豆瓣电影网站

——数据抓取与统计分析

2016.6.16

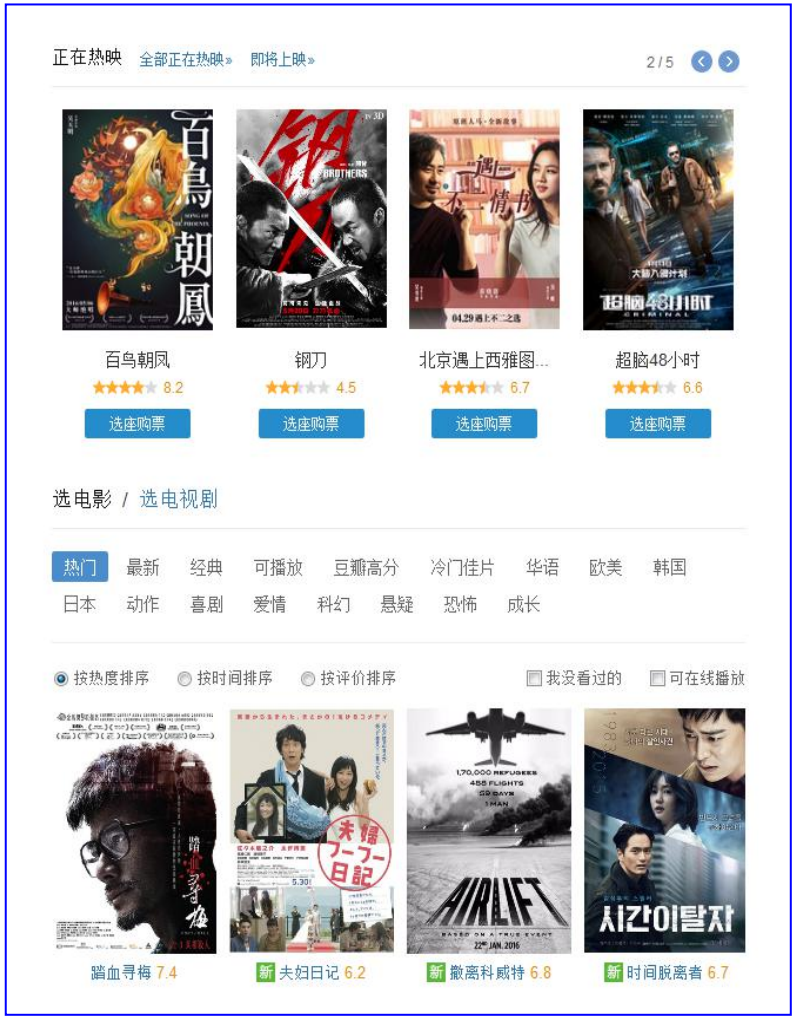
豆瓣电影网分析与抓取

豆瓣电影网分析

主页

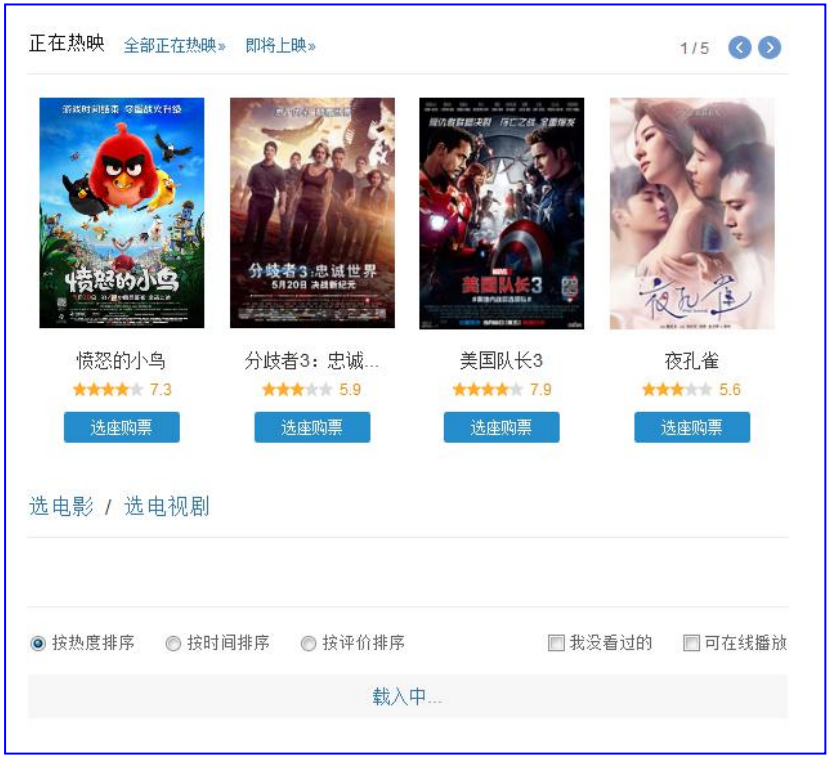
豆瓣电影首页如右图所示，包含“正在热映”、“选电影”两个需要关注的地方。爬虫可以从这里去获取最新的收录的电影的链接地址。

考虑到很多网站使用了 JavaScript 去实现网站数据的获取并呈现，那么豆瓣首页中可能也用上了这样的方式。但爬虫只是实现 HTML 文档的抓取，并不包含对 JavaScript 代码的执行，那么爬虫获取到的文档中就会缺少这样的部分。



使用浏览器工具可以禁用 Javascript 脚本，方便查看爬虫抓取到的文档结构。以 FireFox 为例，在豆瓣页面按下 F12，弹出工具箱窗口，在设置中将“禁用 Javascript”勾选，并保持工具箱窗口存在，然后刷新页面，就能够看到没有执行 Javascript 脚本的页面了。

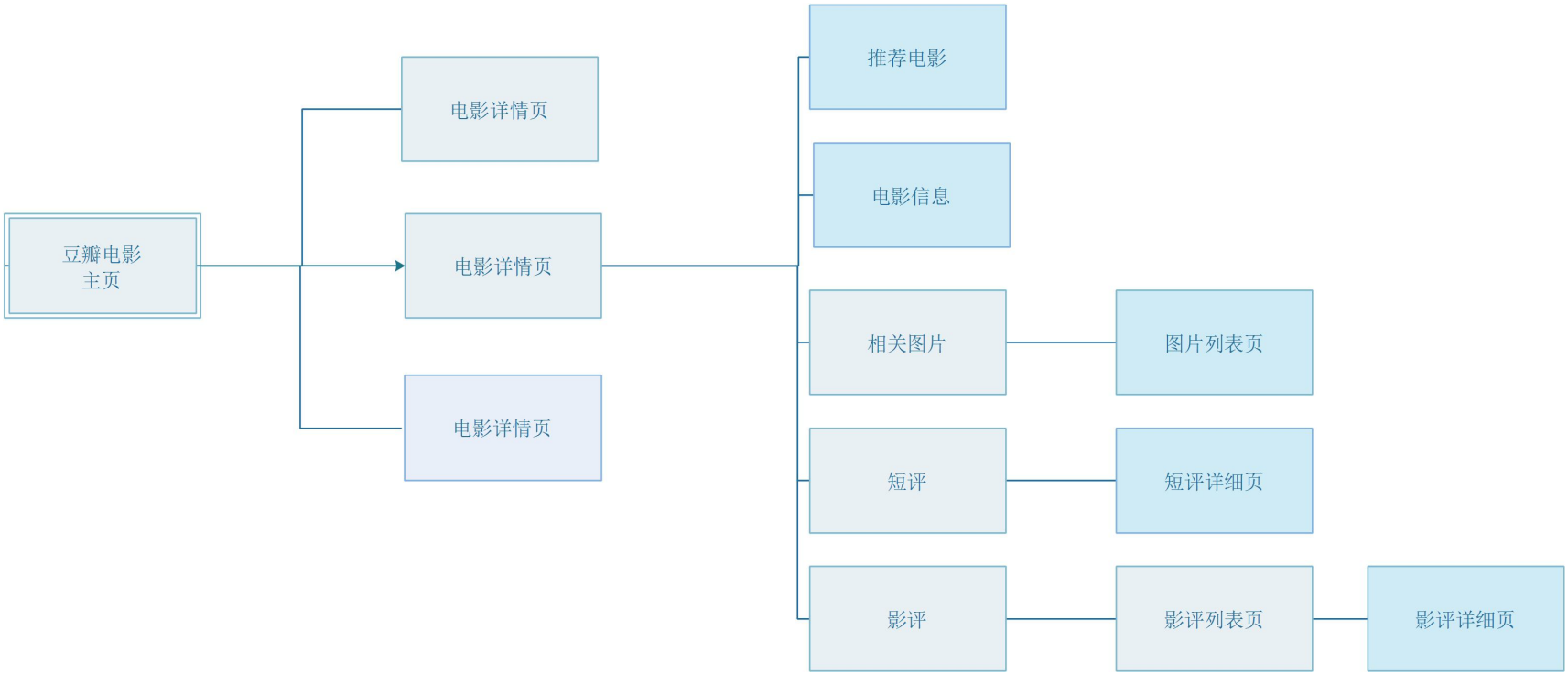
通过 Firefox 的工具箱禁用 JavaScript 后，豆瓣首页变成了如下图所示的样子。首页中“选电影”的地方总是显示“载入中...”，而并不像没有禁用 JavaScript 之前的那样出现了很多候选影片。但是，“正在热映”的地方仍然显示正常，说明它并不是由代码生成的，可以被爬虫抓取与识别。



对于代码生成的部分可以去分析 AJAX 请求的数据，然后在爬虫中模拟请求并自行提取需要的信息。但这里可以简单的忽略掉，只通过“正在热映”部分获取电影的链接地址。一个实际的电影链接页面如下图所示，由于是未登录抓取，所以并不打算去抓取用户数据，而重点关注电影信息和相关的评论、图片等等。

电影详细页

通过对豆瓣电影的结构分析，有如下图所示的层级关系。



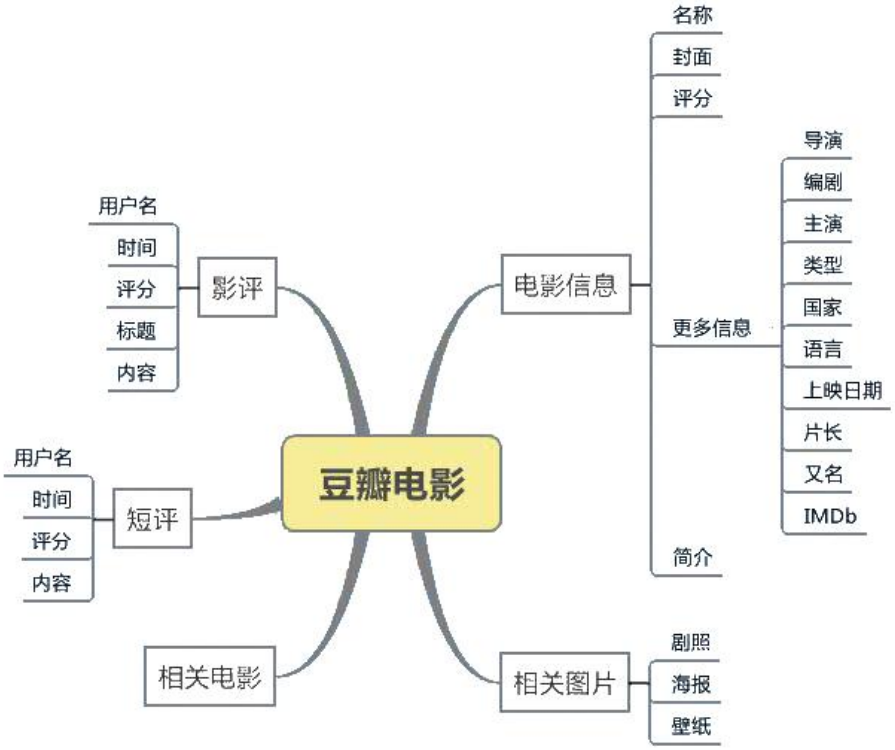
从豆瓣电影主页能够获取到多个电影详情页的链接，每个电影详情页可直接获取到电影信息和推荐电影链接，而相关图片需进入“所有图片”页面去获取图片的实际地址，短评也需要进入短评详细页才能获取到所有短评，而影评信息更深，需先进入列表页，根据列表页的地址进入到实际影评页面获取影评信息。

