

上海大学 2018 ~ 2019 学年 冬季学期 课程论文

课程名：数据分析研究与实战		课程编号：15175121
学 号： 16124278		
姓 名： 王 浩		
论文题目：基于 TF 的 CNN 与 LSTM/GRU 的《流浪地球》影评数据分析与比较		
评分项目	分数分配	得分
问题创新性	10	
论文格式规范性	10	
论文结构及内容条理性	10	
研究方法是否得当、描述是否清楚	10	
数据量及数据内容是否符合研究问题要求	10	
数据预处理是否得当	10	
特征工程是否得当	5	
实验设计是否科学有效、实验结果分析是否得当	15	
代码是否规范完整	15	
代码是否能正确运行	5	
Total	100	

基于 TF 的 CNN 与 LSTM/GRU 的《流浪地球》影评数据分析

王浩

(上海大学悉尼工商学院 · 信息管理与信息系统专业, 上海 嘉定 201800)

摘 要: 为了通过影评数据反应电影《流浪地球》的口碑以及其造成热度, 并核实网络水军恶意对其刷差评现象是否存在, 本文利用猫眼 PC 端接口, 通过控制时间参数, 动态爬取了,《流浪地球》在 2018 年 3 月 5 日之前的 54 万余条影评数据。对这些数据进行预处理操作, 特征处理, 可视化分析后, 引入了卷积神经网络与循环神经网络对影评内容进行了更进一步的情感分析, 对今后的的数据分析预测学习有一定的指导意义。

关键词: 机器学习; 爬虫; 数据可视化; 情感分析; 卷积神经网络; 循环神经网络;

Analysis of Movie Review Data Based on TF CNN and LSTM/GRU

WangHao

(SHU-UTS SILC Business School, Shanghai University, Jiading 201800, China)

Abstract: In this paper, in order to reflect the popularity of the movie "Wandering Earth" and its heat, and to verify whether the phenomenon of malicious brushing of the film by the network Navy exists, this paper uses the cat's eye PC interface to dynamically crawl over 540,000 reviews of "Wandering Earth" before March 5, 2018 by controlling the time parameters. After pretreatment, feature processing and visual analysis of these data, convolution neural network and cyclic neural network are introduced to conduct a deeper emotional analysis of the content of film review, which has certain guiding significance for future data analysis and prediction learning.

Keywords: Machine learning, emotional analysis; Convolution neural network; Crawler ;Data visualization

0 引 言

《流浪地球》作为今年春节档期的唯一一部的投入了巨大资金的科幻题材电影, 在一众贺岁电影中脱颖而出, 凭借其充满诚意的内容打动了观众, 在影评口碑发酵后, 流浪地球的排片率稳步上升, 票房也屡创新高, 根据猫眼实时电影票房数据显示, 截至 2019 年 3 月 5 日, 其票房成绩已达 45.57 亿, 暂列中国电影票房总榜第二名, 成为今年春节档电影的最大赢家, 而在其荣耀的背后却是电影市场的暗流涌动。

中国电影票房总榜 (截至2019年03月05日)

排名		票房(万元)	平均票价	场均人次
1	战狼2 2017-07-27	568305	35	38
2	流浪地球 2019-02-05	455661	44	32
3	红海行动 2018-02-16	365086	39	33
4	唐人街探案2 2018-02-16	339774	38	38
5	美人鱼 2016-02-08	338627	36	44

图 1-中国电影票房榜总榜

1 研究问题

《流浪地球》影评数据分析的获取与分析

1.1 问题背景

流浪地球首日排片率低，票房成绩不佳。随着上映后第一批观众的良好口碑发酵，几大电影售票平台与评分平台纷纷对该电影给予巨大的好评，对其后续票房走势的爆炸产生了重要的推动影响。



图 2-流浪地球排片与票房走势

其结果就是，随着排片率上升，观影人数增加，票房激增，在电影获得巨大成功的同时，网传出现水军恶意刷流浪地球差评，豆瓣电影评分一降再降，网友群起而攻之，集中对豆瓣电影不公平的网络评论及评分回应“网络暴力”，豆瓣 app 评分大量差评，一度降至最低分。

1.2 问题描述

为了动态了解流浪地球影评数据对其票房影响，以及后续差评水军的检测，需要收集不同时间段的大量影评数据作为数据支持并动态追踪，通过影评数据反映出上述事件的爆发点，而另一方面由于许多影评网站已经做了大量的反爬虫机制。

豆瓣网从 2017 年 10 月开始全面禁止爬取数据。在非登录状态下仅仅可以爬取 200 条短评，登录状态下仅可以爬取 500 条数据。白天一分钟最多可爬 40 次，晚上 60 次，超过次数就会封 IP 地址不利于大量数据的获取。

同时可利用影评网站自带的打分系统，形成带感情色彩的影评数据，结合相关算法训练出能够识别出影评情感的模型，可用于推测没有评分数据的影评的情感倾向。

2 解决方法

2.1 问题分析

为了大量稳定的获取影评数据，放弃豆瓣影评数据，改用更易爬取且数据量更大的猫眼影评接口，爬取包含：网名，所在城市，评分，评论内容，已经评分时间的评论人信息。

对数据预处理后，通过可视化图标观察影评数据与城市的关系，观众对应电影的情感以及情感走势与时间的关系，日投票量中的投票占比，同时对影评关键词进行监控，以发现数据和上述事件之间的关系。

利用 Pyecharts 并结合 Echarts 对影评数据进行可视化分析，利用 Wordcloud 提取影评中的关键词，同时利用 Tensorflow 中的 Tensorboard 对模型构建过程进行可视化。

同时，对抽取的影评数据进行进一步处理后，影评内容对应的评分（0.5-5.0 十个档位的评分可作为情感分析的重要参考），结合卷积神经网络和递归神经网络对影评数据进行进一步的情感分析并建立模型。

2.2 算法分析

鉴于神经网络模型通过学习和训练后，能有效地模拟人脑的学习方式，对大量的输入文本信息进行高效的分析，并对文本中的情感进行判断，非常适合用于文本情感分析的研究中，可将爬取得到的影评数据进一步拆分为 训练集，测试集，验证集，并规范格式后分别利用 CNN 与 LSTM 或 GRU 对影评内容进行情感分析，并建立预测模型。

CNN 处理文本的时候，输入就是一个为矩阵的句子，就像原先图像像素的输入一样，不过不是单通道的。矩阵的每一行对应一个单词的 Token，通常是一个单词，但它可以是是一个字符。也就是说，每行是表示单词的向量表示。通常，这些向量是词嵌入向量（低维表示），如 word2vec 或 GloVe，但它们也可以是将单词索引为词汇表的 one-hot 向量。

下面就是使用 CNN 抽取文本信息的原理，“wait for the video and do not rent it” 是我们需要处理的数据。我们将其处理为图像格式的数据，如图中所示，每一行就表示一个词的

词向量，完成“图像”的构造后，就需要对其进行卷积。这里使用的卷积核比较特别，其大小是 (step, Embedding_Size) 在完成卷积后可得到对应的特征图，这里需要注意的是，并不是一次卷积，比如对这个“图像”进行卷积的时候（假设文本长度都为 50），我们采用 (2, 100), (3, 100), (4, 100) 的卷积和进行卷积后分别得到 (49, 1, 126), (48, 1, 126), (47, 1, 126) 的特征图。然后采用 max_pool 采样，依次采样为 (1, 1, 126), (1, 1, 126), (1, 1, 126)，然后进行拼接最后添加分类层。

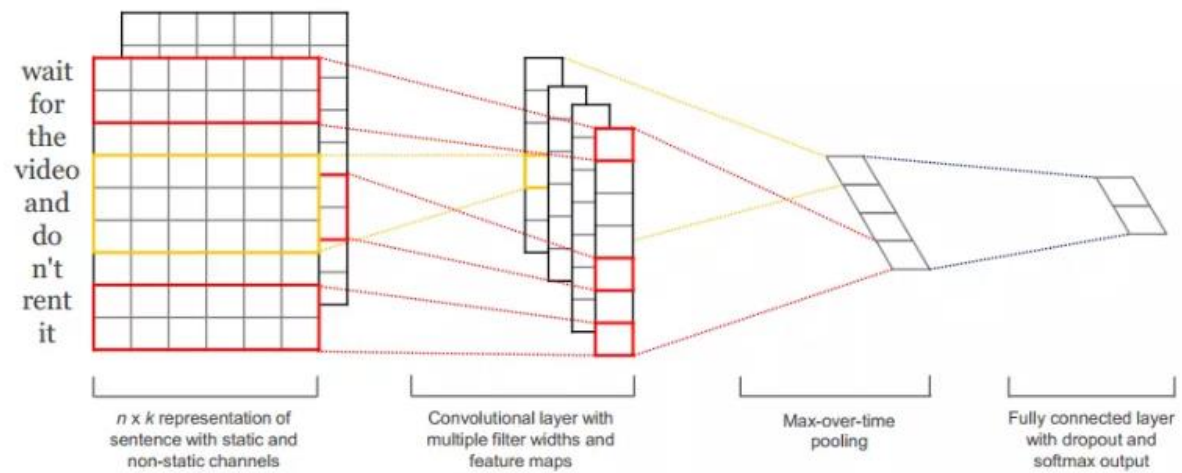


图 3-文本分析中的 CNN 原理图

而 RNN 会受到短时记忆的影响。如果一条序列足够长，那它们将很难将信息从较早的时间步传送到后面的时间步。因此，如果你正在尝试处理一段文本进行预测，RNN 可能从一开始就会遗漏重要信息。

在反向传播期间，RNN 会面临梯度消失的问题。梯度是用于更新神经网络的权重值，消失的梯度问题是当梯度随着时间的推移传播时梯度下降，如果梯度值变得非常小，就不会继续学习。

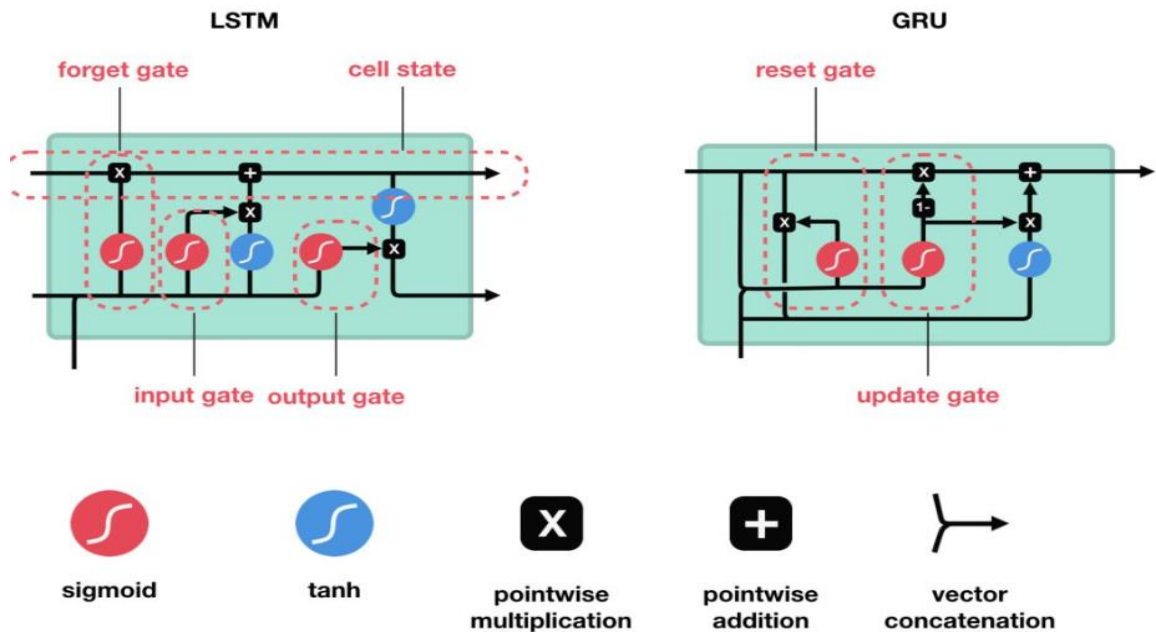


图 4-LSTM 与 GRU

LSTM 是一种特殊的 RNN 类型，一般的 RNN 结构如下图所示，是一种将以往学习的结果应用到当前学习的模型，但是这种一般的 RNN 存在着许多的弊端。举个例子，如果我们要预测“the clouds are in the sky”的最后一个单词，因为只在这一个句子的语境中进行预测，那么将很容易地预测出是这个单词是 sky。在这样的场景中，相关的信息和预测的词位置之间的间隔是非常小的，RNN 可以学会使用先前的信息，这相对于 CNN 无疑更利于文本分析。

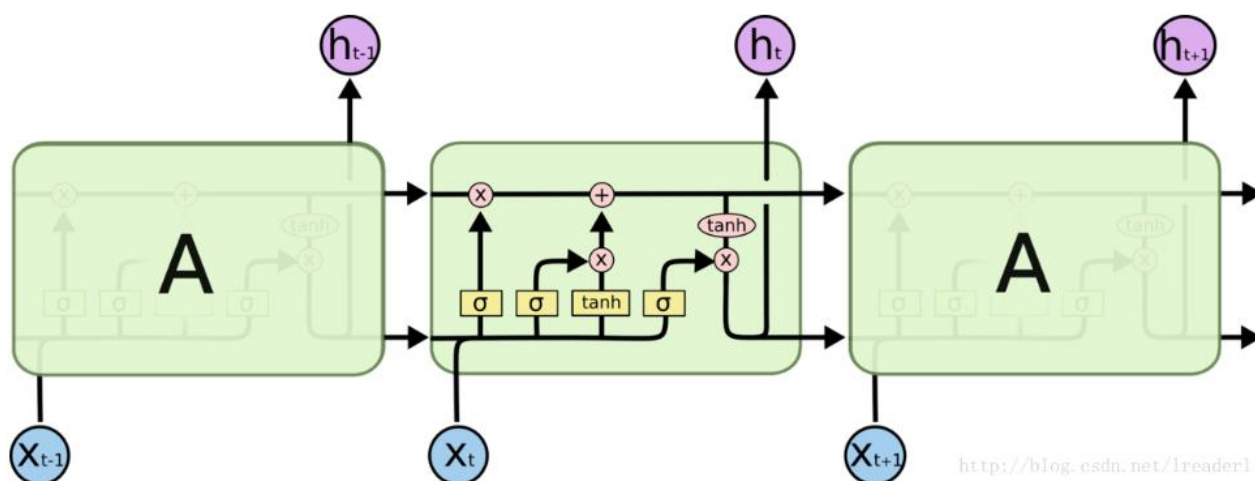


图 5-LSTM 原理图

GRU 作为 LSTM 的一种变体，将忘记门和输入门合成了一个单一的更新门。同样还混合了细胞状态和隐藏状态，加诸其他一些改动。最终的模型比标准的 LSTM 模型要简单，也是非常流行的变体。

