txt  
with open('3-周边配套.txt', 'w', encoding='utf-8') as file:  
 for item in column\_4\_data:  
 file.write("%s\n" % item) # 将第四列数据逐行写入周边配套文件  
  
# 将第一列数据保存为小区命名.txt  
with open('3-小区命名.txt', 'w', encoding='utf-8') as file:  
 for item in column\_5\_data:  
 file.write("%s\n" % item) # 将第一列数据逐行写入小区命名文件  
  
# 处理停用词  
stopwords = set() # 创建空集合用于存储停用词  
with open('StopwordsCN.txt', 'r', encoding='utf-8') as file:  
 for line in file:  
 stopwords.add(line.strip()) # 读取并存储停用词  
  
# 处理小区介绍.txt  
with open('3-小区介绍.txt', 'r', encoding='utf-8') as file:  
 text = file.read() # 读取小区介绍文件内容  
 text = ' '.join(word for word in text.split() if word not in stopwords) # 移除停用词并重新组合文本内容  
  
with open('3-小区介绍.txt', 'w', encoding='utf-8') as file:  
 file.write(text) # 将处理后的文本内容写入小区介绍文件  
  
# 处理周边配套.txt  
with open('3-周边配套.txt', 'r', encoding='utf-8') as file:  
 text = file.read() # 读取周边配套文件内容  
 text = ' '.join(word for word in text.split() if word not in stopwords) # 移除停用词并重新组合文本内容  
  
with open('3-周边配套.txt', 'w', encoding='utf-8') as file:  
 file.write(text) # 将处理后的文本内容写入周边配套文件  
  
# 处理小区命名.txt  
with open('3-小区命名.txt', 'r', encoding='utf-8') as file:  
 text = file.read() # 读取小区命名文件内容  
 text = ' '.join(word for word in text.split() if word not in stopwords) # 移除停用词并重新组合文本内容  
  
with open('3-小区命名.txt', 'w', encoding='utf-8') as file:  
 file.write(text) # 将处理后的文本内容写入小区命名文件

# 文本分析

## LDA-主题分析建模

**目的及功能：**

主要目的为通过LDA模型（Latent Dirichlet Allocation）对文本数据进行主题建模，具体步骤如下。

1. 数据预处理：读取给定文件路径的文本数据，使用结巴分词工具进行分词，并进行名词和形容词的筛选与停用词去除。
2. LDA主题模型构建：将处理后的文本数据转换成适合训练LDA模型的格式。创建词典并将训练数据转换为词袋模型表示，然后使用LDA模型训练生成主题模型。
3. 主题分析和可视化：获取每个主题中的词汇和权重，将结果保存到CSV文件，并生成LDA主题可视化的HTML文件。

模型评估：计算不同主题数下LDA模型的困惑度（perplexity）并绘制图表。同时计算不同主题数下LDA模型的coherence主题一致性并绘制图表。

**代码：**

import pandas as pd # 导入 pandas 模块  
from gensim import corpora # 用于构建文本的字典和语料库  
import jieba # 中文分词工具  
import pyLDAvis.gensim # 用于LDA主题模型可视化  
import matplotlib.pyplot as plt # 用于绘图  
import matplotlib # matplotlib 库  
from gensim.models.coherencemodel import CoherenceModel # 用于计算主题一致性  
from gensim.models.ldamodel import LdaModel # LDA主题模型  
import warnings # 用于警告处理  
import jieba.posseg as pseg # 结巴分词词性标注工具  
  
warnings.filterwarnings('ignore') # 忽略警告  
jieba.setLogLevel(jieba.logging.INFO) # 设置结巴分词的日志级别  
  
  
# 计算困惑度  
def perplexity(ldamodel, corpus):  
 return ldamodel.log\_perplexity(corpus)  
  
  
"""  
 计算LDA模型的困惑度  
  
 参数:  
 ldamodel: gensim中的LDA模型对象  
 corpus: 文档语料库的词袋表示形式  
  
 返回值:  
 float: LDA模型的困惑度  
 """  
  
  
# 计算 coherence 主题一致性  
def coherence(ldamodel, texts, dictionary, corpus):  
 coh = []  
 for num\_topics in range(1, 11):  
 ldamodel = LdaModel(corpus, num\_topics=num\_topics, id2word=dictionary, passes=30, random\_state=1)  
 print(ldamodel.print\_topics(num\_topics=num\_topics, num\_words=10))  
 ldacm = CoherenceModel(model=ldamodel, texts=texts, dictionary=dictionary, coherence='c\_v', corpus=corpus)  
 coh.append(ldacm.get\_coherence())  
 return coh  
  
  
"""  
 计算LDA模型的coherence主题一致性得分  
  
 参数:  
 ldamodel: gensim中的LDA模型对象  
 texts: 文档列表，每个文档包含单词的列表  
 dictionary: gensim中的词典对象  
 corpus: 文档语料库的词袋表示形式  
  
 返回值:  
 list: 包含不同主题数目下的coherence得分的列表  
 """  
  