爬虫模块与数据表

模块划分

由豆瓣电影详情页面包含的信息，可以分析出爬虫需要抓取处理的几个地方，分别是电影信息部分（片名、封面、评分、导演等），相关图片部分，相关电影部分，短评部分，影评部分。

那么就可以确定爬虫的主要模块的功能了，其如下图所示。爬虫还必须的数据库操作模块、日志状态记录模块等没有在图中写出。



数据表

按照模块的划分，创建四个数据库表，分别为 movieInfo、fileInfo、comments、reviews，用来存储电影信息，电影图片信息，短评，影评。

电影信息表 movieInfo

列名	数据类型	允许 Null
Id	nvarchar(16)	true
Cover（封面文件名）	nvarchar(32)	true
Name（电影名）	nvarchar(MAX)	true
Diector（导演）	nvarchar(MAX)	true
Screenwriter（编剧）	nvarchar(MAX)	true
Act（演员）	nvarchar(MAX)	true
Type（类型）	nvarchar(MAX)	true
Country（国家）	nvarchar(MAX)	true
Language（语言）	nvarchar(MAX)	true
Date（上映日期）	date	true
Length（时长）	int	true
Aka（别名）	nvarchar(MAX)	true
Episode（集数）	int	true
IMDb（IMDb 编号）	nvarchar(32)	true
Average（平均评分）	float	true
Summary（简介）	nvarchar(MAX)	true
Ismovie（是否为电影）	bit	true

电影图片信息数据表 fileInfo

列名	数据类型	允许 Null
Id	Nvarchar(16)	true
Name（文件名）	Nvarchar(64)	true
Path（文件存储路径）	Nvarchar(128)	true

短评数据表 Comments

列名	数据类型	允许 Null
Id	Nvarchar(16)	true
Username（用户名）	Nvarchar(MAx)	true
Score（评分）	int	true
Date（日期）	date	true
Comment（内容）	Nvarchar(Max)	true

影评数据表 Reviews

列名	数据类型	允许 Null
Id	Nvarchar（16）	true
Username（用户名）	Nvarchar（MAX）	true
Score（评分）	Int	true
Date（日期）	datetime	true
Review（内容）	Nvarchar（MAX）	true
Title（标题）	Nvarchar（MAX）	true

数据表中均允许置 Null 的原因是由于豆瓣对电影信息的录入有些很全面，有些则很匮乏。有些连演员、简介这些信息都没有，有些电影上映日期有多个国家地区的时间，影片时长也是在不同地区有所不同，即便是单一上映日期，也有多种格式，有标准的，有美式的，还有各种简写的；

短评或是影评中也有用户评论的时候忘记评分的，也有没有写影评标题的。这些字段如果不允许置空将会丢失较多的可用信息，所以为了尽可能地保留信息与增强容错性，就允许了所有字段置空。

对数据库的友好性

数据库中的 Date 类型数据要求输入的字符串格式，但豆瓣上日期格式众多，即使做了前期转换处理，面对更多更复杂的情况很有可能转换出错，一但出错变将其置为一个特殊的时间，如“1227-02-27”,同时在数据表中增加一个字段用来存储原来的日期的字符，方便后期人工查错修复。

特别值得注意的是，在短评、影评、简介等内容中很有可能包含单引号，而写入数据库时使用的也是单引号表示输入字符串，这样就有问题出现，最好在前期的时候将字符串中的单引号全部替换成两个单引号。

从 HTML 提取数据

Jsoup 的使用

网站的数据一般都是结构性的，在特定的标签下的特定属性或是值就是需要抓取的对象。而 java 中有很多包能够实现对 HTML 标签的解析，例如 HTMLParser、Jsoup 等等。这里使用 Jsoup 来提取指定标签的属性值，因为 Jsoup 能够使用类似 CSS 的选择器语法，使用起来很方便。

例如，使用 Jsoup.connect(String url)方法来载入一个 HTML 文档，同时还能够对这词连接设定各种参数以达到各种要求。使用 Element.select(String selector)方法和 Elements.select(String selector)方法来筛选指定的元素。使用 Tagname 来通过标签查找元素，比如：a；使用 ns|tag 来通过标签在命名空间查找元素，比如：可以用 fb|name 语法来查找 <fb:name> 元素；使用#id 来通过 ID 查找元素，比如：#logo；使用.class 来通过 class 名称查找元素，比如：.masthead；使用[attribute]来利用属性查找元素，比如：[href]；使用[^attr]来利用属性名前缀来查找元素，比如：可以用[^data-] 来查找带有 HTML5 Dataset 属性的元素；使用[attr=value]来利用属性值来查找元素，比如：[width=500]；使用[attr^=value], [attr$=value], [attr*=value]来利用匹配属性值开头、结尾或包含属性值来查找元素，比如：[href*=/path/]；使用[attr~=regex]来利用属性值匹配正则表达式来查找元素，比如：img[src~=(?i)\.(png|jpe?g)]；使用*来这个符号将匹配所有元素等等。

数据提取

例如主页中“正在热映”的几部影片的链接通过分析可以发现它们都是类名为“ui-slide-content”的<ul>标签下的类为“poster”的<li>标签下的<a>的“href”属性值。所以，同样的通过分析目标的这些标签与属性，很容易就能够提取出想要的信息。

```
<ul class="ui-slide-content">
    <li class="ui-slide-item s">
        <ul class="">
            <li class="poster">
             <a href="https://movie.douban.com/subject/6873736/?from=showing"></a>
            </li>
            <li>
            ......
            </li>
            ......
        </ul>
    </li>
    <li>
    ......
    </li>
    ......
</ul>
```

从豆瓣电影中也能分析出很多有规律的信息，比如豆瓣电影中电影详情页面的 URL 中所带的数字是该电影在豆瓣中的 ID，对于 URL：“https://movie.douban.com/subject/2131940/?from=showing”，“2131940”就是其 ID，而通过 ID 就能够很快速的得出其图片、影评、短评的 URL，它们分别是“https://movie.douban.com/subject/”+ID+“/all_photos”、“https://movie.douban.com/subject/”+ID+“/reviews”、“https://movie.douban.com/subject/”+ID+“/comments”

豆瓣爬虫的实现

深度优先与去重

豆瓣电影提供的相关电影部分，就好比是当前电影通向其他电影的通道，多个这样的通路层级的组合起来就是一颗树，每次获取到的相关电影的链接都被放在待解析电影链接队列的尾部，每次处理链接又是从头部取出，这就相当与是在深度遍历这颗豆瓣电影组成的树。

相关电影中可能在不同的电影下有重复项目，为防止重复遍历，使用 HashMap 集合将豆瓣电影 ID 作为 key 存入，如果查询到 ID 已存在就不再进入节点中，否则就存入当前 ID 到 HashMap 中并进入该节点。

状态存储与日志记录

爬虫在间隔一定的时间段之后保存当前操作队列中的所有数据到本地文件中，防止因各种意外而出现的程序崩溃导致信息丢失的情况。对于爬虫的每一步操作都生成日志记录，方便在后期优化软件或查找 Bug 的时候提供足够的信息。为防止存储过程占用信息刷新的时间，将存储过程移到额外临时开辟的线程中去执行。

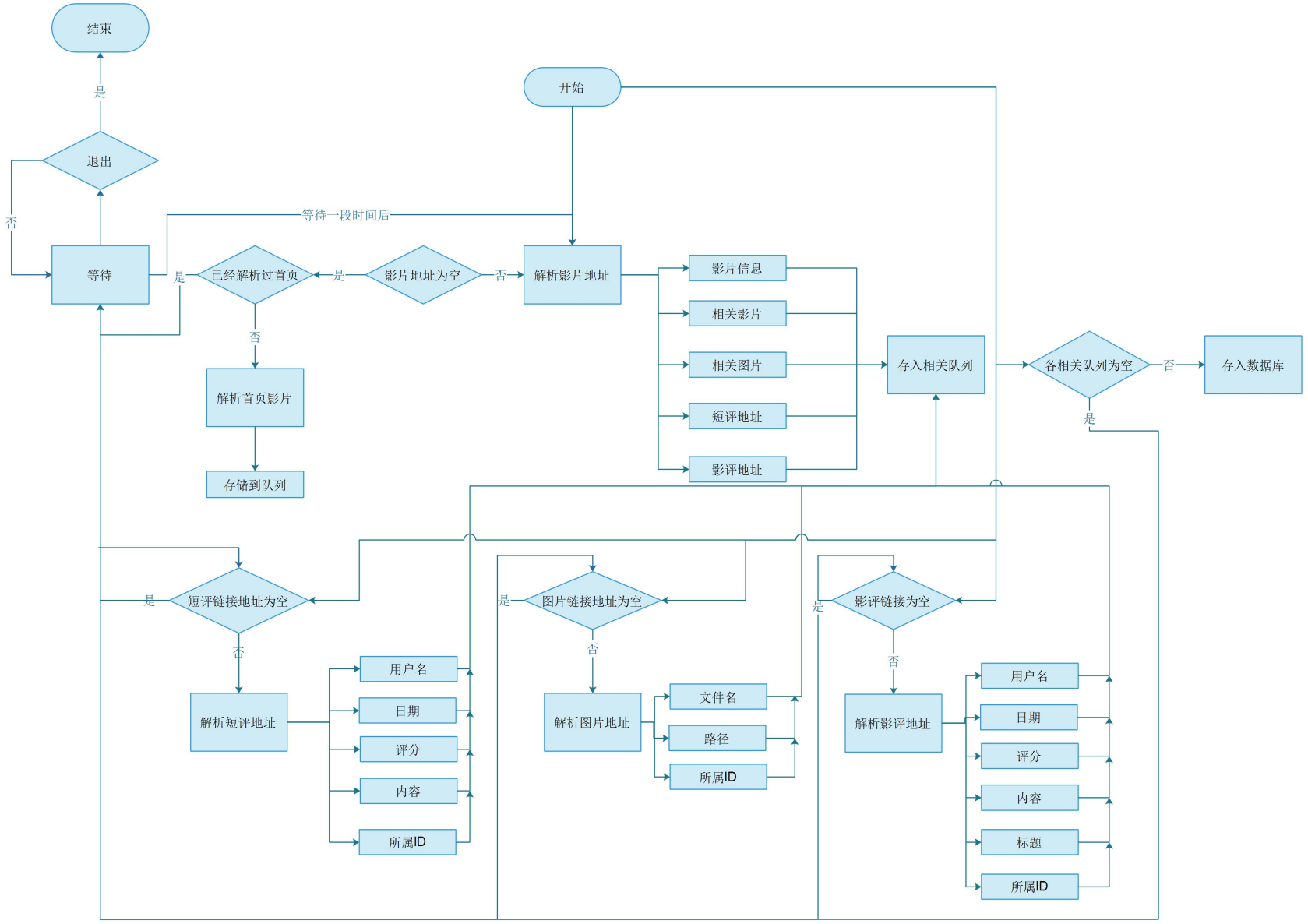
Bean 模式信息存储

按照数据库存储表编写对应的存储 Bean 类，其包涵所有表中对应字段，提供 get、set 方法的同时覆盖原有 toString 方法，将其改写为输出对应的 SQL 插入语句，用在数据库操作阶段出错时提供信息备份机制。

多线程队列

为了后期使用多线程的方便，须在队列上实现多线程的冲突共享。这里使用 Java 的泛型机制，通过类中封装一个 LikendList 类型的集合，向外部提供经 synchronized 关键字修饰的添加与删除方法，目的是保持读写操作在线程间的单一性，这让多个处理线程能够针对同一数据源实现并发操作，加快处理速度。

爬虫流程


结束
是
退出
否
开始
等待一段时间后
等待
是
已经解析过首页
是
影片地址为空
否
解析影片地址
影片信息
相关影片
相关图片
短评地址
影评地址
否
解析首页影片
存储到队列
存入相关队列
各相关队列为空
否
存入数据库
是
是
短评链接地址为空
否
解析短评地址
用户名
日期
评分
内容
所属ID
是
图片链接地址为空
否
解析图片地址
文件名
路径
所属ID
是
影评链接为空
否
解析影评地址
用户名
日期
评分
内容
标题
所属ID