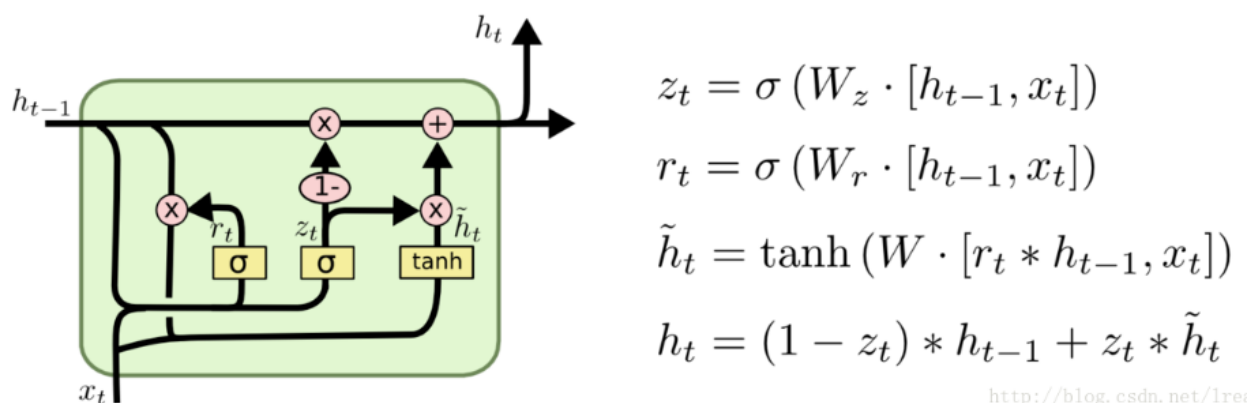


图 6-GRU 原理图

3. 实验研究

3.1 环境与数据集介绍

3.1.1 软硬件环境

操作系统: Windows 10 Insider Preview 17758.1 (rs5_release) x64 ; 开发平台: PyCharm 2018.1.4 (Professional Edition); 第三方库: pyecharts, jieba, wordcloud, tensorflow, json, requests; 额外补充: Echarts 中国地图包。

处理器: Intel (R) Core (TM) [i7-8500uCPU@2.90GHz](#); 内存: 16GB。

3.1.2 数据集爬取

猫眼对接口没有做反爬限制, 可以通过控制该接口的 start Time 的值以 json 数据的形式来获取不同时段的所有影评信息:

<http://m.maoyan.com/mmdb/comments/movie/248906.json?v=yes&offset=0&startTime>

```
{
  "cmts": [
    {
      "approve": 0,
      "approved": false,
      "assistAwardInfo": {
        "avatar": "",
        "celebrityId": 0,
        "celebrityName": "",
        "rank": 0,
        "title": ""
      },
      "avatarurl": "https://img.meituan.net/avatar/d2528b1c63a3228c91e4083b9be81d729195.jpg",
      "cityName": "柳州",
      "content": "国内第一部科幻电影还是可以的, 中国科幻电影会越来越好",
      "filmView": false,
      "id": 1060142694,
      "isMajor": false,
      "juryLevel": 0,
      "majorType": 0,
      "movieId": 248906,
      "nick": "Get 碳酸。",
      "nickName": "Get 碳酸。",
      "oppose": 0,
      "pro": false,
      "reply": 0,
      "score": 5,
      "spoiler": 0,
      "startTime": "2019-03-07 14:35:22",
      "supportComment": true,
      "supportLike": true,
      "sureViewed": 1,
      "userId": 1
    }
  ]
}
```

图 7-猫眼平台影评部分 json 数据

即利用 Get_Data.py 模仿手机客户端 (iPhone ios11) 获取猫眼 PC 网站 json 数据, 设置好 statTime 开始与结束时间后, 然后改变 startTime 字段的值来获取更多评论信息, 把

offset 置为 0，把每页评论数据中最后一次评论时间作为新的 startTime 去重新请求，即可开启爬虫获取数据，获取所有 54 万条数据大概用时 10 小时左右。

3.1.4 数据集

通过爬虫程序获取的数据存入 csv 文件中，其中包含了评论人在内的网名，地域，评论内容，评分，及时间的 524460 条影评数据。

524383	蝶花恋·花醉	乌鲁木齐	不是说吴京是主演吗？	4.5	2018/5/5 2:25
524384	跪下叫吕爷	北京	吴京主演期待，扬我国威	4	2018/5/4 16:06
524385	WOH430556421	自贡	看了原著，希望快点上映啊	5	2018/4/28 18:46
524386	大滑稽帝国	兰州	希望中国的第一部科幻片能够成功！不要像三体那样，也希望把钱都投在科幻	4	2018/4/27 22:02
524387	07的存在	南京	看好这部电影，说是科幻电影其实更多的还是展示人性，把握好人性的在科幻	4	2018/4/24 19:57
524388	FKD59331540	连云港	期待这部作品，毕竟喜欢原著，不过更期待三体	5	2018/4/22 16:57
524389	thefy	杭州	看了原著来的，希望不要失望	5	2018/4/20 23:05
524390	景筱年、	北京	大概上初中的时候看过这个小说，当时发表在科幻世界上面。没想到都拍成电	5	2018/4/19 10:35
524391	龚文字	北京	能把翻拍电影的版权卖给好莱坞吗？反正天朝的公司在那边也有投资，不算什	2.5	2018/4/14 22:06
524392	淦TMD贾贾靴靴	北京	大刘的书我都看的差不多了，只是没想到三体还没出电影，流浪地球就来了，	5	2018/4/14 12:06
524393	Godlike	北京	先不给满星，毕竟还没看到，三体是让我失望了，大刘的科幻小说那么精彩，	4	2018/4/6 22:44
524394	德艺双馨董叔叔	上海	加油！中国第一部硬科幻电影！喜欢大刘，这个原著读了好几遍，可能我们没	5	2018/4/6 0:08
524395	死在玫瑰下的夜莺	南昌	原著粉忽然刷到这个，有点期待	3.5	2018/3/26 9:21
524396	huawei2012	芜湖	科幻和欧美的还是有很大距离的，	0.5	2018/3/19 12:41
524397	Ttq766328364	大庆	等等，这片子怕不是喜天合作的吧？两大主演都是喜天的啊！！	2.5	2018/3/19 11:55
524398	sadzone	天津	地球发动机将不间断地开动500年，到时地球将加速至光速的千分之五，然后	5	2018/3/11 0:22
524399	大道如青天	北京	加油吧，流浪地球，不要再跟三体一样拖拖拉拉的	4.5	2018/3/2 10:23
524400	大西几	花都	就先拍流浪地球就对了，就演员阵容来说，应该是留了不少钱做特效的，只	5	2018/3/1 15:46
524401	岁月神偷	张家港	十多年前在科幻世界杂志连载看到了这部短篇。惊为天人。不过对于国内科幻	5	2018/2/19 18:23
524402	Hexa·今朝醉	邯郸	喜欢刘慈欣，希望这部电影不要让我失望。	5	2018/2/17 18:31
524403	galaxy199026	包头	小说就不咋地，电影估计也不好！	1	2018/2/16 21:03
524404	李佩佩嫁我	锦州	等下？？？？？？	0	2018/2/12 15:50
524405	QJe208700977	泸州	个人认为大刘的作品，科幻只是一部分，起到一种吸引力的作用，最重要的科	5	2018/2/10 15:13
524406	shehuige	武汉	这电影比三体靠谱多了，首先投资方就很稳:中影、北京文化、万达，中影计	5	2018/1/18 1:55
524407	cSu350409419	许昌	大刘的书，但是三体去哪了呢，跳票几次了？	5	2018/1/10 21:56
524408	忍莛	张家港	不管拍的如何，我都会带孩子去剧院支持。辛苦了	5	2017/12/30 17:07
524409	EdP405428553	沈阳	这是我初中时候看的科幻世界杂志里登的文章，算算能有16年了，很期待啊	5	2017/12/28 21:22
524410	dPP585794125	珠海	大刘原著粉，期待！	5	2017/12/27 12:41
524411	被水煮的那条鱼	孝感	看看标题，酷啊！这将会是怎样的一个体验呢，期待你的上映！	5	2017/12/24 19:25
524412	诗云	西安	看过流浪地球原著，貌似是短篇小说，不过内容有很大的想象空间，期待本	5	2017/5/9 21:21

图 8-爬虫获取的影评数据

3.2 数据预处理介绍

在 Get_Data.py 中，通过 json 关键字获取内容，对文本进行初步清洗后存入 csv 文件，去重，排空。

紧接着，在 Analyze_Data.py 中，利用 remove_None() 函数对评论人的地域名称和 Echarts 中的城市名称进行模糊匹配，一一对应，同时将评论时间的格式转换为标准日期格式，便于日期排序。

通过 lambda 表达式借助 judgeTime() 函数，将 19 年 2 月 4 号之前的影评数据全部汇总到 19 年 2 月 4 号，并删除 19 年 3 月 5 号之后的影评数据（最新数据是 19 年 3 月 6 号的数据）。

3.3 特征处理介绍

在 Analyze_Data.py 文件中对预处理的文件进行可视化分析，利用 Pyecharts 并结合 Echarts 对影评数据进行可视化分析，绘制观众地域图，观众地域排行榜单（前 20），观众评论数量与日期的关系，观众情感曲线，观众评论数量与时间的关系图，观众评论走势与时间的关系：

