

浙江大学

垂直搜索引擎

“梦奇电影旅行”项目计划书



学生姓名：	<u>胡洋凡</u>	学号：	<u>3180103167</u>
学生姓名：	<u>蔡灿宇</u>	学号：	<u>3180101972</u>
学生姓名：	<u>刘轩铭</u>	学号：	<u>3180106071</u>
学生姓名：	<u>杨凌霄</u>	学号：	<u>3180103608</u>
学生姓名：	<u>王子腾</u>	学号：	<u>3180102173</u>
学生姓名：	<u>王绍兴</u>	学号：	<u>3180106074</u>

2021 年 07 月 18 日

Verse:1.1

版本变更历史

版本号	作者	修订日期	审核者	审核日期	修订原因
1.0	胡洋凡、蔡灿宇、 王子腾、王绍兴、 杨凌霄、刘轩铭	2021.07.18	蔡灿宇	2021.07.18	原始文档
2.0	胡洋凡、蔡灿宇、 王子腾、王绍兴、 杨凌霄、刘轩铭	2021.07.19	蔡灿宇	2021.07.19	样式的统一和部分修改

目录

1. 深度分析：信息聚合	4
1.1 原理和实现.....	4
1.2 效果展示.....	6
2. 深度分析：信息结构化.....	7
2.1 原理和实现	7
2.2 效果展示.....	8
3. 深度搜索：结果聚类 and 排序	9
3.1 原理和实现	9
3.2 效果展示.....	10
4. 深度搜索：基于多种形式的推荐算法.....	12
4.1 原理和实现	12
4.1.1 特征向量	12
4.1.2 召回.....	14
4.1.3 重排.....	15
4.2 效果展示.....	16

