

Implement a vertical search engine on film with domain-combination search

Yang Bao, 2020/4/07

Github link:

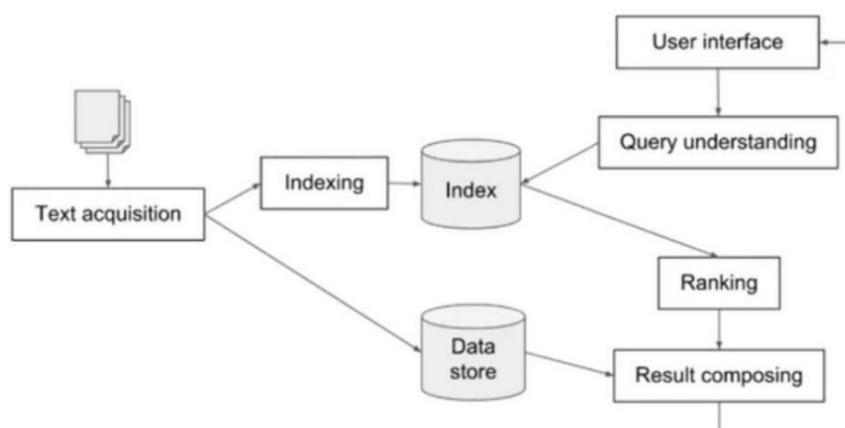
<https://github.com/baoyangisapig/Final-Project-for-information-retrieval>

Abstraction

MetaCritic is a movie review site(<https://www.metacritic.com/>). For each movie, the site would provide the averaged scores by film critics and movie viewers. Also, we can review all the details about a movie including the comments and the summary. As to the search engine of the site, the functionality is basic. This is to say, we can only search for the name of a film. I want to add more functionality to the site to improve the search engine. The improvement is called a combination search.

The information retrieved from the website has several domains: the name of the film, director, movie poster image, comments, a summary of the movie plot, the user score and the professional score. In my opinion, These three domains have more useful information and can be used to locate a movie: name, comments and the summary of a specific movie. So I would implement two kinds of search models: the first kind is searching in one of the three domains and find the most related films, and the second one is to do a search in multiple domains. Of course, the second model is based on the first one. In the following article, I will show details on how to implement the functionality in a search engine.

Here is the basic structure of the information retrieval system



Preparation before designing the system

1. Write a web crawler and clean origin data.

Before we implement the search system, we need to implement a web crawler to get the data we need. I use Selenium to implement the crawler.

Selenium is an open-source tool that automates web browsers. It provides a single interface that lets you write test scripts in programming languages like Ruby, Java, NodeJS, PHP, Perl, Python, and C#, among others.

A browser-driver then executes these scripts on a browser-instance on your device (more on this in a moment).

This is how web crawler works. Firstly, we write a spider script and execute it with web driver. Then we can get the crude data from web. After cleaning the data, we can get organized data. The final step is to write it down in files and store it.

Core Code in the crawler:

```
try {
summary=driver1.findElement(By.xpath("//*[@id=\"main_content\"]/div[1]/div[2]/div[1]/div/div[1]/div[2]/div[2]/div[4]/span[2]/span")).getText();

metascore=driver1.findElement(By.xpath("//*[@id=\"main_content\"]/div[1]/div[1]/div/table/tbody/tr/td[2]/div/table/tbody/tr/td[1]/div/div/div[2]/table/tbody/tr/td[2]/a/span")).getText();

userScore=driver1.findElement(By.xpath("//*[@id=\"main_content\"]/div[1]/div[1]/div/table/tbody/tr/td[2]/div/table/tbody/tr/td[1]/div/div/div[3]/div/table/tbody/tr/td[2]/a/span")).getText();
    List<WebElement> elements=driver1.findElements(By.className("summary"));

director=driver1.findElement(By.className("director")).findElements(By.tagName("span")).get(1).getText();
    for(int k=0;k<=elements.size()-1;k++){
        if (k==0) continue;
        else comment.append(modify(elements.get(k).getText())).append("&&");
    }
}
catch (Exception e){
    continue;
}
```

2. Parse XML file.

Here is the data harvested in the website. I store these data in XML files. So I need to read and parse the files to get organized data.