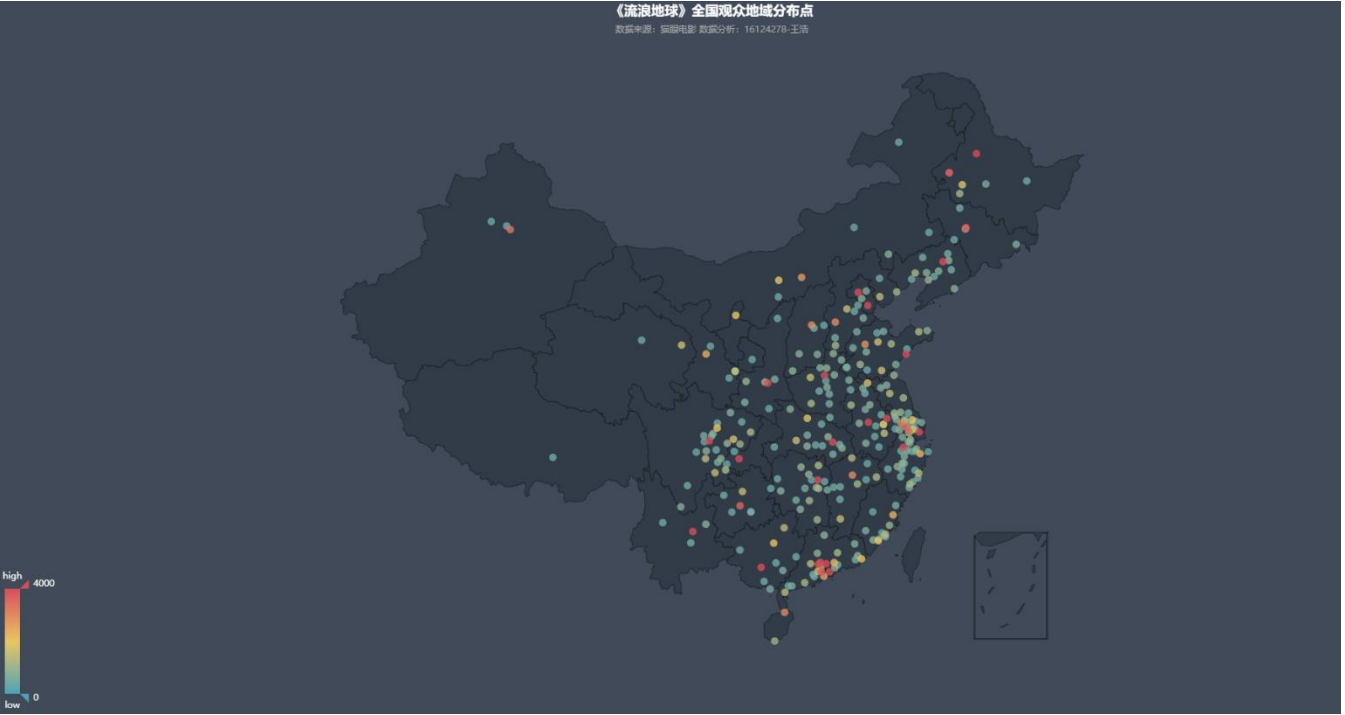


图 9-《流浪地球》全国观众地域分布点

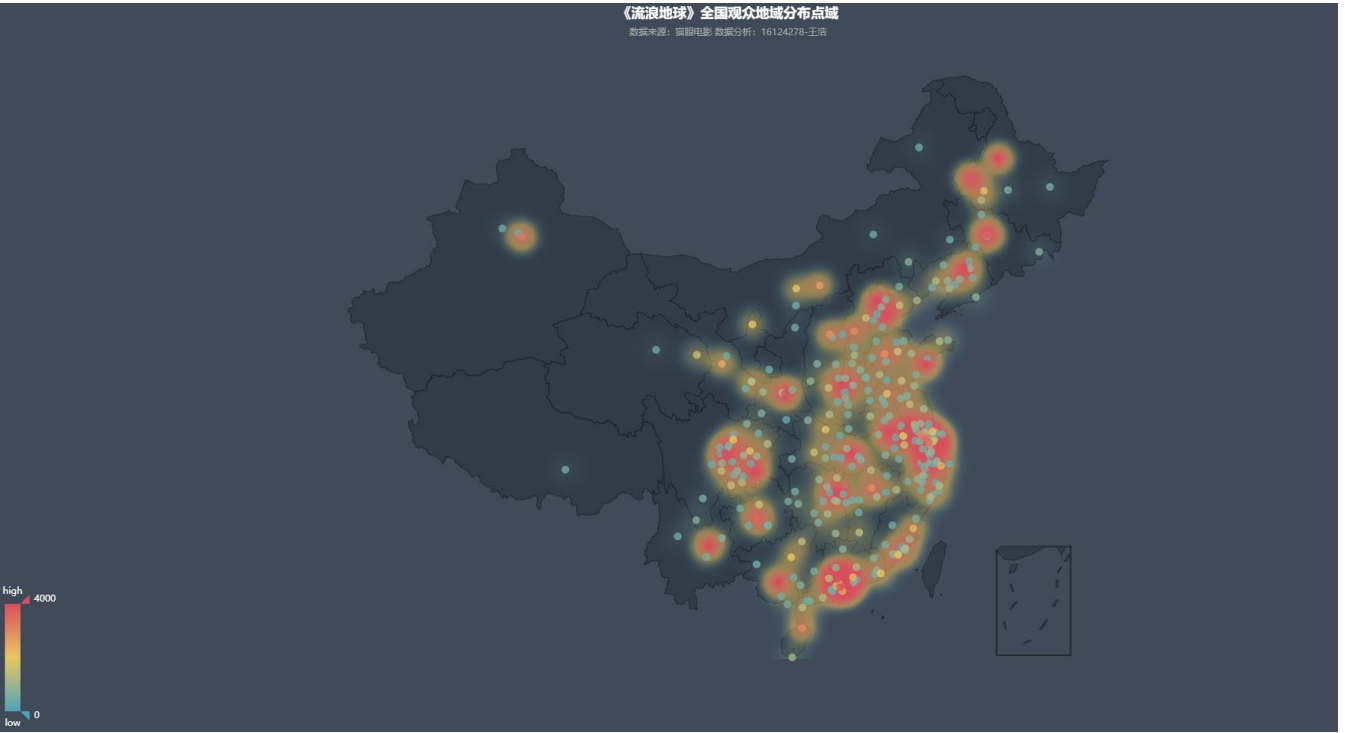


图 10-《流浪地球》全国观众地域分布点域

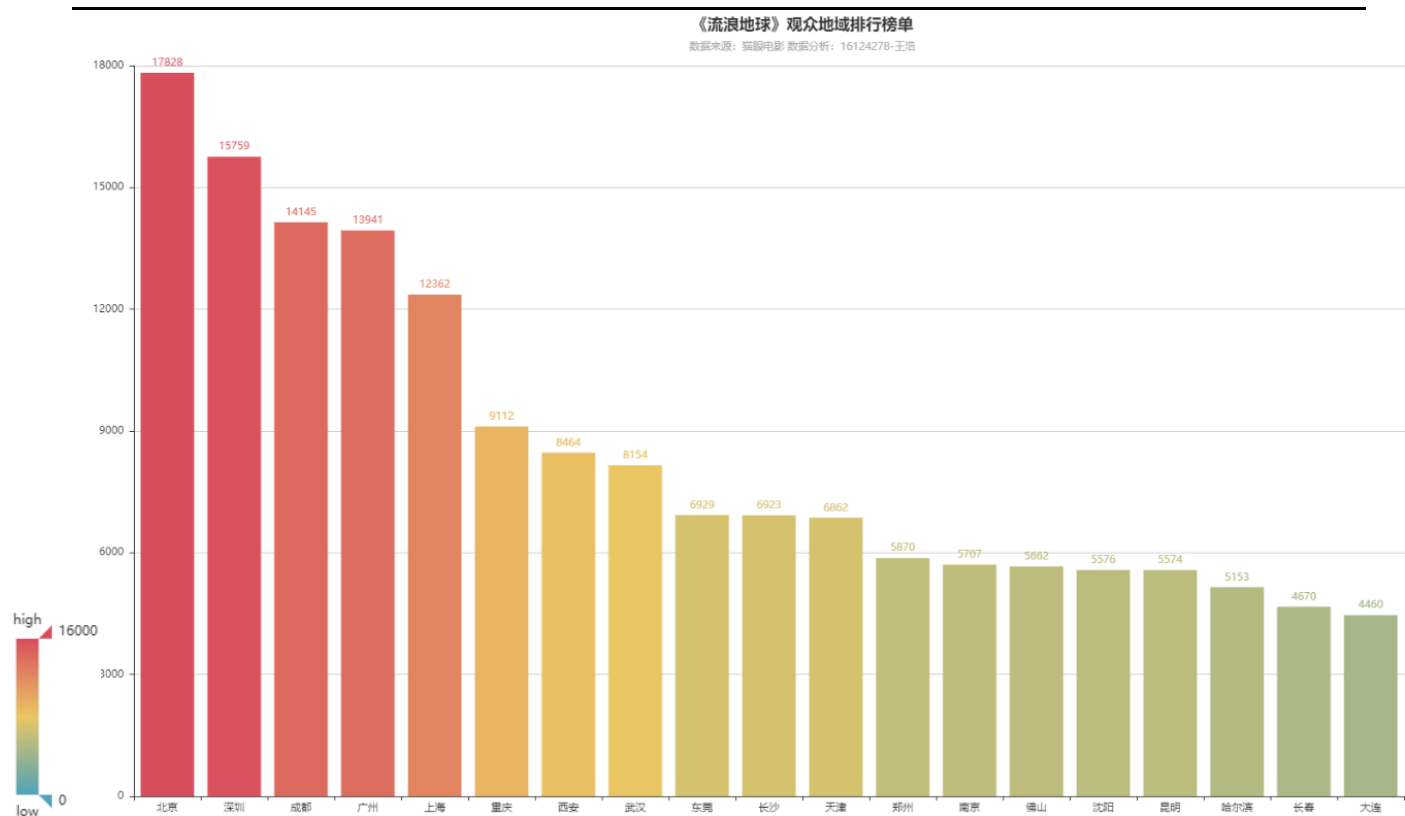


图 11-《流浪地球》观众地域排行榜单

根据全国观众地域分布图以及观众地域排行榜单（图 9，10，11）发现提交影评的观众地域分布正常，没有异常情况，地域影评数据符合正常情况下的人口密度分布（春节期间，上海市人口流失严重，，，，）。

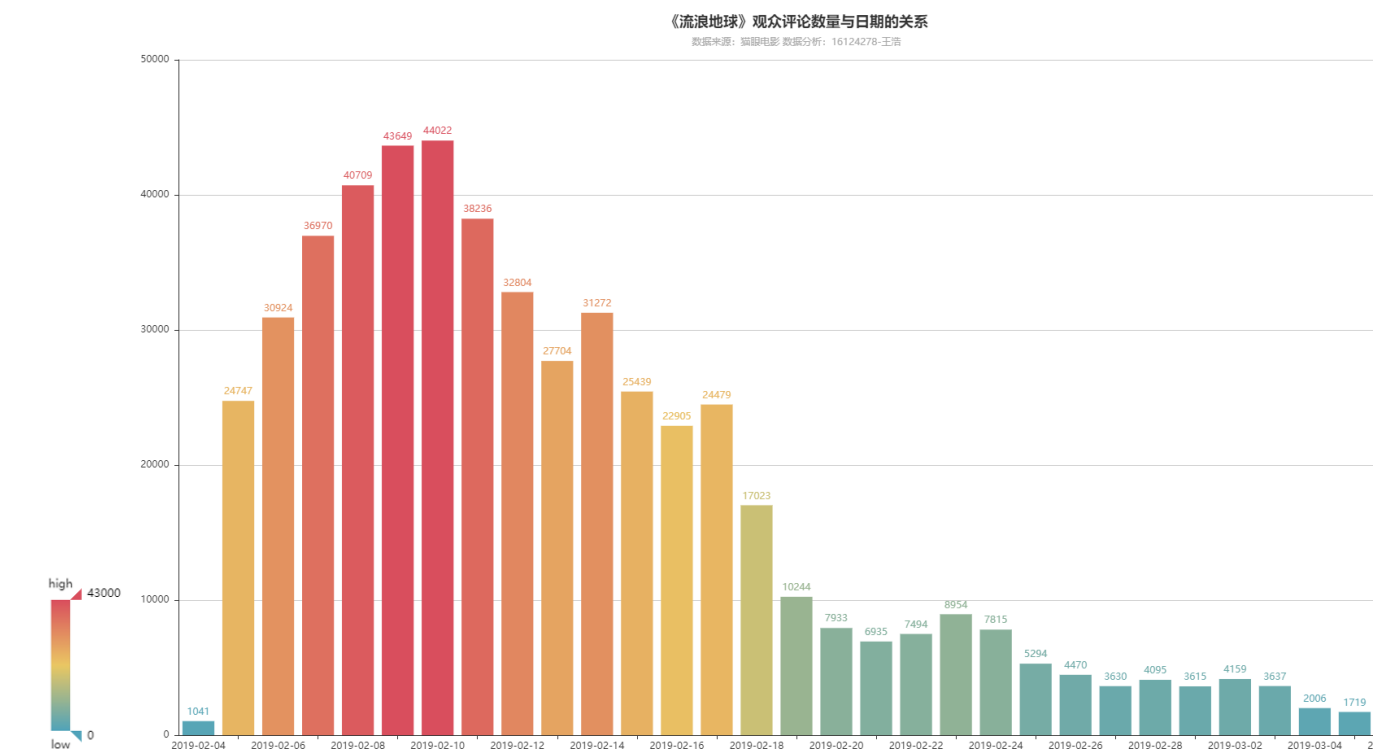


图 12-《流浪地球》观众评论数量与日期的关系

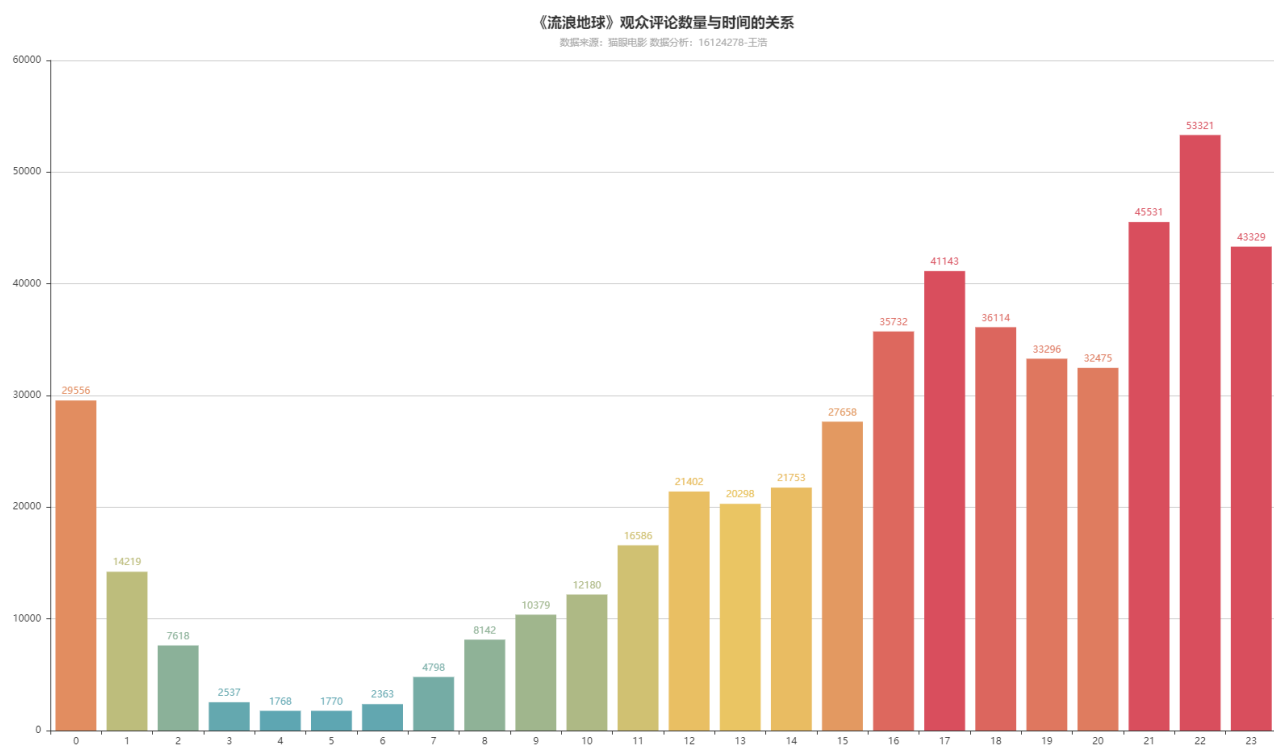


图 13-《流浪地球》观众评论数量与时间的关系

根据《流浪地球》观众评论数量与日期和时间的关系图（图 12，13）观众评论时间正常分布，大部分影评都是在深夜发布，同时观众评论爆发的 7，8，9，10 号也是流浪地球最为火爆的日期，同时观众评论数量会随着周末有一定的波动性，这也与电影票房曲线吻合。



