河海大學

《文本分析与文本挖掘》课程报告

基于豆瓣影视数据的文本挖掘

年级专业 2020 大数据管理与应用

学号姓名 2063110122　何　雷

学号姓名 2063110123　漆银权

学号姓名 2063110124　李宗霖

学号姓名 2063110220　王瑞钰

二〇二三年五月

摘要

当前电影数据海量，观众对于电影质量和内容要求也更加严格。选择一部好的影视能够带给观众视觉上的享受以及精神上的充实；但是一部差的影视却会让人颇为不满，并且感觉时间与情感都被浪费。因此，本文将从豆瓣海量的电影中通过文本挖掘电影简介之中特定的内容以及故事情节，寻找高分与低分电影情节中的差异给观众提供一个通过阅读新影视作品的简介，来简单筛选电影的依据。同时利用情感分析电影的相关评论，提取广大观众更为关注的地方并且通过在情感分析结果中核实高频词，来进一步挖掘用户对电影相关方面的评价，得到对于该电影的评价标准，从而更好地理解其受欢迎或不受欢迎的原因。再通过建立 lstm 模型对新电影的简介以及相关评论进行预测，得出其评分结果；最后利用 Topsis 算法，将这些预测结果进行赋权得出影视的最终得分，然后与以往好坏电影得分进行对比，从而向影视爱好者实现电影的推荐与选择。

关键词： 电影;情感分析;文本挖掘;lstm 算法

目录

1. 绪言

随着现代生活水平的提高，更多的人愿意将时间和精力投入至精神生活，越来越多的人去到电影院。作为国内电影最有影响的莫过于豆瓣（www.douban.com），豆瓣给电影爱好者提供了一个很好的分享及评论平台。每一部电影，在豆瓣上都可以看到很多的评论，可见人们对电影的喜爱程度。另外当前用户面临着海量的电影资源，在观影前可能也就是先看看电影的简介，然后通过查看观众的评论进行选择是否观看这部电影，但是对于每部电影一般都会存在好评与差评。所以当观众选择读取评论若是集中出现好评或者差评很大一定程度就会决定是否观看，那这就有可能出现错失一部偏好的影片，或者浪费自己的情感。由于特定的内容和故事情节可以促进影片的认可，那什么样的电影是大众所喜欢的呢?什么样的电影又是不被大众所喜爱呢？是否这些被观众予以高分的电影或者低分的电影之间存在某些明显不同的地方？通过 Python 爬虫，我们可以获取大量的影视数据来进行分析，以此来挖掘这些数据背后所包含的信息，并且针对电影的简介、评分以及评论进行综合赋分从而实现电影的推荐，让用户能够有一个更好的观影选择。

2. 实验环境

Jupyter notebook、request、pandas、lxml、re、jieba、matplotlib、wordcloud、snownlp、sklearn 等第三方库以及其他工具。

3. 研究思路流程或模型框架

本文的主要研究思路如下图所示：

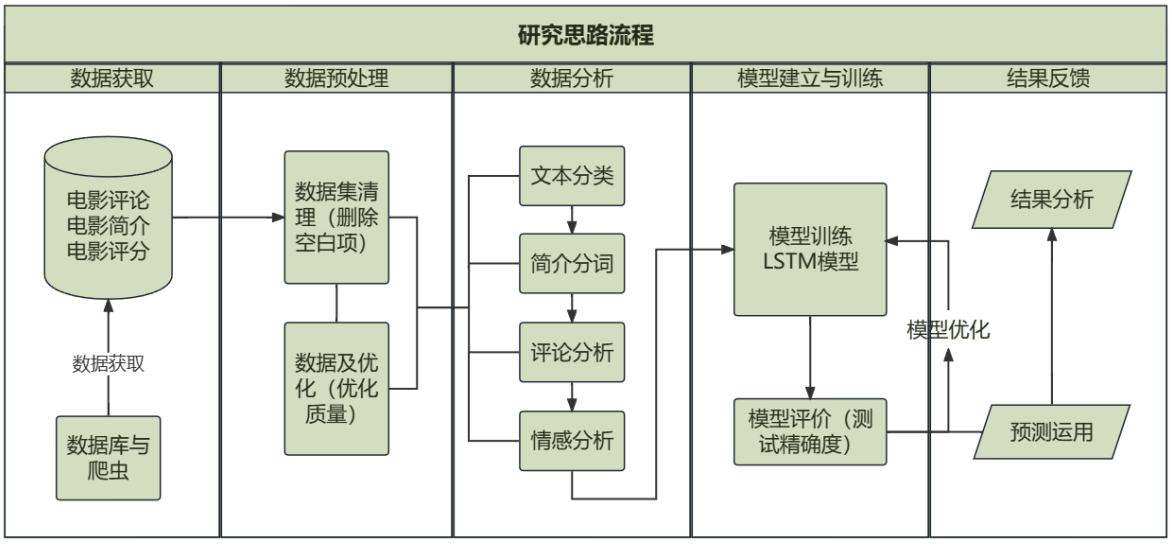


图 1 研究思路流程图

本文主要从电影的评论与简介两个部分进行分析，通过将高分电影的简介与低分电影的简介提取出的关键词进行对比，发掘其各自的特点，对电影简介部分进行优化。其次，对电影评论部分进行情感分析，以排除“高分低评”（打 5 分满分，但是却说电影的不好）的情况。将处理好的评论文本进行中、差、好三类的划分，分别对三类文本数据进行关键词与主题词提取，进一步查看不同类型评价的关注点是哪些内容，以对电影进行优化。最后，使用 LSTM 模型以三种类型的评论文本进行模型训练，完成训练后的模型，可以完成对新出现的评论文本进行快速划分类别。

3.1 关键库介绍

3.1.1 request 库

`requests`库是一个常用的 Python 第三方库，用于发送 HTTP 请求。它提供了一种简洁而方便的方式来与 Web 服务进行交互，并能够处理常见的 HTTP 操作，如 GET、POST、PUT、DELETE 等。

在爬虫部分，我们使用 requests 库的原因是：

1. 简单易用：`requests`库提供了简洁的 API，使发送 HTTP 请求变得非常容易。只需几行代码，就可以发送请求并获取响应。

2. 支持多种请求方法：`requests`库支持常见的 HTTP 请求方法，如 GET、POST、PUT、DELETE 等。您可以根据需要选择适当的方法来与 Web 服务进行交互。

3. 请求和响应处理：可以通过设置请求头、请求参数、请求体等来自定义请求。同时，`requests`库还提供了一系列方法来处理响应，如获取响应状态码、获取响应头、获取响应内容等。

总之，`requests`库是一个功能强大而易于使用的 Python 库，使得与 Web 服务进行 HTTP 交互变得简单。在本文基于 URL 请求对网页进行访问时，`requests`库是最好的选择。

3.1.2 snownlp 库

`snownlp`是一个基于 Python 的自然语言处理（NLP）库，用于中文文本情感分析和文本处理。它提供了一系列功能，包括中文文本分词、情感倾向分析、文本分类等。该库采用了基于概率的算法和机器学习技术，使得处理中文文本变得更加简单和高效。

本文基于`snownlp`库的以下特点和功能，选择其用于情感分析：

1. 文本分词：`snownlp`可以将中文文本进行分词处理，将文本拆分成一个个词

语，方便后续的文本分析和处理。

2. 情感倾向分析：`snownlp`提供了情感分析的功能，可以判断中文文本的情感倾向，例如判断一段文本是正面情感还是负面情感。

需要注意的是，`snownlp`库的性能可能受到训练语料库的质量和规模的影响。因此，在使用时，最好根据实际情况对其结果进行验证和调优，这也是为什么在后续过程中，本文继续使用 LSTM 模型对训练结果进行优化的主要原因。

总而言之，`snownlp`库是一个用于中文文本情感分析和文本处理的 Python 库。它提供了一系列功能，可以帮助我们对中文文本进行分词、情感分析、文本分类等任务，为中文文本处理提供了便利和支持。

3.1.3 sklearn 库

`scikit-learn`（通常简写为`sklearn`）是一个开源的 Python 机器学习库，提供了各种机器学习算法和工具，用于数据挖掘和数据分析任务。它建立在 NumPy、SciPy 和 matplotlib 等科学计算库的基础上，为用户提供了简单且一致的接口，用于构建和应用各种机器学习模型。

我们选择`scikit-learn`库主要原因是：

1. 丰富的机器学习算法：`scikit-learn`库提供了包括回归、分类、聚类、降维、特征选择等在内的多种机器学习算法。它包括线性回归、决策树、随机森林、支持向量机、K 均值聚类等常见的算法，以及许多其他高级算法。在本文中，我们使用基于其的 LSTM 算法。

2. 数据预处理和特征工程：`scikit-learn`库提供了丰富的功能来处理和转换数据。它包括特征缩放、特征选择、特征提取、数据标准化、缺失值处理等方法，简化了我们在数据准备和机器学习中的过程。

3. 评估和模型选择：`scikit-learn`库提供了用于评估模型性能的工具，如交叉验证、指标计算、模型选择等。这些工具可以帮助我们选择最佳的模型和参数配置，以及避免过拟合和欠拟合问题，本文对模型的评价正是基于此的。

总而言之，`scikit-learn`是一个功能强大且易于使用的 Python 机器学习库，提供了丰富的机器

学习算法和工具，适用于各种数据挖掘和数据分析任务。而且非常适合我们这种对于机器学习不甚了解的初学者使用。

3.2 关键技术介绍

3.2.1 网络爬虫

网页解析：爬虫需要能够解析网页的 HTML 或其他标记语言。常用的解析库包括 BeautifulSoup 和 lxml。这些库可以帮助提取网页中的数据和链接，本文主要是用来 lxml 库。

HTTP 请求：爬虫需要发送 HTTP 请求来获取网页内容。本文使用的是 Python 中常用的库 requests，它提供了简单易用的接口来发送 GET、POST 等请求，并可以设置请求头、参数等。

数据提取：爬虫通常需要从网页中提取有用的数据。可以使用解析库提取特定的 HTML 元素，如标签、类名、ID 等，或者使用正则表达式来匹配和提取文本。本文使用了 xpath 对于文本内容进行提取。

3.2.2 文本分词

文本分词（Tokenization）是自然语言处理中的一项基础任务，旨在将连续的文本序列（如句子或段落）切分成单个的词语（或标记）。以下是文本分词中的一些关键技术：

基于规则的分词：这种方法使用事先定义好的规则来切分文本，如基于标点符号、空格或特定的分隔符进行切分。本文引用 re 库，使用正则表达式匹配标点符号进行切分。

基于词典的分词：这种方法使用一个包含常见词语的词典或字典进行切分。将文本与词典进行匹配，找到匹配的词语作为分词结果。这种方法可以处理常见的词语，但对于未登录词（Out-of-vocabulary）会有困难。本文主要使用了 jieba 库进行操作。

基于统计的分词：这种方法利用统计模型和概率算法进行分词。常见的方法是使用马尔可夫模型（如隐马尔可夫模型）或最大熵模型来预测词语边界。这种方法可以根据大量的语料库学习词语的概率分布，从而更好地切分文本。本文引用了 jieba.posseg 模块，通过概率计算，为每个分词结果提供可能的词性标注。

3.2.3 词向量转换

词向量能够表达文本中的词语信息，使得文本数据能够被更好地处理和分析。例如，可以使用词向量作为特征输入机器学习模型，进行文本分类、聚类、情感分析等

任务。通过将文本转换为词向量表示，能够捕捉到词语的语义信息和相对重要性，从而提供更有效的文本表示方式。

本文主要使用的转换技术为 TF-IDF，它的一个常见应用就是将文本转换为词向量表示。TF-IDF 可以将文本中的每个词语转化为一个数值型特征，以便用于机器学习算法或其他文本分析任务。TF-IDF（Term Frequency-Inverse Document Frequency）通过计算词语在文本中的词频（TF）和在整个文集中的逆文档频率（IDF），来评估一个词语在文本中的重要性。词频表示词语在文本中出现的频率，逆文档频率表示词语在整个文集中的普遍程度。