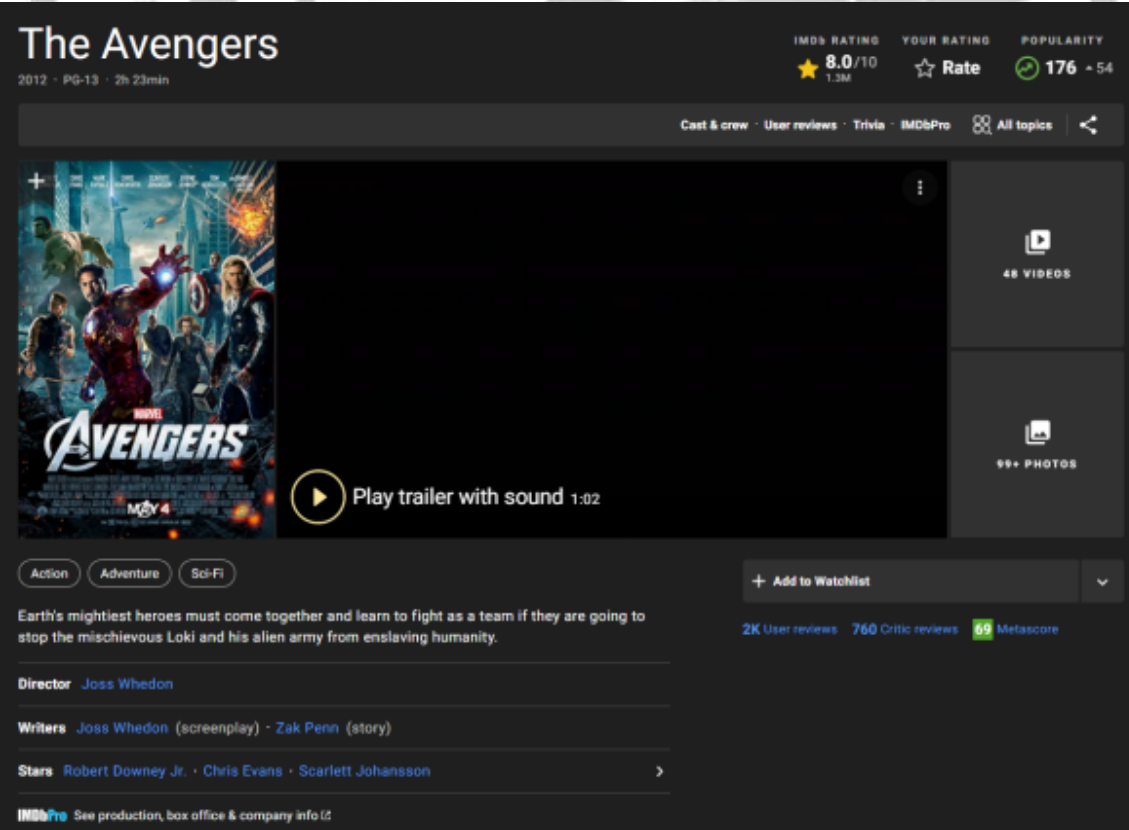
1. 深度分析：信息聚合

由于我们构建的目标是一个基于电影的旅行知识（包括电影取景地、电影音乐等元素）搜索引擎，而目前很难有数据库或网站能一次性获取所需要的资讯信息，所以有必要进行信息的聚合和多次提取整合工作。

1.1 原理和实现

在我们的文档中，需要的数据包括电影的基本信息、取景地信息和背景音乐信息等。

对于基本信息的获取，我们选择 IMDB 这样一个庞大而专业的电影数据网站进行爬取。以电影 Avengers 为例，可以看到这里有我们需要的电影核心内容信息。



The Avengers
2011 · PG-13 · 2h 23min

IMDb Rating: **8.0/10** (1.3M)
Your Rating: ☆ Rate
Popularity: **176** +54

Cast & crew · User reviews · Trivia · IMDbPro · All topics

48 VIDEOS
99+ PHOTOS

Play trailer with sound 1:02

Action Adventure Sci-Fi

+ Add to Watchlist

Earth's mightiest heroes must come together and learn to fight as a team if they are going to stop the mischievous Loki and his alien army from enslaving humanity.

2K User reviews 760 Critic reviews 69 Metascore

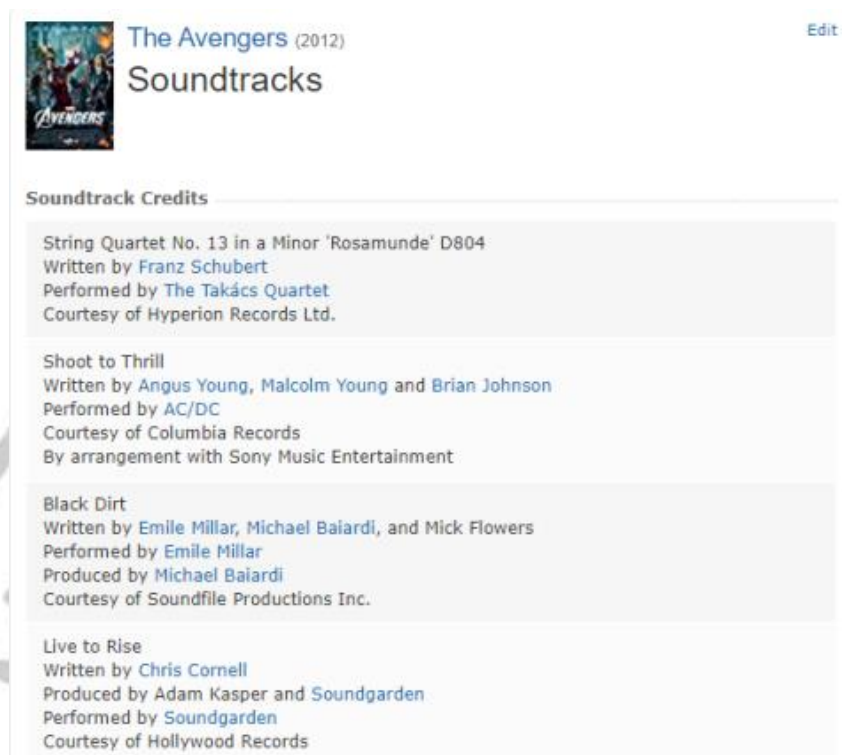
Director: Joss Whedon

Writers: Joss Whedon (screenplay) · Zak Penn (story)

