Douban Movie Advanced Search Engine

SHIBI HE，何石弼 3130000164 求是科学班 13 级 计算机

Abstract

This paper provides a new search engine for common users to search movies on http://movie.douban.com. This new search engine is Douban Movie Advanced Search Engine(DMASE). Compared with the default search engine in http://movie.douban.com, DMASE is more intelligent, efficient, and powerful. The paper explains how DMASE works and what algorithm DAMSE used. To illustrate the advantages of DMASE, this paper lists several examples.

Keywords

Common user, Intelligence, precision

1. Introduction to DMASE

DMASE will get your query and give you a extremely precise and useful results in the blink of an eye. In accordance with your queries, DMASE will show you the movies you probably want to take a view. DMASE is very intelligent, remember, you are forbidden to tell DMASE which direction or category you want to search in. DMASE will find out whether you are searching a movie title, a director, an actor, a description or a genre. You only need to type a few words, and please, let DMASE take care of the other things. DMASE is an advanced search engine, except for a basic searching utility, DMASE also has some advanced functions which will be figured out soon.

DMASE is not designed for analysis, statistics or math purpose. DMASE is not designed for engineers, scientists, administrators, but popular users. Popular users usually search movie by typing a few words, so DMASE is concentrating on search useful, relevant data by phrases or a few words. DMASE is not designed for receiving a long article and using that article to find answers, because common users only want to type less words and still get their results fast and precisely. However, not designed for a long input doesn't mean DMASE could not search by a long article. DMASE is very good at calculating similarity between two papers. For example, if you give DMASE a long article writing about a particular movie scenario, DMASE will quickly find out which movie you are talking about.

DMASE aims at Chinese and English search. Inside DMASE body, there is a mighty parser, it is powerful to do word extraction and sentence segmentation. It could extract phrases, name, date, location from Chinese and English sentences. As you know, most movies in Douban are written in Chinese, and their titles usually consists of both Chinese and English like this: 复仇者联盟 2：奥创纪元 Avengers: Age of Ultron, 盗梦空间 Inception. DMAS are able to analyze them perfectly. But inevitably, some other movies are freaks, their information often like this: 名门绅士之缘定芳林 สุภาพบุรุษจุฑาเทพ คุณชายรัชชานนท์ (2013) and this: 跨越彩虹 오버 더 레인보우 (2006) DMASE parser works perfectly when dealing with Chinese and English. And although DMASE is designed for Chinese and English users, in the under layer of the program, it is achieved by unicode character, so basically it still works flawlessly with your Korean, Russian, Japanese, Indian input!

2. The improvement in DMASE

Suppose you, a common user, are going to find some fictions which you may be interested in. So you will search "科幻" on Douban, you will get these results:

The top answer is “你是不是要找杭州的“科幻影院””. Let’s put aside how stupid this top answer is, just follow its suggestion by clicking. Then you will get:

Oh no. It’s a dead end!

Now I assume you ignored the ridiculous top answer, but look at here:

I am one hundred percent sure that you were not looking for this movie. Just look at this movie, it got no comment, no photo and no rating. How is that possible a popular user wants to watch that kind of movie?

We are users. We are neither a movie librarian nor a Douban movie manager. We are users! So let those kind of movies stay in the dust bin and never bother us again, because we, the popular users, do not need them.

Apparently the search engine on Douban is based on matching the movie title with the given input, then showing users the matched results.

Attention here, the “matched movies” means partially matched or let me put it another way: the title of the movie contains the input word(“科幻电影与未来时代”contains“科幻”). Further more, if you are searching the word:“科幻电影与未来时代”instead of“科幻”, in this case, no matter how unpopular, unfamiliar the movie“科幻电影与未来时代”is,“科幻电影与未来时代”should be the top answer because the user typed exactly the movie’s title. When user typed words like “科幻”, in tens of thousands of movies, the partially title matched results may not the good answers.

Let us still focus on the keyword:"科幻".

When we turn to page 3, we will find these results:

星际穿越 / 星际启示录(港) / 星际效应(台) [可播放]

2014-11-07(美国) / 2014-11-12(中国大陆) / 马修·麦康纳 / 安妮·海瑟薇 / 杰西卡·查斯坦 / 迈克尔·凯恩 / 麦肯吉·弗依 / 蒂莫西·柴勒梅德 / 约翰·利特高 / 韦斯·本特利 / 大卫·吉雅西 / 比尔·欧文 / 马特·达蒙 / 卡西·阿弗莱克 / 托弗·戈瑞斯...

★★★★★ 9.1 (313768人评价)

复仇者联盟2：奥创纪元 / 复仇者联盟2 / 复仇者联盟：奥创时代

2015-05-01(美国) / 2015-05-12(中国大陆) / 小罗伯特·唐尼 / 克里斯·海姆斯沃斯 / 马克·鲁弗洛 / 克里斯·埃文斯 / 斯嘉丽·约翰逊 / 杰瑞米·雷纳 / 詹姆斯·斯派德 / 塞缪尔·杰克逊 / 唐·钱德尔 / 亚伦·泰勒-约翰逊 / 伊丽莎白·奥尔森 /...

★★★☆☆ 7.1 (124094人评价)

分歧者2：绝地反击 / 叛乱者：强权终结(港) / 分歧者2：叛乱者(台)

2015-03-20(美国) / 2015-06-19(中国大陆) / 谢琳·伍德蕾 / 提奥·詹姆斯 / 凯特·温丝莱特 / 奥克塔维亚·斯宾瑟 / 杰·科特尼 / 佐伊·克罗维兹 / 迈尔斯·特勒 / 安塞尔·艾尔高特 / 李美琪 / 娜奥米·沃茨 / 梅奇·费法 / 贾斯蒂斯·利克 / 本·劳埃德-休斯...

★★★☆☆ 5.8 (15417人评价)

彗星来的那一夜 / 相干性 / 相干效应 [可播放]

2013-09-19(奥斯汀奇幻电影节) / 2014-08-06(美国) / 艾米丽·芭尔多尼 / 莫瑞·史特林 / 尼古拉斯·布兰登 / 伊丽莎白·格瑞斯 / 亚历克斯·马努吉安 / 劳伦·马赫 / 雨果·阿姆斯特朗 / 劳伦·斯卡法莉娅 / 美国 / 英国 / 詹姆斯·沃德·布柯特...

★★★★★ 8.3 (73737人评价)

超感八人组 第一季 / 超感猎杀 / 超感八人

2015-06-05(美国) / 米格尔·安赫尔·西尔维斯特 / 杰米·克莱顿 / 布莱恩·J·史密斯 / 裴斗娜 / 阿梅尔·艾米恩 / 塔彭丝·米德尔顿 / 马克思·雷迈特 / 蒂娜·德赛 / 弗莉玛·阿吉曼 / 纳威恩·安德利维斯 / 达丽尔·汉纳 / 豪威·约翰逊 / 亚当·沙皮罗...

