中南大學
CENTRAL SOUTH UNIVERSITY

《Python 程序设计》

实验报告

题　　目：	Python 课程实验报告
学生姓名：	孙重豪
学　　号：	8207191520
指导教师：	周盛超
学　　院：	自动化学院
专业班级：	智能科学与技术 1902 班

2023 年 1 月

目 录

第 1 章　实验一

1.1 实验简介

1.1.1 实验题目

计算 1000 以内的所有水仙花数，并且打印出来。

1.1.2 实验环境

1) 华为 Matebook 13 电脑、Windows 11 操作系统
2) Visual Studio Code 1.74.2
3) Python 3.9.12

1.2 实验流程

1.2.1 题目分析

首先我们应该知道，水仙花数是指一个数的百位数上数字的立方加十位数上数字的立方加个位数上数字的立方等于这个数的本身的数，因此本题思路就是循环遍历 1000 以内的所有数字，通过 python 的除法和取余操作得到每一位上的数字，求出每一位数字的立方和并与该数字本身进行比较，若相等，则该数字为水仙花数，将其打印在屏幕上。

1.2.2 实验过程

详细代码：

```
print('1000 以内的水仙花数：')
for i in range(100, 1000):
    a = int(i/100)  # 求出 i 的百位数
    b = int((i % 100)/10)  # 求出 i 的十位数
    c = int(i % 10)  # 求出 i 的个位数
    if i == (a**3+b**3+c**3):
        print(i)
```

1.3 实验结果及分析

结果输出：

```
1000以内的水仙花数:
153
370
371
407
```

根据结果可知，1000 以内的水仙花数有：153、370、371、407。

第 2 章 实验二

2.1 实验简介

2.1.1 实验题目

给定一个长度为 n 的整数数组，你的任务是判断在最多改变 1 个元素的情况下，该数组能否变成一个非递减数列。非递减数列定义如下：对于数组中所有的 i (1 <= i < n)，满足 array[i] <= array[i + 1]。

2.1.2 实验环境

1) 华为 Matebook 13 电脑、Windows 11 操作系统
2) Visual Studio Code 1.74.2
3) Python 3.9.12

2.2 实验流程

2.2.1 题目分析

首先比较序列相邻元素的大小，设定一个 flag 记录前一个元素大于后一个元素的次数，然后遍历整个序列，若 flag>1 则说明无法通过只改变一次元素使序列变成非递减序列，若 flag<=1，则判断 i 是否是第一个数或者倒数第二个数，是的话则程序继续，不是的话则判断是否满足 lst[i] <= lst[i+2] 或者 lst[i-1] <= lst[i+1]，若满足则说明可以通过只改变一次元素使序列变成非递减序列，若不满足则返回 False，若整个序列通过循环验证，则返回 True，说明该序列可以通过只改变一次元素使序列变成非递减序列。

2.2.2 实验过程

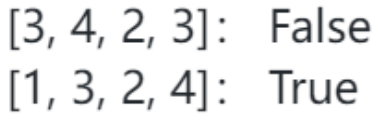
详细代码：

```
def check(lst):
    flag = 0
    for i in range(len(lst)-1):
        if lst[i] > lst[i+1]:
            flag += 1

        if flag > 1:
            return False
        if i in [0, len(lst)-2] or lst[i] <= lst[i+2] or lst[i-1] <= lst[i+1]:
            continue
        return False
    return True
print('[3, 4, 2, 3]：',check([3, 4, 2, 3]))
print('[1, 3, 2, 4]：',check([1, 3, 2]))
```

2.3 实验结果及分析

结果输出：



根据结果可知，序列 [3, 4, 2, 1] 无法通过只改变一次元素变成非递减序列；而序列 [1, 3, 2, 4] 可以通过改变元素 3 或者 2 变成非递减序列。

第 3 章 实验三

3.1 实验简介

3.1.1 实验题目

实验 3. 约瑟夫生死游戏: 30 个人在一条船上，超载，需要 15 人下船。于是人们排成一队，排队的位置即为他们的编号。报数，从 1 开始，数到 9 的人下船。如此循环，直到船上仅剩 15 人为止，请编程求出都有哪些编号的人下船。

3.1.2 实验环境

1) 华为 Matebook 13 电脑、Windows 11 操作系统
2) Visual Studio Code 1.74.2
3) Python 3.9.12

3.2 实验流程

3.2.1 题目分析

首先设定 flag = 0，令其记录报数的数字，设定 nums = 30，令其记录船上现存的人数，设定 n 记录循环移动的位置，设定一个有 31 个元素‘1’的列表 lst 来记录该位置的船员是否下船，‘1’代表在船上，‘0’代表已经下船了。设置一个 while 循环并进行下列操作：检查 lst 表该位置是否为‘1’，若为‘1’则说明该位置的船员还在船上，报数 flag = flag +1，如果 flag 等于 9，则该位置船员下船，即令 lst[n] = 0，同时将 flag 置为 0，船员人数 nums 减 1，并且输出几号下船，船上还有几人等信息。在循环过程中，如果 n 等于 31 则将 n 置为 1。

3.2.2 实验过程

详细代码：

```
flag = 0
nums = 30
n = 1
lst = [1 for i in range(32)]

while nums > 15:
    if lst[n] == 1:
        flag = flag + 1
    if flag == 9:
        flag = 0
        lst[n] = 0
        nums = nums -1
        print('{}号下船了，船上还有{}人'.format(n, nums))
    n = n + 1
    if n == 31:
        n = 1
```

3.3 实验结果及分析

结果输出：

9号下船了，船上还有29人
18号下船了，船上还有28人
27号下船了，船上还有27人
6号下船了，船上还有26人
16号下船了，船上还有25人
26号下船了，船上还有24人
7号下船了，船上还有23人
19号下船了，船上还有22人
30号下船了，船上还有21人
12号下船了，船上还有20人
24号下船了，船上还有19人
8号下船了，船上还有18人
22号下船了，船上还有17人
5号下船了，船上还有16人
23号下船了，船上还有15人

根据结果可知，下船的人的编号依次是：9，18，27，6，16，26，7，19，30，12，24，8，22，5，23。

第 4 章　实验四

4.1 实验简介

4.1.1 实验题目

网络爬虫

1、爬取所有豆瓣电影评分 Top250 的电影的信息

a) 正文链接
b) 英文名（如有），中文名
c) 等等

2、获取每部影片的简介和影评

3、加分项

a) 不限于豆瓣的简介，影评
b) 是否分析了演员与电影类型的关联关系？
c) 是否分析了演员与演员的关系？
d) 是否对简介和影评进行词云分析？
e) 等等

4.1.2 实验环境

1) 华为 Matebook 13 电脑、Windows 11 操作系统
2) Visual Studio Code 1.74.2
3) Python 3.9.12（requests==2.28.1、pandas==1.2.4、beautifulsoup4==4.11.1、tqdm==4.64.0、jieba==0.42.1、numpy==1.24.1、pyecharts==1.9.1）

4.2 实验流程

4.2.1 题目分析

首先本题使用 requests 库进行网页内容的爬取，使用 Beauti4oup4 库和正则表达式相结合的方式进行网页内容的解析。由于豆瓣电影 Top250 共有 25 页每页 10 部电影，且每一页的链接是有规律的，因此可以循环遍历每一页，但是每页的内容有限，我们只能爬取'电影中文名'、'电影英文名'、'链接'、 '年份'、'地区'、'剧情类型'、 '导演'、 '评分'、 '评论人数'、'概述'等信息，而'电影简介'和'短评'则需要进入电影链接内部进行爬取，针对电影短评，因为我们后面要使用短评数据进行文本分析，因此我们需要爬取大量短评，本实验尝试爬取 100 条短评数据，由于每页只有 20 条短评，所以我们可以参考上述方法循环遍历五页获取 100 条短评信息，最终我们将上述爬取到的所有电影信息保存为 csv 文件以便于后续使用读取。

除此之外，本实验对爬取到的简介和影评进行词云分析，主要是使用 jieba 库进行分词，使用 pyecharts 库进行词云的绘制。首先使用 Python 的 join()方法将简介和影评拼接到一起，然后使用 jieba 模块对其进行分词，然后使用 jieba.analyse.extract_tags()方法列出出现频率最高的前 40 个词语，同时要加载停用词文件去除结果中的类似“这个”、“这部”、“一个”等常用高频词，最后从 pyecharts.charts 模块中 WordCloud 对高频词语进行词云绘制，并将词云结果保存为 html 文件方便查看。