Stars: Robert Downey Jr. · Chris Evans · Scarlett Johansson

IMDbPro See production, box office & company info

此外，IMDB 上还有我们需要的背景音乐信息：



The Avengers (2012)
Soundtracks

Soundtrack Credits

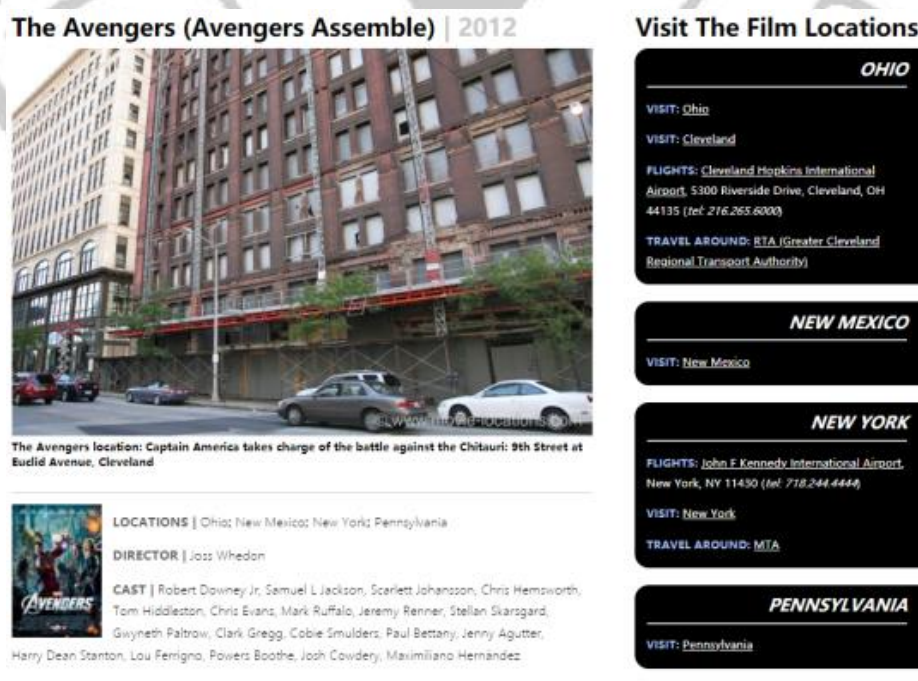
String Quartet No. 13 in a Minor 'Rosamunde' D804
Written by [Franz Schubert](#)
Performed by [The Takács Quartet](#)
Courtesy of Hyperion Records Ltd.

Shoot to Thrill
Written by [Angus Young](#), [Malcolm Young](#) and [Brian Johnson](#)
Performed by [AC/DC](#)
Courtesy of Columbia Records
By arrangement with Sony Music Entertainment

Black Dirt
Written by [Emile Millar](#), [Michael Baiardi](#), and [Mick Flowers](#)
Performed by [Emile Millar](#)
Produced by [Michael Baiardi](#)
Courtesy of Soundfile Productions Inc.

Live to Rise
Written by [Chris Cornell](#)
Produced by [Adam Kasper](#) and [Soundgarden](#)
Performed by [Soundgarden](#)
Courtesy of Hollywood Records

至于片场信息，我们选取了片场网(<https://www.movie-locations.com/>)的信息进行爬取。可以看到，这里不仅有一些我们需要补充的电影信息，还有我们需要的片场信息，以及相应的介绍和图例：



The Avengers (Avengers Assemble) | 2012

The Avengers location: Captain America takes charge of the battle against the Chitauri: 9th Street at Euclid Avenue, Cleveland