★★★★★ 8.7 (11642人评价)

These movies are exactly the results that common users wanted to view. When a big and unspecific word like“科幻”typed, users are expected to find some good and popular movies. Usually people will type a more detailed sentence to find a particular movie which may not be very popular.

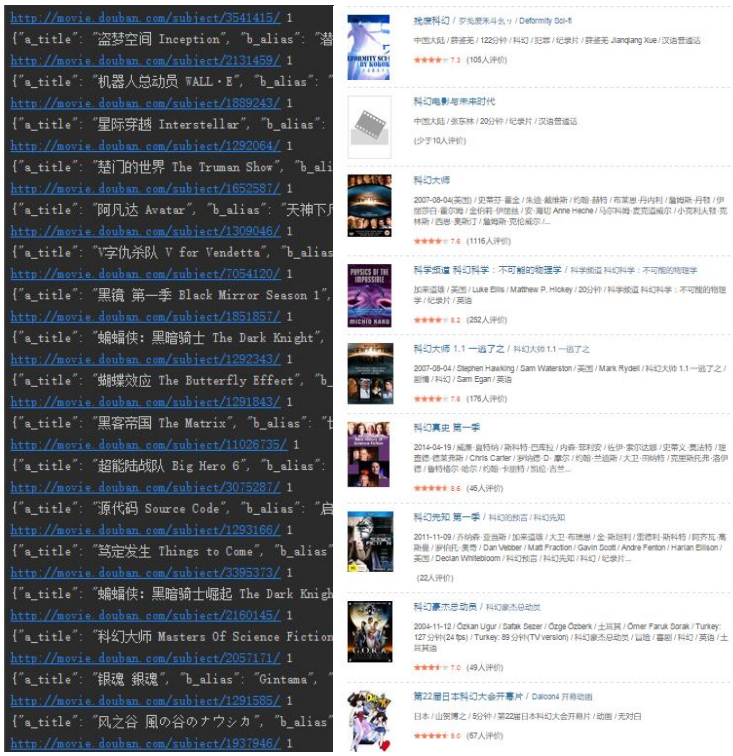
Now the problem is how to search both precisely and intelligently.

Improving the engine’s intelligence is exactly what DMASE is trying to achieve. One way is the recommendation system: after logged in my account, the search engine will give me the answer to “科幻” according to my user habit. But DMASE won’t choose that, because DMASE is a general or universal search engine, designed for popular and common users, DMASE has to give users good answers no matter whether users have a movie account, watching history or not.

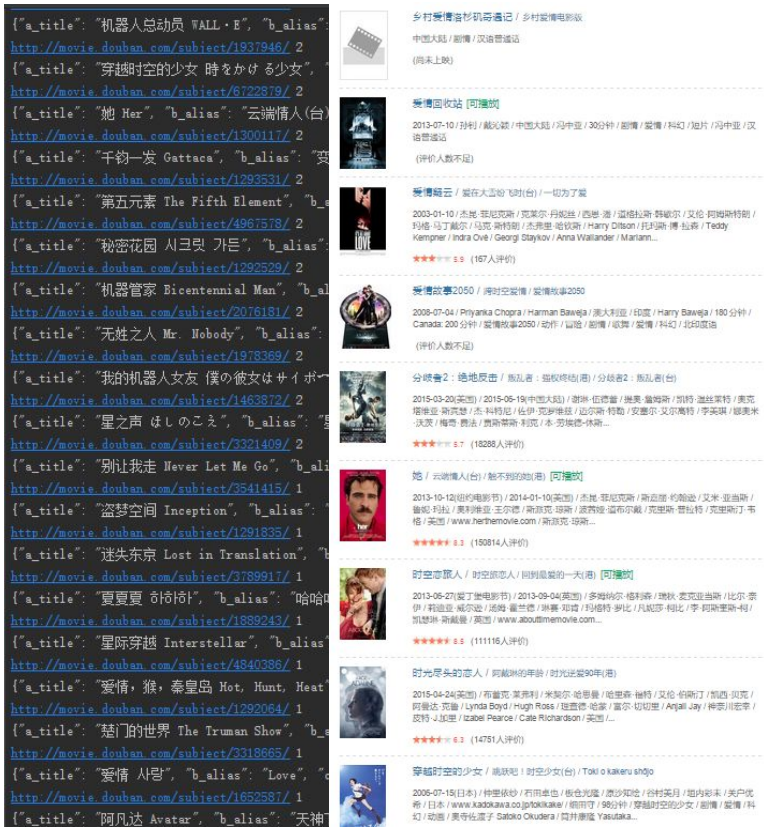
People care a lot about the difference between recommendation system and search system. Recommendation system has its own defects. It always get lost, misled, wrong guided, so pushing completely irrelevant suggestion happens. DMASE is not going to be a recommendation system.

3. Some demos of DMASE

If we search “科幻”, the results in left side are computed from DMASE.

