



# 云南大学软件学院期末课程报告

Final Course Report

School of Software, Yunnan University

## 个人成绩

序号	学号	姓名	成绩
1	20141030029	黄宇鑫	
2	20141120155	冯赫	
3	20141120175	甘京京	
4	20141120145	李佳奇	

学 期： 2016-2017 学年秋季学期

课程名称： 专业实训

任课教师： 王炜

实践题目： 专业实训期末报告

小 组 长： 黄宇鑫

联系电话： 15957127174

电子邮件： [2842343865@qq.com](mailto:2842343865@qq.com)

作业截止时间： 2016 年 12 月 28 日

# 项目成员及分工

(工作量总和为 100%)

序号	学 号	姓 名	工作内容	工作量 (%)
1	20141030029	黄宇鑫	LDA, LSI 算法对电影语料的分析, DBSCAN 算法对用户观影的情况的分析, K-Means, 傅里叶变换的应用。	35
2	20141120155	冯赫	豆瓣网数据的爬取, 八爪鱼的使用, 并行爬虫的学习, scrapy 框架的学习报告编写。	25
3	20141120175	甘京京	豆瓣网数据的爬取, 报告编写。	20
4	20141120145	李佳奇	豆瓣网数据的爬取, 报告编写	20

《专业实训》成绩考核表（每人填一份）

年级： 2014 专业： 软件工程 学号： 201410300029 姓名： 黄宇鑫

本人所做工作及排名： LDA, LSI 算法对电影语料的分析, DBSCAN 算法对用户观影的情况的分析, K-Means, 傅里叶变换的应用。 工作量： 35% 排名 1

指标内容	分值	指标内涵及评估标准				得分
技术路线的可行程度:需求分析	10	合理可行, 具体且有创新	合理可行, 具体	基本合理可行	不够合理或不够具体	
程序或系统设计思路:系统设计	10	程序或系统思路非常清晰、运行正确	程序或系统思路基本清晰、运行正确	程序或系统思路清晰、环境配置错误无法运行	程序或系统思路不清晰, 程序无法运行	
小组成员的工作量 (每个人分别打分)	15	高出平均要求工作量的 15%以上	高出平均要求工作量	达到平均要求工作量	低于平均要求的工作量	
理论知识应用水平:完成相关的必修要求的程度	15	用理论知识对程序设计方法、思路和代码进行详尽、准确地分析和总结	用理论知识对程序设计方法、思路和代码进行较准确地分析和总结	理论知识应用一般	理论知识差	
达到预期目标的程度	10	完全达到	基本达到	无法预见	未能达到	
团队合作精神	10	很强的团队合作精神	合作情况良好	合作情况一般	合作不好, 各自为政	
报告撰写质量包括程序注释的合理程度 (30 分)	5	报告非常完整	报告比较完整	完整程度一般	报告不完整	
	5	逻辑结构清晰	逻辑组织较好	逻辑组织一般	逻辑不清	
	5	内容非常丰富	内容较丰富	内容一般	内容欠缺	
	5	文字表达非常好	文字表达较好	文字表达一般	文字表达差, 意思不明了	
	5	图表制作非常专业化	图件制作良好	图件制作一般	图件制作效果差	
	5	整体效果很好	整体效果良好	整体效果一般	整体效果差	
综合得分 (满分 100 分)						
评语						

《专业实训》成绩考核表（每人填一份）

年级： 2014 专业： 软件工程 学号： 20141120155 姓名： 冯赫

本人所做工作及排名：豆瓣网数据的爬取，八爪鱼的使用，并行爬虫的学习，scrapy 框架的学习报告编写。

工作量：25% 排名 2

指标内容	分值	指标内涵及评估标准				得分
技术路线的可行程度:需求分析	10	合理可行，具体且有创新	合理可行，具体	基本合理可行	不够合理或不够具体	
程序或系统设计思路:系统设计	10	程序或系统思路非常清晰、运行正确	程序或系统思路基本清晰、运行正确	程序或系统思路清晰、环境配置错误无法运行	程序或系统思路不清晰，程序无法运行	
小组成员的工作量（每个人分别打分）	15	高出平均要求工作量的 15%以上	高出平均要求工作量	达到平均要求工作量	低于平均要求的工作量	
理论知识应用水平：完成相关的必修要求的程度	15	用理论知识对程序设计方法、思路和代码进行详尽、准确地分析和总结	用理论知识对程序设计方法、思路和代码进行较准确地分析和总结	理论知识应用一般	理论知识差	
达到预期目标的程度	10	完全达到	基本达到	无法预见	未能达到	
团队合作精神	10	很强的团队合作精神	合作情况良好	合作情况一般	合作不好，各自为政	
报告撰写质量包括程序注释的合理程度（30 分）	5	报告非常完整	报告比较完整	完整程度一般	报告不完整	
	5	逻辑结构清晰	逻辑组织较好	逻辑组织一般	逻辑不清	
	5	内容非常丰富	内容较丰富	内容一般	内容欠缺	
	5	文字表达非常好	文字表达较好	文字表达一般	文字表达差，意思不明了	
	5	图表制作非常专业化	图件制作良好	图件制作一般	图件制作效果差	
	5	整体效果很好	整体效果良好	整体效果一般	整体效果差	
综合得分（满分 100 分）						
评语						

《专业实训》成绩考核表（每人填一份）

年级： 2014      专业： 软件工程      学号： 20141120175      姓名： 甘京京  
 本人所做工作及排名： 豆瓣网数据的爬取，报告编写。      工作比例：20      排名 3

指标内容	分值	指标内涵及评估标准				得分
技术路线的可行程度:需求分析	10	合理可行，具体且有创新	合理可行，具体	基本合理可行	不够合理或不够具体	
程序或系统设计思路:系统设计	10	程序或系统思路非常清晰、运行正确	程序或系统思路基本清晰、运行正确	程序或系统思路清晰、环境配置错误无法运行	程序或系统思路不清晰，程序无法运行	
小组成员的工作量（每个人分别打分）	15	高出平均要求工作量的15%以上	高出平均要求工作量	达到平均要求工作量	低于平均要求的工作量	
理论知识应用水平:完成相关的必修要求的程度	15	用理论知识对程序设计方法、思路和代码进行详尽、准确地分析和总结	用理论知识对程序设计方法、思路和代码进行较准确地分析和总结	理论知识应用一般	理论知识差	
达到预期目标的程度	10	完全达到	基本达到	无法预见	未能达到	
团队合作精神	10	很强的团队合作精神	合作情况良好	合作情况一般	合作不好，各自为政	
报告撰写质量包括程序注释的合理程度（30分）	5	报告非常完整	报告比较完整	完整程度一般	报告不完整	
	5	逻辑结构清晰	逻辑组织较好	逻辑组织一般	逻辑不清	
	5	内容非常丰富	内容较丰富	内容一般	内容欠缺	
	5	文字表达非常好	文字表达较好	文字表达一般	文字表达差，意思不明了	
	5	图表制作非常专业化	图件制作良好	图件制作一般	图件制作效果差	
	5	整体效果很好	整体效果良好	整体效果一般	整体效果差	
综合得分（满分100分）						
评语						

《专业实训》成绩考核表（每人填一份）

年级： 2014 专业： 软件工程 学号： 20141120145 姓名： 李佳奇  
 本人所做工作及排名： 豆瓣网数据的爬取，报告编写。 工作比例:20 排名 3

指标内容	分值	指标内涵及评估标准				得分
技术路线的可行程度:需求分析	10	合理可行，具体且有创新	合理可行，具体	基本合理可行	不够合理或不够具体	
程序或系统设计思路:系统设计	10	程序或系统思路非常清晰、运行正确	程序或系统思路基本清晰、运行正确	程序或系统思路清晰、环境配置错误无法运行	程序或系统思路不清晰，程序无法运行	
小组成员的工作量（每个人分别打分）	15	高出平均要求工作量的 15%以上	高出平均要求工作量	达到平均要求工作量	低于平均要求的工作量	
理论知识应用水平:完成相关的必修要求的程度	15	用理论知识对程序设计方法、思路和代码进行详尽、准确地分析和总结	用理论知识对程序设计方法、思路和代码进行较准确地分析和总结	理论知识应用一般	理论知识差	
达到预期目标的程度	10	完全达到	基本达到	无法预见	未能达到	
团队合作精神	10	很强的团队合作精神	合作情况良好	合作情况一般	合作不好，各自为政	
报告撰写质量包括程序注释的合理程度（30分）	5	报告非常完整	报告比较完整	完整程度一般	报告不完整	
	5	逻辑结构清晰	逻辑组织较好	逻辑组织一般	逻辑不清	
	5	内容非常丰富	内容较丰富	内容一般	内容欠缺	
	5	文字表达非常好	文字表达较好	文字表达一般	文字表达差，意思不明了	
	5	图表制作非常专业化	图件制作良好	图件制作一般	图件制作效果差	
	5	整体效果很好	整体效果良好	整体效果一般	整体效果差	
综合得分（满分 100 分）						
评语						

# 目录

关于电影推荐的自然语言处理.....	1
1.1 语料库的建立.....	1
1.2 TF-IDF .....	2
1.3 LDA 矩阵 .....	3
1.4 LSI 矩阵 .....	5
1.5 数据可视化.....	7
1.6 组后呈现结果。 .....	8
1.7 关联分析.....	9
2 利用 DBSCAN 算法进行观影用户的聚类 .....	10
2.1 对 K-means 算法的学习 .....	10
2.2 K-means 算法的性质 .....	11
2.3 解决非球状簇的聚类问题---DBSCAN 算法的学习 .....	13
2.4 K-means 算法和 DBSCAN 算法的比较 .....	14
2.5 利用 Sklearn 和 pandas 实现 DBSCAN 算法 .....	15
2.6 聚类数据的格式.....	15
3 对支持向量机的学习.....	15
3.1 VC 维: .....	17
4 对各类距离的学习.....	17
4.1 余弦距离: .....	17
4.2 杰卡德相似性度量: .....	17
4.3 杰卡德距离.....	17
5 傅里叶变换的学习与应用.....	18
5.1 傅里叶变化简介.....	18
5.2 利用傅里叶变化处理时间序列.....	19
5.3 傅里叶变换的优点.....	20
6 对于 Pyhon 语言的学习: .....	22
6.1 Pandas:大规模数据读取的必备数据结构 .....	22
6.2 Sklearn 库-最强大的 pyhon 机器学习库.....	23
6.3 函数式编程, 一种高效率 Python 编程方法 .....	25
7 Python 爬虫学习 .....	27
7.1 结果截图:.....	27
7.2 使用豆瓣 API 获得豆瓣电影数据 .....	29
7.3 遇到的问题: .....	35
7.4 总结: .....	36
7.5 参考资料: .....	36
8 使用八爪鱼抓取数据学习.....	36
8.1 结果截图: .....	36
8.2 八爪鱼各个功能学习: .....	37
8.3 参考资料: .....	39
9 数据处理.....	40
9.1 结果截图: .....	40
9.2 Python2.7 读取中文文本文件的问题: .....	41

9.3	使用 jieba 包对中文进行分词处理: .....	42
8.4	正则表达式去除标点符号: .....	42
9.4	提取所有的电影名称: .....	43
9.5	去除数据里的换行符: .....	44
9.6	参考资料: .....	44
10	Pyqt 包使用 Python 编写 GUI 程序.....	44
10.1	环境搭建: .....	45
10.2	学习笔记: .....	45
10.3	参考资料: .....	45
11	Scrapy 爬虫框架学习 .....	45
11.1	环境搭建: .....	46
11.2	遇到的问题: .....	46
11.3	学习笔记: .....	47
11.4	使用 scrapy 基本流程: .....	48
11.5	参考资料: .....	57



# 关于电影推荐的自然语言处理

## 1.1 语料库的建立

进行语料处理时我们会遇到的主要问题之一就是如何将大的数据集读入内存当中然后进行相应的处理。

```
4844 SUPPERS FEATURE MUSIC FILM 2010 中村 哲平 中村 哲平 村上 信五 横山 裕 涉谷 昴 锦户亮 剧情 音乐 2010 10 28 9.1
4845 银河 奥特曼 S ウルトラマンギンガS 2014 坂本浩一 小林 雄次 根岸 拓哉 大浦 龍宇一 滝 裕可里 加藤 貴男
4846 神偷 艳贼 Gambit 2012 迈克尔 霍夫曼 伊桑 科恩 卡梅隆 迪亚茨 科林 费尔斯 艾伦 瑞克 曼 斯坦利 图齐 喜剧 犯罪 20
4847 胡 奇才 决战 新开岭 2016 高力 强 夏宇立 剧情 战争 2016 06 29 故事 主要 描写 我 辽东军区 第四 纵队 司令员 胡 奇
4848 五个 扑水 的 少年 ウォーターボーイズ 2001 矢口 史靖 矢口 史靖 妻夫 木聪 玉木宏 近藤 公园 金子 贵俊 喜剧
4849 冠军 欧洲 2010 贺炜 冠军 欧洲 是 中央电视台 体育频道 专 为 欧洲 冠军联赛 推出 的 一档 特别节目 每期 节目 在 北
4850 第 58 届 格莱美奖 颁奖典礼 The 58th Annual Grammy Awards 2016 Louis J. Horvitz Ken Ehrlich 泰勒 斯威夫特 布鲁
4851 潘金莲 复仇记 2016 吴双 何 欢欢 动作 爱情 2016 03 1 4.1 潘金莲 自幼 在 西门 府 做 丫鬟 因 貌美如花 一直 受到 西
4852 杀出 个 黎明 From Dusk Till Dawn 1996 罗伯特 罗德里格 兹 昆汀 塔伦 蒂诺 乔治 克鲁尼 昆汀 塔伦 蒂诺 哈威 凯特尔
4853 誓约 The Vow 2012 迈克尔 苏 克西 Abby Kohn 查宁 塔图姆 瑞秋 麦克 亚当斯 斯科特 斯 比德曼 杰西卡 麦克 娜 美 剧情
4854 夺宝奇兵 2 Indiana Jones and the Temple of Doom 1984 史蒂文 斯皮尔伯格 威 拉德 赫依克 动作 冒险 1984 05 23 美国
4855 游泳 回家 2016 王一 离 王一 离 剧情 家庭 上映 日期 9.2 八岁 的 方克柔 意外 听说 自己 并非 是 父母 的 亲生 孩子 然
4856 大叔 与 棉花 糖 おじさんとマシュマロ 2016 ひらさわ ひさよし 岡篤志 动画 官方网站 2016 01 07 日本
4857 饥饿 Hunger 2008 史蒂夫 麦 奎因 Enda Walsh 斯图尔特 格雷厄姆 Laine Megaw Brian Milligan Liam McMahon 剧情 传记
4858 反斗 智多星 2 Wayne 's World 2 1993 Stephen Surjik Bonnie Turner 喜剧 音乐 1993 12 10 6.6 韦恩 麦克 迈尔斯 「
4859 唐顿 庄园 中 的 礼仪 Masterpiece The Manners of Downton Abbey 2015 Louise Wardle Alastair Bruce 官方网站 制片
```

(图中所示的数据集是豆瓣网上 5000 部电影描述信息的分词结果，每一行的数据是一部电影的所有信息。不过从数据大小上看，该数据集属于小的数据集，大的数据集从容量上讲就是以 GB 甚至以 TB 为单位)

### 1.1.1 Linecache 模块

对于大的数据，python 提供了很多的功能进行读取，其中之一就是 linecache 模块。Linecache 模块功能非常强大，其中包含了多种函数进行调用 linecache.getlines(filename)可以读取文件中的所有内容。linecache.getline(filename,lineno)可以读取文件中的任意行。另外，普通的文件读取方式都是类似于：m=open("F:\\Programming...", 在 python 语言当中，这样的文件读取方式是非常不适合大型数据的读取的，因为一旦采用这种形式进行数据的读取，就会导致所读文件整体被写入内存当中，如果文件过大，内存就无法存储，读取一定会失败。不过，linecache 模块的读取是一种类似于迭代器形式的读取，从而可以使得在读取大的文件的时候不会将所有的文件一次性全部读入，导致因为内存不够产生的程序错误。

## 1.1.2Corpora 函数

如上图所示，将语料读入程序的数据结构中后，我们就要进行语料库的建立。语料库中所容纳的数据基本可以概括为出现的词的编号以及该词出现的次数。这个工作看似简单，但是如果让研究人员单独编写仍然会耗费非常多的时间，因此，python 其中的一个支持库 `gensim` 中专门提供了语料库的建立工具 `gensim.corpora` 函数对语料进行转换。

```
(5774, 1), (5810, 1), (5823, 2), (5837, 2), (5840, 1), (5855, 1), (5917, 2),
(6425, 1), (6630, 2), (6922, 2), (7772, 1), (7919, 2), (8525, 1), (8613, 1),
(9007, 7), (9035, 1), (9058, 1), (9079, 1), (9091, 1), (9108, 1), (9284, 1),
(9900, 3), (10108, 1), (10120, 2), (10140, 1), (10141, 1), (10142, 2), (10150,
3), (10151, 1), (10152, 1), (10174, 1), (10178, 1), (10183, 1), (10232, 1),
(10743, 2), (10764, 2), (11028, 1), (11938, 1), (12162, 1), (12271, 1), (12649,
1), (12674, 1), (12844, 3), (12945, 1), (13708, 1), (13824, 1), (14433, 1),
(14436, 2), (14672, 1), (15016, 1), (15017, 1), (15309, 1), (16342, 1), (18883,
1), (18925, 1), (21833, 1), (22026, 1), (22046, 1), (22051, 1), (22829, 1),
(23409, 1), (23413, 1), (23486, 1), (24381, 1), (24496, 1), (26276, 1), (27025,
1), (27029, 2), (29305, 2), (30628, 1), (30693, 1), (31179, 1), (31561, 1),
(31623, 1), (34385, 1), (34492, 1), (34840, 1), (34940, 1), (35269, 1), (35669,
1), (36764, 1), (38639, 1), (43473, 1), (44196, 1), (48506, 1), (53887, 1),
(55078, 1), (55987, 2), (56889, 1), (57517, 2), (61335, 1), (66813, 1), (69861,
3), (71521, 1), (71523, 2), (72826, 1), (75572, 2), (83573, 1), (100590, 1),
(100591, 1), (100592, 1), (100593, 1), (100594, 2), (100595, 1), (100596, 2),
(100597, 1), (100598, 1), (100599, 1), (100600, 1), (100601, 1), (100602, 1),
(100603, 1), (100604, 1), (100605, 1)], [(0, 1), (18, 2), (30, 2), (42, 1), (44,
1), (60, 13), (75, 2), (77, 2), (82, 1), (86, 1), (87, 2), (94, 2), (117, 1),
(120, 1), (132, 1), (135, 1), (141, 4), (142, 2), (144, 1), (156, 1), (157, 1),
(202, 2), (221, 2), (224, 2), (236, 1), (239, 2), (262, 3), (264, 1), (272, 1),
(287, 1), (321, 3), (365, 1), (392, 1), (408, 2), (413, 1), (431, 1), (450, 1),
```

用 `gensim.corpora` 函数后进行语料转换后的结果如图所示。每个括号当中的值代表的就是一个词的信息，第一个值是该词的编号，第二个词是这个词在一条电影评论中出现的次数。词语的编号是按照整个语料库中的词语进行编制的。

## 1.2 TF-IDF

### 1.2.1TF\_IDF 简介

语料库建立完毕之后就需要进行对语料中词语的进一步分析。在我们的生活当中，一篇文章，一条说说，一句评语当中的词语的价值并不相同。我们说任何话一般都会用到“的”，因此像“的”这种词语对于区分不同类型的语句的价值就几乎为 0 了，但是加入一句话中出现一个词“吕克贝松”，我们可以立马分析得到这句话很有可能和某一部电影有关。因此，在进行语料的具体分析之前，我们需要对语料中词语的重要性进行分析，这就诞生了 `tf-idf` 方法。