LOCATIONS | Ohio New Mexico New York Pennsylvania

DIRECTOR | Joss Whedon

CAST | Robert Downey Jr., Samuel L. Jackson, Scarlett Johansson, Chris Hemsworth, Tom Hiddleston, Chris Evans, Mark Ruffalo, Jeremy Renner, Stellan Skarsgård, Gwyneth Paltrow, Clark Gregg, Cobie Smulders, Paul Bettany, Jenny Agutter, Harry Dean Stanton, Lou Ferrigno, Powers Boothe, Josh Cowdery, Maximiliano Hernández

Visit The Film Locations

OHIO

VISIT: [Ohio](#)

VISIT: [Cleveland](#)

FLIGHTS: [Cleveland Hopkins International Airport](#), 5300 Riverside Drive, Cleveland, OH 44135 (tel: 216.265.6000)

TRAVEL AROUND: [RTA](#) (Greater Cleveland Regional Transport Authority)

NEW MEXICO

VISIT: [New Mexico](#)

NEW YORK

FLIGHTS: [John F. Kennedy International Airport](#), New York, NY 11430 (tel: 718.244.4444)

VISIT: [New York](#)

TRAVEL AROUND: [MTA](#)

PENNSYLVANIA

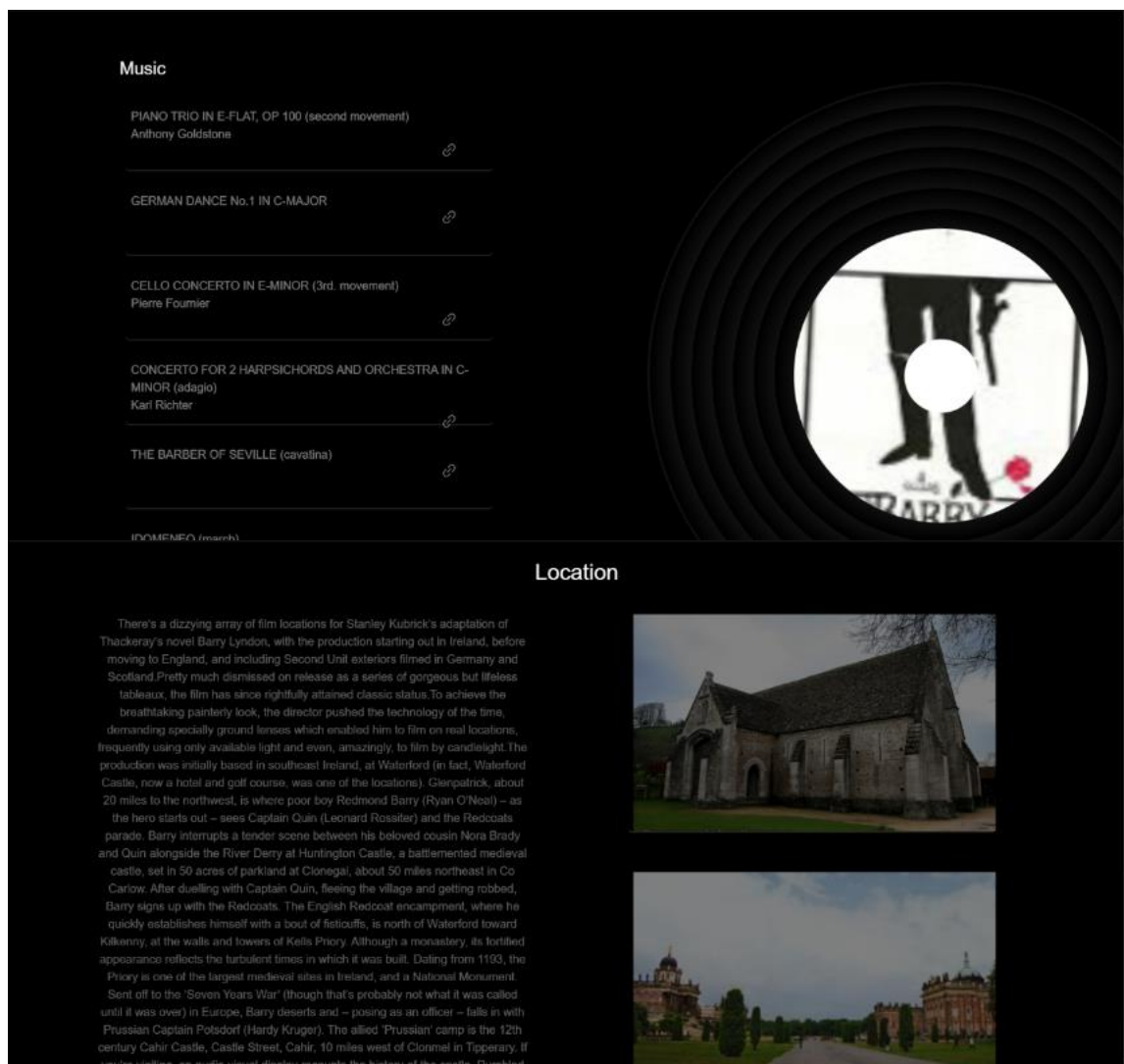
VISIT: [Pennsylvania](#)