  
# 修改数据清理和分词函数  
def tokenize\_file(file\_path):  
 *"""对文件进行分词，并选择名词进行保留"""* with open(file\_path, 'r', encoding='utf-8') as file:  
 data = file.read()  
 words = pseg.cut(data) # 使用 jieba 的词性标注功能  
 stopword = {line.strip() for line in open("StopwordsCN.txt", encoding='utf-8')}  
 filtered\_words = [word for word, flag in words if  
 (flag.startswith('n') or flag.startswith('a')) and word.strip() and word not in stopword]  
 return filtered\_words  
  
  
"""  
 对文件进行分词，并选择名词进行保留  
  
 参数:  
 file\_path: 文件路径字符串  
  
 返回值:  
 list: 分词后的词汇列表  
 """  
if \_\_name\_\_ == '\_\_main\_\_':  
 file\_paths = ['3-小区命名.txt', '3-小区介绍.txt', '3-周边配套.txt']  
  
 for idx, file\_path in enumerate(file\_paths, start=1):  
 print(f"Processing file {idx}: {file\_path}")  
 df\_data = tokenize\_file(file\_path)  
  
 train = [[word] for word in df\_data]  
  
 dictionary = corpora.Dictionary(train)  
 corpus = [dictionary.doc2bow(text) for text in train]  
  
 print("词典构建完成")  
  
 ldamodel = LdaModel(corpus=corpus, id2word=dictionary, num\_topics=5, passes=80, random\_state=0)  
 """  
 LdaModel 参数说明：  
 corpus: 语料库，文档-词袋格式。  
 num\_topics: 主题数量。  
 id2word: 词袋模型对应的词典。  
 passes: 迭代次数。  
 random\_state: 随机数种子。  
 """  
 topic\_list = ldamodel.print\_topics()  
 print(topic\_list)  
 # 获取每个主题中的所有词汇和权重  
 topics\_words = ldamodel.show\_topics(formatted=False, num\_words=10000) # 每个主题显示所有词汇  
  
 # 创建一个空的 DataFrame 以保存主题词汇和权重  
 topics\_data = pd.DataFrame(columns=['主题', '词汇', '权重'])  
  
 # 将词汇和权重添加到 DataFrame  
 for idx, topic in topics\_words:  
 words = [word for word, \_ in topic]  
 weights = [weight for \_, weight in topic]  
 topic\_df = pd.DataFrame({'主题': [idx + 1] \* len(words), '词汇': words, '权重': weights})  
 topics\_data = pd.concat([topics\_data, topic\_df], ignore\_index=True)  
  
 # 将结果保存到 CSV 文件  
 csv\_file = f'4-{file\_path.split("-")[-1].replace(".txt", "")}-LDA主题.csv'  
 topics\_data.to\_csv(csv\_file, index=False)  
 print(f"Saved topics words and weights to CSV file: {csv\_file}")  
  
 data = pyLDAvis.gensim.prepare(ldamodel, corpus, dictionary)  
 html\_file = f'4-{file\_path.split("-")[-1].replace(".txt", "")}-LDA主题分析建模.html'  
 pyLDAvis.save\_html(data, html\_file)  
 print(f"Saved HTML visualization: {html\_file}")  
  
 x = range(1, 11)  
  
 z = [perplexity(ldamodel, corpus) for i in x]  
 print(z)  
 plt.plot(x, z)  
 plt.rcParams['font.sans-serif'] = ['SimHei']  
 plt.xlabel('主题数目')  
 plt.ylabel("困惑度")  
 matplotlib.rcParams['axes.unicode\_minus'] = False  
 plt.title('主题-困惑度变化情况统计')  
 plt.savefig(f'4-{file\_path.split("-")[-1].replace(".txt", "")}-LDA-perplexity.png') # 先保存再显示  
 plt.show()  
 print(f"Perplexity plot {idx} completed")  
  
 # 计算 coherence 并生成图表  
 y = coherence(ldamodel, train, dictionary, corpus)  
 print(y)  
 plt.plot(x, y)  
 plt.rcParams['font.sans-serif'] = ['SimHei']  
 plt.xlabel('主题数目')  
 plt.ylabel("coherence大小")  
 matplotlib.rcParams['axes.unicode\_minus'] = False  
 plt.title('主题-coherence变化情况统计')  
 plt.savefig(f'4-{file\_path.split("-")[-1].replace(".txt", "")}-LDA-coherence.png') # 先保存再显示  
 plt.show()  
 print(f"Coherence plot {idx} completed")

## 词云图（小区介绍&周边配套）

# 导入必要的库  
import matplotlib.pyplot as plt # 用于绘制图表  
from wordcloud import WordCloud # 生成词云图的库  
import jieba # 中文分词库  
import imageio # 用于读取图像文件的库  
import re # 正则表达式库，用于文本处理  
from PIL import Image # PIL库用于处理图像  
  
# 读取两个txt文件中的内容  
file1 = open("3-小区介绍.txt", encoding='utf-8')  
file2 = open("3-周边配套.txt", encoding='utf-8')  
article1 = file1.read()  
article2 = file2.read()  
file1.close()  
file2.close()  
  
# 使用jieba进行分词处理  
seg\_str1 = jieba.cut(article1, cut\_all=False)  
seg\_str2 = jieba.cut(article2, cut\_all=False)  
  
liststr1 = "/".join(seg\_str1) # 连接分词结果  
liststr2 = "/".join(seg\_str2)  
  