This is the structure of the XML files. I read the data line by line and get the value of each domain with BufferedWriter.

```

number: 1
title: Hoop Dreams
director: Steve James
image: https://static.metacritic.com/images/products/movies/2/18e6a88baf392f8b0f84213eed85be13-98.jpg
summary: Two inner-city Chicago boys with hopes of becoming professional basketball players struggle to become college players.
metascore: 98
userScore: 8.0

```

Explanation of each domain in the XML file:

- ①Number: the index of a specific film.
- ②Title: the name of the film.
- ③Director: director of the movie.
- ④Image: the URL of the movie poster image.
- ⑤Summary: the summary of the movie plot.
- ⑥metaScore: the professional score of the movie
- ⑦userScore: user score of a specific movie
- ⑧comment: movie comments

Steps and explanation on how to implement the IR algorithm

1. construct the inverted index.

Steps to build an inverted index:

①Fetch the Document

Removing of Stop Words: Stop words are most occurring and useless words in document like “I” , “the” , “we” , “is” , “an” .

②Stemming of Root Word

Whenever I want to search for “cat”, I want to see a document that has information about it. But the word present in the document is called “cats” or “catty” instead of “cat”. To relate the both words, I’ll chop some part of each and every word I read so that I could get the “root word”. There are standard tools for performing this like “Porter’s Stemmer”.

③Record Document IDs

If word is already present add reference of document to index else create new entry. Add additional information like frequency of word, location of word etc.

④ Repeat for all documents and sort the words.

The algorithm is based on multiple domains, so we need to construct an inverted index for each domain. So, after following the steps above, we would have three inverted indexes: name, comment and summary.

2. implement BM25 algorithm to get the basic score of each document.

The ranking function is based on BM25 algorithm but I will adjust the weight of each variable and add some new variables.

3

This part requests us to add a ranking function to the existed system and evaluate the performance of the whole search engine.

First, we have already built the inverted indexes. Based on the given data. The next thing we should focus on is that how to SCORE a document according to the input query. The higher score one document gets, the higher possible it is required by the user.

Second, we need to deal with the SCORE. Based on the textbook and notes on the class, I choose BM25 algorithm as a "SCORE" function to evaluate the degree of a document fits for the query. The reason I choose BM25 is easy to implement and it performs well on many existed models. Besides, there are a few parameters for programmers to adjust due to a specific document.

Third, we need to implement the BM25 algorithm. BM25 comes from the idea of TF-IDF. It solves the problem in the traditional TF-IDF that TERM frequency will influence the performance when it grows big enough.

```
TF-IDF TF Score = sqrt(tf)
BM25 TF Score = ((k + 1) * tf) / (k + tf)
```

with the help of the parameter k, it helps us to limit the influence from the size of tf. Now, Let's dive deep into it. Below is the method to calculate the BM25 score (here we just omit a large sum of mathematical justification).

$$R(q_i, d) = \frac{f_i \cdot (k_1 + 1)}{f_i + K} \cdot \frac{qf_i \cdot (k_2 + 1)}{qf_i + k_2}$$

$$K = k_1 \cdot (1 - b + b \cdot \frac{dl}{avgdl})$$

R is the result we want, q_i means each word in the Query, d means document, f_i means the frequency of q_i occurs in the document d. K is a constant number and we need to calculate the length of the current document dl and the average document length of the whole database avgdl. K means that the longer a document is, the smaller the document's score will be. And we use

$$Score(Q, d) = \sum_i^n IDF(q_i) \cdot \frac{f_i \cdot (k_1 + 1)}{f_i + k_1 \cdot (1 - b + b \cdot \frac{dl}{avgdl})}$$

to calculate the final results for each document.