4.2.2 实验过程

1、网络爬虫代码：

```
import time
import requests
import re
import pandas as pd
from tqdm import tqdm
from bs4 import BeautifulSoup as bs
import requests,random

class Top250:
    def __init__(self):
        #起始地址
        self.start_url = 'https://movie.douban.com/top250'
        #请求头，浏览器模拟
        self.headers = {
            'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36',
        }
        #爬取页数
        self.page_num = 10

    def get_page_url(self):
        n = 0 #第一页开始,下标 0
        yield self.start_url+'?start={}&filter='.format(n*25)

    def getHtml(self):
        gu = self.get_page_url() #url 生成器
        for url in gu:
            html = requests.get(url,headers=self.headers).text
            yield html

    def getsubHtml(self,url):
        r = requests.get(url,headers=self.headers)
        soup = bs(r.text, "html.parser")
        summ = ''.join(soup.find('span',property="v:summary").text.split())
        return summ

    def getshort(self,link):
        lst=[]
        for i in range(5):
```

```
            kv = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36'}
            url =
link+'comments?start={}&limit=20&status=P&sort=new_score'.format(i*20)
            r = requests.get(url, headers = kv)
            soup = bs(r.text, "html.parser")
            for sp in soup.find_all('div',class_="comment"):
                short = sp.find('span',class_="short").text.replace('\n','').replace('\r','')
                if re.findall(r'star\d{1}0', str(sp)):
                    star = re.findall(r'star\d{1}0', str(sp))[0][-2]
                else:
                    star=None
                lst.append([short,star])
        return str(lst)
#电影数据提取
    def getData(self):
        lst=[]
        gh = self.getHtml() # html 源码生成器
        for html in gh: # html:网页源码
            soup = bs(html, 'lxml')
            for info in tqdm(soup.find_all('div', class_='info')):
                c_name = info.find('span',class_='title').text.strip() #  电影中文名
                try:
                    e_name = info.find_all('span',class_='title')[1].text[3:]
                except:
                    e_name = 'None'
                message = info.select('div.bd p')[0].text.strip() #导演、主演、年份、
地区信息
                yat = re.search('[0-9]+.*\/?', message).group().split('/') #年份、地区、
类型
                year,area,typ = yat[0][:4],yat[1],yat[2]#得到年份、地区、类型
                da = re.search('导演.+\s',message).group().strip()+'...'
                director = re.findall('导演:(.+?)\s',da)[0].strip() #导演
                link = info.find('a')['href']
                try:
                    summary = self.getsubHtml(link)
                except:
                    summary = '无简介'
                mark_info = info.find('div',class_='star')
                score= mark_info.find('span',class_='rating_num').text.strip()#评分
                count = re.search('[0-9]+',mark_info.select('span')[3].text).group() #评
价人数
```

```
                short = self.getshort(link)
                try:
                    quote = info.select('p.quote span')[0].text.strip()
                except IndexError:
                    quote = '无概述'
                lst.append([c_name,e_name,link,year,area,typ,director,score,count,quo
te,summary,short])
        return lst

# 保存到 csv 文件
    def saveTocsv(self,file_name):
        name = ['电影中文名','电影英文名','链接', '年份', '地区', '剧情类型', '导演', '
评分', '评论人数', '概述', '简介', '短评']
        lst = self.getData()
        test = pd.DataFrame(columns=name,data=lst)
        test.to_csv(file_name,index=False)

if __name__ == '__main__':
    start = time.time()
    top = Top250()
    try:
        top.saveTocsv('top250_11.csv')
        print('抓取成功,用时%4.2f'%(time.time()-start)+'秒')
    except Exception as e:
        print('抓取失败,原因:%s'%e)
```

2、词云代码

```
import jieba
import jieba.analyse
import copy

n=9
lst=[]
name = df['电影中文名'][n]
lst.append(df['简介'][n])
short = eval(df['短评'][n])
for i in range(len(short)):
    lst.append(short[i][0])
ss=''.join(lst)
tags = jieba.analyse.extract_tags(ss,topK=40,withWeight=True)
#去停用词
f = open('./stopwords.txt',mode='r',encoding='UTF-8')
```

```
lsst = list(f)
f.close()
lsst = [i.strip() for i in lsst]
tagss = copy.deepcopy(tags)
for i in range(len(tagss)):
    if tagss[i][0] in lsst:
        tags.remove(tagss[i])
print(tags)
#绘制词云
import pyecharts.options as opts
from pyecharts.charts import WordCloud

(
    WordCloud()
    .add(series_name=name, data_pair=tags,
word_size_range=[25,80],width=1200,height=600)
    .set_global_opts(
        title_opts=opts.TitleOpts(
            title=name, title_textstyle_opts=opts.TextStyleOpts(font_size=40)
        ),
        tooltip_opts=opts.TooltipOpts(is_show=True),
    )

    .render(name+".html")

)
```

4.3 实验结果及分析

1、爬取结果截图：

	A	B	C	D	E	F	G	H	I	J	K	L
1	电影中文名	电影英文名	链接	年份	地区	剧情类型	导演	评分	评论人数	概述	简介	短评
2	肖申克的救赎	hawshank Reden	douban.com/subj	1994	美国	犯罪 剧情	弗兰克·德拉邦特	9.7	2762864	希望让人自由。	下帮助典狱长洗黑	型片元素大集成，
3	霸王别姬	None	douban.com/subj	1993	中国大陆 中国香港	剧情 爱情 同性	陈凯歌	9.6	2046113	风华绝代。	段小楼深知戏非	到园内，怎知春色
4	阿甘正传	Forrest Gump	douban.com/subj	1994	美国	剧情 爱情	罗伯特·泽米吉斯	9.5	2071770	一部美国近现代史。	莱特·潘饰），在	现反对这种投机倒
5	泰坦尼克号	Titanic	douban.com/subj	1997	墨西哥 澳大利亚 加	剧情 爱情 灾难	詹姆斯·卡梅隆	9.5	2032311	去的才是永恒的。	虚伪的生活，不愿	的诸多桥段看起
6	这个杀手不太冷	Léon	douban.com/subj	1994	法国 美国	剧情 动作 犯罪	吕克·贝松	9.4	2210407	小萝莉不得不说	全家的惩罚。马蒂	眼标题', '5'], ["他
7	美丽人生	La vita è bella	douban.com/subj	1997	意大利	剧情 喜剧 爱情 战	罗伯托·贝尼尼	9.6	1273320	最美的谎言。	聪明乐天的圭多	那些陪着他在儿子
8	千与千寻	千と千尋の神隠し	douban.com/subj	2001	日本	剧情 动画 奇幻	宫崎骏	9.4	2146254	宫崎骏，最好的久	工作才能不被魔	无脸男。', '5'], ['不
9	辛德勒的名单	Schindler's List	douban.com/subj	1993	美国	剧情 历史 战争	史蒂文·斯皮尔伯格	9.6	1061577	人，就是拯救整	的大屠杀，辛德勒	焚尸堆死掉的红衣
10	盗梦空间	Inception	douban.com/subj	2010	美国 英国	剧情 科幻 悬疑 冒	克里斯托弗·诺兰	9.4	1980461	我们一场无法盗	伯特·费希尔（希	情节拖沓，nolan的
11	星际穿越	Interstellar	douban.com/subj	2014	美国 英国 加拿大	剧情 科幻 冒险	克里斯托弗·诺兰	9.4	1719946	让我们超越时空	感内前NASA成员	上帝创造了诺兰',
12	楚门的世界	The Truman Show	douban.com/subj	1998	美国	剧情 科幻	彼得·威尔	9.4	1610741	到你，祝你早安，	方，但总因种种理	自己的生命。', '5']
13	忠犬八公的故事	Hachi: A Dog's Tal	douban.com/subj	2009	美国 英国	剧情	莱塞·霍尔斯道姆	9.4	1348075	不能忘记你所爱	把他送到车站，	再醒来了，够了，
14	海上钢琴师	nda del pianista su	douban.com/subj	1998	意大利	剧情 音乐	朱塞佩·托纳多雷	9.3	1613646	自己坚定了的路，	祖杰尼听说了190	还是很棒的，只不
15	三傻大闹宝莱坞	3 Idiots	douban.com/subj	2009	印度	剧情 喜剧 爱情 歌	拉库马·希拉尼	9.2	1790610	憨豆，高情商版谢	，甚至还公然顶撞了一遍', '4'], ['还不
16	放牛班的春天	Les choristes	douban.com/subj	2004	法国 瑞士 德国	剧情 音乐	克里斯托夫·巴拉涅	9.3	1259666	童声，是最接近上	乐作品，组织合	的纸飞机和老师带
17	机器人总动员	WALL·E	douban.com/subj	2008	美国	科幻 动画 冒险	安德鲁·斯坦顿	9.3	1265829	小瓦力，大人生。	但随着时间的流逝	拒绝的，但里面的
18	无间道	無間道	douban.com/subj	2002	中国香港	剧情 犯罪 惊悚	刘伟强	9.3	1296302	影史上永不过时的	身背重压，刘健明	人难过。', '4'], ['港

