INFO300 Term Project

1. Introduction

1.1 数据来源

Computerphile是视频网站YouTube上的一个介绍计算机科学的频道 (channel)。在其每一期视频视频中，都会邀请一名计算机科学领域的学者，对该领域中一些有趣或重要的问题进行解释和分析。
在YouTube的视频播放页中，会展示视频的标题、上传日期、播放量、点赞数、评论数和视频简介，通过这些信息，我们可以对视频的内容和质量 (受欢迎程度) 等信息有一个大致的了解。
因此，利用网络爬虫，我们可以构建一个主题为"Computerphile上传到YouTube的讲座"的数据集合，经筛选后共有747条与讲座对应的合法的数据项，利用这些数据，我们将尝试帮助观众们找到更多他们需要的讲座内容。

1.2 项目架构

该项目由三个主要部分构成：ElasticSearch搜索引擎、Web后端和Web前端。
我们利用Scrapy和Selenium构建网络爬虫，以从YouTube上收集所需数据，然后将数据上传到ElasticSearch搜索引擎并索引。
当来自用户的请求到达Web后端后，Web后端将把请求内容放入事先编写好的ElasticSearch查询中，并调用ElasticSearch Python API将查询发送给ElasticSearch搜索引擎，并将搜索引擎返回的结果处理为Web前端可用的形式。
最终呈现给用户的网页效果由Web前端控制，这一部分需要调整的内容包括网页中呈现内容的取舍、网页的样式和网页的操作逻辑等。

1.3 团队分工

- 余正阳: 前端编程，Flask后端编程，评估系统性能，计算MAP
- 胡纪甚: 数据录入，web后端编程
- 王川石: 数据收集、数据清洗、编写ElasticSearch查询

2. Data Collection

2.1 Tools Introduction

我们利用网络爬虫框架 (网络刮削工具? Web scraping tool) Scrapy从互联网上获取项目所需的数据。因为我们要爬取的网站YouTube使用JavaScript动态渲染页面，所以在网站返回的HTML中并不包含网页中所呈现的大多数信息。为了解决动态页面内容获取的问题，我们利用了Selenium。
Selenium是一个自动化测试工具，可以通过它用Python代码驱动浏览器执行点击、下拉等动作，以触发网页中的一些回调将需要的内容渲染出来；在渲染完成后，Python程序还可以通过Selenium获取加载完成的HTML，这样就解决了从动态网页中获取内容的问题。借助Selenium完成动态页面的加载，Scrapy就能像处理静态页面一样处理动态网页中的内容。

2.2 Codes and Explanation

为了创建一个Scrapy项目，首先需要运行如下命令：

```
scrapy startproject computerphile
```

为了使项目能够运行，需要在spider目录下创建文件computerphile.py，其中定义了该爬取任务所需要的spider，`ComputerphileSpider`，为`scrapy.Spider`的子类。computerphile.py中内容如下：

```
import re
import time
import scrapy
from selenium import webdriver
from scrapy.http import HtmlResponse
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys


class ComputerphileSpider(scrapy.Spider):
    name = "computerphile"

    def start_requests(self):
        # 创建浏览器对象
        option = webdriver.ChromeOptions()
        chrome_prefs = dict()
        option.experimental_options["prefs"] = chrome_prefs
        chrome_prefs["profile.default_content_settings"] = {"images": 2}
        chrome_prefs["profile.managed_default_content_settings"] = {"images": 2}
        browser = webdriver.Chrome(options=option)

        # 访问主页并下拉加载所有所需元素
        main_page_url = "https://www.youtube.com/user/Computerphile/videos"
        browser.get(main_page_url)
        while True:
            time.sleep(1)
            browser.find_element(By.TAG_NAME, 'body').send_keys(Keys.END)
            page = HtmlResponse(url=main_page_url, body=browser.page_source.encode())
            load_ring = page.css("#contents > ytd-continuation-item-renderer")
            if len(load_ring) == 0:
                break

        # 获取主页上视频URL并逐一爬取
        main_page_response = HtmlResponse(url=main_page_url,
body=browser.page_source.encode())
        videos = main_page_response.css("#video-title-link::attr(href)")
        yield from main_page_response.follow_all(videos)
        browser.close()

    def parse(self, response, **kwargs):
        # 获取视频标题
        title = response.css("#title > h1 > yt-formatted-string::text").get()

        # 获取播放量和发布时间
        tooltip = response.css("#tooltip::text").getall()
        views_and_date = [x for x in tooltip if '次观看' in x][0]
        views_date_pair = views_and_date.split('•')
```

```
        # 获取播放量
        views = int("".join(re.findall("[0-9]+", views_date_pair[0])))
        # 获取发布时间
        upload_date = "-".join(map(lambda x: x if len(x) >= 2 else '0' + x,
re.findall("[0-9]+", views_date_pair[1])))

        # 获取点赞数
        likes_str = response.css(
            "#segmented-like-button > "
            "ytd-toggle-button-renderer > "
            "yt-button-shape > button > "
            "div.cbox.yt-spec-button-shape-next--button-text-content > "
            "span::text").get()
        try:
            likes = int(likes_str)
        except ValueError:
            likes = int(float(likes_str[:-1]) * 10000)

        # 获取评论数
        comments = int(''.join(response.css("#count > yt-formatted-string > span:nth-
child(1)::text").get().split(',')))

        # 获取视频简介
        intro_raw = response.css("#description-inline-expander > yt-formatted-string
*::text").getall()
        intro_reduced = map(lambda x: x.replace('\n', '').replace('\r', ''),
intro_raw)
        intro_list = [x for x in intro_reduced if len(x)]
        introduction = '\n'.join(intro_list)

        # 一条完整记录
        result = {
            "title": title,
            "upload_date": upload_date,
            "views": views,
            "likes": likes,
            "comments": comments,
            "introduction": introduction
        }
        return result
```