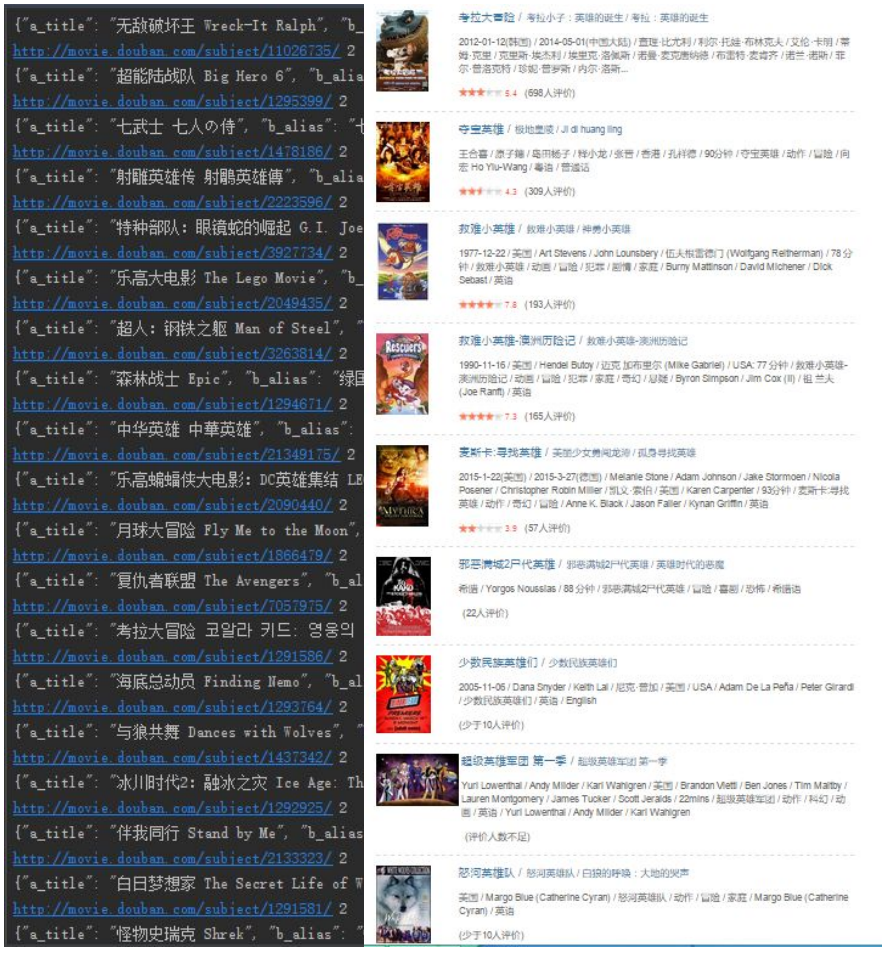


If we search “科幻 爱情”, the results in left side are computed from DMASE.

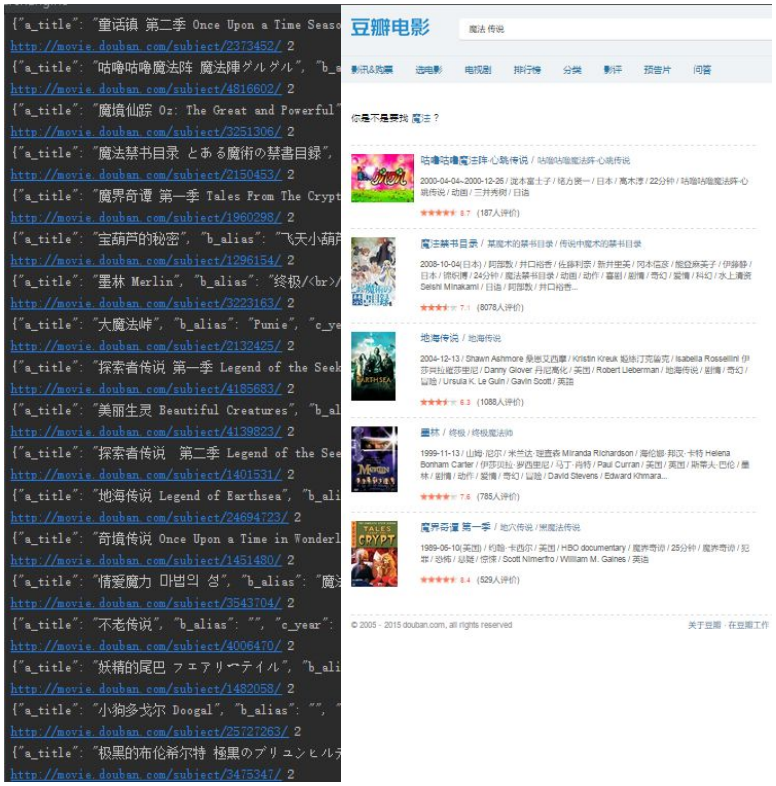


If we search “身手敏捷”, the results in left side are computed from DMASE.

If we search “冒险 英雄”, the results in left side are computed from DMASE.



If we search “身手敏捷”, the results in left side are computed from DMASE.



Both the precision and the intelligence of DMASE are apparently far better than the default search engine in http://movie.douban.com.
For more details, please connect to ZJUWlan, our searching web sever is here:

http://10.214.0.195:10000/

4. Search engine design

4.1 Keywords generation

First I will show some basic algorithm in searching engine:

1. Term Frequency Weight

The log frequency weight of term t in d is defined as follows

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

2. Idf Weight

The document frequency dft is defined as the number of documents that t occurs in We define the idf weight of term t as follows:

$$\mathrm{idf}_t = \log_{10} \frac{N}{\mathrm{df}_t}$$

3. Tf-idf Weight

The tf-idf weight of a term is the product of its tf weight and its idf weight:

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$

4. Cosine Similarity between Query and Document

$$\cos(\vec{q}, \vec{d}) = \mathrm{SIM}(\vec{q}, \vec{d}) = \frac{\vec{q}}{|\vec{q}|} \cdot \frac{\vec{d}}{|\vec{d}|} = \sum_{i=1}^{|V|} \frac{q_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}} \cdot \frac{d_i}{\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

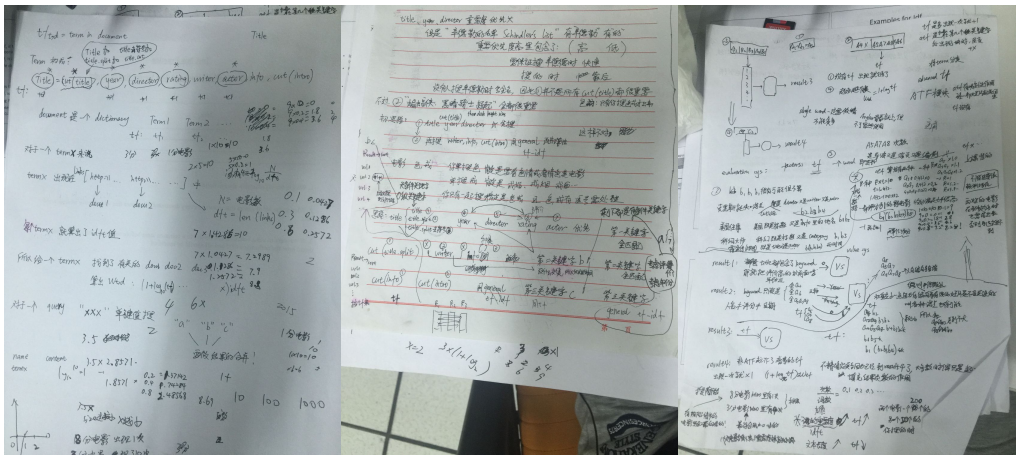
let us consider the Cosine Similarity between Query and Document. Given that the different parts of the information in a movie have different significance(the movie title is apparently more important than the movie description), this algorithm is not suitable for movie search engine. Usually the Cosine Similarity between Query and Document is powerful in general text search.

So the features of a movie search engine are:

1. Common users only want to type less words and still get their results fast and precisely.

2. The information of a movie has different priorities(The keywords in the title and the director is much more significant. But those in movie description are less important).

Based on these features, DMASE is aim at searching extremely precise on a single keyword. Then I built several models for DMASE, let us have a look.



Finally I designed my searching structure using keyword hierarchy.

Generate keywords

Title: Title: a1. Title.split( ): a2 reduceSign(title.split()): a2

“食神”“宿主 The Host” “色，戒”

Alias[]: a1. Alias[].split( ): a2 reduceSign(alias[].split()): a2

Year[0]: Year[0]: a4.