# 删除文本中的所有数字  
article\_no\_numbers1 = re.sub(r'\d+', '', liststr1)  
article\_no\_numbers2 = re.sub(r'\d+', '', liststr2)  
  
# 打开停用词文件  
stopwords\_path = "StopwordsCN.txt"  
f\_stop = open(stopwords\_path, encoding='utf-8')  
f\_stop\_text = f\_stop.read()  
f\_stop.close()  
  
# 创建空列表存储最终的词  
mywordlist1 = []  
mywordlist2 = []  
f\_stop\_seg\_list = f\_stop\_text.split("\n")  
  
# 过滤停用词，将不在停用词表中且长度大于1的词存入列表  
for myword1 in article\_no\_numbers1.split('/'):  
 if not (myword1.strip() in f\_stop\_seg\_list) and len(myword1.strip()) > 1:  
 mywordlist1.append(myword1)  
  
for myword2 in article\_no\_numbers2.split('/'):  
 if not (myword2.strip() in f\_stop\_seg\_list) and len(myword2.strip()) > 1:  
 mywordlist2.append(myword2)  
  
# 创建字典存储词频统计  
word\_freq1 = {}  
word\_freq2 = {}  
  
# 统计每个词的词频  
for myword in mywordlist1:  
 word\_freq1[myword] = word\_freq1.get(myword, 0) + 1  
  
for myword in mywordlist2:  
 word\_freq2[myword] = word\_freq2.get(myword, 0) + 1  
  
# 按词频降序排列  
sorted\_word\_freq1 = sorted(word\_freq1.items(), key=lambda x: x[1], reverse=True)  
sorted\_word\_freq2 = sorted(word\_freq2.items(), key=lambda x: x[1], reverse=True)  
  
# 提取词频前140的词  
top\_140\_words1 = dict(sorted\_word\_freq1[:140])  
top\_140\_words2 = dict(sorted\_word\_freq2[:140])  
  
# 输出词频前140的词  
print(top\_140\_words1)  
print(top\_140\_words2)  
  
# 载入背景图片  
bg\_mask = "5-词云图素材-房子简笔画2.jpg" # 背景图片路径  
mask\_img = imageio.imread(bg\_mask) # 使用imageio库打开图像文件  
  
# 生成词云图1（词频前140的词）  
wordcloud1 = WordCloud(font\_path='msyh.ttc', background\_color='white', mask=mask\_img).generate\_from\_frequencies(top\_140\_words1)  
wordcloud1.to\_file("5-词云图-小区介绍.jpg")  
  
# 生成词云图2（词频前140的词）  
wordcloud2 = WordCloud(font\_path='msyh.ttc', background\_color='white', mask=mask\_img).generate\_from\_frequencies(top\_140\_words2)  
wordcloud2.to\_file("5-词云图-周边设施.jpg")  
  
# 绘制词云图1  
plt.figure(figsize=(10, 8))  
plt.imshow(wordcloud1, interpolation='bilinear') # 使用双线性插值的方式渲染图像  
plt.axis("off") # 不显示坐标轴  
plt.show()  
  
# 绘制词云图2  
plt.figure(figsize=(10, 8))  
plt.imshow(wordcloud2, interpolation='bilinear') # 使用双线性插值的方式渲染图像  
plt.axis("off") # 不显示坐标轴  
plt.show()

## 词云图（小区命名）

# 导入必要的库  
import matplotlib.pyplot as plt # 用于绘制图表  
from wordcloud import WordCloud # 用于生成词云图  
import jieba # 用于中文分词  
import imageio.v2 as imageio # 导入图像处理库  
import re # 用于正则表达式操作  
from PIL import Image # 导入PIL库，用于图像处理  
  
# 打开并读取txt文件内容  
file = open("3-小区命名.txt", encoding='utf-8')  
article = file.read()  
file.close()  
  
# 使用jieba进行中文分词  
seg\_str = jieba.cut(article, cut\_all=False)  
liststr = "/".join(seg\_str)  
  
# 删除文本中的所有数字  
article\_no\_numbers = re.sub(r'\d+', '', liststr)  
  
# 打开并读取停用词文件  
stopwords\_path = "StopwordsCN.txt"  
f\_stop = open(stopwords\_path, encoding='utf-8')  
f\_stop\_text = f\_stop.read()  
f\_stop.close()  
  
# 创建停用词列表  
mywordlist = []  
  
# 将停用词读取并放入列表中  
f\_stop\_seg\_list = f\_stop\_text.split("\n")  
for myword in article\_no\_numbers.split('/'):  
 # 剔除停用词和长度小于等于1的词  
 if not (myword.strip() in f\_stop\_seg\_list) and len(myword.strip()) > 1:  
 mywordlist.append(myword)  
  
# 创建空字典存储词频统计  
word\_freq = {}  
  
# 对每个词进行词频统计  
for myword in mywordlist:  
 word\_freq[myword] = word\_freq.get(myword, 0) + 1  
  
# 按词频降序排列  
sorted\_word\_freq = sorted(word\_freq.items(), key=lambda x: x[1], reverse=True)  
  
# 提取词频前140的词和词频140-270的词  
top\_140\_words = dict(sorted\_word\_freq[:140])  
words\_140\_to\_270 = dict(sorted\_word\_freq[140:270])  
  