通过计算 TF-IDF 值，可以得到一个词语在文本中的权重，这个权重可以看作是该词语在文本中的重要性程度。将文本中的每个词语都转换为相应的 TF-IDF 权重，就可以构建词向量表示。每个词语对应于词向量中的一个维度，维度上的数值即为该词语的 TF-IDF 权重。

词向量表示具有以下特点：

每个维度对应一个词语，维度的数值表示该词语在文本中的重要性；

维度之间是独立的，因此每个词语都有独立的权重；

3.2.4 主题词提取

LDA（Latent Dirichlet Allocation）是一种常用的主题建模算法，用于从文本集合中提取潜在的主题信息。本文使用 LDA 算法识别文档中隐藏的主题和每个主题的关键词。

LDA 算法的基本思想是，将文档看作是由多个主题组成的混合，并假设每个主题都由一组词语构成。通过对文档中词语的分布进行统计推断，LDA 能够估计每个文档的主题分布和每个主题的词语分布。

LDA 的主要步骤包括：

1. 设定主题数量：首先需要设定主题的数量，也就是期望从文本中提取出的主题个数。

2. 文档-主题分布：随机初始化每个文档的主题分布。

3. 主题-词语分布：随机初始化每个主题的词语分布。

4. 迭代训练：通过迭代的方式优化文档-主题分布和主题-词语分布，使得分布能够最好地解释文本数据。

5. 提取关键词：根据训练好的模型，可以根据每个主题的词语分布，提取每个主题的关键词。

通过 LDA 算法，我们可以获得每个文档的主题分布，了解文档中包含的不同主题的比重，以及每个主题的词语分布，即主题所代表的关键词。

LDA 算法在文本挖掘和信息检索中有广泛应用，例如文本主题分析、文档聚类、文档推荐等任务。通过提取文本中的主题信息，我们可以更好地理解文本集合的内容和结构，从而进行更深入的分析和应用。

3.2.5 情感分析

本文情感分析运用的关键技术主要包括：

情感词典：情感词典是包含情感词汇及其情感极性（如积极、消极）的词典或资源。在情感分析中，可以使用情感词典来匹配文本中的情感词，从而判断文本的情感倾向。本文使用了知网的情感词典作为划分依据。除了情感词典，本文使用 SnowNLP 对电影评论进行情感分析时，其还使用了一些规则和启发式的方法来提高情感分类的准确性。例如，它会考虑否定词、程度副词等修饰词对情感的影响，以及句子中情感词的位置和语境等因素。不过，需要注意的是，SnowNLP 的情感分类算法相对简单，适用于一般的情感分析任务。

机器学习算法：机器学习算法可以用于情感分析的分类任务，如情感分类器。常用的算法包括朴素贝叶斯、支持向量机、决策树、随机森林等。这些算法可以通过训练数据学习文本的情感模式，并预测未标记文本的情感倾向。本文使用的 LSTM 模型就是基于机器学习的。LSTM（Long Short-Term Memory）是一种常用的循环神经网络（Recurrent Neural Network, RNN）的变体，用于处理序列数据和时间序列数据。LSTM 通过引入门控机制，能够有效地解决传统 RNN 中的长期依赖问题。在传统的 RNN 中，信息在序列中的传递容易受到梯度消失或梯度爆炸等问题的影响，导致难以捕捉到长期依赖关系。而 LSTM 通过使用门控单元来控制信息的流动，解决了这一问题。

LSTM 的核心组件包括：

遗忘门（Forget Gate）：用于决定上一个时间步的记忆是否传递给当前时间步。它通过一个 sigmoid 函数输出一个介于 0 和 1 之间的值，表示是否遗忘上一个时间步的记忆。

输入门（Input Gate）：用于决定当前时间步的输入信息。它通过一个 sigmoid 函数输出一个介于 0 和 1 之间的值，表示当前时间步有多少信息被传递给记忆单元。

更新记忆单元（Update Memory Cell）：根据输入门的输出和当前时间步的输入，更新记忆单元的值。

输出门（Output Gate）：用于决定当前时间步的输出。它通过一个 sigmoid 函数输出一个介于 0 和 1 之间的值，表示记忆单元中的信息有多少被传递给输出。

通过这些门控机制，LSTM 能够选择性地存储和遗忘信息，有效地捕捉长期依赖关系。这使得 LSTM 在处理序列数据时表现出更好的记忆能力和建模能力。

4. 实验分析

4.1 数据来源

本文研究数据来自 CSMAR 数据库中的《电影评价研究数据库》，我们选取了其中的豆瓣影评信息表（MEV_DouBanMovRew）作为本文研究数据的主要来源。此外，本文通过爬虫获取了所选电影对应的简介，作为数据的进一步补充，数据具体说明如下：

表 1 数据字段字段说明

字段名称	中文名称	字段类型	单位	字段说明
SeqNum	序号	C	50	
SgnDate	统计日期	C	10	无实际意义。
MovicID	电影 ID	N	20	yyyy-mm-dd,数据更新日期。
MovieTitle	电影名称	C	100	豆瓣电影网址上显示的电影编 号 。
ReviewTime	影评发布日期	C	10	
Reviewer	影评发布者	C	100	yyyy-mm-dd.
ReviewTitle	影评标题	C	400	
ReviewStar	影评星级	C	10	
Contents	影评内容	TEXT		
UsefulNum	认为有用人数	N	20	
ViewedNum	看过评论人数	N	20	
UsfViewRatio	评论有用比	N	4	
ReplyNum	回复评论人数	N	20	评论有用比=认为有用人数/看过评论人数*100。
Introduction	简介	TEXT		
Type	类型	C	10	
Score	评分	N	10	

需注意的是，本表主要记录 2013 年起每部电影具体评论的时间和内容，但由于每部电影尤其是评分较高的电影评论量较多，评论内容所占存储较大，所以目前只收录 2013 年起 100 部的电影的评论内容。

下面，本文以电影的简介、类型、评分数据的爬取进行程序说明：

图 2 电影介绍页内容分析

首先，我们根据电影介绍页的 URL 可以分析出，每部电影对应的介绍页链接为：'https://movie.douban.com/subject/'+'MovieID'，在源数据已给出 MovieID 的情况下，我们所需要做的，就是在爬取时对所需的 URL 进行字符串的拼接工作。

具体代码如下：

```
import requests
import pandas as pd
df = pd.read_excel('./MEV_DOUBANMOVREW.xlsx')
movie_id = df['MovieID'].unique().tolist()
url_list = []
for item in movie_list:
    ul = f"https://movie.douban.com/subject/{item}/"
    url_list.append(ul)
```

此时，url_list 中保存的连接信息就是所需页面的链接了。

下面，本文通过对一部电影的爬取操作进行说明，代码部分如下：

```
url = "https://movie.douban.com/subject/5996801/"
headers = {略}
r = requests.get(url=url,headers=headers)
content = r.text
html = etree.HTML(content)
# 保存一部电影
a = []
# 电影名称
name = html.xpath(r'//div[@id="content"]/h1/span[1]/text()')[0]
print(f'电影名称：{name}')
a.append(name)
# 电影简介
```

```
# introduce = html.xpath(r'//span[@class="all hidden"]/text()')[0].strip()
introduce = html.xpath(r'//div[@class="indent"]/span[1]/text()')[0].strip()
print(f'电影简介：{introduce}')
a.append(introduce)
# 电影类型
variety = html.xpath(r'//span[@property="v:genre"]/text()')
variety = " ".join(variety)
print(f'电影类型：{variety}')
a.append(variety)
# 电影评分
score = html.xpath(r'//strong[@class="ll rating_num"]/text()')[0]
print(f'电影评分：{score}')
a.append(score)
Data_list.append(a)
```

从代码中可以初步看到，本文选用的是通过 Xpath 的属性对所需内容进行定位、爬取的方式，具体流程如下：

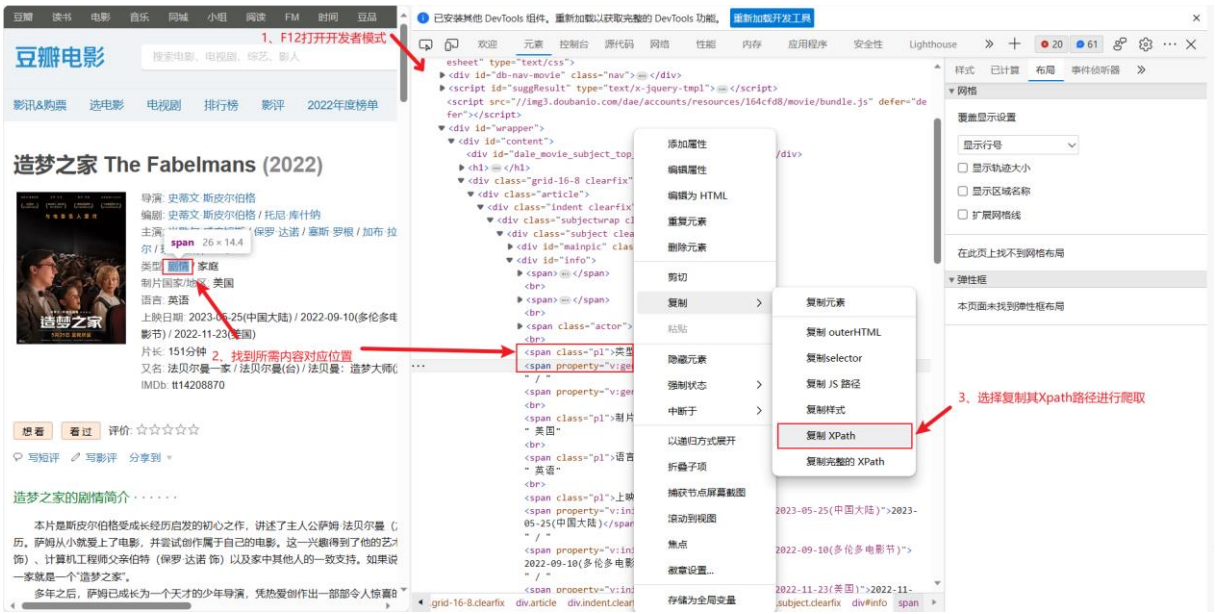


图 3 电影界面爬取

我们可以通过进入开发者模式，在“元素”属性中，找到“类型”文本所在的位置，然后通过右键，复制其 xpath 路径，最后，通过 html.xpath()的方法进行定位，爬取所需内容。

在一部电影的信息爬取成功后，本文选用了循环的方式，对列表中的电影数据信息进行循环爬取。需要注意的是，为了避免豆瓣网站的反爬机制，本文使用了 try--catch 的方式，截取错误信息（在代码运行没问题时，错误主要来自于频繁访问网页后被拒），然后通过 time.sleep()方法进行等待，避免被拒。

4.2 数据清洗与预处理

本文的数据清理与预处理主要包括两个部分，因为大部分的数据来自于数据库平

台，所以那一部分的数据可靠性较强。本文仅对爬取到的数据进行处理，首先是对数据中的空白项进行填充处理，个别电影数据的评分项缺失，因此，本文通过平均值与删除的方式对空白项进行了处理。

其次，由于电影数据较多，在后续的情感分析过程中，受电脑运行限制，本文将数据量进行缩减。首先，本文只使用了电影数据集中的 1000 部电影的数据，其次，本文对每部电影的评论数进行取舍，仅选择部分电影评论数据进行分析。

为此，本文制定了以下函数进行提取：

```
def extract_rows_by_id(input_file, output_file, id_column):
    # 创建一个字典来存储每个数据 ID 对应的行
    rows_by_id = {}
    with open(input_file, 'r', newline='', encoding='utf-8') as file:
        reader = csv.DictReader(file)
        for row in reader:
            # 获取当前行的数据 ID
            data_id = row[id_column]

            # 将当前行添加到对应数据 ID 的列表中
            if data_id in rows_by_id:
                rows_by_id[data_id].append(row)
            else:
                rows_by_id[data_id] = [row]
    with open(output_file, 'w', newline='', encoding='utf-8') as file:
        writer = csv.DictWriter(file, fieldnames=reader.fieldnames)
        writer.writeheader()
        # 遍历字典中的每个数据 ID 的前 100 行，并写入新文件
        for data_id, rows in rows_by_id.items():
            for row in rows[:100]:
                writer.writerow(row)
```