Director[]: Director[] and split( ): a7

***Rating[]:** *Rating[]: a5

Actor[]: Actor and split( ): a6

Writer[]: writer and split( ): a6

Important roles are first five actor and director

Cut_list(title.split()): b1 **Only cut when new words generated**

Cut_list(alias[].split()): b1

Cut_list(director[]): b2

Cut_list(actor[]): b2

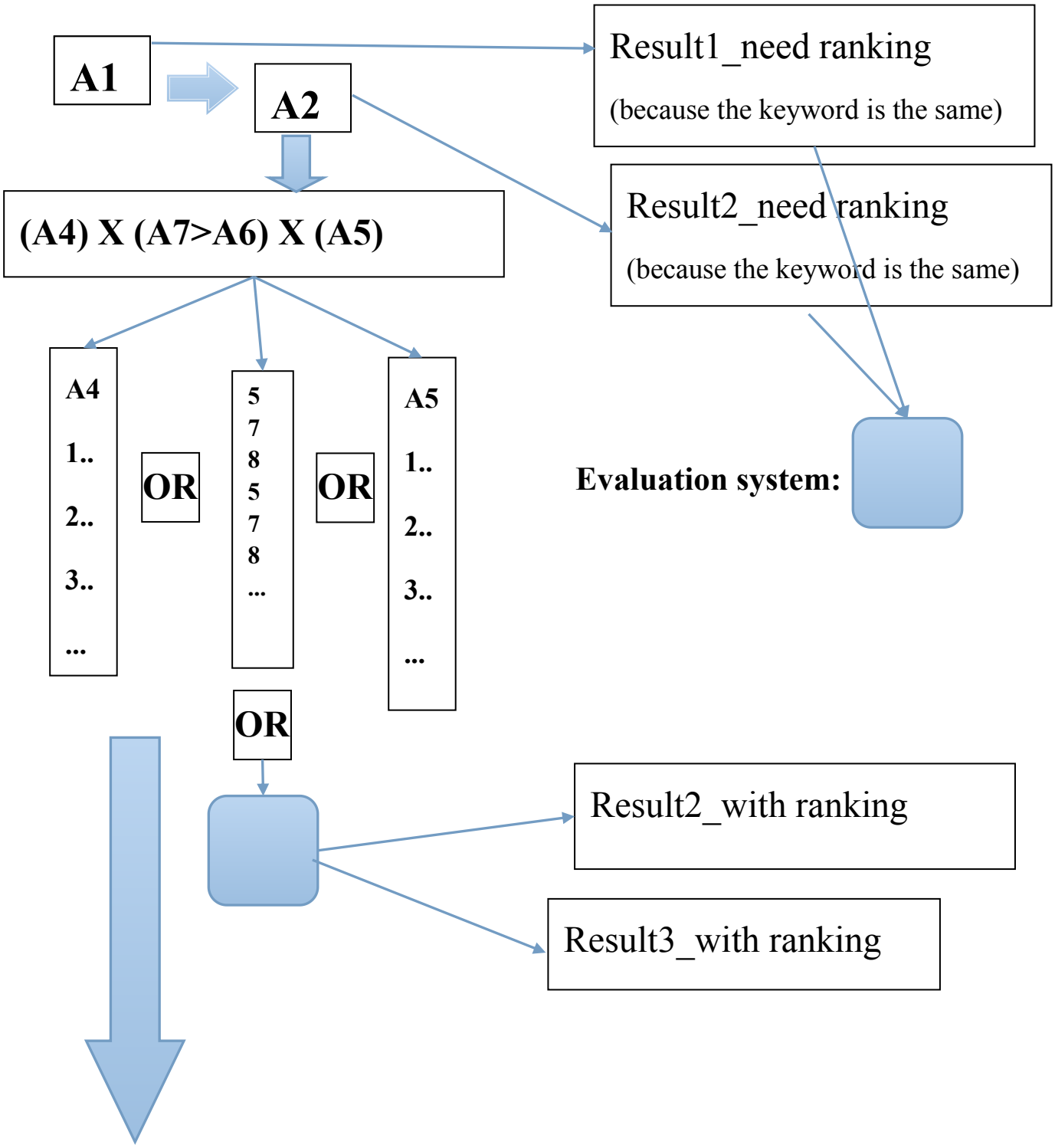
Cut_list(writer[]): b2

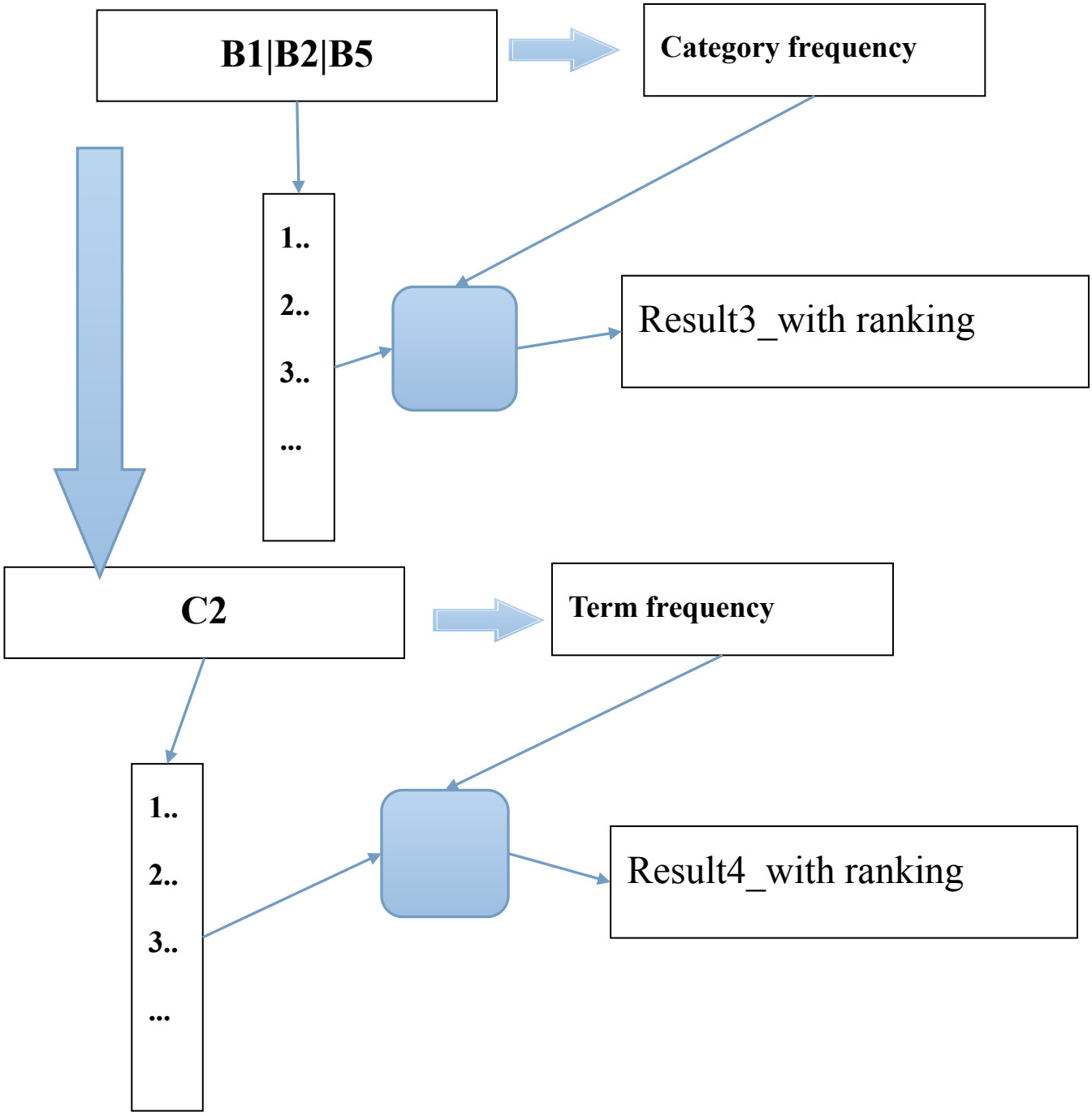
Info[]: b5

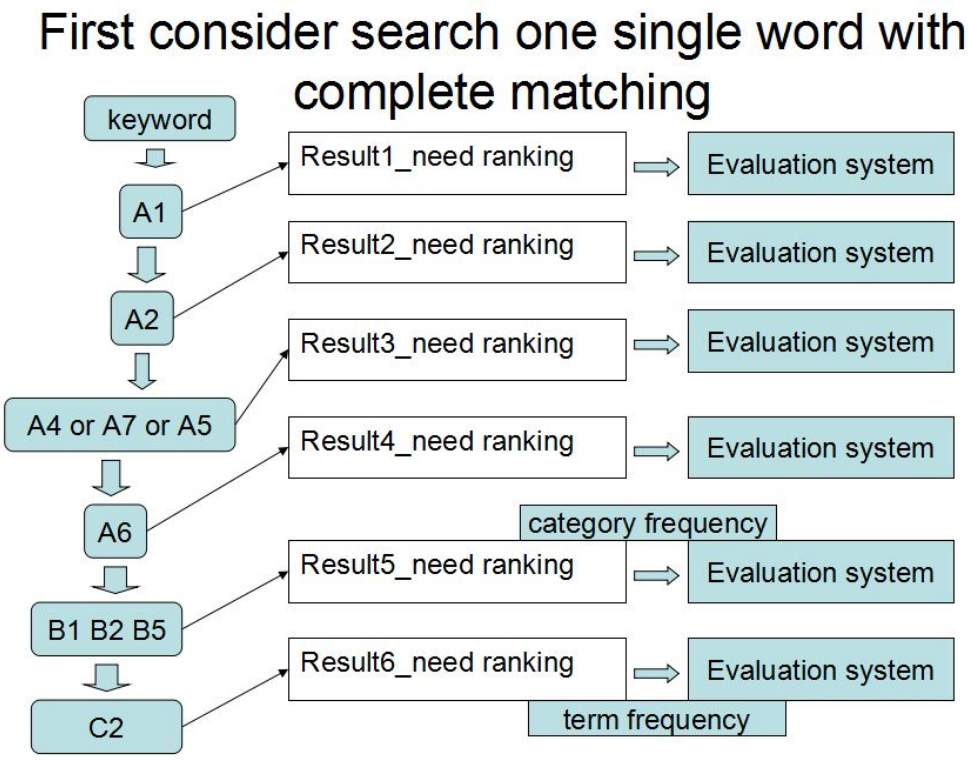
Cut_list(info[]): c1 united with b5

Cut_list(intro[]): c2

The number a1, a2, b1, b2 are the priorities of the keywords. DMASE will search the highest priority keywords first, then the lower priority keywords, finally the least important keywords.


A1
A2
Result1_need ranking
(because the keyword is the same)
Result2_need ranking
(because the keyword is the same)
(A4) X (A7>A6) X (A5)
A4
1..
2..
3..
...
OR
5
7
8
5
7
8
...
OR
A5
1..
2..
3..
...
Evaluation system:
OR
Result2_with ranking
Result3_with ranking


B1|B2|B5
Category frequency
1..
2..
3..
...
Result3_with ranking
C2
Term frequency
1..
2..
3..
...
Result4_with ranking



The single-word query will be insert into A1, A2, and so on, and we will have six Result*_need ranking.

4.2 Evaluation and ranking

4.2.1 rating and comments

How to evaluate a movie is very interesting. The following two movies have extremely high ratings, but *The Shawshank Redemption* has 628860 people who commented on this movie while *the Beethoven Symphony No.9* only has 84 people who has made comments.

The statistics show that