# 载入背景图片  
bg\_mask = "5-词云图素材-房子简笔画.jpg" # 背景图片路径  
mask\_img = imageio.imread(bg\_mask) # 使用PIL库打开图像文件  
  
# 生成词云图1（词频前140的词）  
wordcloud1 = WordCloud(font\_path='msyh.ttc', background\_color='white', mask=mask\_img).generate\_from\_frequencies(top\_140\_words)  
wordcloud1.to\_file("5-词云图-小区命名1.jpg")  
  
# 生成词云图2（词频140-270的词）  
wordcloud2 = WordCloud(font\_path='msyh.ttc', background\_color='white', mask=mask\_img, max\_font\_size=60).generate\_from\_frequencies(words\_140\_to\_270)  
wordcloud2.to\_file("5-词云图-小区命名2.jpg")  
  
# 绘制词云图1  
plt.figure(figsize=(10, 8))  
plt.imshow(wordcloud1, interpolation='bilinear')  
plt.axis("off")  
plt.show()  
  
# 绘制词云图2  
plt.figure(figsize=(10, 8))  
plt.imshow(wordcloud2, interpolation='bilinear')  
plt.axis("off")  
plt.show()

## K-means聚类

**目的及功能：**

这段代码的目的是依据相关数据对北京小区数据进行聚类分析。具体步骤为：

从Excel文件读取北京小区数据，进行数据预处理，包括从销售情况和房价文本中提取相关信息，选择特定特征用于K-means聚类，转换分类变量为数值型，然后利用KMeans模型进行聚类。最后，将聚类结果保存为CSV文件，并通过散点图可视化不同簇的房价和在售房数，标注聚类中心点，并将图表保存为图片。

**代码：**

# 导入必要的库  
import pandas as pd  
from sklearn.cluster import KMeans  
from sklearn.preprocessing import LabelEncoder  
import re  
import matplotlib.pyplot as plt  
  
# 读取Excel文件中的数据  
df = pd.read\_excel('1-北京小区数据.xlsx', sheet\_name='Sheet1')  
  
  
# 数据预处理  
  
# 处理销售情况数据：提取成交数量和正在出租的数量  
def extract\_sales(text):  
 *"""  
 从销售情况文本中提取成交数量和正在出租的数量。  
  
 参数:  
 - text: 销售情况文本  
  
 返回:  
 - 成交数量, 正在出租数量  
 """* # 通过正则表达式提取所有数字  
 digits = re.findall(r'\d+', str(text))  
  
 if digits:  
 # 返回第一个数字作为成交数量，最后一个数字作为正在出租数量  
 return int(digits[0]), int(digits[-1])  
 return pd.NA, pd.NA  
  
  
# 应用extract\_sales函数到 '销售情况' 列，并将结果分别赋给 '成交数量' 和 '正在出租' 列  
df['成交数量'], df['正在出租'] = zip(\*df['销售情况'].apply(extract\_sales))  
  
  
# 处理房价数据：提取数字并转换为浮点数  
def extract\_price(text):  
 *"""  
 从房价文本中提取数字并转换为浮点数。  
  
 参数:  
 - text: 房价文本  
  
 返回:  
 - 房价（浮点数）  
 """* if '万' in str(text):  
 # 通过正则表达式提取所有数字（包括小数）  
 digits = re.findall(r'\d+\.\*\d\*', str(text))  
  
 if digits:  
 # 返回提取的数字作为浮点数  
 return float(digits[0])  
 return pd.NA  
  
  
# 应用extract\_price函数到 '房价' 和 '在售房数' 列  
df['房价'] = df['房价'].apply(extract\_price)  
df['在售房数'] = df['在售房数'].apply(extract\_price)  
  
# 特征提取  
selected\_features = ['建筑材质', '房价', '在售房数', '成交数量', '正在出租']  
  
# 将分类变量转换为数值型数据  
label\_encoder = LabelEncoder()  
# 使用LabelEncoder将 '建筑材质' 列中的文本标签转换为数字  
df['建筑材质'] = label\_encoder.fit\_transform(df['建筑材质'])  
  
# 创建特征矩阵X  
X = df[selected\_features]  
  
# 将缺失值用0填充  
X.fillna(0, inplace=True)  
  
# 实例化KMeans模型并应用于数据  
kmeans = KMeans(n\_clusters=7) # 设置聚类的数量  
clusters = kmeans.fit\_predict(X)  
  
# 将聚类结果添加到原始数据中  
df['Cluster'] = clusters  
  
# 打印聚类结果  
print(df[['小区名称', 'Cluster']])  
# 将结果保存为CSV文件  
df.to\_csv('6-kmeans结果.csv', index=False)  
  
# 可视化聚类结果  
plt.figure(figsize=(8, 6))  
for i in range(7):  
 # 根据每个聚类结果绘制散点图  
 plt.scatter(X[clusters == i]['房价'], X[clusters == i]['正在出租'], label=f'Cluster {i}')  
  
# 绘制聚类中心点  
plt.scatter(kmeans.cluster\_centers\_[:, 1], kmeans.cluster\_centers\_[:, 4], s=100, c='black', label='Centroids')  
  
# 设置中文显示  
plt.rcParams['font.sans-serif'] = ['SimHei']  
plt.rcParams['axes.unicode\_minus'] = False  
plt.xlabel('房价', loc='right')  
plt.ylabel('在售房数', loc='top')  
plt.title('K-means Clustering Visualization')  
plt.legend()  
# 将图表保存为图片文件  
plt.savefig("6-k-means聚类.jpg")  
plt.show()