Finally, we have already known the power SCORE function, bm25, and then we could (1)score every documents (2)use a priorityqueue to get the top-k documents as the result of best fit document (3)return document ids to the user. Another thing is to separate the input query into a bag of words. This procedure can be ascribed as the following steps: (1)change them into a word array (2)find all stop words (3)there may be phrase in the query, so pick them out and deal with them separately (4)some words may not exist in the documents so find a proper way to deal with them (5)rearrange the score list and return the result to the user.

3. implement single-domain search

This part is very easy. We have implemented the BM25 algorithm to calculate the score of each film. Then we use a min Heap to sort these film by their score. So we can provide a Restful API to select the top K films given s specific movie id.

So we can get the top K movies in a single domain. For example, the query is “teenager love”, we can search the keywords in comments and get the top K results based on the algorithm above.

4 implement combination-domain search

This is the core part of the information retrieval system. I would use three domains to implement the search: name, comment, and summary. When we use the single-domain search, we can know the score of each film. The higher the score, the more related the movie is to the query. So based on the three inverted indexes, we can get three kinds of a score, the scoring base on the film name S_1 , the score based on film summaries S_2 and the score based on film comments S_3 .

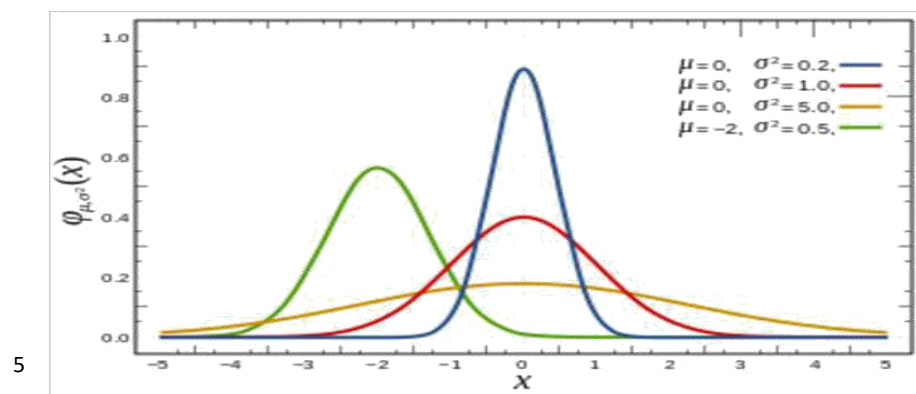
As we know, it is a combination search, the user can search for any kind of information related to the target film, so we need to analyze the intention of users. My assumption is that the closer the length of query keywords is to the average length of movie names, the more likely the user is to query the film by its name. So when the length of query keywords is close to the length of the averaged film name, the weight of S_1 increases in the final score in this query.

Explanation for the assumption: When the user use domain-combination search, there are two possible usage scenarios: the first one is that the user has a target film before search, so he will search the film by name, another one is that the searcher wants to find some interesting films by keywords (exploratory search) without a target one. So the weight assignment method is based on the assumption that: the closer the length of query keywords is to the average length of movie names, the more likely the user is to query the film by its name. With such assumption, I can improve the BM25 algorithm and have a better performance.

Suppose that the average length of film names in the data set is l_1 , then the weight increases when the length of the query approaches l_1 . I use the Gaussian function to simulate the process.

This is the gaussian function. σ is the standard deviation and μ is the mathematical expectation.

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$



Normalized Gaussian curves with expected value μ and variance σ^2 . The corresponding parameters are $a = \frac{1}{\sigma\sqrt{2\pi}}$, $b = \mu$, and $c = \sigma$.

From the graph above, we can know that when $x = \mu$, the value of function reaches the highest point. Also, the highest point is smaller than 1. I use the value of gaussian function to simulate the weight of S_1 in the final score.

We let $\sigma = 1$, $\mu = l_1$, so when the length of query is equal to l_1 , the weight of S_1 is the highest, which is the value of $g(x)$. So we can get the calculation formula of final Score.