top famous movies usually have 10^5 comments;

top popular movies usually have 10^4 comments;

recent common movies usually have 10^3 comments;

infamous or unpopular movie usually have 10^2 or 10^1 comments.

Considering the number of comments is significant, I designed the final rating for DMASE, combined both rating and comment number.

Final rating = rating(0~10)*log(comments_amount) (1~ 60)

4.2.2 category frequency

If a keyword is both in a movie actor's name and in that movie's title, than that movie will probably be the user's first choice. The category frequency is trying to find a keyword that appears in multiple categories.

The category frequency algorithm was implemented in B1 B2 B5, which are cut titles, cut names and movie information. For example, if we search "美国", in the movie "美国往事", "美国" is a part of the cut title, and "美国" is the region in movie information, then the movie "美国往事" will be more significant in the ranking system, because it has strong connection with the word "美国". Another example is "霍金传". "霍金" both appears in title and names.

Cut_list(title.split()): b1

Cut_list(alias[].split()): b1

Cut_list(director[]): b2

Cut_list(actor[]): b2

Cut_list(writer[]): b2

Info[]: b5

Cut_list(info[]): c1 united with b5

B1 B2 B5

In 4.2.1, rating and comments system will always put the famous movies on the top of the results. However, sometimes a keyword is so important that it can identify some close related movies immediately, in this case rating and comments system is not efficient because it only considered one principle: good movie first.