电影中文名,电影英文名,链接,年份,地区,剧情类型,导演,评分,评论人数,概述,简介,短评
肖申克的救赎,The Shawshank Redemption,https://movie.douban.com/subject/1292052/,1994, 美国 , 犯罪 剧情,弗兰克·德拉邦特,9.7,2762864,希望让人自由。,一场谋杀案使银行家安迪（蒂姆·罗宾斯TimRobbins饰）蒙冤入狱，谋杀妻子及其情人的指控将囚禁他终生。在肖申克监狱的首次现身就让监狱“大哥”瑞德（摩根·弗里曼MorganFreeman饰）对他另眼相看。瑞德帮助他搞到一把石锤和一幅女明星海报，两人渐成患难之交。很快，安迪在监狱里大显其才，担当监狱图书管理员，并利用自己的金融知识帮助监狱官避税，引起了典狱长的注意，被招致麾下帮助典狱长洗黑钱。偶然一次，他得知一名新入狱的小偷能够作证帮他洗脱谋杀罪。燃起一丝希望的安迪找到了典狱长，希望他能帮自己翻案。阴险伪善的狱长假装答应安迪，背后却派人杀死小偷，让他唯一能合法出狱的希望泯灭。沮丧的安迪并没有绝望，在一个电闪雷鸣的风雨夜，一场暗藏几十年的越狱计划让他自我救赎，重获自由！老朋友瑞德在他的鼓舞和帮助下，也勇敢地奔向自由。本片获得1995年奥...,"[['人的生命不过是从一个洞穴通往另一个世界..然后在那个世界的雨中继续刺斗.i hope', '4'], ['当年的奥斯卡颁奖礼上，被如日中天的《阿甘正传》掩盖了它的光彩，而随着时间的推移，这部电影在越来越多的人们心中的地位已超越了《阿甘》。每当现实令我疲意得产生无力感，翻出这张碟，就重获力量。毫无疑问，本片位列男人必看的电影前三名！回顾那一段经典台词：“有的人的羽翼是如此光辉，即使世界上最黑暗的牢狱，也无法长久地将他围困！“ ', '5'], ['不需要女主角的好电影', '5'], ['恐惧让你沦为囚犯，希望让你重获自由。——《肖申克的救赎》', '5'], [' “这是一部男人必看的电影。”人人都这么说。但单纯从性别区分，就会让这电影变狭隘。《肖申克的救赎》突破了男人电影的局限，通篇几乎充满令人难以置信的温馨基调，而电影里最伟大的主题是“希望”。当我们无奈地遇到了如同肖申克一般囚禁了心灵自由的那种囹圄，我们是无奈的老布鲁克，灰心的瑞德，还是智慧的安迪？运用智慧，信任希望，并且勇敢面对恐惧心理，去打败它？经典的电影之所以经典，因为他们都在做同一件事——让你从不同的角度来欣赏希望的美好。', '5'], ['策划了19年的私奔……', '5'], ['关于希望最强有力的注释。', '5'], ['有种鸟是关不住的.', '5'], ['超级喜欢超级喜欢,不看的话人生不圆满.', '5'], ['Fear Can Hold You Prisoner, Hope Can Set You Free', '5'], ['这样的男人谁会舍得背叛。。。', '5'], ['一部没有爱情与美女的电影,却光芒四射', '5'], ['Hope is a good thing, and maybe the best thing of all.', '4'], ['没有人会不喜欢吧！书和电影都好。', '5'], ['因为1994年台湾引进了一部比较卖座的老片The Sting，被错译成了《刺激》。到了1995年本片上映时，片商觉得其剧情与《刺激》有类似的地方（大概都属于高智商的复仇？），因此被译成了《刺激1995》，1998年又有一部片子Return To Paradise因为含有牢狱情节，被译成《刺激1998》！', '5'], [' “你知道，有些鸟儿是注定不会被关在牢笼里的，它们的每一片羽毛都闪耀着自由的光辉。“ 一部经得起时间检验的电影，豆瓣、时光网、IMDb各大电影网站都排名第一的电影。', '5'], ['不愧是好莱坞大片，一环扣一环，没想到最后不但逃生生天还偷偷把典狱长的贪污转移了出来，证据也送达了出去，帮助了曾经关心过自己的人，情节安排的巧妙，演员演绎的也好，是美国的，有爱国情节的我给打四分', '4'], ['一部没有女主，没有爱情，没有特技的好片子，也是很多大学英语老师喜欢放给学生看的片子，这部片子讲诉的就是一个很纯粹的故事，一次的救赎，换回了自由，也是一种信念驱使他这么做，这部片子是感人的，它也让很多人成长。', '5'], ['还记得老brooks上吊自杀时心里有多难过。自由来的太晚，生命早已自行放弃。', '5'], ['这部电影没什么好说的，史上最完美的电影，没有一秒尿尿时间！', '5'], ['hope is a good thing', '5'], ['绝对是期待过高导致有点失望', '4'], ['对不起，我戒酒了。', '5'], ['越狱我感觉就是改编自这个', '5'], ['肖申克中生存体制又如当今社会体制，“痛改前非”的人一次次被重新投入黑暗之中，当权者干着肮脏的勾当却会在公众面前宣讲“拯救之道”，似乎只有当性格没有棱角，思想失去火花，被“体制化”的人们习惯逆来顺受的时候才可以平安生存', '5'], ['通过下水道逃出去，这真的是太震撼了，不得不说男主对自由的执念克服了一切，即使待在监狱里，因为不断学习，没和外面的世界脱轨，他才能如此从容面对一切，甚至即使是在监狱这小小世界里自己也是主宰，所以说：学习很重要，学到老活到老。', '4'], ['在我的心目中,它一直都是最被高估的电影。', '3'], ['这是我每年都要回顾的一部片子，每当心情低落，人生低谷的时候，或者随便什时候打开，听着摩根弗里曼低沉的声音娓娓道来故事，就会感觉好了很多。好的电影是好的人生陪伴', '5'], ['明修栈道，暗渡陈仓', '5'], ['好片，但是远比这部片更好的还很多，大家酸天把这部电影稀奇得不行，其实同年的电影我更喜欢阿甘。', '5'], ['看完让人很振奋', '5'], ['几年后，五星改成三星。', '3'], ['若你试自由如生命，你愿意用20年等来一日的逃脱。', '5'], ['何为救赎？只有重燃人们心中的希望才能救赎。有希望才有一切可能。即使环境再恶劣，安迪都没有放弃，坚持不懈的写信为大家换来更好的图书馆，帮助有需要的人拿到高中文凭使他们可以在社会上有生存能力，就算知道会惹怒看守也想让人们都听到美妙的歌声，点燃希望之光。有一种鸟儿是永远也关不住的。', '5'], ['怒赞，很难有一部电影能比《肖申克的救赎》更好的诠释梦想与救赎这两个词的关联，电影予人带来心理的洗涤震撼是如此深刻，对比安迪，我们生活中看似无以能迈不过的坎又算什么？当你若能一直心拥梦想，哪怕失败，也定能获得希望的救赎。', '5'], ['救赎什么？灵魂？自由？信念？你看看这部电影就有体会了', '4'], ['酣畅淋漓', '5'], ['太感人的电影！完整看过四遍！安迪从下水道爬出后，在水里在雨中仰天长啸时，就瞬间泪奔！', '5'], ['发现自己最近真的是愿意回看经典 尤其是没有女主的经典 自由与希望 多么美好的词汇 可是一个迂腐的人现在才体会到 这两个词如果一年前对我提起 那么我会表示鄙夷 会觉得这两个词太虚无缥缈 假大空 而现在 如果你向我提起 那么我会说 我爱自由和希望', '5'], ['这是一部关于自由、友情、正义的电影！“信念不灭，希望永存”。这其实是他们全体人的救赎，只是选择的方式不同而已。', '5'], ['高中第一节心理课。“有一种鸟是关不住的，因为它们的每一片羽毛都闪耀着自由的光辉”，但后来不幸成了我的逆反宣言……哈哈哈。', '5'], ['我最喜欢的电影', '5'], ['非常好，可惜那年有阿甘正传，但我个人认为还是这部最佳。', '5'], ['不多说了', '5'], ['这部被称为《刺激1995》的影片在中国影迷间也有极好的口碑，可见电影超越国界的神奇之处。', '5'], ['真的很金典，每个人心里都有渴望不一样的自由，却又不得不被现实生活所束缚，很矛盾的个体，希望每个人能像安迪一样找到获得真正的自由。很喜欢摩根弗里曼', '4'], ['其实理解不了为什么这部影片会排名第一...', '4'], ['很久之前看的了，印象深刻的是一位老人在出狱后不知道如何生活了，外面的一切都是他所不熟悉的，无法正常的工作，交流，生活。最终选择结束了自己的生命。习惯真的是一种极其强大的可怕的力量！', '5'], ['几年前被母亲推荐看的，她很推崇，看过后感觉可以，黑暗是每个国家都存在的，总的来说希望好人一生平安！', '5'], ['习惯了监狱生活而接受不了正常生活的人，既真实又恐怖。勇敢面对生活，逃离自己的舒适区。', '5'], ['#重看#@影城；超越年代的“主流心灵鸡汤”依然灌得眼泪直冒；世上本无希望，若我多行一步即是希望，在绝望黑暗处生出倔强微光，卡夫卡式的悖论；每人都背负着有形无形的肖申克，注定无法被囚禁的鸟儿一定会挣脱桎梏，天才善于做梦；类型片元素大集成，充满匠气的完美作品。', '5'], ['2008年11月28日，第一次看肖申克。那时候观影量还不到五百吧。给五星是见识浅。现在差不多九年过去，我在豆瓣标注看过的已经超过三千，第二次看，还是给了五星。它当然不是最好

2、词云结果

星际穿越

辛德勒的名单

霸王别姬

第 5 章　实验五

5.1 实验简介

5.1.1 实验题目

机器学习

1、实现对获取的电影数据的统计分析

a) 可以考虑类型、语言、地区或演员等特征维度
b) 可以考虑对简介，影评进行语义分析出来的结果进行统计
c) 绘制相关图形

2、实现某种分类算法（随意），用于测试某种分类

a) 例如通过简介或影评对电影的类型分类；
b) 例如通过演员的组合分类电影，亦或是反过来；
c) 等等

5.1.2 实验环境

1) 华为 Matebook 13 电脑、Windows 11 操作系统
2) Visual Studio Code 1.74.2
3) Python 3.9.12（pandas==1.2.4、snownlp==0.12.3、multiprocessing、jieba==0.42.1、matplotlib==3.5.1、numpy==1.24.1、sklearn==1.2.0、torch==1.10.0）