加快爬虫的爬取速度

多线程

在 java 中通过继承 Thread 类或是使用 Runnable 接口并实现 run 方法就能够使用多线程运行该类的实例。爬虫在获取指定 URL 的时候有一定的等待时间，解析的时候又有一定的等待时间，特别是短评的数量一般比较庞大，基本都是成千上万条，如果依次处理，这些时间的等待是极大的浪费，如果一个线程获取解析，另外的线程作其他操作，他们的所需的信息与解析出的结果都分别放入一个能加同步锁的队列中，这样就可以多线程并发，加快处理速度。

User-Agent

如果单单是使用多线程，那么往往快速的访问一段时间之后便会出现无法访问的错误，原因是服务器出于安全的考虑，会将访问频繁的 IP 的请求拒绝掉，多线程应该配合多 User-Agent 来实现模拟多用户的访问，防止服务器拒绝请求。

反爬虫设置

设置 User-Agent

User-Agent 是网站识别用户的重要手段之一，通过更换 User-Agent 能够防止因过于频繁而被服务器拒绝访问的危险，因为随机的更换 User-Agent 就类似于将自己伪装成网关，有多个用户通过自己访问网站。

常见的 User-Agent 有 Windows IE 浏览器家族的，Mozilla 的 Firefox 浏览器的，Google 的 Chrome 浏览器等等。通过设置不同的版本、操作系统等能够衍生出更多的 User-Agent。

在 Jsoup 中使用 `Document dc=Jsoup.connect(link).userAgent(userAgant[s])` 来设置 User-Agent。

设置 Cookie

访问豆瓣有时会有键为_utma, _utmb, _utmc, _utmz 的 cookies，它们都是 Google Analytics 生成的，与豆瓣关系不大。如果是通过百度而不是直接输入豆瓣电影地址访问的话，还会在 cookies 中出现像_pk_ref, _pk_cvar, _pk_id, _pk_ses 的键名，这与豆瓣无关，它们是百度使用 Piwik 而添加的。如果使用豆瓣帐号登录，那么还会有多个 cookies 生成，这里不作分析。

通过直接在地址栏输入“movie.douban.com”进行访问，那么会接收到两个 cookies。

ll	"108309"
bid	11 位数字、字母、特殊字符混合的字符串

在测试的三次中，ll 的值一直是"108309"，而 bid 是变化的。那么只需要注意 did 的设置即可。服务器可能还会对每个 ID 的访问作单位时间内的次数限制，所以还需要每访问一定次数之后重新获取新的 ID。

虽然可以不对其他 Cookies 做处理，但是保险起见，对于凡是接收到的 Cookies，除了 did 重点关注以外，其余的都原样发送给服务器，当需要更换 did 的时候，其余 Cookies 一同更换。

在 Jsoup 中使用 `Document dc=Jsoup.connect(link).cookies(cookies[s])`来设置

设置 Referer

Referer 是 HttpHead 中用来标记访问来源的字段，部分网站中会对来源地址进行审查，如果不符合规定将拒绝访问。豆瓣中有个别情况下会出现使用地址直接访问短评页或是影评页会出现无法访问的情况，极有可能是对 Referer 字段进行了审查。所以有必要在访问短评或是影评的页面的时候最好设置 Referer 字段为其对应的电影详情页面地址。

在 Jsoup 中使用 `Document dc=Jsoup.connect(link).referrer(purl)`来设置

SQL 中进行数据处理

重复记录清除

由于运行爬虫的机器内存硬盘都比较小，抓取速度过快导致队列中信息过多，内存占用太大最终程序崩溃，而下载的图片没有及时打包上传到网盘，导致本地硬盘写满，数据库需要扩展空间的时候就会出错，导致写入数据库失败，接着对应的 SQL 错误处理就会将出错未写入的信息保存到指定的 SQL 文件，但硬盘已经写满，所以这些信息并不会写入文件，就永久丢失了这些数据。

虽然可以从已有的记录文件中恢复到某一时刻的队列状态，但是已经写入数据库的数据没办法一起回退，所以有些数据是重复的，就需要对数据库中重复的数据进行清理。

例如对于 movieInfo 中的数据，先使用 SQL 语句：

```
select COUNT([Id]) from [DouBan].[dbo].[movieinfo] where [id] in
(select [id] from [DouBan].[dbo].[movieinfo] group by [Id] having count([Id]) >= 2)
```

查找出所有重复的记录，将其保存在临时表中，然后在原表中删除他们，最后后使用 Distinct 关键字修饰[id]字段，将结果写回即可。

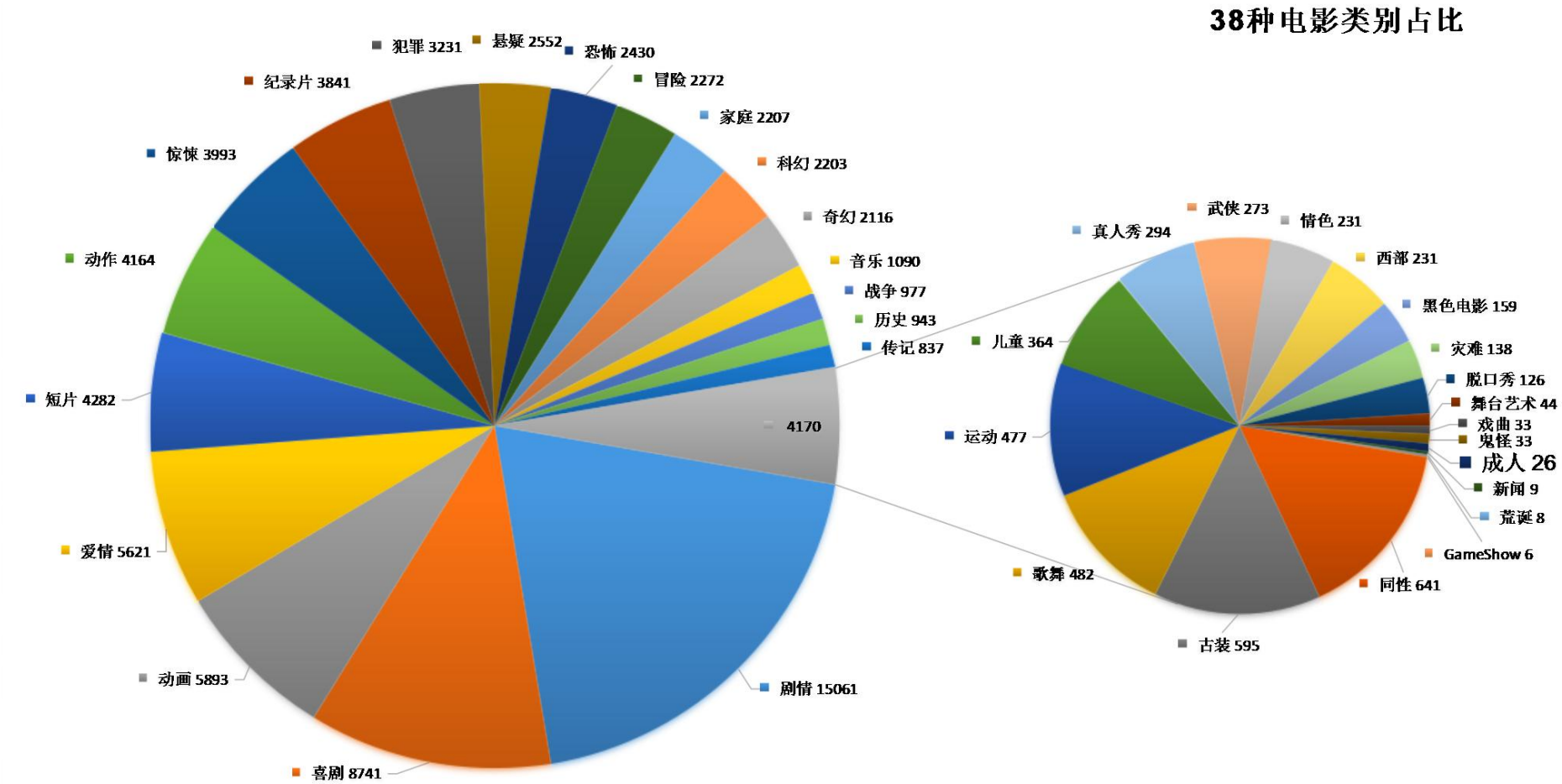
对于影评或是短评中的重复记录使用，使用下面的 SQL 语句（以 review 为例）：

```
select * from [DouBan].[dbo].[reviews] where [id]+[Username]+[Review] in
(select [id]+[Username]+[Review] from [DouBan].[dbo].[reviews] group by [id],[Username],[Review] having COUNT(*)>1)
order by [UserName]
```

查找出所有重复的记录，将其保存在临时表中，然后在原表中删除他们，最后后使用 Distinct 关键字修饰[Review]字段或是[Comment]字段，将结果写回。

数据的统计分析

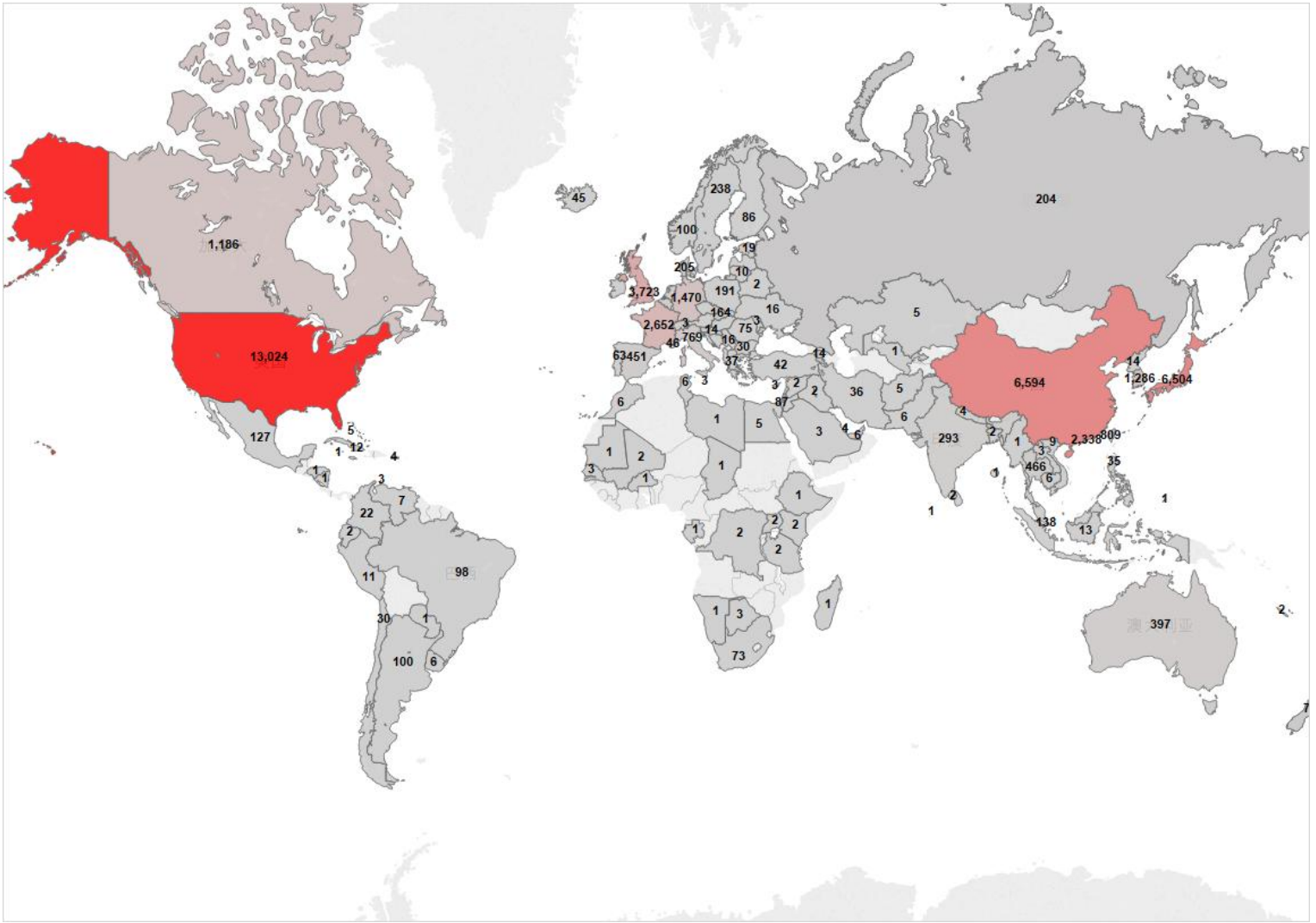
截止目前爬虫共爬取到 39844 条电影记录。按照影片的类型分类（具有多个类型标签的电影对每个标签都增加计数一），共计有 38 种电影类型，其中以剧情、喜剧、动画三种类型占比最高，分别达到了 20%，11%，8%。如果加上爱情类电影，基本占据了电影类型中数量的一半。