`ComputerphileSpider` 有两个类方法：`start_requests` 和 `parse`。其中 `start_requests` 为生成器函数，在访问Computerphile的YouTube主页后，利用Selenium控制浏览器不断下拉页面以将全部视频加载出来，然后从完成加载的页面中获取所有Computerphile投稿视频的URL提供给Scheduler进行后续的下载和处理。`parse` 用于处理来自Downloader Middlewares的response，它将response中的内容解析为字典形式并返回，并将由Item Pipeline生成我们最终收集的数据。
如前文所述，因为动态渲染的存在，传递给 `parse` 的response中包含的内容是借助Selenium完成加载的，在Scrapy中，这可以通过编写一个包含了调用Selenium进行加载的逻辑的Downloader Middleware实现，这意味着需要在文件middlewares.py中添加以下代码以创建 `ComputerphileMiddleware` 类：

```
import time
from selenium import webdriver
from scrapy.http import HtmlResponse
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
```

```
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC


class ComputerphileMiddleware:
    def __init__(self):
        option = webdriver.ChromeOptions()
        chrome_prefs = dict()
        option.experimental_options["prefs"] = chrome_prefs
        chrome_prefs["profile.default_content_settings"] = {"images": 2}
        chrome_prefs["profile.managed_default_content_settings"] = {"images": 2}
        self.browser = webdriver.Chrome(options=option)

    def __del__(self):
        self.browser.close()

    def process_request(self, request, spider):
        # 访问视频详情页
        url = request.url
        self.browser.get(url)

        time.sleep(1)

        # 展开加载视频简介
        show_intro = self.browser.find_element("css selector", "#expand")
        show_intro.click()

        # 下拉加载评论
        self.browser.find_element(By.TAG_NAME, 'body').send_keys(Keys.END)
        wait = WebDriverWait(self.browser, 120)
        wait.until(
            EC.presence_of_element_located((By.CSS_SELECTOR, "#count > yt-formatted-
string > span:nth-child(1)")))

        # 返回页面内容
        response_body = self.browser.page_source.encode()
        response = HtmlResponse(url=url, body=response_body)
        return response
```

Downloader Middleware接收来自Scheduler的request，并将得到的对应的response交给Spider的 `parse` 方法处理，可见其核心逻辑正是 `process_request` 方法，该方法利用Selenium调用浏览器对request中包含的URL进行访问，控制浏览器渲染需要的内容，并取得渲染完成的页面HTML，渲染结果包装成 `HtmlResponse` 对象并返回。
为了使 `ComputerphileMiddleware` 生效，在settings.py中需要有如下内容：

```
DOWNLOADER_MIDDLEWARES = {
   'computerphile.middlewares.ComputerphileMiddleware': 543,
}
```

运行如下命令，即可进行对Computerphile的视频信息的爬取，爬取结果将保存于文件computerphile.csv中：

```
scrapy crawl computerphile -O computerphile.csv
```