1.2 效果展示

#	location	music	name
0	"http://movie-locations.com/movies/6/8-Mile-Schulz"	"Shook Ones Pt. II", "composer": ["Prodigy", "Havoc"],	8 Mile
1	"[\"images\": [], \"visit\": [\"France\"], \"introduction\": \"Jean-Hugues\", \"name\": \"Betty Et Zang\", \"composer\": [\"Gabriel Yared\"], \"name\": \"Betty Blue / 37° 2' Le Marin\"		
2	"http://movie-locations.com/movies/1/5/95Steps-8-Big"	"Rumors Dance", "composer": ["Robert Bath"],	The Thirty-Nine Steps
3	"http://movie-locations.com/movies/6/3-Men-And-A-Bab"	"The Minute I Saw You", "composer": ["Harvin Hallen"],	3 Men And A Baby
4	"http://movie-locations.com/movies/6/28-Days-Later-C"	"East Wastings", "composer": ["Godspeed You! Black Em"],	28 Days Later...
5	"[\"images\": [], \"visit\": [\"Tennessee\", \"New Mexico\"], \"introduction\": \"Can We Get Together\", \"composer\": [\"Renae Vandenberg\"],		21 Grams
6	"http://movie-locations.com/movies/6/21-Days-Later-Bradd"	"Time to Pretend", "composer": ["Andrew VanDerGrinten"],	21
7	"http://movie-locations.com/movies/6/1984-Senate-Hou"	"Oceans, This For Thee", "composer": ["Dominic Mulrow"],	1984
8	"http://movie-locations.com/movies/6/4/Age-Of-Innocenc"	"I Always Chasing Rainbows", "composer": ["Harry Ca"],	Awakenings
9	"http://movie-locations.com/movies/6/Club-De-Sac-Lind"	"", "composer": [{}],	Club De-Sac
10	"http://movie-locations.com/movies/6/Dumb-And-Dumber"	"Boyz n-the-A-Lak", "composer": ["Apache Indian"],	Am Dumb And Dumber
11	"http://movie-locations.com/movies/6/Foux-Sans-Visag"	"", "composer": [{}],	Les Yeux Sans Visage (Eyes Without)
12	"http://movie-locations.com/movies/1/Full-Monty-Shir"	"The Zodiac", "composer": ["David Lindup"],	The Full Monty
13	"http://movie-locations.com/movies/1/Funny-Face-Chat"	"Overture: Funny Face / 5 Wonderfo / Think Pink!", "com",	Funny Face
14	"http://movie-locations.com/movies/6/Rush-Hour-3-King-Cou"	"When the Roll Is Called Up Yonder", "composer": ["Ja"],	Ride The High Country (Suns In The
15	"http://movie-locations.com/movies/6/Hush-Hush-Sweet"	"Hush...Hush, Sweet Charlotte", "composer": ["Black De"],	Hush...Hush, Sweet Charlotte
16	"http://movie-locations.com/movies/6/Bud-And-Lois-You-Com"	"Bud And Lois / You Come But Tonight", "composer": ["B"],	Bonderful Life
17	"http://movie-locations.com/movies/6/Italian-Job-200"	"The Wreckoning", "composer": ["Gallin Manning"],	Tar The Italian Job
18	"http://movie-locations.com/movies/6/Justice-League"	"Between Theme", "composer": ["Danny Elfman"],	The Justice League
19	"http://movie-locations.com/movies/6/Just-Jay-Willson-L"	"Bittersweet", "composer": ["Chris Rea"],	The Krays
20	"[\"images\": [], \"visit\": [\"California\", \"Hawaii\"], \"introduction\": \"If You Don't Love Me (I'll Kill Myself)\", \"composer\": \"		Outbreak
21	"http://movie-locations.com/movies/6/Rushmore-house"	"Making Time", "composer": ["Eddie Phillips"],	Rushmore
22	"http://movie-locations.com/movies/6/...InMadness-Of"	"Au clair de la lune", "composer": ["Jean-Baptiste Lu"],	Quills
23	"http://movie-locations.com/movies/6/swingers-derby"	"You're Nobody 'Til Somebody Loves You", "composer": "R",	Swingers
24	"http://movie-locations.com/movies/6/Rush-Hour-Foo-C"	"Another Part of Me", "composer": ["Michael Jackson"],	Rush Hour
25	"http://movie-locations.com/movies/6/Suspiria-Haus-2"	"Fairest of the Season", "composer": ["Jackson Browne"],	Suspiria
26	"http://movie-locations.com/movies/6/Two-Lane-Black"	"", "composer": [{}],	Two Lane Blacktop
27	"http://movie-locations.com/movies/6/Sweet-Small-Of"	"", "composer": [{}],	The Sweet Small Of Success