图 14-《流浪地球》观众评论走势与时间的关系

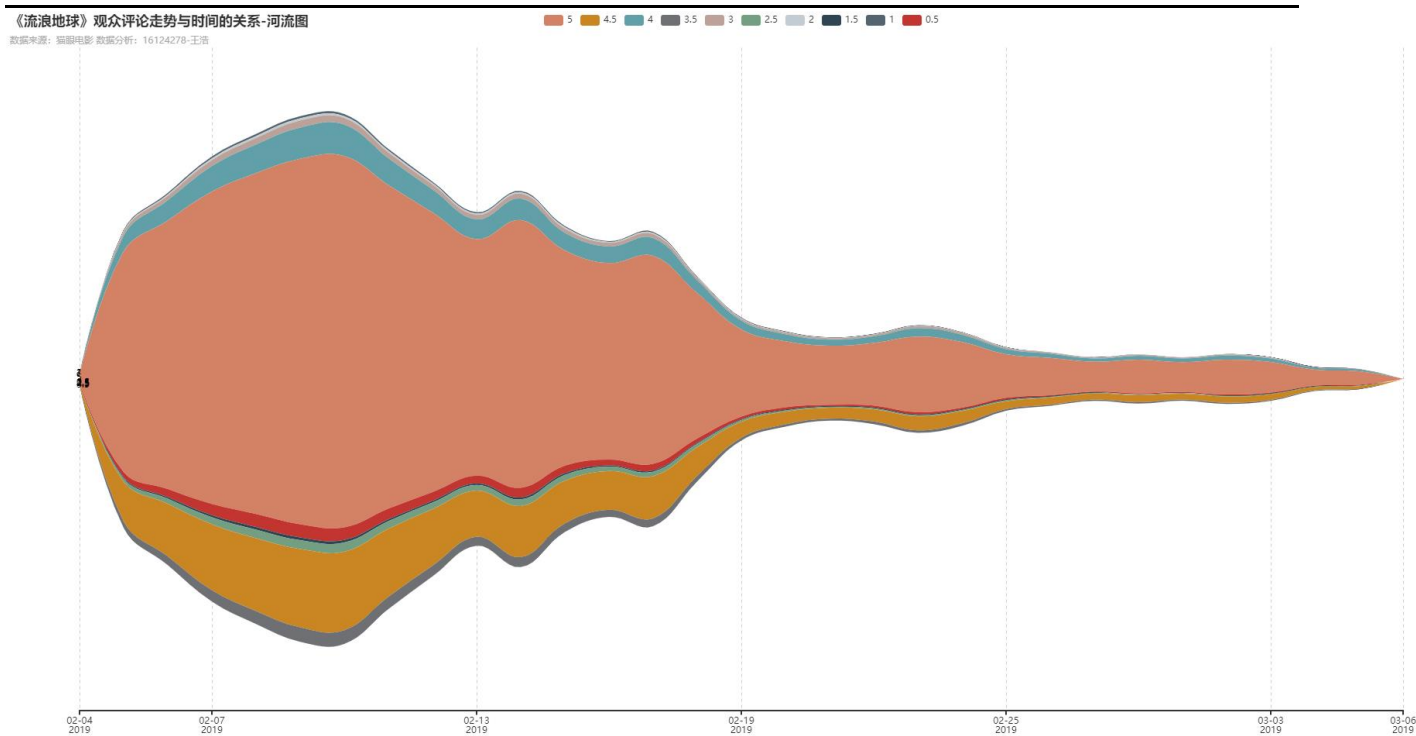


图 15-观众评论走势与时间的关系-河流图

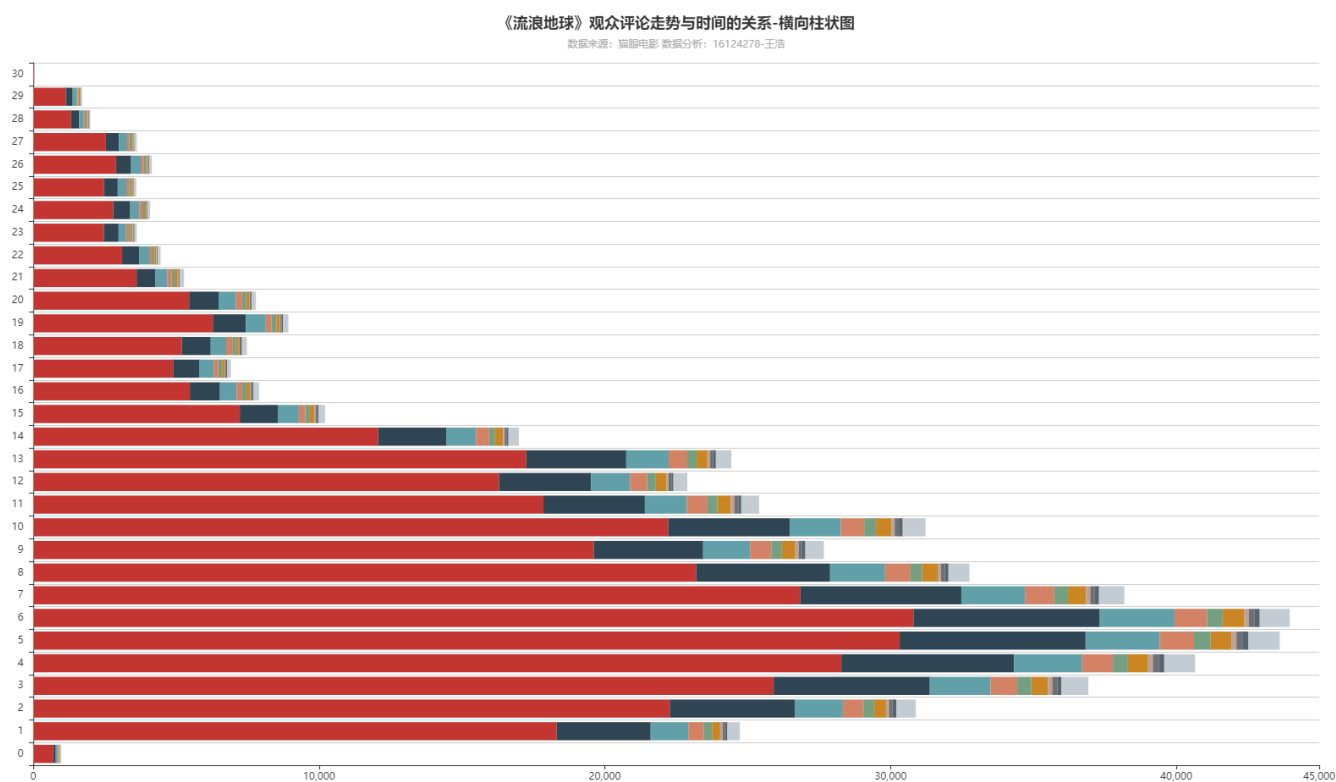


图 16-观众评论走势与时间的关系-横向柱状图

根据三幅观众评论走势与时间的关系图，折线图，河流图，横向柱状图（图 14，15，16）发现，观众的主要评论都是 4.5 星与 5 星，绝大多数观众都对这部电影给予了满分或者接近满分的好评，各种影评比例占比基本没有变化，网传的水军恶意刷差评影响获取并不是特别大，同时影评数量的波动也体现了春节档电影票房的走势。

《流浪地球》观众评论走势与时间的关系-河流图

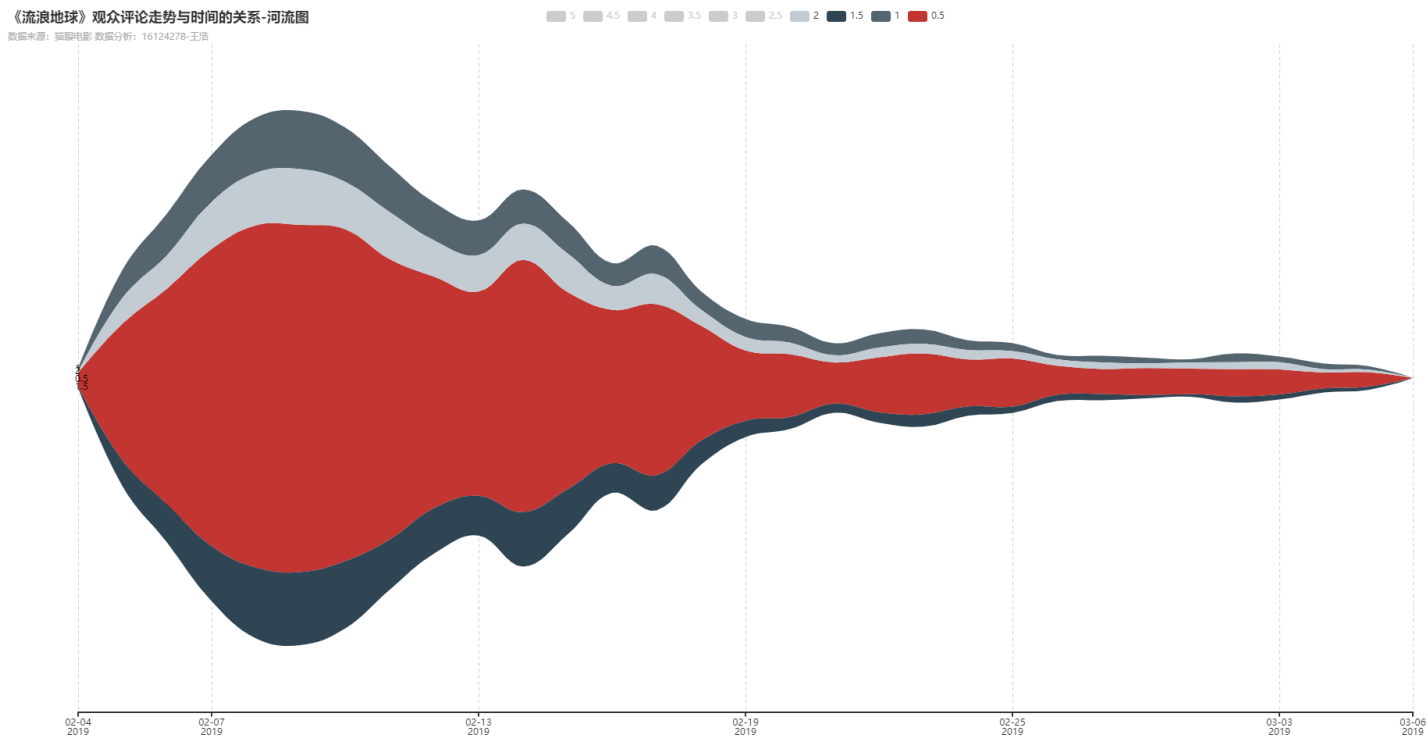


图 17-观众差评评论走势与时间的关系-河流图

但是，结合观众差评评论走势与时间的关系（图 17）在差评评论中，最低分 0.5 分（红色）的占比却明显的高于 2，1.5，1 分的占比，甚至高于这些分数的总和，由此可以说明，对于这部电影，还是有许多网友在评论的时候是抱有偏见的，想打差评的人在恶意差评拉低电影评分，由于数据量在整体比例中所占比例不高，对整体评分的也没有造成影响。

《流浪地球》观众评论数量与时间的关系

数据来源：猫眼电影 数据分析：16124278-王浩

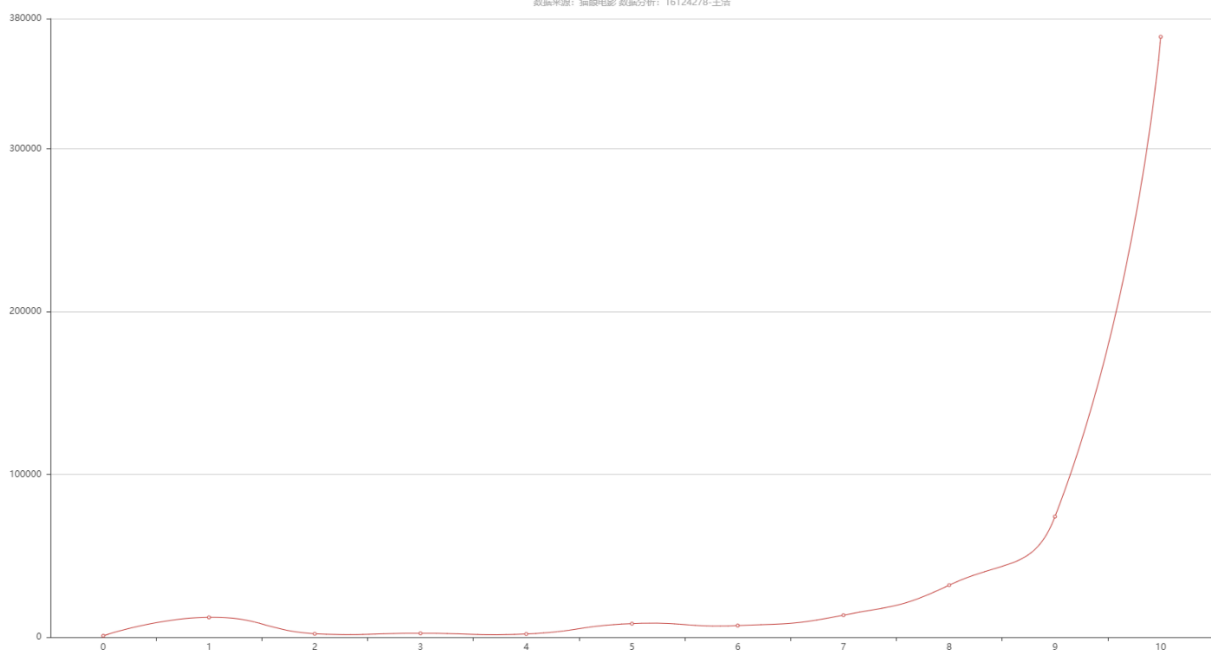
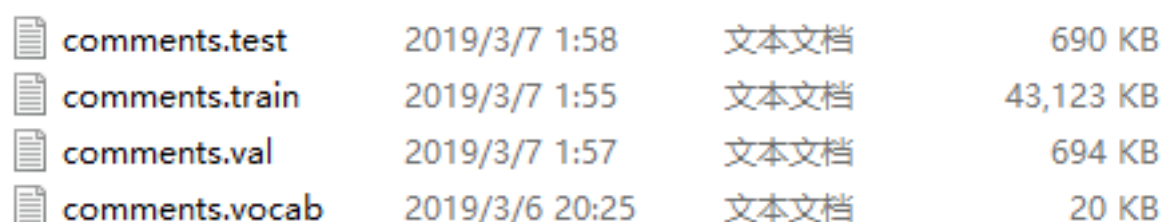


图 18-《流浪地球》观众情感曲线

4 情感分析介绍

4.1 数据处理（深一步）

通过 `Comments_Group.py`, 对爬虫获取的 54 万条再做进一步的预处理, 后删除掉 2 万余条数据, 再除去影评内容空格, 换行, 制表符等会对后面模型数据导入产生影响的符号后, 根据评分将评论数据分为中评好评和差评 (`judgeRank(score)`), 10000 条数据用做测试集, 10000 条数据用作验证集, 剩余的 503046 条有效数据用作训练集输出, 并根据影评内容常用字典 (5000 字) 便于文字 id 化。