其大致思路为，定义一个字典 rows_by_id 来存储每个 ID 对应的行。通过打开输入文件 input_file 并使用 csv.DictReader 进行读取，将每行作为一个字典添加到 rows_by_id 中，其中键为 ID，值为行的字典列表。对于每一行，先获取指定列（id_column）的值作为 data_id，然后检查 data_id 是否已经存在于 rows_by_id 字典中。如果存在，则将当前行添加到对应的列表中；如果不存在，则为该 data_id 创建一个新列表，并将当前行添加进去。在读取完输入文件后，打开输出文件 output_file，并使用 csv.DictWriter 创建一个写入对象。这个写入对象的列名为原来的列名。为新文件写入列名，然后遍历 rows_by_id 字典中的每个键值对，也就是每个 data_id 及其对应的行列表。对于每个 data_id，仅选择前 100 行（如果存在的话），然后将它们写入新文件。

至此，全部的数据爬取与数据清洗工作全部结束。

4.3 文本分类

将已经进行数据处理的影视数据进行后续操作。由于电影具有很多种类型，比如武侠、战争、悬疑、惊悚等等，因此我们起初是想将影视按照这些类型进行各自分析，提取出相同类型的代表情节以及不同之处，从而寻找观众更为偏好的情节以及较为反感的剧情。但是该数据集中的电影并非是单类型的分类，有的甚至同时被分成了好几类，比如《扫毒》，它既是剧情、犯罪也是动作、悬疑，因此不好将其严格划分，所以我们单从影视评分进行分析。考虑到每一部电影都会附带一个电影的简介，并且该简介中也会包含电影的主要情节。那么我们从这些简介中提取出这些影视的主要情节，然后根据电影的评分将电影人共分成好片与烂片。考虑到电影的评分是从 0.0 到 10 的分数区间，因此我们将电影以 5.0 作为分界分成两类。电影评分在 5.0 以上的认为是好片，5.0 以下的认为是烂片。

而对于评论而言，观众会将自己的观影感受在评论中体现出来。我们可以从这些评论中提取观众提及的影片情节。由于评论是按照 0 到 5 星进行评级的，因此我们将 4 星以上的认为是对电影的好评，3 星级则是中评，2 星级以下的作为差评。由于该数据集包含的评论数据较大，并且包含了许多与需要分析的无关数据，因此需要先将需要分析的数据提取出来。部分分类算法如下：

```
senti_sort=[]
for i in film_introduce["ReviewStar"]:
  if ( i>3.0 ):
    senti_sort.append(1)
  elif (i==3.0):
    senti_sort.append(0)
  else:
    senti_sort.append(-1)
```

SeqNum	SgnDate	MovieID	MovieTitle	ReviewTin	Reviewer	ReviewTitl	ReviewSta	Contents	UsefulNur	ViewedNu	UsfViewRa	ReplyNum	rating	movie_typ	movie_introduce
79	######	5308265	[illegible]	7	[illegible]	3	[illegible]	0	0	0	7.2	[illegible]			[illegible]
80	######	5308265	[illegible]	G-Class	[illegible]	5	[illegible]	0	0	0	0	7.2	[illegible]		[illegible]
81	######	5308265	[illegible]	[illegible]	[illegible]	3	[illegible]	0	0	0	7.2	[illegible]			[illegible]
82	######	5308265	[illegible]	[illegible]	[illegible]	5	80	2	7.2	[illegible]			[illegible]		
83	######	5308265	[illegible]	[illegible]	4	[illegible]	0	0	0	7.2	[illegible]			[illegible]	
112	######	5308265	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0	7.2	[illegible]			[illegible]	
113	######	5308265	[illegible]	[illegible]	[illegible]	[illegible]	7	85.7	2	7.2	[illegible]			[illegible]	
114	######	5308265	[illegible]	[illegible]	[illegible]	[illegible]	0	0	0	0	7.2	[illegible]			[illegible]
115	######	5308265	[illegible]	...ing	[illegible]	[illegible]	0	0	0	7.2	[illegible]			[illegible]	
116	######	5308265	[illegible]	[illegible]	[illegible]	2	[illegible]	0	0	0	7.2	[illegible]			[illegible]
117	######	5308265	[illegible]	[illegible]	[illegible]	[illegible]	3	66.7	0	7.2	[illegible]			[illegible]	
242	######	5308265	[illegible]	[illegible]	3	[illegible]	0	0	0	7.2	[illegible]			[illegible]	
243	######	5308265	[illegible]	[illegible]	[illegible]	5	[illegible]	0	0	0	0	7.2	[illegible]		[illegible]
						[illegible]									

图 4 原数据集部分数据展示

4.4 简介分词

首先将提取好的电影数据利用 pandas 库进行读取，由于某些电影虽然附带了评分但是它的简介是缺失的，并且对于豆瓣影视上的该电影也是没有简介的，因此我们直接将其删去。然后利用正则表达式将简介当中的数字以及字母连同电影相关的词语进行去除。接着将原来已经进行分类的影视简介从好片和烂片分别进行分析。

利用 jieba 库对简介进行分词处理，由于利用 pandas 迭代分词需要较长的时间，

因此我们先将这些影视简介单独保存到一个 txt 文件下，并且用 utf-8 格式进行存储。然后采用文本读取，直接使用 jieba.lcut()方法对整个文本进行分词处理。接着，使用了哈工大的停用词表对分词结果进行去除停用词，然后进行词频的统计与高频词的提取。再利用 TF-IDF 进行关键词的提取。

其中，TF-IDF（Term Frequency-inverse Document Frequency）是一种统计方法，其中 TF（Term Frequency）的意思是词频，IDF（Inverse Document Frequency）的意思是逆文本频率指数，TF-IDF 算法所求实际上就是这两者相乘所得的乘积。该算法的主要思想为:若某词在一类指定的文本中出现的频率很高，而这个词在其他类文本中出现的频率很低，那么认为该词具有此类文本某些代表性的特征，可用词对此类文本进行分类。主要采用的方法如下：

```
#定义计算函数
def tf(word,count):
    return count[word]/sum(count.values())
def idf(word,freq_word):
    n_contain = sum([1 for count in freq_word if word in count])
    return math.log(len(freq_word)/(1+n_contain))
def tf_idf(word,count,freq_word):
    return tf(word,count) * idf(word,freq_word)
import math
scores = {word:tf_idf(word,counts,freq_word_) for word in counts}
scores_word = sorted(scores.items(),key=lambda x:x[1],reverse=True)
for word,score in scores_word:
    print('word:{},TF-IDF:{}'.format(word,round(score,5)))
```

最后，根据关键词提取结果利用词语图的形式进行展现，从中可以初步得到这些电影简介之间的相似之处以及不同的地方。主要采用的代码有：

```
import matplotlib.pyplot as plt
from wordcloud import WordCloud
backgroud_Image=plt.imread(r'E:\Jupyter\test_2\文本分析\4.jpg')
# 自己上传中文字体到 kesci
font_path = 'msyh.ttc'
wordcloud = WordCloud(font_path=font_path, # 设置字体，不设置就会出现乱码
                      max_words=100,
                      background_color='white',
                      mask=backgroud_Image)# 词云形状
my_wordcloud = wordcloud.fit_words(frequencies)
plt.imshow(my_wordcloud)
plt.axis('off')
plt.show()
```

由于最终得到的结果当中有很多词，比如发生、来到、过程、名为等词都是没有明显含义的词语，并不不能体现出很好的故事情节或者场景，因此我们后续将一些无关紧要的词语添加到停用词表当中。并且这些得到的词语很多程度上都是相似的，也体现不出好片与烂片之间的差异，因此，我们试着使用三个词的词频、四个词的词频

进行进一步分析，发现它们之间存在较为明显的差异，因此可以对其进行一定的分析。

4.5 评论分析

对于评论的处理，我们继续沿用类似简介的处理。首先，先将分类好的评论进行正则处理。然后利用 jieba 进行分词，较为不同的是这次我们将分词的词性也标注出来了。由于在对简介分词的结果当中可以看出很多的词语都是没有实际意义的，而且是动词居多，因此这次我们将具有代表性意义的名词、人名等词语单独提取出来进行分析。同样进行后续的词频统计，高频词的提取，以及制作相关的词语图进行后续对比分析。为了更好验证这一结果的准确性，后续对这些评论进行情感分析。此过程关键的代码如下：

```
with open('./result/film_mid.txt','r',encoding="utf-8") as fo:
    s2 = fo.read()
#词性分词
mid_contents=jieba.posseg.lcut(s2)
#提取较为代表性的词语
mid_contents=[x.word for x in mid_contents if x.flag=="n" or x.flag=="nr" or x.flag=="nrt" or x.flag=="i" or x.flag=="l"]
#去除停用词
stopwords = [line.strip() for line in open('./stop/stop_word.txt',encoding='UTF-8').readlines()]
mid_word =[x for x in mid_contents if len(x) >1 and x not in stopwords]
```

4.6 情感分析

由于评论文本数据量较大，将评论按照差评，中评，好评分别进行处理。即便是划分后，三个评论文件依旧包含着几万条长文本数据，因此按好评差评中评分别按比例抽取了 1000 左右评论，以便于后续过程的开展，降低运行难度。

4.6.1 词典匹配

将上述过程所得到的分词结果，读入正向情感词表以及负向情感词表，读取两个词表的词，并分别赋予 1 和-1 的得分，将分词结果与情感词典之间进行匹配，定位情感词在评论语句中的位置，代码如下：

```
# 读入正面、负面情感评价词
pos_comment = pd.read_csv(r"/home/aistudio/data/data218693/praise.txt", header=None,sep="\n",
                          encoding = 'utf-8', engine='python')
neg_comment = pd.read_csv(r"/home/aistudio/data/data218693/degrade.txt", header=None,sep="\n",
                          encoding = 'utf-8', engine='python')
positive = list(set(pos_comment.iloc[:,0]))
negative = list(set(neg_comment.iloc[:,0]))

positive = pd.DataFrame({"word":positive,
                         "weight":[1]*len(positive)})
negative = pd.DataFrame({"word":negative,
```

```
                "weight":[-1]*len(negative)})

posneg = positive.append(negative)
# 将分词结果与正负面情感词表合并，定位情感词
data_posneg = posneg.merge(word, left_on = 'word', right_on = 'word', how = 'right')
data_posneg = data_posneg.sort_values(by = ['index_content','index_word'])
```

由于否定词的存在，当正向情感词前出现否定词时，该字段将表达一个负向的情感，例如：“不好”、“不容易”等，因此需要根据否定词在评论中的位置进行修正情感倾向，减少或避免系统对评论情感值的误判，进行情感极性分析。载入否定词表，定位否定词在句子中的位置，修正经过否定词修饰过的情感值，提高模型的准确率。

```
# 载入否定词表
notdict = pd.read_csv(r"/home/aistudio/not.txt")
# 构造新列，作为经过否定词修正后的情感值
data_posneg['amend_weight'] = data_posneg['weight']
data_posneg['id'] = np.arange(0, len(data_posneg))
# 只保留有情感值的词语
only_inclination = data_posneg.dropna().reset_index(drop=True)

index = only_inclination['id']
for i in np.arange(0, len(only_inclination)):
  # 提取第 i 个情感词所在的评论
  review = data_posneg[data_posneg['index_content'] == only_inclination['index_content'][i]]
  review.index = np.arange(0, len(review))
  # 第 i 个情感值在该文档的位置
  affective = only_inclination['index_word'][i]
  if affective == 1:
    ne = sum([i in notdict['不'] for i in review['word'][affective - 1]])%2
    if ne == 1:
      data_posneg['amend_weight'][index[i]] = -data_posneg['weight'][index[i]]
  elif affective > 1:
    ne = sum([i in notdict['不'] for i in review['word'][[affective - 1,
          affective - 2]]])%2
    if ne == 1:
      data_posneg['amend_weight'][index[i]] = -data_posneg['weight'][index[i]]
```