## 1.2.2 TF\_IDF 应用

tf-idf 的主要思想是：如果某个词或短语在一篇文章中出现的频率（TF）很高，并且在其他文章或者评论中出现很少，则认为此词或者短语具有很好的类别区分能力，适合用来分类。很多人或许会困惑 tf 和 idf 两个词的实际意义，TF 表示的是词频（Term Frequency），词频比较好理解，即是某个词在整个文档中出现的频率。然而，光用词频来表示词在整个数据集当中的重要程度是不够科学的，比如前文中提到的“的”“这个词语，出现的频率会在素有的电影评论中出现非常高，虽然高，但很明显不具有什么区分能力。因此，还需要引入另外一个概念，即 IDF。IDF 表示的含义是越少的文档（本项目中代表的是一条电影的描述信息）包含这个词，说明这个词有更好的信息区分能力。

```
[[ 0.02016028 0.06983666 0.03595935 ..., 0. 0. 0. ]
 [ 0. 0. 0. ..., 0. 0. 0. ]
 [ 0. 0. 0. ..., 0. 0. 0. ]
 ...,
 [ 0. 0.1411273 0. ..., 0. 0. 0. ]
 [ 0.03966628 0.03925911 0. ..., 0. 0.12370797
 0.12370797]
 [ 0. 0.05295209 0. ..., 0. 0. 0. ]]
```

上图所示的是用 tf-idf 对语料库进行进一步分析后的结果矩阵的一部分

## 1.3 LDA 矩阵

### 1.3.1 LDA 模型的介绍

LDA 模型的全称是隐含狄利克雷分配（LDA，Latent Dirichlet Allocation）。是一种今年来发展起来的一种非常重要的离散数据集合的建模方法。其主要的功能就是可以生成文本的主题分布向量，分析然后挖掘出文本的潜在知识。是一种非常优秀的文本聚类的预处理方法。

对于一般的语料分析：“今天下雨了”和“我今天淋湿了”一定是几乎没有相似点的两句话，然而在实际的生活当中，这两句话很有可能表示的是几乎相同的意思。这就需要引入 LDA 模型进行。其基本的思想就是先对类似的文本进行聚类，确定每个文本的隐含主题，然后进行分析。

其主要的优点在于会考虑潜在的语义信息，不单纯从词频角度进行分析，还可以映射至内部的隐含主题，过滤噪音。

### 1.3.2 LDA 语料处理的实现

[illegible]

如上图所示是利用 **lda** 模型对语料进行分析的结果，所得结果的数值的含义是。每一个列表中所有的数值组成了一条电影信息的向量。每条向量都唯一标识了一条电影的信息的各个特征。在本次实训当中，我们一共有近 **5000** 条电影信息，因此共生成 **5000** 条向量标识各个电影信息。

### 1.3.3LDA 模型语料的查询

建立好 LDA 矩阵后，我们就可以利用查询的语句的生成的 LDA 模型对应的向量和系统中原先就计算好的 LDA 模型中的向量进行比较。选择出和查询语句距离最小的前 10 个向量。过匹配得到的向量，利用实现编写好的 mapping 函数就可以找出这些向量对应的电影编号，最终找出匹配的电影。





## 1.4.2 LSI 模型语料处理的实现

```
-0.010831815170754258, 0.039620099948590046, 0.044486606203516181,  
0.013886117585870578, 0.014230990787190201, -0.016804458736606801,  
0.020096836653143168, 0.0096338538173749225, -0.0071630787994862461,  
0.037267065445656306, -0.012508854384282854, -0.0093450700213645356,  
0.017261169909650837, 0.019629566103008228, 0.0068131421801245155,  
-0.041947030189031002, 0.010141830089789538, 0.0066258845982423492,  
-0.0012419904797977125, -0.019464700146200248, -0.021553134261842941,  
-0.007001526257144345, 0.01195638669812849, -0.017906064562664494,  
-0.0494296959371677, -0.036768790314457733, -0.055569112095363957,  
0.014511751752577235, 0.015519104433111072, 0.041364692176217549,  
-0.0041738852001879777, -0.021879604391372154, -0.023561607926904864,  
-0.00082741451160216607, -0.0072458405319049584, -0.015960136830881766,  
0.0028389384020491475, -0.0065040321337449351, -0.038423600215496879,  
-0.0077541861319511858, -0.017396295094632881, 0.022940739891305205,  
-0.00094360076311963742, 0.025878110793026678, 0.027916049352631131,  
-0.045214521921417845, -0.017019048805314945, -0.00038249968896007417,  
0.012187377236042213, 0.0016239427735364048, -0.011105253822116506,  
-0.0089015955706466306, 0.036389482220988575, 0.0054613216060504588,  
-0.05613076703590951, -0.0044958212350844741, -0.006320200589525881,  
-0.027514030330581463, 0.033036088492427569, -0.013075968711696218,  
-0.0084953072921779109, 0.011444804764465549]]
```

上图是利用 LSI 模型进行语料处理后生成的矩阵，从该矩阵中我们可以发现，不同于 LDA 矩阵，该矩阵不是一种稀疏矩阵，因而每条向量直接的相似度更加低，最后处理的结果也更加好。

下图是通过描述电影风暴最后搜索得到的结果：

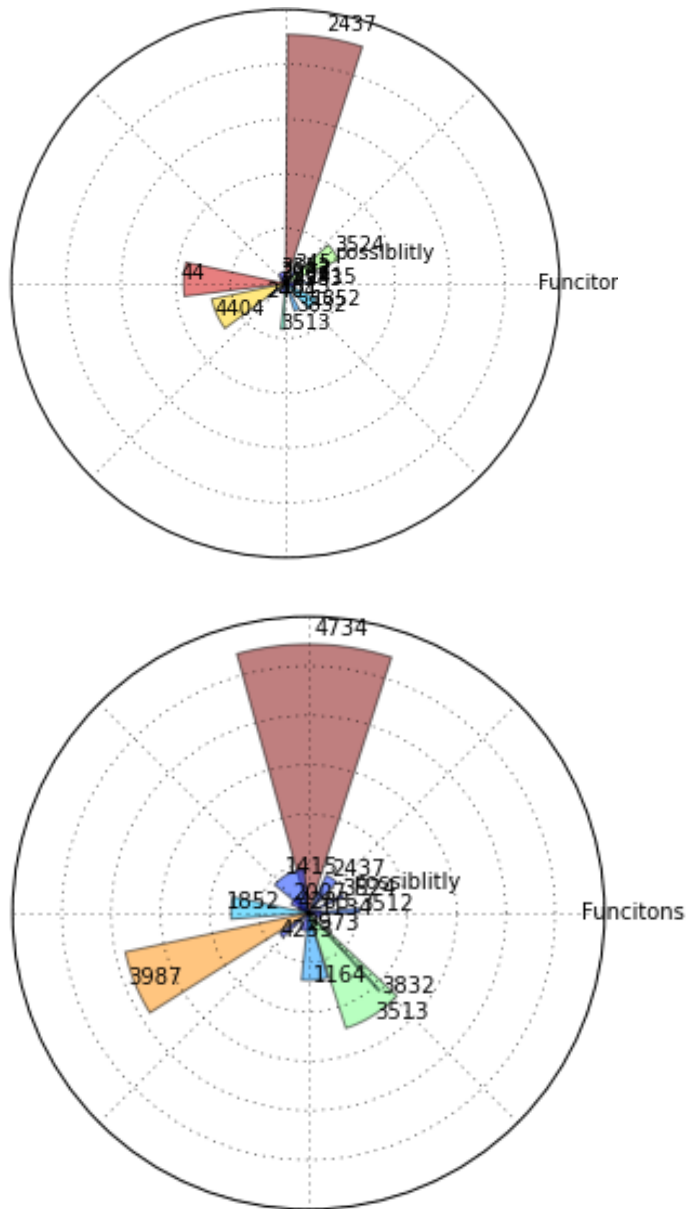
```
3524  
我怒了 I Am Wrath  
1415  
暗花  
4404  
痞子英雄2：黎明升起 痞子英雄2：黎明再起  
2186  
喋血双雄  
3129  
风暴 風暴  
3512  
五路追杀令2：刺客舞会 Smokin' Aces 2: As  
1164  
心理罪  
1009  
线人 線人  
----
```

可见利用 LSI 进行语料分析最终得到的结果准确度还是非常不错的。

## 1.5 数据可视化

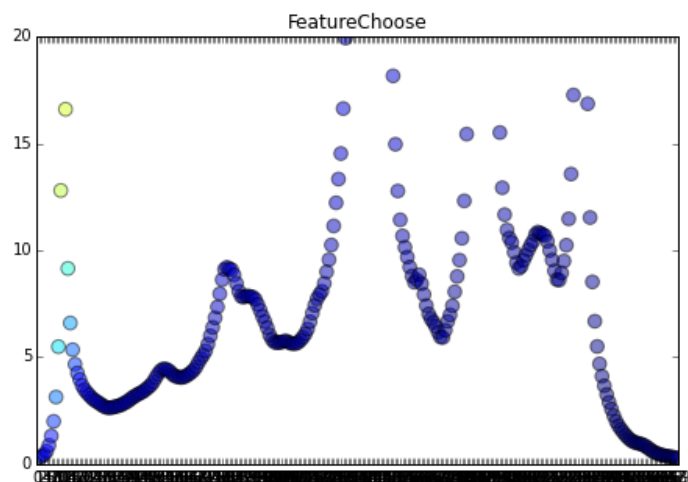
### 1.5.1 扇形图

利用 python 进行数据处理的一个很大的优点就是有很多支持它进行画图的库。其中使用得最多的就是 matplotlib 库，利用 matplotlib 可以绘制各种类型的图表，实现数据的可视化。在此次实训当中，我们小组主要利用了基于 matplotlib 库中的 pylab 库进行绘图。



对电影进行匹配后每个电影相关性大小的匹配结果可以使用扇形图进行表示。

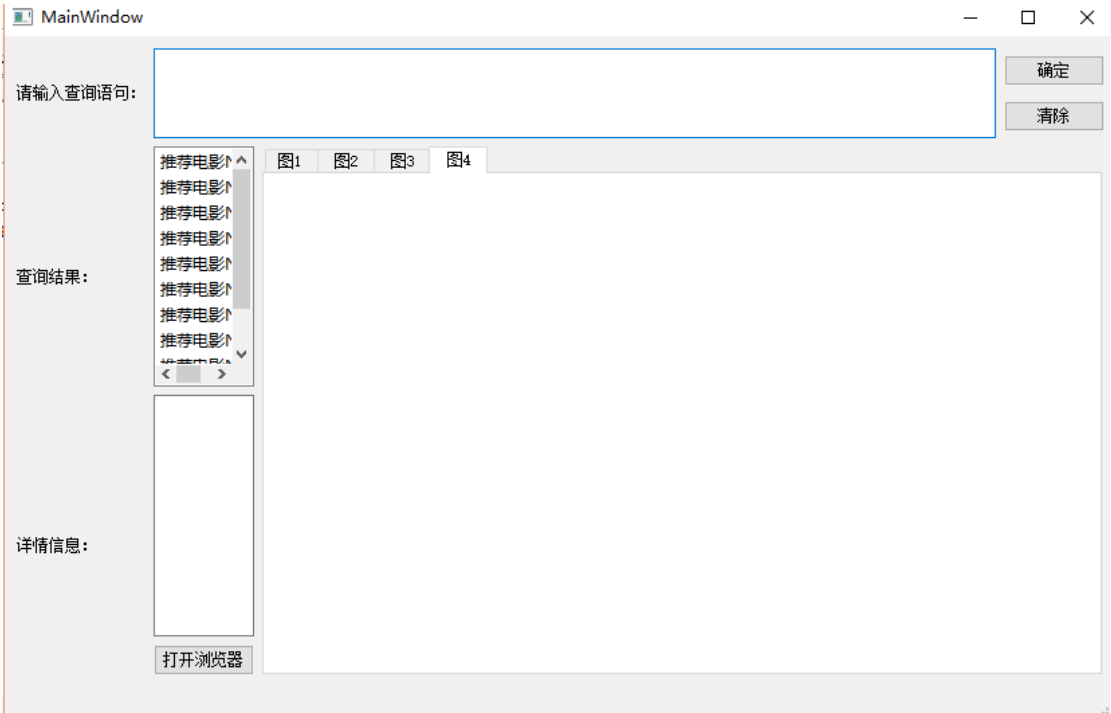
1.5.2点状图



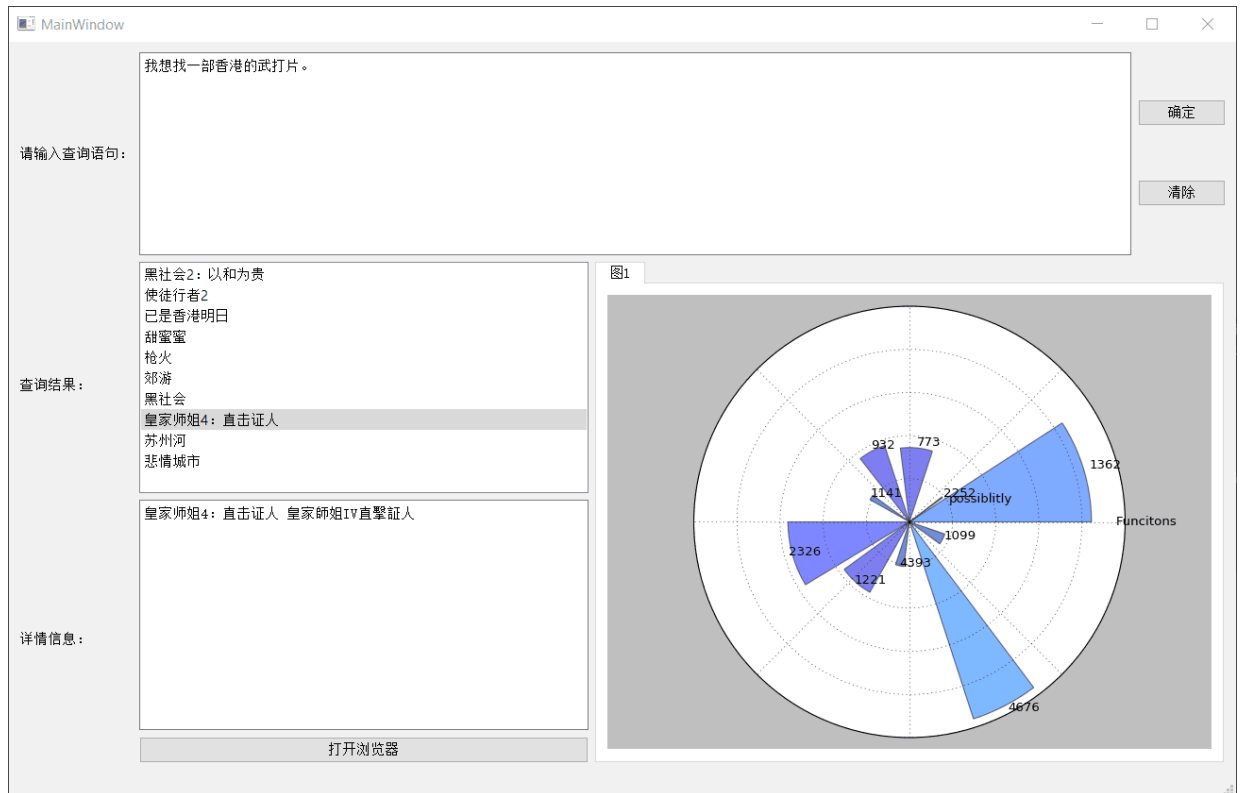
在进行特征提取的时候可以使用点状图来描述各个向量特征的权重。

其他图形比如箱图，3 维图等都可以在 [matplotlib](#) 官网上寻找到相应的素材，略加修改后就可以绘制非常美观的图片了。

1.6 组后呈现结果







上图所示是小组所编写的 GUI 程序的最终界面。

## 1.7 关联分析

### 1.7.1 关联分析简介

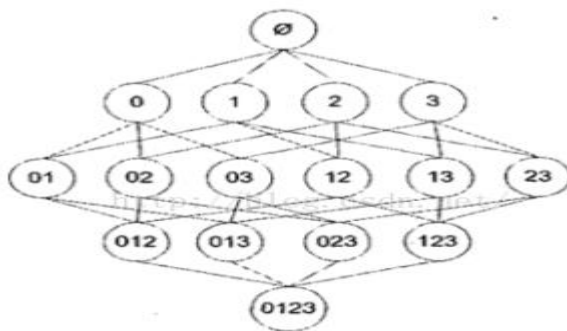
交易号码	商品
0	豆奶, 莴苣
1	莴苣, 尿布, 葡萄酒, 甜菜
2	豆奶, 尿布, 葡萄酒, 橙汁
3	莴苣, 豆奶, 尿布, 葡萄酒
4	莴苣, 豆奶, 尿布, 橙汁

图11-1 一个来自Hole Foods天然食品店的简单交易清单

上图是一个非常普通的食物交易单。虽然看似简单，但其中却包含了许多关系。在关联分析当中，我们将上图所示的商品交易列表抽象为频繁项集。

当寻找频繁项集时，有两个概念比较重要：支持度和可信度。一个项集的支持度被定义为数据集中包含该项集的记录所占的比例。可信度或置信度（confidence）是针对关联规则来定义的。可被定义为  $\text{可信度}(B) = \frac{\text{支持度}(\{A,B\})}{\text{支持度}(\{A\})}$ 。

### 1.7.2 Apriori 算法:



上图所示的是{0, 1, 2, 3}中所有可能的项集组合

Apriori 算法的本质其实是一种频繁集的优化算法，算法的核心思想在于，如果一个集合的某一个子集是频繁的，那么这个集合一定是频繁的。

### 1.7.3 采用关联分析进行电影的推荐

Python 的支持库非常强大，在本次实训当中，我么小组采用的是 orange 库中的 `rules=Orange.associate.AssociationRulesSparseInducer(data,confidence=0.4,support=0.23,max_item_sets=1000000)` 该函数进行关联分析。Orange 库中的关联函数默认使用的是 Apriori 算法。另外，需要注意的是，进行关联分析数据格式需要是 basket 为后缀名的数据。

## 2 利用 DBSCAN 算法进行观影用户的聚类

### 2.1 对 K-means 算法的学习

使用 K 均值聚类算法对数据进行聚类的过程很简单，只需要人为指定 K 的值即可。这里的 K 值表示将要把数据聚成 K 个簇。

基本算法:

人为设置 k 的值。

随机选择 k 个初始点作为初始质心（可以认为是每个簇的中心），计算每个数据点距离这些质心的距离，对每个数据点找出距离最近的那个质心，把这个数据点指派给这个质心，所有被指派给同一个质心的所有数据点形成一个簇。然后根据指派给簇的点，重新确定每个簇新的质心点。重复这个计算距离，指派质心，更新簇的质心的过程，直到质心不再发生变化为止。

伪码描述:

选择 k 个点作为初始质心

Repeat

将每个点指派给最近的质心，形成  $k$  个簇

重新计算每个簇的质心

Untill 质心  $k$  不再变化

判断每个数据点到底被指派给哪个质心可采用不同的距离，比如：欧氏距离。欧式距离就是两点之间的欧式空间直线距离。我这里使用了欧式距离。

最终判断聚类质量，我使用了老师上课讲的方法来判断。通过计算实际聚类结果出来的类标序列和真实的类标序列带入到老师提供的公式中来衡量聚类的质量。

## 2.2 K-means 算法的性质

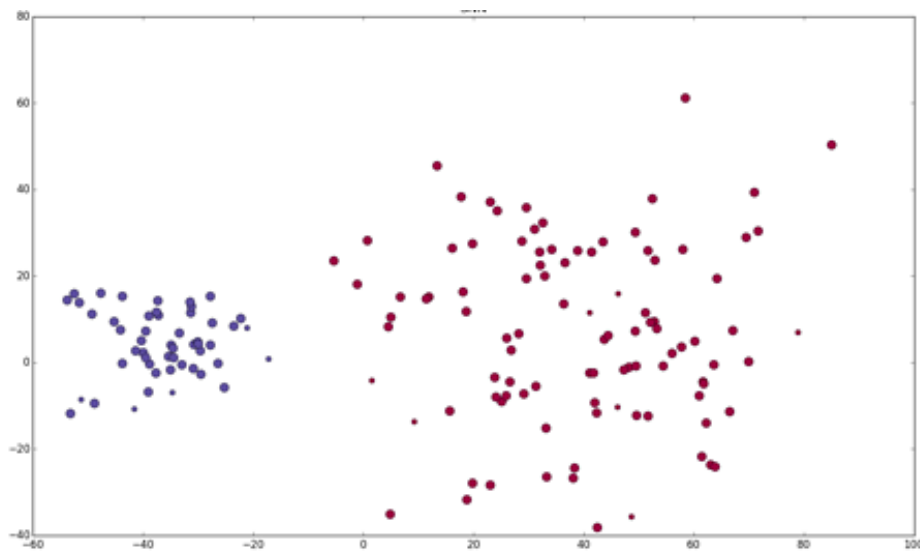
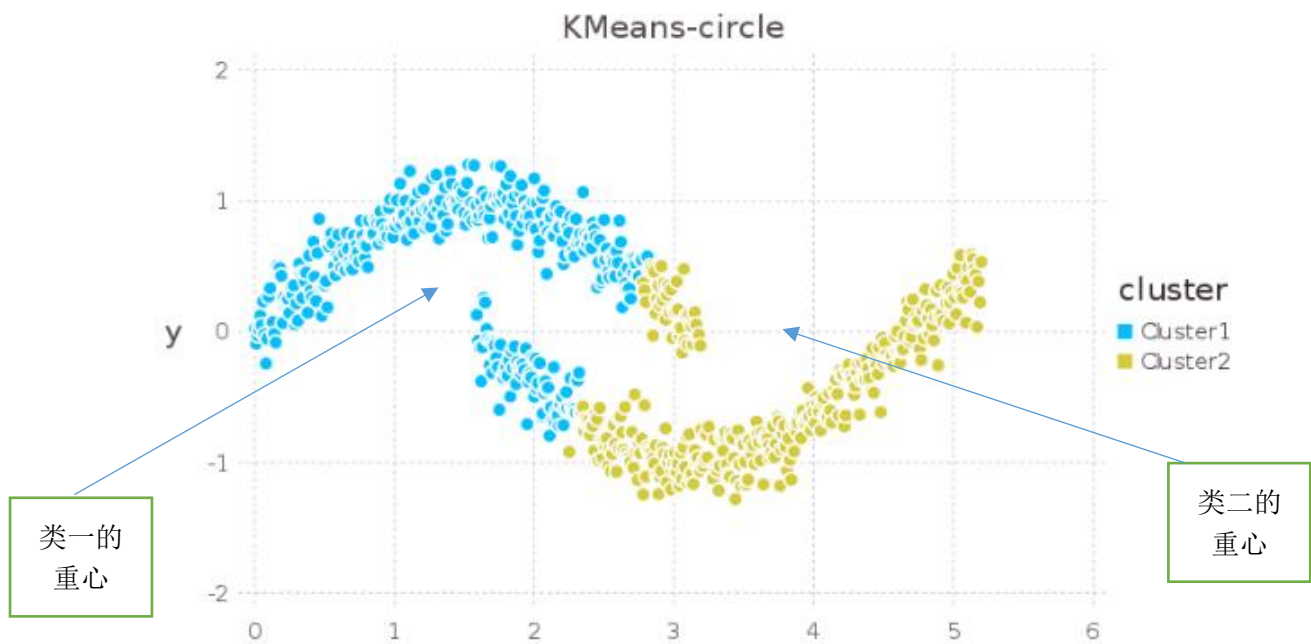
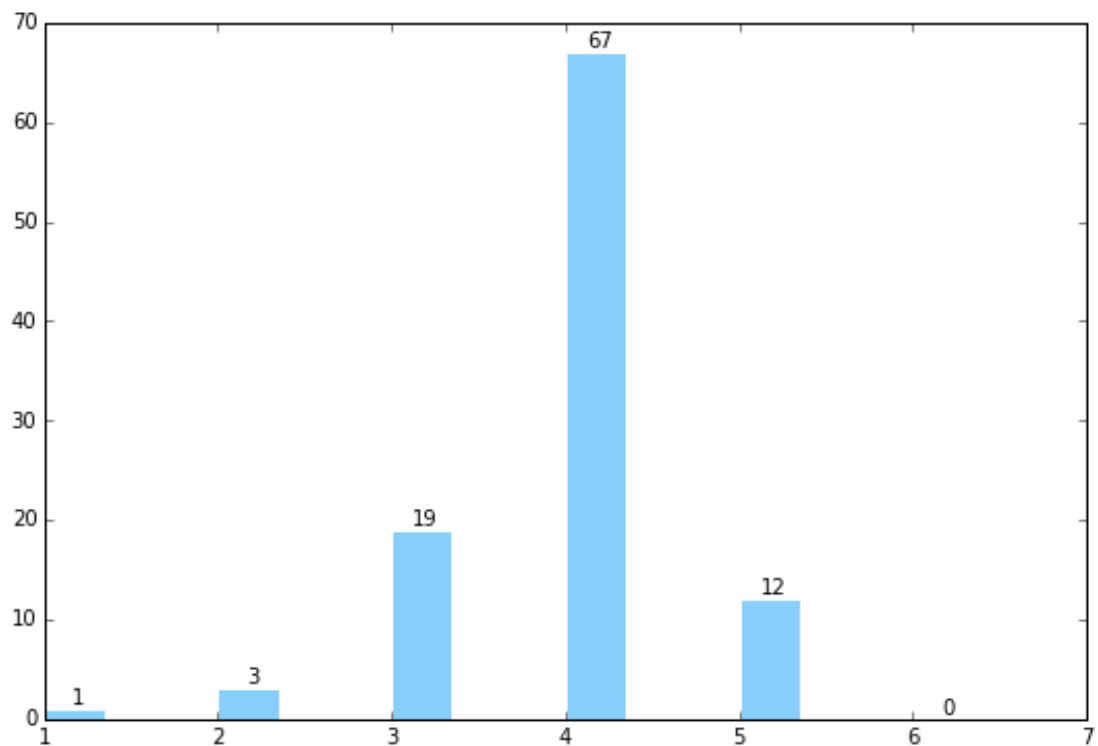


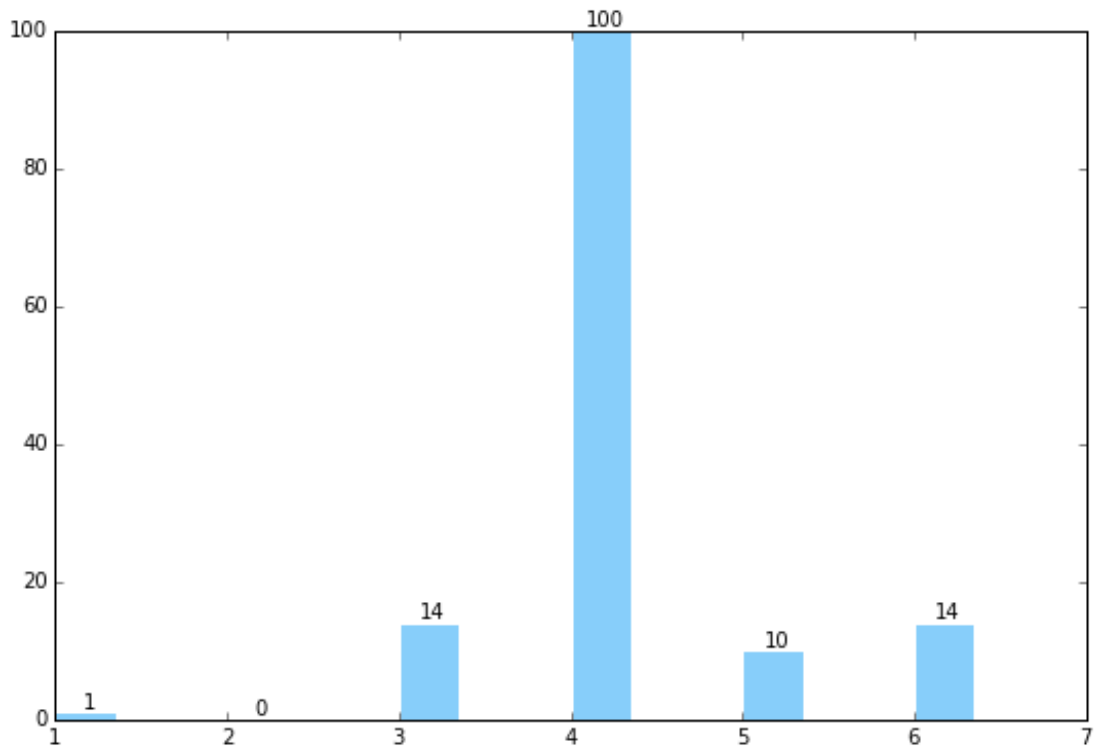
Figure 1

如上图所示，使用 K-means 算法的对球状簇的聚类往往会有一个比较好的效果。但是，如下图所示：



如果是像图中的条状数据，K-means 算法就会显现出它本身的局限性。即无法对于非球状的数据集进行符合其自身形状的聚类。从图中的聚类结果我们可以发现，原本应该聚为一类的数据在 k-means 算法的结果中被拆分了。在此基础之上，我们小组对观影人员的习惯进行调查，发现用户的观影习惯很有可能不会呈现一种球状的分布。





如上图所示，用户的观影习惯的分布往往是对某一类型的电影有着很大的兴趣，因此，聚出的簇的形状很有可能会变成中类似长条形的形状。

## 2.3 解决非球状簇的聚类问题---DBSCAN 算法的学习

DBSCAN 算法是一种代表性的基于密度的聚类算法，它将簇定义为密度相连的点的最大集合，能够把具有足够高密度的区域划分为簇，并可在噪声的空间数据库中发现任意形状的聚类。

DBScan 需要二个参数:扫描半径 (epsilon)和最小包含点数(minPoints)。

算法的步骤可以划分为：

1. 任选一个未被访问的点开始，找出与其距离在 epsilon 之内(包括 eps)的所有附近点，对每一个点的附近点的个数进行记录。
2. 统计好每个点的附近点个数后就可以对所有的点进行划分了。DBSCAN 算法中共可以划分出三种类型的点：
  - i. 核心点：指的是附近点的个数超过了给定阈值(minPoints)的点。这些点再基于密度的簇内部。
  - ii. 边界点：不是核心点，但是落在核心点的邻域内。
  - iii. 噪声点：噪声点是非核心点也非边界点的任何点。
3. 删除所有的噪声点
4. 为距离在 Eps 之内的所有核心点之间赋予一条边。
5. 每组联通的核心点形成一个簇
6. 将每个边界点只拍到一个与之关联的核心点的簇中，形成最终的结果。

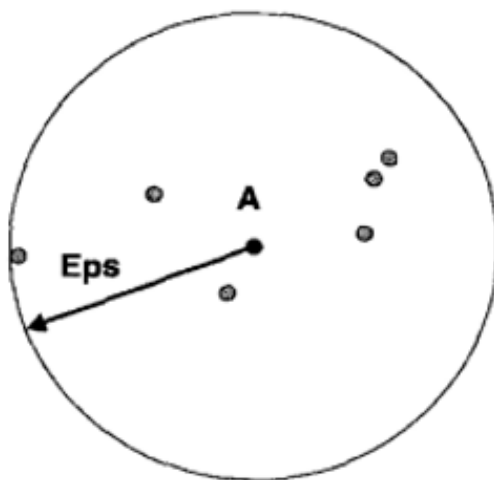
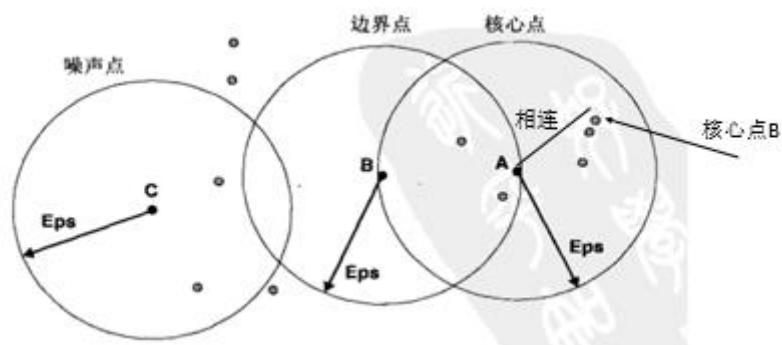


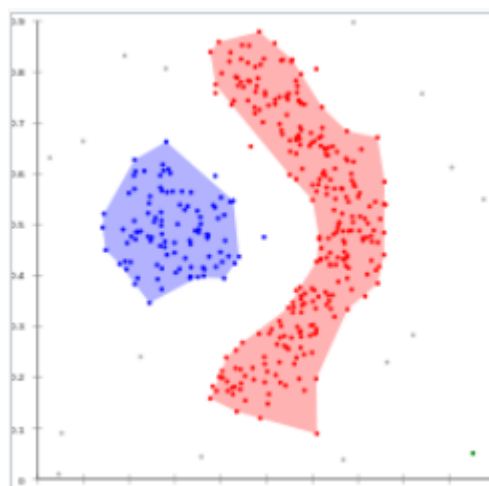
图 8-20 基于中心的密度



在确定点的类别的时候，是利用点的扫描半径以及扫描半径内点的数数目和  $\text{minpoints}$  的值的比较得出的。

## 2.4 K-means 算法和 DBSCAN 算法的比较

利用 DBSCAN 进行聚类的效果可以如下图所示：



从图中我们可以发现，DBSCAN 聚类可以产生如图中所示的效果，即条状数据的聚类。相比于 DBSCAN，K-means 算法只能产生球状的簇，因此会对一些形状的数据集运行出非常不令人满意的效果。

## 2.5 利用 Sklearn 和 pandas 实现 DBSCAN 算法

对 DBSCAN 算法的学习之处，我们小组对 dbscan 算法的实现方式是纯粹自己更具算法来进行编写，不过，后来我们小组发现利用 python 进行数据聚类的好处之一就是 python 中有很强大的 sklearn 库。Sklearn 中的 dbscan 算法功能非常强大，支持多个参数，eps, min\_samples, metric\_algorithm 等。Eps 参数就是代表的就是点的扫描半径的大小。Min\_samples 代表的是最小包含点数。而 metric 是 sklearn 库中几乎每个算法都带有的距离方式，在其中可以设置距离为欧式距离，余弦距离，或者为自定义的距离（precomputed）。

```
dbscan = cluster.DBSCAN(eps=0.3,min_samples=3,algorithm='brute',metric='euclidean')
```

（以往需要大量代码完成的 DBSCAN 算法，利用 sklearn 库只需要一行代码）

## 2.6 聚类数据的格式

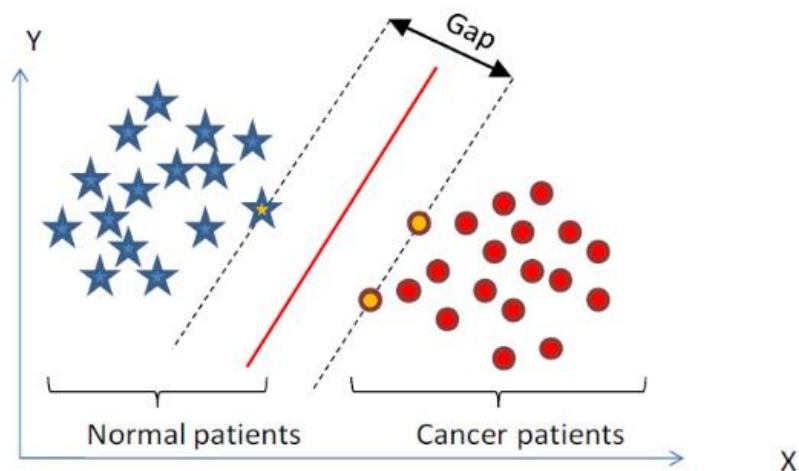
```
2 4 7 8 1 2
3 4 5 9 8 1
4 5 6 8 9 1
8 4 3 2 1 0
9 8 12 2 1 9
3 4 2 4 1 5
8 6 14 2 8 9
1 3 4 5 9 8
0 8 1 1 1 1
0 9 1 1 1 1
0 12 1 1 1 0
0 14 2 1 1 0
1 2 1 1 1 1
0 3 2 1 1 0
```

如上图所示，在本次实训当中，我们小组对豆瓣网上各个用户的观影情况进行了爬取，得到关于各个用户的观影情况数据。上图所示的每一行数据代表每一个用户关于“爱情类”，“动作类”，“科幻类”，“恐怖类”，“喜剧类”，“动画类”电影的历史观看情况。将这 6 类电影作为 6 个维度，使用 dbscan 算法进行用户的观影的习惯分析。

## 3 对支持向量机的学习

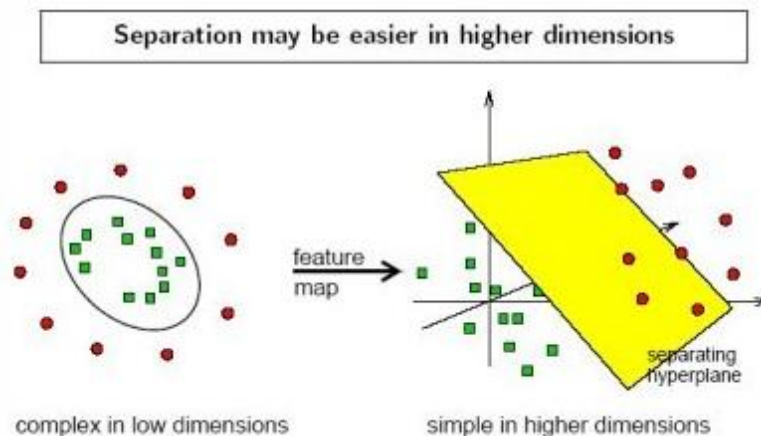
支持向量机(Support Vector Machine)是 Cortes 和 Vapnik 于 1995 年首先提出的基于统计学习理论的 **VC 维理论**和**结构风险最小原理**基础上的机器学习方法，根据有限的样本信息在模型的复杂性（即对特定训练样本的学习精度，Accuracy）和学习能力（即无错误地识别任意样本的能力），通过结构化风险最小来提高学习机泛化能力，经验风险和置信范围的最小化，从而达到在统计样本量较少的情况，亦能获得良好统计规律的目的。从而在小样本、非线性及高维模式识别中表现出许多特有的优势，并能够推广应用到函数拟合等其他机器学习问题中。

通俗来讲，它是一种二类分类模型，其基本模型定义为特征空间上的间隔最大的**线性分类器**，即支持向量机的学习策略便是间隔最大化（最大间隔分类器），最终可转化为一个凸二次规划问题的求解。



**函数间隔 functional margin** 和 **几何间隔 geometrical margin** 相差一个  $\|w\|$  的缩放因子。对一个数据点进行分类，当它的 **margin** 越大时候，分类 **confidence** 越大。对于一个包含  $n$  个点的数据集，我们可以很自然地定义它的 **margin** 为所有这  $n$  个点的 **margin** 值中最小的那个。于是，为了使得分类的 **confidence** 高，我们希望所选择的超平面 **hyper plane** 能够最大化这个 **margin** 值。

实际中，我们会经常遇到线性不可分的样例，此时，我们常用做法是把样例特征映射到高维空间中去(如下图)以此使数据重新线性可分。这转化的关键便是核函数：



在新的维上，搜索最优分离超平面。**SVM** 通过搜索最大间隔超平面来处理最佳超平面问题，分离超平面：

$$g(x) = W \bullet X + b = 0$$

由最优超平面定义的分类决策函数为：

$$f(x) = \text{sgn}(g(x))$$



### 3.1 VC 维:

所谓 VC 维是对函数类的一种度量，可以简单的理解为问题的复杂度，VC 维越高，一个问题就越复杂。因为 SVM 关注的是 VC 维，后面我们可以看到，SVM 解决问题时候，和样本的维数无关（甚至样本是上万维的都可以，引入了核函数，这使得 SVM 很适合用来解决文本分类问题）。

## 4 对各类距离的学习

### 4.1 余弦距离:

余弦距离，也称为余弦相似度，是用向量空间中两个向量夹角的余弦值作为衡量两个个体间差异的大小的度量。余弦距离的主要作用在于可以对高维空间中的数据进行更好的衡量。

### 4.2 杰卡德相似性度量:

两个集合 A 和 B 交集元素的个数在 A、B 并集中所占的比例，称为这两个集合的杰卡德系数，用符号  $J(A,B)$  表示。杰卡德相似系数是衡量两个集合相似度的一种指标（余弦距离也可以用来衡量两个集合的相似度）。