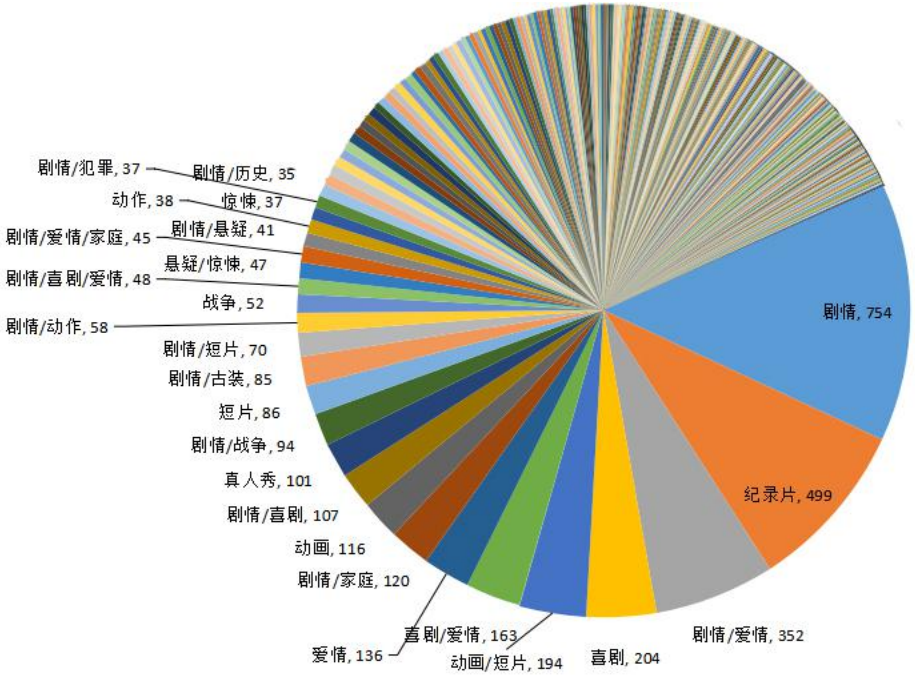
电影收录数量国别分布

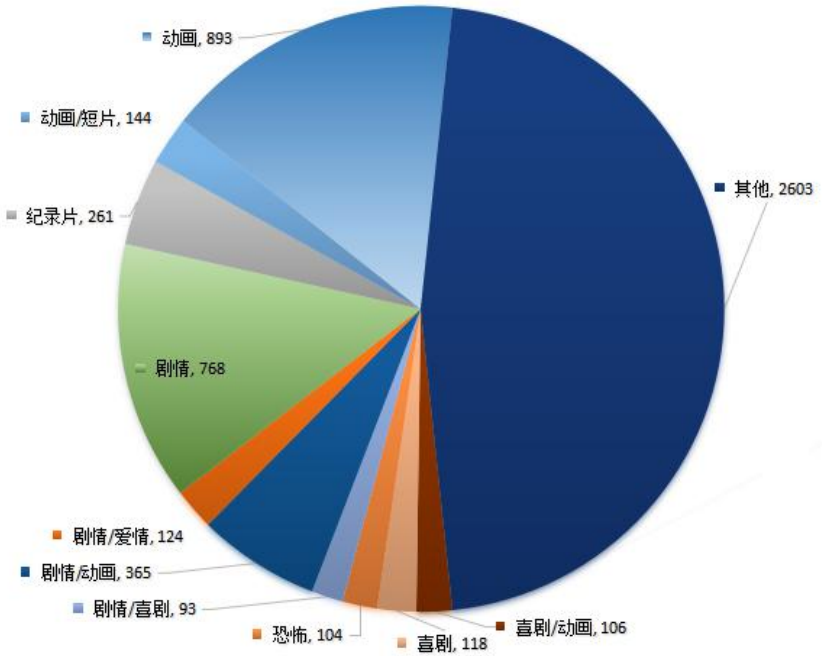
从现有数据分析，按照国别统计电影成片数量，美国以13024部影片数量高居榜首，比重为28.23%；中国以6594部影片数量排名第二，约摸只是美国的一半，占比约为14.29%；日本以6504部电影数量紧随其后，占比也约为14.10%；之后英国以3723部电影数量排名第四，占比约为8.07%。

整体上，北美、欧洲、东亚是主要的影片出品地区，非洲、中亚、拉美地区相比之下数量匮乏。



中国与日本电影类型占比





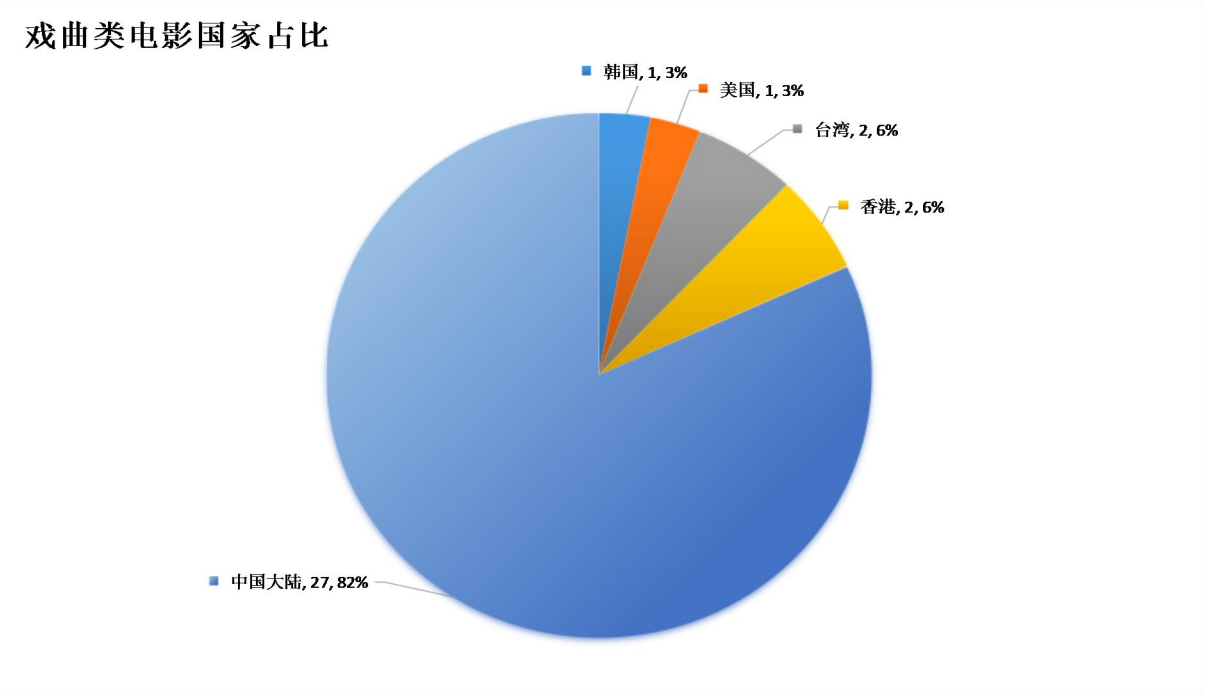
在中国的电影类型中，以剧情、纪录片、喜剧、爱情类型占比最多。在日本电影中动画、剧情、剧情/动画、纪录片类型占比较大。虽然同是东亚地区国家，但是日本与中国在电影类型占比上还是有不同，日本作为动漫产业大国，其动画电影类型的数量占比相对其他国家还是很高的。

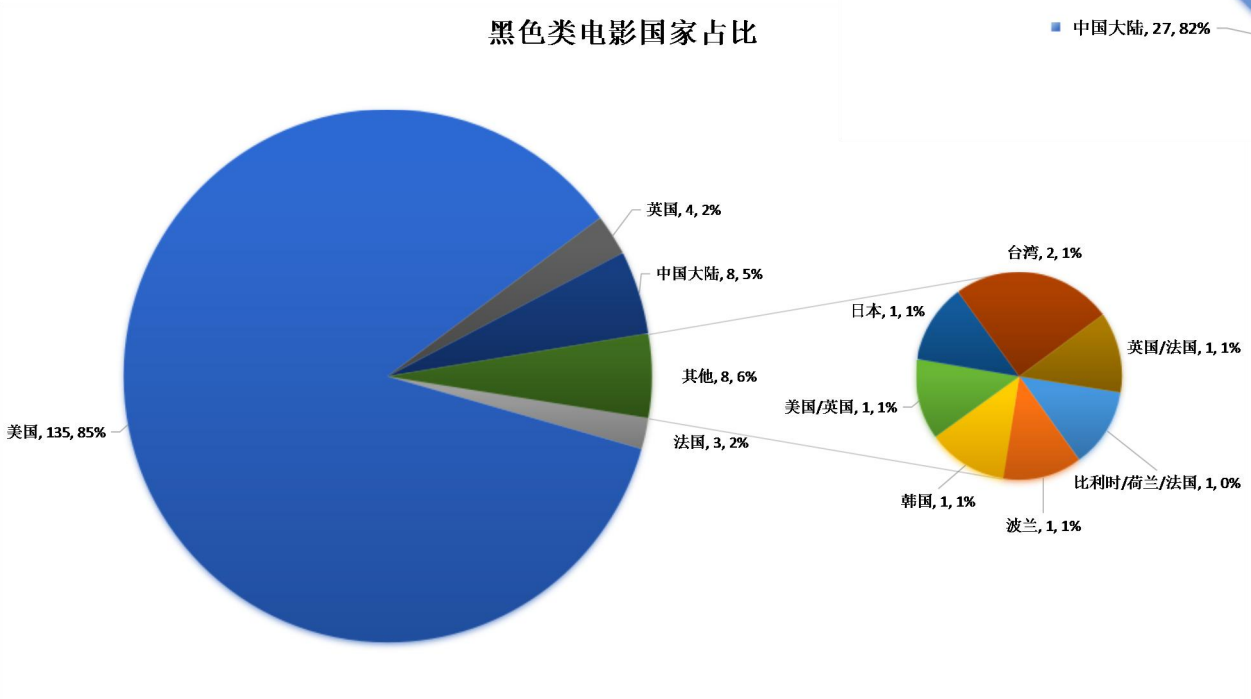
美国的电影和英国电影的类型占比具有相似性，这也许他们同是西方国家的原因。美国的电影类型前三名分别是纪录片（770）、喜剧（740）、剧情（620），之后四名都是喜剧、剧情、爱情混合类型，而恐怖/惊悚（223）与纯恐怖（203）类型电影分列第八第九。