In category frequency algorithm, I am trying to find the movies that are closed related with the keywords, even if those movies are not famous or popular. The simplest way is to use the score in rating and comment system to multiply a coefficient. If the keyword appears in only one category, the coefficient is 1. If the keyword appears in two or three categories, we can make the coefficient into 1.3 or 1.5. But this method is not good enough. An unpopular movie may has few comments, so its score in rating and comment system is very low(20 or 30 of 60). Whatever the coefficient is, the movie will always in the bottom of the ranking results compared with the famous and popular movies which always get 50 or even 55.

My solution is $((\frac{60}{x}-1)*coefficient+1)*x$ where x is the score from rating and comment system

When cf=1 coefficient=0 appears in only one category

When cf=2 coefficient=0.6 appears in two categories

When cf=3 coefficient=0.8 appears in three categories

Let us find out why my solution is more reliable. Suppose an unpopular movie has got 30 from rating and comment system, if the keyword appears only once, then final score is 30. If the keyword appears twice, the score will be 48, and if the keyword appears three times, the final score will be 54! Thus an unpopular movie will show up when the time is right.

4.2.3 term frequency

When we need to test the keyword in movie description, the term frequency is a powerful algorithm to find which movie is more related with the keyword.

1. Term Frequency Weight

The log frequency weight of term t in d is defined as follows

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

2. Idf Weight

The document frequency dft is defined as the number of documents that t occurs in We define the idf weight of term t as follows:

$$\mathrm{idf}_t = \log_{10} \frac{N}{\mathrm{df}_t}$$

3. Tf-idf Weight

The tf-idf weight of a term is the product of its tf weight and its idf weight:

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$

In my database, there are three kinds of term frequency: ABC, BC and C. Each one has different utilities. The C term frequency only comes from movie description and others' range are wider.

With term frequency, we can easily find the certain movies that contains the keyword many times.

5. Framework

Here are my four general steps to build DMASE:

1. **Crawler**
2. **Extract data from the web page**
3. **Build index**
4. **Serving the service**

1. Crawler

A crawler is also called a spider which is a program that is capable of iteratively and automatically extract data from website.

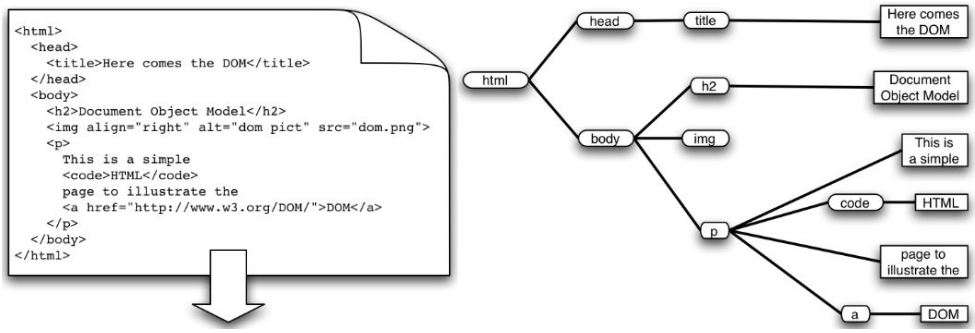
1. Begin with "seeds" URLs
2. Fetch and parse html body
3. Extract new URLs and find where they point to
4. Place the extracted URLs on a queue
5. Fetch each URL on the queue and repeat
6. Storing the scraped item

Given that python Scrapy framework is easily extensible, and portable, I chose Scrapy to build my spider .

2. Extract data from the web page

To extract data from the HTML, I have many tools: BeautifulSoup, lxml, etc.

To select nodes in XML, I chose XPath and CSS for they are easy to learn and easy to use.



3. Build index

Before index, I must do word segmentation. Here lists some of my alternative choices:

Chinese word segmentation:

Jieba

NLPIR

Although NLPIR is more powerful, I still choose Jieba to build DMASE. The reason is I love Jieba' s elegance, delicacy and simplicity.

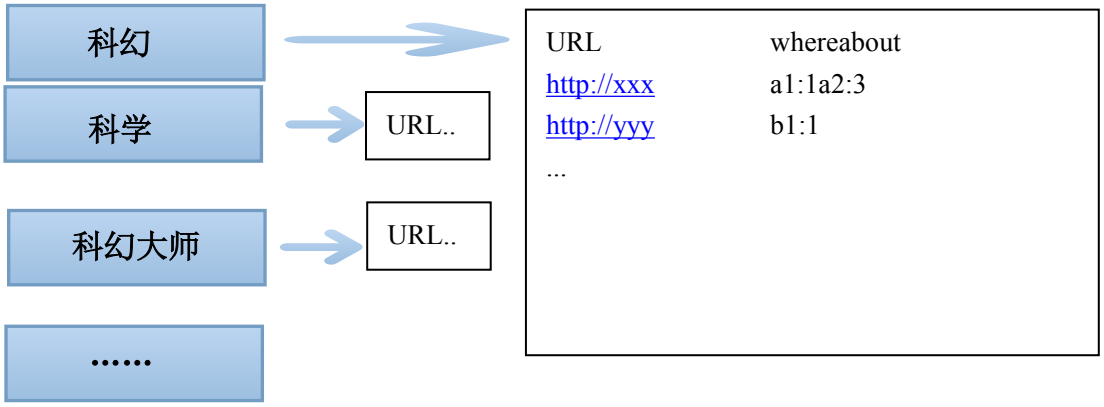
Index:

Lucene

Xapian

Sphinx

Well, I did not use any of them. I build my index myself. The structure of mine is:



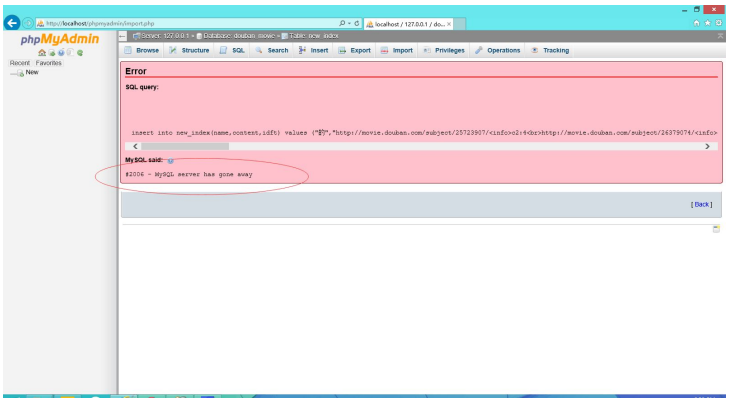
I stored my index and movie information in the JSON format.

4. Serving the service

The Database I chose is MySQL. The connection between MySQL and python is implemented by MySQLdb.