接下来更新只保留情感值的数据，计算每条评论的情感值，将评论中各个词的情感得分相加，所得情感值作为该评论的情感得分，在最终的评论列表中去除情感值为 0 的评论。将情感值大于 0 的划分为积极评论，小于 0 的则为消极评论。

	index_content	a_type
1	1	neg
2	2	pos
3	3	pos
4	4	neg
5	5	pos
6	6	pos
7	7	pos
8	8	pos
9	9	pos
10	10	pos
11	11	pos
12	12	pos
13	13	pos
14	14	pos
15	15	neg
16	16	pos
17	17	pos

图 5 部分积极消极评论分类结果

4.6.2 snownlp 情感分析

SnowNLP 基于朴素贝叶斯分类模型进行计算情感得分，依据情感得分结果用于将中文文本分类为积极和消极。针对分类后评论文本，采用 snownlp 库中的 SnowNLP(x).sentiments 函数对其进行情感分析，将情感得分结果进行统计，绘制情感得分直方图。以 0.5 作为界限，情感得分高于 0.5 作为积极评论，低于 0.5 则作为消极评论，将情感得分与评论星级进行绘制折线图，探究两者之间的关系。部分关键代码如下：

```
from snownlp import SnowNLP  #情感分析

df1['emotion'] = df1['Contents'].apply(lambda x:SnowNLP(x).sentiments)
df1.head(10)
#情感分直方图
import matplotlib.pyplot as plt
import numpy as np
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus'] = False
bins=np.arange(0,1.1,0.1)
plt.hist(df1['emotion'],bins,color='#4F94CD',alpha=0.9)
plt.xlim(0,1)
plt.xlabel('情感分')
plt.ylabel('数量')
```

```
plt.title('情感分直方图')
plt.show()
```

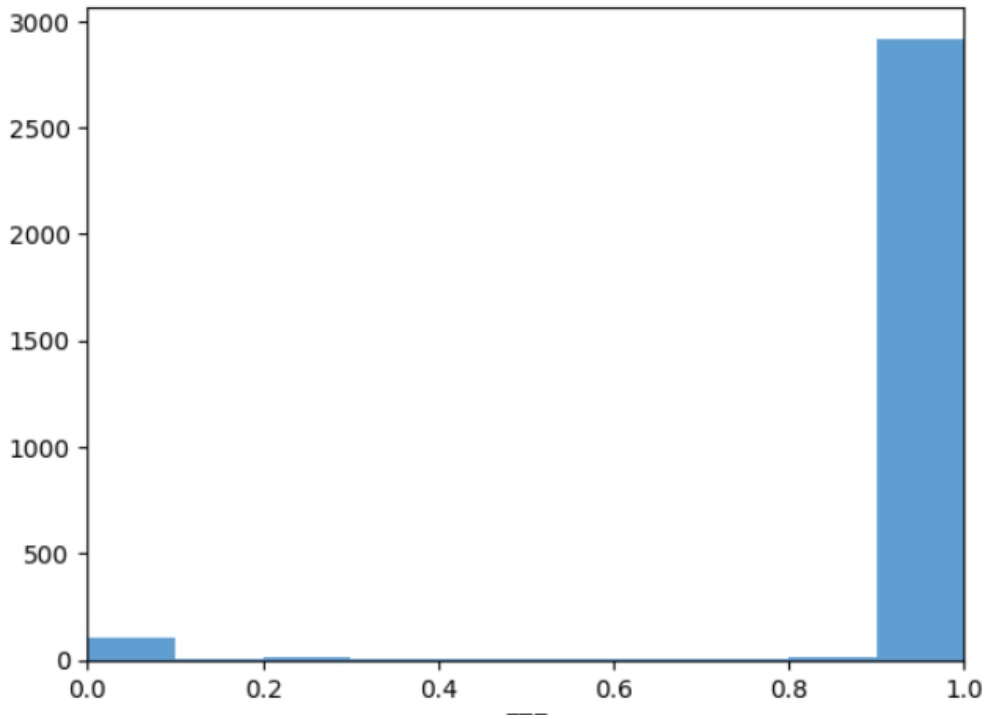


图 6 情感分类直方图结果

```
# 随机抽取 100 行数据
fo_sampled = fo.sample(n=100, random_state=1)
fo_sampled
from pyecharts.charts import Line
from pyecharts import options as opts


# 示例数据
cate = [str(i) for i in range(1, 101)]
data1 = fo_sampled['emotion'][1:101]
data2 = fo_sampled['ReviewStar'][1:101]
from pyecharts.charts import Line
from pyecharts import options as opts

line = Line()
line.add_xaxis(cate)
line.add_yaxis("情感得分", data1)
line.add_yaxis("星级得分", data2)
line.set_global_opts(title_opts=opts.TitleOpts(title="情感分析-走势图"))

# 将图表保存为 HTML 文件
line.render("emotionstar.html")
```

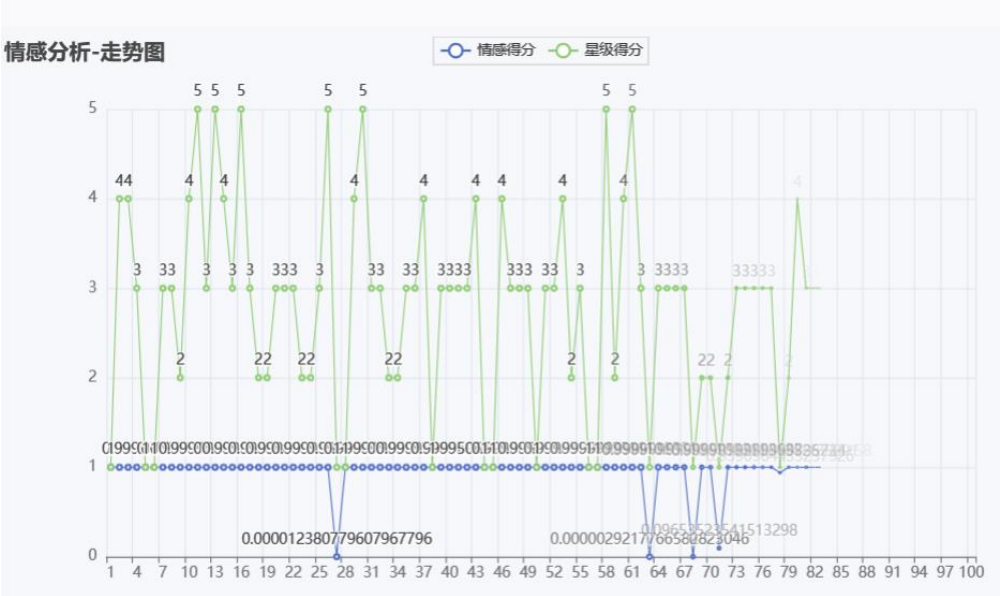


图 7 情感分析走势图

4.6.3 LDA 情感主题词提取

分别读取经过基于词典匹配情感分析后的正面和负面数据集：posdata 和 negdata，接下来，使用 Gensim 库中的 Dictionary()函数建立词典。然后分别基于正面和负面评论数据集，将每个词转换为包含单词编号及其出现次数的向量，组成了正面和负面的语料库（即 pos_corpus 和 neg_corpus）。这个步骤用到了词袋模型（Bag of Words Model），是自然语言处理中常用的文本表示方式。通过余弦相似度函数，计算两个向量之间的余弦相似度；定义了一个 lda_k 函数，用于对给定的文本数据进行主题数寻优。主题数的范围为 2-10，对于每个主题数，都会训练一个 LDA 模型，并计算各个主题之间的相似度。具体流程如下：

1. 使用 gensim.models.LdaModel 训练 LDA 主题模型；

2. 使用 show_topics 方法获取每个主题的关键词；

3. 提取各个主题的关键词，并构造词频向量；

4. 计算各个主题之间的余弦相似度；

5. 计算平均余弦相似度。

将语料库代入函数进行计算，得出其平均余弦相似度，并绘制主题平均余弦相似度图形，依据图形结果可得出正负向语料的最优主题数，将主题数输入到 lda 模型中去得到最后的主题关键词。

```
# 余弦相似度函数
def cos(vector1, vector2):
  dot_product = 0.0;
```

```
    normA = 0.0;
    normB = 0.0;
    for a,b in zip(vector1, vector2):
        dot_product += a*b
        normA += a**2
        normB += b**2
    if normA == 0.0 or normB==0.0:
        return(None)
    else:
        return(dot_product / ((normA*normB)**0.5))
# 主题数寻优
def lda_k(x_corpus, x_dict):

    # 初始化平均余弦相似度
    mean_similarity = []
    mean_similarity.append(1)

    # 循环生成主题并计算主题间相似度
    for i in np.arange(2,11):
        # LDA 模型训练
        lda = models.LdaModel(x_corpus, num_topics = i, id2word = x_dict)
        for j in np.arange(i):
            term = lda.show_topics(num_words = 50)

        # 提取各主题词
        top_word = []
        for k in np.arange(i):
            top_word.append([''.join(re.findall('"(.*)"',i)) \
                        for i in term[k][1].split('+')]) # 列出所有词

        # 构造词频向量
        word = sum(top_word,[]) # 列出所有的词
        unique_word = set(word) # 去除重复的词

        # 构造主题词列表，行表示主题号，列表示各主题词
        mat = []
        for j in np.arange(i):
                    top_w = top_word[j]
            mat.append(tuple([top_w.count(k) for k in unique_word]))

        p = list(itertools.permutations(list(np.arange(i)),2))
        l = len(p)
        top_similarity = [0]
        for w in np.arange(l):
            vector1 = mat[p[w][0]]
            vector2 = mat[p[w][1]]
            top_similarity.append(cos(vector1, vector2))


                # 计算平均余弦相似度
        mean_similarity.append(sum(top_similarity)/l)
```

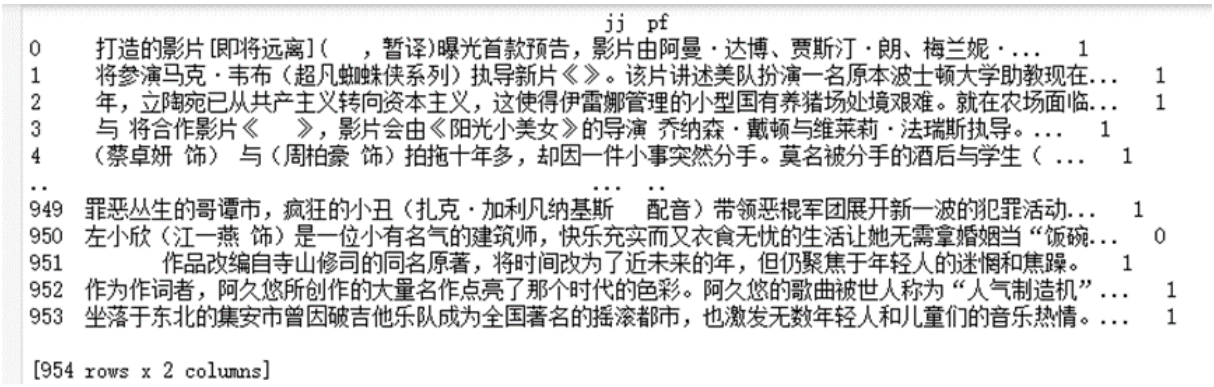
```
return(mean_similarity)
```

4.7 lstm 预测

通过对先前处理分类的数据进行汇总，进行再次处理，对评论以及简介中存在特殊符号以及无法识别的内容进行删除。按照之前评论的积极、中性、消极的分类分别设置（1、0、-1）简介对应电影高分和低分设置为（1，0）进行标记。