Based on BM25, the score of a film in single domain is:

$$Score(Q, d) = \sum_i^n IDF(q_i) \cdot \frac{f_i \cdot (k_1 + 1)}{f_i + k_1 \cdot (1 - b + b \cdot \frac{dl}{avgdl})}$$

Define the final score as S_f

$$S_f = k_1 \cdot S_1 + k_2 \cdot S_2 + k_3 \cdot S_3$$

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}.$$

$k_1 = g(x)$, let $\sigma = 1$, $\mu = l_1$. $K_2 = (1 - k_1) \cdot \epsilon_1$, $K_3 = (1 - k_1) \cdot \epsilon_2$, let $\epsilon_1 = 0.4$, $\epsilon_2 = 0.6$.

Based on that, we can get the calculation formula of final score

$$S_f = g(x) * S_1 + (1-g(x)) * \epsilon_1 * S_2 + (1-g(x)) * \epsilon_2 * S_3.$$

Now, we still need to do some improvement on S_f , because the value of S_1, S_2 and S_3 are not in the same range. So we need to do Normalization on the original Score.

$$S_{i,j} = S_{i,j} / \sum_{k=1}^N S_{i,k}$$

So the value of $S_{i,j}$ is also in $[0,1]$. Also, we multiply the result by N to offset the effect of normalization on the result value, in which N is the number of films in the data set.

$$S_f = [g(x) * S_1 / \sum_{k=1}^N S_{1,k} + (1-g(x)) * \epsilon_1 * S_2 / \sum_{k=1}^N S_{2,k} + (1-g(x)) * \epsilon_2 * S_3 / \sum_{k=1}^N S_{3,k}] * N.$$

Base on the calculation formula, we can calculate the final score of each film. With a min Heap, we can sort all these films and get the top K films based on the user's query. This is the result of the combination-domain search.

Core code to calculate the final score:

```
public double combineScore(String recordId, String query, Map<String, Double> weight1,
Map<String, Double> weight2){
    if (k1 == -1)
        k1 = gaussianFunction(query.replaceAll("[^a-zA-Z]", " ").split("\\s+").length);
    double k2 = (1 - k1) * 0.6;
    double k3 = (1 - k1) * 0.4;
    return
k1 * calScore(query, recordId) + k2 * scoreOfDocument(weight2, recordId, wordFreqForSummary, summary) + k3 * scoreOfDocument(weight1, recordId, wordFreqForComment, comment);
}
```

Implement the search engine based on improved ranking algorithm.

I implement the core part of the search engine without open-source code. The search engine is based on Restful APIs in the back-end part of the website. When the user inputs the query in the search bar, with the Ajax technology, the front end would be sent an HTTP request to the back-end with necessary parameters. The back end calls the corresponding Restful API with the ranking service, then the MVC controllers of the back end part would format the return value as JSON objects. Finally, the message of all the top-ranking items would be parsed and represented as images and film descriptions on the front page.

As to the web frameworks, I implement the front-end part of the website with Bootstrap and implement the back-end part with Spring Boot. I integrated them to construct an MVC structure.

As to the user query procession and experience, when we search on the website, the request will be redirected to a Restful API by a corresponding controller in the MVC structure. Based on the information retrieval algorithm and ranking function, we can get the results of a query. All the calculations would be finished in less than 1 second based on the Ajax and Calculation acceleration with HashMap. So the user can get the result immediately.

Core code in the interaction of back end and front end:

Back end:

```
@ResponseBody
@RequestMapping("searchTop")
public String SearchTop(@RequestParam(value="query")String
query,@RequestParam(value="type") String type,@RequestParam("metaScore") String
metaScore,@RequestParam("userScore")String userScore) {
    Queue<Pair<String,Double>> queue=null;
    if (type.equals("1")){
        if (metaScore.equals("")||userScore.equals(" ")) {
            queue=Modified_BM25.selectTopByName(query,6,40,4);
        }
        else
            queue=Modified_BM25.selectTopByName(query,6,Double.parseDouble(metaScore),Double.parseDouble(userScore));
    }
    else if (type.equals("2")){
        if (metaScore.equals("")||userScore.equals(" ")) {
            queue=Modified_BM25.selectTopK(query,6,Modified_BM25.wordFreqForComment,Modified_BM25.comment,40,4);
        }
        else
            queue=Modified_BM25.selectTopK(query,6,Modified_BM25.wordFreqForComment,Modified_BM25.comment,Double.parseDouble(metaScore),Double.parseDouble(userScore));
    }
    else if (type.equals("3")){
        if (metaScore.equals("")||userScore.equals(" ")) {
            queue=Modified_BM25.selectTopK(query,6,Modified_BM25.wordFreqForSummary,Modified_BM25.summary,40,4);
        }
        else
            queue=Modified_BM25.selectTopK(query,6,Modified_BM25.wordFreqForSummary,Modified_BM25.summary,Double.parseDouble(metaScore),Double.parseDouble(userScore));
    }
    else if (type.equals("4")){
        if (metaScore.equals("")||userScore.equals(" ")) {
            queue= new JointSearch().selectTopK(query,6,40,4);
        }
        else
            queue= new JointSearch().selectTopK(query,6,Double.parseDouble(metaScore),Double.parseDouble(userScore));
    }
}
```



```

    }

    List<Film> list=new ArrayList<>();
    while (!queue.isEmpty()){
        Film film=new Film();
        String key=queue.poll().getKey();
        film.setDirector(Modified_BM25.director.get(key));
        film.setTitle(Modified_BM25.title.get(key));
        film.setImage(Modified_BM25.image.get(key));
        film.setMetaScore(Double.parseDouble(Modified_BM25.metaScore.get(key)));
        film.setUserScore(Double.parseDouble(Modified_BM25.userScore.get(key)));
        film.setComment(Modified_BM25.comment.get(key));
        film.setSummary(Modified_BM25.summary.get(key));
        list.add(film);
    }
    Collections.reverse(list);
    return new JSONArray(list).toString();
}

```

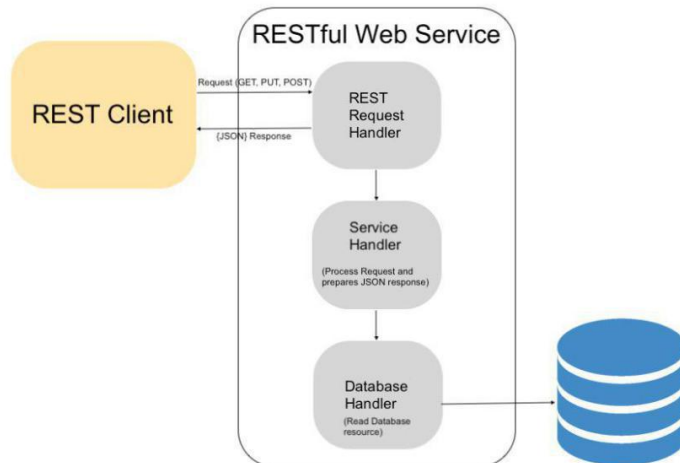
Front end (Ajax):

```

<script>
    window.onload=function (ev) {
        $.post(
            "http://localhost:8080/demo/searchTop",
            {
                query:"big data",
                type: "2",
                metaScore:40,
                userScore:4
            },
            function (data) {
                var data1=JSON.parse(data);
                $('#biao').html("");
                for(i=0;i<data1.length;i++){
                    var title=data1[i].title;
                    var director=data1[i].director;
                    var summary=data1[i].summary;
                    var comment=data1[i].comment;
                    var image=data1[i].image;
                    var metaScore=data1[i].metaScore;
                    var userScore=data1[i].userScore;
                    var str= "<tr><td><br><br><br><br><br><br>" + title +
                        "</td><td><br><br><br><br><br><br>" + director + "</td><td>" + comment +
                        "</td><td><img src='" + image + "' width='380px' height='500px'/><br>" + summary +
                        "</td><td><br><br><br><br><br><br> <div style='position:relative;'><img
                        src='\"https://harperlibrary.typepad.com/.a/6a0105368f4fef970b01b7c8ada1a9970b-800wi\"
                        style='\"width: 100px;height: 100px\"' /><div style='position:absolute; z-index:2;
                        left:34px; top:25px;font-size: 35px;color:
                        greenyellow\"><strong>"+metaScore+"</strong></div></div></td><td><br><br><br><br><br><br>
                        br><br> <div style='position:relative;'><img
                        src='\"https://harperlibrary.typepad.com/.a/6a0105368f4fef970b01b7c8ada1a9970b-800wi\"
                        style='\"width: 100px;height: 100px\"' /><div style='position:absolute; z-index:2;
                        left:34px; top:25px;font-size: 35px;color:
                        red\"><strong>"+userScore+"</strong></div></div></td></tr>";
                    $('#biao').append(str);
                }
            }
        ).error(function (msg) {
            alert("error");
        })
    }
</script>