	<code>comments.test</code>	2019/3/7 1:58	文本文档	690 KB
	<code>comments.train</code>	2019/3/7 1:55	文本文档	43,123 KB
	<code>comments.val</code>	2019/3/7 1:57	文本文档	694 KB
	<code>comments.vocab</code>	2019/3/6 20:25	文本文档	20 KB

图 20-处理后的影评数据

在模型开始训练, 对数据 id 进行 id 化, 利用 `comments_loader.py` 中的 `build_vocab`

() 根据影评数据提取影评中的常用 5000 字构建词汇表, 然后根据[“好评”, “中评”, “差评”]制作分类目录, 建立一个类别和 id 的字典; 一句词汇表建立词汇目录, 为每一条影评中的每个字建立一个 id, 将每一条影评数据 id 化。

4.2 CNN 与 LSTM, RGU 模型建立

CNN 模型以及参数储存在 `Cnn_Model.py` 中, LSTM 模型 GRU 模型和参数统一储存在 `Rnn_Model.py` 中, 均基于 `tensorflow` 构建网络。

考虑到做文本情感分析, 本文中采用的 CNN 模型为一层卷积层加最大池化层以及一层全连接层的配置, 后面接 `dropout` 避免过拟合以及 `relu` 激活, 采用交叉熵作为损失函数, `AdamOptimizer` 作为优化器更新整个网络。

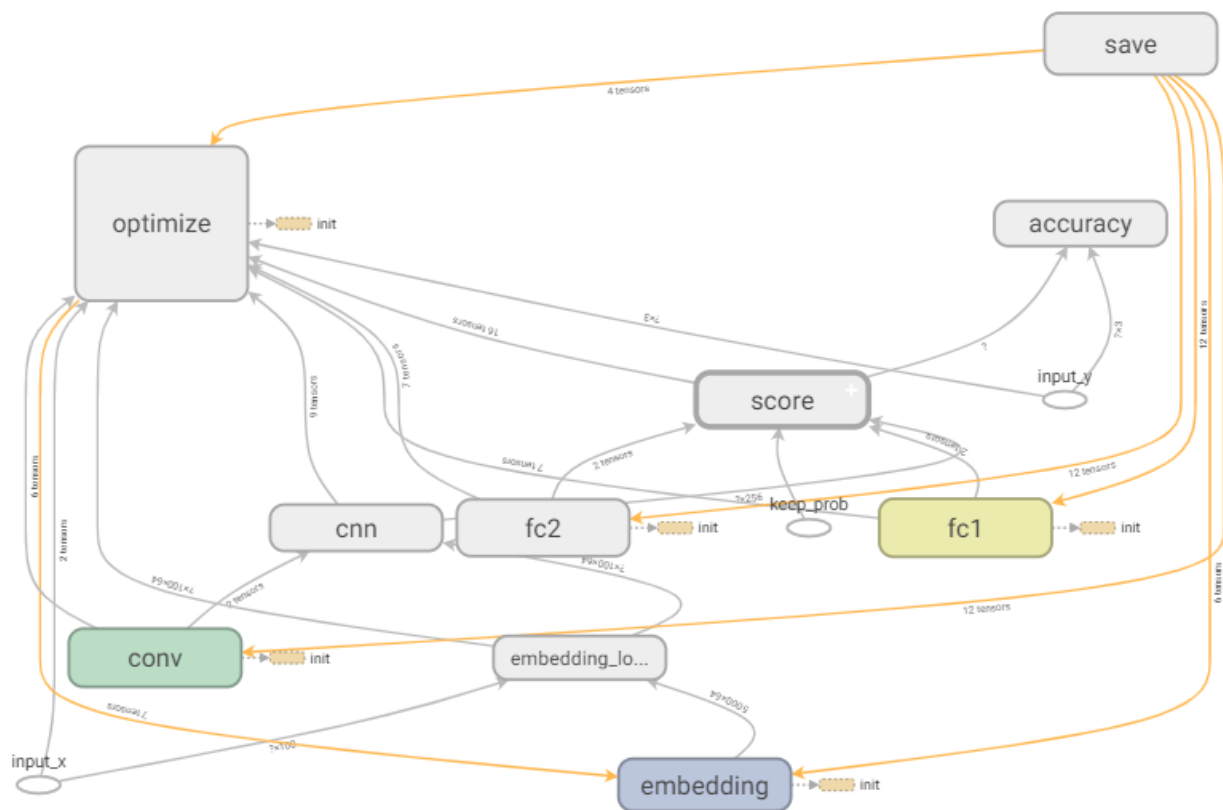


图 21- CNN 模型

同样的，LSTM 模型与 GRU 模型由各自的 lstm 核与 gru 和构建，双层组装，基于时序分析，利用 `dynamic_rnn()` 取最后一个时序输出作为结果输出后，同样接 `dropout` 避免过拟合以及 `relu` 激活，采用交叉熵作为损失函数，`AdamOptimizer` 作为优化器更新整个网络分别完成 LSTM 模型与 GRU 模型的构建。

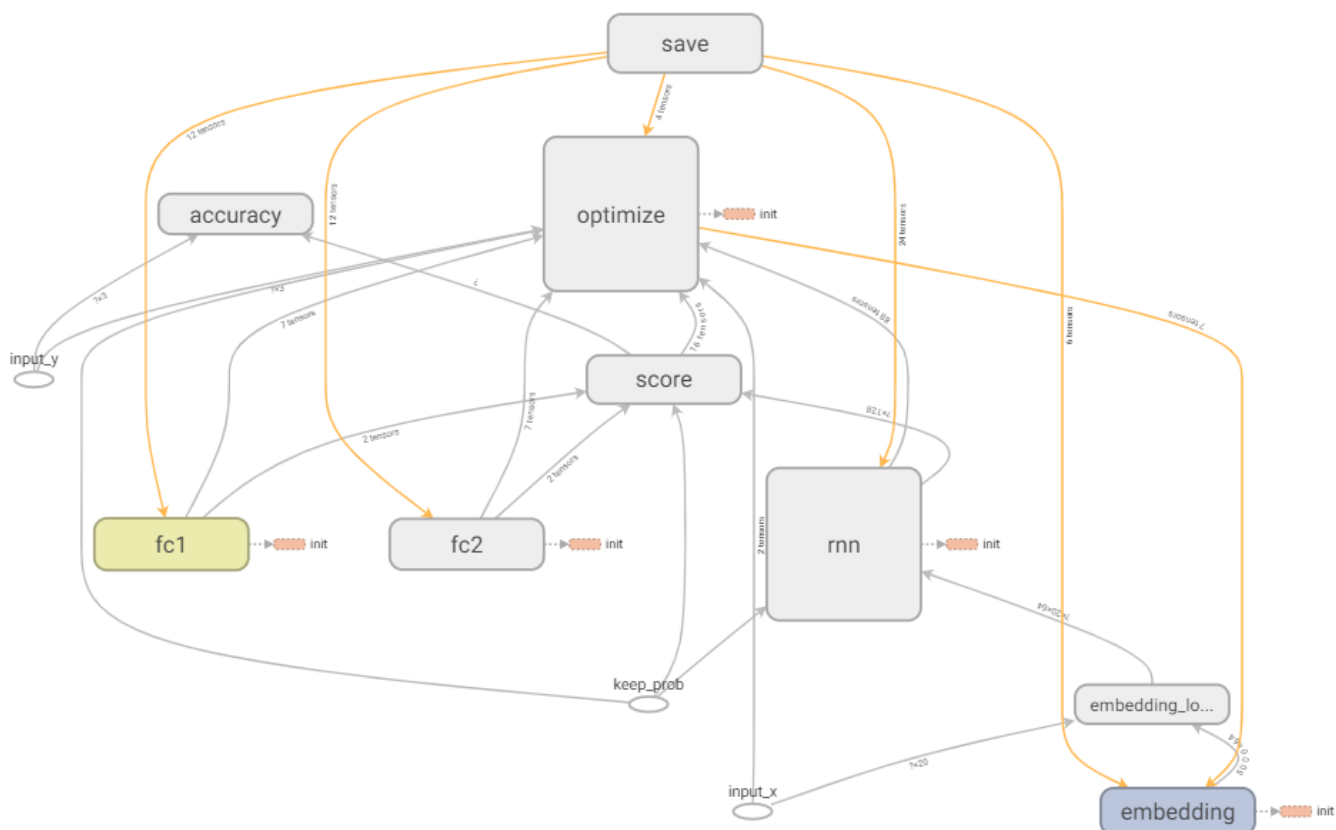


图 22- LSTM 与 GRU 模型

4.3 模型训练（以 LSTM 模型为例）

在 Run_Rnn.py 中根据预先设置好的参数初始化 LSTM 模型后，（训练产生的模型输出至./checkpoints，日志记录输出的参数记录保存至./tensorboard）开始训练。

配置 TensorBoard 和 Saver 保存可视化参数，载入训练集与验证集，利用添加的 process_file（）函数对载入的数据进行归一化处理使数据得以输入，即使用 keras 提供的 pad_sequences 来将文本 pad 为固定长度,取每句影评后 max_length 个字，取不到的地方则置为 0，并将将标签转换为 one-hot 编码表示。

设置实时数据显示后，开始模型训练，该模型在验证集上的准确率到达 98%以上或者超过 600 轮未提升，则提前自动结束训练。

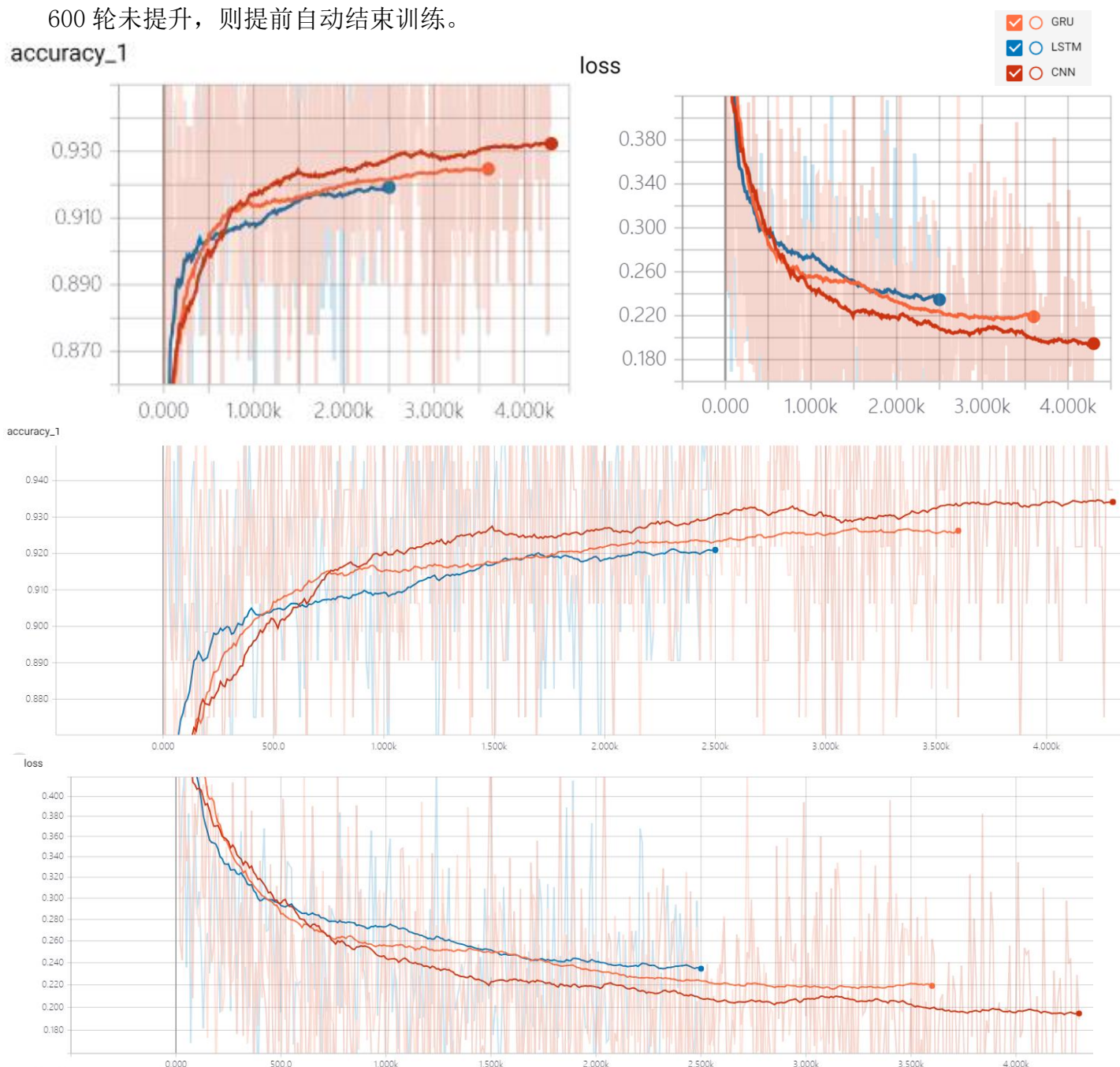


图 23- CNN, LSTM,GRU 模型训练过程（准确值与 loss 值）

4.4 模型验证（以 CNN 和 GRU 模型为例）

CNN 模型与 LSTM, GRU 模型在最终测试集的准确率相差不大，在参数调优并长时间训练之后，在验证集最高的准确率也仅仅在 93%左右，测试集则徘徊在 92%左右，召回率，F1 均值也相差无几，这三个模型均在好评准确率很高，而在中评和差评和处的准确率较低，从混淆矩阵的输出结果也可以看出，训练出的模型对中差评力不从心。

评论内容	预测结果
真是没啥意思，很一般	差评
非常满意，好看，人少环境不错，就是去晚了，开头没看全	好评
很感人 很励志 很科幻 户口很帅，	好评
不管情节怎样，起码我们国家终于有了	好评
没看懂！！！！	好评
特效确实不错，就是故事的情节我觉得还不够丰富，带着爸爸妈妈看的，	好评
特效爆炸，吹爆流浪地球	好评

