北京邮电大学
2023--2024 学年第 1 学期实验报告

课程名称： 人工智能原理

实验名称： 文本分类

实验完成人：

姓名： 陈朴炎 **学号：** 2021211138

日　　期： 2023 年 12 月

目录

文本数据的分类与分析

1 实验内容

1.1 实验目的

1．掌握数据预处理的方法，对训练集数据进行预处理；

2．掌握文本建模的方法，对语料库的文档进行建模；

3．掌握分类算法的原理，基于有监督的机器学习方法，训练文本分类器；

4．利用学习的文本分类器，对未知文本进行分类判别；

5．掌握评价分类器性能的评估方法。

1.2 实验类型

数据挖掘算法的设计与编程实现

1.3 实验要求

1．文本类别数：10 类；

2．训练集文档数：>=50000 篇；每类平均 5000 篇。

3．测试集文档数：>=50000 篇；每类平均 5000 篇。

4．分组完成实验，组员数量<=3，个人实现可以获得实验加分。

1.4 实验内容

利用分类算法实现对文本的数据挖掘，主要包括：

1．语料库的构建，主要包括利用爬虫收集 Web 文档等；

2．语料库的数据预处理，包括文档建模，如去噪，分词，建立数据字典，

使用词袋模型或主题模型表达文档等；

注：使用主题模型，如 LDA 可以获得实验加分；

3．选择分类算法（朴素贝叶斯/SVM/其他等），训练文本分类器，理解所选的分类算法的建模原理、实现过程和相关参数的含义；

4．对测试集的文本进行分类

5．对测试集的分类结果利用正确率和召回率进行分析评价：计算每类正确率、召回率，计算总体正确率和召回率，以及 F-score。

1.5 实验验收

1．编写实验报告，实验报告内容必须包括对每个阶段的过程描述，以及实验结果的截图展示。

2．以线上方式验收实验。

2 分工说明

本次实验一个人独立完成。

实验过程大致分为：

1．文档爬取

2．类型选择

3．文档分词、去停用词

4．构建 TF-IDF 矩阵空间 / 构建 LDA 主题模型

5．使用 SVM 训练文本分类器并预测评估

共收集一百三十多万个文本数据。筛选、分词后分出十类，每一类的训练集

数量为 8000 篇，每一类的测试集数量介于 5000 到 15000 之间。

3 实验环境

windows11 操作系统

python 版本：3.10.7 64-bit

数据集收集：实现爬虫爬取数据

分词工具：jieba 0.42.1

scikit-learn 版本：1.3.0

TF-IDF 工具：

sklearn.feature_extraction.text 中的 TfidfVectorizer

LDA 主题模型：

sklearn.decomposition 中的 LatentDirichletAllocation

预测工具：

sklearn.metrics 中的 accuracy_score、classification_report、confusion_matrix

VScode 版本：1.85.1

4 实验设计及准备

4.1 分词

分词我通过 jieba 库来实现。

jieba 的算法如下：

1. 基于前缀词典实现高效的词图扫描，生成句子中汉字所有可能成词情况所构成的有向无环图（DAG）

2．采用了动态规划查找最大概率路径，找出基于词频的最大切分组合

3．对于未登录词，采用了基于汉字成词能力的 HMM 模型，使用了 Viterbi 算法

jieba 的主要功能如下：

jieba.cut 方法接受四个输入参数：需要分词的字符串；cut_all 参数用来控制是否采用全模式；HMM 参数用来控制是否使用 HMM 模型；use_paddle 参数用来控制是否使用 paddle 模式下的分词模式，paddle 模式采用延迟加载方式，通过 enable_paddle 接口安装 paddlepaddle-tiny，并且 import 相关代码；

jieba.cut_for_search 方法接受两个参数：需要分词的字符串；是否使用 HMM 模型。该方法适合用于搜索引擎构建倒排索引的分词，粒度比较细

待分词的字符串可以是 unicode 或 UTF-8 字符串、GBK 字符串。注意：不建议直接输入 GBK 字符串，可能无法预料地错误解码成 UTF-8

jieba.cut 以及 jieba.cut_for_search 返回的结构都是一个可迭代的 generator，可以使用 for 循环来获得分词后得到的每一个词语(unicode)，或者用

jieba.lcut 以及 jieba.lcut_for_search 直接返回 list

jieba.Tokenizer(dictionary=DEFAULT_DICT) 新建自定义分词器，可用于同时使用不同词典。jieba.dt 为默认分词器，所有全局分词相关函数都是该分词器的映射。

jieba 的词性名称如下：

paddle模式词性和专名类别标签集合如下表，其中词性标签 24 个（小写字母），专名类别标签 4 个（大写字母）。

标签	含义	标签	含义	标签	含义	标签	含义
n	普通名词	f	方位名词	s	处所名词	t	时间
nr	人名	ns	地名	nt	机构名	nw	作品名
nz	其他专名	v	普通动词	vd	动副词	vn	名动词
a	形容词	ad	副形词	an	名形词	d	副词
m	数量词	q	量词	r	代词	p	介词
c	连词	u	助词	xc	其他虚词	w	标点符号
PER	人名	LOC	地名	ORG	机构名	TIME	时间

图 4-1 jieba 词性标号示意图

4.2 TF-IDF

4.2.1 TF-IDF 原理

TF-IDF 有两层意思：一层是词频“Term Frequency”缩写为 TF，另一层是逆文档频率“Inverse Document Frequency”缩写为 IDF。

词频指的是某一个给定的词语在该文档中出现的频率。这个数字是对词数的标准化，以防止它偏向长的文档。

逆向档案频率是一个词语普遍重要性的度量。某一特定词语的 idf，可以由总文档数目除以包含该词语之档案的数目，再将得到的商取以 2 为底的对数得到

$$\mathrm{idf_i} = \lg \frac{|D|}{|\{j : t_i \in d_j\}|}$$

|D|为语料库中的文档总数；$|\{j:t_i \in d_j\}|$是包含词语 t_i 的文档总数目，如果词语不在语料库中，就会导致分母为零，因此一般情况下会在分母 + 1。

字词的重要性随着它在档案中出现的次数成正比增加，但同时会随着它在语

料库中出现的频率成反比下降

当有 TF 和 IDF 后，将这两个词相乘，就能得到一个词的 TF-IDF 的值。某个词在文章中的TF-IDF越大,那么一般而言这个词在这篇文章的重要性会越高，所以通过计算文章中各个词的 TF-IDF，由大到小排序，排在最前面的几个词，就是该文章的关键词。

可以看到，TF-IDF 与一个词在文档中的出现次数成正比，与该词在整个语言中的出现次数成反比。所以，自动提取关键词的算法就很清楚了，就是计算出文档的每个词的 TF-IDF 值，然后按降序排列，取排在最前面的几个词。

4.2.2 TF-IDF 算法步骤

步骤 1．计算词频：

TF = 某个词在文章中的出现次数

考虑到文字有长短之分，为了便于不同文章比较，进行“词频”标准化

TF = 某个词在文章中的出现次数 ÷ 文章总次数

步骤 2．计算逆文档频率

需要一个语料库，来模拟语言的使用环境

$$\mathrm{idf_i} = \lg \frac{|D|}{|\{j : t_i \in d_j\}|}$$

步骤 3．计算 TF-IDF

TF-IDF = TF × IDF

4.3 LDA

LDA 主题模型主要用于推测文档的主题分布，可以将文档集中每篇文档的主题以概率分布的形式给出根据主题进行主题聚类或文本分类。

LDA 主题模型不关心文档中单词的顺序，通常使用词袋特征（bag-of-word feature）来代表文档。

LDA 模型认为主题可以由一个词汇分布来表示，而文章可以由主题分布来表示。比如有两个主题，美食和美妆。LDA 说两个主题可以由词汇分布表示，他们分别是：

{面包：0.4，火锅：0.5，眉笔：0.03，腮红：0.07}

{眉笔：0.4，腮红：0.5，面包：0.03，火锅：0.07}

同样，对于两篇文章，LDA 认为文章可以由主题分布这么表示：

《美妆日记》{美妆：0.8，美食：0.1，其他：0.1}

《美食探索》{美食：0.8，美妆：0.1，其他：0.1}

所以想要生成一篇文章，可以先以一定的概率选取上述某个主题，再以一定的概率选取那个主题下的某个单词，不断重复这两步就可以生成最终文章。

在 LDA 模型中，一篇文档生成的方式如下：

1．从迪利克雷分布α中取样生成文档 i 的主题分布θ_i

2．从主题的多项式分布θ_i中取样生成文档 i 的第 j 个词的主题 z_i

3．从迪利克雷分布β中取样生成主题 $z_{i,j}$ 对应的词语分布$\Phi z_{i,j}$

4．从词语的多项式分布$\Phi_{z_{i,j}}$中采样生成最终词语 $w_{i,j}$

4.4 SVM

4.4.1 SVM 特点

所谓 VC 维是对函数类的一种度量，可以简单的理解为问题的复杂程度，VC 维越高，一个问题就越复杂。

正是因为 SVM 关注的是 VC 维，后 SVM 解决问题的时候，与样本的维数是无关的（甚至样本是上万维的都可以，这使得 SVM 很适合用来解决文本分类的问题，当然，有这样的能力也因为引入了核函数）