3. Baseline IR System

3.1 Imported data into elasticsearch

将数据处理完之后，我们便开始对网站的后端进行搭建，而第一步便是将清洗好的数据放入elasticsearch中。
我们在本地配置了elasticsearch和kibana，分别启动两个程序。
其中elasticsearch的端口号为 `9200` ,kibana的端口号为 `5601` 。随后在 `Upload a file` 中将数据文件放入其中即可。

3.2 Build the backend with flask

我们将这个项目的前后端分开进行，后端主要负责接收前端传进来的 `keyword` 并根据不同的算法查找出elasticsearch中的有效信息，进行一定的加工后发回给前端。
为了完成这个任务，我们建立了一个名为 `search.py` 的文件。使用flask搭建两个页面，主界面 `home` 负责输入需要查找的信息，分界面 `results` 负责展示输出的结果。
算法部分后面会有改进，这里只展现一种算法。
代码如下：

```
from flask import Flask, url_for
from flask import request
from flask import render_template
from elasticsearch import Elasticsearch
from elasticsearch.connection import create_ssl_context

app = Flask(__name__,  static_url_path='/static')

'''主界面'''
@app.route('/')
def home():
    return render_template('home.html')

es = Elasticsearch(
    ["127.0.0.1"],
    port=9200,
    sniff_on_start=True,
    sniff_on_connection_fail=True,
    sniff_timeout=60
)


'''结果界面'''
@app.route('/search', methods=['get'])
def search():
    keywords = request.args.get('keywords')

    query1 =  {
    "query": {
        "multi_match": {
        "query": "Python game",
        "fields": [
            "title",
            "introduction"
```

```
      ]
    }
  }
}

    res = es.search(index="results", body=query1)
    hits = res["hits"]["total"]["value"]
    return render_template("results.html", keywords=keywords, hits=hits,
doc=res["hits"]["hits"])
```

3.3 Build the front end with HTML and CSS

The front-end is written in HTML and CSS, and the folder structure (including the Flask back-end) is as follows.

```
project
|   search.py
└─────static
|    |   listview.css
|    |   style.css
|
└─────templates
     |   result.html
     |   search.html
```

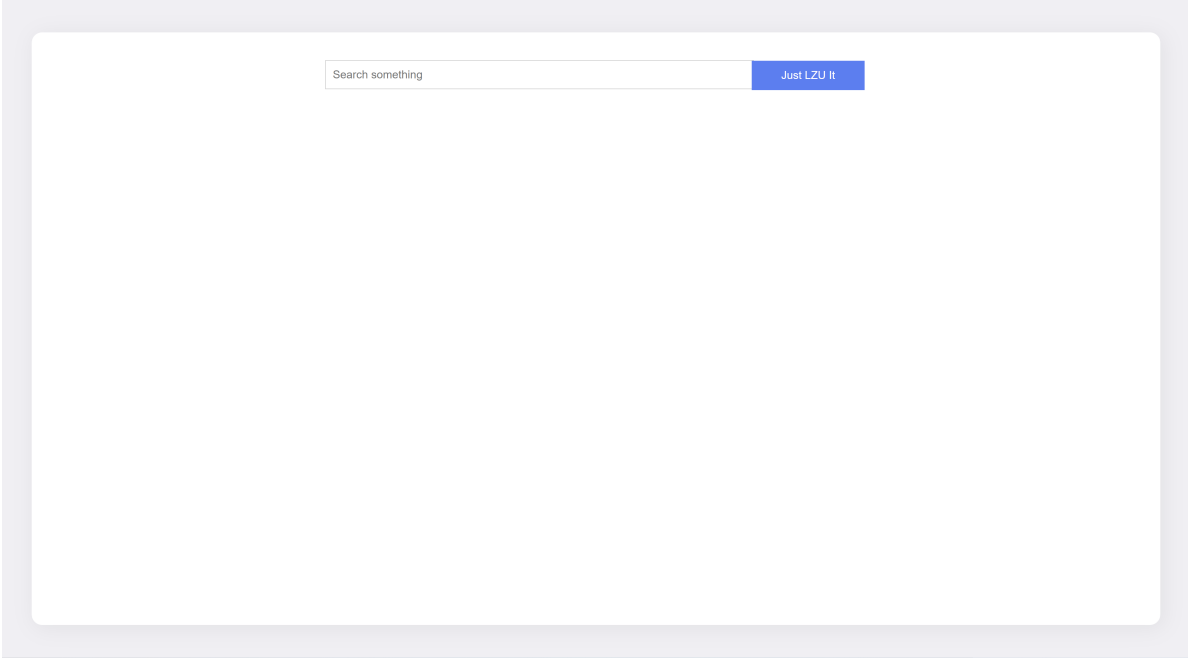
The running process is as follows. First we need to start search.py (as explained in 3.2), and then the program will automatically call search.html.

```
<head>
    <link rel="stylesheet" type="text/css" href="../static/style.css">
</head>

<div class="container">
    <form action="/result" method="get" class="parent">
        <input type="text" id="keywords" name="keywords" placeholder="Search
something">
        <input type="submit" type="button" value="Just LZU It">
    </form>
</div>
```