## TF-IDF关键词提取

**目的及功能：**

主要功能包括关键词提取、TF-IDF特征提取、PCA数据降维并进行可视化展示。具体步骤如下：

1. 文件读取和处理：通过os模块读取文件内容，使用jieba对中文文本进行分词和关键词提取。然后将提取的关键词和权重存储到一个数据框中，并将这些数据保存为CSV文件。
2. 文本特征提取：使用TfidfVectorizer从关键词列表构建TF-IDF特征矩阵。TF-IDF是一种常用于信息检索和文本挖掘的技术，用于评估一个词语在文档集合中的重要程度。
3. 数据降维和可视化：对TF-IDF特征矩阵使用PCA进行二维降维，将高维的TF-IDF特征映射到一个二维平面上，并使用散点图（scatter plot）进行可视化展示。图像保存和展示：设置图形的标题、图例、字体样式等参数，并将生成的可视化图像保存为JPG文件，最后展示图像。

**代码：**

# 导入必要的库  
import re # 用于正则表达式的处理  
import os # 用于文件路径操作  
import jieba.analyse # 结巴分词中的关键词提取工具  
import matplotlib.pyplot as plt # 用于绘图  
import pandas as pd # 用于数据处理  
import numpy as np # 用于数值计算  
from sklearn.feature\_extraction.text import TfidfVectorizer # 用于构建 TF-IDF 特征矩阵  
from sklearn.decomposition import PCA # 用于主成分分析（PCA）  
  
# 停用词文件路径  
stopwords\_path = "StopwordsCN.txt"  
  
# 设置结巴分词的停用词表  
jieba.analyse.set\_stop\_words(stopwords\_path)  
  
# 要处理的文件名列表  
file\_names = ['3-周边配套.txt', '3-小区介绍.txt', '3-小区命名.txt']  
  
# 遍历每个文件  
for file\_name in file\_names:  
 # 读取文件内容  
 data\_path = os.path.join(os.getcwd(), file\_name)  
 text = open(data\_path, "r", encoding="utf-8").read()  
  
 # 使用jieba进行关键词提取（TF-IDF方法）  
 result0 = jieba.analyse.extract\_tags(text, topK=400, withWeight=True)  
 print(result0)  
  
 # 构建关键词与词频占比的数据框  
 result\_dict = dict(result0)  
 test = pd.DataFrame(result\_dict.items(), columns=["关键词", "词频占比"])  
  
 # 生成输出的CSV文件名  
 chinese\_part = re.findall(r'[\u4e00-\u9fa5]+', file\_name)  
 csv\_name = "7-" + chinese\_part[0] + "-TF-IDF关键词提取.csv"  
  
 # 将结果保存到CSV文件  
 test.to\_csv(csv\_name, encoding="gbk")  
  
 # 从CSV文件读取数据  
 data = pd.read\_csv(csv\_name, encoding="gbk")  
  
 # 构建 TF-IDF 特征矩阵  
 corpus = data['关键词'].values.tolist()  
 vectorizer = TfidfVectorizer()  
 X = vectorizer.fit\_transform(corpus)  
  
 # 使用PCA进行二维降维  
 pca = PCA(n\_components=2) # 降至二维  
 X\_pca = pca.fit\_transform(X.toarray())  
  
 # 可视化降维后的数据  
 plt.figure(figsize=(10, 5))  
 plt.scatter(X\_pca[:, 0], X\_pca[:, 1], alpha=0.5)  
 plt.title(f'PCA Visualization of TF-IDF - {file\_name}')  
 plt.rcParams['font.sans-serif'] = ['SimHei']  
 plt.rcParams['axes.unicode\_minus'] = False  
 image\_name = "7-" + chinese\_part[0] + "-TF-IDF关键词提取.jpg"  
 plt.savefig(image\_name) # 保存图像为文件  
 plt.show() # 显示图像

## TEXtrank关键词提取

**目的及功能：**

对给定文本进行关键词、关键短语、关键句子的分析，并可视化展示关键词的词云图

**代码：**

# 导入所需模块  
import imageio.v2 as imageio  
from matplotlib import pyplot as plt  
from textrank4zh import TextRank4Keyword, TextRank4Sentence  
from wordcloud import WordCloud  
import jieba.analyse  
  
  
# 从文件中读取文本内容  
def read\_text\_from\_file(file\_path):  
 *"""  
 读取指定文件的文本内容。  
  
 Parameters:  
 - file\_path (str): 文件路径。  
  
 Returns:  
 - str: 读取到的文本内容。  
 """* with open(file\_path, 'r', encoding='utf-8') as file:  
 return file.read()  
  
  
# 提取关键词  
def keywords\_extraction(text):  
 *"""  
 使用 TextRank4Keyword 进行关键词分析。  
  
 Parameters:  
 - text (str): 待分析的文本。  
  
 Returns:  
 - list: 包含关键词的列表。  
 """* tr4w = TextRank4Keyword()  
 tr4w.analyze(text, window=2, lower=True)  
 keywords = tr4w.get\_keywords(num=100, word\_min\_len=2)  
 return [item.word for item in keywords]  
  