高维模式识别是指样本维数很高，例如文本的向量表示，如果没有经过降维处理，出现几万维的情况很正常，其他算法基本就没有能力应付了，SVM 却可以，主要是因为 SVM 产生的分类器很简洁，用到的样本信息很少（仅仅用到那些称之为“支持向量”的样本），使得即使样本维数很高，也不会给存储和计算带来大麻烦（相对照而言，kNN 算法在分类时就要用到所有样本，样本数巨大，每个样本维数再一高，训练难度非常大）。

4.4.2 SVM 基础知识

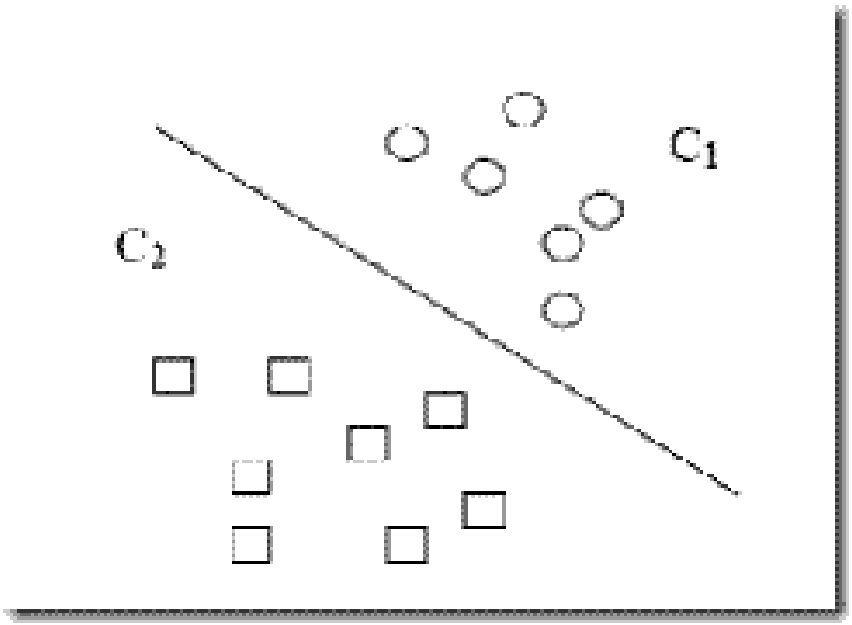

图 4-2 线性分类器示意图

C1 和 C2 是要区分的两个类别，在二维平面中它们的样本如上图所示。中间的直线就是一个分类函数，它可以将两类样本完全分开。

一般的，如果一个线性函数能够将样本完全正确的分开，就称这些数据是线性可分的，否则称为非线性可分的。

可以这样想：在一维空间里就是一个点，在二维空间里就是一条直线，三维空间里就是一个平面，可以如此想象下去，如果不关注空间的维数，这种线性函数还有一个统一的名称—超平面（Hyper Plane）

分类间隔：

训练样本 Di=(xi,yi)：

xi 就是文本向量（维数很高），yi 就是分类标记。

在二元的线性分类中，这个表示分类的标记只有两个值，1 和-1（用来

表示属于还是不属于这个类)。

样本点到某个超平面的间隔： δi=yi(wxi+b) = |wxi+b|=|g(xi)|

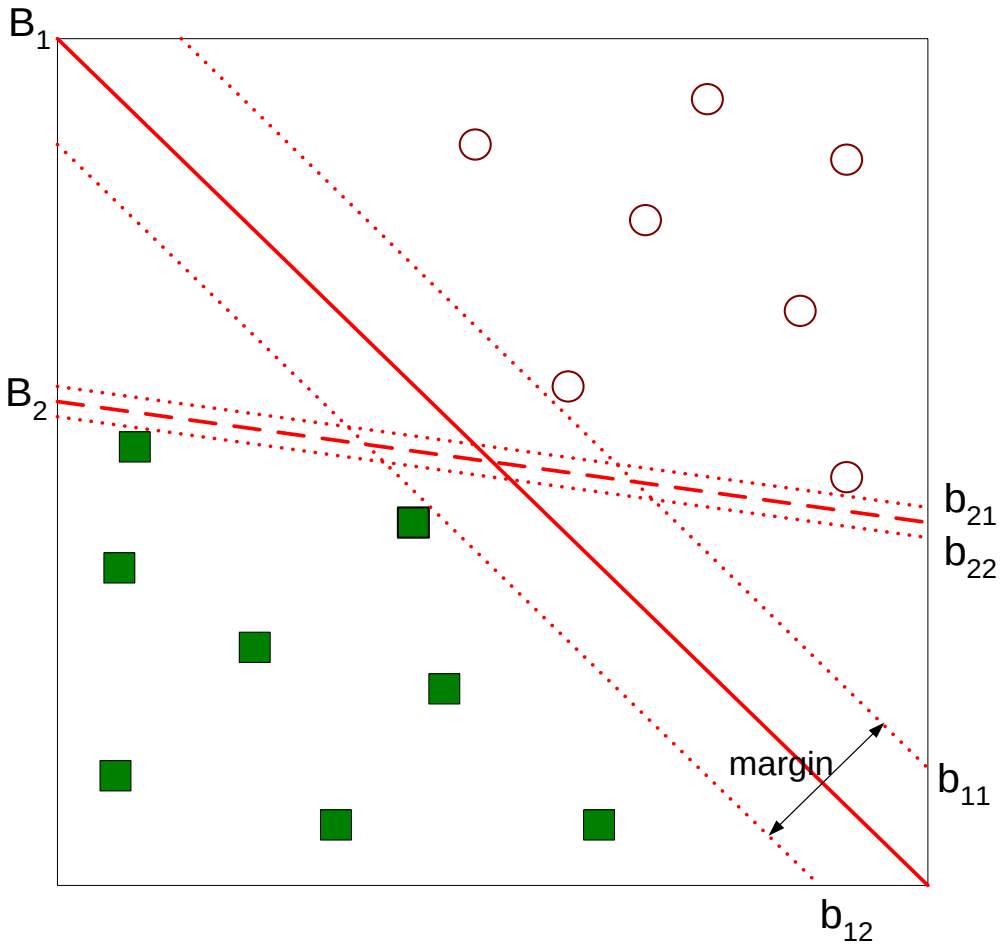

图 4-3 分类间隔示意图

一个点的集合（就是一组样本）到某个超平面的距离为此集合中离超平面最近的点的距离。

误分次数：

误分次数：代表分类器的误差。误分次数的上界由几何间隔决定!

几何间隔 δ 越大的解，它的误差上界越小。

因此最大化几何间隔成了我们训练阶段的目标。

误分次数 <= (2R/δ)^2

δ是样本集合到分类面的间隔

R 是所有样本向量长度的最长值，代表着样本分布的范围有多广

松弛变量：

并非所有的样本点都有一个松弛变量与其对应。

实际上只有“离群点”才有，所有没离群的点松弛变量都等于 0

松弛变量的值实际上标示出了对应的点到底离群有多远，值越大，点就越远。

惩罚因子 C 决定了你有多重视离群点带来的损失，显然当所有离群点的松弛变量的和一定时，你定的 C 越大，对目标函数的损失也越大，此时就暗示着你非常不愿意放弃这些离群点，最极端的情况是你把 C 定为无限大，这样只要稍有一个点离群，目标函数的值马上变成无限大，马上让问题变成无解，这就退化成了硬间隔问题。

惩罚因子 C 不是一个变量，整个优化问题在解的时候，C 是一个你必须事先指定的值；

指定这个值以后，得到一个分类器，然后用测试数据看看结果怎么样，如果不够好，换一个 C 的值，再解一次优化问题，得到另一个分类器，再看看效果，如此就是一个参数寻优的过程，

这与优化问题本身决不是一回事，优化问题在解的过程中，C 一直是定值。优化问题求解的过程：

就是先试着确定一下 w，也就是确定了前面图中的三条直线，这时看看间隔有多大，又有多少点离群，把目标函数的值算一算，再换一组三条直线

分类的直线位置如果移动了，有些原来离群的点会变得不再离群，而有的本来不离群的点会变成离群点)，再把目标函数的值算一算，如此往复（迭代)，直到最终找到目标函数最小时的 w。