5.2 实验流程

5.2.1 题目分析

1、数据处理

由于爬取到的数据非常繁杂，并且里面存在许多未知字符或者是 None，因此数据处理就显得十分重要，本实验要做的文本分类是通过短评来预测电影类型，因为电影类型众多，这里只挑选“喜剧”和“犯罪”两类，所以要从原始数据中分离出这两类数据，并且将其写入 txt 文件中，以便后续进行机器学习和深度学习实验时进行调用。

2、数据统计分析

本实验通过数据分析统计，分别输出电影类型&电影数量柱状图、电影国家或地区&电影数量柱状图、电影导演&电影数量柱状图、电影年份&电影数量折线图、电影排名与评论人数相关度分析散点图，最后使用 snownlp 库分析每一条短评的情感指数，并与爬取的该短评的评分进行比较。

3、基于机器学习的文本分类

本实验将采用传统机器学习逻辑回归和随机森林进行文本分类，重点是使用 tf-idf 算法将文本向量化，然后切分训练集和测试集，最后调用 sklearn 库里面的的逻辑回归和随机森林模型进行训练，输出分类报告矩阵。

4、基于深度学习的文本分类

本实验使用 TextRNN 进行文本分类，TextRNN 擅长捕获更长的序列信息。具体到文本分类任务中，BiLSTM 从某种意义上可以理解为可以捕获变长且双向的 N-Gram 信息。最后模型通过训练测试同样输出分类报告矩阵。

5.2.2 实验过程

1、生成机器学习数据

```
lst=[]
for i in range(df.shape[0]):
    if '喜剧'in df['剧情类型'][i].split():
        short1=eval(df['短评'][i])
        for j in range(len(short1)):
            lst.append(['喜剧',short1[j][0]])
    elif '犯罪'in df['剧情类型'][i].split():
        short2=eval(df['短评'][i])
        for j in range(len(short2)):
            lst.append(['犯罪',short2[j][0]])
random.shuffle(lst)

def savetxt(data,filename):
    t=''
    with open (filename,'w',encoding='utf-8') as q:
        for i in data:
            for e in range(len(data[0])):
                t=t+str(i[e])+'\t'
            q.write(t.strip(' '))
            q.write('\n')
            t=''
savetxt(lst[:int(len(lst)*0.1)],'./mldata/test.txt')
savetxt(lst[int(len(lst)*0.1):int(len(lst)*0.2)],'./mldata/val.txt')
savetxt(lst[int(len(lst)*0.2):],'./mldata/train.txt')
```

2、生成深度学习数据

```
lst=[]
for i in range(df.shape[0]):
    if '喜剧'in df['剧情类型'][i].split():
        short1=eval(df['短评'][i])
        for j in range(len(short1)):
            lst.append([short1[j][0],'0'])
    elif '犯罪'in df['剧情类型'][i].split():
        short2=eval(df['短评'][i])
        for j in range(len(short2)):
            lst.append([short2[j][0],'1'])
random.shuffle(lst)

def savetxt(data,filename):
    t=''
```

```
    with open (filename,'w',encoding='utf-8') as q:
        for i in data:
            for e in range(len(data[0])):
                t=t+str(i[e])+'\t'
            q.write(t.strip(' '))
            q.write('\n')
            t=''
savetxt(lst[:int(len(lst)*0.1)],'./mydata/data/test.txt')
savetxt(lst[int(len(lst)*0.1):int(len(lst)*0.2)],'./mydata/data/dev.txt')
savetxt(lst[int(len(lst)*0.2):],'./mydata/data/train.txt')
```

3、绘制电影类型与电影数量柱状图

```
count_dict = dict()
classs = []
[classs.extend(i.split()) for i in list(df['剧情类型'])]
for item in classs:
    if item in count_dict:
        count_dict[item] += 1
    else:
        count_dict[item] = 1
res = sorted(count_dict.items(),key = lambda x:x[1],reverse = True)
x=[]
y=[]
for item in res:
    x.append(item[0])
    y.append(item[1])

plt.figure(figsize=(20,6))
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.title("类型&电影数量")
plt.xticks(rotation=45)
plt.bar(x, y)
```

4、绘制国家或地区与电影数量柱状图

```
count_dict = dict()
classs = []
[classs.extend(i.split()) for i in list(df['地区'])]
for item in classs:
    if item in count_dict:
        count_dict[item] += 1
    else:
        count_dict[item] = 1
```

```
res = sorted(count_dict.items(),key = lambda x:x[1],reverse = True)
x=[]
y=[]
for item in res:
    x.append(item[0])
    y.append(item[1])

plt.figure(figsize=(20,6))
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.title("国家或地区&电影数量")
plt.xticks(rotation=45)
plt.bar(x,y)
```

5、绘制电影导演与电影数量柱状图

```
count_dict = dict()
classs = []
[classs.extend(i.split()) for i in list(df['导演'])]
for item in classs:
    if item in count_dict:
        count_dict[item] += 1
    else:
        count_dict[item] = 1
res = sorted(count_dict.items(),key = lambda x:x[1],reverse = True)
x=[]
y=[]
for item in res:
    x.append(item[0])
    y.append(item[1])

plt.figure(figsize=(20,6))
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.title("导演&电影数量")
plt.xticks(rotation=45)
plt.bar(x[:25],y[:25])
```

6、绘制年份与电影数量折线图

```
year = list(map(int,list(df['年份'])))
count_dict = dict()
for item in year:
    if item in count_dict:
        count_dict[item] += 1
    else:
```

```
            count_dict[item] = 1
res = sorted(count_dict.items(),key = lambda x:x[0],reverse = False)
x=[]
y=[]
for item in res:
    x.append(item[0])
    y.append(item[1])

plt.figure(figsize=(20,6))
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.title("年份&电影数量")
plt.xticks(rotation=45)
plt.plot(x,y, marker='o')
```

7、电影排名与评论人数相关性分析

```
plt.figure(figsize=(20,5))
plt.subplot(1,2,1)
rank = [i+1 for i in range(250)]
com = list(map(int,list(df['评论人数'])))
# 绘制散点图
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.scatter(rank,com)
plt.xlabel('电影排名')
plt.ylabel('评论人数')
# 绘制直方图
plt.subplot(1,2,2)
plt.hist(com)
```

8、短评语义情感分析

```
from snownlp import SnowNLP
import matplotlib.pyplot as plt
import numpy as np

sentimentslist = []
n=0
short = eval(df['短评'][n])
print('{}、《{}》'.format(n+1,df['电影中文名'][n]))
for i in short:
    s = SnowNLP(i[0])
    print('情感指数：{}\t 评分：{}'.format(s.sentiments,i[1]))
    sentimentslist.append(s.sentiments)
plt.rcParams['font.sans-serif'] = ['SimHei']
```

```
plt.rcParams['axes.unicode_minus']=False
plt.hist(sentimentslist, bins = np.arange(0, 1, 0.01), facecolor = 'g')
plt.xlabel('Sentiments Probability')
plt.ylabel('Quantity')
plt.title('Analysis of Sentiments')
plt.show()
#情感波动分析
result = []
i = 0
while i<len(sentimentslist):
    result.append(sentimentslist[i]-0.5)
    i = i + 1
#可视化画图
import matplotlib.pyplot as plt
import numpy as np
plt.plot(np.arange(0, 100, 1), result, 'k-')
plt.xlabel('Number')
plt.ylabel('Sentiment')
plt.title('Analysis of Sentiments')
plt.show()
```

9、文本向量化 tf-idf

```
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer

def constructDataset(path):
    """
    path: file path
    rtype: lable_list and corpus_list
    """
    label_list = []
    corpus_list = []
    with open(path, 'r',encoding='utf-8') as p:
        for line in p.readlines():
            label_list.append(line.split('\t')[0])
            corpus_list.append(line.split('\t')[1])
    return label_list, corpus_list

datapath = './mldata/'
write_list = [datapath+'train_token.txt',
datapath+'test_token.txt',   datapath+'val_token.txt']
```

```
train_label, train_set = constructDataset(write_list[0]) # 50000
test_label, test_set = constructDataset(write_list[1]) # 10000
val_label, val_set = constructDataset(write_list[2])
#  计算 tf-idf
corpus_set = train_set + val_set + test_set #  全量计算 tf-idf
print ("length of corpus is: " + str(len(corpus_set)))
vectorizer = CountVectorizer(min_df=1e-5) # drop df < 1e-5,去低频词
transformer = TfidfTransformer()
tfidf = transformer.fit_transform(vectorizer.fit_transform(corpus_set))
words = vectorizer.get_feature_names_out()
print ("how many words: {0}".format(len(words)))
print ("tf-idf shape: ({0},{1})".format(tfidf.shape[0], tfidf.shape[1]))
```