  
# 生成词云  
def generate\_wordcloud(keywords, title, bg\_image):  
 *"""  
 生成并展示词云图。  
  
 Parameters:  
 - keywords (list): 包含关键词的列表。  
 - title (str): 词云图的标题。  
 - bg\_image (str): 背景图片的路径。  
  
 Returns:  
 - None  
 """* keywords\_str = ' '.join(keywords)  
 imgbg = imageio.imread(bg\_image)  
 wordcloud = WordCloud(width=800, height=400, background\_color='white', font\_path='msyh.ttc', mask=imgbg).generate(  
 keywords\_str)  
 plt.figure(figsize=(10, 5))  
 plt.rcParams['font.sans-serif'] = ['SimHei']  
 plt.imshow(wordcloud, interpolation='bilinear')  
 plt.axis('off')  
 plt.title(f'Word Cloud - {title}')  
 wordcloud.to\_file(f"{title}.png")  
 plt.show()  
  
  
# 提取关键短语  
def keyphrases\_extraction(text):  
 *"""  
 使用 TextRank4Keyword 进行关键短语分析。  
  
 Parameters:  
 - text (str): 待分析的文本。  
  
 Returns:  
 - list: 包含关键短语的列表。  
 """* tr4w = TextRank4Keyword()  
 tr4w.analyze(text, window=2, lower=True)  
 keyphrases = tr4w.get\_keyphrases(keywords\_num=20, min\_occur\_num=1)  
 return keyphrases  
  
  
# 提取关键句子  
def keysentences\_extraction(text):  
 *"""  
 使用 TextRank4Sentence 进行关键句子分析。  
  
 Parameters:  
 - text (str): 待分析的文本。  
  
 Returns:  
 - list: 包含关键句子的列表。  
 """* tr4s = TextRank4Sentence()  
 tr4s.analyze(text, lower=True)  
 keysentences = tr4s.get\_key\_sentences(num=5, sentence\_max\_len=12)  
 return keysentences  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # 文件路径  
 data\_path = r'3-小区介绍.txt'  
 bg\_image\_path = "5-词云图素材-房子简笔画.jpg"  
  
 # 从文件中读取文本内容  
 text = read\_text\_from\_file(data\_path)  
  
 # 提取关键词并生成词云图  
 keywords = keywords\_extraction(text)  
 generate\_wordcloud(keywords, '小区介绍关键词词云', bg\_image\_path)  
  
 # 提取关键短语并打印输出  
 keyphrases = keyphrases\_extraction(text)  
 print(keyphrases)  
  
 # 提取关键句子并打印输出  
 keysentences = keysentences\_extraction(text)  
 print(keysentences)

# 分类算法

目的：进一步掌握本学期所学分类算法相关知识

## **KNN**

import pandas as pd  
from sklearn.preprocessing import LabelEncoder  
from sklearn.model\_selection import train\_test\_split  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.metrics import classification\_report, confusion\_matrix  
import matplotlib.pyplot as plt  
import numpy as np  
  
def plot\_confusion\_matrix(y, yp, filename):  
 *"""可视化混淆矩阵并保存为 JPG 文件"""* cm = confusion\_matrix(y, yp)  
 plt.figure(figsize=(10, 10)) # 设置图形的尺寸  
 plt.matshow(cm, cmap=plt.cm.Blues, fignum=1) # fignum 参数指定要操作的图形编号  
 plt.colorbar()  
 for x in range(len(cm)):  
 for y in range(len(cm)):  
 plt.annotate(cm[x, y], xy=(y, x), horizontalalignment='center', verticalalignment='center')  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')  
 plt.savefig(filename) # 保存为 JPG 文件  
 plt.show()  
  
import re  
  
def extract\_number(text):  
 *"""提取数字"""* digits = re.findall(r'\d+\.\*\d\*', str(text))  
 if digits:  
 return float(digits[0])  
 return pd.NA  
  
def process\_data(data):  
 *"""数据预处理"""* # 选择需要的列  
 data = data[['建造年份', '房价', '在售房数', '建筑材质','区划','区域']].copy()  
  
 # 对'建造年份'列进行处理，提取年份作为数值特征  
 data.loc[:, '建造年份'] = data['建造年份'].str.extract('(\d{4})').astype(float)  
  
 # 对'房价'列进行处理，提取数字部分并转换为数值特征  
 data.loc[:, '房价'] = data['房价'].apply(extract\_number)  
  
 # 处理'在售房数'列，提取数字部分并转换为数值特征  
 data.loc[:, '在售房数'] = data['在售房数'].apply(extract\_number)  
  
 # 将非数值特征编码为数值  
 label\_encoder = LabelEncoder()  
 data.loc[:, '建筑材质'] = label\_encoder.fit\_transform(data['建筑材质'])  
 data.loc[:, '区划'] = label\_encoder.fit\_transform(data['区划'])  
 data.loc[:, '区域'] = label\_encoder.fit\_transform(data['区域'])  
  
 # 用每列的中位数填充缺失值  
 data.fillna(data.median(), inplace=True)  
  
 return data  
  
  
def main():  
 *"""主函数"""* data = pd.read\_excel(r'1-北京小区数据.xlsx')  
  