```

Service-based MVC structure of the website:



Introduction on the user interface and result display.

You can choose search in one domain or in multiple domains. Also, you can set the limit value of userScore and metaScore, in which the userScore is the average score of users and the metaScore is the average score of professionals.

There are four function of the web interface:

① Search by Name:

The screenshot shows a web browser displaying a search results page for the movie 'Love, Simon'. The search bar at the top has 'Love' entered, and the search button is labeled 'Search'. Below the search bar, there is a table with columns: Film name, Director, Comment, Movie poster and summary, metaScore, and userScore. The first row shows the movie 'Love, Simon' by Greg Berlanti. The comment section contains several paragraphs of text. The movie poster is displayed next to the comment. The metaScore is 88% and the userScore is 8.1. The bottom of the page shows a file explorer with various files.


Film name	Director	Comment	Movie poster and summary	metaScore	userScore
Love, Simon	Greg Berlanti	<p>Director Greg Berlanti, who has helmed a string of hit television shows as producer and writer, uses the familiar teenage romance genre to tell an LGBTQ story, and in so doing makes these tropes feel fresh, fun, entertaining.</p> <p>It?? not exactly like the novel, but it captures the best parts of it.</p> <p>There?? some real, weird fun in secondary characters like Tony Hale?? desperate-to-be-down principal, Natasha Rothwell?? exasperated drama teacher, and Logan Miller?? Martin, a theater kid so eager to please he practically turns himself inside out.</p> <p>It??? a canny blend of ??grassi?? and John Hughes, but here the kids mostly behave like angels. Love, Simon is the rare, feel-good gay movie.</p> <p>As it turns out, the Ferris wheel is the other perfect parallel to Love, Simon, not the most thrilling ride in the park, a little slow, utterly predictable, perhaps even welcoming the label of ??oring.?? But like the chorus of a latter-day Taylor Swift song, it will lift you up.</p>		88%	8.1

② Search By Movie Comments:

localhost:8080/demo/logindemo

应用 唯品会 淘宝网 天猫商城 聚划算 京东商城 苏宁易购 百度一下 My Subscriptions... Dianrong JIRA git rebase 冲突解... rebase(merge的... My issue

FILM Search by Comment teenager love 60 6 Search

Film name	Director	Comment	Movie poster and summary	metaScore	userScore
Circle of Friends	Pat O'Connor	<p>Their sweet, determined, gently understated struggle for fulfillment in a superstitiously conservative society makes this densely, deftly packed movie a quiet joy to behold.</p> <p>Driver's tough core of honesty and wit is bewitching. So's the movie.</p> <p>Although there isn't anything startlingly original in this tale of three Catholic girls falling in love in late-fifties Ireland, it gets a sweet telling in Pat O'Connor's pretty film.</p> <p>Naturally charming without being beautiful, Driver brings extraordinary intensity and tenderness to a role that easily could have become sappy.</p> <p>As Benny, a small-town Irish teenager in the '50s who goes off to university in Dublin, Minnie Driver has a touchingly awkward prettiness. Her jaw may be as square as a picture frame, but her smile lights her up from within.</p> <p>As Benny (short for Bernadette), a big-boned, headstrong lass who strains</p>		88%	7