search.html will use style.css

```
/*此代码为style.css*/
body {
    width: 100%;
    height: 100vh;
    background: rgb(240,239,243);
    margin:auto;
    display: flex;
    align-items: center;
    justify-content: center;
}

.container {
    width: 95%;
    height: 90%;
```

```
    margin: 100px auto;
    background: #fff;
    border-radius: 15px;
    box-shadow: 4px 4px 30px rgba(0, 0, 0, 0.06);
}

.parent {
    width: 100%;
    height: 42px;
    top: 40px;
    left: 0px;
    position: relative;
    /*border: 1px solid #ccc;*/
}

.parent>input:first-of-type {
    width: 38%;
    height: 100%;
    border: 1px solid #ccc;
    font-size: 16px;
    padding-left:10px;
    outline: none;
    left: 26%;
    position: relative;
}

.parent>input:first-of-type:focus {
    border: 1px solid #317ef3;
    padding-left: 10px;
}

.parent>input:last-of-type {
    width: 10%;
    height: 100%;
    position: absolute;
    background: #317ef3;
    border: 1px solid #317ef3;
    color: #fff;
    font-size: 16px;
    outline: none;
    top: 2.5%;
    left: 25.5%;
    position: relative;
}
```



When the user enters content and clicks search, search.py will automatically run result.html. In this html file, we embed python loop code that can be recognized by Flask, so that as many results as the backend returns, as many search items will appear on the frontend

```
<!This file is result.html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>

    <link rel="stylesheet" href="../static/listView.css">
</head>
<body>

<div id="mainContentDiv">

    <div class="mainDivMainImgDiv" >
        <div class="headLeftDiv headLeftDivFont">Search Results</div>
        <div class="link-top"></div>
        <div class="headLineBlowDiv">
            <div class="headLeftDiv">
                Hit<span>【{{HITS}}】</span>results
            </div>
        </div>
    </div>

    <div class="mainDivMainInfoiv">
        <div class="mainInfoSubDiv">
            {% for item in Data %}
                <div class="mainDIvMainInfoDivSubInfoDiv" >
                    <div class="mainDivMainInfoiv_HeadTextDiv " >
                        <div class="mainDivMainInfoiv_HeadTextDiv_TextBox
cardInfoTitle findKey" >
                            {{item['title']}}
                        </div>
                    </div>
                    <div class="mainDivMainInfoiv_mainTextDiv  findKey">
```

```
                        {{item['introduction']}}
                    </div>
                    <div class="InfoDiv_Right_2 rightFlexFont">
                        {{item['date']}}
                    </div>
                </div>
            {% endfor %}
        </div>
    </div>
</div>
</body>
</html>
```