 # 数据预处理  
 processed\_data = process\_data(data)  
  
 # 将数据集拆分为特征集和目标变量  
 X\_whole = processed\_data.drop('区划', axis=1)  
 y\_whole = processed\_data['区划']  
  
 # 将数据集拆分为训练集和测试集  
 x\_train\_w, x\_test\_w, y\_train\_w, y\_test\_w = train\_test\_split(X\_whole, y\_whole, test\_size=0.2, random\_state=0)  
  
 # 使用K近邻分类器进行模型训练  
 knn = KNeighborsClassifier(n\_neighbors=10)  
 knn.fit(x\_train\_w, y\_train\_w)  
  
 # 预测训练集并输出分类报告和混淆矩阵  
 train\_predicted = knn.predict(x\_train\_w)  
 train\_report = classification\_report(y\_train\_w, train\_predicted, output\_dict=True)  
 train\_cm = confusion\_matrix(y\_train\_w, train\_predicted)  
 print("Train Classification Report:")  
 print(train\_report)  
  
 # 保存训练集的分类报告和混淆矩阵  
 pd.DataFrame(train\_report).transpose().to\_csv('9-knn训练集分类报告.csv')  
 plot\_confusion\_matrix(y\_train\_w, train\_predicted, '9-knn训练集混淆矩阵.jpg')  
  
 # 预测测试集并输出分类报告和混淆矩阵  
 test\_predicted = knn.predict(x\_test\_w)  
 test\_report = classification\_report(y\_test\_w, test\_predicted, output\_dict=True)  
 test\_cm = confusion\_matrix(y\_test\_w, test\_predicted)  
 print("Test Classification Report:")  
 print(test\_report)  
  
  
 # 保存测试集的分类报告和混淆矩阵  
 pd.DataFrame(test\_report).transpose().to\_csv('9-knn测试集分类报告.csv')  
 plot\_confusion\_matrix(y\_test\_w, test\_predicted, '9-knn测试集混淆矩阵.jpg')  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

## 朴素贝叶斯

import pandas as pd  
from sklearn.preprocessing import LabelEncoder  
from sklearn.model\_selection import train\_test\_split  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.metrics import classification\_report, confusion\_matrix  
import matplotlib.pyplot as plt  
import re  
  
def cm\_plot(y, yp, filename):  
 *"""可视化混淆矩阵并保存为 JPG 文件"""* cm = confusion\_matrix(y, yp)  
 plt.figure(figsize=(10, 10)) # 设置图形的尺寸  
 plt.matshow(cm, cmap=plt.cm.Blues, fignum=1) # fignum 参数指定要操作的图形编号  
 plt.colorbar()  
 for x in range(len(cm)):  
 for y in range(len(cm)):  
 plt.annotate(cm[x, y], xy=(y, x), horizontalalignment='center', verticalalignment='center')  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')  
 plt.savefig(filename) # 保存为 JPG 文件  
 plt.show()  
  
def extract\_number(text):  
 *"""提取数字"""* digits = re.findall(r'\d+\.\*\d\*', str(text))  
 if digits:  
 return float(digits[0])  
 return pd.NA  
  
def process\_data(data):  
 *"""数据预处理"""* # 选择需要的列  
 data = data[['建造年份', '房价', '在售房数', '建筑材质','区划','区域']].copy()  
  
 # 对'建造年份'列进行处理，提取年份作为数值特征  
 data.loc[:, '建造年份'] = data['建造年份'].str.extract('(\d{4})').astype(float)  
  
 # 对'房价'列进行处理，提取数字部分并转换为数值特征  
 data.loc[:, '房价'] = data['房价'].apply(extract\_number)  
  
 # 处理'在售房数'列，提取数字部分并转换为数值特征  
 data.loc[:, '在售房数'] = data['在售房数'].apply(extract\_number)  
  
 # 将非数值特征编码为数值  
 label\_encoder = LabelEncoder()  
 data.loc[:, '建筑材质'] = label\_encoder.fit\_transform(data['建筑材质'])  
 data.loc[:, '区划'] = label\_encoder.fit\_transform(data['区划'])  
 data.loc[:, '区域'] = label\_encoder.fit\_transform(data['区域'])  
  
 # 用每列的中位数填充缺失值  
 data.fillna(data.median(), inplace=True)  
  
 return data  
  
def main():  
 data = pd.read\_excel('1-北京小区数据.xlsx')  
  
 # 数据预处理  
 processed\_data = process\_data(data)  
  
 X\_whole = processed\_data.drop('区划', axis=1)  
 y\_whole = processed\_data['区划']  
  
 # 将数据集拆分为训练集和测试集  
 x\_train\_w, x\_test\_w, y\_train\_w, y\_test\_w = train\_test\_split(X\_whole, y\_whole, test\_size=0.2, random\_state=0)  
  
 # 使用朴素贝叶斯分类器进行模型训练  
 classifier = MultinomialNB(alpha=1)  
 classifier.fit(x\_train\_w, y\_train\_w)  
  
 # 预测训练集并输出分类报告和混淆矩阵  
 train\_predicted = classifier.predict(x\_train\_w)  
 train\_report = classification\_report(y\_train\_w, train\_predicted, output\_dict=True)  
 print("Train Classification Report:")  
 print(train\_report)  
 cm\_plot(y\_train\_w, train\_predicted, '91-朴素贝叶斯-训练集混淆矩阵.jpg') # 保存混淆矩阵为 JPG 文件  
  