6. Details

6.1 thiry thousand is different from three thousand: the problem happens when data is huge



6.2 back slash in mySQL

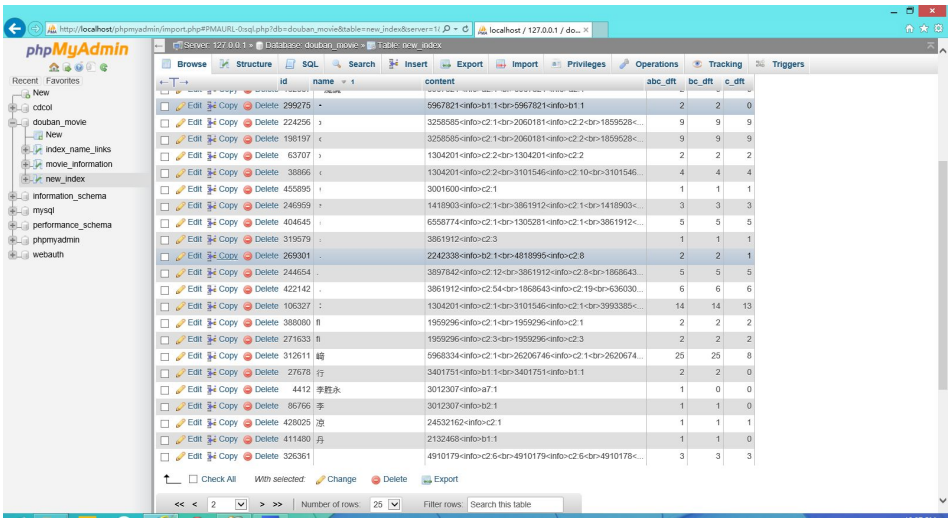
```
def processSigns(string):
    stringSplit = string.split('\\')
    s = u""
    for littleS in stringSplit:
        s += littleS + r"\\"
    string = s[:-2]
    stringSplit = string.split('"')
    s = u""
    for littleS in stringSplit:
        s += littleS + r'\"'
    s = s[:-2]
    return s
```

6.3 problems in movie titles

哈利波特 1：神秘的魔法石(港/台) / 哈 1 / Harry Potter and the Philosopher's Stone

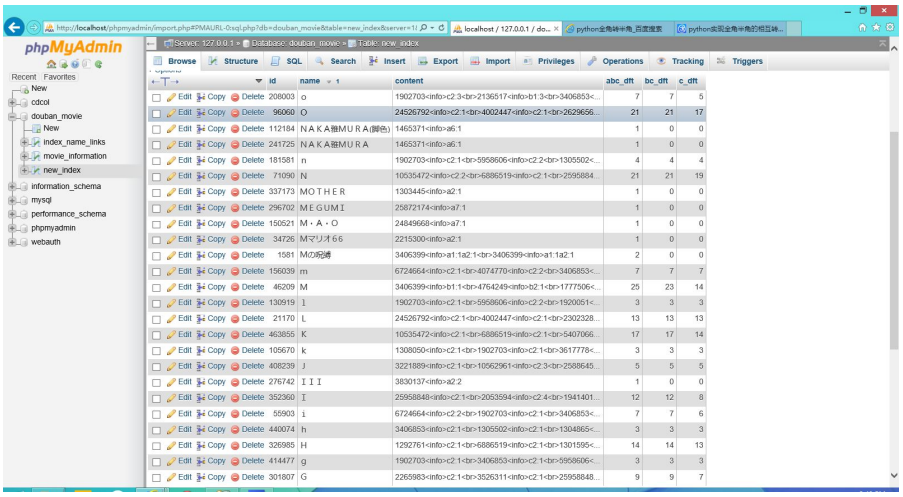
"仙履奇缘(港", "台)"

6.4 uncommon signs



```python
SIGN_STRING = ur"""  `-=[]\;',./~!@#$%^&*()_+{}|:"<>?•【】、；‘，。、～！@#￥%……&*（）——+{}|：“《》？”’
àáâãäåæçèéêëìíîïðñòóôõöøùúûüýÿ ¯ ˘ ¨ ‘’ 々～‖：”’ ‘｜〃（）《》「」『』．〖〗【】（）〔〕｛｝
≈≡≠＝≤≥＜＞≮≯∷±＋－×÷／∫∮∝∞∧∨∑∏∪∩∈∵∴⊥∥∠⌒⊙≌∽√•
°′〃$￡￥‰％℃￠￡§№☆★○●◎◇◆□■△▲※→←↑↓〓#&@\^_…•－＝【】\；‘，。、～！@#￥%……&×
）——+｛｝｜：“《》？”’ *
"""
SIGN_SET = set(SIGN_STRING)
def isSign(s):
    for ch in s:
        if ch in SIGN_SET:
            return True
        if ch.encode("utf-8") in SIGN_SET:
            return True
    return False
```

6.5 half-pitch and full-pitch!!



```python
def strQ2B(ustring):
    """全角转半角"""
    rstring = u""
    for uchar in ustring:
        inside_code = ord(uchar)
        if inside_code == 12288:                          # 全角空格直接转换
            inside_code = 32
        elif inside_code >= 65281 and inside_code <= 65374:  # 全角字符（除空格）根据关系转化
            inside_code -= 65248

        rstring += unichr(inside_code)
    return rstring
# print strQ2B(u"ＮＡＫＡ雅ＭＵＲＡ(脚色) 是 速度ａｂｓｄ（）｛｝【】")
```

movie_information

Column	Type	Null	Default	Comments	MIME
id	int(11)	No			
title	varchar(1024)	No			
alias	text	No			
year	char(10)	Yes	*NULL*		

director	varchar(512)	Yes	*NULL*		
rating	char(10)	Yes	*NULL*		
comment_amount	int(11)	No			
betterthan	text	No			
intro	text	Yes	*NULL*		
link	char(100)	No			
writer	text	Yes	*NULL*		
actor	text	Yes	*NULL*		
info	text	Yes	*NULL*		

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	id	31408	A	No	
link	BTREE	Yes	No	link	31408	A	No	
link_search	BTREE	No	No	link		A	No	
rating_search	BTREE	No	No	rating		A	Yes	

new_index

Column	Type	Null	Default	Comments	MIME
id	int(11)	No			
name	char(35)	No			
content	mediumtext	No			
abc_dft	int(6)	No			
bc_dft	int(6)	No			
c_dft	int(6)	No			