同时，四国多种类型混合的影片也非常多，充分体现了电影类型的多种多样与人们选择的空间，但各国人在影片类型上各自的倾向，唯一不变的是剧情永远是电影的一大要素。

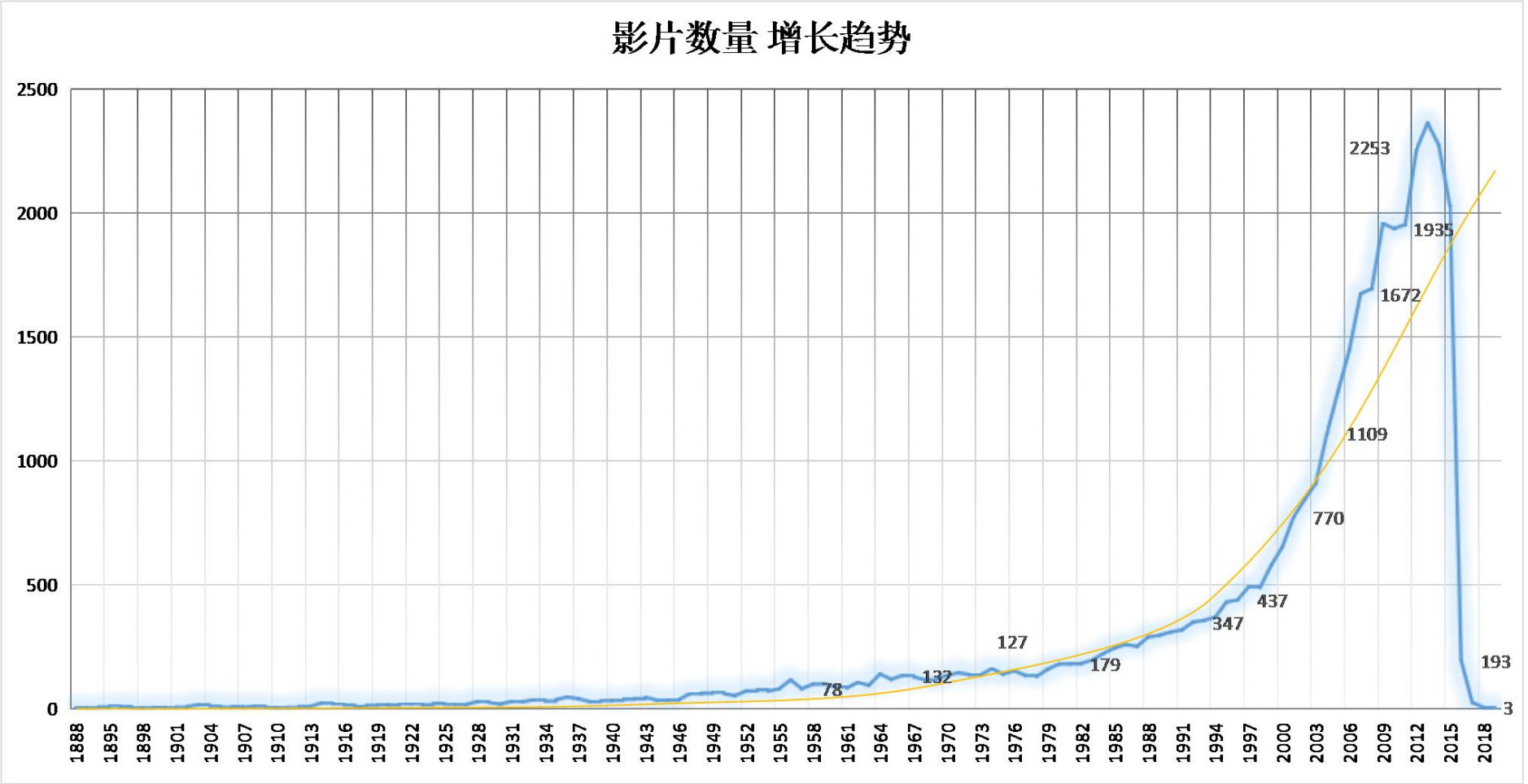
在所有电影分类标签中，有一些极具国家或区域特征的电影类别，例如戏曲类和黑色电影类。

在戏曲类电影中，大中华地区占比达到了 94%，受中国文化的影响，韩国也有该类电影，而美国出现戏曲类电影也许是受华人移民带去的影响。可想而知，欧洲著名的歌剧形式在欧洲占比也是很大的，在东亚地区可能占比就没有那么高了。

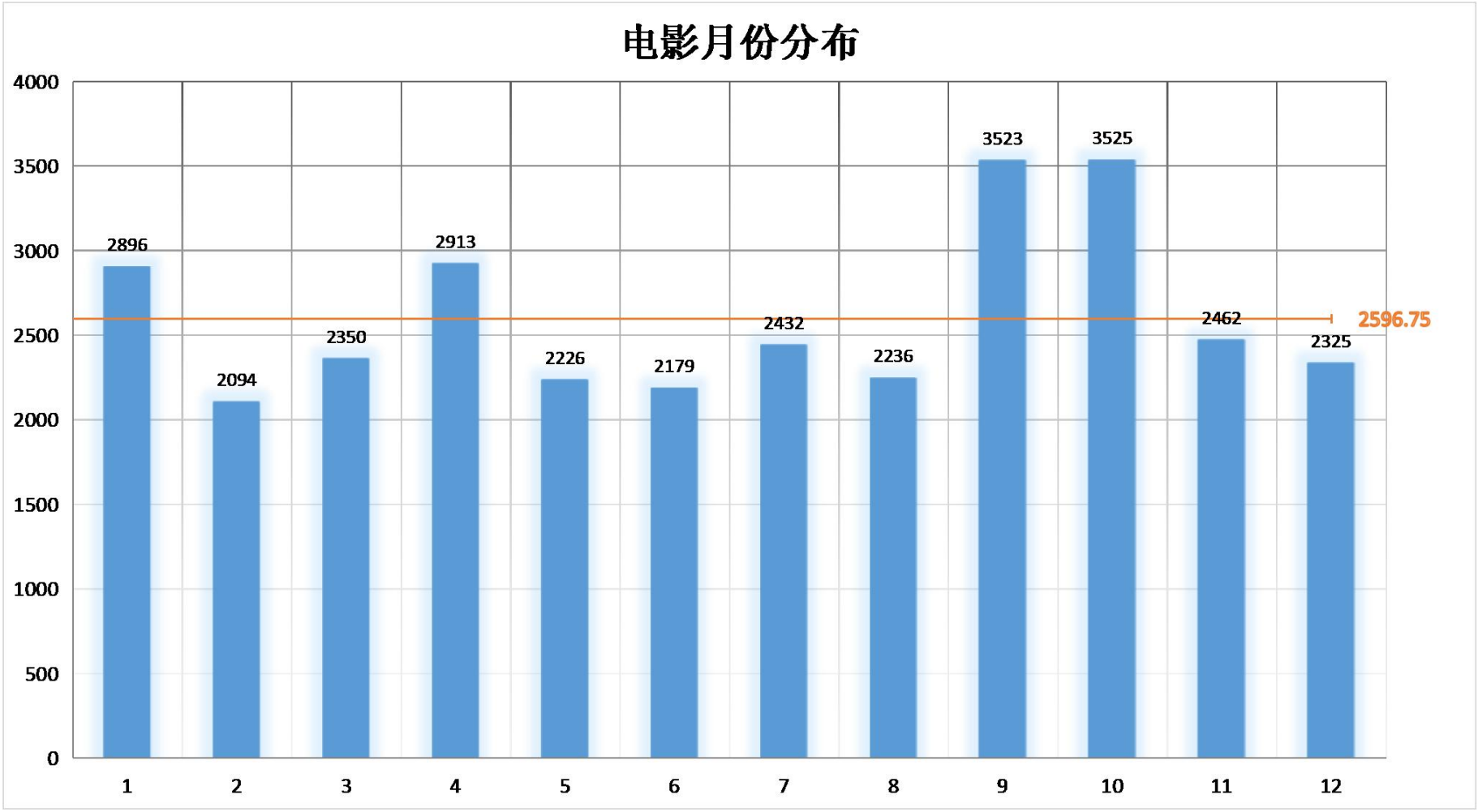




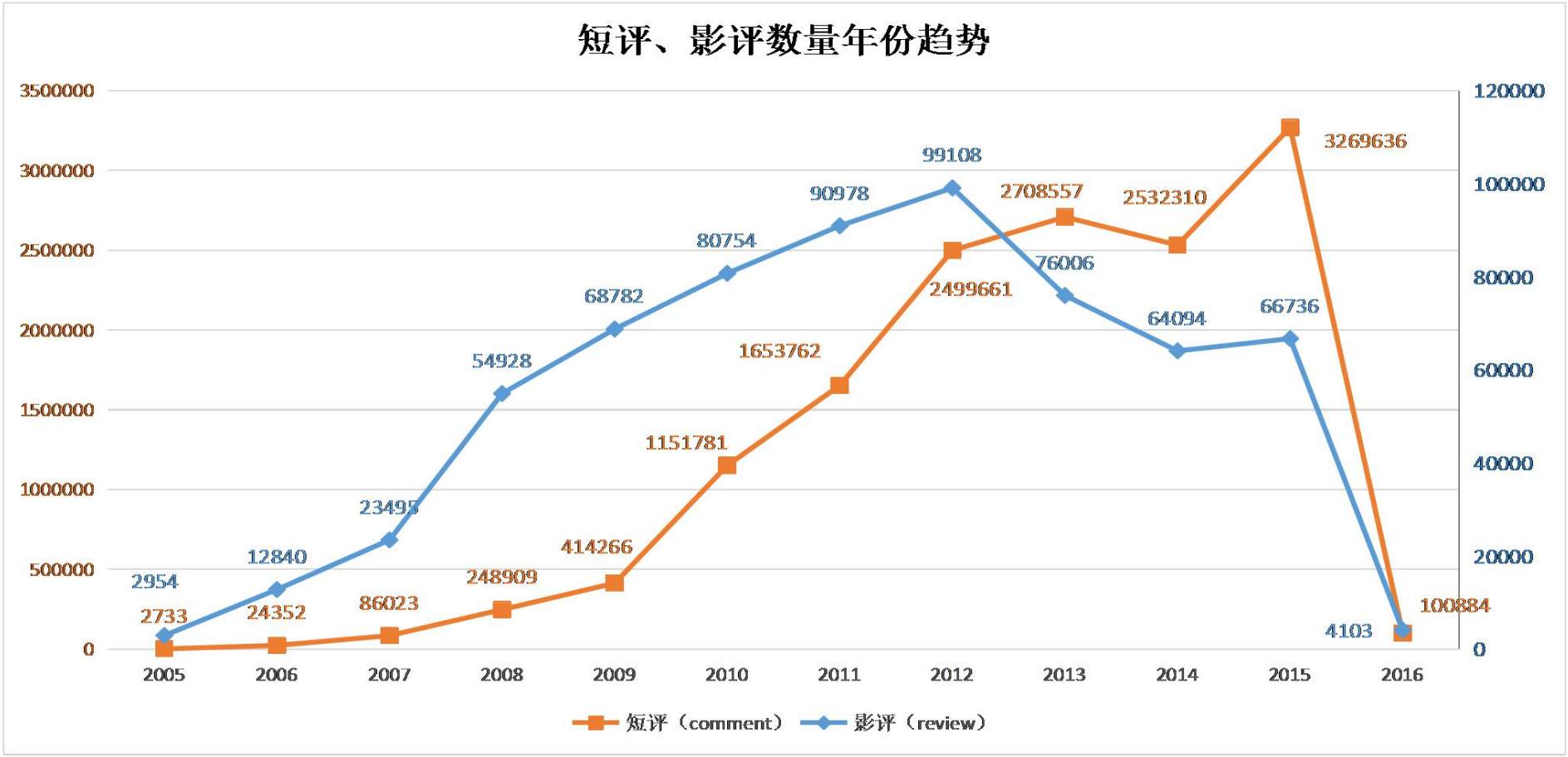
与戏曲类电影相比，黑色类电影更加具有西方特性。美国与欧洲国家黑色类型电影数量占比超过了 89%，中日韩三国有少量这种类型的电影。



豆瓣收录的电影年限范围是 1888-2019，出现超出当前年份的记录的原因是豆瓣将一些还未上映但预计能上上映的电影提前生成了网页页面，而 2016 年的数据只爬取到截止二月份的数据，所以只有相比 2015 年少很多。从图中可以看出，在上世纪 90 年代之前，影片数量一直呈缓慢上升的态势，在 90 年代开始逐步增长，到了 21 世纪初开始爆发式增长，基本符合指数增长曲线趋势。