 # 保存训练集的分类报告  
 pd.DataFrame(train\_report).transpose().to\_csv('91-朴素贝叶斯-训练集分类报告.csv')  
  
 # 预测测试集并输出分类报告和混淆矩阵  
 test\_predicted = classifier.predict(x\_test\_w)  
 test\_report = classification\_report(y\_test\_w, test\_predicted, output\_dict=True)  
 print("Test Classification Report:")  
 print(test\_report)  
 cm\_plot(y\_test\_w, test\_predicted, '91-朴素贝叶斯-测试集混淆矩阵.jpg') # 保存混淆矩阵为 JPG 文件  
  
 # 保存测试集的分类报告  
 pd.DataFrame(test\_report).transpose().to\_csv('91-朴素贝叶斯-测试集分类报告.csv')  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

## 随机森林

import pandas as pd  
from sklearn.preprocessing import LabelEncoder  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import classification\_report, confusion\_matrix  
import matplotlib.pyplot as plt  
import re  
  
def cm\_plot(y, yp, filename):  
 *"""可视化混淆矩阵并保存为 JPG 文件"""* cm = confusion\_matrix(y, yp)  
 plt.figure(figsize=(10, 10)) # 设置图形的尺寸  
 plt.matshow(cm, cmap=plt.cm.Blues, fignum=1) # fignum 参数指定要操作的图形编号  
 plt.colorbar()  
 for x in range(len(cm)):  
 for y in range(len(cm)):  
 plt.annotate(cm[x, y], xy=(y, x), horizontalalignment='center', verticalalignment='center')  
 plt.ylabel('True label')  
 plt.xlabel('Predicted label')  
 plt.savefig(filename) # 保存为 JPG 文件  
 plt.show()  
  
def extract\_number(text):  
 *"""提取数字"""* digits = re.findall(r'\d+\.\*\d\*', str(text))  
 if digits:  
 return float(digits[0])  
 return pd.NA  
  
def process\_data(data):  
 *"""数据预处理"""* # 选择需要的列  
 data = data[['建造年份', '房价', '在售房数', '建筑材质','区划','区域']].copy()  
  
 # 对'建造年份'列进行处理，提取年份作为数值特征  
 data.loc[:, '建造年份'] = data['建造年份'].str.extract('(\d{4})').astype(float)  
  
 # 对'房价'列进行处理，提取数字部分并转换为数值特征  
 data.loc[:, '房价'] = data['房价'].apply(extract\_number)  
  
 # 处理'在售房数'列，提取数字部分并转换为数值特征  
 data.loc[:, '在售房数'] = data['在售房数'].apply(extract\_number)  
  
 # 将非数值特征编码为数值  
 label\_encoder = LabelEncoder()  
 data.loc[:, '建筑材质'] = label\_encoder.fit\_transform(data['建筑材质'])  
 data.loc[:, '区划'] = label\_encoder.fit\_transform(data['区划'])  
 data.loc[:, '区域'] = label\_encoder.fit\_transform(data['区域'])  
  
 # 用每列的中位数填充缺失值  
 data.fillna(data.median(), inplace=True)  
  
 return data  
  
def main():  
 *"""主函数"""* data = pd.read\_excel('1-北京小区数据.xlsx')  
  
 # 数据预处理  
 processed\_data = process\_data(data)  
  
 X\_whole = processed\_data.drop('区划', axis=1)  
 y\_whole = processed\_data['区划']  
  
 # 将数据集拆分为训练集和测试集  
 x\_train\_w, x\_test\_w, y\_train\_w, y\_test\_w = train\_test\_split(X\_whole, y\_whole, test\_size=0.2, random\_state=0)  
  
 # 使用随机森林分类器进行模型训练  
 rf = RandomForestClassifier(n\_estimators=100, max\_features=0.8, random\_state=0)  
 rf.fit(x\_train\_w, y\_train\_w)  
  
 # 预测训练集并输出分类报告和混淆矩阵  
 train\_predicted = rf.predict(x\_train\_w) # 使用随机森林模型进行预测  
 train\_report = classification\_report(y\_train\_w, train\_predicted, output\_dict=True)  
 print("Train Classification Report:")  
 print(train\_report)  
 cm\_plot(y\_train\_w, train\_predicted, '92-随机森林-训练集混淆矩阵.jpg') # 保存混淆矩阵为 JPG 文件  
  
 # 保存训练集的分类报告  
 pd.DataFrame(train\_report).transpose().to\_csv('92-随机森林-训练集分类报告.csv')  
  
 # 预测测试集并输出分类报告和混淆矩阵  
 test\_predicted = rf.predict(x\_test\_w) # 使用随机森林模型进行预测  
 test\_report = classification\_report(y\_test\_w, test\_predicted, output\_dict=True)  
 print("Test Classification Report:")  
 print(test\_report)  
 cm\_plot(y\_test\_w, test\_predicted, '92-随机森林-测试集混淆矩阵.jpg') # 保存混淆矩阵为 JPG 文件  
  
 # 保存测试集的分类报告  
 pd.DataFrame(test\_report).transpose().to\_csv('92-随机森林-测试集分类报告.csv')  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()