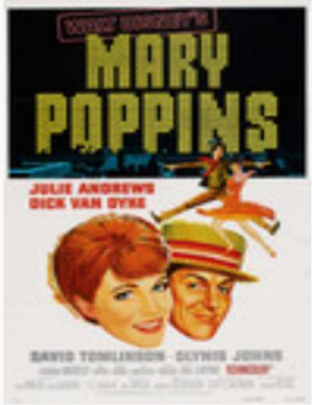
8aa9ff808602c2...svg ER图 (2).png ER图 (1).png ER图.png 24379 GRAPHPRESENT...pdf 全部显示

③ Search By Summary

localhost:8080/demo/logindemo

应用 唯品会 淘宝网 天猫商城 聚划算 京东商城 苏宁易购 百度一下 My Subscriptions... Dianrong JIRA git rebase 冲突解... rebase(merge的... My issue

FILM Search by summary happy family 60 6 Search

Film name	Director	Comment	Movie poster and summary	metaScore	userScore
Mary Poppins	Robert Stevenson	<p>And a most wonderful, cheering movie it is, with Julie Andrews, the original Eliza of My Fair Lady, playing the title role and with its splices and seams fairly splitting with Poppins marvels turned out by the Walt Disney studio.</p> <p>Julie Andrews?? first appearance on the screen is a signal triumph and she performs as easily as she sings, displaying a fresh type of beauty nicely adaptable to the color cameras. Van Dyke, as the happy-go-lucky jack-of-all-trades, scores heavily, the part permitting him to showcase his wide range of talents.</p> <p>What holds Mary Poppins back from being absolutely perfect (as opposed to practically perfect) is due to the episodic structure of the story. Since scenes and songs were drawn from a range of stories from the book series, they play as a series of random adventures, not an organic progression of a single tale.</p> <p>While it doesn't have the soft-edged sense of wonder that the Travers books have, Walt Disney's 1964 version of the Mary Poppins story does manage to avoid the usual saccharine excesses of his live-action work.</p> <p>Mary Poppins is a near-masterpiece. It?? the best of the first wave of Disney live-action features, and the most complete and satisfying musical of any kind that the studio produced until Beauty...</p>		82%	8.2

8aa9ff808602c2...svg ER图 (2).png ER图 (1).png ER图.png 24379 GRAPHPRESENT...pdf 全部显示

④ Domain-Combination Search

When the user search “artificial intelligence” in the combination-search engine. He can get some movies related to the topic of AI.




social.pdf x PowerPoint x PowerPoint x slides_chap x mc059900 x To appear x To appear x 谷歌翻译 x PDF转Word x Title x + -

localhost:8080/demo/logindemo

应用 唯品会 淘宝网 天猫商城 聚划算 京东商城 苏宁易购 百度一下 My Subscriptions... Dianrong JIR

artificial intelligence 2/4

My issue 88%

Film name	Director	Comment	Movie poster and summary	metaScore	userScore
Demon Seed	Donald Cammell	<p>Excellent performances and direction (Donald Cammell), from a most credible and literate screenplay (from a novel by Dean R. Koontz), make production an intriguing achievement in story-telling.</p> <p>Though the clumsy geometric tentacle that does most of the machine?? evil will cries out for morphing, this is remarkably prescient in its tackling of issues the cinema is only now catching up with, and Christie adds depth to the lady-in-peril heroine. Well worth reassessment.</p> <p>Underrated science-fiction thriller about a superintelligent thinking machine, Proteus IV, designed by obsessive computer wizard Alex Harris (Fritz Weaver).</p> <p>How did Cammell convince a studio to back a movie in which Julie Christie is violated by what looks like a copper Rubik's snake? Better not to ask, or to dwell on the film's less savory aspects, and soak in its moments of visionary hysteria, including the pulsating geometry of images borrowed from experimental filmmaker Jordan Belson.</p> <p>Demon Seed might have been a genuinely witty and terrifying thriller if someone had taken advantage of the story's glaring sadomasochistic implications. Nevertheless, Cammell plays it dumb at a thematic level, ignoring the sci-fi sexual bondage satire staring him in the face. [08 Apr</p>	 <p>A scientist creates Proteus—an organic super computer with artificial intelligence which becomes obsessed with human beings, and in particular the creators wife.</p>		