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

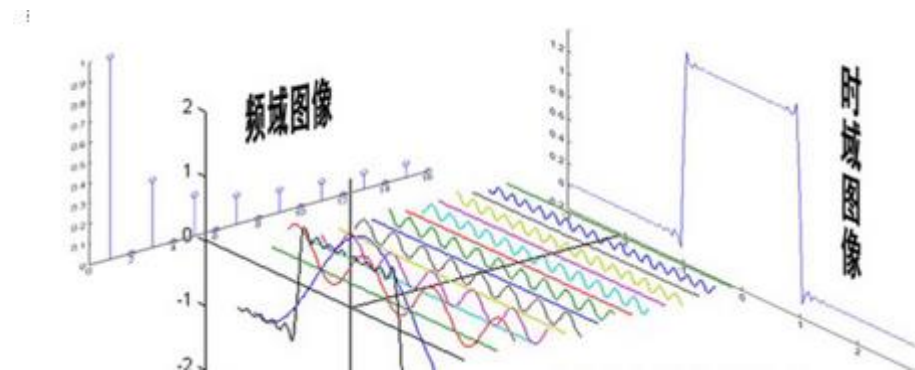
### 4.3 杰卡德距离

$$J_{\delta} = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

## 5 傅里叶变换的学习与应用

### 5.1 傅里叶变化简介

信号的一种最为自然的表达方式，就是看成自变量为时间，因变量为强度的函数形式，但事实上，这里还存在第二种方式，就是把信号分解成不同频率的三角函数。傅立叶变换就是将原来难以处理的时域信号转换成了易于分析的频域信号



（任何一种波形的信号都可以认为是无数正弦波信号的无穷叠加），  
以我们所学过的傅里叶级数（类似傅里叶变换）公式表述：

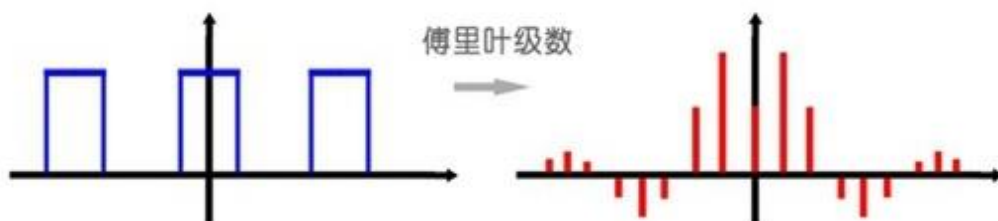


Figure 错误!文档中没有指定样式的文字。-2 傅里叶级数

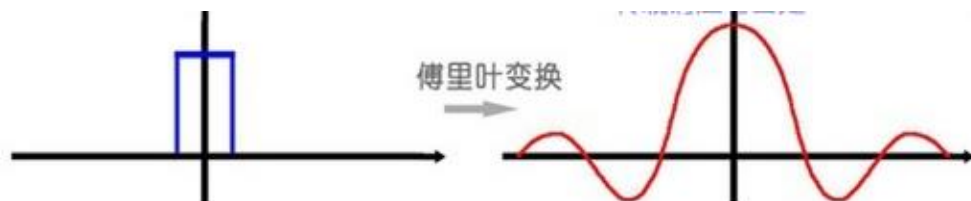
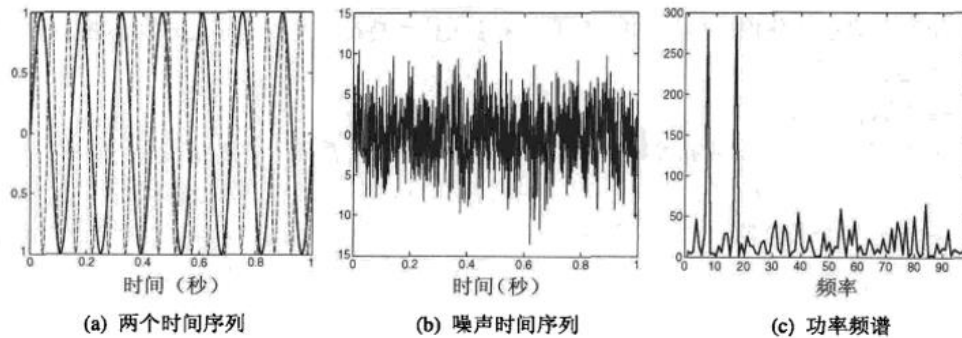


Figure 错误!文档中没有指定样式的文字。-3 傅里叶变换

上面两张图大致阐释了傅里叶级数和傅里叶变换之间的不同之处

通过对信号的分解，我们可以得到该信号所包含的各种正弦波信号的权重  $a_n, b_n$ ，而这些权重其实就是原有信号在另外一个空间当中的映射，因此，我们可以采集信号分解后在不同频率上的权重来进行处理，比如特征提取，另外还可以过滤不必要的噪声。



如上图所示，原本的时间序列式图 a，而当图 a 中的序列加入噪声后（图 b），会产生非常复杂的时间序列。但是，通过傅里叶变化，我们可以发现，原本的时域空间中的时间序列在频域空间中的映射会非常突显原本的时间序列的形式。如上图所示，频率为 10 和 20 的波的权重非常非常大。在这种情况下，我们就可以忽略权重较小的频段的波，只选取权重大的频率的波，这样可以达到一种去除噪声的作用。

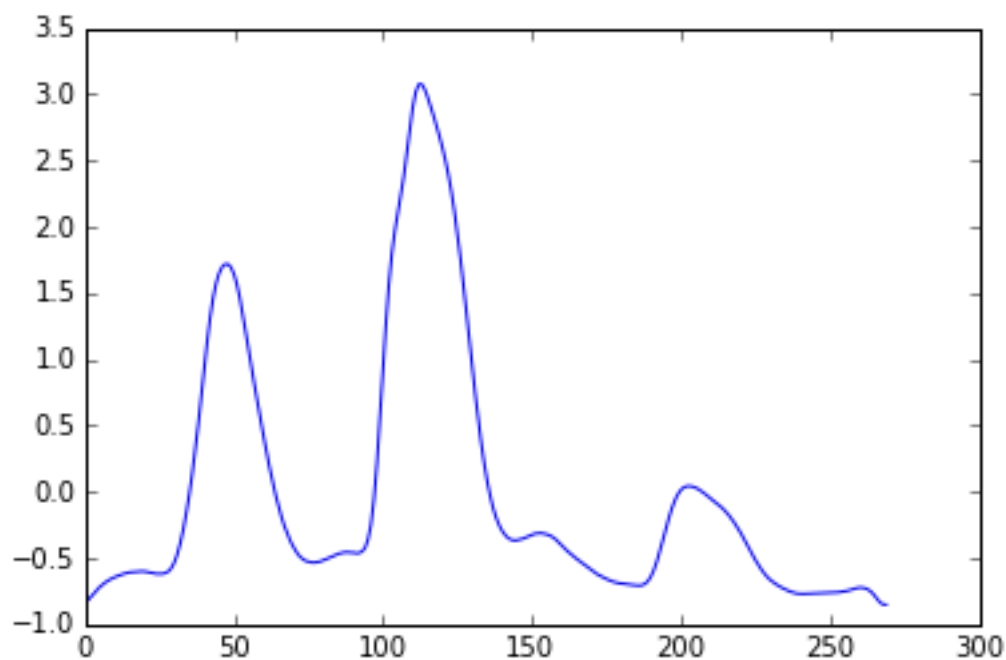
## 5.2 利用傅里叶变化处理时间序列

用 python 实现傅里叶变换:

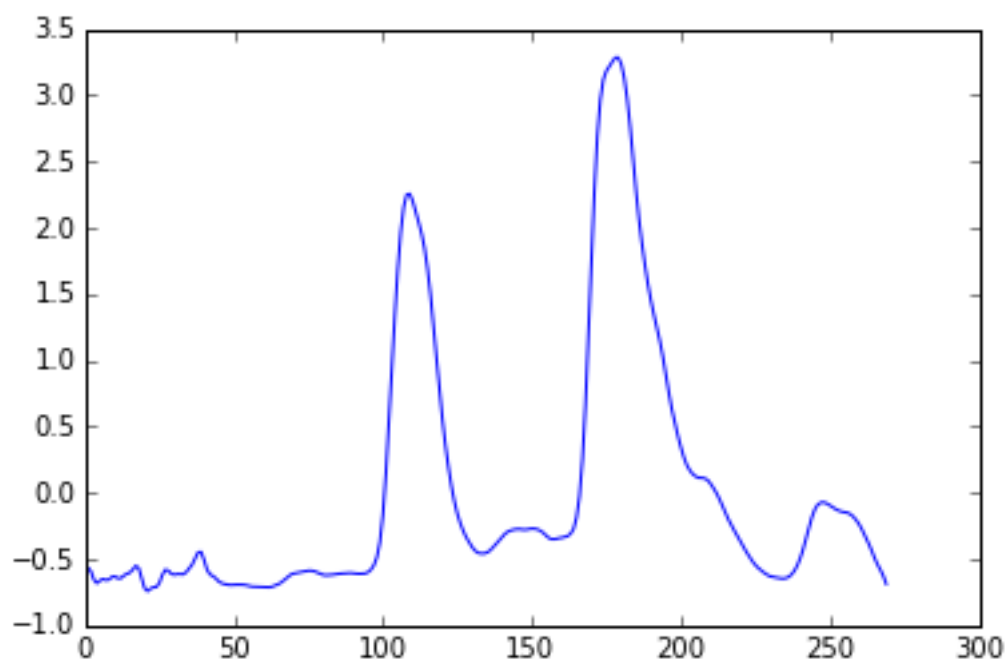
```
def FourierTransAndSaveTxt():
    list = GetFileList('E:\\DataMining\\Data_collection1', [])
    for e in list:
        f=open(e)
        res=[]
        Mx=[]
        lis=[]
        for line in f:
            lis=line.strip().split(',')
            cl=lis[0]
            lis=[float(i) for i in lis]
            res.append([cl])
            Mx.append(lis[1:])
        f.close()
        #print Mx
        dim=len(lis)-1
        j=0
        for i in Mx:
            waveone=np.array(i)
            print waveone.shape[1]
            x = np.linspace(0, 30, 30)
            plt.figure(1)
            plt.subplot(111)
            plt.plot(waveone)
            plt.show()
            Matrix=np.array(Mx)
            transWave=np.fft.fft(waveone)
            choseWave=transWave[0:21]
            choseWave=[float(i) for i in choseWave]
            res[j]+=choseWave
            #print transWave
            plot(transWave)
            show()
            j+=1
        # print res
        res=np.array(res)
```

Python 中的 `numpy` 库中的 `np.fft.fft` 可以将时域空间中的时间序列直接通过傅里叶变换得到频域空间中的数据。在得到的结果 `transWave` 中，我们可以得到原先波在各个频率上的权值。

### 5.3 傅里叶变换的优点



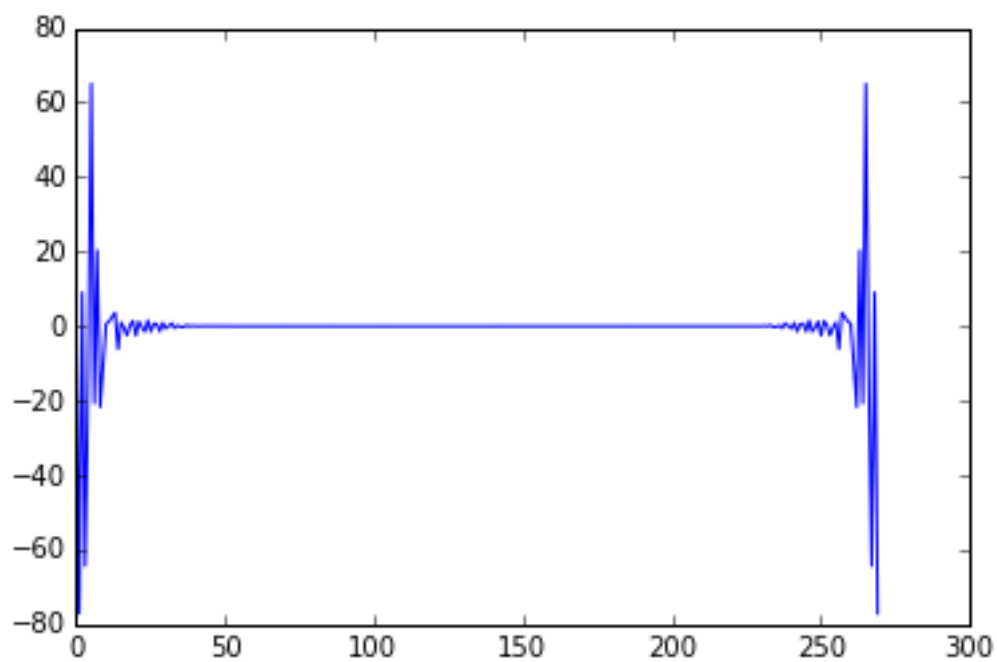
某数据集中的第 1 条时间序列（时域序列）



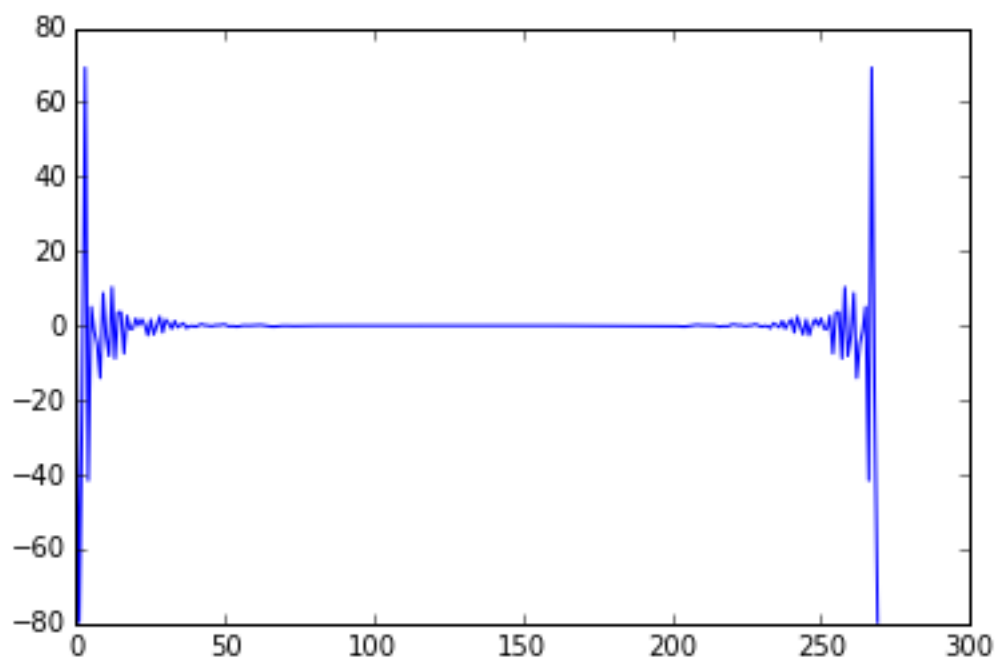
某个数据集中的第 2 条时间序列（时域序列）

上面两张图中我们可以发现，虽然两条序列都属于同一类，但是如果对每个时域上的维度进

行一一对比，两条时间序列有着很大的差距。



某数据集中的第 1 条时间序列（傅里叶变换后）



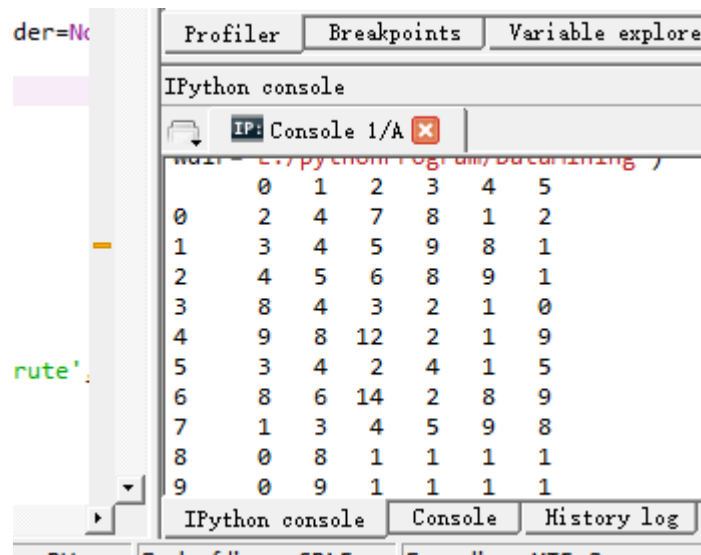
某个数据集中的第 2 条时间序列（傅里叶变换后）

上图如果使用了傅里叶变换，时域上的序列转换成了频域上的序列，两条时间序列的频域波形就可以发现是非常相似的。

因此，利用傅里叶变换可以消除因为时间序列的相位不同导致的聚类问题。

## 6 对于 Python 语言的学习:

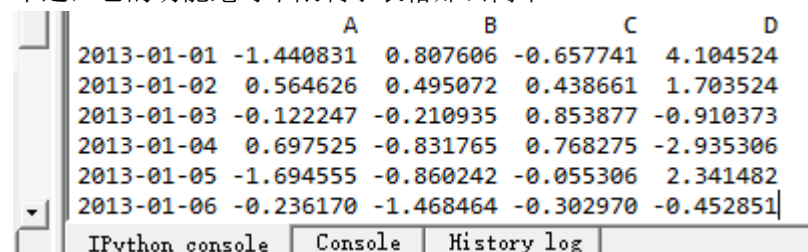
### 6.1 Pandas:大规模数据读取的必备数据结构



The screenshot shows an IPython console window with a DataFrame displayed. The DataFrame has 10 rows (indexed 0 to 9) and 6 columns (indexed 0 to 5). The data is as follows:

	0	1	2	3	4	5
0	2	4	7	8	1	2
1	3	4	5	9	8	1
2	4	5	6	8	9	1
3	8	4	3	2	1	0
4	9	8	12	2	1	9
5	3	4	2	4	1	5
6	8	6	14	2	8	9
7	1	3	4	5	9	8
8	0	8	1	1	1	1
9	0	9	1	1	1	1

如上图所示是利用 Python 中的 pandas 中的 DataFrame 功能对数据进行读取之后的结果。DataFrames 是一款非常强大的 pandas 数据结果。其本身的性质非常类似于一种二维表格，不过，它的功能绝对不限制于表格那么简单。



The screenshot shows an IPython console window with a DataFrame displayed. The DataFrame has 6 rows (indexed 0 to 5) and 4 columns labeled A, B, C, and D. The data is as follows:

	A	B	C	D
2013-01-01	-1.440831	0.807606	-0.657741	4.104524
2013-01-02	0.564626	0.495072	0.438661	1.703524
2013-01-03	-0.122247	-0.210935	0.853877	-0.910373
2013-01-04	0.697525	-0.831765	0.768275	-2.935306
2013-01-05	-1.694555	-0.860242	-0.055306	2.341482
2013-01-06	-0.236170	-1.468464	-0.302970	-0.452851

如上图所示，Pandas 的航标比仅仅可以用普通的数字表示，还可以使用日期来进行标识。

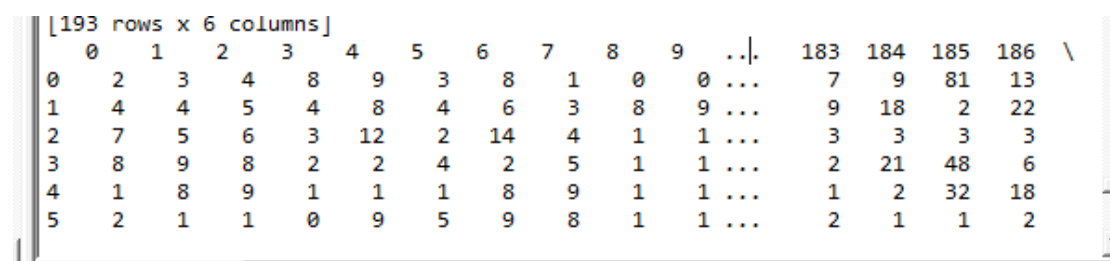
```
dtype: int64
```

DataFrame 显示的数据的都会具有一定的数据格式。

用

```
reader=reader.T
```

DataFrame 的数据结构如果调用.T 就会将原有的矩阵进行转置。



The screenshot shows an IPython console window with a transposed DataFrame displayed. The DataFrame has 193 rows and 6 columns. The first few rows are as follows:

	0	1	2	3	4	5	6	7	8	9	...	183	184	185	186	...
0	2	3	4	8	9	3	8	1	0	0	...	7	9	81	13	...
1	4	4	5	4	8	4	6	3	8	9	...	9	18	2	22	...
2	7	5	6	3	12	2	14	4	1	1	...	3	3	3	3	...
3	8	9	8	2	2	4	2	5	1	1	...	2	21	48	6	...
4	1	8	9	1	1	1	8	9	1	1	...	1	2	32	18	...
5	2	1	1	0	9	5	9	8	1	1	...	2	1	1	2	...