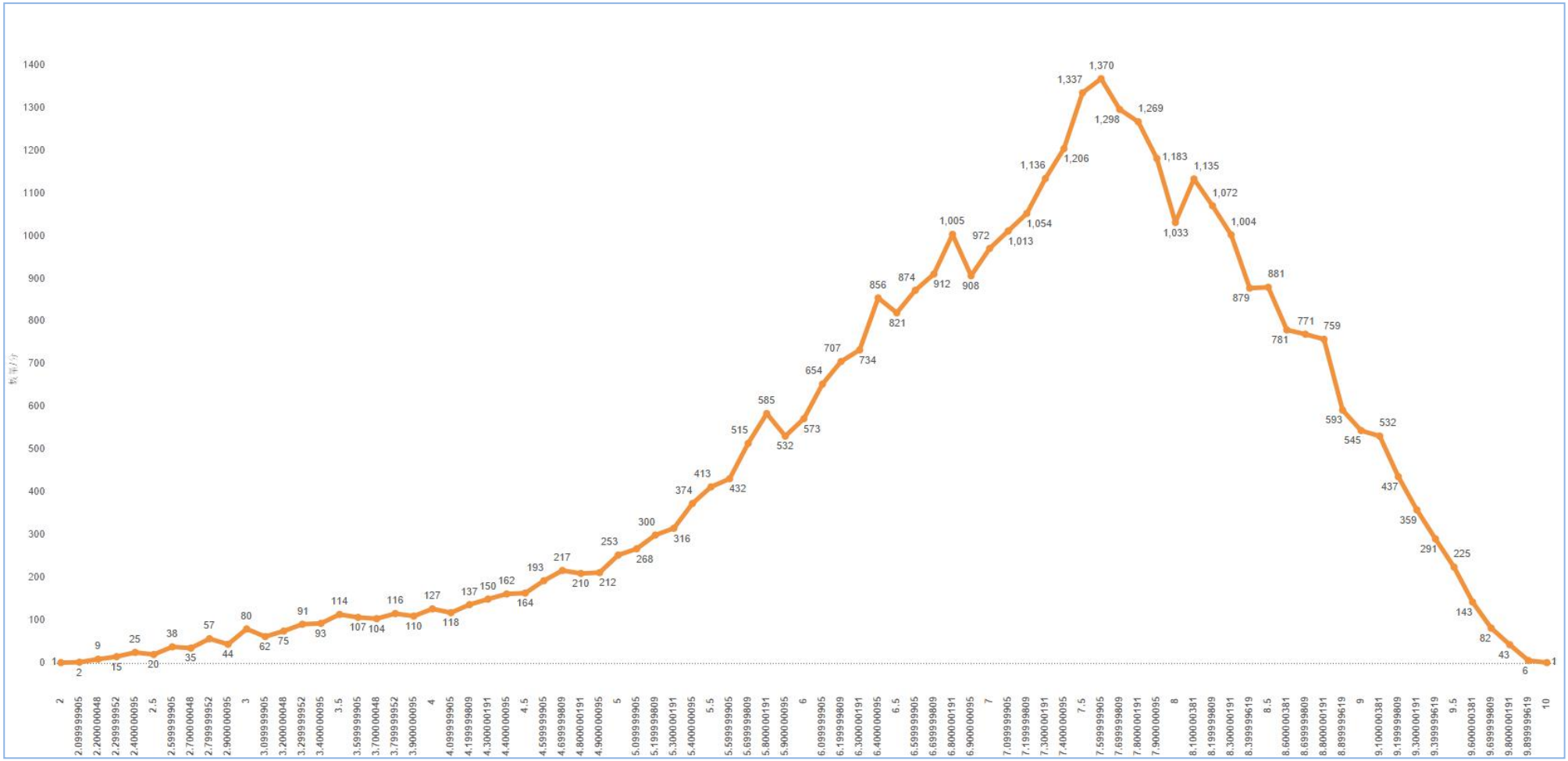


每个行业都有自己的“金九银十”，电影行业也不例外。上述图表中九月与十月的电影上映数量是十二个月中最多的，双双占比超过 11%，一月（9.29%）与四月（9.34%）也有不错的表现。或许是九十月份的各种商业的“金九银十”活动促使人们消费的同时也带动了当时电影的消费（比如逛街买换季的衣服，看到有好看的电影就顺便看了）。

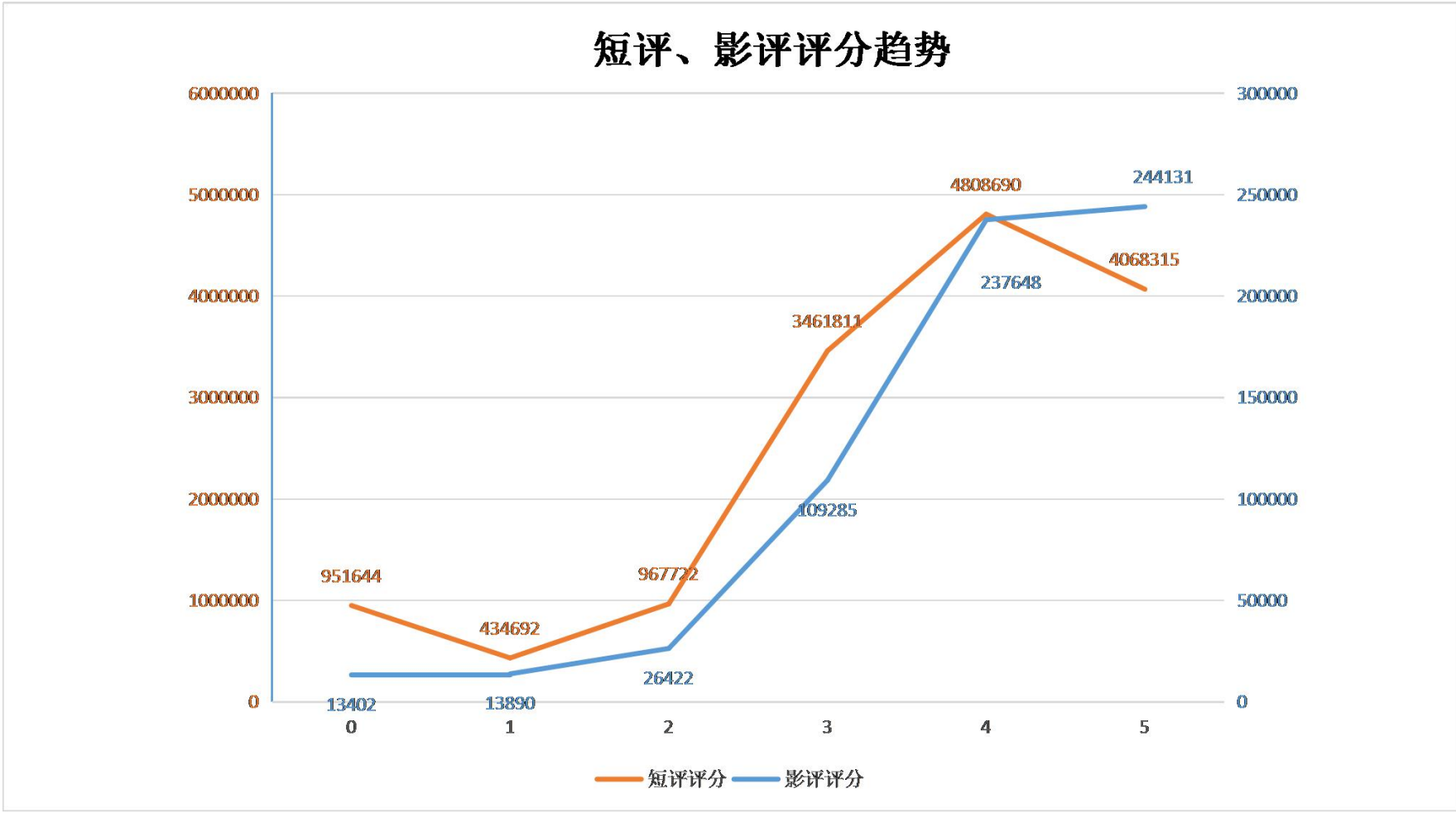


豆瓣是 2005 年上线的，所以最早的影评或短评都不可能早于那个时间。从 2005 年到 2015 年（2016 年只有头两个月的数据）十年时间里，豆瓣用户对影评和短评功能一直都有使用，除开头一年里两者不相伯仲，到后来成倍的使用次数差距，豆瓣用户更加倾向于发表简短的短评。使用短评功能的次数也是逐年递增。相比之下，影评功能使用次数较短评功能的少，但仍有一批资深的电影人在使用它。其在 2012 年达到使用高峰，之后便呈现出颓势，使用次数开始下降，直到 2015 年才有微弱的上升。

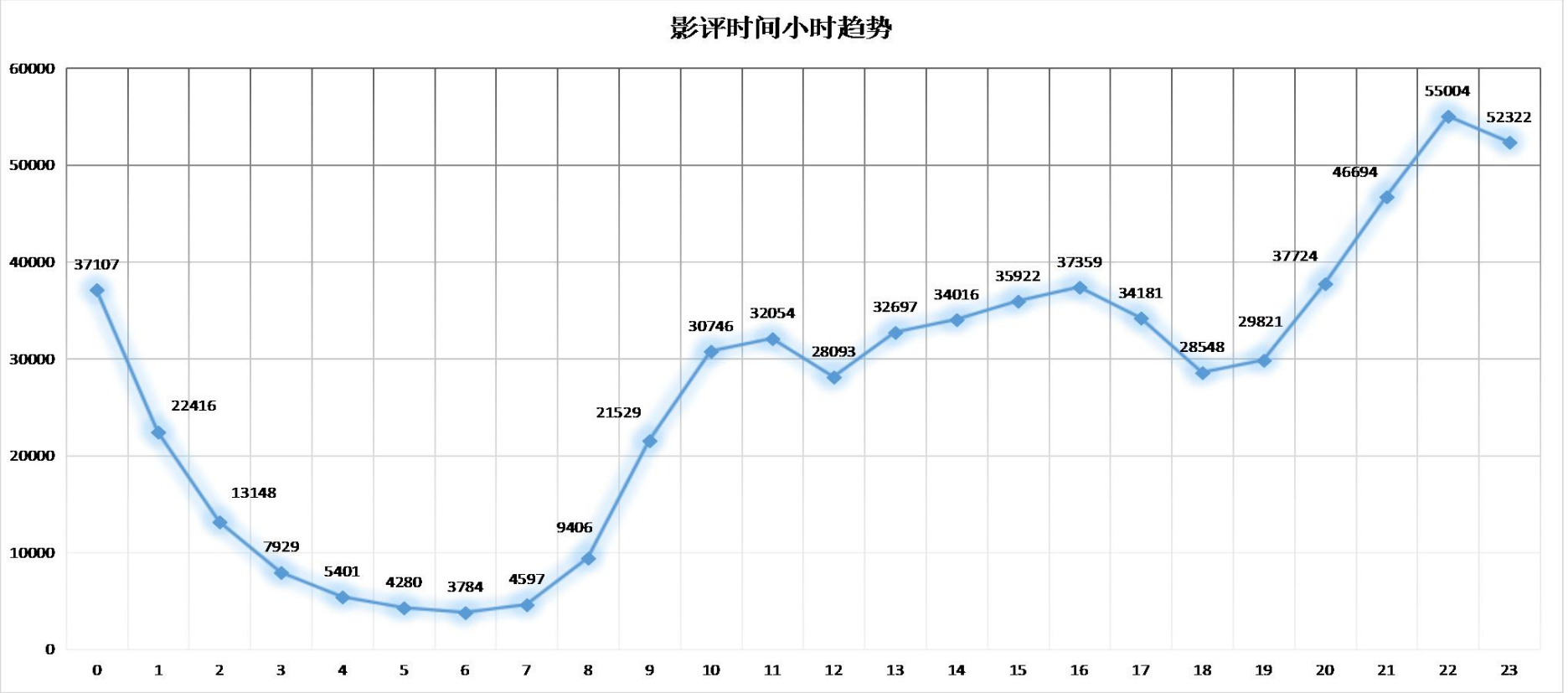
电影平均评分分布



豆瓣对每一部收录的电影都有一个平均评分，它是所有用户评分的平均值。上图展示了评分的分布，7.5 分处取得最高出现次数，基本上豆瓣电影的评分是倾向于 5.5 以上的，超过 5.5 分的评分有 87.6%。总的来说，豆瓣用户的评分还是偏仁慈的。



电影的平均得分应该是按照短评评分和影评评分计算而来的，所以两者呈现出一样的态势，4 到 5 分的高分居多，1 到 2 分的低分只有高分的十分之一左右，3 分的评分在两者之间，占高分的约一半。0 分的情况无法判定，或许是由于用户不小心忘记评分，也可能是用户觉得就是应该打 0 分，这还有待通过用户评价的分类进行分析。



通过上图，我们可以看出豆瓣用户使用影评功能的时间分布，在下午时间段和晚上九十点是两个高峰期，其中以晚上十点又为最多。说明豆瓣用户一般在下午闲时及晚上饭后观影并写下影评。虽然夜里时间段数量很少，但还是有不少豆瓣用户喜欢在晚上观影写影评的。

在八点到十点这一时间段里，数量差不多以一万为差值递增，使用人数骤然增多，与大多数人在早上七点到九点起床的作息时间相吻合。

如果人们大多是观影后间隔一小段时间就来写评论，那么可以想像这么一些场景：五点到六点是下班放学的时间，学生党在食堂吃完饭就看电影来了；而上班族还在回家的拥堵的路上，而且很多是下午晚餐时间有可能在下午六点到七点这段时间，所以就有了后来 17 点到 18 点下降而之后又上升的走势。