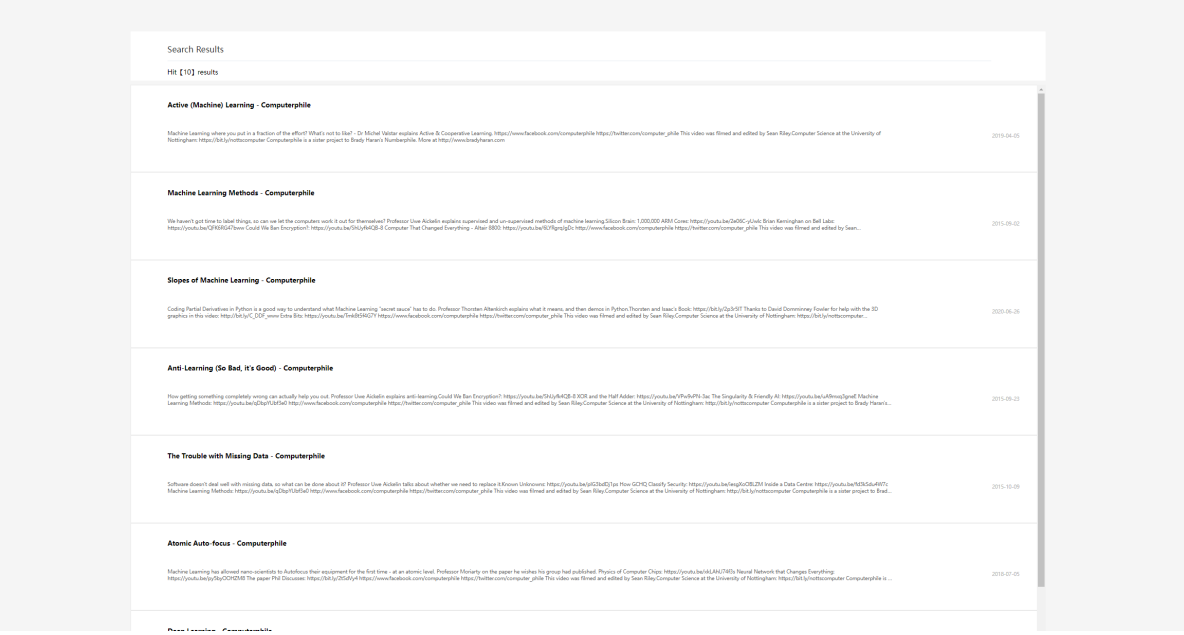
reuslt.html will also use listview.css

```
/*This file is listview.css*/
body {
    background: whitesmoke;
}

#mainContentDiv {
    position: absolute;
    width: 70%;
    height: 100%;
    background: whitesmoke;
    top: 10%;
    left: 10%;
}

.mainDivMainImgDiv {
    position: absolute;
    width: 100%;
    height: 120px;
    background: white;
}

.mainDivMainInfoiv {
    position: absolute;
    width: 100%;
    height: 100%;
    background: whitesmoke;
    top: 130px;
}

.mainInfoSubDiv{
    position: relative;
    width: 100%;
    height: 100%;
    background: whitesmoke;
    overflow-y: auto;
    overflow-x: hidden;
}

.headLeftDiv {
    position: absolute;
    width: 50%;
```

```
    height: 100%;
    left: 4%;
    top: 25%;
}

.headLeftDivFont {
    font-weight: 500;
    /*line-height: 58px;*/
    font-size: 20px;
    color: #333;
}

.headRightDiv {
    position: absolute;
    width: 40%;
    height: 100%;
    right: 2%;
    top: 20%;
}

.link-top {
    position: absolute;
    top: 60%;
    left: 4%;
    width: 90%;
    height: 1px;
    border-top: solid #e8edf3 1px;
}

.headLineBlowDiv {
    position: absolute;
    top: 63%;
    height: 40%;
    width: 100%;
}

.mainDIvMainInfoDivSubInfoDiv {
    position: relative;
    width: 100%;
    height: 13%;
    background: white;
    border: 1px solid #eaeaea;
}

.mainDIvMainInfoDivSubInfoDiv:hover {
    background: rgba(0, 0, 0, 0.05);
}

.mainDivMainInfoiv_HeadTextDiv {
    position: absolute;
    top: 10%;
    left: 4%;
    width: 30%;
    height: 30%;
    background: rgba(0, 0, 0, 0);
```

```
}

.mainDivMainInfoiv_mainTextDiv {
    position: absolute;
    top: 52%;
    left: 4%;
    width: 80%;
    background: rgba(0, 0, 0, 0);
    overflow: hidden;
    text-overflow:ellipsis;
    display:-webkit-box;
    -webkit-box-orient:vertical;
    -webkit-line-clamp:2;
    font-size: 12px;
    color: rgb(102, 102, 102);
}

.mainDivMainInfoiv_HeadTextDiv_TextBox {
    position: absolute;
    top: 25%;
    width: 100%;
    height: 50%;
    background: rgba(0, 0, 0, 0);
}

.cardInfoTitle {
    font-weight: 700;
    /*color: #1f264d;*/
    height: 22px;
    display: inline-block;
    max-width: 600px;
    overflow: hidden;
    text-overflow: ellipsis;
    white-space: nowrap;
    cursor: pointer;
}

.rightFlexFont {
    color: #b3b3b3;
    font-weight: 500;
    text-align: right;
    font-size: 12px;
    color: rgb(179, 179, 179);
}

.InfoDiv_Right_2 {
    position: absolute;
    top: 55%;
    right: 2%;
    width: 18%;
    height: 30%;
    background: rgba(0, 0, 0, 0);
}
```