惩罚因子 C

C 所起的作用：表征你有多么重视离群点，C 越大越重视，越不想丢掉它们

没有任何规定说必须对所有的松弛变量都使用同一个惩罚因子，我们完全可以给每一个离群点都使用不同的 C，这时就意味着你对每个样本的重视程度都不一样

使用惩罚因子可以处理数据偏斜问题：给样本数量少的负类更大的惩罚因子，表示我们重视这部分样本

5 实验过程

5.1 文本数据收集

5.1.1 收集 url

本次的实验数据我是在中国新闻网上收集的。

在即时新闻页面上可以查看往日的所有新闻，即使新闻链接为：

https://www.chinanews.com.cn/scroll-news/news1.html

页面示意图如下：

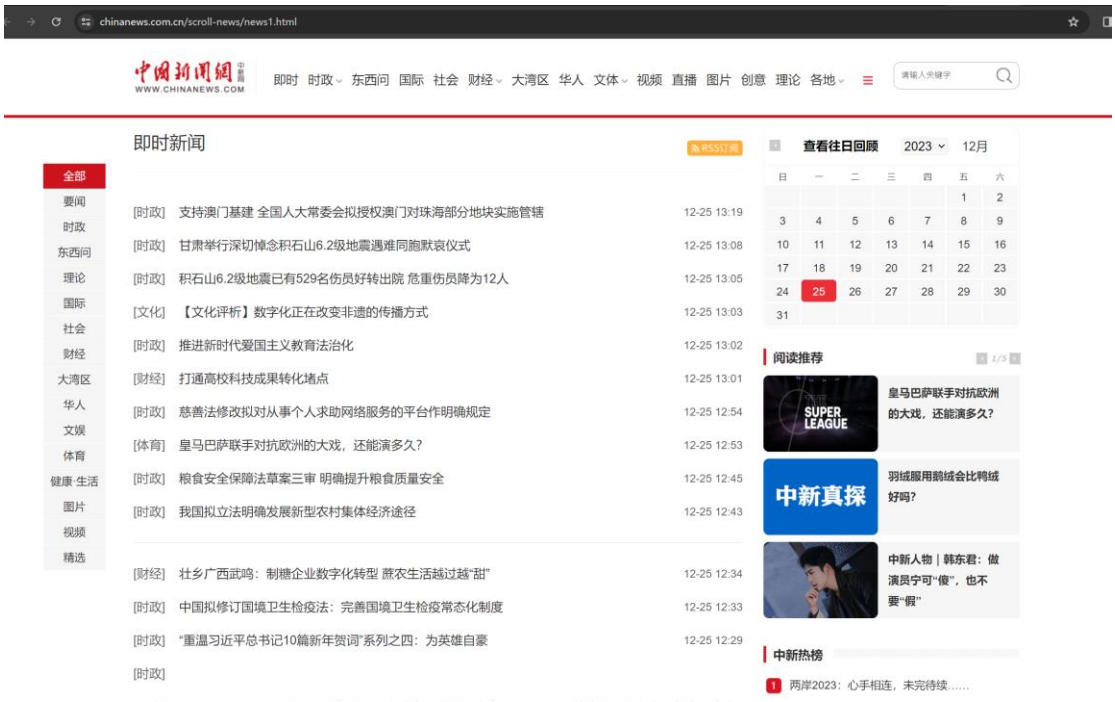

图 5-1 中国新闻网即时新闻示意图

我们可以发现 url 为 https://www.chinanews.com.cn/scroll-news/2023/1214/news.shtml

其中，https://www.chinanews.com.cn/ 为中国新闻网的基础域名，/scroll-news/ 表示是滚动新闻页面，2023 是年份，1214 是月份和日期。

打开网页的 Elements，我们查看这个网页的 html 格式，可以看到，左端的 url 都保存在<content-left>/<content_list>下的<url>里面。

因此，可以通过爬取这个前端元素来获取我们所需要的新闻的 url。

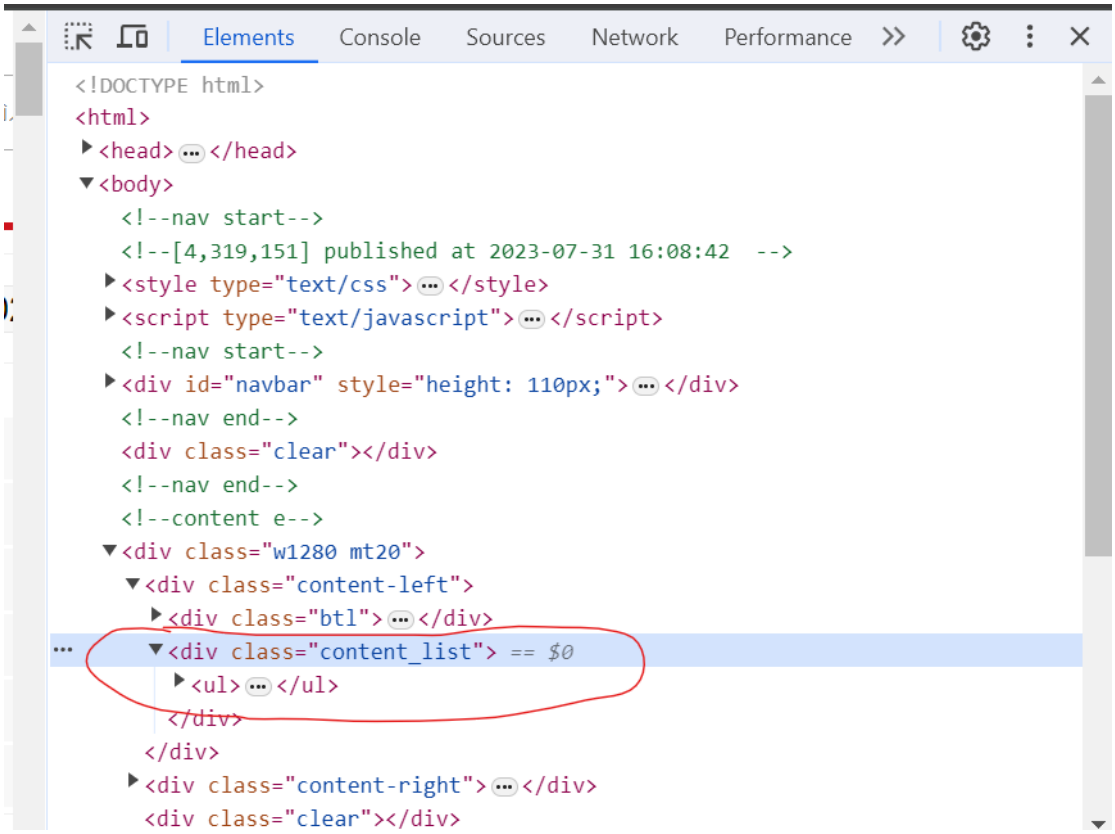

图 5-2 网页页面元素示意图

通过以上信息，我们可以使用 python 的 bs4、urllib 和 request 来爬取中国新闻网滚动新闻的 url 信息，主要操作步骤如下：

1．设置好基础的 url 部分，以及年月日列表

2．通过 request 里的 get 函数来到达某一日滚动新闻页面

3．通过 html parser 解析 html 结构，找到 content_list 和 url 列表

4．将 url 爬取下来，放到列表中，写入到文件中

程序如下所示：

```python
import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin

def crawl_news_urls(base_url, year, month, day):
    url = f"{base_url}/{year}/{month:02d}{day:02d}/news.shtml"
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    news_urls = []
    for li in soup.select('.content_list ul li'):
        div_bt = li.find('div', class_='dd_bt')
        if div_bt:
            link = div_bt.a
            if link:
                news_url = urljoin(url, link['href'])
                news_urls.append(news_url)

    return news_urls

# 主程序
base_url = "http://www.chinanews.com/scroll-news"

years = [2011,2012, 2013, 2014, 2015, 2016, 2017,2018,2019,2020,2021,2022, 2023]
months = [1, 2, 3,4,5,6,7,8,9,10,11,12]
days = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28]

# 爬取新闻页面的 URL
with open('./urls.txt', 'a', encoding='utf8') as file:
    for year in years:
        for month in months:
            for day in days:
                news_urls = crawl_news_urls(base_url, year, month, day)
                print(news_urls)
                for news_url in news_urls:
                    file.write(news_url + '\n')
```

图 5-3 收集 url 程序示意图

```
4059602    http://www.chinanews.com/sh/2023/12-18/10130359.shtml
4059603    http://www.chinanews.com/sh/2023/12-18/10130358.shtml
4059604    http://www.chinanews.com/sh/2023/12-18/10130357.shtml
4059605    http://www.chinanews.com/sh/2023/12-18/10130356.shtml
4059606    http://www.chinanews.com/gj/2023/12-18/10130355.shtml
4059607    http://www.chinanews.com/sh/2023/12-18/10130348.shtml
4059608    http://www.chinanews.com/sh/2023/12-18/10130349.shtml
4059609    http://www.chinanews.com/cj/2023/12-18/10130350.shtml
4059610    http://www.chinanews.com/sh/2023/12-18/10130351.shtml
4059611    http://www.chinanews.com/cul/2023/12-18/10130352.shtml
4059612    http://www.chinanews.com/cj/2023/12-18/10130353.shtml
4059613    http://www.chinanews.com/cj/2023/12-18/10130354.shtml
4059614    http://www.chinanews.com/cj/2023/12-18/10130347.shtml
4059615    http://www.chinanews.com/gj/2023/12-18/10130346.shtml
4059616    http://www.chinanews.com/sh/2023/12-18/10130345.shtml
4059617    http://www.chinanews.com/gj/2023/12-18/10130344.shtml
4059618    http://www.chinanews.com/ty/2023/12-18/10130330.shtml
4059619    http://www.chinanews.com/dwq/2023/12-18/10130343.shtml
4059620    http://www.chinanews.com/sh/2023/12-18/10130342.shtml
4059621    http://www.chinanews.com/cj/2023/12-18/10130341.shtml
```

图 5-4 url 收集结果图

如上图所示，我从 2011 到 2023 年收集到了共 4059621，四百万余条数据。

5.1.2 收集文本

打开收集到的某个 url，里面的网页结构如图所示：

图 5-5 新闻结构示意图

可以看到，新闻的主体内容是在<body>/<main_content w1280>/<con_left>/<content_maincontent_content>下的<left_zw>

因此我们可以通过爬取 left_zw 里<p>中的文字信息来获取新闻内容。

具体步骤如下：

1. 打开 urls 文件，获取 urls 列表

2. 通过 request.get 获取网页结构，并通过 html parser 解析

3. 查找 left_zw 类，将<p>中的文本扒下来

4. 保存到文件中

经过本人试验，url 中有 hd 的为无效 url，有 bbs 的也为无效数据。同时要注意，2019 年之前的数据是 GB2312 编码的，2019 年之后的文本数据是 UTF-8 编码的，在爬取中得注意鉴别。