4.7.1 简介

先对简介的数据进行输出。一共 954 条可用简介与评分：

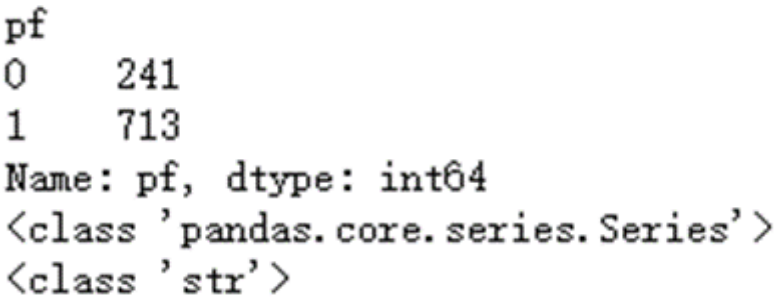


```
                                                   jj  pf
0     打造的影片[即将远离](    ，暂译)曝光首款预告，影片由阿曼·达博、贾斯汀·朗、梅兰妮·...   1
1     将参演马克·韦布（超凡蜘蛛侠系列）执导新片《》。该片讲述美队扮演一名原本波士顿大学助教现在...   1
2     年，立陶宛已从共产主义转向资本主义，这使得伊雷娜管理的小型国有养猪场处境艰难。就在农场面临...   1
3     与 将合作影片《    》，影片会由《阳光小美女》的导演 乔纳森·戴顿与维莱莉·法瑞斯执导。...   1
4     （蔡卓妍 饰） 与（周柏豪 饰）拍拖十年多，却因一件小事突然分手。莫名被分手的酒后与学生（ ...   1
..                                                  ...  ..
949   罪恶丛生的哥谭市，疯狂的小丑（扎克·加利凡纳基斯    配音）带领恶棍军团展开新一波的犯罪活动...   1
950   左小欣（江一燕 饰）是一位小有名气的建筑师，快乐充实而又衣食无忧的生活让她无需拿婚姻当“饭碗...   0
951          作品改编自寺山修司的同名原著，将时间改为了近未来的年，但仍聚焦于年轻人的迷惘和焦躁。    1
952   作为作词者，阿久悠所创作的大量名作点亮了那个时代的色彩。阿久悠的歌曲被世人称为“人气制造机”...   1
953   坐落于东北的集安市曾因破吉他乐队成为全国著名的摇滚都市，也激发无数年轻人和儿童们的音乐热情。...   1

[954 rows x 2 columns]
```

图 8 提取的部分简介数据

统计好评与差评电影出现次数：差评（241）、好评（713）以及数据类型：



```
pf
0    241
1    713
Name: pf, dtype: int64
<class 'pandas.core.series.Series'>
<class 'str'>
```

图 9 好片与烂片数量统计结果

统计句子长度以及出现频数：

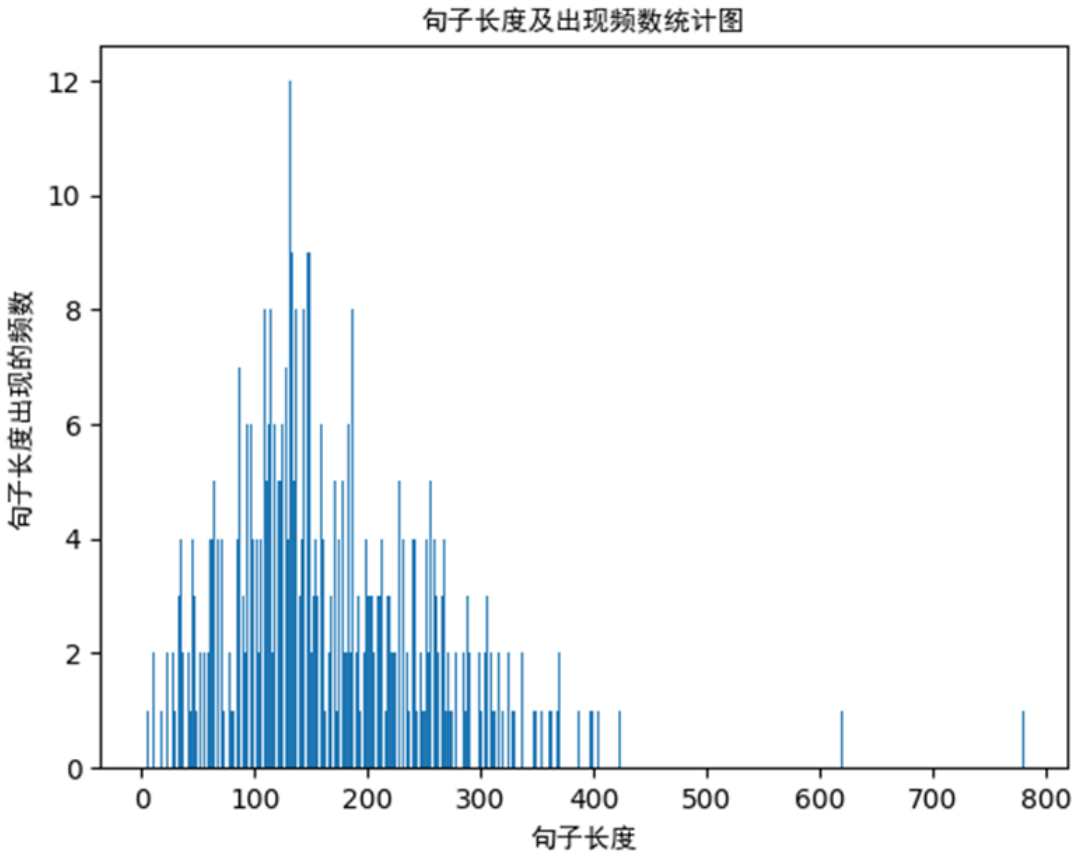


图 10 句子长度以及出现频数统计图

绘制句子长度累积分布函数图：

找分位点为 0.91 的句子长度：276.

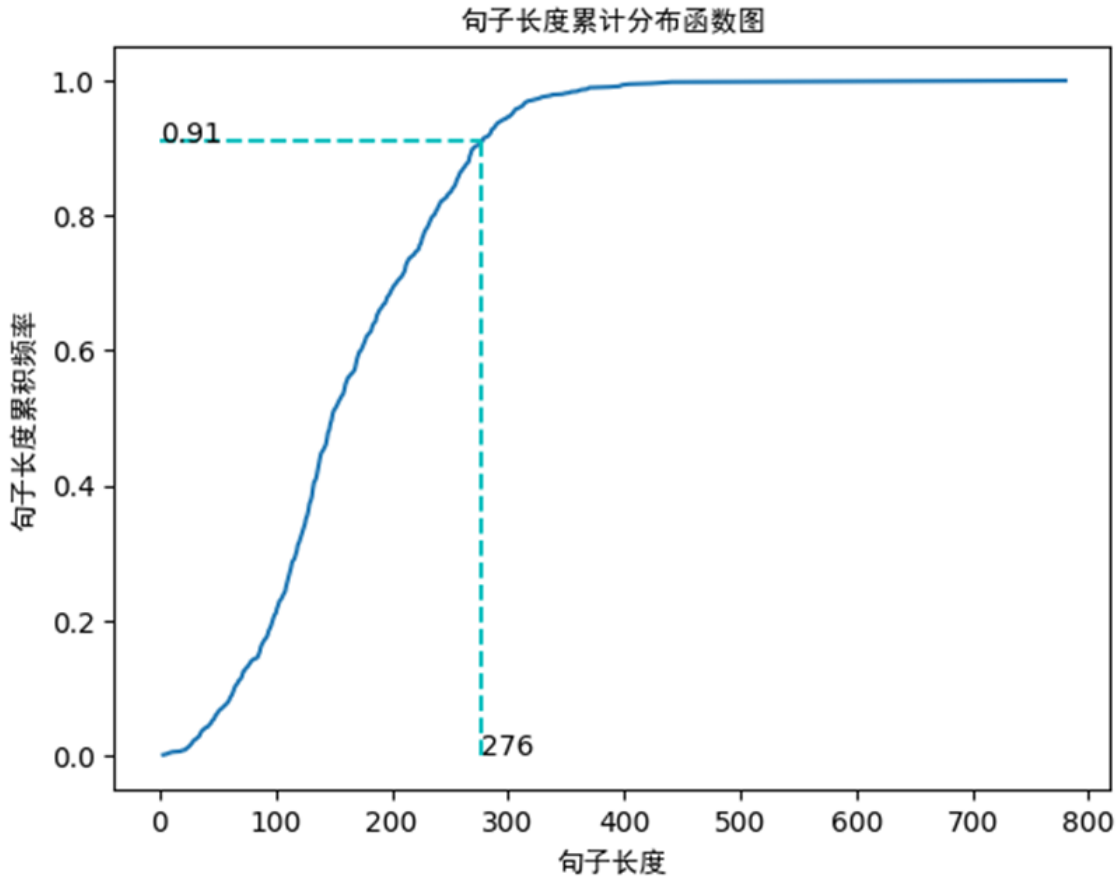


图 11 句子长度累积分布函数图

此处分为点数选 0.91 的原因是能过较好涵盖 91%句子的长度为接下来的序列填充做准备。

进行字典构建，首先筛选出不同的简介以及不同的标签；对简介进行字符切割后加入字典，标签也是。具体过程如下：

```
def load_data(filepath, input_shape=20):
    df = pd.read_excel(filepath)
    # 标签及词汇表
    labels, vocabulary = list(df['pf'].unique()), list(df['jj'].unique())
    # print(len(labels))
    # print(len(vocabulary))
    # 构造字符级别的特征
    string = ''
    for word in vocabulary:
        string += word
    # print(string)
    vocabulary = set(string)
    # print(vocabulary)
    # 字典列表
    word_dictionary = {word: i + 1 for i, word in enumerate(vocabulary)}
    with open('word_dict.pk', 'wb') as f:
```

```
    pickle.dump(word_dictionary, f)
inverse_word_dictionary = {i + 1: word for i, word in enumerate(vocabulary)}
label_dictionary = {label: i for i, label in enumerate(labels)}
with open('label_dict.pk', 'wb') as f:
    pickle.dump(label_dictionary, f)
output_dictionary = {i: labels for i, labels in enumerate(labels)}
```

因为每一个句子的长度不尽相同，因此生成的词向量的大小不统一，回到导致结果出错，因此在这里进行一步序列填充，来保证程序的正常运行。而填充值则是根据刚才的分位图进行确定。具体代码如下：

```
 vocab_size = len(word_dictionary.keys()) # 词汇表大小
label_size = len(label_dictionary.keys()) # 标签类别数量
# print(vocab_size, labels)
# 序列填充，按 input_shape 填充，长度不足的按 0 补充
x = [[word_dictionary[word] for word in sent] for sent in df['jj']]
x = pad_sequences(maxlen=input_shape, sequences=x, padding='post', value=0)
y = [[label_dictionary[sent]] for sent in df['pf']]
y = [np_utils.to_categorical(label, num_classes=label_size) for label in y]
y = np.array([list(_[0]) for _ in y])
return x, y, output_dictionary, vocab_size, label_size, inverse_word_dictionary
```

进行 LSTM 模型搭建，具体代码如下：

```
def create_LSTM(n_units, input_shape, output_dim, filepath):
    x, y, output_dictionary, vocab_size, label_size, inverse_word_dictionary = load_data(filepath)
    model = Sequential()
    model.add(Embedding(input_dim=vocab_size + 1, output_dim=output_dim,
                        input_length=input_shape, mask_zero=True))
    model.add(LSTM(n_units, input_shape=(x.shape[0], x.shape[1])))
    model.add(Dropout(0.2))
    model.add(Dense(label_size, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    plot_model(model, to_file='./model_lstm.png', show_shapes=True)
    model.summary()
    return model
```

划分训练集与测试集。这里选择训练集与测试集占比为 9：1 具体代码如下：

```
def model_train(input_shape, filepath, model_save_path):
    # 将数据集分为训练集和测试集，占比为 9:1
    x, y, output_dictionary, vocab_size, label_size, inverse_word_dictionary = load_data(filepath, input_shape)
    train_x, test_x, train_y, test_y = train_test_split(x, y, test_size=0.1, random_state=42)
    # 模型输入参数，需要自己根据需要调整
    n_units = 100
    batch_size = 32
    epochs = 20
    output_dim = 20
    # 模型训练
    lstm_model = create_LSTM(n_units, input_shape, output_dim, filepath)
    lstm_model.fit(train_x, train_y, epochs=epochs, batch_size=batch_size, verbose=1)
    # 模型保存
    lstm_model.save(model_save_path)
```