Implementaear....docx CACM-Jun-2018....pdf

全部显示




social.pdf x PowerPoint x PowerPoint x slides_chap x mc059900 x To appear x To appear x 谷歌翻译 x PDF转Word x Title x + -

localhost:8080/demo/logindemo

应用 唯品会 淘宝网 天猫商城 聚划算 京东商城 苏宁易购 百度一下 My Subscriptions... Dianrong JIR

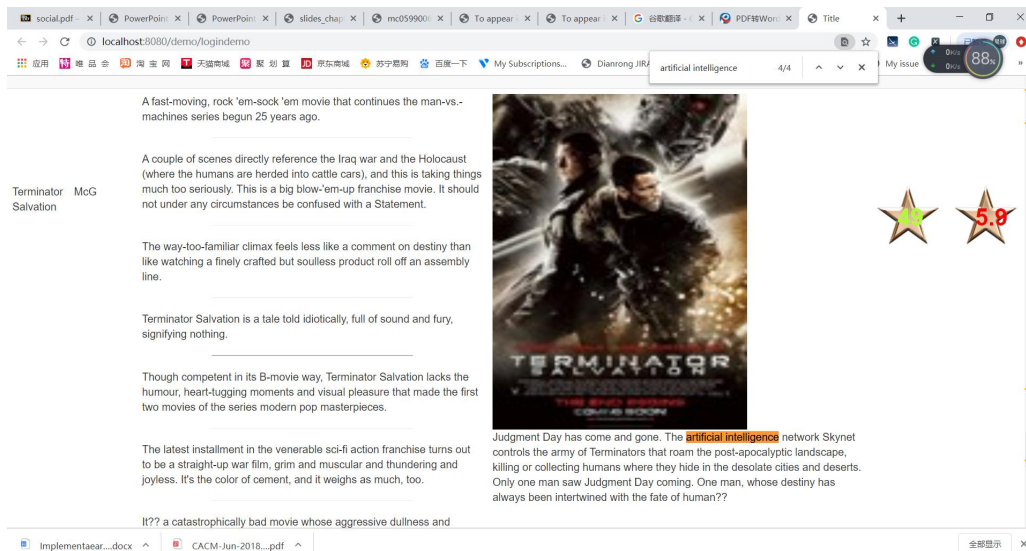
artificial intelligence 3/4

My issue 88%

Marjorie Prime	Michael Almereyda	<p>The sci-fi chamber drama Marjorie Prime is exquisite ?? beautiful, intense, shivering with empathy.</p> <p>Leave it to Michael Almereyda (Experimenter) to make a science fiction movie that consists of little more than scenes of two characters talking in plushly appointed living rooms.</p> <p>There?? more going on in this movie?? 90-plus minutes than in many summer blockbusters nearly twice its length.</p> <p>It?? a haunting little film that ends with a somewhat overwhelming poignancy.</p> <p>It?? an exquisitely challenging production, one that calls for repeat viewings over years, all the better to persuade the film to surrender its meaning.</p> <p>Marjorie Prime is superbly acted, and it?? certainly interesting. Hamm strikes a wonderful balance as a talking re-creation that feels almost human, and the rest of the cast is equally nuanced.</p> <p>Almereyda?? feature is rich in acting talent, but this stagey, flat drama can?? match the wattage of its leads.</p>	 <p>In the near future, a time of artificial intelligence, 86-year-old Marjorie (Lois Smith)?? jumble of disparate, fading memories??as a handsome new companion (Jon Hamm) who looks like her deceased husband and is programmed to feed the story of her life back to her. What would we remember, and what would we forget, if given the chance?</p>		
----------------	-------------------	--	---	---	---

Implementaear....docx CACM-Jun-2018....pdf

全部显示



The users can use the four kinds of search function to find their target film in the search engine. They will get all the related film after clicking the search button. There are several components of a film's description: film name and poster, director, comment, summary and film scores given by professionals and general users.

These are the target films that the system think the user may interested in based on the input query.