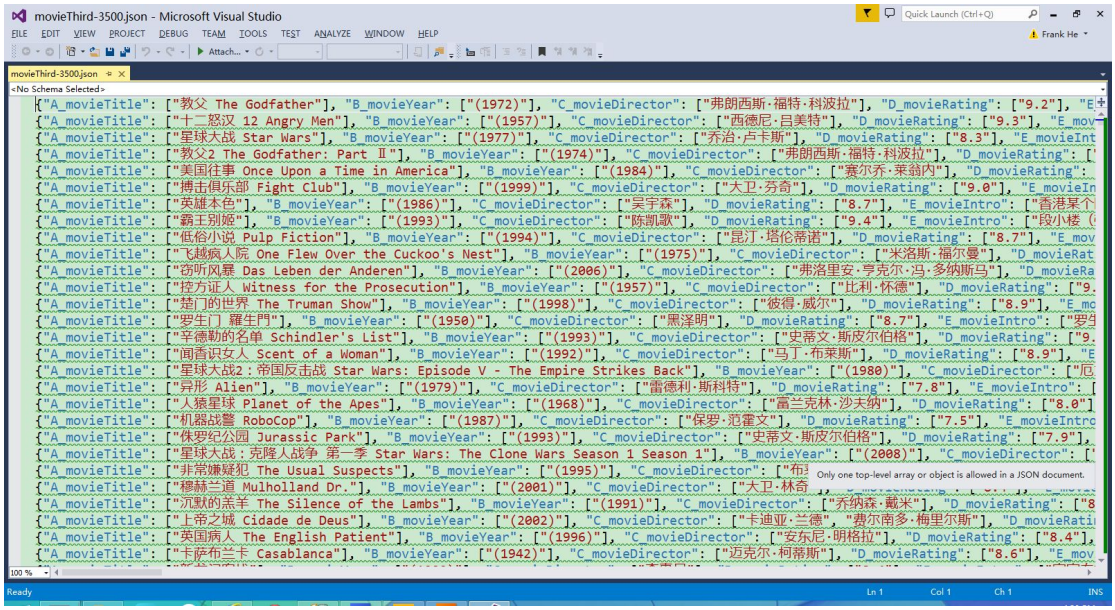
Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	id	459442	A	No	
name	BTREE	Yes	No	name	459442	A	No	
name_search	BTREE	No	No	name		A	No	

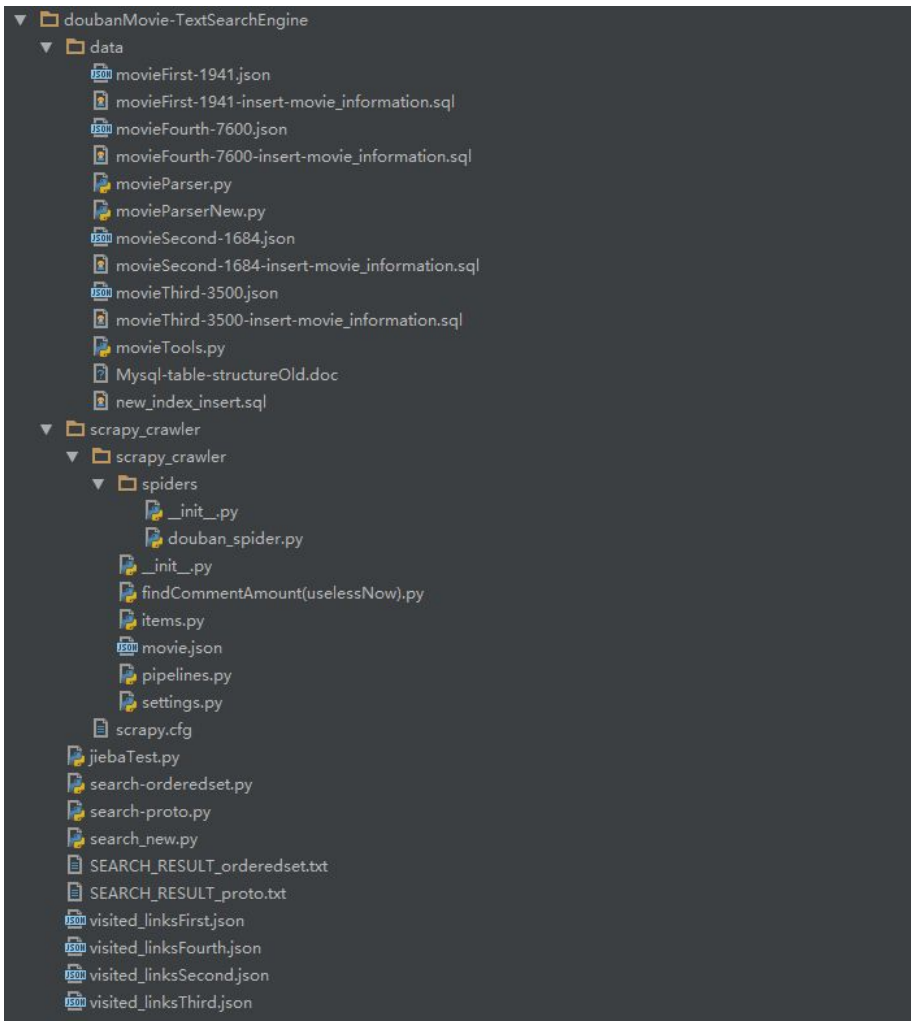
Index:

3	美满生活	12.7989	http://movie.douban.com/subject/1483239/<info>c2
4	楼兰	12.7989	http://movie.douban.com/subject/6721670/<info>c2/c2
5	RollsRoyce	0.0000	http://movie.douban.com/subject/5044456/<info>a3
6	跑步	12.7989	http://movie.douban.com/subject/10502527/<info>c2
7	Marie-Anne Fliegel	0.0000	http://movie.douban.com/subject/2213597/<info>a7
8	Jamie Oliver	0.0000	http://movie.douban.com/subject/2269936/<info>a7
9	小腿	12.7989	http://movie.douban.com/subject/3153640/<info>c2
10	未作	11.2137	http://movie.douban.com/subject/5383525/<info>c2 http://movie.douban.com/subject/4312428/<info>c2 http://movie.douban.com/subject/1899664/<info>c2
11	埋名	8.9915	http://movie.douban.com/subject/11601131/<info>c2 http://movie.douban.com/subject/6875610/<info>c2 http://movie.douban.com/subject/4845728/<info>c2 http://movie.douban.com/subject/1418660/<info>c2
12	129分钟(中国大陆)	0.0000	http://movie.douban.com/subject/10485647/<info>b3
13	hanging	12.7989	http://movie.douban.com/subject/3182807/<info>c2
14	电视频道	10.7985	http://movie.douban.com/subject/3835489/<info>c2 http://movie.douban.com/subject/25830985/<info>c2 http://movie.douban.com/subject/3581466/<info>c2 http://movie.douban.com/subject/4944009/<info>c2

Movie information:



Project:



7. Expectations

1. Words filter parser.

   Some words like “йцук”, “ðšè” will never be searched by Chinese and English users, thus storing those words is waste of space.

2. Actor and director evaluation system

   To make a good ranking system, actor and director evaluation system is very helpful.

3. Use a movie to search the movies

   Given a already known movie, we can search movies by calculate the similarity between two movies.

4. Word expansion. Word correction.
5. Multi-cache or memory.
6. Multi-thread web server.