10、逻辑回归分类器

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

lr_model = LogisticRegression()
lr_model.fit(train_set, train_label)
print( "val mean accuracy: {0}".format(lr_model.score(val_set, val_label)))
y_pred = lr_model.predict(test_set)
print(classification_report(test_label, y_pred,target_names=['喜剧', '犯罪']))
```

11、随机森林分类器

```
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=200, random_state=1080)
rf_model.fit(train_set, train_label)
print( "val mean accuracy: {0}".format(rf_model.score(val_set, val_label)))
y_pred = rf_model.predict(test_set)
print(classification_report(test_label, y_pred,target_names=['喜剧', '犯罪']))
```

12、TextRNN 模型结构与参数

```
class Model(nn.Module):
    def __init__(self, config):
        super(Model, self).__init__()
        if config.embedding_pretrained is not None:
            self.embedding =
nn.Embedding.from_pretrained(config.embedding_pretrained, freeze=False)
        else:
            self.embedding = nn.Embedding(config.n_vocab, config.embed,
padding_idx=config.n_vocab - 1)
```

```
        self.lstm = nn.LSTM(config.embed, config.hidden_size, config.num_layers,
                            bidirectional=True, batch_first=True,
dropout=config.dropout)
        self.fc = nn.Linear(config.hidden_size * 2, config.num_classes)

    def forward(self, x):
        x, _ = x
        out = self.embedding(x)  # [batch_size, seq_len, embeding]=[128, 32, 300]
        out, _ = self.lstm(out)
        out = self.fc(out[:, -1, :])  # 句子最后时刻的 hidden state
        return out
        self.dropout = 0.5    # 随机失活
        self.require_improvement = 1000  # 若超过 1000batch 效果还没提升，则提
前结束训练
        self.num_classes = len(self.class_list)  # 类别数
        self.n_vocab = 0                # 词表大小，在运行时赋值
        self.num_epochs = 20            # epoch 数
        self.batch_size = 128           # mini-batch 大小
        self.pad_size = 32              # 每句话处理成的长度(短填长切)
        self.learning_rate = 1e-3       # 学习率
        self.embed = self.embedding_pretrained.size(1)\
            if self.embedding_pretrained is not None else 300          # 字向量维度,
若使用了预训练词向量，则维度统一
        self.hidden_size = 128                                  # lstm 隐藏层
        self.num_layers = 2                             # lstm 层数
```

13、TextRNN 模型训练

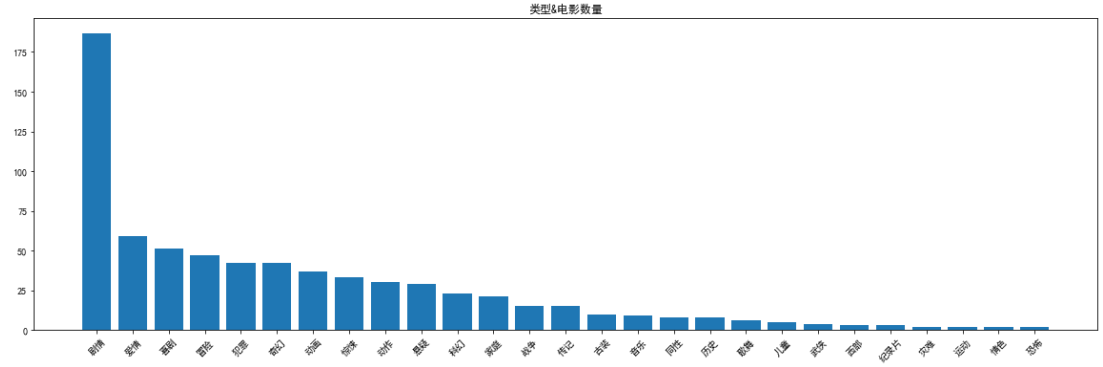
```
def train(config, model, train_iter, dev_iter, test_iter):
    start_time = time.time()
    model.train()
    optimizer = torch.optim.Adam(model.parameters(), lr=config.learning_rate)