豆瓣用户分析

影评、短评数量前十用户

用户名	影评数	用户名	短评数
Jack	1304	Summer	8576
涅瓦纳	1087	echo	8219
做事不能设假设	735	M	6081
云飞扬	702	L	5606
小众电影	573	s	5584
竹子	551	.	5317
林下之风	530	Iris	4891
Betty Louis	510	zoe	4815
IORI	502	已注销	4654
Riobluemoon	481	。	4505

由表中数据可看出，用户的短评数与影评数和整体上短评数远远大于影评数的状态是相符的。由于短评包含的信息较少，这里就取影评数前两名作分析，数据仅限于他们的影评。

用户“涅瓦纳”

用户“涅瓦纳”评论过的电影类型包括 38 种类型的 30 种（爱情、传记、动画、动作、短片、儿童、犯罪、歌舞、古装、黑色电影、荒诞、纪录片、家庭、惊悚、剧情、科幻、恐怖、历史、冒险、奇幻、情色、同性、武侠、西部、喜剧、悬疑、音乐、运动、灾难、战争）。

当列出该用户所观看电影的信息，并按照时间倒排的时候，我们发现了一个问题：他评论的时间间隔太过于接近了，下图中是其最早写的几条影评记录，它们间的时间相差只有几秒的时间，而每条影评的字数却不少，要想在短时间内打出这么多字，关键是持续打这么多条，基本上是不可能的，那么只有怀疑

这些字符是粘贴上去的。果然，将这些影评与电影的简介做对比，发现他们是一模一样的！

勇敢的心 Braveheart	Braveheart	2011/07/24 12:27.34	威廉•华莱士童年时，其父、苏格兰民
蝙蝠侠：黑暗骑士 The Dar	The Dark Knight	2011/07/24 12:27.33	从亲眼目睹父母被人杀死的阴影中走出
剪刀手爱德华 Edward Sci	Edward Scissorh	2011/07/24 12:27.30	爱德华（约翰尼•戴普 饰）是一个机
钢琴家 The Pianist	The Pianist	2011/07/24 12:27.29	史标曼（艾德里安•布洛迪 Adrien B
指环王2：双塔奇兵 The Lo	The Lord of the	2011/07/24 12:27.28	第二部在延续第一部风格的同时，故事
闻香识女人 Scent of a W	Scent of a Woma	2011/07/24 12:27.24	查理（克里斯•奥唐纳 Chris O'Donn
泰坦尼克号 Titanic	Titanic	2011/07/24 12:27.23	1912年4月10日，号称 “世界工业史
放牛班的春天 Les choris	Les choristes	2011/07/24 12:27.20	1949年的法国乡村，音乐家克莱门特
海上钢琴师 La leggenda	La leggenda del	2011/07/24 12:27.16	本片讲述了一个钢琴天才传奇的一生
天堂电影院 Nuovo Cinema	Nuovo cinema Pa	2011/07/24 12:27.13	意大利南部小镇，古灵精怪的小男孩
辛德勒的名单 Schindler&i	Schindler's	2011/07/24 12:27.12	1939年，波兰在纳粹德国的统治下，
这个杀手不太冷 Léon	Leon	2011/07/24 12:27.10	昂（让•雷诺饰）是名孤独的职业杀手
阿甘正传 Forrest Gump	Forrest Gump	2011/07/24 12:27.07	阿甘（汤姆·汉克斯 饰）于二战结束
机器人总动员 WALL·E	WALL·E	2011/07/24 12:27.04	公元2700年，人类文明高度发展，却
肖申克的救赎 The Shawsha	The Shawshank R	2011/07/24 12:27.00	1946年，年青的银行家安迪（蒂姆•罗
杀死比尔 Kill Bill: Vol	Kill Bill: Vol.	2011/07/24 12:26.58	这是一部令人眼花缭乱，绝对能带给
西雅图未眠夜 Sleepless	Sleepless in Se	2011/07/24 12:26.57	自从妻子去世后，萨姆（汤姆·汉克斯
真实的谎言 True Lies	True Lies	2011/07/24 12:26.54	哈里（阿诺•施瓦辛格 Arnold Schwa
变形金刚 Transformers	Transformers	2011/07/24 12:25.45	“霸天虎”的先遣部队旋风和毒蝎袭

由此，得出两个结论：一是豆瓣的数据库可能出过什么问题，用户记录下的这些时间并不是原来的时间，而是恢复数据的时候由系统自己生成的，二是这个用户使用了软件自动发布影评之类的。

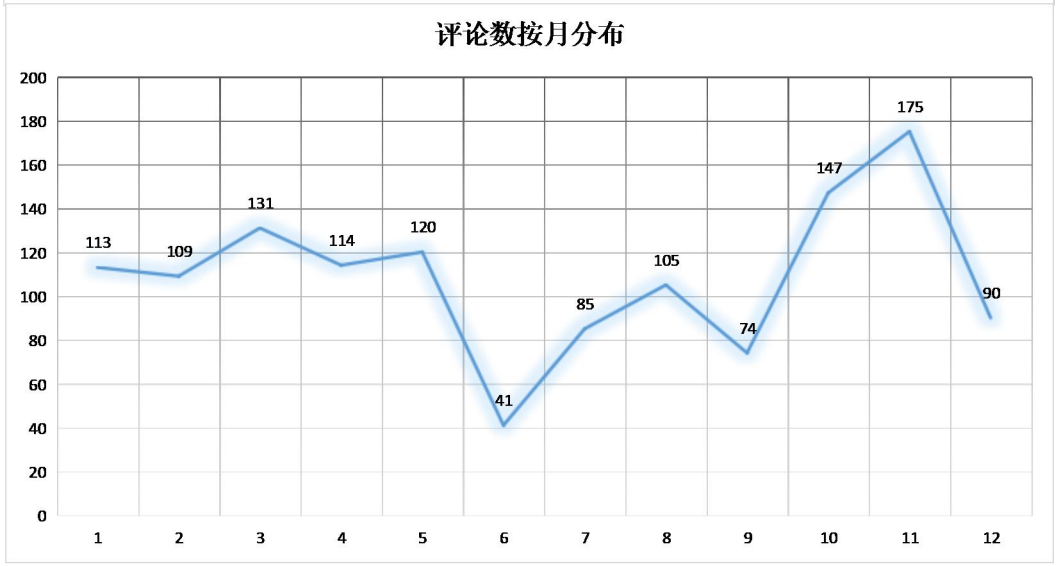
用户“Jack”

相比排名第二的用户，这个名为“Jack”的用户则是真正的电影爱好者，无论是他发表的短评还是影评，大部分表达的是自己的感情与观点，有写电影甚至隔了个一两天又发一条它的评论。

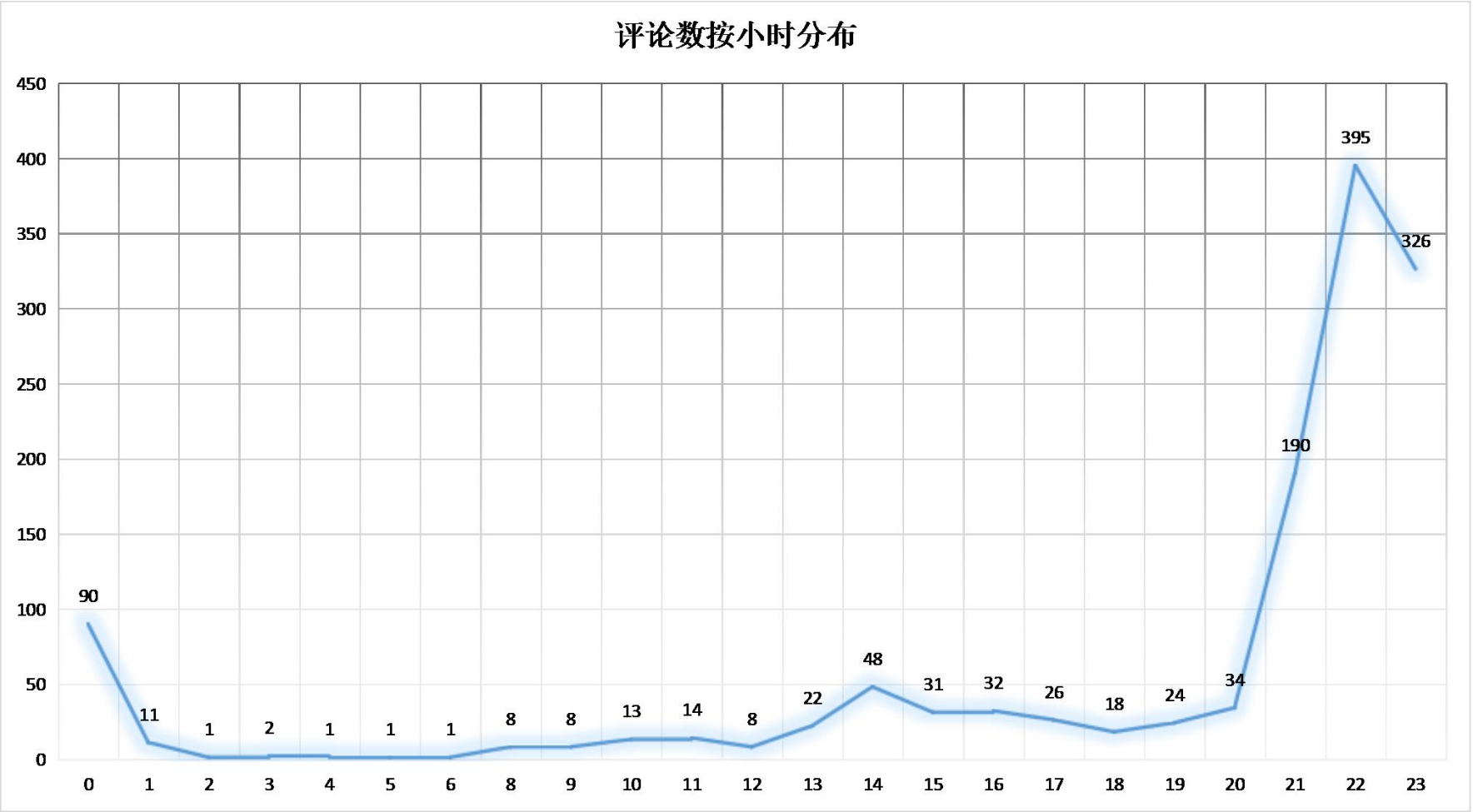
他评论过的电影类型包括 38 种类型的 29 种。一共评论过 993 部影视作品，其中 20.94%为电视连续剧,剩下的都是电影。他评论过的影视作品中有 57 部

是剧情类，56 部是喜剧类，爱情类 17 部，纪录片 13 部，其余为剧情、爱情、喜剧、动作等其他类型混合。这些影视作品的出版发行方的国别与整体上的国别占有率相当，美国（425 部）、中国（113）、日本（58）分列前三，英国(39)、韩国（37）紧随其后。

他在 2006 年首次使用豆瓣电影的评论功能，到 09 年以前都是浅尝辄止，从这一年开始是频繁的使用，到了 12 年就用得顺手了，使用次数是前年的三倍多，之后也是稳步上升，2016 年只有一次的原因是爬虫抓取的 2016 年的数据很少。



从评论数的月分布上可以看出，在十月十一月两个月达到高峰，十二月到五月比较平稳，六月是低谷。结合前面电影的上映日期的月分布图可以看出，他基本在新电影上映一个月左右会去观影，之后在豆瓣上写下评论，而六月是他最不喜欢的观影月份，可能是这一时期上映的电影类型多是他不喜欢的。



从上表看来，用户“Jack”并不是一个喜欢在半夜和上午写评论的人，即使是在下午，他也是不时地写一写，更多的是在晚上八点以后，也就是九点到晚上十一点，一般不会超过一点。这说明他应该是在上班或是上学之类的有事做的状态，而且是比较忙或是他本人是一个专心做事的人，也许只有在周末或是节假日才会偶尔在白天去写评论，更多的是在白天忙完之后。

豆瓣分词器

通过影评构建分词字典

用符号切分句子

这里仅仅考虑中文的分词与词典的构建，非中文的字符会被过滤掉。

影评文章通常比短评用词多，也更加符合语言规范。对每篇影评按照句号、逗号等标点符号分割为多个文章句子片段，这些片段往往都具有完整的意思，同时，这样的切分操作也能够减轻字与字之间错误的连接（被标点符号隔开的两字之间，在此处一定不具有成词的用意）。