另外，DataFrame 还支持直接对数据进行图像的绘制。不过，最重要的是 DataFrame 类对大

规模数据的读取速度非常非常快。

## 6.2 Sklearn 库-最强大的 python 机器学习库



Sklearn 是所有使用 python 进行机器学习研究的人所比不可少的一个重要的代码库。在本次本学的学习当中，我们小组主要使用到了其中的两个库函数 k-means 和 DBSCAN。但是，sklearn 中的库函数远远不止于这些，就拿聚类算法举例：

Method name	Parameters	Scalability	Usecase
K-Means	number of clusters	Very large <code>n_samples</code> , medium <code>n_clusters</code> with <a href="#">MiniBatch code</a>	General-purpose, even cluster size, flat geometry, not too many clusters
Affinity propagation	damping, sample preference	Not scalable with <code>n_samples</code>	Many clusters, uneven cluster size, non-flat geometry
Mean-shift	bandwidth	Not scalable with <code>n_samples</code>	Many clusters, uneven cluster size, non-flat geometry
Spectral clustering	number of clusters	Medium <code>n_samples</code> , small <code>n_clusters</code>	Few clusters, even cluster size, non-flat geometry
Ward hierarchical clustering	number of clusters	Large <code>n_samples</code> and <code>n_clusters</code>	Many clusters, possibly connectivity constraints
Agglomerative clustering	number of clusters, linkage type, distance	Large <code>n_samples</code> and <code>n_clusters</code>	Many clusters, possibly connectivity constraints, non Euclidean distances
DBSCAN	neighborhood size	Very large <code>n_samples</code> , medium <code>n_clusters</code>	Non-flat geometry, uneven cluster sizes
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation
Birch	branching factor, threshold, ...	Large <code>n_clusters</code> and <code>n_samples</code>	Large dataset, outlier removal, data reduction.

图中所示的就是 sklean 库中的各种类型的聚类算法。使用 sklean 中的库函数可以较大程度减轻在机器学习代码编写过程中编写的代码的负担，因而可以在现有算法的基础之上对更新的算法进行研究。

```

def defCore(movieData):
    # movieData=pd.DataFrame(movieData)
    #print movieData
    i=0
    for point in movieData:
        count=0
        for point2 in movieData:
            if dis(point,point2)<=eps:
                count+=1
                if count>minpoint:
                    core.append(i);break
        i=i+1
    i=0
    for point in movieData:
        if i in core:
            continue
        j=0
        for point2 in movieData:
            if j in core:
                if dis(point,point2)<= eps:
                    boup.append(i)
            j+=1
        i+=1

    for i in range(len(movieData)):
        if i not in boup and i not in core:
            noise.append(i)
    #print noise

def dis(point,point2):
    distance=0
    for i in range(len(point)):
        distance+=(point[i]-point2[i])**2
    distance=distance**0.5
    return distance

def getCluster(movieData):
    #首先将第一个点初始化为第一个簇的起点
    clus=[[0]]
    i=1 #簇的下一个最大的标号
    s=0
    for num in core[1:]: #对于所有的点
        s=0
        for clusNum in range(len(clus)): #对于所有现在聚好的簇
            for pn in clus[clusNum]: #对于现在簇中的所有点
                if dis(movieData[num],movieData[pn])<=eps:
                    if(s==0):
                        clus[clusNum].append(num);s=1
            if s == 0:
                clus.append([num]) #如果都没有聚拢成功，那么添加一个新的簇
    print clus

```

上图是本小组原先实现 DBSCAN 算法的具体过程。

导入 sklearn 包后只需要一行代码即可完成 DBSCAN。如下图所示：

```

dbscan = cluster.DBSCAN(eps=0.3,min_samples=3,algorithm='brute',metric='euclidean')
dbscan.fit(random)

```

并且在 sklearn 中的 DBSCAN 算法的功能更加强大，不仅可以使使用普通的欧式距离，还可以使用例如余弦距离， 马氏距离，曼哈顿距离等等。另外，最让我感觉有用的是可以通过“precomputed”将距离设置成自己定义的类型。如下图所示：



```

def snn_sim_matrix(X, k=5):
    try:
        X = np.array(X)
    except:
        raise ValueError("输入的数据集必须为矩阵")
    #print X #没有类标的二维数据点
    samples_size, features_size = X.shape # 数据集样本的个数和特征
    nbrs = NearestNeighbors(n_neighbors=k, algorithm='kd_tree').fit(X)
    knn_matrix = nbrs.kneighbors(X, return_distance=False) # 记录
    #print "knn_matrix=%s"%knn_matrix
    sim_matrix = 0.5 + np.zeros((samples_size, samples_size)) # s
    #每个矩阵的初始相似度为0.5

    for i in range(samples_size):
        t = np.where(knn_matrix == i)[0]
        # print "t=%s"%t #t是将矩阵中的每一行取了出来
        c = list(combinations(t, 2))
        # print "c=%s"%c # 每行打包一下
        for j in c:
            if j[0] not in knn_matrix[j[1]]:
                continue
            sim_matrix[j[0]][j[1]] += 1
    # np.savetxt(r'F:\Programming\ImpaceAnalysis\jEdit\sim_matrix_s
    sim_matrix_d = 1 / sim_matrix #将相似度矩阵转化为距离矩阵
    sim_matrix_d = np.triu(sim_matrix_d)
    sim_matrix_d += sim_matrix_d.T - np.diag(sim_matrix_d.diagonal)
    np.savetxt(r'F:\Programming\ImpaceAnalysis\jEdit\sim_matrix_15
    return sim_matrix_d

```

上图所示是通过点和点之间共享最邻近点数 SNN 定义出来的 SNN 距离。使用当基于图中的 SNN 距离的 DBSCAN 时，我们可以通过如下实现：

```
db = DBSCAN(eps=epsilon, min_samples=min_points, metric='precomputed').fit(sim_matrix_d)
```

即可。

## 6.3 函数式编程，一种高效率 Python 编程方法

Lambda 函数：

lambda 的主体是一个表达式，而不是一个代码块。仅仅能在 lambda 表达式中封装有限的逻辑进去。该表达式是起到一个函数速写的作用。允许在代码内嵌入一个函数的定义。

```

IDLE 2.6
>>> f=lambda x,y,z:x+y+x
>>> f(1,2,3)
4
>>> f=lambda x,y,z:x+y+z
>>> f(1,2,3)
6
>>>
>>> n=5
>>> reduce(lambda x,y:x*y,range(1,n+1))
120
>>>

```

reduce()函数：

python 中的 reduce 内建函数是一个二元操作函数，他用来将一个数据集合（链表，元组等）中的所有数据进行下列操作：用传给 reduce 中的函数 func()（必须是一个二元操作函数）

先对集合中的第 1, 2 个数据进行操作, 得到的结果再与第三个数据用 `func()` 函数运算, 最后得到一个结果。

```
def myadd(x,y):  
    return x+y  
sum=reduce(myadd,(1,2,3,4,5,6,7))  
print sum  
sum=reduce(lambda x,y:x+y,(1,2,3,4,5,6,7))  
print sum
```

Filter 函数:

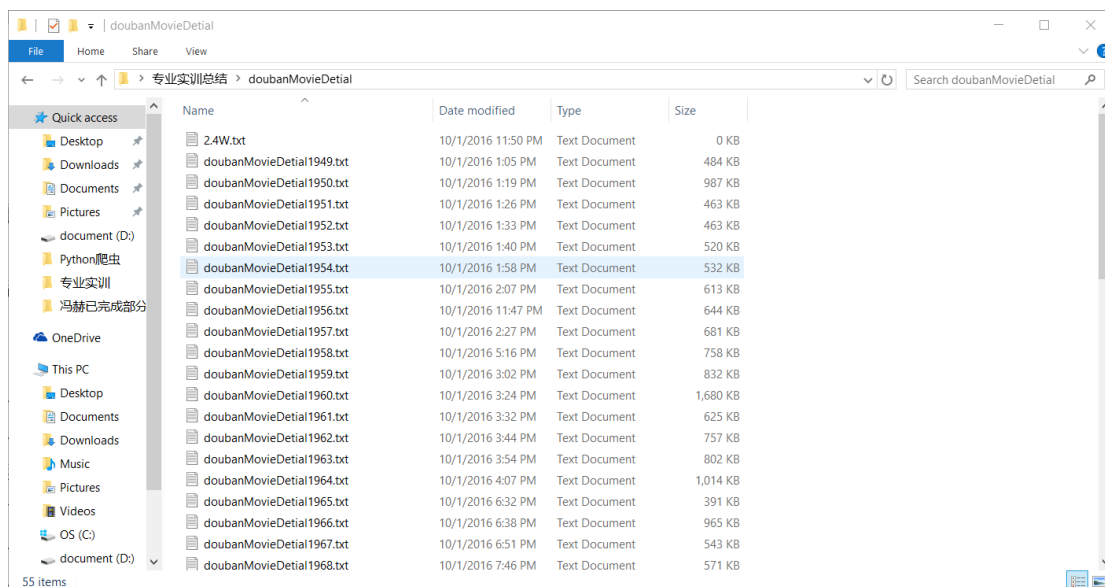
Filter 函数类似于一种过滤器函数, 其本质就是去除一个数据结构中符合要求的数据。

```
scores=[55,80,83,64,91,100,90,79]
```

```
def score_filter(score):  
    return score >=80 and score <90
```

## 7 Python 爬虫学习

### 7.1 结果截图:



爬虫的原理:

爬虫做的事情和浏览器（如果是为了抓取网页的话）是类似的。访问 <http://www.163.com> 这个网页，首先在浏览器输入这个 URL，URL 定位到这个需要访问的 html 文件（默认的是 index.html 文件），然后发出请求，获得请求，将获得的 html 文件进行解析展示在用户面前。

爬虫需要做的就是通过 URL 访问需要抓取的网页，获得返回的 html 文件。但是不需要解析。

爬虫除了抓取 html 文件，还需要分析网页的链接结构，通过循环的方式来抓取整个网站的内容（浏览器则不需要，用户需要访问那个 URL 标识的资源才会发出请求）

一个简易的爬虫:

```
import urllib2
content = urllib2.urlopen('http://www.163.com').read();
println(content);
```

1. python 内置的 urllib2 包可以通过 url 来抓取资源。
2. url 是 uri 的一种细化，子集。uri 可以认为就是用来唯一标识网络上资源的标号(主机号 + 路径 + 文件名称)，url 还需要标识出访问的方式，是 http，ftp 还是其他的应用层协议。
3. 通过 url 访问资源，返回的数据包通过 read() 来获得字符串。可以获得 html 文件，也可以是其他的。可以是 http 也可以是 ftp 的。
4. urlopen() 函数接受一个 URL 字符串或者一个 Request，通过传递 Request 可以实现更多的请求。

1. request = urllib2.Request(URL 字符串)。
2. request = urllib2.Request(URL 字符串, form 表单)。
3. request = urllib2.Request(URL 字符串, Header)。

可以模拟浏览器通过 URL 访问资源，浏览器通过 post 或者 get 发送 form 数据，浏览器发送自己的类型 (IE, Chrome 版本号)。

有的请求是将 form 传递给后台某个 php 文件，需要 post 或者 get 一个 form 给 php 文件。

添加 header 也可以通过 request 的 add\_header() 来实现。

```
import urllib
import urllib2

url = 'XXX';
values = {'name': 'fenghe',
          'grade': 60
        };
form = urllib.urlencode(values);
request = urllib.Request(url, form);
content = urllib2.urlopen(request, form).read();
```

有的网站可能需要识别浏览器的版本，python 爬虫默认的是

```
user_agent = 'xxx';
headers = { 'User-Agent' : user_agent };
...
request = urllib2.Request(url, form, headers);
```

使用 python 编写爬虫:

- 1.python 内置的 urllib2 包可以通过 url 来抓取资源。
- 2.通过 url 访问资源，返回的数据包通过 read()来获得字符串。可以获得 html 文件，也可以是其他的。可以是 http 也可以是 ftp 的。
- 3.urlopen()函数接受一个 URL 字符串或者一个 Request，通过传递 Request 可以实现更

多的请求。

4.可以模拟浏览器通过 URL 访问资源，浏览器通过 post 或者 get 发送 form 数据，浏览器发送自己的类型（IE,Chrome 版本号）

5.添加 header 也可以通过 request 的 add\_header()来实现。

6.有的网站可能需要识别浏览器的版本，python 爬虫默认的是 Python-urllib/x.y。可以伪装成 IE, Chrome。

## 7.2 使用豆瓣 API 获得豆瓣电影数据

Python 可以很方便的实现爬虫。爬虫模拟浏览器，通过循环来抓取网页。但是，似乎也没有多么神奇，目前无法分析一个网页里面有哪些超链接，如果是抓取一个网页之后，通过解析 html 找到所有超链接的话，似乎可以实现。但是目前的抓取方式是找到网页之间的关联，比如贴吧的地址，第 1 页和第 n 页之间只有一个数字的差别，并且是满足倍数关系的，所以直接循环即可。

豆瓣提供了 API 来访问豆瓣的数据，通过豆瓣账号登陆，获得豆瓣的数据，这些都提供有 API，但是目前只对企业开发，所以个人无法获得 APIKey。

就算有 APIKey，对访问频率也有要求，每分钟不超过 50 次，如果没有，每分钟不超过 10 次。

通过访问豆瓣提供的 API（GET URL），豆瓣返回 JSON 格式的数据，比如电影的详细信息。

豆瓣上的电影页面，都是 <https://movie.douban.com/subject/id/> 格式的 url。（如果是通过某个标签进入的话，后面会加入其他的）问题是 id 的数目过于庞大，id 为 8 位，至少有 300W 中可能。并且豆瓣电影的 id 不是连续的，而且找不到规律。所以，从 00000001 到 30000000 挨着试不可能，豆瓣似乎还会检测同一个 IP 的访问是否频率过快。

通过 API 访问，还是需要 ID，我们并不知道电影和 ID 之间的关系，还是需要一个一个试，显然不合适大规模的数据抓取。

id 和年份，种类无关。都是 8 位 id。

通过年份 TAG 抓取所有电影的纵览之后，解析里面的超链接，即可获得对应 ID。

1. 通过年份 TAG 抓取所有的页面，注意设置时间间隔，每分钟 10 个就好。
2. 抓取页面后，对所有页面中电影的超链接进行解析，提取后，再提取里面的

id, 获得所有电影的 id。

3. 有了 ID 后, 通过豆瓣 API 获取所有电影的 JSON 信息。

大概步骤:

1. 根据豆瓣网提供的, 根据 Tag 查找电影的功能, 将所有 Tag = 年份的网页都爬下来。start 总是 20 的倍数, 不同的年份也只是更改 tag 后的文件夹名称。所以, 需要做的事情和扒取贴吧网页一样了, 注意豆瓣检测 IP 访问的频率, 所以最好一分钟不要超过 10 次。年份是固定的, 写成 for 循环即可, 每个年份内部的电影数目未知, while 循环直到返回 status 为 404 为止。

2. 扒取所有 Tag 为年份的网页后, 通过 HTML parse 来提取出所有 class 为 nbg 的 tag 里面的超链接。这样, 获得豆瓣网所有电影的 id。

3. 获得 id 后, 通过豆瓣网提供的 API 直接访问对应的 URI, 电影的信息会以 JSON 的格式返回。但是频率有限制, 每分钟不超过 50 次。

扒取单个年份网页并通过 BeautifulSoup4 来 parseHTML 的代码:

```
import string
import urllib2
from bs4 import BeautifulSoup

content =
urllib2.urlopen(u'https://movie.douban.com/tag/2016?start=0&type=T').
read();
f = open('douban0ID.txt', 'w+')

soup = BeautifulSoup(content, 'html.parser')

for link in soup.find_all('a', class_='nbg'):
    resultTemp = string.replace(link.get('href'),
    'https://movie.douban.com/subject/', '')
    result = string.replace(resultTemp, '/', '')
    result = result + ' '
    print(result)
    f.write(result)

f.close()
```

1. bs4 很强大啊, HTML 和 XML 都可以 parse。

2. bs4parseHTML, 可以直接通过 TAG 操作, 可以通过 ID, 可以通过 CLASS, 获取所有超链接也是直接提供的功能, 所以 soup.find\_all('a', class\_='nbg') 直接搞定。

通过 ID 获得电影数据:

```
url1 = u'http://api.douban.com/v2/movie/subject/25986180'
content = urllib2.urlopen(url1).read()
```

1. 豆瓣网专门的 API 获得 movie, book... 的数据, 访问 `http://api.douban.com/v2/XXX/XXX/ID` 来获得返回的 JSON 数据。
2. movie 可以替换成 book 或者其他。
3. subject 表示获得基本信息, 也可以获得其他信息, 具体参见 DOC。

处理返回的 JSON 数据:

1. JSON: 一种从 JS 的对象字面量表示法中启发得到的轻量级数据传输格式, 其实功能和 XML 一样。
2. JSON 分为 {} 和 [] 两种格式。前一种就是 C++ 里的 struct, js 里的 object, python 里的 dictionary, lua 里的 map, C# 里的关联数组, 就是键值对的集合, 键只能是字符串, 值可以是任何数据类型, 包括 {} 和 []。后一种就是数组, 数组在 lua 这些脚本语言中是作为一种特殊的 map 处理的, 只不过值必须是连续的脚标值。
3. JSON 的 encode 过程表示将 JSON (map, array...) dump 成对应的 string 类型, 因为数据传输必须是 string。
4. JSON 的 decode 过程表示将 string 类型转换成语言对应的数据结构 (不同语言有不同的数据结构, js 就是获得 object, python 就是获得 dictionary)。
5. 通过访问豆瓣 API 获得就是 string 存储的 json 数据。需要通过 decode 编程 python 中的 dictionary。

python 语言处理 json:

python 有专门的 json 包来处理 json 数据。

1. encode: `json.dumps(dictionary/array)`。返回 string 类型的数据。
2. decode: `json.loads(string)`。返回 dictionary 或者 list。

代码实现:

```
doubanMovieJson = json.loads(content)
print(doubanMovieJson['summary'])
//通过返回的 dictionary 直接访问需要的数据。
```

获取豆瓣电影的 ID 列表:

豆瓣反爬虫, 但是提供豆瓣 API, 通过 API 可以访问豆瓣上电影的信息。但是 id 是未知的, 需要首先获取所有电影的 ID 信息。(豆瓣电影的 ID 有 8 位和 9 位的, 没有规律, 并且爬虫限制频率, 所以遍历不可行。通过扒取 TAG 下所有网页, parse 出 ID 的方式可以获得 ID 列表)

```
import string
import urllib2
from bs4 import BeautifulSoup #python 一个很好用的 HTML/XML
parse 库。
```

import time #爬虫需要限制频率，豆瓣会检测 IP 的访问频率，过快会封 IP。

```
f = open('doubanMovieID.txt', 'w+')

i = 1949

while(i < 2017):
    urlToOpen0 = 'https://movie.douban.com/tag/'
    urlToOpen1 = str(i)
    urlToOpen2 = '?start='
    urlToOpen3 = 0
    urlToOpen4 = '&type=T'
    index = 0
    #这里做的事情，和扒取百度贴吧是一样的。

    f.write(str(i) + " : ")
    while (True):
        #获取网页。
        urlToOpen3 = 20 * index
        content = urllib2.urlopen(urlToOpen0 +
urlToOpen1 + urlToOpen2 + str(urlToOpen3) + urlToOpen4).read()

        #通过 bs4parse 出 ID 信息。
        soup = BeautifulSoup(content, 'html.parser')
        hrefList = soup.find_all('a', class_='nbg')
# 'nbg' 这个 class 正好对应所有链接。
        if(hrefList):
            for link in soup.find_all('a',
class_='nbg'):
                resultTemp =
string.replace(link.get('href'), 'https://movie.douban.com/subject/',
'')
                result =
string.replace(resultTemp, '/', '')
                result = result + ' '
                print(result)
                f.write(result)
            time.sleep(5)
            index = index + 1
        else:
            i = i + 1
            break

f.close()
```



记录:

1. 最后一共扒取了大概 4.5W 到 5W 个 id 数据。写到了 txt 中，用空格分开，方便后期直接 split 变成 list。
2. 豆瓣通过年份分类的电影信息存在错误，有 1950 年的电影被分配到 1949 年的情况。所以 ID 是会出现重复的。
3. 豆瓣通过年份分类页面访问所有电影信息是不完全的。通过网页下方的标签来访问只能访问这一年的部分电影。似乎是存在 bug，比如一个页面显示只有 1-25 个标签。但是通过 URL 中的 start = XXX，可以访问到第 26 个页面，同时开启隐藏的 26-50 页面(\_\_\_\_)b。
4. 扒取的过程中，本来打算通过 404 来判断这一年已经扒取完了，但是豆瓣的设计是就算是 start 超过了，也不会返回 404. 所以通过判断解析出的 list 是否为空来判断是否扒取完了。但是存在扒取中途突然出现 null 异常的问题。导致这一年的扒取不完。不清楚是什么问题。
5. 代码没有做异常处理，有的时候会出现 URLError。