将爬取的信息在 Web 端进行展示，结果如图所示：





2. 深度分析：信息结构化

2.1 原理和实现

由 1 所使用的方法，我们从非结构化的两个网页中提取数据，然后使用编程语言映射的特性，将获取的数据转化成结构化的对象，然后进行数据库的存储。

2.2 效果展示

在 Java 中进行存储的结构化数据对象如图所示：

```
/**
 * @author liuxuanming
 * @date 2021/7/13 10:54
 * @description: 电影数据对象
 */
@Data
public class MovieDO {

    private String id;

    private String name;

    private List<String> director;

    private String country;

    private String year;

    private Double rating;

    private List<String> genre;

    private String post;

    private String popularity;

    private String description;

    private List<MusicDO> music;

    private LocationDO location;

    private String update_time;
}
```

一条文档在 MongoDB 中的结构化展示如图所示：

```
{
  "_id": {"$oid": "60ee4fc37526f206a1083182"},
  "country": "United States",
  "description": "The setting is Detroit in 1995. The city is divided by",
  "director": ["Curtis Hanson"],
  "genre": ["Drama", "Music"],
  "location": {
    "images": ["http://movie-locations.com/movies/8/8-Mile-Schultes-Avenue"],
    "visit": ["Detroit", "Michigan"],
    "introduction": "Jimmy Smith Jr, aka B-Rabbit (Eminem) struggles to c",
  },
  "music": [
    {
      "name": "Shook Ones Pt. II ",
      "composer": ["Prodigy", "Havoc", "Mobb Deep"]
    },
    {
      "name": "Survival Of The Fittest ",
      "composer": ["Prodigy", "Havoc", "Mobb Deep"]
    },
    {
      "name": "Unbelievable ",
      "composer": ["R. Kelly", "DJ Premier", "The Notorious B.I.G.", "The",
    },
    {
      "name": "Times Up ",
      "composer": ["Anthony Best", "O.C."]
    }
  ]
}
```