    # 学习率指数衰减，每次 epoch：学习率 = gamma * 学习率
    # scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.9)
    total_batch = 0  # 记录进行到多少 batch
    dev_best_loss = float('inf')
    last_improve = 0  # 记录上次验证集 loss 下降的 batch 数
    flag = False  # 记录是否很久没有效果提升
    writer = SummaryWriter(log_dir=config.log_path + '/' +
time.strftime('%m-%d_%H.%M', time.localtime()))
    for epoch in range(config.num_epochs):
        print('Epoch [{}/{}]'.format(epoch + 1, config.num_epochs))
```

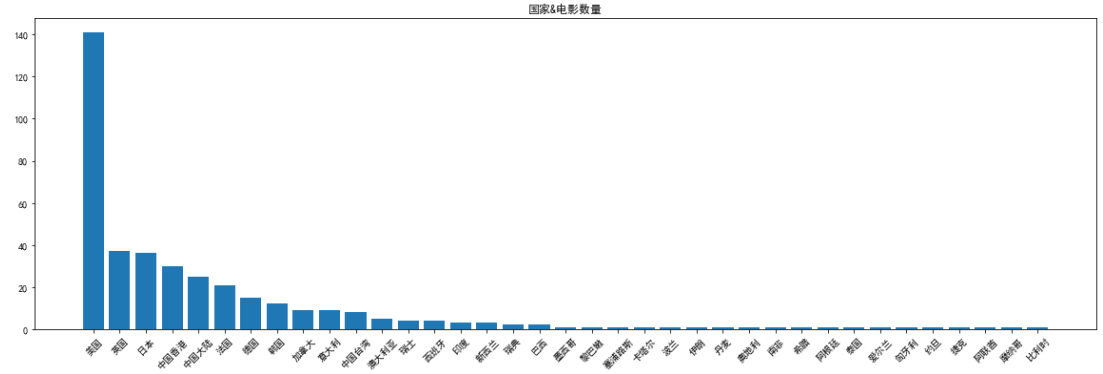
```
        # scheduler.step() # 学习率衰减
        for i, (trains, labels) in enumerate(train_iter):
            outputs = model(trains)
            model.zero_grad()
            loss = F.cross_entropy(outputs, labels)
            loss.backward()
            optimizer.step()
            if total_batch % 55 == 0:
                # 每多少轮输出在训练集和验证集上的效果
                true = labels.data.cpu()
                predic = torch.max(outputs.data, 1)[1].cpu()
                train_acc = metrics.accuracy_score(true, predic)
                dev_acc, dev_loss = evaluate(config, model, dev_iter)
                if dev_loss < dev_best_loss:
                    dev_best_loss = dev_loss
                    torch.save(model.state_dict(), config.save_path)
                    improve = '*'
                    last_improve = total_batch
                else:
                    improve = ''
                time_dif = get_time_dif(start_time)
                msg = 'Iter: {0:>6},  Train Loss: {1:>5.2},  Train Acc:
{2:>6.2%},  Val Loss: {3:>5.2},  Val Acc: {4:>6.2%},  Time: {5} {6}'
                print(msg.format(total_batch, loss.item(), train_acc, dev_loss, dev_acc,
time_dif, improve))
                writer.add_scalar("loss/train", loss.item(), total_batch)
                writer.add_scalar("loss/dev", dev_loss, total_batch)
                writer.add_scalar("acc/train", train_acc, total_batch)
                writer.add_scalar("acc/dev", dev_acc, total_batch)
                model.train()
            total_batch += 1
            if total_batch - last_improve > config.require_improvement:
                # 验证集 loss 超过 1000batch 没下降，结束训练
                print("No optimization for a long time, auto-stopping...")
                flag = True
                break
        if flag:
            break
    writer.close()
    test(config, model, test_iter)
```

5.3 实验结果及分析

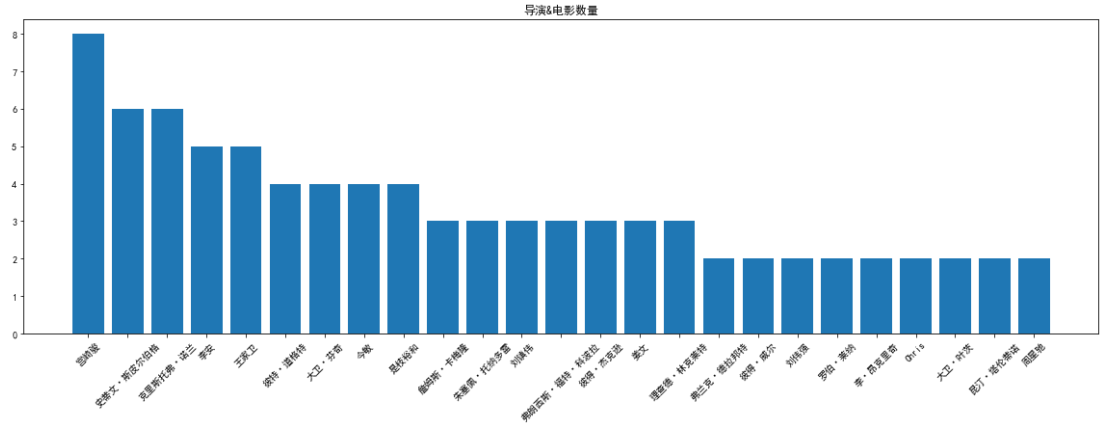
1、电影类型与电影数量柱状图



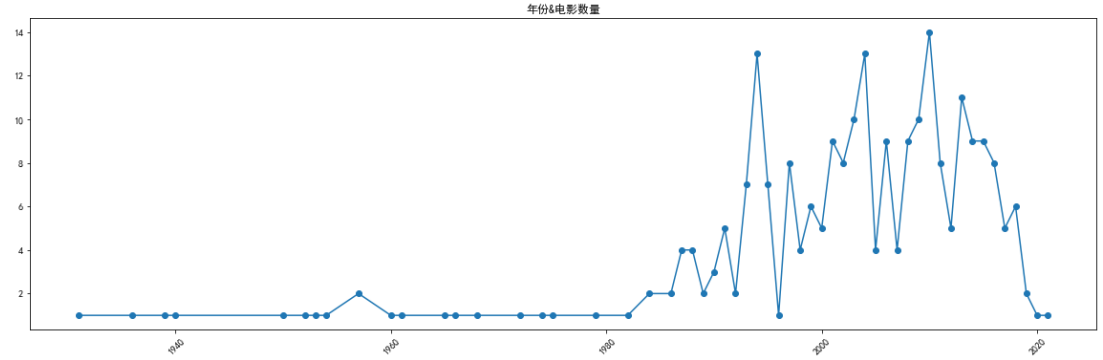
2、国家或地区与电影数量柱状图



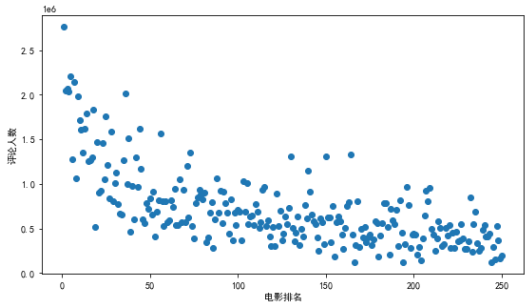
3、电影导演与电影数量柱状图

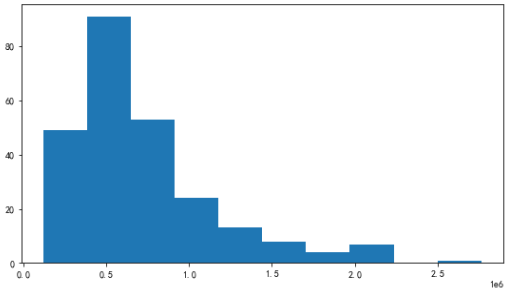


4、年份与电影数量折线图

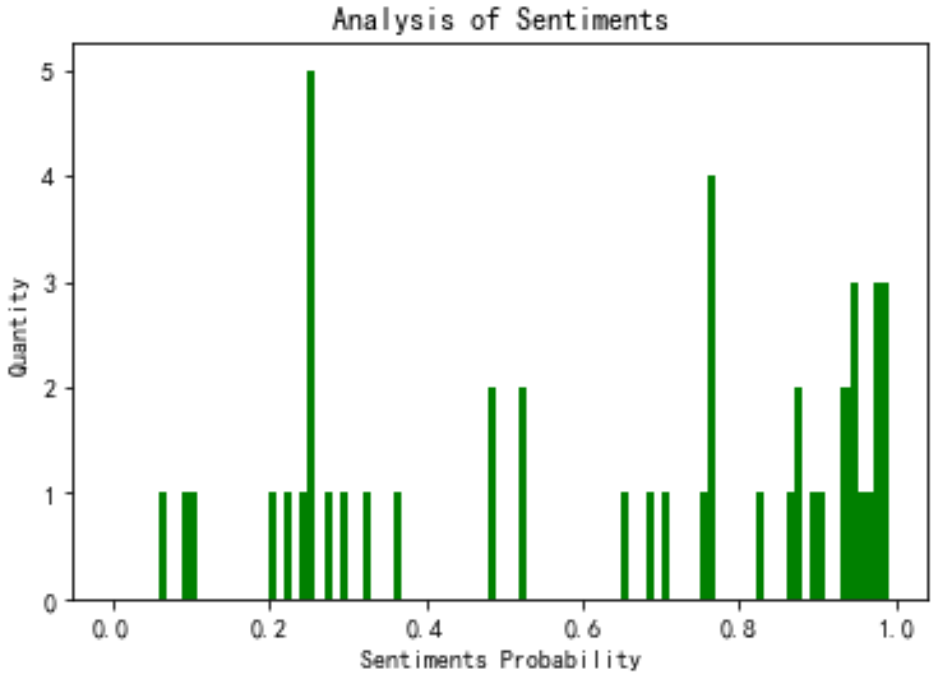


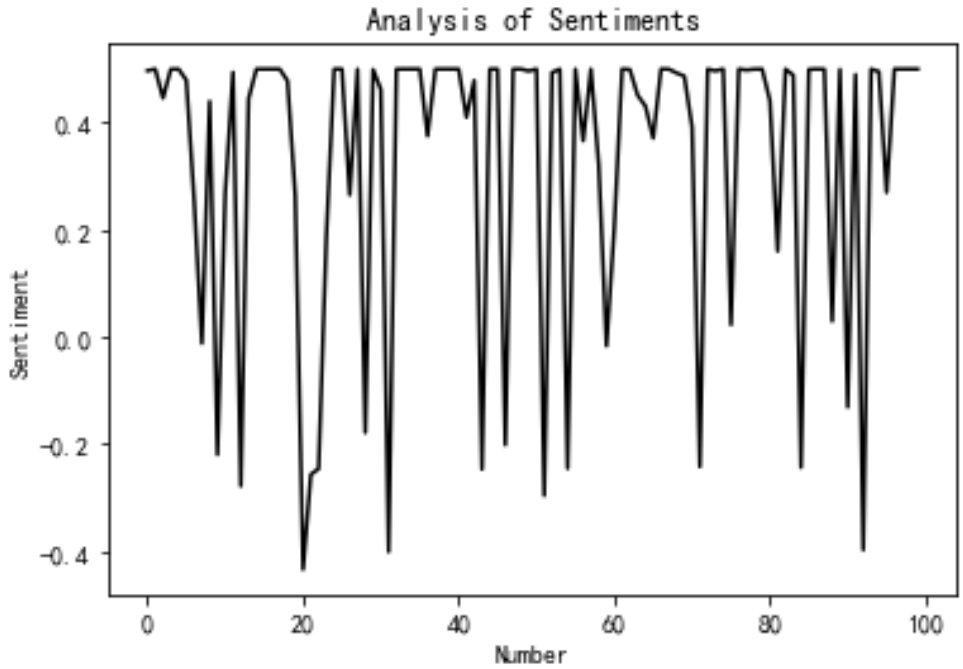
5、电影排名与评论人数相关性分析





6、短评语义情感分析





7、逻辑回归分类器分类结果

val mean accuracy: 0.7829099307159353

	precision	recall	f1-score	support
喜剧	0.81	0.93	0.87	541
犯罪	0.84	0.65	0.73	325
accuracy			0.82	866
macro avg	0.83	0.79	0.80	866
weighted avg	0.83	0.82	0.82	866

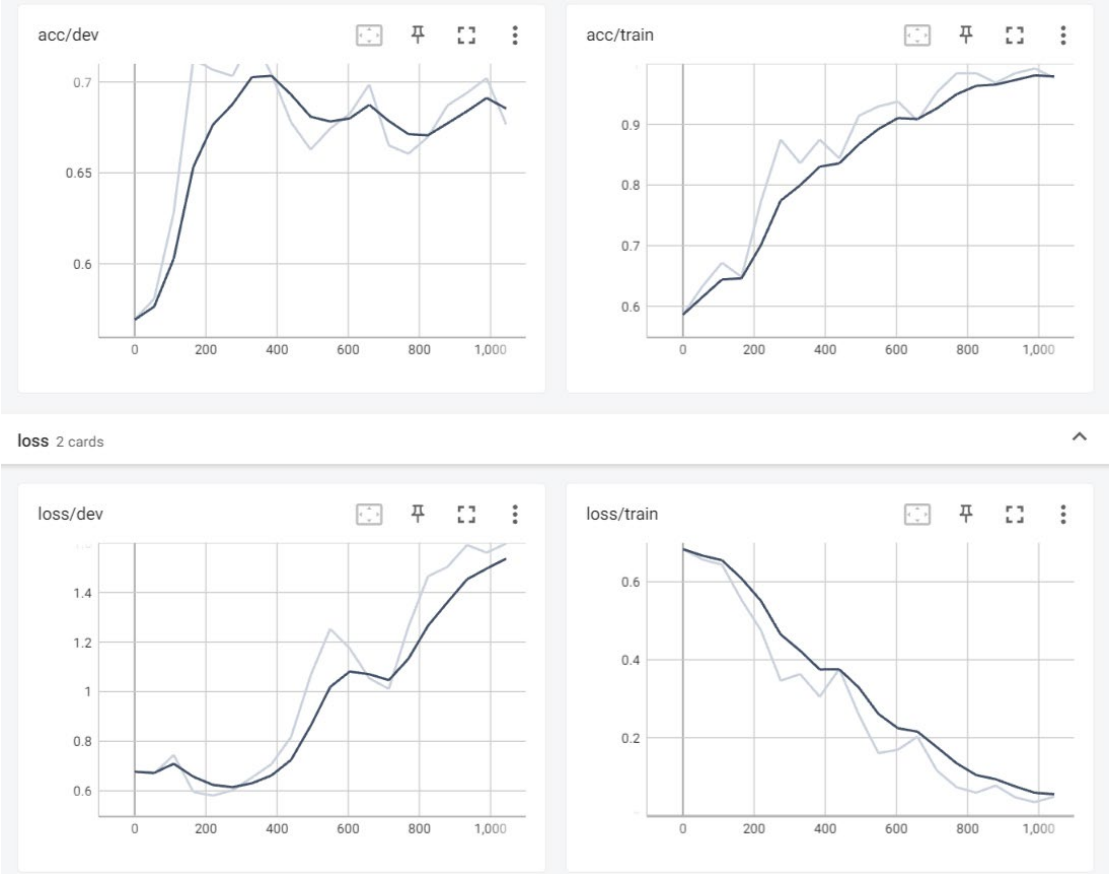
8、随机森林分类器分类结果

val mean accuracy: 0.7794457274826789

	precision	recall	f1-score	support
喜剧	0.78	0.92	0.85	541
犯罪	0.82	0.58	0.68	325
accuracy			0.79	866
macro avg	0.80	0.75	0.76	866
weighted avg	0.80	0.79	0.78	866

9、TextRNN 训练过程