根据 ID 获取电影信息:

上一步只扒取了部分的 id 信息，一方面是因为只设置了 1949-2016. 另一方面，上面的代码存在 bug，有的时候会出现没有异常，但是就是无法继续扒取下去的问题。

有了 ID 之后，split 处理成 list。遍历所有 list 通过豆瓣 API 获取电影条目信息。

```
import string
import urllib2 #get 请求不需要使用 httplib, urllib2 内部也是使用 httplib 实现的。
import time #访问限制为 40 次/min。等待 2 秒钟。

itemCount = 0

for i in range(1949, 2017):
    fIdName = 'doubanMovieId/doubanMovieID' + str(i) +
'.txt' #读取存储了 id 的文件。
    fId = open(fIdName, 'r').read()
    fIdStringArray = fId.split(' ') #获得 ID list。
    del fIdStringArray[len(fIdStringArray) - 1] #最后会多
出一个空格。去掉它。

    fDetial = open('doubanMovieDetial/doubanMovieDetial'
+ str(i) + '.txt', 'w+')
    for i in fIdStringArray:
        urlHeader =
'http://api.douban.com/v2/movie/subject/' #豆瓣 API 前缀。获得 subject
里的信息。(subject 是电影条目信息，还有 celebrity，这个是电影人信息)
        urlToOpen = urlHeader + i
```

```

        try:
            movieDetail = urllib2.urlopen(urlToOpen).read()
            fDetail.write(movieDetail + '\n')
            itemCount = itemCount + 1
            print 'get ' + str(itemCount) + 'th
itme!'

        except URLError, e:
            print 'There is an error!'
            time.sleep(10000)

        finally:
            time.sleep(2)

    fDetail.close()

```

记录:

1. 对抓取下来的数据不是很满意，因为豆瓣 subject 中有的信息，比如演员和导演，是只给了 id 的，还需要在抓取演员的信息。
2. 和抓取 id 的时候一样，使用 API 的时候有的时候会出现就是获取不到下一条数据，然后程序就停在那里。不报错，也不说出现异常。所以，没有把所有 id 放在一起，而是分块的进行抓取。难道说抓取 40 分钟就需要休息一下？
3. 通过 subject 可以获得的，感觉有用的信息（通过用户的 query 进行匹配的数据）：评分，想看人数，在看人数，收藏人数，条目分类，国家，日期，年代。
4. 演员，导演这个只有 id，需要另外抓取。
5. 抓取的数据通过 string 存储在 txt 中，读取之后，通过'\n'来 split, loads 之后获得对应的 list。

小结:

1. 豆瓣对于爬虫，似乎会检测 IP 的访问频率是否过高。
2. 豆瓣提供了 API 来访问豆瓣的数据。通过豆瓣账号登陆，获得豆瓣的数据，这些都提供有 API，但是目前只对企业开发，所以个人无法获得 APIKey。
3. 就算有 APIKey，对访问频率也有要求，每分钟不超过 50 次，如果没有，每分钟不超过 10 次。
4. 通过访问豆瓣提供的 API (GET URL)，豆瓣返回 JSON 格式的数据，比如电影的详细信息。
5. 豆瓣上的电影页面，都是 <https://movie.douban.com/subject/id/> 格式的 url。（如果是通过某个标签进入的话，后面会加入其他的）问题是 id 的数目过于庞大，id 为 8 位，至少有 300W 中可能。并且豆瓣电影的 id 不是连续的，而且找不到规律。所以，从 00000001 到 30000000 挨着试不可能，豆瓣似乎还会检测同一个 IP 的访问是否频率过快。
6. 不使用 id 访问每个电影的明细页面，通过 TAG 来抓取数据，为了抓取所有的电影，最好使用年份 TAG 来抓取。
7. 通过年份 TAG 抓取所有电影的纵览之后，解析里面的超链接，即可获得对应 ID。
8. 通过年份 TAG 抓取所有的页面，注意设置时间间隔，每分钟 10 个就好。
9. 抓取页面后，对所有页面中电影的超链接进行解析，提取后，再提取里面的 id，获得所有电影的 id。

- 10.有了 ID 后，通过豆瓣 API 获取所有电影的 JSON 信息。
- 11.最后总共扒取了豆瓣网上 2.4W 条电影的信息，应付专业实训估计足够了。
- 12.本次扒取因为豆瓣网虽然提供了 API，但是限制 API 频率，而且当访问总次数很高的时候，也会被限制。所以开了 4 台电脑。感觉就像手动分布式一样。分布式可以提高效率，毕竟同时开了好几台电脑。
- 13.豆瓣网会检测 IP 的访问次数，所以这次准备了 6 个 IP。每个 IP 最多扒取的数据个数在 2000 到 7000 左右。之后就会 403。
- 14.添加 header 的方法不可行，直接用 fiddler 截包获得的 header 也不行。
- 15.如果一个 IP 被封了，那过一段时间就可以恢复，但是这次一般 200 左右就会再次被封。

## 7.3 遇到的问题：

- 1.豆瓣通过年份分类的电影信息存在错误，有 1950 年的电影被分配到 1949 年的情况。所以 ID 是会出现重复的。
- 2.豆瓣通过年份分类页面访问所有电影信息是不完全的。通过网页下方的标签来访问只能访问这一年的部分电影。似乎是存在 bug，比如一个页面显示只有 1-25 个标签。但是通过 URL 中的 start = XXX，可以访问到第 26 个页面，同时开启隐藏的 26-50 页面(-\_\_-)b。
- 3.扒取的过程中，本来打算通过 404 来判断这一年已经扒取完了，但是豆瓣的设计是就算是 start 超过了，也不会返回 404.所以通过判断解析出的 list 是否为空来判断是否扒取完了。但是存在扒取中途突然出现 null 异常的问题。导致这一年的扒取不完。不清楚是什么问题。
- 4.对扒取下来的数据不是很满意，因为豆瓣 subject 中有的信息，比如演员和导演，是只给了 id 的，还需要在扒取演员的信息。
- 5.和扒取 id 的时候一样，使用 API 的时候有的时候会出现就是获取不到下一条数据，然后程序就停在那里。不报错，也不说出现异常。所以，没有把所有 id 放在一起，而是分块的进行扒取。难道说扒取 40 分钟就需要休息一下？
- 6.通过 subject 可以获得的，感觉有用的信息（通过用户的 query 进行匹配的数据）：评分，想看人数，在看人数，收藏人数，条目分类，国家，日期，年代。
- 7.演员，导演这个只有 id，需要另外扒取。
- 8.扒取的数据通过 string 存储在 txt 中，读取之后，通过'\n'来 spilt，loads 之后获得对应的 list。
- 1.?看来就算是用 API 访问，如果超过次数，或者访问次数很多。那还是会被封掉。
- 2.?可能需要对 header 进行模拟或者需要模拟登陆。（浏览器提示该网站可能需要登录。）
- 3.?添加 headers 的中 user-agent 是不行的。
- 4.?看来 IP 直接被封了。使用 API 还是不行，可能豆瓣提供 API 并不是给爬虫用的。链接不同的网络，更换 IP 吧。

## 7.4 总结:

- 1.最后总共扒取了豆瓣网上 2.4W 条电影的信息,估计足够了。
- 2.本次扒取因为豆瓣网虽然提供了 API,但是限制 API 频率,而且当访问总次数很高的时候,也会被限制。所以开了 4 台电脑。感觉就像手动分布式一样。分布式可以提高效率,毕竟同时开了好几台电脑。
- 3.豆瓣网会检测 IP 的访问次数,所以这次准备了 6 个 IP。每个 IP 最多扒取的数据个数在 2000 到 7000 左右。之后就会 403。
- 4.添加 header 的方法不可行,直接用 fiddler 截包获得的 header 也不行。
- 5.如果一个 IP 被封了,那过一段时间就可以恢复,但是这次一般 200 左右就会再次被封。

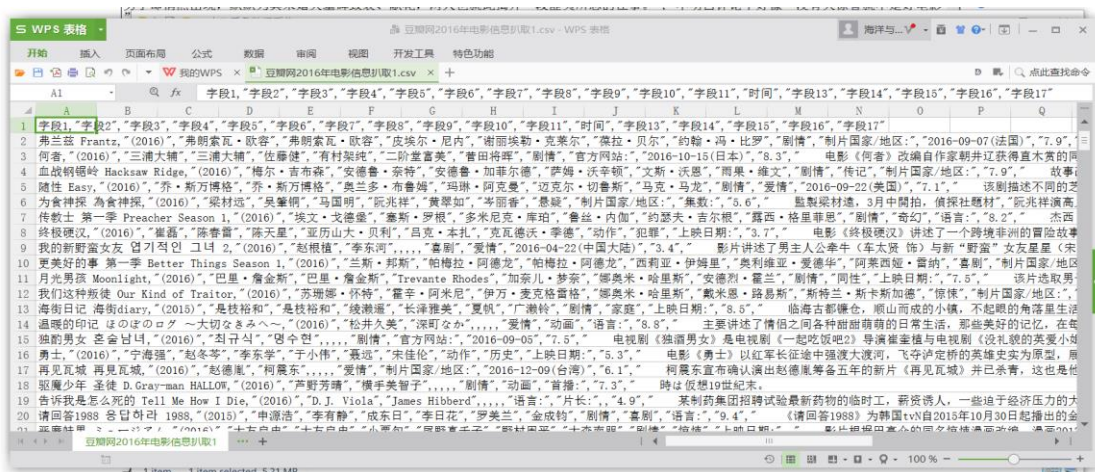
## 7.5 参考资料:

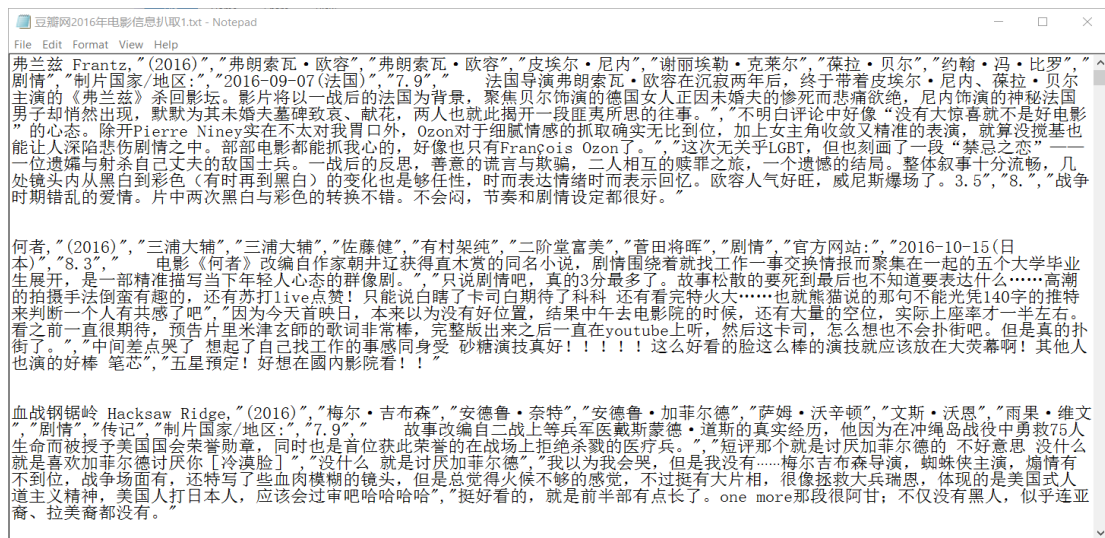
- 1.<http://blog.csdn.net/pleasecallmewhy/article/details/8927832>(汪海的实验室讲解使用 urllib2 编写爬虫的部分)
- 2.[https://developers.douban.com/wiki/?title=api\\_v2](https://developers.douban.com/wiki/?title=api_v2)([https://developers.douban.com/wiki/?title=api\\_v2](https://developers.douban.com/wiki/?title=api_v2) (豆瓣 API 官方手册))
- 3.<http://www.chinaz.com/web/2016/0909/578949.shtml> (应对反爬虫的一些措施)

# 8 使用八爪鱼扒取数据学习

## 8.1 结果截图:

使用八爪鱼扒取豆瓣网电影信息





## 8.2 八爪鱼各个功能学习：

之前扒取的是豆瓣网上的电影的详情信息，感觉数据收集的工作真实蛋疼的可以，首先是数据的来源，网页上的内容可以使用爬虫，但是很多网站都反爬虫，动不动就 403 了，后来发现八爪鱼真是个好东西啊。

扒取下来的数据还一般不能直接使用，还必须要做处理，比如删除掉文本当中不需要的字符，使用正则表达式可以很好的完成。

数据的预处理部分，自然语言的分词工作一般都有现成的包可以使用，比如 Python 的 jieba 包。

用户登录：

八爪鱼提供了屏幕动作录制的功能。

有的网页在打开的时候要求登陆，在八爪鱼里面很简单，在扒取之前打开登陆的页面，录制下来登陆的过程即可（点击用户名输入框，输入文本，点击密码输入框，输入文本，点击登陆按钮）

登陆之后，再去打开其他网页即可。

登陆在 cookie 里面会保存下来，所以每次登陆之后为了执行登陆脚本的时候不会因为 cookie 自动登陆，需要点击八爪鱼浏览器那里的清空浏览器缓存。

循环翻页+进入详细页面+动作录制+循环翻页+进入详细页面+数据扒取：

单个的循环翻页+详细页面+数据扒取可以直接使用八爪鱼提高的任务来实现，但是经常遇到需要执行两次循环的操作。需要手动实现。

八爪鱼有的时候还挺难用的，主要是不熟悉该怎么操作，经常会弹出对话框说出现异常，是否自动恢复，也不用恢复，只要弹出这个对话框，就跑不了了。

注意：这种需要弄两次的情况，首先先完成第一次，注意，先不要把进入详



细页面和抓取数据的那部分循环操作放入到翻页的循环中，需要接下来完成第二次操作，然后把第二次操作中进入详细页面和抓取数据的那部分循环放入到第二次的循环翻页内部，然后那整个第二次循环放入到第一次的动作录制的后面然后再放入到第一次循环翻页的内部。注意顺序，否则八爪鱼不识别这样的操作。

这里的操作首先需要看看提供的有关翻页+进入详细页面+数据抓取的那个教程。

条件分支的使用：

有的电影根本没有分类的标签，抓取的时候会抓取下来一个无意义的文本信息。

那个无意义的，我们做预处理可以去掉，但是不如直接在八爪鱼中判断是否存在类别的信息，然后再决定是否要抓取信息。

可以首先判断是否有“类型：”这个信息，然后再抓取数据，八爪鱼有个XPath的概念，指的是每个HTML的DOM中的某个标签，每次抓取数据的时候，就是根据这个XPath来抓取的数据。//DIV[@id='info']/SPAN[11]这个就是一个XPath。

查看网页的代码可以发现，显示信息的那部分放在id为DIV的里面，通过id可以唯一的定位这个DIV，但是SPAN因为没有使用ID，只是简单的罗列信息，没有唯一定位的方法，而八爪鱼是通过XPath来抓取文本的，八爪鱼只认//DIV[@id='info']/SPAN[11]这个位置的数据，而无法判断这个位置是不是类别的信息。所以之前抓取的时候，当没有类别这个Span的时候，就会抓取到“地区：”这个信息。

这就导致，条件分支中，无法通过判断某个元素是否存在的方式来判断是否抓取数据。

那就通过“判断文本是否存在”来判断是否抓取数据。判断“类型：”这个文本是否存在。

如果分支结构放在循环结构内部，会出现本次循环中是否出现那个文本，注意，这个不是检查网页文本的。

XPath的问题：

每次通过八爪鱼抓取的数据，数据会有一个XPath表示他的位置，//DIV[@id='info']/SPAN[11]，可以发现，Span的定位只是找到第11个span而已，因为不是ID的精确定位，导致如果两个网页的格式不一样，八爪鱼就会抓取下来错误的数据。比如综艺节目也被豆瓣当作电影给放到用户看过的电影中去了。电影和综艺节目两种格式的网页不一样，同样的XPath表示了不同的信息。

问题在于，如何在循环中提取不规则位置的数据！XPath的理解有误，这个东西和正则一样，是一种专门的工具语言，可以用来定位XML文件中元素。八爪鱼内部有一套XPath的引擎，使得XPath能够支持对HTML文件的查询。XPath能很好的解决循环提取数据的过程中，数据位置不规则的问题。

并不是某些电影没有类型这个信息，而是因为电影和真人秀都被认为是电影，但是两个网页的格式是不一致的，而八爪鱼内部是使用XPath来抓取的数据，豆瓣网在显示信息的时候，每个信息条目是一个span，并且没使用id，所以导致当span的顺序换了一下之后就会抓取到错误的数据。

## 遇到的问题:

1. 八爪鱼有的时候还挺难用的，主要是不熟悉该怎么操作，经常会弹出对话框说出现异常，是否自动恢复，也不用恢复，只要弹出这个对话框，就跑不了了。这种需要弄两次的情况，首先先完成第一次，注意，先不要把进入详细页面和抓取数据的那部分循环操作放入到翻页的循环中，需要接下来完成第二次操作，然后把第二次操作中进入详细页面和抓取数据的那部分循环放入到第二次的循环翻页内部，然后那整个第二次循环放入到第一次的动作录制的后面然后再放入到第一次循环翻页的内部。注意顺序，否则八爪鱼不识别这样的操作。
2. 有的电影根本没有分类的标签，抓取的时候会抓取下来一个无意义的文本信息。那个无意义的，我们做预处理可以去掉，但是不如直接在八爪鱼中判断是否存在类别的信息，然后再决定是否要抓取信息。可以首先判断是否有“类型：”这个信息，然后再抓取数据，八爪鱼有个 XPath 的概念，指的是每个 HTML 的 DOM 中的某个标签，每次抓取数据的时候，就是根据这个 XPath 来抓取的数据。//DIV[@id='info']/SPAN[11] 这个就是一个 XPath。查看网页的代码可以发现，显示信息的那部分放在 id 为 DIV 的里面，通过 id 可以唯一的定位这个 DIV，但是 SPAN 因为没有使用 ID，只是简单的罗列信息，没有唯一定位的方法，而八爪鱼是通过 XPath 来抓取文本的，八爪鱼只认 //DIV[@id='info']/SPAN[11] 这个位置的数据，而无法判断这个位置是不是类别的信息。所以之前抓取的时候，当没有类别这个 Span 的时候，就会抓取到“地区：”这个信息。这就导致，条件分支中，无法通过判断某个元素是否存在的方式来判断是否抓取数据。那就通过“判断文本是否存在”来判断是否抓取数据。判断“类型：”这个文本是否存在。如果分支结构放在循环结构内部，会出现本次循环中是否出现那个文本，注意，这个不是检查网页文本的。
3. 每次通过八爪鱼抓取的数据，数据会有一个 XPath 表示他的位置，//DIV[@id='info']/SPAN[11]，可以发现，Span 的定位只是找到第 11 个 span 而已，因为不是 ID 的精确定位，导致如果两个网页的格式不一样，八爪鱼就会抓取下来错误的数据。比如综艺节目也被豆瓣当作电影给放到用户看过的电影中去了。电影和综艺节目两种格式的网页不一样，同样的 XPath 表示了不同的信息。问题在于，如何在循环中提取不规则位置的数据！XPath 的理解有误，这个东西和正则一样，是一种专门的工具语言，可以用来定位 XML 文件中元素。八爪鱼内部有一套 XPath 的引擎，使得 XPath 能够支持对 HTML 文件的查询。XPath 能很好的解决循环提取数据的过程中，数据位置不规则的问题。并不是某些电影没有类型这个信息，而是因为电影和真人秀都被认为是电影，但是两个网页的格式是不一致的，而八爪鱼内部是使用 XPath 来抓取的数据，豆瓣网在显示信息的时候，每个信息条目是一个 span，并且没使用 id，所以导致当 span 的顺序换了一下之后就会抓取到错误的数据。
4. 一部电影在豆瓣网上可能会同时有多个分类，但是这么弄就只抓取了第一个。  
\*\*//span[@property='v:genre']/following-sibling::span[1] 可以抓取下一个兄弟节点，但是我们不知道一个电影总共有几个类标。