并且只有一个线程跑的话爬取一天都只能爬三万多条文本数据。因此我打算开多个线程。这里我开了 400 个线程（实际效果并不是单线程的 400 倍），同时爬取文本数据。

```python
if __name__ == "__main__":
    urls = open('./data/urls/urls.txt').read().split('\n')
    base_count = 36240

    urls = urls[base_count:]
    chunk_size = 10000
    finished_num = 5000
    threads = []
    try:
        for i in range(base_count, len(urls), chunk_size):
            base = i + finished_num
            thread = threading.Thread(target=fetch_and_save, args=(urls[base:i+chunk_size], base))
            threads.append(thread)
            thread.start()
            print("开启线程: {}".format(i))

        # 等待所有线程完成
        for thread in threads:
            thread.join()
    except Exception as e:
        print("出现问题:{}".format(e))
```

图 5-6 多线程爬取文本数据

```python
def fetch_and_save(urls, count):
    try:
        for url in urls:
            if not url.startswith("http://www.chinanews.com/"):
                continue
            parsed_url = urlsplit(url)
            # base_url = f"{parsed_url.scheme}://{parsed_url.netloc}"
            path_parts = parsed_url.path.split('/')
            year = path_parts[2]
            if not year.isdigit() :
                continue
            type = path_parts[1]
            count += 1
            if type not in type_list:
                continue
            filepath = f'./data/news/{type}/'
            if not os.path.exists(filepath):
                os.makedirs(filepath)
            filename = f'{filepath}{str(count)}.txt'
            if os.path.exists(filename):
                continue

```

图 5-7 爬取文本数据示意图 1

```
            # 开始获取文本信息
            try:
                response = requests.get(url, timeout=5)
            except requests.Timeout:
                print("{} timeout".format(count))
                continue
            except requests.exceptions.RequestException as e:
                print(f"请求发生错误: {e}")

            response.encoding = 'gb2312' if int(year) < 2019  else 'utf-8'
            soup = BeautifulSoup(response.text, 'html.parser')

            # 查找 left_zw 类（用来保存新闻内容的）
            left_zw = soup.find('div', class_='left_zw')
            if left_zw:
                # 提取文本内容
                news_contents = ''
                paragraphs = left_zw.find_all('p')
                for paragraph in paragraphs:
                    news_contents += paragraph.text.strip()

                # 保存到文件
                with open(filename, 'w', encoding='utf-8', errors='ignore') as f:
                    f.write(news_contents)
                print("完成: {} 的读取".format(count))
    except  Exception as e:
        print(f"{count} 发生错误: {e}")
```

图 5-8 爬取文本数据示意图 2

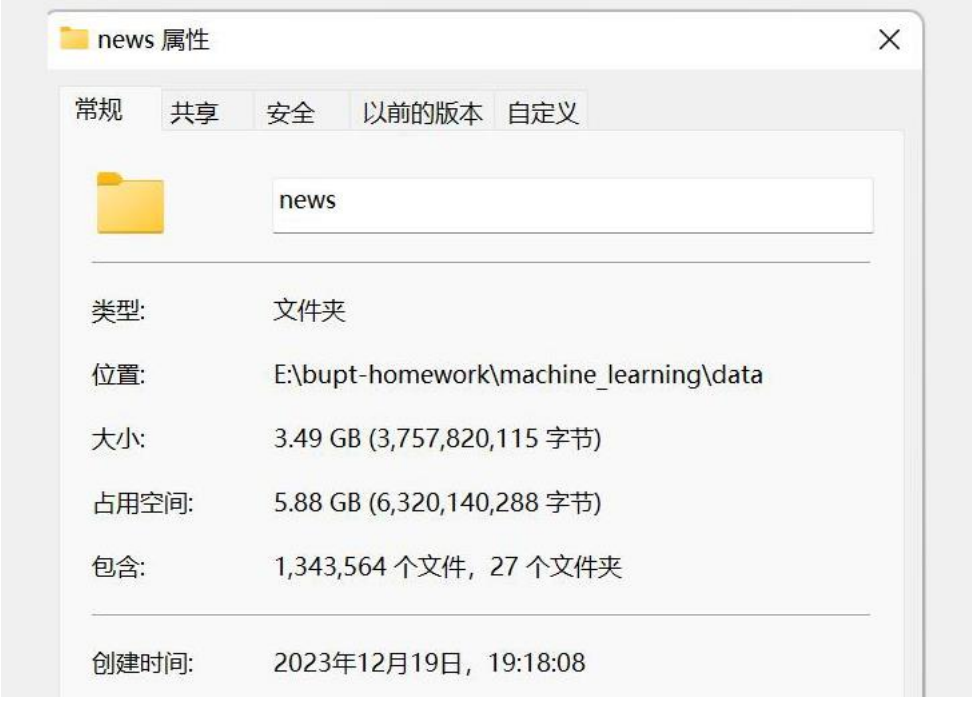

图 5-9 爬取文本数据结果示意图

在爬取完大一百三十多万条数据之后 (此时已经花费两天时间)，我统计了各个类型文本数据的分布，文本数量较多的类型和数量对应如下：

```
auto 表示汽车，有 39615 个文本
business 企业 有 11292 个文本
cj 有 121708 个文本 财经
cul 有 58619 个文本，文娱
edu 有 19205 个文本，代表教育
fortune 25852 也是财经金融
fz 47607 法制
gj 128062 国际
gn 150657 国内
house 26141 房产
hr 27305 华人
mil 28976 军事
ny 22734 能源
sh 203571 社会
stock 34326 证券
tw 36639 台湾
ty 87245 体育
yl 70816 娱乐
```

我选择了比较有代表性的几类，如下

1. 汽车--auto

2. 企业--business

3. 财经--cj

4. 文娱--cul

5. 教育--edu

6. 法制--fz

7. 房产--house

8. 军事--mil

9. 能源--ny

10. 体育--ty

11．娱乐--yl

其中，由于财经数量太多了，并且其中内容和其他类型的数据重合度较高，因此我将它剔除，剩下最后 10 类，共 561724，平均一类有五万多篇。

在本次实验中，为了能够更准确的划分每一类型的数据，需要用到的词性是名词，且不为人名。

分词的步骤如下：

1．读取文件到列表中

2．对于每一个文件，使用 cut 函数进行分词

3．给分词后的每个词后加空格

4．判断是否是名词且不是人名，且不在停用词集合中

5.组合成一个字符串，写入到新的文件中

在分词中，为了提高效率，我给每一类别的数据都开了一个线程，最后还是用了几十分钟才分词完毕，程序示意图如下：

```
def process_text(doc, stop_words):
    words = pseg.cut(doc)
    word_str = ''
    first = True
    for word in words:
        now_word = str(word.word)
        # 判断词是否是名词，且不是人名，同时不在停用词表中
        if 'n' in word.flag and word.flag != 'nr' and is_all_chinese(now_word) and now_word not in stop_words:
            if first:
                first = False
            else:
                word_str += ' '
            word_str += word.word
    return word_str
```

图 5-10 分词核心函数处理示意图

```python
if __name__ == "__main__":
    type_list = ["auto", "business", "cj", "cul", "edu", "fz", "house", "mil", "ny", "ty", "yl"]
    stop_word_path = "./data/stop/stop_words_ch.txt"

    # 读取停用词表
    with open(stop_word_path, 'r', encoding='gbk') as stop_word_file:
        stop_words = set(stop_word_file.read().split('\n'))

    base_dir = './data/news/'
    threads = []
    try:
        for category in type_list:
            thread = threading.Thread(target=process_category, args=[base_dir, category, stop_words])
            threads.append(thread)
            thread.start()

        for thread in threads:
            thread.join()
    except Exception as e:
        print("出现问题:{}".format(e))
```

图 5-11 开启多个线程分词处理示意图

```python
def process_category(base_dir, category, stop_words):
    dir_path = os.path.join(base_dir, category)
    count = 0
    out_dir = "./data/splice/" + category
    os.makedirs(out_dir, exist_ok=True)  # 确保文件夹存在，如果不存在则创建
    for _, _, files in os.walk(dir_path):
        for f in files:
            out_path = os.path.join('./data/splice/' + category, f)
            if os.path.exists(out_path):
                print(f"文件 {out_path} 已存在，跳过处理。")
                continue

            file_path = os.path.join(dir_path, f)
            with open(file_path, 'rb') as file_obj:
                doc = file_obj.read()
                count += 1

            word_str = process_text(doc, stop_words)

            with open(out_path, 'w', encoding='utf-8') as out:
                out.write(word_str)
            if(count % 100 == 0):
                print("{} : {}".format(category, count))
    print(count)
```

图 5-12 每一类别的分词处理示意图

5.2.2 提取特征数据

由于每篇文章的词数区别很大，在后面提取 TF-IDF 矩阵的时候会出现很多没必要的行和列，所以我将词数限制在 150 到 500 个词。

相关的逻辑简单，直接放程序：

```python
import os

type_list = ["auto", "business", "cul", "edu", "fz", "house", "mil", "ny", "ty", "yl"]
data_dir = "./data/splice"

def count_words(text):
    words = text.split()
    return len(words)

def select_data():
    for t in type_list:
        dir_path = os.path.join(data_dir, t)
        print(t)
        count = 0
        os.makedirs("./data/select/"+t, exist_ok=True)  # 确保文件夹存在，如果不存在则创建
        for root, _, files in os.walk(dir_path):
            for f in files:

                file_path = os.path.join(root, f)
                with open(file_path, 'r', encoding='utf-8') as file:
                    content = file.read()
                word_count = count_words(content)
                if word_count >= 150 and word_count <=500:
                    count += 1
                    out_path = os.path.join('./data/select/' + t, f)
                    with open(out_path, 'w', encoding='utf-8') as out:
                        out.write(content)
        print(count)

if __name__ == "__main__":
    select_data()
```