3. 深度搜索：结果聚类和排序

我们提供电影名、音乐名、走访片场名三种搜索的目的和途径。用户可以在进行完一次广泛的搜索后，根据上面三类进行结果的聚类和排序，得到更加细致的结果。

3.1 原理和实现

在 Elasticsearch 中，我们设置了如下的查询语句：

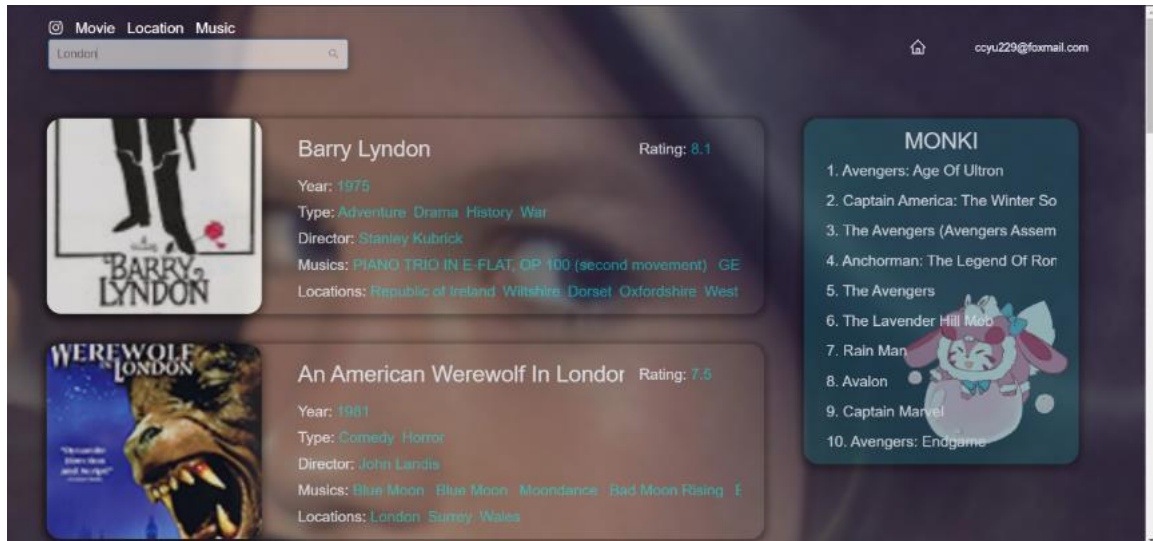
```
    "should": [
      {
        "match": {
          "name": {
            "query": "Aven Berlin",
            "fuzziness": "AUTO",
            "boost": 0.25
          }
        }
      },
      {
        "match": {
          "visits": {
            "query": "Aven Berlin",
            "fuzziness": "AUTO",
            "boost": 0.25
          }
        }
      },
      {
        "match": {
          "musics": {
            "query": "Aven Berlin",
            "fuzziness": "AUTO",
            "boost": 0.25
          }
        }
      }
    ]
  }
}
```

根据用户查询时选择的优先查询选项，后端会对不同查询类型的权重加以改变，这样会让优先级更高的属性所在的电影进行提升。此外，这样的改变是有一定限制的。例如，在 Location 中查询符合度较低的文档，即使用户选择了优先 Location 的搜索规则，也不一定能够查询到该文档，因为其他属性下可能有符合度更高的文档存在。这使得搜索结果能够让用户更加满意。