表 1- RGU 模型单句预测结果

	准确率	召回率	F1 均值	样本数量
好评	0.94	0.99	0.96	9000
中评	0.61	0.24	0.35	688
差评	0.59	0.57	0.58	312
avg/total	0.91	0.92	0.91	10000

混淆矩阵			
		8890	74 36
		434	166 88
		104	30 178

表 2- CNN 模型测试测结果

	准确率	召回率	F1 均值	样本数量
好评	0.94	0.99	0.96	9000
中评	0.62	0.19	0.29	688
差评	0.56	0.48	0.52	312
avg/total	0.9	0.92	0.9	10000

混淆矩阵			
	8930	61	36
	474	132	82
	141	20	151

表 3- GRU 模型测试测结果

4.5 实验结果分析

三个模型的最终效果接近，尤其是本应好于 CNN 的 LSTM 与 GRU 模型在部分测试中甚至不敌 CNN，这是由于，影评单条数据量不大，文字内容普遍在 20 字左右，很难让拥有记忆性的 LSTM 和 GRU 模型的优势发挥出来，如果文本内容时整篇文章形式的长文本，LSTM 和 GRU 应该会有更好的表现。

中差评效果不好是因为数据样本太少，尽管总数据很大，但，其中，中差评数量占比极低，样本的不充裕导致模型的“偏科”

参考文献：

- [1] "Tiled convolutional neural networks". Q. V. Le, J. Ngiam, Z. Chen. Neural Information Processing Systems . 2010
- [2] 基于语义的中文在线评论情感分析[J]. 史伟, 王洪伟, 何绍义. 情报学报. 2013 (08)
- [3] 大规模情感词典的构建及其在情感分类中的应用[J]. 赵妍妍, 秦兵, 石秋慧, 刘挺. 中文信息学报. 2017 (02)
- [4] 情感分类中基于词性嵌入的特征权重计算方法[J]. 于海燕, 陆慧娟, 郑文斌. 计算机工程与应用. 2017 (22)
- [5] 用于微博情感分析的一种情感语义增强的深度学习模型[J]. 何炎祥, 孙松涛, 牛菲菲, 李飞. 计算机学报. 2017 (04)
- [6] 融合显性和隐性特征的中文微博情感分析[J]. 陈铁明, 缪茹一, 王小号. 中文信息学报. 2016 (04)
- [7] 基于情感词典与语义规则的微博情感分析[J]. 陈国兰. 情报探索. 2016 (02)
- [8] 结合卷积神经网络和词语情感序列特征的中文情感分析[J]. 陈钊, 徐睿峰, 桂林, 陆勤. 中文信息学报. 2015 (06)

[9]基于多样化特征的中文微博情感分类方法研究[J]. 张志琳,宗成庆. 中文信息学报. 2015(04)

[10]基于主体句和句法依赖的微博情感倾向性分析[J]. 张书卿,周文,欧阳纯萍,饶婕,刘志明,阳小华. 南华大学学报(自然科学版). 2015(01)

[11]基于卷积记忆神经网络的微博短文本情感分析[J]. 郑啸,王义真,袁志祥,秦锋. 电子测量与仪器学报. 2018(03)

[12]基于双向 LSTM 模型的文本情感分类[J]. 任勉,甘刚. 计算机工程与设计. 2018(07)

附录：流浪地球影评爬取与分析 部分核心代码

16124278 王浩 流浪地球 Get_Data.py

-*- coding: UTF-8 -*-

import requests # 爬虫库

import json # 评论网站 (猫眼) json 数据解析

http://m.maoyan.com/mMDB/comments/movie/248906.json?v_=yes&offset=0&startTime=2019-02-05%2020:28:22

import time # 程序内部时间控制

import datetime # 获取时间

import pandas as pd

请求评论 api 接口

def requestApi(url):

headers = {

'accept': '*/*',

'user-agent': 'Mozilla/5.0 (iPhone; CPU iPhone OS 11_0 like Mac OS X) AppleWebKit/604.1.38 (KHTML, like Gecko) Version/11.0 Mobile/15A372 Safari/604.1',

} # 模拟手机客户端 (iPhone ios11 Safari) 获取网站 json 数据

try:

r = requests.get(url, headers=headers)

r.raise_for_status()

return r.text

except requests.HTTPError as e: # 异常抛出

print(e)

except requests.RequestException as e:

print(e)

except:

```
print("获取数据出错")

# 解析接口返回数据
def getData(html):
    json_data = json.loads(html)['cms']

    comments = []

    # 解析数据并存入数组
    try:
        for item in json_data:
            comment = []

            comment.append(item['nickName']) # 用户名

            comment.append(item['cityName'] if 'cityName' in item else "") # 所在城市（如果有则记录该数据，否则不记录）

            comment.append(item['content'].strip().replace('\n', "")) # 删除评论中的换行

            comment.append(item['score']) # 评分星级

            comment.append(item['startTime']) # 评论上交时间

            comments.append(comment)

        return comments

    except Exception as e:
        print(comment)
        print(e)

# 保存数据，写入 excel
def saveData(comments):
    filename = './input/Comments_new.csv'

    # 将评论数据以 csv 文件格式保存在当前文件夹下

    dataObject = pd.DataFrame(comments)

    dataObject.to_csv(filename, mode='a', encoding="utf_8_sig", index=False, sep=',', header=False)

    # 使用 utf_8_sig 对字节进行有序编码，防止在系统中用某些软件直接浏览浏览时出现乱码问题，影响阅读；

# 爬虫主函数
def main():
```

```
start = datetime.datetime.now()

# 当前时间

# start_time = datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')

# 开始抓取时间

start_time = '2019-03-06 00:00:00'

# 流浪地球电影上映前的所有评论

end_time = '2018-01-01 00:00:00'


init_time = start_time # 记录程序开始运行的时间

print('开始获取数据!', init_time)


while start_time > end_time:

    url = 'http://m.maoyan.com/mmdb/comments/movie/248906.json?_v=yes&offset=0&startTime=' + start_time.replace(
        ' ', '%20') # 改变 startTime 字段的值来获取更多评论信息, 把 offset 置为 0, 把每页评论数据中最后一次评论时间作为新的 startTime
    去重新请求

    html = None

    print(url)

    try:

        html = requestApi(url)

    except Exception as e: # 如果有异常,暂停一会再爬

        time.sleep(1) # 暂停一秒

        html = requestApi(url)

    # else: #开启慢速爬虫, 防止封禁 ip 地址

    # time.sleep(0.5)

    comments = getData(html)

    # print(url)

    start_time = comments[14][4] # 获取每页中最后一条评论时间,每页有 15 条评论

    # print(start_time)


    # 最后一条评论时间减一秒, 避免爬取重复数据

    start_time = datetime.datetime.strptime(start_time, '%Y-%m-%d %H:%M:%S') + datetime.timedelta(seconds=-1)

    end = datetime.datetime.now()

    print(start_time)

    saveData(comments)


print('获取数据完成! ')
```

```
print('程序运行时:', start - end)
```

```
if __name__ == '__main__':  
    main()
```

16124278 王浩 流浪地球 Analyze_Data.py

```
# -*- coding: UTF-8 -*-
```

```
import pandas as pd
```

```
from collections import Counter # 计数器
```

```
from pyecharts import Geo, Bar, Page, Style, ThemeRiver, Line # 数据可视化图表
```

```
from pyecharts.datasets.coordinates import search_coordinates_by_keyword # 关键字匹配（地名）
```

```
import jieba # 结巴分词
```

```
import jieba.analyse
```

```
import matplotlib.pyplot as plt # 绘图
```

```
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator # 词云
```

```
import datetime # 日期
```

```
startTime_tag = datetime.datetime.strptime('2019-02-04', '%Y-%m-%d') # 将 2019.2.4 号之前的数据汇总到 2.4 统一标识为电影上映前影评数据
```

```
# 提前自定义 pyecharts 渲染图格式
```

```
style_map = Style(  
    title_color="#fff",  
    title_pos="center",  
    width=1900,  
    height=950,  
    background_color='#404a59'  
)
```

```
style_others = Style(  
    title_pos='center',  
    width=1900,  
    height=950  
)
```

```
style_size = Style(  
    width=1900,  
    height=950  
)
```

读取 csv 文件数据

```
def read_csv(filename, titles):  
    print("正在读取 csv 文件数据.....")  
    comments = pd.read_csv(filename, names=titles, encoding='utf_8_sig') # 指定读取格式 utf_8_sig  
    return comments
```

数据城市名称与热力地图名称匹配（模糊匹配到热力图中的地址名称：例如 上海 = 上海市 舟山 = 舟山市）

```
def remove_None(areas, values):  
    None_areas = [] # 储存无效区域  
    i = 0 # 计数器  
    while (i != len(areas)):  
        # 取前两个字符作模糊查询  
        if search_coordinates_by_keyword(areas[i][:2]) == {}: # 匹配失败，删除这条数据  
            None_areas.append(areas[i])  
            areas.remove(areas[i])  
            values.remove(values[i])  
        else:  
            # 将模糊查询结果替代原地名  
            areas[i] = search_coordinates_by_keyword(areas[i][:2]).popitem()[0]  
            i = i + 1 # 计数器加一  
    if len(None_areas) == 0:  
        print('区域名称已全部匹配完成！')  
    else:  
        print('无效区域：', None_areas)  
    return areas, values
```