```
Epoch [1/20]
Iter:      0,  Train Loss:  0.68,  Train Acc: 58.59%,  Val Loss:  0.68,  Val Acc: 56.93%,  Time: 0:00:02 *
Epoch [2/20]
Iter:     55,  Train Loss:  0.66,  Train Acc: 63.28%,  Val Loss:  0.67,  Val Acc: 58.08%,  Time: 0:00:31 *
Epoch [3/20]
Iter:    110,  Train Loss:  0.64,  Train Acc: 67.19%,  Val Loss:  0.75,  Val Acc: 62.82%,  Time: 0:00:59
Epoch [4/20]
Iter:    165,  Train Loss:  0.55,  Train Acc: 64.84%,  Val Loss:   0.6,  Val Acc: 71.25%,  Time: 0:01:27 *
Epoch [5/20]
Iter:    220,  Train Loss:  0.48,  Train Acc: 77.34%,  Val Loss:  0.58,  Val Acc: 70.67%,  Time: 0:01:57 *
Epoch [6/20]
Iter:    275,  Train Loss:  0.35,  Train Acc: 87.50%,  Val Loss:   0.6,  Val Acc: 70.32%,  Time: 0:02:30
Epoch [7/20]
Iter:    330,  Train Loss:  0.36,  Train Acc: 83.59%,  Val Loss:  0.65,  Val Acc: 72.40%,  Time: 0:03:04
Epoch [8/20]
Iter:    385,  Train Loss:  0.31,  Train Acc: 87.50%,  Val Loss:  0.71,  Val Acc: 70.44%,  Time: 0:03:31
Epoch [9/20]
Iter:    440,  Train Loss:  0.38,  Train Acc: 84.38%,  Val Loss:  0.82,  Val Acc: 67.78%,  Time: 0:04:00
Epoch [10/20]
Iter:    495,  Train Loss:  0.26,  Train Acc: 91.41%,  Val Loss:   1.1,  Val Acc: 66.28%,  Time: 0:04:30
Epoch [11/20]
Iter:    550,  Train Loss:  0.16,  Train Acc: 92.97%,  Val Loss:   1.3,  Val Acc: 67.44%,  Time: 0:05:04
Epoch [12/20]
Iter:    605,  Train Loss:  0.17,  Train Acc: 93.75%,  Val Loss:   1.2,  Val Acc: 68.24%,  Time: 0:05:39
Epoch [13/20]
Iter:    660,  Train Loss:   0.2,  Train Acc: 90.62%,  Val Loss:   1.1,  Val Acc: 69.86%,  Time: 0:06:14
Epoch [14/20]
Iter:    715,  Train Loss:  0.12,  Train Acc: 95.31%,  Val Loss:   1.0,  Val Acc: 66.51%,  Time: 0:06:48
Epoch [15/20]
Iter:    770,  Train Loss: 0.073,  Train Acc: 98.44%,  Val Loss:   1.3,  Val Acc: 66.05%,  Time: 0:07:23
```



10、TextRNN 测试结果

```
Test Loss:  0.62,  Test Acc: 67.32%
Precision, Recall and F1-Score...
              precision    recall  f1-score   support

      comedy     0.8393    0.5739    0.6817       528
       crime     0.5545    0.8284    0.6643       338

    accuracy                         0.6732       866
   macro avg     0.6969    0.7011    0.6730       866
weighted avg     0.7281    0.6732    0.6749       866
```

第 6 章　实验六

6.1 实验简介

6.1.1 实验题目

用 minist 数据集和 AlexNet 卷积神经网络做一个手写体识别程序。

6.1.2 实验环境

1) 华为 Matebook 13 电脑、Windows 11 操作系统
2) Visual Studio Code 1.74.2
3) Python 3.9.12 （torch==1.10.0、torchvision==0.11.1、numpy==1.24.1、matplotlib==3.5.1）

6.2 实验流程

6.2.1 MNIST 数据集介绍

MNIST 手写数字数据集来源于是美国国家标准与技术研究所，是著名的公开数据集之一，通常这个数据集都会被作为深度学习的入门案例。数据集中的数字图片是由 250 个不同职业的人纯手写绘制。

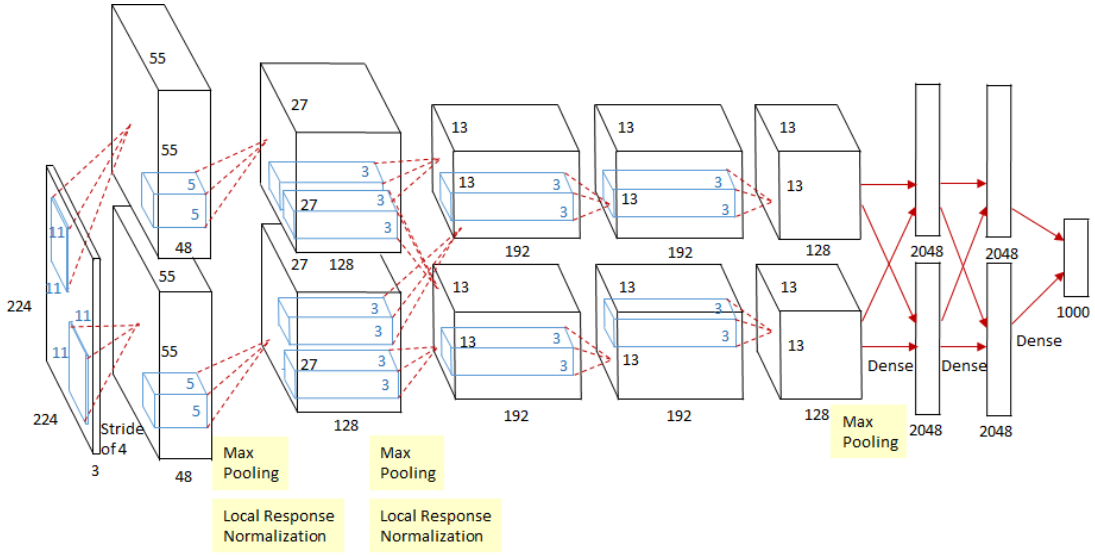
具体来看，MNIST 手写数字数据集包含有 60000 张图片作为训练集数据，10000 张图片作为测试集数据，且每一个训练元素都是 28*28 像素的手写数字图片，每一张图片代表的是从 0 到 9 中的每个数字。该数据集样例如下图所示：

如果我们把每一张图片中的像素转换为向量，则得到长度为 28*28=784 的向量。因此我们可以把 MNIST 数据训练集看作是一个[60000,784]的张量，第一个维度表示图片的索引，第二个维度表示每张图片中的像素点。而图片里的每个像素点的值介于 0-1 之间。此外，MNIST 数据集的类标是介于 0-9 的数字，共 10 个类别。通常我们要用独热编码（One_Hot Encoding）的形式表示这些类标。所谓的独热编码，直观的讲就是用 N 个维度来对 N 个类别进行编码，并且对于每

个类别，只有一个维度有效，记作数字 1 ；其它维度均记作数字 0。例如类标 1 表示为：([0,1,0,0,0,0,0,0,0,0])；同理标签 2 表示为：([0,0,1,0,0,0,0,0,0,0])。最后我们通过 softmax 函数输出的是每张图片属于 10 个类别的概率。

6.2.2 网络介绍

AlexNet 整体的网络结构包括：1 个输入层（input layer）、5 个卷积层（C1、C2、C3、C4、C5）、2 个全连接层（FC6、FC7）和 1 个输出层（output layer）。



模型创新点：

1) 使用 Relu 替换之前的 sigmoid 的作为激活函数。
2) 使用了数据增强策略（Data Augmentation），抑制过拟合。
3) 使用了重叠的最大池化（Max Pooling）。此前的 CNN 通常使用平均池化，而 AlexNet 全部使用最大池化，成功避免了平均池化带来的模糊化效果。
4) 提出 LRN（局部响应归一化）。
5) 成功使用 Dropout 机制，抑制过拟合。

6.2.2 实验过程

核心代码：

1、网络结构