图 5-13 提取词数相近的文本

5.3 构建 TF-IDF 矩阵空间

TF-IDF 矩阵的向量通过 Sklearn 中的 TfidfVectorizer 构建。构造器的参数如下：

max_features：这个参数指定了构建的词汇表（vocabulary）中最大特征的数量。如果指定为一个整数，则选择出现频率最高的前 N 个特征；如果是一个浮点数，则表示选择前面的百分之多少的特征。

sublinear_tf：这个参数控制 TF（Term Frequency）的缩放。如果设置为 True，则使用 sublinear scaling，即用 1 + log(TF) 替代原始的 TF。

max_df： 这个参数是一个词频的阈值。如果一个词语在文档中的出现频率超过了设定的阈值，那么它就会被忽略。可以是一个整数（表示词频），也可以是一个浮点数（表示词频占比）。

min_dif：这个参数是一个词频的下限阈值。如果一个词语在文档中的出现频率低于设定的阈值，那么它就会被忽略。

在构造矩阵空间之前，我先用 Bunch 将各个文档给封装起来，包括每个文档的词语列表、文档的类别、文档的名称，这样可以更好的进行训练。同时，我还将训练集和测试集划分开来。训练集为每类 8000 个文档，剩下的作为测试集。

```
def divide_train_test():
    train_obj = Bunch(label=[], filenames=[], contents=[])
    test_obj = Bunch(label=[], filenames=[], contents=[])
    training_num = 8000
    # 首先将所有的数据读取到训练、测试对象中
    for t in type_list:
        dir_path = os.path.join(base_dir, t)
        file_list = []
        for _, _, files in os.walk(dir_path):
            file_list.extend(files)

        random.shuffle(file_list)
        # 选8000个出来，剩下的用来测试
        train_files = file_list[:training_num]
        test_files = file_list[training_num:]

        for f in train_files:
            file_path = os.path.join(dir_path, f)
            with open(file_path, 'r', encoding="utf-8") as file:
                doc = file.read()
                train_obj.filenames.append(str(f))
                train_obj.label.append(t)
                train_obj.contents.append(str(doc))
        print("{} select to train over".format(t))
        for f in test_files:
            file_path = os.path.join(dir_path, f)
            with open(file_path, 'r', encoding="utf-8") as file:
                doc = file.read()
                test_obj.filenames.append(str(f))
                test_obj.label.append(t)
                test_obj.contents.append(str(doc))
```

图 5-14 封装文档，并划分训练集

在这之后，我将 TF-IDF 矩阵空间生成出来，只要用到的是

TfidfVectorizer 中的 fit_transform 函数。该函数用于将原始文本数据转换为 TF-IDF 矩阵。

```python
# 通过TF-IDF提取训练数据特征空间，生成N篇文档的TF-IDF向量空间
def tfidf_train_test(train_obj, train_out_path, test_obj, test_out_path):

    vec_train = TfidfVectorizer(max_features=2000,sublinear_tf=True, max_df=0.2, min_df=0.001)
    tfidf_space = Bunch(filenames=train_obj.filenames,
                        weights=[], dictionary={}, labels=train_obj.label)

    tfidf_space.weights = vec_train.fit_transform(train_obj.contents)
    tfidf_space.dictionary = vec_train.vocabulary_

    # 保存特征空间，以便后续分类器使用
    with open(train_out_path, "wb") as file:
        pickle.dump(tfidf_space, file)

    # 生成测试集的TF-IDF向量空间
    vec_test = TfidfVectorizer(sublinear_tf=True, max_df=0.3, min_df=0.001, vocabulary=tfidf_space.dictionary)
    tfidf_space.weights = vec_test.fit_transform(test_obj.contents)
    tfidf_space.labels = test_obj.label
    with open(test_out_path, 'wb') as file:
        pickle.dump(tfidf_space, file)
```

图 5-15 生成训练集和测试集的矩阵空间

要注意的是，要将测试集的词典和训练集的词典对应上，才能更好的进行预测，否则之后的预测只是对于测试集上做聚类而已。所以只需要将测试集的矩阵空间中的文档以及标签更改了就行。

5.4 LDA 主题模型构建

通过 LatentDirichletAllocation 来构造 lda 模型，通过这个类里的 fit_transform 来将 tfidf 矩阵空间中的文档转换成最终词语。程序如下：

```python
def setup_lda():
    with open("./data/lda/train_space", 'rb') as file:
        tfidf_space_train = pickle.load(file)
    with open("./data/lda/test_space", 'rb') as file:
        tfidf_space_test = pickle.load(file)
    # 构建train的lda
    lda = LatentDirichletAllocation(n_components=10, random_state=0)
    lda_train_feature = lda.fit_transform(tfidf_space_train.weights)
    # 构建test的lda
    lda_test_feature = lda.fit_transform(tfidf_space_test.weights)
    with open("./data/lda/train_lda", 'wb') as file:
        pickle.dump(lda_train_feature, file)
    with open("./data/lda/test_lda", 'wb') as file:
        pickle.dump(lda_test_feature, file)
```

图 5-16 构造 lda 主题模型

其中，LatentDirichletAllocation 构造器中，n_components 代表待分类的类别数，按照实验要求是 10，random_state 是初始的状态。

fit_transform 过程较长，需要耐心等待。

5.5 SVM

首先要将上一步 TF-IDF 的矩阵空间和 LDA 主题模型导入。

```python
def get_train_test_tfidf():
    with open("./data/lda/train_space", 'rb') as file:
        tfidf_train = pickle.load(file)

    with open("./data/lda/test_space", 'rb') as file:
        tfidf_test = pickle.load(file)
    return tfidf_train, tfidf_test
```

图 5-17 读取 TF-IDF 矩阵空间

```
type_list = [ auto ,  business ,  cul ,  edu ,  f2 ,  house
# 从文件中加载lda模型
def get_lda_model():
    with open("./data/lda/train_lda", 'rb') as file:
        lda_train = pickle.load(file)
    with open("./data/lda/test_lda", 'rb') as file:
        lda_test = pickle.load(file)

    return lda_train, lda_test
```

图 5-18 读取 LDA 主题模型

```
(0, 1311)     0.18236972137059113
(0, 40)       0.18402451150560853
(0, 845)      0.19023885299093363
(0, 819)      0.19641819104963298
(0, 1885)     0.11773395384915164
(0, 122)      0.15038366344850382
(0, 1700)     0.11124857465971863
(0, 1028)     0.16690960406768662
(0, 1710)     0.19602083362621214
:     :
(79999, 271)  0.17776860017520835
(79999, 1564) 0.18199938255863246
(79999, 1445) 0.18525574472032677
(79999, 643)  0.16691406823219174
(79999, 1294) 0.18122312644183583
(79999, 237)  0.1889712243564025
(79999, 1392) 0.18020515998704065
(79999, 1391) 0.29321113281819766
(79999, 1224) 0.17672657514090817
(79999, 185)  0.1409227874823588
(79999, 262)  0.1983250343978118
```

图 5-19 训练集权重

```
  (79999, 447)  0.11202357681191255
  (79999, 1384) 0.2423028998255706
  (79999, 1951) 0.11498468306773772
  (79999, 1247) 0.11650256131390997
  (79999, 1700) 0.10997529884068444
The dictionary size is : 2000
```

图 5-20 字典大小示意图

之后就可以对训练集训练，对测试集预测分析

```python
# 使用TF-IDF矩阵空间训练并预测
def tfidf_classify(train, test, class_weight=None):
    x_train = train.labels
    y_train = train.weights
    svc_linear = LinearSVC(C=1, tol=1e-4, loss='hinge', max_iter=750, class_weight=class_weight)
    start_train = time.time()
    svc_linear.fit(y_train,x_train)
    end_train = time.time()
    print("training time is:{}".format(end_train-start_train))

    start_test = time.time()
    predict = svc_linear.predict(test.weights)
    end_test = time.time()
    print("predicting time: {}".format(end_test-start_test))
    accuracy = accuracy_score(test_space.labels, predict)
    print(f"Accuracy: {accuracy}")
    print ('精度:{0:.3f}'.format(metrics.precision_score(test.labels, predict, average='weighted')))
    print ('召回:{0:0.3f}'.format(metrics.recall_score(test.labels, predict, average='weighted')))
    print ('f1-score:{0:.3f}'.format(metrics.f1_score(test.labels, predict, average='weighted')))

    report = classification_report(test.labels, predict)
    print(report)
    pd.set_option('display.max_columns', None)
    pd.set_option("display.max_rows", None)
    confuse_matrix = pd.DataFrame(confusion_matrix(test.labels, predict), columns=type_list, index=type_list)
    print(confuse_matrix)
```