观众地域图

```
def draw_map(comments):  
    print("正在处理观众地域图.....")  
    try:  
        page = Page() # 页面储存器  
        attr = comments['cityName'].fillna("zero_token") # 将不包含城市数据的数据置为空标志  
        data = Counter(attr).most_common(300) # 计数并选取数据最多的 300 个城市  
        data.reverse() # 反转数据，为了热力图有更好的显示效果（数据量大的城市在最上方）  
        data.remove(data[data.index([(i, x) for i, x in (data) if i == 'zero_token'][0])]) # 删除没有城市名称的空标志数据
```

```
geo = Geo("《流浪地球》全国观众地域分布点", "数据来源: 猫眼电影 数据分析: 16124278-王浩", **style_map.init_style) # 初始一张热力图并设置外观
```

```
# attr, value = geo.cast(data)

attr, value = remove_None(geo.cast(data)[0], geo.cast(data)[1]) # 地图名称模糊匹配

geo.add("", attr, value, visual_range=[0, 4000], maptype='china', visual_text_color="#fff", symbol_size=10,

        is_visualmap=True, is_legend_show=False,

        tooltip_formatter='{b}',

        label_emphasis_textsize=15,

        label_emphasis_pos='right') # 加入数据并设置热力图风格参数-点图

page.add(geo) # 添加到渲染队列
```

```
geo = Geo("《流浪地球》全国观众地域分布域", "数据来源: 猫眼电影 数据分析: 16124278-王浩", **style_map.init_style) # 初始一张热力图并设置外观
```

```
geo.add("", attr, value, type="heatmap", is_visualmap=True,

        visual_range=[0, 4000], visual_text_color='#fff',

        is_legend_show=False) # 加入数据并设置热力图风格参数-区域图

page.add(geo) # 添加到渲染队列
```

```
geo = Geo("《流浪地球》全国观众地域分布点域", "数据来源: 猫眼电影 数据分析: 16124278-王浩", **style_map.init_style) # 初始一张热力图并设置外观
```

```
geo.add("", attr, value, visual_range=[0, 4000], maptype='china', visual_text_color="#fff", symbol_size=10,

        is_visualmap=True, is_legend_show=False,

        tooltip_formatter='{b}',

        label_emphasis_textsize=15,

        label_emphasis_pos='right')

geo.add("", attr, value, type="heatmap", is_visualmap=True,

        visual_range=[0, 4000], visual_text_color='#fff',

        is_legend_show=False) # 加入数据并设置热力图风格参数-区域图

page.add(geo) # 添加到渲染队列
```

```
page.render("./output/观众地域分布-地理坐标图.html") # 渲染热力图
```

```
print("全国观众地域分布已完成!!!")
```

```
except Exception as e: # 异常抛出
```

```
print(e)
```

```
# 观众地域排行榜单 (前 20)
```

```
def draw_bar(comments):
```

```
print("正在处理观众地域排行榜单.....")

data_top20 = Counter(comments['cityName']).most_common(20) # 筛选出数据量前二十的城市

bar = Bar('《流浪地球》观众地域排行榜单', '数据来源: 猫眼电影 数据分析: 16124278-王浩', **style_others.init_style) # 初始化柱状图

attr, value = bar.cast(data_top20) # 传值

bar.add('', attr, value, is_visualmap=True, visual_range=[0, 16000], visual_text_color='black', is_more_utils=True,

        is_label_show=True) # 加入数据与其它参数

bar.render('./output/观众地域排行榜单-柱状图.html') # 渲染

print("观众地域排行榜单已完成!!!")
```

```
# lambda 表达式内置函数
```

```
# 将 startTime_tag 之前的数据汇总到 startTime_tag
```

```
def judgeTime(time, startTime_tag):
```

```
    if time < startTime_tag:
```

```
        return startTime_tag
```

```
    else:
```

```
        return time
```

```
# 观众评论数量与日期的关系
```

```
def draw_DateBar(comments):
```

```
    print("正在处理观众评论数量与日期的关系.....")
```

```
    time = pd.to_datetime(comments['startTime']) # 获取评论时间并转换为标准日期格式
```

```
    time = time.apply(lambda x: judgeTime(x, startTime_tag)) # 将 2019.2.4 号之前的数据汇总到 2.4 统一标识为电影上映前影评数据
```

```
    timeData = []
```

```
    for t in time:
```

```
        if pd.isnull(t) == False: # 获取评论日期（删除具体时间）并记录
```

```
            t = str(t) # 转换为字符串以便分割
```

```
            date = t.split(' ')[0]
```

```
            timeData.append(date)
```

```
data = Counter(timeData).most_common() # 记录相应日期对应的评论数
```

```
data = sorted(data, key=lambda data: data[0]) # 使用 lambda 表达式对数据按日期进行排序
```

```
bar = Bar('《流浪地球》观众评论数量与日期的关系', '数据来源: 猫眼电影 数据分析: 16124278-王浩', **style_others.init_style) # 初始化柱状图
```

```
attr, value = bar.cast(data) # 传值
```

```
bar.add('', attr, value, is_visualmap=True, visual_range=[0, 43000], visual_text_color='black', is_more_utils=True,
```

```
        is_label_show=True) # 加入数据和其它参数
```

```
bar.render('./output/观众评论日期-柱状图.html') # 渲染  
print("观众评论数量与日期的关系已完成!!!")
```

```
# 观众情感曲线
```

```
def draw_sentiment_pic(comments):  
    print("正在处理观众情感曲线.....")  
    score = comments['score'].dropna() # 获取观众评分  
    data = Counter(score).most_common() # 记录相应评分对应的评论数  
    data = sorted(data, key=lambda data: data[0]) # 使用 lambda 表达式对数据按评分进行排序  
    line = Line('《流浪地球》观众评论数量与时间的关系', '数据来源: 猫眼电影 数据分析: 16124278-王浩', **style_others.init_style) # 初始化  
    attr, value = line.cast(data) # 传值  
  
    for i, v in enumerate(attr): # 将分数修改为整数便于渲染图上的展示  
        attr[i] = v * 2  
  
    line.add("", attr, value, is_smooth=True, is_more_utils=True, yaxis_max=380000, xaxis_max=10) # 加入数据和其它参数  
    line.render("./output/观众情感分析-曲线图.html") # 渲染  
    print("观众情感曲线已完成!!!")
```

```
# 观众评论数量与时间的关系图
```

```
def draw_TimeBar(comments):  
    print("正在处理观众评论数量与时间的关系.....")  
    time = comments['startTime'].dropna() # 获取评论时间  
    timeData = []  
    for t in time:  
        if pd.isnull(t) == False: # 获取评论时间（当天小时）并记录  
            time = t.split(' ')[1]  
            hour = time.split(':')[0]  
            timeData.append(int(hour)) # 转化为整数便于排序  
  
    data = Counter(timeData).most_common() # 记录相应时间对应的评论数  
    data = sorted(data, key=lambda data: data[0]) # 使用 lambda 表达式对数据按时间进行排序  
  
    bar = Bar('《流浪地球》观众评论数量与时间的关系', '数据来源: 猫眼电影 数据分析: 16124278-王浩', **style_others.init_style) # 初始化柱状图  
    attr, value = bar.cast(data) # 传值  
    bar.add("", attr, value, is_visualmap=True, visual_range=[0, 40000], visual_text_color='black', is_more_utils=True,
```

```
is_label_show=True) # 加入数据和其它参数

bar.render('./output/观众评论时间-柱状图.html') # 渲染

print("观众评论数量与时间的关系已完成!!!")


# 观众评论走势与时间的关系

def draw_score(comments):

    print("正在处理观众评论走势与时间的关系.....")

    page = Page() # 页面存储器

    score, date, value, score_list = [], [], [], []

    result = {} # 存储评分结果


    d = comments[['score', 'startTime']].dropna() # 获取评论时间

    d['startTime'] = d['startTime'].apply(lambda x: pd.to_datetime(x.split(' ')[0])) # 获取评论日期（删除具体时间）并记录

    d['startTime'] = d['startTime'].apply(lambda x: judgeTime(x, startTime_tag)) # 将 2019.2.4 号之前的数据汇总到 2.4 统一标识为电影上映前影评数据


    for indexs in d.index: # 一种遍历 df 行的方法（下面还有第二种，iterrows）

        score_list.append(tuple(d.loc[indexs].values[:])) # 评分与日期连接 转换为 tuple 然后统计相同元素个数

    print("有效评分总数量为：", len(score_list), " 条")

    for i in set(list(score_list)):

        result[i] = score_list.count(i) # dict 类型，统计相同日期相同评分对应数


    info = []

    for key in result:

        score = key[0] # 取分数

        date = key[1] # 日期

        value = result[key] # 数量

        info.append([score, date, value])

    info_new = pd.DataFrame(info) # 将字典转换成为数据框

    info_new.columns = ['score', 'date', 'votes']

    info_new.sort_values('date', inplace=True) # 按日期升序排列 df，便于找最早 date 和最晚 data，方便后面插值


    # 以下代码用于插入空缺的数据，每个日期的评分类型应该有 10 种，依次遍历判断是否存在，若不存在则往新的 df 中插入新数值

    mark = 0

    creat_df = pd.DataFrame(columns=['score', 'date', 'votes']) # 创建空的 dataframe

    for i in list(info_new['date']):

        location = info_new[(info_new.date == i) & (info_new.score == 5.0)].index.tolist()

        if location == []:
```

```
    creat_df.loc[mark] = [5.0, i, 0]

    mark += 1

location = info_new[(info_new.date == i) & (info_new.score == 4.5)].index.tolist()

if location == []:

    creat_df.loc[mark] = [4.5, i, 0]

    mark += 1

location = info_new[(info_new.date == i) & (info_new.score == 4.0)].index.tolist()

if location == []:

    creat_df.loc[mark] = [4.0, i, 0]

    mark += 1

location = info_new[(info_new.date == i) & (info_new.score == 3.5)].index.tolist()

if location == []:

    creat_df.loc[mark] = [3.5, i, 0]

    mark += 1

location = info_new[(info_new.date == i) & (info_new.score == 3.0)].index.tolist()

if location == []:

    creat_df.loc[mark] = [3.0, i, 0]

    mark += 1

location = info_new[(info_new.date == i) & (info_new.score == 2.5)].index.tolist()

if location == []:

    creat_df.loc[mark] = [2.5, i, 0]

    mark += 1

location = info_new[(info_new.date == i) & (info_new.score == 2.0)].index.tolist()

if location == []:

    creat_df.loc[mark] = [2.0, i, 0]

    mark += 1

location = info_new[(info_new.date == i) & (info_new.score == 1.5)].index.tolist()

if location == []:

    creat_df.loc[mark] = [1.5, i, 0]

    mark += 1

location = info_new[(info_new.date == i) & (info_new.score == 1.0)].index.tolist()

if location == []:

    creat_df.loc[mark] = [1.0, i, 0]

    mark += 1

location = info_new[(info_new.date == i) & (info_new.score == 0.5)].index.tolist()

if location == []:

    creat_df.loc[mark] = [0.5, i, 0]

    mark += 1
```

```

info_new = info_new.append(creat_df.drop_duplicates(), ignore_index=True)

score_list = [] # 重置 score_list

info_new = info_new[~(info_new['score'] == 0.0)] # 剔除无评分的数据

info_new.sort_values('date', inplace=True) # 按日期升序排列 df, 便于找最早 date 和最新 date, 方便后面插值

for index, row in info_new.iterrows(): # 第二种遍历 df 的方法

    score_list.append([row['date'], row['votes'], row['score']])


tr = ThemeRiver('《流浪地球》观众评论走势与时间的关系-河流图', '数据来源: 猫眼电影 数据分析: 16124278-王浩', **style_size.init_style)
# 河流图初始化

tr.add([5.0, 4.5, 4.0, 3.5, 3.0, 2.5, 2.0, 1.5, 1.0, 0.5], score_list, is_label_show=True,

        is_more_utils=True) # 设置参数

page.add_chart(tr) # 加入渲染队列


attr, v1, v2, v3, v4, v5, v6, v7, v8, v9, v10 = [], [], [], [], [], [], [], [], [], []

attr = list(sorted(set(info_new['date'])))

bar = Bar('《流浪地球》观众评论走势与时间的关系-横向柱状图', '数据来源: 猫眼电影 数据分析: 16124278-王浩', **style_others.init_style)
# 初始化图表

for i in attr:

    v1.append(int(info_new[(info_new['date'] == i) & (info_new['score'] == 5.0)][ 'votes']))

    v2.append(int(info_new[(info_new['date'] == i) & (info_new['score'] == 4.5)][ 'votes']))

    v3.append(int(info_new[(info_new['date'] == i) & (info_new['score'] == 4.0)][ 'votes']))

    v4.append(int(info_new[(info_new['date'] == i) & (info_new['score'] == 3.5)][ 'votes']))

    v5.append(int(info_new[(info_new['date'] == i) & (info_new['score'] == 3.0)][ 'votes']))

    v6.append(int(info_new[(info_new['date'] == i) & (info_new['score'] == 2.5)][ 'votes']))

    v7.append(int(info_new[(info_new['date'] == i) & (info_new['score'] == 2.0)][ 'votes']))

    v8.append(int(info_new[(info_new['date'] == i) & (info_new['score'] == 1.5)][ 'votes']))

    v9.append(int(info_new[(info_new['date'] == i) & (info_new['score'] == 1.0)][ 'votes']))

    v10.append(int(info_new[(info_new['date'] == i) & (info_new['score'] == 0.5)][ 'votes']))

bar.add(5.0, attr, v1, is_stack=True)

bar.add(4.5, attr, v2, is_stack=True)

bar.add(4.0, attr, v3, is_stack=True)

bar.add(3.5, attr, v4, is_stack=True)

bar.add(3.0, attr, v5, is_stack=True)

bar.add(2.5, attr, v6, is_stack=True)

bar.add(2.0, attr, v7, is_stack=True)

bar.add(1.5, attr, v8, is_stack=True)

bar.add(1.0, attr, v9, is_stack=True)

bar.add(0.5, attr, v10, is_stack=True, is_convert=True, is_more_utils=True, xaxis_max=45000)