用特殊汉字切分句子

在中文中有些字是有非常高的出现频率的，比如“的”、“是”、“我”等等，但这些字往往只能和较少的其他字组成词语，比如“的确”，而其他的往往是错误的切分组合，而且由于其出现的频率高，其后或其前无论出现何字，其频率也是相对较高的，会对后期的处理产生影响。于是在句子切分阶段，就可以将这些字符同样视为和标点符号一样的作用，也是切分的标志。

N-Gram

N-Gram 是 NLP 中一种常用的方法，它将输入的字符切出小于等于其长度的所有字符的子集以用于后期分词处理。这里将经过上面两步处理后的句子片段进行 N-Gram 处理，并统计每种子串在整个影评数据中出现的次数。

N-Gram 虽然能够切出各种长度的子串，但中文中，一字词语、二字词语、三字词语及四字词语占去了绝大部分，只有少量词语超过了四个字，所以使用 N-Gram 切出长度大于 4 的子串意义不大，限定其切出长度反而还能提高效率。

词语筛选

对于结果中出现频率低于一定值的字串，我们可以将其剔除，因为在一个足够大的资料中该字串出现的次数越小说明其越不可能是一个词语，对其删除不会影响正确的结果。

有一些词语之间往往有较大的几率一起或是间隔较小的距离出现，那么就有可能会出现词 ABC 与词 BCD 的情况，这时候，需要通过 A 与 BC 同时出现的概率及 BC 与 D 同时出现的概率来判断谁的成词概率更高，同时如果 ABC 或 BCD 出现的次数远远小于 BC 的出现次数，那么可视为 BC 是一个词，而 ABC、BCD 不是。同样，还可能存在 ABCD、BCD、CD、BC 这样的父串与子串的情况。这里也需要通过概率或是出现次数去判断那一个更可能是一个词语。

中文中有些字是固定出现在词的末尾或是头部的，比如“子”字往往出现在词的末尾，“第”字往往出现在词头部，虽然他们也有特例的情况，但是其数量都是很少的，可以后期手工添加。那么，我们可以将不符合这些规则的词语删除，以提高字典的准确度。

后期优化

目前生成的字典可能因为其原始语料的局限性，并没有很好的包括一些常用的短语或是特有名词，像成语、地名、机构名等等。为了增加字典的覆盖面，可以手工录入一些固有词语，并将这些词语的性质标记为“固有”，即在任何状态下不受出现概率的扰动，一旦有子串匹配它就确定其为一个正确的切分。

同时还需要录入前期删除的像满足以“第”字结尾这样规则的字的特殊例子，比如“府第”、“门第”。

分词方法

正向最大匹配与逆向最大匹配

正向最大匹配就是指在待分词的句子按顺序经过 N-Gram 切出的所有子串中满足两个条件：在字典中存在、子串最长，那么这个子串便是一个正向最大匹配；对应的，逆向最大匹配就是由句子的末尾开始向句子开头前进，并满足正向最大匹配的条件的子串就是一个逆向最大匹配。

通过同时运用正向最大匹配与逆向最大匹配，对切分出来的相同部分视为正确的切分，对于不同的部分视为不正确切分（即此处出现歧义），那么对不

同的切分部分需要进行进一步处理。

歧义消解

歧义消解是指消除不符合当前语境的切分。例如“长安全城一片欢腾”，正向最大匹配切分结果可能是“长安 全城 一片 欢腾”，逆向最大匹配切分结果可能是“长 安全 城 一片 欢腾”，这里面“长安全城”四字有歧义，需进行处理。这里采用比较歧义字段进行正向最小匹配切分，并将最大概率视为正确的切分然后依次处理后面的字符，如 P(长安)=5006，P(安全)=2365，P(全城)=1654，那么最终结果应为“长安 全城”；仅当它们概率相同或是相近的时候将前后字段连接并视为一个正确的切分。这样虽然不能解决所有歧义的消解，但是能够处理大部分。

未登录词识别

无论如何提高字典的覆盖面，也无法完全包括所有词语，特别是对于人的名字，姓氏虽然只有有限的几个，但与名组合却能有无法数清的个数。让其自动识别这些词语显得很必要。

我们考虑在对文章分词前，先建立一个针对于当前文章的小词典，只统计其 N-Gram 划分出的所有子串的出现次数，然后删除只出现一次的子串，并去掉在原词典中已有的词语，最后按照原词典生成方法生成这个小词典。仅当分词时无法查询到某个词语时，在小词典中进行查找，如果找到就将这个子串视作为登录的词进行切分。

分词器设计

Hash 树

为了加快查询的速度，按照子串的长度分割，每一个字符存放在一层的 Hash 表中，并且每一个字符都有一个指向其后字符的 Hash 表的引用，而首字放入一个额外的 Hash 索引中，这样就将字典分割成了一颗 Hash 树。

在 Hash 树中存放的是一个 Word Struct，它包括如下的成员变量：

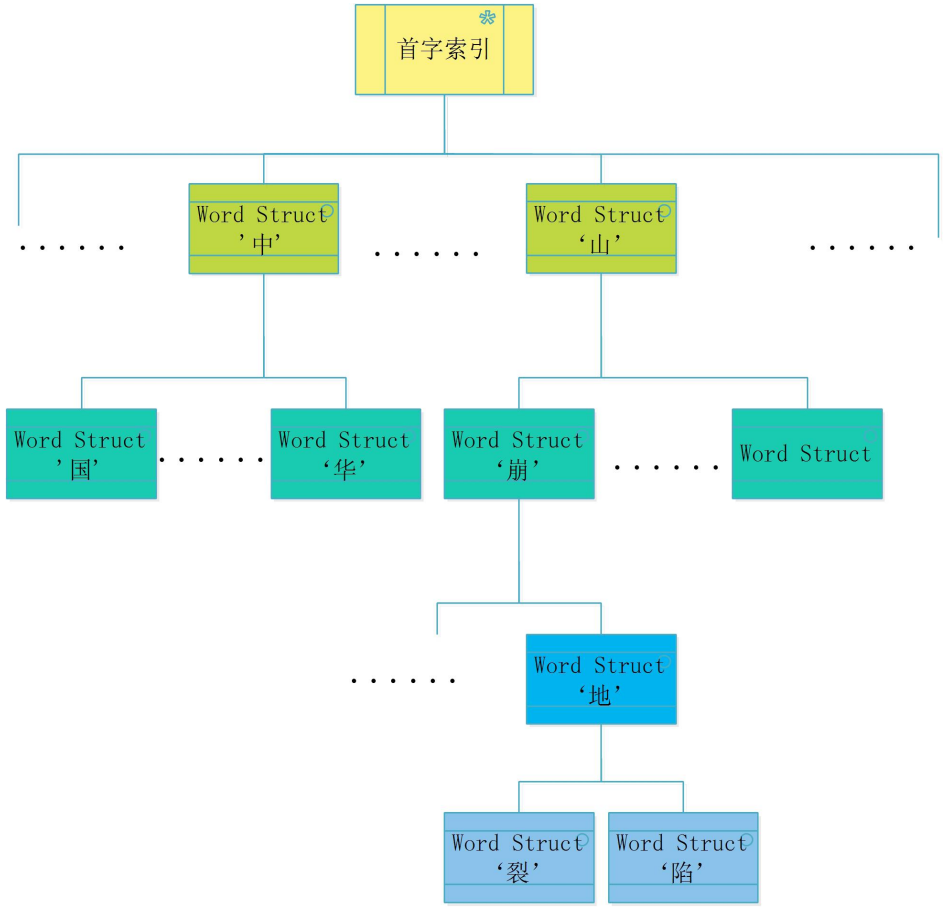
WordStruct 成员变量表

作用	数据类型
Word（存储当前字符）	Character
P（当前字符及其以前字符组成字串出现的次数）	Double
Type（当前字符及其以前字符组成字串的词性）	Enum
Sound（当前字符在与其以前字符组成字串的发音）	String
HashMapPtr（当前字符以后的字符所在的 Hash 表的引用）	LinkedHashMap<Character,WordStruct>

由此就能够生成像右图所示的树状结构，添加、删除、获取或是查找的时候使用递归调用就能够方便的获取到信息。

以右图中“山崩地裂”一词为例，如果“山”字在首字索引中不存在，那么将 new 一个 WordStruct，并将它存入首字索引中，“山”字会被写到这个 WordStruct 的 Word 变量中，由于它不是字串的结尾，P 默认为零，Type 和 Sound 依照参数或是默认设置，并 new 一个 LinkedHashMap<Character,WordStruct>给 HashMapPtr，并进入这个 LinkedHashMap 中；如果“山”字存在，那么直接进入其下 HashMapPtr 所引用的 LinkedHashMap。

如此重复，直达操作到了字串的最后一个字符，这个字符所在的 WordStruct 的 P 被设置为整个字符串出现的次数，而 HashMapPtr 直接设置为 NULL，表示已经到达词尾。



二进制存储

对于字典数据的存储有很多中方案，比如条目存储、转换为 XML 文件存储，但这样都有一个转换过程，字典大了就会极为耗时，最好的方法就是按照内存中字典的结构去存储，使用的时候直接载入内存即可。

幸好 Java 提供了序列化对象的接口，能够方便地进行二进制存储与网络发送。只要使用 Serializable 接口，并声明变量 private static final long serialVersionUID 就能够将这个类进行序列化读写。既降低了转换的时间，同时也降低了存储字典所需的空间大小。

值得注意的是，应该自己为 serialVersionUID 赋值，而不是使用默认值。如果使用默认值的话，在之后增加或是删除类中的成员后，便无法载入之前所存储的二进制文件，原因是默认值 JVM 通过类名、接口名、成员方法及属性等来生成一个 64 位的哈希字段，这意味着任意的更改这个类都会导致最终编译的时候 serialVersionUID 的值会与上一次编译的值不同，就会导致读取原来生成的二进制文件反序列化失败。

豆瓣零分短评分类

评分分析方法

词袋模型

简单的来说，词袋模型就是将一个类别中所有对象中所包含的元素（在文本中是词）无序的列出，对于同类别的另外一个对象，对应这个类中所有的元素，有就置一，没有就置零，最终会获得一个向量，这个向量就是这个对象的特征。这个类别所有的元素就是这个类别的词袋。

由于词袋模型需要列出一个类别的所有元素，所以，像影评这样规模比较大的数据会导致最终的词袋很大，其中的任意一个对象的特征就是一个包含很多零的稀疏向量，极大的增加了运算量。所以使用短评这样较小的规模的数据能够加快计算速度。

K 最近邻

K 最近邻算法（KNN）主旨是在一堆给定的带标签的数据中取出与待监测点最近的 K 个数据，看这 K 个数据中那一个标签的最多，就判定待检测点属于这个标签，由此实现了对数据的分类判别。

KNN 要求必须有一堆已经有标签的数据，而这个数据恰好在使用词袋模型获取评论文本的特征向量的时候已经按照该评分所属的分值段有了标签，这样一来就能够很方便的使用 KNN 算法来实现对零分值的评论进行分类了，如果最终结果是高分值段，说明这是用户忘了评分，如果是低分值段，那么就是用户认为评分就是零。

特征向量的优化

剔除共有的高频词及各有的低频词

对于高分段的短评的词袋与低分段的短评的词袋中共用的高频词不仅不利于分类结果的准确度，还加重了运算量，通过剔除这些高频词，能够加深两个分值之间的差异度，同时削减词袋的大小也就缩短的其特征向量的长度，有效的加快了运算速度。

同共有高频词一样，各自的低频词汇对区分两者的贡献度不大，反而也增加了特征向量的长度，对运算速度有影响，删去之后更有利于运算。

使用 TFIDF 替代词频

TFIDF 即词频翻转文档频率，它能够评估某元素对该对象的重要程度。TF 是指给定词的出现频率，即在该样本中这个词出现次数与该样本所有词出现次数的比值；IDF 是指给定词的普遍重要性的度量，它由总样本数与包含该给定词的样本数的商取对数得到；最后计算 TF 与 IDF 的乘积就得到了 TFIDF 的最终结果。例如，在一个短评里有词语重复了多次，但在整个用于生成词袋的所有短评里这个词出现的次数相对比较小，那么这个词就能够相对别的它所包含的词更加能够代表其特征。所以使用 TFIDF 代替手工确定的高频剔除阀值和低频剔除阀值能够得到更加优化的词袋，使分类结果更加准确。