## 8.3 参考资料:

<http://bbs.bazhuayu.com/> （八爪鱼官方论坛和问题集结）

<http://www.bazhuayu.com/tutorial/dgwbhdl.aspx?t=0> （八爪鱼官方手册）

<http://www.w3school.com.cn/xpath/> （学习 xpath 手册）

# 9 数据处理

## 9.1 结果截图：

处理之后的数据





## 9.2 Python2.7 读取中文文本文件的问题：

通过八爪鱼扒取下来的数据可以直接保存为 txt 文件。通过 python 的文件操作读取的时候，会读取错误的数据。检查后发现，python 原生的文件操作模块并只支持 ascii 的编码方式。对于 utf 的编码方式需要使用 Python 的 codecs 包来处理。

```
#Python 处理 UTF 编码文件的包
```

```
import codecs
```

```
#和 ascii 的操作类似。第三个参数表示文本的编码方式。
```

```
#文本的编码方式有 utf-8，utf-16 等。编码方式错误会导致出错。
```

```
#windows 的记事本。默认使用 unicode 编码。也就是 utf-16。
```

```
#windows 的记事本，对于没有中文的，默认使用 ANSI 编码。
```

```
f = codecs.open('content.txt', 'r', 'utf-16')
```

```
#读取一行。（根据回车来读取）
```

```
fContent = f.readline()
```

```
print fContent
```

```
f.close()
```

## 9.3 使用 jieba 包对中文进行分词处理：

jieba 是一个很好用的 Python 中文分词处理包。

```
import string

import codecs

import jieba

f = codecs.open('result.txt', 'r', 'utf-16')

fSeg = codecs.open('seg.txt', 'w', 'utf-16')

for line in f:

    seg_list = jieba.cut(line)

    stringTemp = ''

    for i in seg_list:

        stringTemp += i

    stringTemp += ' '

    fSeg.write(stringTemp)

fSeg.close()

f.close()
```

## 8.4 正则表达式去除标点符号：

Python 正则表达式：

1. 正则表达式是一套独立的体系。不同的语言都可能支持正则表达式（Python 使用 re 模块）。正则表达式是一门用来进行字符串匹配的语言。
2. 正则表达式有自己的语法，通过正则表达式引擎将正则表达式编译成用来描述匹配规则的正则表达式对象，通过对文本进行匹配获得结果。
3. 正则表达式的效率不如 string 里面的函数效率高。但是功能强大。
4. 正则表达式是用来进行字符串内容匹配的。

5. 正则表达式写出来就是一个字符串。Python 有两个方便的地方，r 打头的字符串，r''，里面不会进行转义。'' 和 "" 都可以表示字符串。
6. 正则表达式也需要确定编码方式。

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

import string
import codecs
import re

f = codecs.open('seg.txt', 'r', 'UTF-8')
content = f.readline()
print re.sub('兰', 'l', content)

f.close()
```

1. 最上面的魔法注释，作用只是表示这个程序的源文件是 UTF-8 的编码方式存储的。并不会影响里面的操作。比如文件读取依然是 ansi 的格式。
2. print re.sub('兰', 'l', content)。正则表达式需要对 UTF-8 编码的字符串进行匹配。正则表达式也需要进行编码格式的确定：re.sub('兰'.decode('UTF-8'), 'l'.decode('UTF-8'), content)
3. '.' :任意的字符串
4. '[abc]': [] 里面可以定义一个字符集。只要是这个字符集里的字符都会被匹配到。

小结：

1. 正则表达式是一套独立的体系。不同的语言都可能支持正则表达式(Python 使用 re 模块)。正则表达式是一门用来进行字符串匹配的语言。
2. 正则表达式有自己的语法，通过正则表达式引擎将正则表达式编译成用来描述匹配规则的正则表达式对象，通过对文本进行匹配获得结果。
3. 正则表达式的效率不如 string 里面的函数效率高。但是功能强大。
4. 正则表达式是用来进行字符串内容匹配的。
5. 正则表达式写出来就是一个字符串。Python 有两个方便的地方，r 打头的字符串，r"，里面不会进行转义。"和""都可以表示字符串。

## 9.4 提取所有的电影名称：

cvs 可以直接转换成 xls 格式的文件。创建空的 xls，数据，从外部导入数据，选择 cvs 文件

## 9.5 去除数据里的换行符：

一开始打算首先将文件内容读成字符串，然后用'\n'进行 split。然后 list 进行 remove('\n')。问题：那些值似乎并不是'\n'啊。是一个特殊的值。split('\n')的确可以把内容开。但是之前看到的那些空行并不是'\n'，那些换行符一样的东西，好像是空格和 TAB 的组合啊。这两个条目里面看起来好像是第一个条目写完之后，换了两次行。但实际上是一个条目结束后，后面用空格和那个两个空格的特殊字符的组合来形成了一个行。然后再换行最后发现直接把这个特殊的字符给复制过来了。

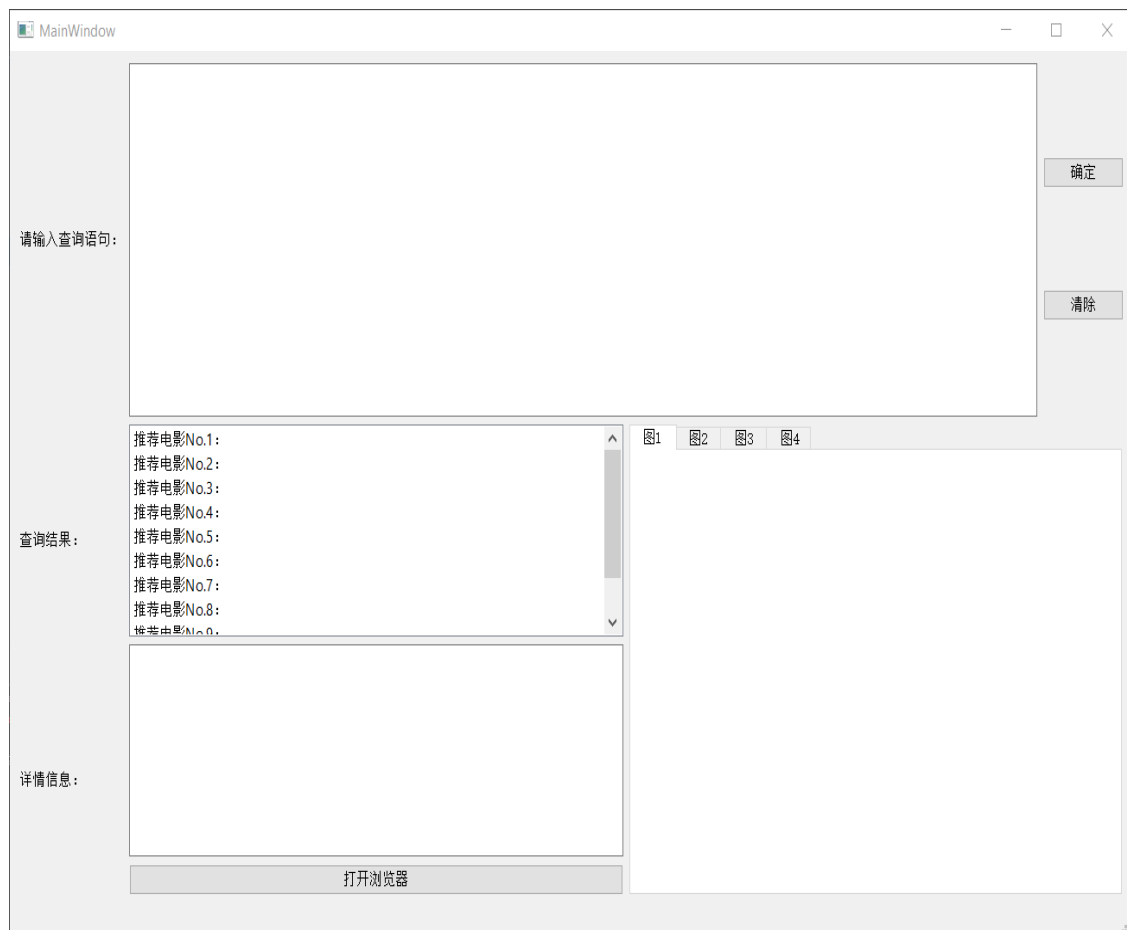
## 9.6 参考资料：

<http://www.jb51.net/tools/zhengze.html> （正则表达式学习资料）

<https://github.com/fxsjy/jieba> （jieba 手册）

## 10 Pyqt 包使用 Python 编写 GUI 程序

结果截图：



## 10.1 环境搭建:

- 1.Pyqt 是 Qt 的 python 版本, 可以直接下载 Pyqt 这个包, 然后 import 即可使用。
- 2.界面的绘制可以下载 Qt designer。

## 10.2 学习笔记:


- 1.界面的编写可以直接使用 Qt designer 来绘制, 通过可视化的方式编辑好界面之后, 导出 ui 文件, 使用 pyqt 自带的 compileUiFile 程序来转换成 py 文件, 就可以直接在 pyqt 中使用了。
- 2.Pyqt 的原理和 Qt 类似, 写好界面之后, 通过信号和槽的机制即可编辑事件。

## 10.3 参考资料:

<http://www.cnblogs.com/zouzf/p/4308912.html> (pyqt4 环境搭建)  
<http://pyqt.sourceforge.net/Docs/PyQt4/> (pyqt4 官方手册)

## 11 Scrapy 爬虫框架学习

结果截图:



```
{
  "score": [7.9],
  "type": ["官方网站"],
  "name": ["地球脉动 第二季 Planet Earth Season 2"]
}, {
  "score": [7.6],
  "type": ["剧情"],
  "name": ["航海王之黄金城 ONE PIECE FILM GOLD"]
}, {
  "score": [8.0],
  "type": ["剧情"],
  "name": ["追凶者也"]
}, {
  "score": [7.4],
  "type": ["剧情"],
  "name": ["请叫我英雄 アイムアヒーロー"]
}, {
  "score": [7.8],
  "type": ["喜剧"],
  "name": ["佩小姐的奇幻城堡 Miss Peregrine's Home for Peculiar Children"]
}, {
  "score": [5.6],
  "type": ["动作"],
  "name": ["机械师2: 复活 Mechanic: Resurrection"]
}, {
  "score": [9.2],
  "type": ["短片"],
  "name": ["鹼 Piper"]
}, {
  "score": [8.2],
  "type": ["动作"],
  "name": ["金山行 부산행"]
}, {
  "score": [9.0],
  "type": ["科幻"],
  "name": ["西部世界 第一季 Westworld Season 1"]
}, {
  "score": [7.2],
  "type": ["喜剧"],
  "name": ["香肠派对 Sausage Party"]
}, {
  "score": [8.7],
  "type": ["剧情"],
  "name": ["黑镜 第三季 Black Mirror Season 3"]
}, {
  "score": [8.1],
  "type": ["动作"],
  "name": ["湄公河行动"]
}, {
  "score": [6.9],
  "type": ["剧情"],
  "name": ["我不是潘金莲"]
}, {
  "score": [7.6],
  "type": ["剧情"],
  "name": ["七月与安生"]
}, {
  "score": [8.4],
  "type": ["剧情"],
  "name": ["驴得水"]
}, {
  "score": [8.6],
  "type": ["剧情"],
  "name": ["比利·林恩的中场战事 Billy Lynn's Long Halftime Walk"]
}, {
  "score": [8.1],
  "type": ["剧情"],
  "name": ["神奇动物在哪里 Fantastic Beasts and Where to Find Them"]
}, {
  "score": [7.9],
  "type": ["动作"],
  "name": ["奇异博士 Doctor Strange"]
}, {
  "score": [6.6],
  "type": ["剧情"],
  "name": ["大鱼海棠"]
}, {
  "score": [7.2],
  "type": ["剧情"],
  "name": ["阿修罗 아수라"]
}, {
  "score": [7.0],
  "type": ["喜剧"],
  "name": ["九条命 Nine Lives"]
}, {
  "score": [9.5],
  "type": ["喜剧"],
  "name": ["我们这一天 This Is Us"]
}, {
  "score": [5.5],
  "type": ["喜剧"],
  "name": ["从你的全世界路过"]
}, {
  "score": [7.2],
  "type": ["奇幻"],
  "name": ["圆梦巨人 The BFG"]
}, {
  "score": [6.7],
  "type": ["奇幻"],
  "name": ["蓝色大海的传说 푸른 바다의 전설"]
}, {
  "score": [7.2],
  "type": ["剧情"],
  "name": ["美人鱼"]
}, {
  "score": [8.2],
  "type": ["剧情"],
  "name": ["萨利机长 Sully"]
}, {
  "score": [4.7],
  "type": ["剧情"],
  "name": ["锦绣未央"]
}, {
  "score": [6.3],
  "type": ["动画"],
  "name": ["名侦探柯南: 纯黑的恶梦 名探偵コナン 純黒の悪夢"]
}, {
  "score": [7.5],
  "type": ["喜剧"],
  "name": ["爱宠大机密 The Secret Life of Pets"]
}, {
  "score": [6.9],
  "type": ["剧情"],
  "name": ["间谍同盟 Allied"]
}, {
  "score": [7.3],
  "type": ["动作"],
  "name": ["谍影重重5 Jason Bourne"]
}, {
  "score": [8.6],
  "type": ["剧情"],
  "name": ["比海更深 海よりもまだ深く"]
}, {
  "score": [7.0],
  "type": ["惊悚"],
  "name": ["屏住呼吸 Don't Breathe"]
}, {
  "score": [6.1],
  "type": ["喜剧"],
  "name": ["自杀小队 Suicide Squad"]
}, {
  "score": [7.2],
  "type": ["剧情"],
  "name": ["法医秦明"]
}, {
  "score": [7.6],
  "type": ["动画"],
  "name": ["魔弦传说 Kubo and the Two Strings"]
}, {
  "score": [7.8],
  "type": ["剧情"],
  "name": ["小姐 아가씨"]
}, {
  "score": [8.6],
  "type": ["剧情"],
  "name": ["荼蘼"]
}, {
  "score": [9.2],
  "type": ["剧情"],
  "name": ["王冠 The Crown"]
}, {
  "score": [7.0],
  "type": ["剧情"],
  "name": ["寒战2"]
}, {
  "score": [9.2],
  "type": ["喜剧"],
  "name": ["疯狂动物城 Zootopia"]
}, {
  "score": [7.4],
  "type": ["喜剧"],
  "name": ["海底总动员2: 多利去哪儿 Finding Dory"]
}, {
  "score": [8.5],
  "type": ["剧情"],
  "name": ["血战钢锯岭 Hacksaw Ridge"]
}, {
  "score": [7.8],
  "type": ["喜剧"],
  "name": ["多利去哪儿 Finding Dory"]
}
```

## 11.1 环境搭建:

- 1.Scrapy 就是 Python 的一个包，可以通过 pip 直接安装。pip install scrapy。
- 2.安装成功后，直接 import scrapy，然后通过继承 Scrapy 提供的类来使用这个框架。

## 11.2 遇到的问题:

- 1.git bash 似乎对 python 的支持不是很好，好像缺少 python 的某些 dll。所以今天运行 scrapy shell 的时候就会卡住。
- 2.pip install scrapy 在安装 lxml 的总是失败。scrapy 有很多其他的依赖包，lxml 是其中一个。pip 有个好处就是能帮我们自动分析包的依赖关系，自动下载需要的包。但是 pip 的使用总是会出现一些莫名其妙的错误。当出现错误的时候，记下是哪个包安装失败了，手动下载 whl 文件，然后使用 pip install XXXX.whl 手动安装即可。
- 3.如果需要退出 shell。Ctrl+c。
- 4.每次执行-o 的操作，内容是 append 的。会被附加到原有文件的后面。而且会破坏这个 JSON 文件。可以使用.json 文件来存储。JSON 文件如果直接 append 的话，那个[]会出现两次（JSON 文件中实际上是一个 list），所以得到的 JSON 文件是错误的。JSON 文件没有[]的问题，里面存储的都是 dic。
- 5.scrapy shell 被拒绝，403，访问豆瓣网的时候，想要启动 shell 看看 selector 对不对，直接就是 403.但是直接运行爬虫就可以。scrapy shell 有参数可以添加 user\_agent。
- 6.Windows 操作系统 cmd 指令需要注意：用双引号，不要用单引号。会出 Errorno 11001。
- 7.scrapy.log 不用了，也不需要使用 scrapy.log.start()来启用 log 服务了。debug, info, warning, error, critical 5 个级别，注意，一律使用 DEBUG 就好，爬虫运行时显示的那些信息也都是 INFO，设置成 ERROR 了就不显示了。
- 8.有的 python 包安装不成功可能是 pip 的版本不是最新的。不过在 pip 更新的时候提示 IOError13 访问 pip.exe 没有权限，被拒绝。这个问题就算通过管理员身份运行 cmd 在执行也不行，最好直接去 pip 官网下载最新版本的 pip 然后重新安装。

Hello Scrapy:

```
//导入这个包，即可使用 Scrapy 框架。
import scrapy

//自己的爬虫需要通过继承 scrapy 的 Spider 类来编写。
class QuotesSpider(scrapy.Spider):
    name = "quotes"
    //需要抓取数据的 url 放入到 start_urls 数组中，会被自动
    抓取。
    start_urls = [
        'http://quotes.toscrape.com/tag/humor/',
    ]
    //重写 parse() 方法，response 是抓取后获得 DOM 文件。
```