图 5-21 使用 TF-IDF 矩阵空间训练并预测

```python
# 使用LDA主题模型训练并预测
def lda_classify(lda_train, train_space, lda_test, test_space):
    start_train = time.time()
    svm_linear = LinearSVC(C=1, tol=1e-5, loss='hinge', max_iter=2000)
    svm_linear.fit(lda_train, train_space.labels)
    end_train = time.time()
    print("training time is:{}".format(end_train-start_train))

    start_test = time.time()
    prediction = svm_linear.predict(lda_test)
    end_test = time.time()
    print("predicting time: {}".format(end_test-start_test))
    accuracy = accuracy_score(test_space.labels, prediction)
    print(f"Accuracy: {accuracy}")
    print ('精度:{0:.3f}'.format(metrics.precision_score(test_space.labels, prediction,average='weighted')))
    print ('召回:{0:0.3f}'.format(metrics.recall_score(test_space.labels, prediction,average='weighted')))
    print ('f1-score:{0:.3f}'.format(metrics.f1_score(test_space.labels, prediction,average='weighted')))

    report = classification_report(test_space.labels, prediction)
    print(report)
    pd.set_option('display.max_columns', None)
    pd.set_option("display.max_rows", None)
    confuse_matrix = pd.DataFrame(confusion_matrix(test_space.labels, prediction), columns=type_list, index=type_list)
    print(confuse_matrix)
```

图 5-22 使用 LDA 主题模型训练并预测

```
  warnings.warn(
training time is:4.794305086135864
predicting time: 0.04567980766296387
Accuracy: 0.7115838114771114
精度:0.709
召回:0.712
f1-score:0.685
              precision    recall  f1-score   support

        auto       0.78      0.68      0.73     46339
    business       0.28      0.72      0.41     20180
         cul       0.62      0.13      0.22     50790
         edu       0.67      0.88      0.76     25954
          fz       0.78      0.83      0.80     63963
       house       0.70      0.79      0.74     44290
         mil       0.82      0.91      0.86     53839
          ny       0.15      0.06      0.08     33798
          ty       0.95      0.91      0.93     79507
          yl       0.71      0.89      0.79     63064

    accuracy                           0.71    481724
   macro avg       0.65      0.68      0.63    481724
weighted avg       0.71      0.71      0.69    481724
```

图 5-23 使用 LDA 主题模型预测

```
              auto  business   cul    edu     fz  house    mil    ny     ty  \
auto         31549     10254    55    222   1528   2010    344    52    195
business      1531     14594   338   1119    561    788    425    96    278
cul           1941      2359  6751   4008   4117    880   4705  5871   1311
edu             68       364   201  22720   1169    320    306   294    188
fz             538       726  1407   2146  53123   1884    538  2963    188
house          646      4332   413    466   2436  34844    129   724    126
mil           1866       665   127    655    459    127  48792   245    352
ny             762     16521   915    277   2620   8414   2138  1896    158
ty             976       562   265    965    693    202   1227   242  72307
yl             605      1194   446   1187   1554    189    552   233    893

                yl
auto           130
business       450
cul          18847
edu            324
fz             450
house          174
mil            551
ny              97
ty            2068
yl           56211
PS E:\bupt-homework\machine_learning>
```

图 5-24 使用 LDA 主题模型的预测混淆矩阵

```
PS E:\bupt-homework\machine_learning> python -u "e:\bupt-homework\machine_learning\svm.py"
D:\Anaconda3\Lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The default value of `dual` will change from `True` to `'auto'` in 1.5. Se
t the value of `dual` explicitly to suppress the warning.
  warnings.warn(
D:\Anaconda3\Lib\site-packages\sklearn\svm\_base.py:1242: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
  warnings.warn(
training time is:3.0411155223846436
predicting time: 0.13503026962280273
Accuracy: 0.8810293861215136
精度:0.883
召回:0.881
f1-score:0.881
              precision    recall  f1-score   support

        auto       0.94      0.92      0.93     46339
    business       0.70      0.78      0.74     20180
         cul       0.83      0.76      0.79     50790
         edu       0.78      0.89      0.83     25954
          fz       0.90      0.88      0.89     63963
       house       0.90      0.88      0.89     44290
         mil       0.87      0.93      0.90     53839
          ny       0.82      0.85      0.83     33798
          ty       0.97      0.93      0.95     79507
          yl       0.89      0.89      0.89     63064

    accuracy                           0.88    481724
   macro avg       0.86      0.87      0.86    481724
weighted avg       0.88      0.88      0.88    481724
```

图 5-25 使用 TF-IDF 训练分析

```
             auto  business    cul    edu     fz  house    mil     ny     ty  \
auto        42537      1070    113    151    722    235    396    904    126
business      393     15747    457    319    152    675   1307    874    134
cul           121      1258  38491   2075    896    561   2240    661    448
edu            50       279    759  23160    814    170    178    224    142
fz            704       405    567   1876  56408   1665    388   1480    113
house         157      1157    464    379   1332  38943    321   1331    107
mil            94       357   1155    459    432     96  50174    485    244
ny           1069       827    460    270    841    839    674  28673     62
ty            143       584    734    559    383    137   1140    233  74085
yl             91       653   2940    591    927    181    765    140    581

              yl
auto          85
business     122
cul         4039
edu          178
fz           357
house         99
mil          343
ny            83
ty          1509
yl         56195
PS E:\bupt-homework\machine_learning>
```

图 5-26 使用 TF-IDF 矩阵空间预测的混淆矩阵

可以看到，单次预测的平均准确度有 88.3%。单类最高准确率有 97%。错误主要是出在 business 类上。在 class_weight 上，把 business 的权重调低，如下图所示：

```
class_weight = {'auto':1.0, 'business':0.5, "cul":0.8, "edu":0.5, "fz":1.0, "house":1.0, "mil":1.0, "ny":1.0, "ty":1.0, "yl":1.0}
lda_classify(lda_train, train_space=train_space, lda_test=lda_test, test_space=test_space)
tfidf_classify(train=train_space, test=test_space,class_weight=class_weight)
```

图 5-27 参数修改

传入 class_weight 参数，再次执行训练+分析，结果如下：

```
  warnings.warn(
training time is:2.864349126815796
predicting time: 0.12447571754455566
Accuracy: 0.8833190789746826
精度:0.884
召回:0.883
f1-score:0.883
              precision    recall  f1-score   support

        auto       0.93      0.92      0.93     46339
    business       0.76      0.76      0.76     20180
         cul       0.84      0.75      0.79     50790
         edu       0.81      0.88      0.84     25954
          fz       0.89      0.89      0.89     63963
       house       0.89      0.89      0.89     44290
         mil       0.87      0.94      0.90     53839
          ny       0.81      0.86      0.83     33798
          ty       0.97      0.94      0.95     79507
          yl       0.89      0.90      0.89     63064

    accuracy                           0.88    481724
   macro avg       0.87      0.87      0.87    481724
weighted avg       0.88      0.88      0.88    481724
```

图 5-29 调参后的执行结果

可以看到，business 类的准确率提升了 6%，整体的精确度提升了 0.1%。

6 附录

6.1 get_url.py

```
# 这个程序是用来爬取 类似 https://www.chinanews.com/scroll-
news/2019/1211/news.shtml 这个网站里的 url 的

import requests
from bs4 import BeautifulSoup
from urllib.parse import urljoin

def crawl_news_urls(base_url, year, month, day):
    url = f"{base_url}/{year}/{month:02d}{day:02d}/news.shtml"
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    news_urls = []
    for li in soup.select('.content_list ul li'):
        div_bt = li.find('div', class_='dd_bt')
        if div_bt:
            link = div_bt.a
            if link:
```

```
                news_url = urljoin(url, link['href'])
                news_urls.append(news_url)

    return news_urls

# 主程序
base_url = "http://www.chinanews.com/scroll-news"

years = [2011,2012, 2013, 2014, 2015, 2016,
2017,2018,2019,2020,2021,2022, 2023]
months = [1, 2, 3,4,5,6,7,8,9,10,11,12]
days = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20, 21, 22, 23, 24, 25, 26, 27, 28]

# 爬取新闻页面的 URL
with open('./urls.txt', 'a', encoding='utf8') as file:
    for year in years:
        for month in months:
            for day in days:
                news_urls = crawl_news_urls(base_url, year, month, day)
                print(news_urls)
                for news_url in news_urls:
                    file.write(news_url + '\n')
```

6.2get_news_treads.py

```
from urllib.parse import urlsplit
import requests
from bs4 import BeautifulSoup
import os
import threading

type_list = ["fz", "business", "edu", "mil", "ny", "house"]

def fetch_and_save(urls, count):
    try:
        for url in urls:
            if not url.startswith("http://www.chinanews.com/"):
                continue
            parsed_url = urlsplit(url)
```

```
            # base_url = f"{parsed_url.scheme}://{parsed_url.netloc}"
            path_parts = parsed_url.path.split('/')
            year = path_parts[2]
            if not year.isdigit() :
                continue
            type = path_parts[1]
            count += 1
            if type not in type_list:
                continue
            filepath = f'./data/news/{type}/'
            if not os.path.exists(filepath):
                os.makedirs(filepath)
            filename = f'{filepath}{str(count)}.txt'
            if os.path.exists(filename):
                continue

            # 开始获取文本信息
            try:
                response = requests.get(url, timeout=5)
            except requests.Timeout:
                print("{} timeout".format(count))
                continue
            except requests.exceptions.RequestException as e:
                print(f"请求发生错误: {e}")

            response.encoding = 'gb2312' if int(year) < 2019  else 'utf-
8'
            soup = BeautifulSoup(response.text, 'html.parser')

            # 查找 left_zw 类（用来保存新闻内容的）
            left_zw = soup.find('div', class_='left_zw')
            if left_zw:
                # 提取文本内容
                news_contents = ''
                paragraphs = left_zw.find_all('p')
                for paragraph in paragraphs:
                    news_contents += paragraph.text.strip()

                # 保存到文件
                with open(filename, 'w', encoding='utf-8',
errors='ignore') as f:
                    f.write(news_contents)
```

```
                print("完成：{} 的读取".format(count))
    except  Exception as e:
        print(f"{count} 发生错误：{e}")



if __name__ == "__main__":
    urls = open('./data/urls/urls.txt').read().split('\n')
    base_count = 36240

    urls = urls[base_count:]
    chunk_size = 10000
    finished_num = 5000
    threads = []
    try:
        for i in range(base_count, len(urls), chunk_size):
            base = i + finished_num
            thread = threading.Thread(target=fetch_and_save,
args=(urls[base:i+chunk_size], base))
            threads.append(thread)
            thread.start()
            print("开启线程：{}".format(i))

        # 等待所有线程完成
        for thread in threads:
            thread.join()
    except Exception as e:
        print("出现问题:{}".format(e))
```