```

```
page.add_chart(bar)
```

```
line = Line('《流浪地球》观众评论走势与时间的关系', '数据来源: 猫眼电影 数据分析: 16124278-王浩', **style_others.init_style) # 初始化图表
```

```
line.add(5.0, attr, v1, is_stack=True, mark_line=["average"])
line.add(4.5, attr, v2, is_stack=True, mark_line=["average"])
line.add(4.0, attr, v3, is_stack=True, mark_line=["average"])
line.add(3.5, attr, v4, is_stack=True, mark_line=["average"])
line.add(3.0, attr, v5, is_stack=True, mark_line=["average"])
line.add(2.5, attr, v6, is_stack=True, mark_line=["average"])
line.add(2.0, attr, v7, is_stack=True, mark_line=["average"])
line.add(1.5, attr, v8, is_stack=True, mark_line=["average"])
line.add(1.0, attr, v9, is_stack=True, mark_line=["average"])
line.add(0.5, attr, v10, is_stack=True, is_convert=False, mark_line=["average"], is_more_utils=True,
        yaxis_max=45000)
page.add_chart(line)
```

```
page.render("/output/观众评论与日投票-走势图.html") # 渲染
```

```
print("观众评论走势与时间的关系已完成!!!")
```

```
# 绘制词云
```

```
def draw_wordCloud(comments):
```

```
    print("数据量较大, 正在分词中, 请耐心等待.....")
```

```
    data = comments['content'] # 获取评论内容
```

```
    comment_data = []
```

```
    for item in data:
```

```
        if pd.isnull(item) == False:
```

```
            comment_data.append(item)
```

```
    comment_after_split = jieba.cut(str(comment_data), cut_all=False) # jieba 分词
```

```
    words = ' '.join(comment_after_split) # 连接分词
```

```
    backgroud_Image = plt.imread('/input/worldcloud_sample.jpg') # 设置词云背景图
```

```
    # 自定义停用词
```

```
    stopwords = STOPWORDS.copy()
```

```
    with open('/input/stopwords.txt', 'r', encoding='utf-8') as f: # 打开文件读取停用词
```

```
        for i in f.readlines():
```

```
            stopwords.add(i.strip('\n'))
```

```
f.close()
```

```
# 字体路径
```

```
wc = WordCloud(width=1024, height=768, background_color='white',
```

```
               mask=backgroud_Image, font_path="C:\\simhei.ttf",
```

```
               stopwords=stopwords, max_font_size=500,
```

```
               random_state=80) # 设置词云参数
```

```
wc.generate_from_text(words) # 传入关键词
```

```
img_colors = ImageColorGenerator(backgroud_Image) # 取背景图色彩
```

```
wc.recolor(color_func=img_colors) # 给词云上色美化
```

```
# plt.figure(figsize=(10, 8))
```

```
plt.imshow(wc) # 设置参数
```

```
plt.axis('off') # 关闭坐标轴显示
```

```
plt.savefig('./output/WordCloud.png', dpi=300) # 保存高清打印图片
```

```
plt.show() # 展示
```

```
if __name__ == "__main__":
```

```
    start_time = datetime.datetime.now()
```

```
    filename = "./input/Comments.csv"
```

```
    titles = ['nickName', 'cityName', 'content', 'score', 'startTime']
```

```
    comments = read_csv(filename, titles)
```

```
    draw_map(comments)
```

```
    draw_bar(comments)
```

```
    draw_DateBar(comments)
```

```
    draw_TimeBar(comments)
```

```
    draw_score(comments)
```

```
    draw_sentiment_pic(comments)
```

```
    draw_wordCloud(comments)
```

```
    end_time = datetime.datetime.now()
```

```
    print("全部完成!!!")
```

```
    print('程序运行用时(秒):', (end_time - start_time).seconds)
```

```
# 16124278 王浩 流浪地球 Run_Rnn.py
```

```
# -*- coding: utf-8 -*-
```

```
from __future__ import print_function
```

```
import os

import sys

import time

from datetime import timedelta


import numpy as np

import tensorflow as tf

from sklearn import metrics


from Rnn_Model import TRNNConfig, TextRNN

from comments.comments_loader import read_vocab, read_category, batch_iter, process_file, build_vocab


base_dir = './comments/'

train_dir = os.path.join(base_dir, 'comments.train.txt') # 训练集

test_dir = os.path.join(base_dir, 'comments.test.txt')# 测试集

val_dir = os.path.join(base_dir, 'comments.val.txt')# 验证集

vocab_dir = os.path.join(base_dir, 'comments.vocab.txt')# 影评字典


save_dir = './checkpoints/texttrnn'

save_path = os.path.join(save_dir, 'best_validation') # 最佳验证结果保存路径


def get_time_dif(start_time):

    """获取已使用时间"""

    end_time = time.time()

    time_dif = end_time - start_time

    return timedelta(seconds=int(round(time_dif)))


def feed_data(x_batch, y_batch, keep_prob):

    feed_dict = {

        model.input_x: x_batch,

        model.input_y: y_batch,

        model.keep_prob: keep_prob

    }

    return feed_dict
```

```
def evaluate(sess, x_, y_):

    """评估在某一数据上的准确率和损失"""

    data_len = len(x_)

    batch_eval = batch_iter(x_, y_, 128)

    total_loss = 0.0

    total_acc = 0.0

    for x_batch, y_batch in batch_eval:

        batch_len = len(x_batch)

        feed_dict = feed_data(x_batch, y_batch, 1.0)

        y_pred_class, loss, acc = sess.run([model.y_pred_cls, model.loss, model.acc], feed_dict=feed_dict)

        total_loss += loss * batch_len

        total_acc += acc * batch_len

    return y_pred_class, total_loss / data_len, total_acc / data_len
```

```
def train():

    print("正在配置 TensorBoard 和 Saver.....")

    # 配置 Tensorboard，重新训练时，请将 tensorboard 文件夹删除，不然图会覆盖

    tensorboard_dir = './tensorboard/text rnn'

    if not os.path.exists(tensorboard_dir):

        os.makedirs(tensorboard_dir)

    tf.summary.scalar("loss", model.loss)

    tf.summary.scalar("accuracy", model.acc)

    merged_summary = tf.summary.merge_all()

    writer = tf.summary.FileWriter(tensorboard_dir)

    # 配置 Saver

    saver = tf.train.Saver()

    if not os.path.exists(save_dir):

        os.makedirs(save_dir)

    print("载入训练集与验证集.....")

    # 载入训练集与验证集

    start_time = time.time()

    x_train, y_train = process_file(train_dir, word_to_id, cat_to_id, config.seq_length)

    x_val, y_val = process_file(val_dir, word_to_id, cat_to_id, config.seq_length)
```

```
time_dif = get_time_dif(start_time)

print("Time usage:", time_dif)


# 创建 session

session = tf.Session()

session.run(tf.global_variables_initializer())

writer.add_graph(session.graph)


print('正在训练评估中...')

start_time = time.time()

total_batch = 0 # 总批次

best_acc_val = 0.0 # 最佳验证集准确率

last_improved = 0 # 记录上一次提升批次

require_improvement = 600 # 如果超过 1000 轮未提升，提前结束训练


flag = False

for epoch in range(config.num_epochs):

    print('Epoch:', epoch + 1)

    batch_train = batch_iter(x_train, y_train, config.batch_size)

    for x_batch, y_batch in batch_train:

        feed_dict = feed_data(x_batch, y_batch, config.dropout_keep_prob)


        if total_batch % config.save_per_batch == 0:

            # 每多少轮次将训练结果写入 tensorboard scalar

            s = session.run(merged_summary, feed_dict=feed_dict)

            writer.add_summary(s, total_batch)


        if total_batch % config.print_per_batch == 0:

            # 每多少轮次输出在训练集和验证集上的性能

            feed_dict[model.keep_prob] = 1.0

            loss_train, acc_train = session.run([model.loss, model.acc], feed_dict=feed_dict)

            y_pred, loss_val, acc_val = evaluate(session, x_val, y_val) # todo


            if acc_val > best_acc_val:

                # 保存最好结果

                best_acc_val = acc_val

                last_improved = total_batch

                saver.save(sess=session, save_path=save_path)
```

```
        improved_str = '*'
    else:
        improved_str = ''

    time_dif = get_time_dif(start_time)

    msg = 'Iter: {0:>6}, Train Loss: {1:>6.2}, Train Acc: {2:>7.2%}, ' \
        + ' Val Loss: {3:>6.2}, Val Acc: {4:>7.2%}, Time: {5} {6}'

    print(msg.format(total_batch, loss_train, acc_train, loss_val, acc_val, time_dif, improved_str))

    session.run(model.optim, feed_dict=feed_dict) # 运行优化

    total_batch += 1

    if total_batch - last_improved > require_improvement:
        # 验证集正确率长期不提升，提前结束训练

        print("验证集正确率长期不提升，自动提前结束训练!!!")

        flag = True

        break # 跳出循环

    if flag: # 同上

        break


def test():

    print("加载测试集.....")

    start_time = time.time()

    x_test, y_test = process_file(test_dir, word_to_id, cat_to_id, config.seq_length)

    session = tf.Session()

    session.run(tf.global_variables_initializer())

    saver = tf.train.Saver()

    saver.restore(sess=session, save_path=save_path) # 读取保存的模型

    print('测试中.....')

    y_pred, loss_test, acc_test = evaluate(session, x_test, y_test)

    msg = 'Test Loss: {0:>6.2}, Test Acc: {1:>7.2%}'

    print(msg.format(loss_test, acc_test))

    batch_size = 128

    data_len = len(x_test)

    num_batch = int((data_len - 1) / batch_size) + 1
```

```
y_test_cls = np.argmax(y_test, 1)

y_pred_cls = np.zeros(shape=len(x_test), dtype=np.int32) # 保存预测结果

for i in range(num_batch): # 逐批次处理

    start_id = i * batch_size

    end_id = min((i + 1) * batch_size, data_len)

    feed_dict = {

        model.input_x: x_test[start_id:end_id],

        model.keep_prob: 1.0

    }

    y_pred_cls[start_id:end_id] = session.run(model.y_pred_cls, feed_dict=feed_dict)


# 评估

print("准确率, 召回率, F1 均值.....")

print(metrics.classification_report(y_test_cls, y_pred_cls, target_names=categories))


# 混淆矩阵

print("混淆矩阵处理中.....")

cm = metrics.confusion_matrix(y_test_cls, y_pred_cls)

print(cm)


time_dif = get_time_dif(start_time)

print("用时:", time_dif)


if __name__ == '__main__':

    config = TRNNConfig()

    if not os.path.exists(vocab_dir): # 如果不存在词汇表, 重建

        build_vocab(train_dir, vocab_dir, config.vocab_size)

    categories, cat_to_id = read_category()

    words, word_to_id = read_vocab(vocab_dir)

    config.vocab_size = len(words)

    model = TextRNN(config)

    option='train'

    if option == 'train':

        train()

    else:

        test()
```

16124278 王浩 流浪地球 Rnn_Model.py

```
# -*- coding: utf-8 -*-
```

```
import tensorflow as tf
```

```
class TRNNConfig(object):
```

```
    """RNN 配置参数"""
```

```
    # 模型参数
```

```
    embedding_dim = 64 # 词向量维度
```

```
    seq_length = 20 # 序列长度
```

```
    num_classes = 3 # 类别数
```

```
    vocab_size = 5000 # 词汇表达小
```

```
    num_layers = 2 # 隐藏层层数
```

```
    hidden_dim = 128 # 隐藏层神经元
```

```
    rnn = 'lstm' # lstm 或 gru
```

```
    dropout_keep_prob = 0.8 # dropout 保留比例
```

```
    learning_rate = 1e-3 # 学习率
```

```
    batch_size = 128 # 每批训练大小
```

```
    num_epochs = 10 # 总迭代轮次
```

```
    print_per_batch = 100 # 每多少轮输出一次结果
```

```
    save_per_batch = 10 # 每多少轮存入 tensorboard
```

```
class TextRNN(object):
```

```
    """文本分类，RNN 模型"""
```

```
    def __init__(self, config):
```

```
        self.config = config
```

```
    # 三个待输入的数据
```

```
    self.input_x = tf.placeholder(tf.int32, [None, self.config.seq_length], name='input_x') # 输入的样本
```

```

self.input_y = tf.placeholder(tf.float32, [None, self.config.num_classes], name='input_y') #标签

self.keep_prob = tf.placeholder(tf.float32, name='keep_prob')#避免过拟合


self.rnn()#初始化 rnn 模型


def rnn(self):
    """rnn 模型"""

    def lstm_cell(): # lstm 核
        return tf.contrib.rnn.BasicLSTMCell(self.config.hidden_dim, state_is_tuple=True)

    def gru_cell(): # gru 核
        return tf.contrib.rnn.GRUCell(self.config.hidden_dim)

    def dropout(): # 为每一个 rnn 核后面加一个 dropout 层
        if (self.config.rnn == 'lstm'):
            cell = lstm_cell()
        else:
            cell = gru_cell()
        return tf.contrib.rnn.DropoutWrapper(cell, output_keep_prob=self.keep_prob)

    # 词向量映射
    with tf.device('/cpu:0'):
        embedding = tf.get_variable('embedding', [self.config.vocab_size, self.config.embedding_dim])#每一个字用一个随机初始化的
        embedding_dim 维向量表示

        embedding_inputs = tf.nn.embedding_lookup(embedding, self.input_x)

    with tf.name_scope("rnn"):
        # 多层 rnn 网络
        cells = [dropout() for _ in range(self.config.num_layers)] # 定义 cell
        rnn_cell = tf.contrib.rnn.MultiRNNCell(cells, state_is_tuple=True) # 将两层的 lstm 组装起来

        _outputs, _ = tf.nn.dynamic_rnn(cell=rnn_cell, inputs=embedding_inputs,
                                         dtype=tf.float32) # _outputs 表示最后一层的输出 【?, 600,128】; "_": 表示每一层的最后一个 step 的输出, 也
        就是 2 个 【?, 128】, 几层就有几个 【?, 128】

        last = _outputs[:, -1, :] # 取最后一个时序输出作为结果

    with tf.name_scope("score"):
        # 全连接层, 后面接 dropout 以及 relu 激活

```

```
fc = tf.layers.dense(last, self.config.hidden_dim, name='fc1')

fc = tf.contrib.layers.dropout(fc, self.keep_prob)

fc = tf.nn.relu(fc)

# 分类器

self.logits = tf.layers.dense(fc, self.config.num_classes, name='fc2')

self.y_pred_cls = tf.argmax(tf.nn.softmax(self.logits), 1) # 预测类别


with tf.name_scope("optimize"):

    # 损失函数，交叉熵

    cross_entropy = tf.nn.softmax_cross_entropy_with_logits(logits=self.logits, labels=self.input_y)

    self.loss = tf.reduce_mean(cross_entropy)

    # 优化器

    self.optim = tf.train.AdamOptimizer(learning_rate=self.config.learning_rate).minimize(self.loss)#更新整个网络


with tf.name_scope("accuracy"):

    # 准确率

    correct_pred = tf.equal(tf.argmax(self.input_y, 1), self.y_pred_cls)

    self.acc = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

16124278 王浩 流浪地球 comments_loader.py

```
# coding: utf-8
```

```
import sys

from collections import Counter

from imp import reload

import numpy as np

import tensorflow.contrib.keras as kr


if sys.version_info[0] > 2:

    is_py3 = True

else:

    reload(sys)

    sys.setdefaultencoding("utf-8")

    is_py3 = False
```

```
def open_file(filename, mode='r'):
    """
    常用文件操作，可在 python2 和 python3 间切换.
    mode: 'r' or 'w' for read or write
    """
    if is_py3:
        return open(filename, mode, encoding='utf-8', errors='ignore')
    else:
        return open(filename, mode)
```

```
def read_file(filename):
    """读取文件数据"""
    contents, labels = [], []
    with open_file(filename) as f:
        for line in f:
            try:
                label, content = line.strip().split('\t') #分割
                if content:
                    contents.append(list(content))
                    labels.append(label)
            except:
                pass
    return contents, labels #返回标签和内容
```

```
def build_vocab(train_dir, vocab_dir, vocab_size=5000):
    """根据训练集构建词汇表，存储"""
    data_train, _ = read_file(train_dir)

    all_data = []
    for content in data_train:
        all_data.extend(content)

    counter = Counter(all_data)
    count_pairs = counter.most_common(vocab_size - 1)
    words, _ = list(zip(*count_pairs))
```

```
# 添加一个 <PAD> 来将所有文本 pad 为同一长度

words = ['<PAD>'] + list(words)

open_file(vocab_dir, mode='w').write('\n'.join(words) + '\n')


def read_vocab(vocab_dir):

    """读取词汇表"""

    words = open_file(vocab_dir).read().strip().split('\n')

    word_to_id = dict(zip(words, range(len(words)))) #为每一个词建立一个 id by 位置

    return words, word_to_id #返回


def read_category():

    """读取分类目录，固定"""

    categories = ["好评", "中评", "差评"]

    cat_to_id = dict(zip(categories, range(len(categories)))) #建立一个类别和 id 的字典

    return categories, cat_to_id #返回


def to_words(content, words):

    """将 id 表示的内容转换为文字"""

    return ''.join(words[x] for x in content)


#数据预处理

# max_length=600

def process_file(filename, word_to_id, cat_to_id, max_length=100):

    """将文件转换为 id 表示"""

    contents, labels = read_file(filename) #文件读取


    data_id, label_id = [], []

    for i in range(len(contents)): #将影评和标签全部 id 化

        data_id.append([word_to_id[x] for x in contents[i] if x in word_to_id]) #把影评中的每个字按词汇目录转换为 id

        label_id.append(cat_to_id[labels[i]])#把标签按分类目录转换为 id


    # 使用 keras 提供的 pad_sequences 来将文本 pad 为固定长度

    x_pad = kr.preprocessing.sequence.pad_sequences(data_id, max_length) #取每句影评后 max_length 个字，取不到的地方则置为 0

    y_pad = kr.utils.to_categorical(label_id, num_classes=len(cat_to_id)) # 将标签转换为 one-hot 表示
```

```
return x_pad, y_pad
```

```
def batch_iter(x, y, batch_size=64):
```

```
    """生成批次数据"""
```

```
    data_len = len(x)
```

```
    num_batch = int((data_len - 1) / batch_size) + 1
```

```
    indices = np.random.permutation(np.arange(data_len))
```

```
    x_shuffle = x[indices]
```

```
    y_shuffle = y[indices]
```

```
    for i in range(num_batch):
```

```
        start_id = i * batch_size
```

```
        end_id = min((i + 1) * batch_size, data_len)
```

```
        yield x_shuffle[start_id:end_id], y_shuffle[start_id:end_id]
```