```

def parse(self, response):
    //response 的 css() 方法可以很方便的解析 HTML 文
    件，就像 BeautifulSoup 一样。
    for quote in response.css('div.quote'):
        //解析和扒取的工作是异步执行的。
        yield {
            'text':
quote.css('span.text::text').extract_first(),
            'author':
quote.xpath('span/small/text()').extract_first(),
        }

        next_page = response.css('li.next
a::attr("href")').extract_first()
        if next_page is not None:
            next_page =
response.urljoin(next_page)
            yield scrapy.Request(next_page,
callback=self.parse)

```

0. 在 Pycharm 里面编写好代码，需要使用 Python 解释器来执行（因为需要参数）。
1. 在 cmd 中执行 scrapy 的 runspider 指令。scrapy runspider XXX.py -o XXX.json。
2. 结果存储在 XXX.json 文件里面。

## 11.3 学习笔记：

1. 在 Pycharm 里面编写好代码，需要使用 Python 解释器来执行（因为需要参数）。
2. 在 cmd 中执行 scrapy 的 runspider 指令。scrapy runspider XXX.py -o XXX.json。
3. 结果存储在 XXX.json 文件里面。
4. Scrapy 文件操作和发起请求是异步的。使用八爪鱼的时候，请求之间是同步的，第一个网页打开了，parse 了数据，才打开第二个网页。
5. next\_page 的 Request 是通过 for in 遍历 Generator 的方式来获取的。好处是这样我们可以动态的增加需要访问的 URL 并且使得 Scrapy 解析上一个请求的并写入文件中和发起下一个请求之间是异步的。一旦获得了下一个 URL，不管当前这个请求是否完成了解析和文件操作，都可以直接发起下一个请求。
6. Scrapy 的速度很快，一次性可以发送很多个请求出去，所以我们要手动设置请求之间的间隔。
7. 写好的 Spider，如果选择器写错了，趴下来的东西就没用，如果每次都 scrapy crawl spiderName 来运行 spider 检查选择器是否正确，效率太低，很麻烦。所以写好的选择器先用 scrapy shell 检查一下看是否正确然后再运行 spider。

8.CSS selector 的编写可以借助一些工具,Chrome 的开发者工具里面,右击 tag 然后选择 copy,就可以 copy css 和 xpath 了。

9.动态地向 URL 列表中增加新的 URL, URL 列表可能是一开始就设置好的,用一个循环直接可以获得。比如一个贴吧的所有帖子。也可以是随着扒取的进行,一次增加新的 URL。这种一般是打开了一个网页之后,通过获得 href 标签中的文本来获得新的 URL。

## 11.4 使用 scrapy 基本流程:

1.创建项目,使用 Scrapy 控制台指令来创建项目,运行 `spider.scrapy startproject projectName`。  
2.编写自己的 spider (必须是 `scrapy.Spider` 的子类,必须定义好初始的 `URLlist` 和对数据的处理方式 `parse()`方法)

3.在控制台上使用 `scrapy shell "URL"`启动 scrapy shell。可以通过交互式的方式去检查自己编写的 XPath 或者 CSS 选择器是否正确。

4.对扒取下来的网页进行 `parse`, `parse(self, response)`这个函数负责在 spider 获得 response 之后对 response 进行处理。`parse HTML` 文件就是通过某种筛选或者定位的方式来获取 HTML 文件中我们需要的数据。那些不包含 HTML 的 tag 的数据。`response` 是一个结构体,里面有很多属性和方法。和 Servlet 的 `HTTPResponse` 类似,里面包括 HTTP 包的 header 和 body。

5.把扒取到的数据提取出来并存储成 JSON 文件,提取数据,Scrapy 框架在处理扒取到的数据的时候使用的是 Generator。所以需要提取出来的数据按照一个 list 来存储。list 里面可以是 dictionary。执行 `scrapy crawl spiderName -o fileName.json` 来将数据存储到 JSON 文件中。

6.`parse` 返回 item, item 用来存储扒取到的数据,类似 dict,因为扒取到的数据一般都是 dict 结构。功能和 dict 类似,但是多了设定字段的元数据,可以复写来扩展的功能。item 和 dict 之间可以自由转换。在没有 item 的时候,直接返回 {} 也可以。

7.使用 Selector, 获得 response 之后,无论这个 response 是什么(403 还是 200, 正常页面还是验证码页面)都会调用创建 Request 的时候设置的 callback 函数。这个函数可能是默认的 `parse()`函数也可以是任何一个返回了 item 或者 Request 的函数。`response.selector()`获得这个 response 的选择器。这个选择器可以调用 Scrapy 内置的 css 和 xpath 选择器方法。

8.Feed Export。如果我们扒取的过程中突然被 403 了,又没有补救措施,这个时候只要在 cmd 里面 `ctrl+c` 结束,已经爬到的数据就会被存储下来。Feed Exports 有几种序列化格式和几种存储方式。存储方式:本地文件系统、FTP、S3、标准输出。由 `FEED_URI` 指定例如, `FEED_URI = u'file:///G:/program/$(name)s/$(time).csv'`。序列化格式:JSON, JSON lines, CSV, XML, Pickle, Marshal。 `FEED_FORMAT`

9.Item loader, `scrapy.item.Item` 类是 Scrapy 内置的一种类似 dict 的数据类型。item 和 dict 不同,我们通过定义一个派生自 `scrapy.item.Item` 类的类来定义 item。通过 `scrapy.Field()`来给 item 添加 field (field 在 python 中就是 dict 的 key, class 的成员变量),同时, `Field()`接受参数,可以为 field 定义元操作,比如这个 field 的序列化函数是谁。`scrapy.loader.ItemLoader` 可以认为是一个 item 的容器。使用 item 的时候,我们通过 `item['key'] = value` 的方式来给 item 填充值。现在有了 itemLoader,我们只需要把需要填充进去的值“扔给”itemLoader,他就给我自动处理了。填充完毕,再把这个 item 取出来返回出去就行了。

10.Item pipeline, `parse()`函数返回 item (dict 还转成 item), scrapy 执行 request, 获得 response 之后将 response 交给 `parse()`函数处理,获得 request 则加入请求队列获得 item 则将 item 放入 item pipeline 中进行处理。处理后的结果通过 feed export 存储下来。pipeline 只是一个函数,可以对爬取下来的数据做处理。这个很重要啊,就是数据的处理阶段,比如之前扒取豆



瓣数据的时候进行的去除标点符号以及分词的操作。

对扒取下来的网页进行 parse:

def parse(self, response) 这个函数负责在 spider 获得 response 之后对 response 进行处理。

parse HTML 文件就是通过某种筛选或者定位的方式来获取 HTML 文件中我们需要的数据。那些不包含 HTML 的 tag 的数据。

response 是一个结构体，里面有很多属性和方法。和 Servlet 的 HTTPResponse 类似，里面包括 HTTP 包的 header 和 body。

我们主要是对 body 进行操作。body 就是 HTML 文件，通过 css 或者 xpath 选择器获得需要的数据。

```
//对 response 使用 css 选择器，返回的是选择器对象而非我们需要的数据。
```

```
//选择器对象包括 xpath 和数据两部分。
```

```
//返回的是一个选择器的 list。
```

```
>>> response.css('title')
```

```
[<Selector                                xpath='descendant-or-self::title'
data='<title>Quotes to Scrape</title>'>]
```

```
//调用选择器对象的 extract() 方法，获得选择器所有的数据。
```

```
//数据也是一个 list。
```

```
>>> response.css('title').extract()
```

```
[u'<title>Quotes to Scrape</title>']
```

```
//对于数据，我们不需要 HTML 的 tag 信息，只需要 tag 里面的文本。
```

```
//这个返回的也是 list。
```

```
>>> response.css('title::text').extract()
```

```
['Quotes to Scrape']
```

```
//extract_first() 返回 list 的第一个元素。
```

```
>>> response.css('title::text').extract_first()
```

```
'Quotes to Scrape'
```

```
//也可以直接从选择器 list 中确定某一个选择器，提取数据。
```

```
//但是这样可能会出现空指针的错误。
```

```
//extract() 方法是 Selector 对象的方法，用来获得 data 的值。
```

```
>>> response.css('title::text')[0].extract()
```

```
'Quotes to Scrape'
```

```
//response 调用 css() 方法，返回的是 selector 的 list。
```

```
//css 内部的参数影响 data 部分的值。
```

```
>>> response.css('title::text')
```

```
[<Selector                                xpath=u'descendant-or-self::title/text()'
data=u'Quotes to Scrape'>]
```

//Selector 对象也可以调用 re() 方法, 对 data 的值首先进行正则表达式的匹配, 然后输出符合要求的 list。

```
>>> response.css('title::text').re(r'Quotes.*')
['Quotes to Scrape']
```

CSS selector 的编写可以借助一些工具:

1. FireBug。火狐浏览器的插件。
2. SelectorGadget。Chrome 的插件。被墙掉了。

css 选择器:

0. css() 里面的参数是个字符串。

1. 参数一定是某个标签的名字打头。div, span, small, a
2. 标签.XX, XX 表示标签后面的 class。用 class 来定位。small.author
3. ::text 即可获得标签内部的文本内容。span.text::text
4. div.tags a.tag::text, 空格表示子 tag。
5. extract\_first(), 表示只提取一个。
6. extract() 表示有好几个, 需要提取一个 list 出来。
7. 其实 Chrome 的开发者工具里面, 右击 tag 然后选择 copy, 就可以 copy css 和 xpath 了。

使用 FireBug 来获得 XPath:

0. 打开火狐浏览器, 打开需要抓取数据的那个网页。
1. 点击需要抓取的数据, 右键, 使用 firebug 审查元素。
2. 右击对应的元素, 复制 XPath 到粘贴板。
3. 和八爪鱼类似, 存在绝对定位的问题。XPath 还是需要学习的。Chrome 直接 Copy 的 xpath 也存在绝对定位的问题。

把抓取到的数据提取出来并存储成 JSON 文件:

提取数据, Scrapy 框架在处理抓取到的数据的时候使用的是 Generator。所以需要提取出来的数据按照一个 list 来存储。list 里面可以是 dictionary。

```
def parse(self, response):
    //response.css('div.quote') 返回的是一个 selector 的
list。
    for quote in response.css('div.quote'):
        //yield 在执行 parse() 获得 generator 的时候并不
会调用。
        //在对 Generator 使用 for in 的时候 yield 后面的
才会被调用。
        //每次调用返回一个 dictionary
        yield{
            'text':
quote.css('span.text::text').extract_first(),
```

```

        'author' : quote.css('span
small.author::text').extract_first(),
        'tags' : quote.css('div.tags
a.tag::text').extract()
    }

```

执行 `scrapy crawl spiderName -o fileName.json` 来将数据存储到 JSON 文件中。每次执行 `-o` 的操作，内容是 `append` 的。会被附加到原有文件的后面！而且会破坏这个 JSON 文件。可以使用 `.jl` 文件来存储。

JSON 文件如果直接 `append` 的话，那个 `[]` 会出现两次（JSON 文件中实际上是一个 `list`），所以得到的 JSON 文件是错误的。JL 文件没有 `[]` 的问题，里面存储的都是 `dic`。

动态地向 URL 列表中增加新的 URL：

URL 列表可能是一开始就设置好的，用一个循环直接可以获得。比如一个贴吧的所有帖子。

也可以是随着扒取的进行，一次增加新的 URL。这种一般是打开了一个网页之后，通过获得 `href` 标签中的文本来获得新的 URL。

```

#这个函数在获得一个 response 之后被调用。
def parse(self, response):
    #parse 数据。
    for quote in response.css('div.quote'):
        yield{
            'text' :
quote.css('span.text::text').extract_first(),
            'author' : quote.css('span
small.author::text').extract_first(),
            'tags' : quote.css('div.tags
a.tag::text').extract()
        }

    #从当前获得的这个网页里获得下一个网页的超链接。
    next_page = response.css('li.next
a::attr(href)').extract_first()
    #动态的向 URL 队列中加入数据。
    if next_page is not None:
        #next_page = 'http://quotes.toscrape.com' +
next_page

        #因为 URL 是类似的，所以可以直接 urljoin()
        next_page = response.urljoin(next_page)
        #注意 callback 的自动补全，parse 不能有 ()
        yield scrapy.Request(next_page,
callback=self.parse)

```

给自己编写 spider 添加 cmd 参数:

```
#-a key=value 这种方式可以给自己的 spider 添加 cmd 参数。  
scrapy crawl quotes -o quotes-humor.json -a tag=humor
```

在自己编写的爬虫中可以通过 `getattr(self, 'tag', None)` 的方式来获得这个 cmd 参数。tag 是 key 值。会获得 'humor' 这个 cmd 参数。

Selector:

获得 response 之后, 无论这个 response 是什么 (403 还是 200, 正常页面还是验证码页面) 都会调用创建 Request 的时候设置的 callback 函数。这个函数可能是默认的 `parse()` 函数也可以是任何一个返回了 item 或者 Request 的函数。

`response.selector()` 获得这个 response 的选择器。这个选择器可以调用 Scrapy 内置的 `css` 和 `xpath` 选择器方法。`response.selector().css()`

`response.selector().css()` 的简写是 `response.css()`。

`css()` 和 `xpath()` 方法返回的是一个 `selectorList`。这个 list 里面的元素是 dict。dict 有两个 key, 一个是定位表达式, 另一个是定位后获得的字符串。

注意, 我们需要的是那个字符串数据, 也就是 data 字段。

对于选择器来说, tag 和 tag 内部的文本以及 tag 的某个属性的值都是一样的, 他们都是字符串。但是我们需要通过 `/text()`, `/@href()` 的方式来定位到 tag 内部文本或者某个属性的值上面。

获得 `selectorList` 之后, 通过 `[index]` 就可以直接访问每一个 dict 了。然后调用 `extract()` 可以获得对应的 data 的 string。

一般这样处理: `selectorList.extract()` 或者 `extract_first()`。前者返回一个 string 的 list, 后者返回单个 string。

Feed Export:

Feed Export 是很好用啊。如果我们扒取的过程中突然被 403 了, 又没有补救措施, 这个时候只要在 cmd 里面 `ctrl+c` 结束, 已经爬到的数据就会被存储下来。

1. 在 `settings.py` 文件里面定义。或者直接 `scrapy crawl name -o fileName.XX`

2. Feed Exports 有几种序列化格式和几种存储方式。

存储方式: 本地文件系统、FTP、S3、标准输出。由 `FEED_URI` 指定例如, `FEED_URI = u'file:///G:/program/%(name)s/%(time).csv'`。

其中, time 由输出时的时间戳代替, 而 name 则是爬虫的 name 属性。也可以有其它的命令参数, 比如 `%(site_id)s`, 此时爬虫需要有 `site_id` 属性。

序列化格式: JSON, JSON lines, CSV, XML, Pickle, Marshal。FEED\_FORMAT 选项指定, 若没有指定, 则根据 `FEED_URI` 的后缀来猜测。例如, `FEED_FORMAT = 'jsonlines'`

`FEED_STORE_EMPTY` 可以用来控制是否输出空的 feed。默认为 False。

`FEED_EXPORT_FIELDS` 来控制输出的字段以及其顺序, 如 `FEED_EXPORT_FIELDS = ["foo", "bar", "baz"]`。这点对于那些有固定的 header 栏的 csv 文件尤其有用。

Item loader:

scrapy.item.Item 类是 Scrapy 内置的一种类似 dict 的数据类型。parse() 函数要求返回一个 dict 或者 item 或者 Request。

item 和 dict 不同, 我们通过定义一个派生自 scrapy.item.Item 类的类来定义 item。通过 scrapy.Field() 来给 item 添加 field (field 在 python 中就是 dict 的 key, class 的成员变量), 同时, Field() 接受参数, 可以为 field 定义元操作, 比如这个 field 的序列化函数是谁

scrapy.loader.ItemLoader 可以认为是一个 item 的容器。使用 item 的时候, 我们通过 item['key'] = value 的方式来给 item 填充值。现在有了 itemLoader, 我们只需要把需要填充进去的值“扔给”itemLoader, 他就给我自动处理了。填充完毕, 再把这个 item 取出来返回出去就行了。

```
import scrapy

#虽然 Item 的元数据很好用, 但是目前我就把 Item 当成 dict 来使用。
class HelloItem(scrapy.Item):
    name = scrapy.Field()
    type = scrapy.Field()

#parse() 仍然返回 item。
def parse_movie(self, response):
    #使用 ItemLoader 来“代理”填充数据的工作。
    itemLoader = ItemLoader(item=HelloItem(),
response=response)
    #对于 itemLoader, 我只需要“把那个 value 赋值给这个 key”
    #我不需要管 css, XPath 还是 value。
    #如果某个 field 是一个 list, 无所谓, 我只管扔数据给 loader
    即可。

    itemLoader.add_xpath('name',
'/html/body/div[3]/div[1]/h1/span[1]/text()')
    itemLoader.add_xpath('type',
'/html/body
/div[3]/div[1]/div[2]/div[1]/div[1]/div[1]/div[2]/div[1]/div[2]/strong/text()')

    #填充完了, 拿出 item, 返回。
    yield itemLoader.load_item()
```

举个例子: 如果某个 field 是一个 list, 而这个 list 可能来自不同的位置, 原来我需要 append 到后面, 现在只要扔给 loader 即可。loader 会知道放到哪里去。

Item pipeline:

parse()函数返回 item (dict 还转成 item)，scrapy 执行 request，获得 response 之后将 response 交给 parse()函数处理，获得 request 则加入请求队列获得 item 则将 item 放入 item pipeline 中进行处理。处理后的结果通过 feed export 存储下来。

pipeline 只是一个函数，可以对爬取下来的数据做处理。这个很重要啊，就是数据的处理阶段，比如之前扒取豆瓣数据的时候进行的去除标点符号以及分词的操作。

实现自己的 ItemPipeline:

1. ItemPipeline 就是一个 Python 类，但是这个类必须有一个 process\_item(item, spider) 函数，这个函数必须返回一个 item。
2. 写好的这个类，在 settings.py 中配置一下 ITEM\_PIPELINES 的值。（这个就没有通过继承框架的类或者实现接口的方式来使用，就像使用 Java 程序使用注解，可能注解也是通过单独维护一个类似 ITEM\_PIPELINES 的东西实现的吧）
3. open\_spider(spider) 可以执行一些初始化操作。
4. close\_spider(spider) 可以执行一些释放资源的操作。
5. class 自己的 \_\_init\_\_() 函数。

//去除这个 item。之后的 pipeline 就不会处理这个 item，并且 item 不会被 feed export 写入到文件中。

```
from scrapy.exceptions import DropItem

class PricePipeline(object):

    vat_factor = 1.15

    def process_item(self, item, spider):
        if item['price']:
            if item['price_excludes_vat']:
                item['price'] = item['price']
* self.vat_factor
            return item
        else:
            raise DropItem("Missing price in %s" %
item)

//去重操作。
class DuplicatesPipeline(object):

    def __init__(self):
        //{}是 dict, []是 list, ‘’ 是 string。
        //set 就是集合，要求元素不重复，无序，可以执行
集合操作。

        //用 list 也一个意思。
        self.ids_seen = set()
```

```

        def process_item(self, item, spider):
            if item['id'] in self.ids_seen:
                raise DropItem("Duplicate item
found: %s" % item)
            else:
                self.ids_seen.add(item['id'])
                return item

```

settings.py 文件中启动 item pipeline:

```

//后面的值表示调用顺序。0-1000，从低到高执行。
ITEM_PIPELINES = {
    'myproject.pipelines.PricePipeline': 300,
    'myproject.pipelines.JsonWriterPipeline': 800,
}

```

Logging:

```

self.logger.info('log info!!!!!!!!!!!!!!')
self.logger.warning('log warning!!!!!!!!!!!!!!')
self.logger.error('log error!!!!!!!!!!!!!!')
self.logger.critical('log critical!!!!!!!!!!!!!!')
self.logger.debug('log debug!!!!!!!!!!!!!!')

```

调用这个函数可以直接在运行爬虫的时候在 cmd 上面显示出来信息。print 也成。以前的那个 scrapy.log 的模块已经被弃用了，现在不能使用。

scrapy.log 不用了，也不需要使用 scrapy.log.start() 来启用 log 服务了。debug, info, warning, error, critical 5 个级别，注意，一律使用 DEBUG 就好，爬虫运行时显示的那些信息也都是 INFO，设置成 ERROR 了就不显示了。

scrapy crawl name --loglevel DEBUG --logfile 'log.txt'，保存成文件还是很方便的。

Scrapy 内置的智能限速功能:

# Enable and configure the AutoThrottle extension (disabled by default)

#

See

<http://doc.scrapy.org/en/latest/topics/autothrottle.html>

#AUTOTHROTTLER\_ENABLED = True

# The initial download delay

#AUTOTHROTTLER\_START\_DELAY = 5

# The maximum download delay to be set in case of high latencies

#AUTOTHROTTLER\_MAX\_DELAY = 60

# The average number of requests Scrapy should be sending in

```
parallel to
    # each remote server
    #AUTOTHROTTLER_TARGET_CONCURRENCY = 1.0
    # Enable showing throttling stats for every response received:
    #AUTOTHROTTLER_DEBUG = False
```

遇到的问题:

1. scrapy shell 被拒绝, 403.

访问豆瓣网的时候, 想要启动 shell 看看 selector 对不对, 直接就是 403. 但是直接运行爬虫就可以。难道是 header 的问题? scrapy shell 是否还有其他参数。

scrapy shell 有参数可以添加 user\_agent

```
scrapy shell -s USER_AGENT="Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36" "https://movie.douban.com/tag/2016?start=0&type=T"
```

需要注意: 用双引号, 不要用单引号。会出 Error no 11001。

2. 怎么防止被 ban.

使用 user agent 池, 轮流选择之一来作为 user agent。池中包含常见的浏览器的 user agent (google 一下一大堆), Scrapy 好像是有这种中间件。

禁止 cookies (参考 COOKIES\_ENABLED), 有些站点会使用 cookies 来发现爬虫的轨迹。

设置下载延迟 (2 或更高)。参考 DOWNLOAD\_DELAY 设置。

使用 IP 池。例如免费的 Tor 项目 或付费服务 (ProxyMesh)。

使用高度分布式的下载器 (downloader) 来绕过禁止 (ban), 您就只需要专注分析处理页面。这样的例子有: Crawlara

scrapy 的自动限速拓展。

Jobs 暂停爬虫。

图像识别来搞定验证码。

3. 将 unicode 编码转成 UTF-8

抓取下来的数据都是 unicode 的编码, 如果只是想 print 一下, 直接使用 string 的 decode() 方法即可。

如果需要存储成文件, 需要对文件进行 UTF-8 编码格式的操作, 需要使用 codecs 这个包。使用方法和 python 原生的文件操作类似。

使用 pipeline 来存储成 UTF-8 的文件:

```
import json
import codecs
```

```
class W3SchoolPipeline(object):
    def __init__(self):
        self.file
```

=



```
codecs.open('w3school_data_utf8.json', 'wb', encoding='utf-8')

    def process_item(self, item, spider):
        line = json.dumps(dict(item)) + '\n'
        # print line
        self.file.write(line.decode("unicode_escape"))
)

    return item
```

## 11.5 参考资料:

<https://doc.scrapy.org/en/1.2/topics/firebug.html#topics-firebug> (使用 **firebug** 辅助抓取数据)  
[http://www.w3school.com.cn/css/css\\_selector\\_type.asp](http://www.w3school.com.cn/css/css_selector_type.asp) (CSS 选择器文档)