6.3 splice_word.py

```
import os
import jieba.posseg as pseg
import threading

def is_all_chinese(strs):
    for _char in strs:
        if not ('\u4e00' <= _char <= '\u9fa5'):
            return False
    return True

def process_text(doc, stop_words):
```

```
    words = pseg.cut(doc)
    word_str = ''
    first = True
    for word in words:
        now_word = str(word.word)
        # 判断词是否是名词，且不是人名，同时不在停用词表中
        if 'n' in word.flag and word.flag != 'nr' and
is_all_chinese(now_word) and now_word not in stop_words:
            if first:
                first = False
            else:
                word_str += ' '
            word_str += word.word
    return word_str

def process_category(base_dir, category, stop_words):
    dir_path = os.path.join(base_dir, category)
    count = 0
    out_dir = "./data/splice/" + category
    os.makedirs(out_dir, exist_ok=True)  # 确保文件夹存在，如果不存在则创建
    for _, _, files in os.walk(dir_path):
        for f in files:
            out_path = os.path.join('./data/splice/' + category, f)
            if os.path.exists(out_path):
                print(f"文件 {out_path} 已存在，跳过处理。")
                continue

            file_path = os.path.join(dir_path, f)
            with open(file_path, 'rb') as file_obj:
                doc = file_obj.read()
                count += 1

            word_str = process_text(doc, stop_words)

            with open(out_path, 'w', encoding='utf-8') as out:
                out.write(word_str)
            if(count % 100 == 0):
                print("{} : {}".format(category, count))
    print(count)

if __name__ == "__main__":
```

```
    type_list = ["auto", "business", "cj", "cul", "edu", "fz", "house",
"mil", "ny", "ty", "yl"]
    stop_word_path = "./data/stop/stop_words_ch.txt"

    # 读取停用词表
    with open(stop_word_path, 'r', encoding='gbk') as stop_word_file:
        stop_words = set(stop_word_file.read().split('\n'))

    base_dir = './data/news/'
    threads = []
    try:
        for category in type_list:
            thread = threading.Thread(target=process_category,
args=[base_dir, category, stop_words])
            threads.append(thread)
            thread.start()

        for thread in threads:
            thread.join()
    except Exception as e:
        print("出现问题:{}".format(e))
```

6.4 select_data.py

```
import os

type_list = ["auto", "business", "cul", "edu", "fz", "house", "mil",
"ny", "ty", "yl"]
data_dir = "./data/splice"

def count_words(text):
    words = text.split()
    return len(words)

def select_data():
    for t in type_list:
        dir_path = os.path.join(data_dir, t)
        print(t)
        count = 0
        os.makedirs("./data/select/"+t, exist_ok=True)  # 确保文件夹存
在，如果不存在则创建
        for root, _, files in os.walk(dir_path):
```

```
            for f in files:

                file_path = os.path.join(root, f)
                with open(file_path, 'r', encoding='utf-8') as file:
                    content = file.read()
                word_count = count_words(content)
                if word_count >= 150 and word_count <=500:
                    count += 1
                    out_path = os.path.join('./data/select/' + t, f)
                    with open(out_path, 'w', encoding='utf-8') as out:
                        out.write(content)
        print(count)

if __name__ == "__main__":
    select_data()
```

6.5 tfidf.py

```
import os
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.utils import Bunch
import numpy as np
import pickle
import random

# 文件夹路径
base_dir = './data/splice/'
type_list = ["auto", "business", "cul", "edu", "fz", "house", "mil",
"ny", "ty", "yl"]
# document_counts =
[54339,28180,58790,33954,71963,52290,61839,41798,87507,71064]
total_doc = 561890
train_out_path = "./data/trainset/train"
test_out_path = "./data/testset/test"

def divide_train_test():
    train_obj = Bunch(label=[], filenames=[], contents=[])
    test_obj = Bunch(label=[], filenames=[], contents=[])
    training_num = 8000
    # 首先将所有的数据读取到训练、测试对象中
    for t in type_list:
        dir_path = os.path.join(base_dir, t)
```

```
        file_list = []
        for _, _, files in os.walk(dir_path):
            file_list.extend(files)

        random.shuffle(file_list)
        # 选 8000 个出来，剩下的用来测试
        train_files = file_list[:training_num]
        test_files = file_list[training_num:]

        for f in train_files:
            file_path = os.path.join(dir_path, f)
            with open(file_path, 'r', encoding="utf-8") as file:
                doc = file.read()
                train_obj.filenames.append(str(f))
                train_obj.label.append(t)
                train_obj.contents.append(str(doc))
        print("{} select to train over".format(t))
        for f in test_files:
            file_path = os.path.join(dir_path, f)
            with open(file_path, 'r', encoding="utf-8") as file:
                doc = file.read()
                test_obj.filenames.append(str(f))
                test_obj.label.append(t)
                test_obj.contents.append(str(doc))

        print("{} select to test over".format(t))
    # 将二进制数据写入文件
    with open(train_out_path, 'wb') as f1:
        pickle.dump(train_obj, f1)
    with open(test_out_path, "wb") as f2:
        pickle.dump(test_obj, f2)

# 通过 TF-IDF 提取训练数据特征空间，生成 N 篇文档的 TF-IDF 向量空间
def tfidf_train_test(train_obj, train_out_path, test_obj,
test_out_path):

    vec_train = TfidfVectorizer(max_features=2000,sublinear_tf=True,
max_df=0.2, min_df=0.001)
    tfidf_space = Bunch(filenames=train_obj.filenames,
                        weights=[], dictionary={},
labels=train_obj.label)
```

```
    tfidf_space.weights = vec_train.fit_transform(train_obj.contents)
    tfidf_space.dictionary = vec_train.vocabulary_

    # 保存特征空间，以便后续分类器使用
    with open(train_out_path, "wb") as file:
        pickle.dump(tfidf_space, file)

    # 生成测试集的 TF-IDF 向量空间
    vec_test = TfidfVectorizer(sublinear_tf=True, max_df=0.3,
min_df=0.001, vocabulary=tfidf_space.dictionary)
    tfidf_space.weights = vec_test.fit_transform(test_obj.contents)
    tfidf_space.labels = test_obj.label
    with open(test_out_path, 'wb') as file:
        pickle.dump(tfidf_space, file)


def setup_lda():
    with open("./data/lda/train_space", 'rb') as file:
        tfidf_space_train = pickle.load(file)
    with open("./data/lda/test_space", 'rb') as file:
        tfidf_space_test = pickle.load(file)
    # 构建 train 的 lda
    lda = LatentDirichletAllocation(n_components=10, random_state=0)
    lda_train_feature = lda.fit_transform(tfidf_space_train.weights)
    # 构建 test 的 lda
    lda_test_feature = lda.fit_transform(tfidf_space_test.weights)
    with open("./data/lda/train_lda", 'wb') as file:
        pickle.dump(lda_train_feature, file)
    with open("./data/lda/test_lda", 'wb') as file:
        pickle.dump(lda_test_feature, file)


# 读取处理好的训练集和测试集
def get_train_test():
    with open(train_out_path, 'rb') as train:
        train_obj = pickle.load(train)
    with open(test_out_path, 'rb') as test:
        test_obj = pickle.load(test)
    return train_obj, test_obj



if __name__ == "__main__":
```

```
    # divide_train_test()
    # train, test = get_train_test()
    print("get data over")
    # tfidf_train_test(train_obj=train,
train_out_path="./data/lda/train_space", test_obj=test,
test_out_path="./data/lda/test_space")
    setup_lda()


# 失败的加权模型
# 将训练集转换为 TF-IDF 矩阵，为矩阵加权，喂给 LDA 主题模型,并保存
# def lda_fit(train_obj):
#     # 计算文档频率（DF）
#     vec_train = TfidfVectorizer(max_features=2000, max_df=0.2,
token_pattern=r'\b\w+\b')
#     df_matrix = vec_train.fit_transform(train_obj.contents)
#     print(df_matrix.shape)
#     # doc_freqs = df_matrix.transform(train_obj.contents)
#     weight_array = np.array(train_obj.weights)
#     print(weight_array.shape)
#     # 将 TF-IDF 矩阵按照样本权重进行加权
#     weighted_tfidf_matrix =
df_matrix.multiply(1/np.sqrt(weight_array[:, np.newaxis]))
#     # 构建 LDA 模型并在 TF-IDF 矩阵上训练
#     lda = LatentDirichletAllocation(n_components=11, random_state=42)
#     lda.fit(weighted_tfidf_matrix)
#     with open("./data/lda/lda_model", 'wb') as lda_file:
#         pickle.dump(lda, lda_file)
#     with open("./data/lda/tfidf", "wb") as tfidf_file:
#         pickle.dump(vec_train, tfidf_file)



# 本来想直接读取，然后加权得到矩阵，但是内存分配不够
# # 存放所有文档的列表
# documents = []
# weights = []
# # 添加每个已分词文件到列表中
# for t in type_list:
#     dir_path = os.path.join(base_dir, t)
#     count = 0
#     for root, dirs, files in os.walk(dir_path):
#         for f in files:
```

```
#             path = os.path.join(root, f)
#             with open(path, 'r', encoding='utf-8') as file:
#                 # 读取文件内容
#                 documents.append(file.read())
#             weights.append(total_doc / len(files))
#             count += 1
#             if count % 10000 == 0 :
#                 print(count)
#             flag = not flag

# weights_array = np.array(weights)
# # 将文档列表转换为 TF-IDF 矩阵
# # max_df 表示如果一个单词在百分之 max_df 的文档中出现，就要省略
# vec_train = TfidfVectorizer(max_features=500, max_df=0.3,
token_pattern=r'\b\w+\b')
# tfidf_matrix = vec_train.fit_transform(documents)
# print(len(weights))
# print(tfidf_matrix.shape[0])
# weighted_tfidf_matrix = tfidf_matrix.multiply(np.repeat(weights,
tfidf_matrix.shape[0]))
# # 构建 LDA 模型
# lda = LatentDirichletAllocation(n_components=11, random_state=42)

# # 在 TF-IDF 矩阵上训练 LDA 模型
# lda.fit(weighted_tfidf_matrix)

# # 打印每个主题的前几个关键词
# feature_names = vec_train.get_feature_names_out()
# n_top_words = 10
# for topic_idx, topic in enumerate(lda.components_):
#     top_words_idx = topic.argsort()[:-n_top_words - 1:-1]
#     top_words = [feature_names[i] for i in top_words_idx]
#     print(f"Topic #{topic_idx}: {', '.join(top_words)}")
```