我们将神经元数设置为 100、一次训练所选样本数大小设置为 32 条，训练轮次为 20 轮、经过 embedding 层降维后的数据由 20 个元素组成。

进行测试集的测试，并评估准确率（所有正确预测值与所有预测值的比值）：

```
N = test_x.shape[0]  # 测试的条数
  predict = []
  label = []
  for start, end in zip(range(0, N, 1), range(1, N + 1, 1)):
    sentence = [inverse_word_dictionary[i] for i in test_x[start] if i != 0]
    y_predict = lstm_model.predict(test_x[start:end])
    label_predict = output_dictionary[np.argmax(y_predict[0])]
    label_true = output_dictionary[np.argmax(test_y[start:end])]
    print(''.join(sentence), label_true, label_predict)  # 输出预测结果
    predict.append(label_predict)
    label.append(label_true)
  acc = accuracy_score(predict, label)  # 预测准确率
  print('模型在测试集上的准确率为: %s.' % acc)
```

在定义完词典、模型、训练集与测试集之后，开始进行模型训练以及测试：

```
if __name__ == '__main__':
  filepath = "D:\\360MoveData\\Users\\王瑞钰（0^0）\\Desktop\\comment.xlsx"
  input_shape = 280
  # load_data(filepath, input_shape)
  model_save_path = './corpus_model.h5'
  model_train(input_shape, filepath, model_save_path)
```

这里选择填充至大小 280，可以覆盖 91%简介的全部文字。能够更有效的构建词典以及词索引。

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 180, 20)	75160
lstm_1 (LSTM)	(None, 100)	48400
dropout_1 (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 2)	202

Total params: 123,762
Trainable params: 123,762
Non-trainable params: 0

图 12 构建的模型基本参数

输出模型各层得分与总分。然后开始训练：

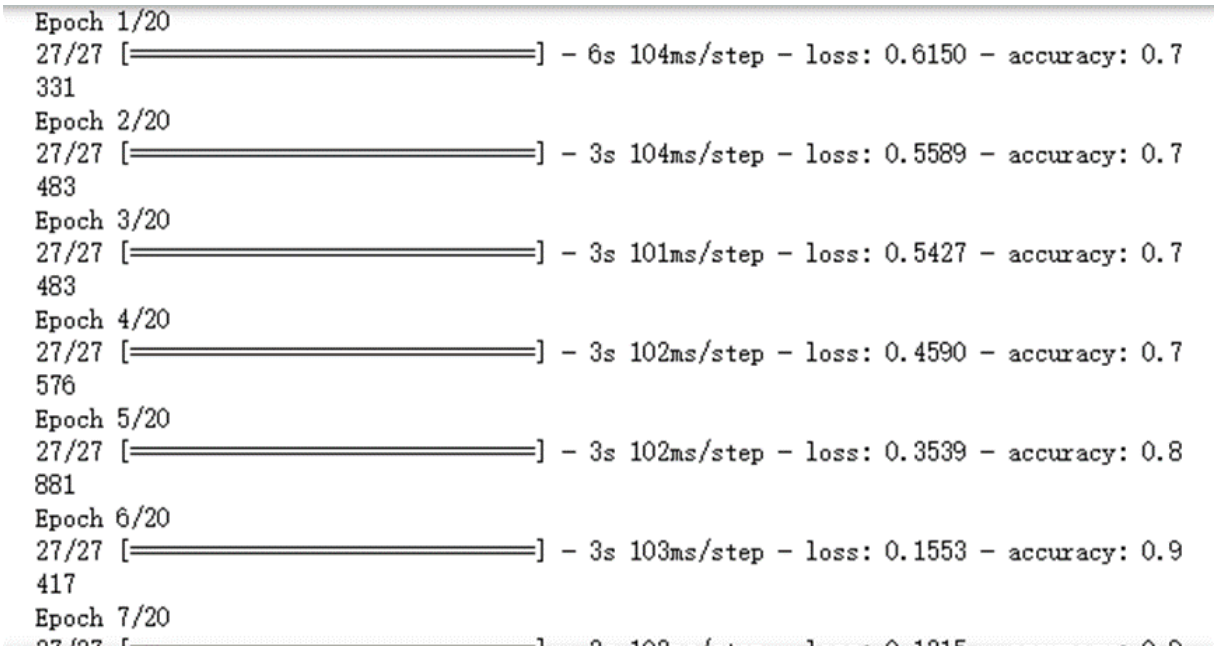


图 13 模型训练部分过程

训练结束后，对先前划分的测试集进行逐条测试：

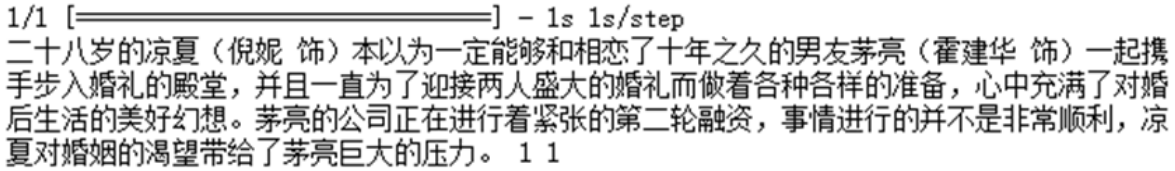


图 14 评论预测示例结果

最后两个数字为实际值与预测值。可以看到上述例子预测正确。

通过对 10%的测试集进行预测后计算准确率：

图 15 模型预测准确度

该预测率会随着参数改变而改变，目前测试平均值在 67%左右

接下来通过到出的模型进行预测：具体代码如下：

```
# Import the necessary modules
import pickle
import numpy as np
from keras.models import load_model
from keras_preprocessing.sequence import pad_sequences
# 导入字典
with open('word_dict.pk', 'rb') as f:
    word_dictionary = pickle.load(f)
with open('label_dict.pk', 'rb') as f:
    output_dictionary = pickle.load(f)
try:
    # 数据预处理
```

```
    input_shape =280
    sent = "该片根据同名小说改编，讲述了未来世界的外星黑暗势力突袭地球，江洋和林澜肩负起末世下拯救地球的重任，和幸存的人类在中国上海与外星文明展开终极大战的故事。"
    x = [[word_dictionary[word] for word in sent]]
    x = pad_sequences(maxlen=input_shape, sequences=x, padding='post', value=0)
    # 载入模型
    model_save_path = './corpus_model.h5'
    lstm_model = load_model(model_save_path)
    # 模型预测
    y_predict = lstm_model.predict(x)
    label_dict = {v: k for k, v in output_dictionary.items()}
    print('输入语句: %s' % sent)
    print('情感预测结果: %s' % label_dict[np.argmax(y_predict)])
except KeyError as err:
    print("您输入的句子有汉字不在词汇表中，请重新输入！")
    print("不在词汇表中的单词为：%s." % err)
```

通过字典以及导出的模型对 sent 内的电影简介进行电影得分的预测。因为预测需要字典配合，因此当简介包含字典内不存在的词时会进行提示。面对提示有两种解决方法，一种是将带有该字的简介加入训练集中进行训练，另一种则是删除该字。

这里将《搏击俱乐部》的电影简介带入，提示有数字，将数字删去后得到如下结果：

```
1/1 [==============================] - 1s 1s/step
输入语句:《搏击俱乐部》是世纪福斯电影公司于年发行的一部悬疑惊悚片，电影改编自恰克·帕拉尼克的同名小说，由大卫·芬奇执导，布拉德·皮特、爱德华·诺顿、海伦娜·伯翰·卡特等主演。 该片讲述了生活苦闷的泰勒为了找寻刺激与好友杰克组成“搏击俱乐部”，在那里他们可以把一切不快的情绪宣泄，借着自由搏击获得片刻快感的故事。
情感预测结果: 1
```

图 16 简介预测示例结果

4.7.2 评论

选择 34518 条好评、中评、差评。首先观察评论数据信息：

```
[34518 rows x 2 columns]
senti_sort
-1     8795
 0    15791
 1     9932
Name: senti_sort, dtype: int64
<class 'pandas.core.series.Series'>
<class 'str'>
```

图 17 评论信息图

观察句子长度累计分布图：

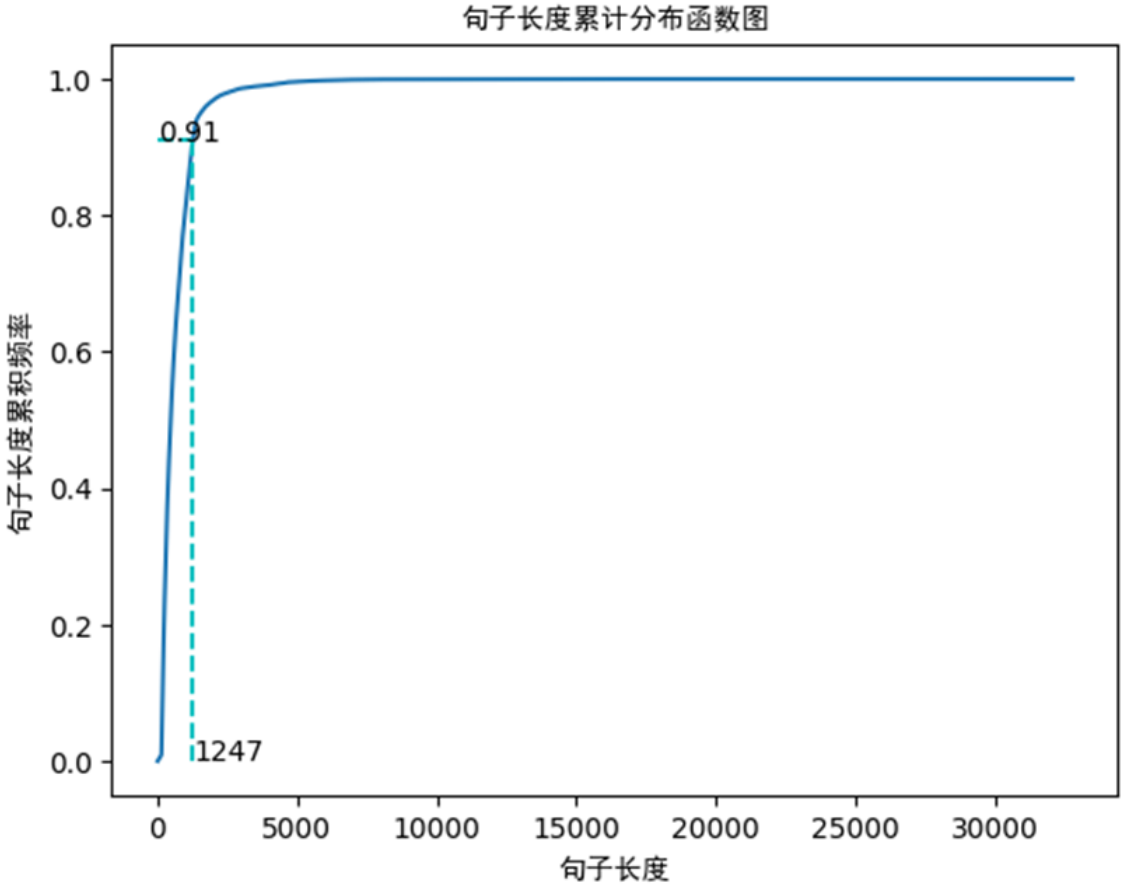


图 18 评论句子长度累计分布图

可以得到在评论中 91%以内的句子长度都在 1247 字以内，对之后的序列填充有一定的参考意义。之后按照研究简介中的代码进行操作，修改参数 batch_size 改为 32，训练轮次改为 20 轮，Embedding 降维改为 20，节点个数保持 100 不变。设置输入大小为 400。设置为 400 的原因是当设置为 1200 时程序计算过大，每一轮训练时间过长，最后的精确度不高。当输入大小为 400 时，训练时间适中，准确度在 50%左右。