```
class AlexNet(nn.Module):
    def __init__(self):
        super(AlexNet,self).__init__()

        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.relu1 = nn.ReLU()

        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        self.pool2 = nn.MaxPool2d(kernel_size=2,stride=2)
        self.relu2 = nn.ReLU()

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1)
```

```
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1)

        self.conv5 = nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1)
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.relu3 = nn.ReLU()

        self.fc6 = nn.Linear(256*3*3, 1024)
        self.fc7 = nn.Linear(1024, 512)
        self.fc8 = nn.Linear(512, 10)

    def forward(self,x):
        x = self.conv1(x)
        x = self.pool1(x)
        x = self.relu1(x)
        x = self.conv2(x)
        x = self.pool2(x)
        x = self.relu2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        x = self.pool3(x)
        x = self.relu3(x)
        x = x.view(-1, 256 * 3 * 3)#Alex: x = x.view(-1, 256*6*6)
        x = self.fc6(x)
        x = F.relu(x)
        x = self.fc7(x)
        x = F.relu(x)
        x = self.fc8(x)
        return x
```

2、数据加载

```
trainset =
torchvision.datasets.MNIST(root='./data',train=True,download=True,transform=transform)
trainloader = torch.utils.data.DataLoader(trainset,
batch_size=100,shuffle=True,num_workers=0)
testset =
torchvision.datasets.MNIST(root='./data',train=False,download=True,transform=transform1)
testloader =
torch.utils.data.DataLoader(testset,batch_size=100,shuffle=False,num_workers=0)
```

3、模型训练

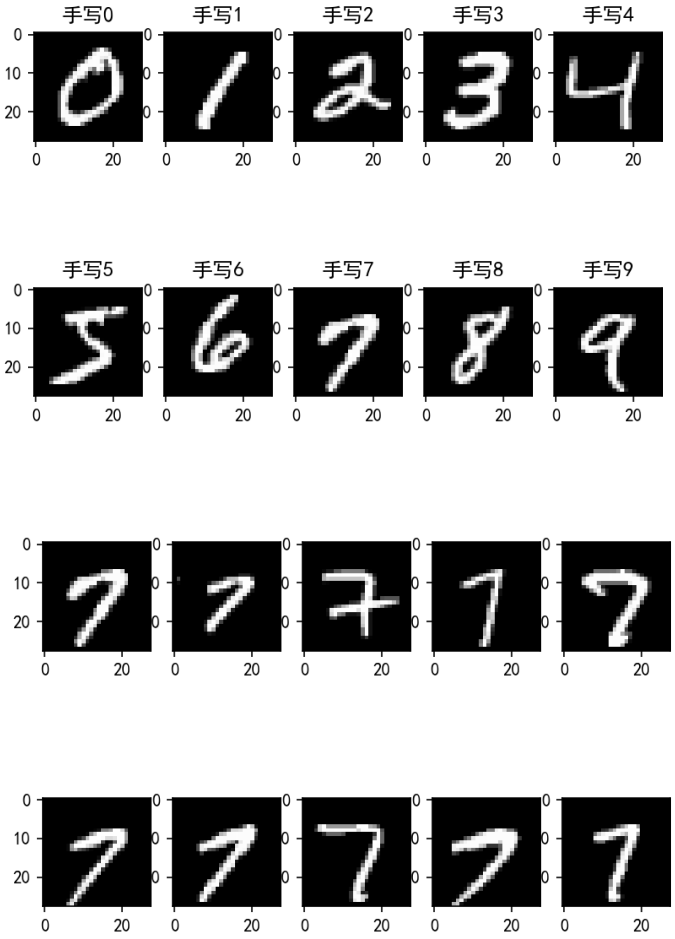
```
start=time.time()
train_loss = []
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(),lr=1e-3, momentum=0.9)
print("Start Training!")
num_epochs = 12 #训练次数
for epoch in range(num_epochs):
    running_loss = 0
    batch_size = 100
    for i, data in enumerate(trainloader):
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print('[%d, %5d] loss:%.4f' % (epoch + 1, (i + 1) * 100, loss.item()))
    train_loss.append(loss.item())
#保存训练模型
torch.save(net, 'AlexNet.pth')
end=time.time()
print("taining time:%s s"%(end-start))
plt.plot([i+1 for i in range(num_epochs)],train_loss)
plt.xlabel("epoch")
plt.ylabel("loss")
plt.show()
```

4、模型测试

```
with torch.no_grad():
    correct = 0
    total = 0
    for data in testloader:
        images, labels = data
        images, labels = images.to(device), labels.to(device)
        out = net(images)
        _, predicted = torch.max(out.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
    print('Accuracy of the network on the 10000 test images： {}%'.format(100 *
correct / total))
```

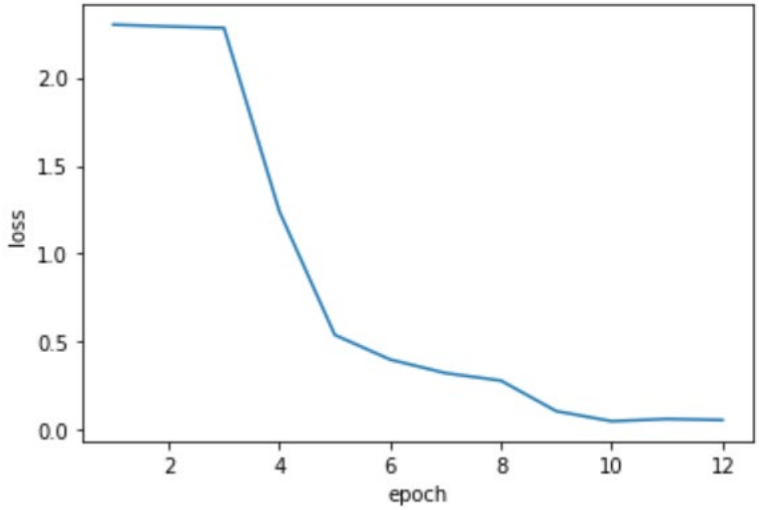
6.3 实验结果及分析

1、数据情况



2、训练损失变化

```
Start Training!
[1, 60000] loss:2.3004
[2, 60000] loss:2.2903
[3, 60000] loss:2.2821
[4, 60000] loss:1.2406
[5, 60000] loss:0.5398
[6, 60000] loss:0.3998
[7, 60000] loss:0.3230
[8, 60000] loss:0.2803
[9, 60000] loss:0.1070
[10, 60000] loss:0.0501
[11, 60000] loss:0.0627
[12, 60000] loss:0.0567
taining time:3155.0371057987213 s
```



3、测试结果

Accuracy of the network on the 10000 test images:96.21%

根据结果可知，训练好的AlexNet网络在10000张测试图片上的准确率为96.21%。

第 7 章 实验总结

俗话说“纸上得来终觉浅，绝知此事要躬行”，在做这些实验之前，我对 Python 的一些用法包括网络爬虫、数据转换、数据分析、文件保存、文本分类、深度学习等还是不甚了解的，大部分都是知其然而不知其所以然。例如，我之前只知道使用 Python 网络爬虫需要安装 request 库，然而并不知道具体怎么进行网页爬取和遇到问题该怎么解决，还有就是我只了解数据分析后续的操作，却不知道数据分析最重要的就是数据清洗，因为数据里面可能存在一些未知的问题，比如存在缺失值或者一些特殊的字符等等。在做实验的时候，我不理解的这些知识必然会在实验中出现，而我又必须要解决它们，这就是实验给我带来的最大好处，能够强迫我去弄懂我之前理解不透彻的知识，并将其应用。

除此之外，通过这次实验，我遇到问题解决问题的能力得到了很大的提升，比如当爬取豆瓣电影 Top250 的时候，由于我爬取的速度过快，内容过多导致我的 IP 被封禁，我最终通过更换 IP 和设置代理进行解决。还有就是在使用 TextRNN 网络进行文本分类的时候，遇到了 torch 版本与 torchtext 版本不匹配的问题，由于之前从未遇到过这种问题，导致这个问题卡了我很久，最终通过降低 torch 版本完美解决。同时，在实验中我也学到了很多非常便捷好用 python 模块，例如 map()方法改变列表中元素类型，eval()方法将 str 转为列表，scrapy 框架更有利于用户进行网页爬取等等。这次实验还提高了我的动手能力和编程能力，对我今后的学习有很大的帮助。

总而言之，通过本学期的课程和这次实验，我对 Python 这门语言有了更加深刻的了解，同时激发了我未来使用它的热情，之后我将继续循着自己的兴趣，在这条学习的道路上坚定地走下去，学着更好地使用这门强大的语言工具，争取在未来为 Python 的发展贡献自己的一份力。

参考文献

[1] 周志华. 机器学习. 北京：清华大学出版社,2016

[2] 田帅. 基于python抓取豆瓣电影TOP250的数据及进行分析[J]. 通讯世界, 2018(10):2.

[3] 冯莎. 豆瓣电影评论文本的情感分析研究——基于2017年电影《乘风破浪》爬虫数据[J]. 中国统计, 2017(7):4.

[4] Hu W， Gu Z， Xie Y , et al. Chinese Text Classification Based on Neural Networks and Word2vec[C]// 2019 IEEE Fourth International Conference on Data Science in Cyberspace (DSC). IEEE, 2019.