User finally see the results



3.4 Search work correctly

在这个项目中，我们使用的是`Flask`框架。这是一个基于python编写的轻量级web应用框架。
首先运行如下命令，设置`Flask`的运行环境:

```
set FLASK_EVN=development
```

其次运行如下命令，调入刚刚写好的Python文件:

```
set FLASK_APP=server.py
```

最后运行如下命令，运行`Flask`:

```
flask run
```

在界面中输入想要查询的内容，例如`python game`,返回搜索到的结果。

4. Enhanced IR System

5. Evaluation and Comparison

5.1 Run different queries and make relevant judgments for the top 10 hits

We create a table for each query, each table contains information about the search results of different systems. 1 means the search results are relevant, 0 means the search results are not relevant

Search query: Information search

Rank	system1	system2	system3
1	0	0	0
2	1	0	0
3	1	0	0
4	0	0	1
5	0	0	0
6	0	0	0
7	0	0	1
8	0	0	0
9	1	1	0
10	0	1	0

Search query: Python game

Rank	system1	system2	system3
1	1	1	1
2	1	1	1
3	0	0	0
4	1	1	1
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0	0
10	0	0	0

Search query: Web security

Rank	system1	system2	system3
1	1	0	0
2	1	0	0
3	0	1	1
4	1	1	0
5	0	0	1
6	0	1	0
7	1	0	0
8	1	1	0
9	1	1	1
10	1	0	0

Search query: Machine learing

Rank	system1	system2	system3
1	1	1	1
2	1	1	1
3	1	1	1
4	1	0	0
5	0	0	1
6	0	0	1
7	1	1	1
8	0	0	0
9	1	0	0
10	1	1	0

Search query: Information

Rank	system1	system2	system3
1	1	1	1
2	1	1	1
3	0	0	0
4	0	0	0
5	0	0	0
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0	0
10	0	0	0

Search query: Big data

Rank	system1	system2	system3
1	1	1	1
2	1	1	1
3	1	1	1
4	0	1	1
5	1	0	1
6	0	1	0
7	1	0	0
8	0	0	0
9	0	0	1
10	0	0	0

5.2 Compare the results across 3 systems

- **Our queries cover a wide range of channels**
- **System 1's search yielded the most results**
- **System 3 tends to search for results that System 1 and System 2 can't**
- **The performance of system 2 is relatively mediocre, the relevance, breadth and priority of search results are not as good as system 1 and system 2**
- **However, on some of the more explicit queries, the three systems performed surprisingly well together**

5.3 Calculate MAP for each of the queries on each system

For the sake of accuracy, automation, we wrote a Python code to calculate MAP

```
def MAP(sequence):
    '''
    input: Boolean Sequence
    output: MAP value
    '''
    sum=0
    result=0
    count=0
    for times,item in enumerate(sequence):
        sum+=item
        if item==1:
            count+=1
            result+=sum/(times+1)
    print(result/count)
```

MAP matrix:

Query	System1	System2	System3
Information search	0.500	0.160	0.268
Python game	0.917	0.917	0.917
Web security	0.759	0.478	0.356
Machine learning	0.869	0.814	0.915
Information	1.000	1.000	1.000
Big data	0.903	0.967	0.944
	0.825 (mean)	0.723 (mean)	0.734 (mean)

5.4 Discuss and report the comparison results

- The calculated MAP results are similar to the human-perceived results when we collected the data at the beginning
    - Among the three systems, system 1 performs the best, followed by system 2, and system 3 performs average
- However, from the MAP data, we can also see the different characteristics of the three systems
    - System 1 is characterized by being very stable, with better prediction results in most problems
    - Systems 2 and 3 are characterized by the fact that in some problems, their search results are much better than the other two systems
- Combining the MAP calculations and the conclusions of 5.2
    - System 1 is suitable for most of the searches and can be used as a baseline

- System 3 is suitable for use when the other two systems return single results, and it can often search for results that are not searched by the other systems but are very relevant.
- Systems 2 and 3, which we have improved, are better at searching for certain hot issues and can be used as a complement to System 1