接下来按照同样的方法进行字典构建以及 lstm 模型搭建，训练集测试集改为 1%. 具体代码如下所示:

```
import pickle
import numpy as np
import pandas as pd
from keras.utils import np_utils
from keras.utils.vis_utils import plot_model
from keras.models import Sequential
from keras.utils import pad_sequences
from keras.layers import LSTM, Dense, Embedding,Dropout
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# load dataset
# ['evaluation'] is feature, ['label'] is label
def load_data(filepath,input_shape=20):
    df=pd.read_excel(filepath)
```

```
    # 标签及词汇表
    labels,vocabulary=list(df['senti_sort'].unique()),list(df['Contents'].unique())
    # 构造字符级别的特征
    string=''
    for word in vocabulary:
        string+=word
    vocabulary=set(string)
    # 字典列表
    word_dictionary={word:i+1 for i,word in enumerate(vocabulary)}
    with open('word_dict.pk','wb') as f:
        pickle.dump(word_dictionary,f)
    inverse_word_dictionary={i+1:word for i,word in enumerate(vocabulary)}
    label_dictionary={label:i for i,label in enumerate(labels)}
    with open('label_dict.pk','wb') as f:
        pickle.dump(label_dictionary,f)
    output_dictionary={i:labels for i,labels in enumerate(labels)}

    # 词汇表大小
    vocab_size=len(word_dictionary.keys())
    # 标签类别数量
    label_size=len(label_dictionary.keys())
    # 序列填充，按 input_shape 填充，长度不足的按 0 补充
    x=[[word_dictionary[word] for word in sent] for sent in df['Contents']]
    x=pad_sequences(maxlen=input_shape,sequences=x,padding='post',value=0)
    y=[[label_dictionary[sent]] for sent in df['senti_sort']]

    y=[np_utils.to_categorical(label,num_classes=label_size) for label in y]
    y=np.array([list(_[0]) for _ in y])
    return x,y,output_dictionary,vocab_size,label_size,inverse_word_dictionary

# 创建深度学习模型， Embedding + LSTM + Softmax
def create_LSTM(n_units,input_shape,output_dim,filepath):
    x,y,output_dictionary,vocab_size,label_size,inverse_word_dictionary=load_data(filepath)
    model=Sequential()
    model.add(Embedding(input_dim=vocab_size+1,output_dim=output_dim,
                        input_length=input_shape,mask_zero=True))
    model.add(LSTM(n_units,input_shape=(x.shape[0],x.shape[1])))
    model.add(Dropout(0.2))
    model.add(Dense(label_size,activation='softmax'))
    model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
    plot_model(model,to_file='./model_lstm.png',show_shapes=True)
    # 输出模型信息
    model.summary()
    return model

# 模型训练
def model_train(input_shape,filepath,model_save_path):
    # 将数据集分为训练集和测试集，占比为 9：1
    # input_shape=100
    x,y,output_dictionary,vocab_size,label_size,inverse_word_dictionary=load_data(filepath,input_shape)
    train_x,test_x,train_y,test_y=train_test_split(x,y,test_size=0.01,random_state=42)
```

```
    # 模型输入参数，需要根据自己需要调整
    n_units=100
    batch_size=32
    epochs=20
    output_dim=20
    # 模型训练
    lstm_model=create_LSTM(n_units,input_shape,output_dim,filepath)
    lstm_model.fit(train_x,train_y,epochs=epochs,batch_size=batch_size,verbose=1)
    # 模型保存
    lstm_model.save(model_save_path)
    # 测试条数
    N= test_x.shape[0]
    predict=[]
    label=[]
    for start,end in zip(range(0,N,1),range(1,N+1,1)):
        print(f'start:{start}, end:{end}')
        sentence=[inverse_word_dictionary[i] for i in test_x[start] if i!=0]
        y_predict=lstm_model.predict(test_x[start:end])
        print('y_predict:',y_predict)
        label_predict=output_dictionary[np.argmax(y_predict[0])]
        label_true=output_dictionary[np.argmax(test_y[start:end])]
        print(f'label_predict:{label_predict}, label_true:{label_true}')
        # 输出预测结果
        print(''.join(sentence),label_true,label_predict)
        predict.append(label_predict)
        label.append(label_true)
    # 预测准确率
    acc=accuracy_score(predict,label)
print('模型在测试集上的准确率:%s'%acc)
```

观察模型得分：

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 400, 20)	171860
lstm (LSTM)	(None, 100)	48400
dropout (Dropout)	(None, 100)	0
dense (Dense)	(None, 3)	303

Total params: 220,563
Trainable params: 220,563
Non-trainable params: 0

图 19 评论模型得分

模型准确率如下图所示：

```
模型在测试集上的准确率:0.5606936416184971
```

图 20 评论模型准确率

接下来进行预测,具体代码如下：

```
import pickle
import numpy as np
from keras.models import load_model
from keras_preprocessing.sequence import pad_sequences

# 导入字典
with open('word_dict.pk', 'rb') as f:
    word_dictionary = pickle.load(f)
with open('label_dict.pk', 'rb') as f:
    output_dictionary = pickle.load(f)
try:
    # 数据预处理
    input_shape =400
    sent = "这个电影挺好看的，缺点就是不太好看，剧情垃圾，演员演技差，毁童年"
    x = [[word_dictionary[word] for word in sent]]
    x = pad_sequences(maxlen=input_shape, sequences=x, padding='post', value=0)

    # 载入模型
    model_save_path = './corpus_model1.h5'
    lstm_model = load_model(model_save_path)

    # 模型预测
    y_predict = lstm_model.predict(x)
    label_dict = {v: k for k, v in output_dictionary.items()}
    print('输入语句: %s' % sent)
    print('情感预测结果: %s' % label_dict[np.argmax(y_predict)])

except KeyError as err:
    print("您输入的句子有汉字不在词汇表中，请重新输入！")
print("不在词汇表中的单词为：%s." % err)
```

这里在 sent 里填充评论运行即可得到结果.

如例句中所示：″这个电影挺好看的，缺点就是不太好看，剧情垃圾，演员演技差，毁童年″

这句评论的的结果为：

```
1/1 [==============================] - 1s 949ms/step
输入语句: 这个电影挺好看的，缺点就是不太好看，剧情垃圾，演员演技差，毁童年
情感预测结果: -1
```

图 21 评论预测示例结果

4.8 Topsis 评分

将本文选取豆瓣当中新的电影的简介以及相关的评论经 LSTM 预测电影的简介以及评论的结果保存至一个新的文件中，并将该电影原有的评分同样添加到新的文件下。然后利用 topsis 算法，以简介、评论以及评分三个维度计算电影的综合得分，然后将其

与现有的电影同样进行相同操作的结果进行对比，一次来评价该电影是否是值得去观赏的。其中主要实现该过程的代码如下所示：

```
# 定义 position 接收需要进行正向化处理的列
position = np.array([0, 1, 2])
# 定义处理类型:1 - > 极小型 2 - > 中间型 3 - > 区间型
Type = np.array([2, 1, 3])
# 定义正向化函数
def positivization(x: np.array, pos: int, type: int) -> np.array:
    if type == 1:
        print(f"第{pos}列是极小型，正在正向化")
        x = x.max() - x
    elif type == 2:
        print(f"第{pos}列是中间型，正在正向化")
        best = 1 # 最佳值
        abs_max = np.max(np.abs(x - best))
        x = 1 - np.abs(x - best) / abs_max
    else:
        print(f"第{pos}列是区间型，正在正向化")
        left, right = 10, 20 # 区间的上界和下界
        max_tem = max(left - x.min(), x.max() - right)
        x = np.where(x < left, 1 - (left - x) / max_tem, x)
        x = np.where(x > right, 1 - (x - right) / max_tem, x)
        x = np.where(x > 1, 1, x)
    print(f"处理后的数据为:\n{x}")
    return x
for i in range(len(position)):
    print(f"当前处理的列为:\n{matrix[:, position[i]]}")
    matrix[:, position[i]] = positivization(matrix[:, position[i]], position[i], Type[i])
print(f"正向化后的矩阵为:\n{matrix}")
Z = matrix / np.sum(matrix * matrix, axis=0) ** 0.5
print(f"标准化后的矩阵为:\n{Z}")

max_score = np.max(Z, axis=0)
min_score = np.min(Z, axis=0)
max_dist = np.sum((max_score - Z) * (max_score - Z), axis=1) ** 0.5
min_dist = np.sum((min_score - Z) * (min_score - Z), axis=1) ** 0.5

final_score = (min_dist / (max_dist + min_dist))
final_score /= np.sum(final_score)
final_score = np.around(final_score, decimals=3) # 保留精度为 3
```

4.9 结果分析

4.9.1 简介结果

依据已有的电影评分，本文以 5.0 作为界限将电影分为了好电影和差电影，然后分别对这些电影的简介进行分析。通过 jieba 分词的词性标注，本文将能够一定程度上代表故事情节以及特定场景的普通名词、人名、成语等此进行提取。在去除停用词之后，对这些词语进行了词频统计，然后根据统计结果将其绘制出词云图，进而对比分析好电影与差电影之间的特定场景以及故事情节之间的差异。得到的结果如下所示：

图 22 烂片词长为 2 的词云

图 23 好片词长为 2 的词云

图 24 烂片词长为 3 的词云

图 25 好片词长为 3 的词云

图 26 烂片词长 4 以上的词云　　图 27 好片词长 4 以上的词云

从以上结果可以看出，无论是好的电影与差的电影大都是围绕家庭或者工作进行展开的。仅从 2 个字的词云来看，两者呈现的结果基本都是类似的，电影内容主要都是围绕着主角的人生进行展开的，通过一些特别的经历去展现与恋人的爱情、与朋友的友情等情感。从 3 字词云来看，提取的内容大都是拍摄影视相关的演员的名字。这反映出演员在电影评价中的重要性。比如，"古天乐"在电影圈拍摄了很多电影并且演技也非常出色，因此观众对他的评价也非常高；而像“郭德纲”“岳云鹏”他们则主要是从事相声行业的人员，他们的相声技巧极强，但是对于演技方面可能相对较为落后，正所谓“术业有专攻”。对于“孙悟空”这一传说形象，中国拍摄了许多有关该话题的电影，但情节大多相似，观众难免审美疲劳，也从侧面展现出中国电影缺乏更有活力的题材。对于 4 字以上的词语结果来看，烂片主要情节有“阴差阳错”、”啼笑皆非“、“发现自己”说明电影剧本的逻辑并不清晰，导致观众在观影过程中云里雾里。而好的电影则主要体现在”相依为命“、”前所未有“、”犯罪团伙“等词，这也说明这些电影的主要情节涉及到各种比较强烈的情感，这会让很多观众产生了情感上的共鸣；而“犯罪团体”体现出观众对于悬疑刺激的场景兴趣较高。

终上所述，一部优秀的电影应当具有清晰的逻辑，让观众在观影结束后仍感到困惑；另外，电影创作者应更注重电影情节中人物情感的表达，比如通过展现遭遇苦难的家庭之间的亲情、生死之际的爱情等，才更容易打动观众；大部分观众都是崇尚积极向上的，反映了人民对美好生活的向往，所以影片的内容也不能过于悲观。最后针对从事影视相关行业的演员，一位好演员对于影片呈现效果影响很大，因此导演也必须应该依据影片所展示的内容去选择恰当的演员。

4.9.2 评论结果

通过与电影简介类似的处理方式，依据已有的评分，本文将三星评论作为中评，三星以上的作为好评，三星以下的作为差评，然后分别对这些电影的评论进行分析。通过 jieba 进行分词处理，在去除停用词之后，对这些词语进行了词频统计，然后根据统计结果将其绘制出词云图，进而对比分析各类影评之间的差异。得到的结果如下所示：

1.好评图·

图 28 好评图词云结果

2.中评图

图 29 中评图词云结果

3.差评图

图 30 差评图词云结果

由上述结果可知，各类评论词云结果差别较大。在 2 个字的词云结果中，观众主要针对电影的“情感”、“风格”、“作品”、“画面”、“逻辑”、“结局”和“经历”等方面来做出的评价，说明观众对于影片本身的优缺点十分关注，观众往往会依据这些不同的方面对影片进行评论，例如：“情感” 描述了电影中所呈现的情感元素，比如亲情、爱情、友情等。观众会针对这些情感要素进行评论和感受。好影片往往更容易让观众共情，将自身融入到影片之中。“画面” 描述了电影中所呈现的视觉效果，包括电影的场景、背景、颜色等。观众会针对电影的画面给出评价，例如画面的美感和视觉效果的震撼力。“逻辑” 描述了电影在故事情节和结构上的严密程度和连贯度，观众会根据电影的逻辑性进行评论和评价，例如故事是否紧凑，逻辑是否清晰。

在 3 个字的不同的词云结果都出现了诸如“冯小刚”、“周星驰”、“孙悟空”、“刘亦菲”、“郭敬明”等字词，这些大多是演员、导演或者影片人物，这些关乎着电影制作人员的方方面面。