6.6 svm.py

```
import os
from sklearn.svm import LinearSVC
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.utils import Bunch
```

```
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import numpy as np
import pickle
import time
from sklearn import metrics
import pandas as pd
type_list = ["auto", "business", "cul", "edu", "fz", "house", "mil",
"ny", "ty", "yl"]
# 从文件中加载 lda 模型
def get_lda_model():
    with open("./data/lda/train_lda", 'rb') as file:
        lda_train = pickle.load(file)
    with open("./data/lda/test_lda", 'rb') as file:
        lda_test = pickle.load(file)

    return lda_train, lda_test

def get_train_test_tfidf():
    with open("./data/lda/train_space", 'rb') as file:
        tfidf_train = pickle.load(file)

    with open("./data/lda/test_space", 'rb') as file:
        tfidf_test = pickle.load(file)
    return tfidf_train, tfidf_test


# 读取处理好的训练集和测试集
def get_train_test():
    with open("./data/trainset/train_data", 'rb') as train:
        train_obj = pickle.load(train)
    with open("./data/testset/test_data", 'rb') as test:
        test_obj = pickle.load(test)
    return train_obj, test_obj

# 使用 LDA 主题模型训练并预测
def lda_classify(lda_train, train_space, lda_test, test_space):
    start_train = time.time()
    svm_linear = LinearSVC(C=1, tol=1e-5, loss='hinge', max_iter=2000)
    svm_linear.fit(lda_train, train_space.labels)
    end_train = time.time()
    print("training time is:{}".format(end_train-start_train))
```

```
    start_test = time.time()
    prediction = svm_linear.predict(lda_test)
    end_test = time.time()
    print("predicting time: {}".format(end_test-start_test))
    accuracy = accuracy_score(test_space.labels, prediction)
    print(f"Accuracy: {accuracy}")
    print ('精
度:{0:.3f}'.format(metrics.precision_score(test_space.labels,
prediction,average='weighted')))
    print ('召
回:{0:0.3f}'.format(metrics.recall_score(test_space.labels,
prediction,average='weighted')))
    print ('f1-score:{0:.3f}'.format(metrics.f1_score(test_space.labels,
prediction,average='weighted')))

    report = classification_report(test_space.labels, prediction)
    print(report)
    pd.set_option('display.max_columns', None)
    pd.set_option("display.max_rows", None)
    confuse_matrix = pd.DataFrame(confusion_matrix(test_space.labels,
prediction), columns=type_list, index=type_list)
    print(confuse_matrix)


# 使用 TF-IDF 矩阵空间训练并预测
def tfidf_classify(train, test, class_weight=None):
    x_train = train.labels
    y_train = train.weights
    svc_linear = LinearSVC(C=1, tol=1e-4, loss='hinge', max_iter=750,
class_weight=class_weight)
    start_train = time.time()
    svc_linear.fit(y_train,x_train)
    end_train = time.time()
    print("training time is:{}".format(end_train-start_train))

    start_test = time.time()
    predict = svc_linear.predict(test.weights)
    end_test = time.time()
    print("predicting time: {}".format(end_test-start_test))
    accuracy = accuracy_score(test_space.labels, predict)
    print(f"Accuracy: {accuracy}")
    print ('精度:{0:.3f}'.format(metrics.precision_score(test.labels,
predict, average='weighted')))
```

```
    print ('召回:{0:0.3f}'.format(metrics.recall_score(test.labels,
predict, average='weighted')))
    print ('f1-score:{0:.3f}'.format(metrics.f1_score(test.labels,
predict, average='weighted')))

    report = classification_report(test.labels, predict)
    print(report)
    pd.set_option('display.max_columns', None)
    pd.set_option("display.max_rows", None)
    confuse_matrix = pd.DataFrame(confusion_matrix(test.labels,
predict), columns=type_list, index=type_list)
    print(confuse_matrix)

if __name__ == "__main__":
    lda_train, lda_test = get_lda_model()
    train_space, test_space = get_train_test_tfidf()
    # train_obj, test_obj = get_train_test()
    # svm_classify(lda, tfidf, train_obj, test_obj)
    class_weight = {'auto':1.0, 'business':1.0, "cul":1.0, "edu":1.0,
"fz":1.0, "house":1.0, "mil":1.0, "ny":1.0, "ty":1.0, "yl":1.0}
    lda_classify(lda_train, train_space=train_space, lda_test=lda_test,
test_space=test_space)
    # tfidf_classify(train=train_space,
test=test_space,class_weight=class_weight)



# def svm_classify(lda, tfidf, train, test, loss="hinge",
weigh_param=None):
#     start_transform = time.time()
#     lda_train_feature = lda.transform(train.weights)
#     lda_test_feature = lda.transform(test.weights)
#     end_transform = time.time()
#     print(f"Time for transformation: {end_transform - start_transform}
seconds")

#     # 构造分类器
#     svm_linear = LinearSVC(class_weight=weigh_param, loss=loss)

#     # 训练 SVM
#     start_training = time.time()
#     # dense_train = lda_train_feature.toarray()
#     svm_linear.fit(lda_train_feature, train.labels)
```

```
#     end_training = time.time()
#     print(f"Time for training: {end_training - start_training}
seconds")

#     # 预测
#     start_prediction = time.time()
#     predictions = svm_linear.predict(lda_test_feature)
#     end_prediction = time.time()
#     print(f"Time for prediction: {end_prediction - start_prediction}
seconds")

#     # 评估
#     accuracy = accuracy_score(test.labels, predictions)
#     report = classification_report(test.labels, predictions)

#     print(f"Accuracy: {accuracy}")
#     print("Classification Report:")
#     print(report)
# def svm_classify(lda, tfidf, train_obj, test_obj):
#     start_transform = time.time()
#     lda_train_feature =
lda.transform(tfidf.transform(train_obj.contents))
#     lda_test_feature =
lda.transform(tfidf.transform(test_obj.contents))
#     end_transform = time.time()
#     print("transform time is : {}".format(end_transform-
start_transform))
#     # 构建 SVM 分类器
#     linear_svm = SVC(kernel='linear')
#     start_train = time.time()
#     linear_svm.fit(lda_train_feature, train_obj.labels)
#     end_train = time.time()
#     print("train time is : {}".format(end_train-start_train))
#     # 预测
#     predict_linear = linear_svm.predict(lda_test_feature)
#     # 评估
#     accuracy_linear = accuracy_score(test_obj.labels, predict_linear)
#     report_linear = classification_report(test_obj.labels,
predict_linear)
#     print(f"Accuracy: {accuracy_linear}")
#     print("Classification Report:")
#     print(report_linear)
```

```
#     print("predict time is {}".format(time.time() - end_train))


    # poly_svm = SVC(kernel="poly")
    # poly_svm.fit(lda_test_feature, test_obj.labels)
    # # 预测
    # predict_poly = poly_svm.predict(lda_test_feature)
    # # 评估
    # accuracy_poly = accuracy_score(test_obj.labels, predict_poly)
    # report_poly = classification_report(test_obj.labels, predict_poly)
    # print(f"Accuracy: {accuracy_poly}")
    # print("Classification Report:")
    # print(report_poly)
```