在四个字的词云结果中出现了部分影片名字，例如“大话西游”、“乘风破浪”和“后会无期”等；在差评和中评的词云结果中包含了许多类似于“莫名其妙”、“不知所云”、“乱七八糟”等的字词，说明部分电影存在着表述不清，剧情混乱等缺点。

4.9.3 得分结果

通过构建的 LSTM 模型，可以将新的电影的简介进行机器分类并赋予标签，同样也

可以将相关电影的评论进行机器赋分。将已有的电影同样进行相关的操作最终将它们的得分保存至新的文件下，通过 Topsis 算法，得到每部电影对于的得分进行比较。本文从动画电影的角度，选取了已经存在的“寻梦环游记”、“魁拔|||战神崛起”、“大耳朵图图”等九部以前的电影，它们当中有高评分好电影，也有低评分的烂片。另外，选取了即将上映的疯狂元素城的电影的简介以及评分进行预测，在进行综合评分结果如下：

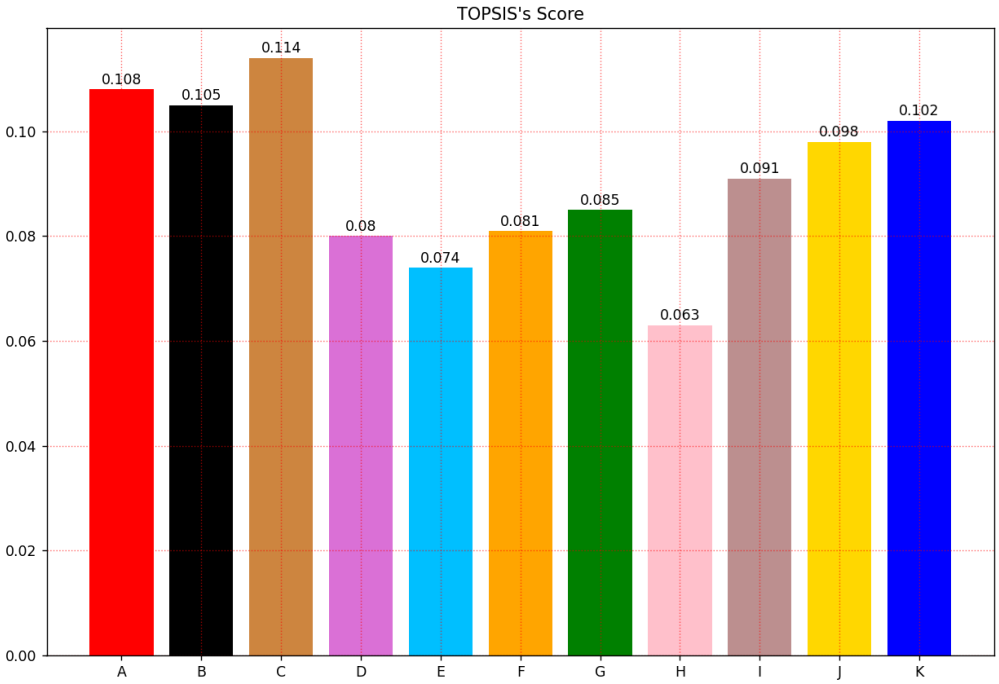


图 31 各影片综合得分情况

由于该影片暂未上映，因此对于该影片的评分目前还是没有确定的。因此先将其选定初值为分界值 5.0。蓝色的图案就是对应的综合评分结果。可以看出，虽然这部影视暂未上映但是根据机器得出来的结果比那些 5.0 以上的评分电影得出来的结果都要稍微高一些，因此观众还是可以对其抱有期待的。

5. 总结

这次实验中，为了充分体现本学期本课程所学到的知识，我们小组运用了爬虫技术、lda、snowNLP、lstm、jieba.posseg 和 tfidf 技术与方法。对文本数据进行爬取、清洗，分词、建立词云以及模型搭建和预测。对电影简介与评论完成了文本挖掘与情感分析。但是我们的实验也存在不足：首先，电影简介的数据量和电影评论的数据量相差较大。在爬取数据的过程中有一定的问题。其次，Lstm 模型的搭建不够充分，没有能够通过调参使得准确率上升。对模型运用与调参不够熟悉。另外，对于关键词提取方法的掌握不足，使得关键词内有部分意义不明的词语，后续还需要进一步完善停用词表或者选取更好的提取关键词的算法。最后，对于特点场景以及故事情节对比分析不够明显。

6.参考文献

CSDN 博客：

python 神经网络使用 Keras 进行模型的保存与读取---Python 无霸哥

自然语言处理(NLP)之使用 LSTM 进行文本情感分析-----IT 之一小佬

7.小组分工及心得体会

7.1 小组分工

1. 数据爬虫与数据预处理-李宗霖
2. 提取关键词构建词云-漆银权
3. 情感分析-何雷
4. lstm 模型搭建及预测-王瑞钰

7.2 心得体会

李宗霖

在文本分析与文本挖掘课程中，我学到了许多关于处理文本数据和提取有用信息的技术和方法。通过学习和实践，我对文本分析的流程和技术有了更深入的理解，并且掌握了一些实用的工具和技巧。

首先，我认识到文本数据具有丰富的信息和潜在的价值。通过合理的数据获取和处理，可以从文本数据中挖掘出有关观众喜好、评价标准和影片质量的关键信息。这对于电影推荐、市场分析和决策制定等方面具有重要意义。

其次，我学会了使用爬虫技术从网络上获取电影简介和评论数据。通过分析网页结构和利用相应的工具，我能够编写爬虫代码来自动化获取大量的文本数据。这为后续的分析和挖掘提供了数据基础。

在数据清洗和预处理方面，我了解了处理空值、异常值和重复值的常用方法。清洗数据可以提高后续分析的准确性和可靠性，并减少对无效数据的影响。我学会了使用 Python 编程语言和相关库来处理和转换文本数据，使其适合于后续的分析任务。

在文本挖掘和情感分析方面，我学习了如何提取关键词、特征和情感倾向等信息。通过使用自然语言处理技术和机器学习算法，我能够分析文本数据中的情感和观点，并提取出对电影评价重要的特征。这使我能够更好地理解观众对电影的喜好和评价标准。

最后，我学习了如何建立预测模型和评分算法来预测和评估电影的质量和受欢迎程度。通过使用 LSTM 模型和 Topsis 算法，我能够对电影的简介、评论和评分进行预测和综合评估。这些模型和算法为电影推荐和决策提供了有力的支持，帮助观众更好地选择适合自己的影片。

通过这门课程，我不仅学到了理论知识和技术方法，还锻炼了分析问题和解决问题的能力。我深入了解了文本分析与文本挖掘的实际应用，认识到了它在各个领域中

的重要性和潜力。这门课程为我今后在数据分析和人工智能领域的发展打下了坚实的基础，并提供了丰富的经验和启示。我相信在今后的工作中，能够运用所学知识，更好地应对和解决实际问题。

漆银权

通过本次文本挖掘可以发现自己还存在很多的不足之处。比如在开始做之前的预期很美好，但是当自己真正做起来就会发现会存在很多问题。比如就拿电影类型分类来说，要实现文本的聚类相较于用数值进行聚类是非常的不容易实现。起初通过对于电影的简介利用机器学习的方式对其进行关键特征提取，然后将电影名称作为标签进行聚类，但是聚类的结果是关键特征的提取是对于整个句子当中一部分提取的，而且很多的电影都没有被很好的的分开，所以，最终还是选择使用电影的评分进行分类。

然后，在关键词的提取过程中会发现很多提取的结果也是与自己所想的并不相同。这可以是自己对于相关算法使用并不是很熟悉，或者没有找到更好的算法。很多的理论知识并没有进一步去理解其中所包含的逻辑，只是一味地套用模型进行分析，缺乏自己的主观认识，这是非常糟糕的。但是很多时候我也会问机器学习的结果一定就比人工进行分析的结果更好的，但是人工进行的分析在很多时候会被看起来很低级，而且会缺乏一定的信服力，因为存在很大的主观意识。所以，归根结底还是学艺不精，后续还需要不断完善自己的能力。

最后，通过这个完整的实验下来，确实也是收获了不少东西的。最基本的就是对于数据清洗分词的流程以及较为熟悉了。在遇到问题的时候，积极寻找各种解决措施，当结果成功实现的时候，特别是结果还是比较符合预期的时候也会会收获一定的满足感的。虽然我们的成果并不是很完善，但是对于基本的情节的筛选还是较为符合现实的，只是在进行算法推荐的时候，还是比较模糊的，因为建立的模型的精度还是不算高的，也会存在一定的误判。因此，想要更好实现这一功能要么继续优化，要么寻找更好的模型。一套流程下来，我们也对数据分析有了一个更好的认识，这对于以后遇到相同工作还是有一定的帮助的。

何雷

在学习文本分析课程期间，我们小组深入了解了如何利用豆瓣电影数据进行文本挖掘和情感分析，以便更好地理解观众对电影的评价和喜好。通过对电影简介和评论的文本进行处理和分析，我们能够提取出关键信息并揭示高分与低分电影之间的差异。

首先，我们学习了如何通过文本挖掘方法从豆瓣海量的电影简介中提取特定的内容和故事情节。通过对简介文本的处理和特征提取，我们能够挖掘出与观众喜好密切相关的关键词和句子。这有助于为观众提供一个初步的电影筛选依据，使他们能够通过阅读简介来判断电影是否符合自己的兴趣和口味。

其次，我们学习了情感分析的方法，并应用于电影评论数据。通过情感分析，我们能够了解观众对电影的评价以及他们在评论中关注的方面。通过提取评论中的高频词和情感倾向，可以更好地理解观众对电影的评价标准。这有助于深入挖掘观众对电影的态度和情感，并找出影响电影受欢迎或不受欢迎的原因。在针对评论进行情感分析，尝试去使用多种方式进行情感分析。但是很多有关情感分析的库使用后，结果都差不多，精确度不高，难以达到预期效果。

通过词典进行情感分析的效果就显然比通过 snownlp 进行情感分析效果要更好。

通过文本分析课程的学习，我深刻认识到文本数据的重要性以及如何利用各种技术方法从中挖掘有价值的信息。本次大作业也是对自身能力的一种锻炼，通过对豆瓣电影数据的分析和挖掘，我能够为观众提供更准确和全面的电影推荐，使他们能够更好地选择适合自己的影片。

王瑞钰

在文本分析与文本挖掘大作业中，我和漆银权同学一起确定了选题。然后我选择了 lstm 模型这一部分进行实验，在这过程中又许多的困难，尤其是 lstm 模型的搭建、存储、加载。其中的大多数问题都是通过 csdn 的相关博客解决，同时也包括和小组同学的诸多讨论。在这其中最大的问题是训练以后的模型存储以及后续进行加载预测的代码过程。同时也面临着数据处理不到位的问题，尤其是许多爬取到的数据有大量的数字以及标点、特殊符号。这使得在构建模型时报错。最后我们还是搭建出来了两个可以运用的模型，对简介以及评论的建模比较成功。我这一步的构建离不开前面每一位同学的贡献。只有前面同学对数据处理得当我才能过成功建模。才能够预测，虽然预测精度不够高，简介训练集的条数不够多。对模型运用与调参不够熟悉。都是现在我面临的问题。

通过这一整套的实验流程下来，我对文本挖掘与文本分析的流程更加了解，也学会了 lstm 的代码，尤其是关于模型存储以及加载。同时也让我意识到到，并不是研究思考就能成功的,只有亲自实践才能成功。许多的知识只有使用过后，才能够成为自己的武器。理论上的顺利，往往在实践中会被击破。只有理论与实践一起，才能真真正正的实现掌握。

接下来，如果还能够继续学习与运用相关知识的话，我将会从模型参数以及其它相关模型上继续努力，将现阶段的空缺争取填补完整。阅读有关论文，多看看别人是如何对类似情况进行处理。这次实践的完结不代表着对文本挖掘以及文本分析的终结，更是一个新的台阶。这也为我在以后的学习生活就业中有着极大的帮助。研究中存在的错漏之处希望老师指正，谢谢老师。