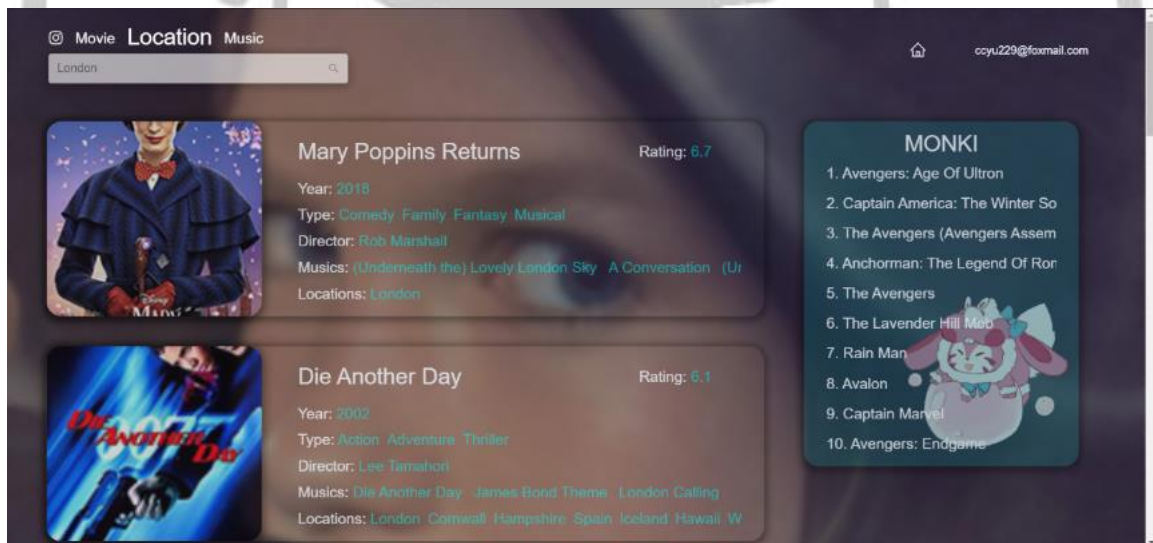
此外，我们的搜索还综合考虑了用户打分、用户流量等因素，会在一定程度上将上述两种因素纳入搜索打分评价的考虑范围。

3.2 效果展示

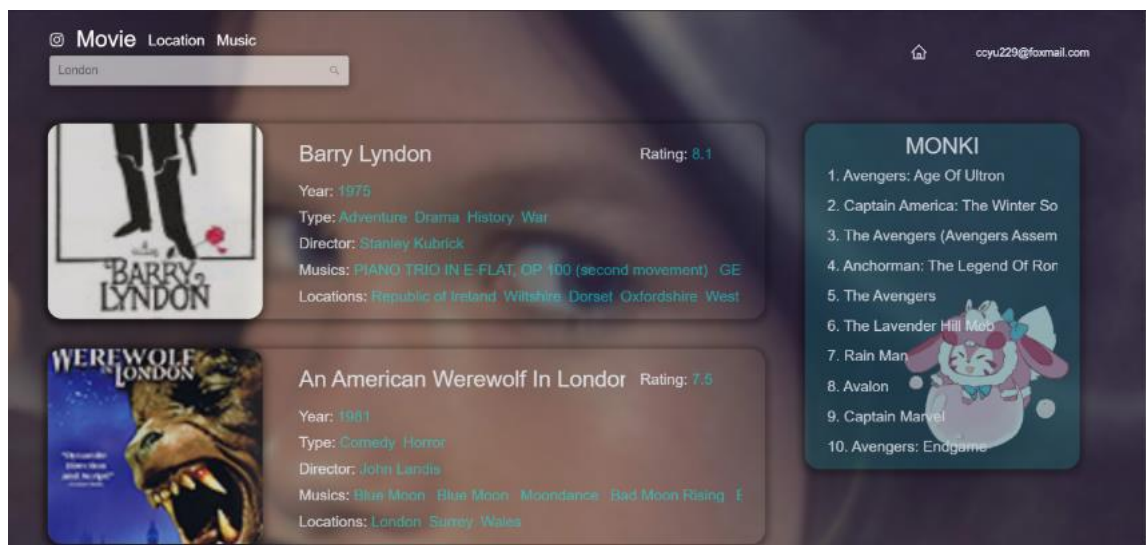
以搜索 London 为例，当默认排序时，搜索结果如下：



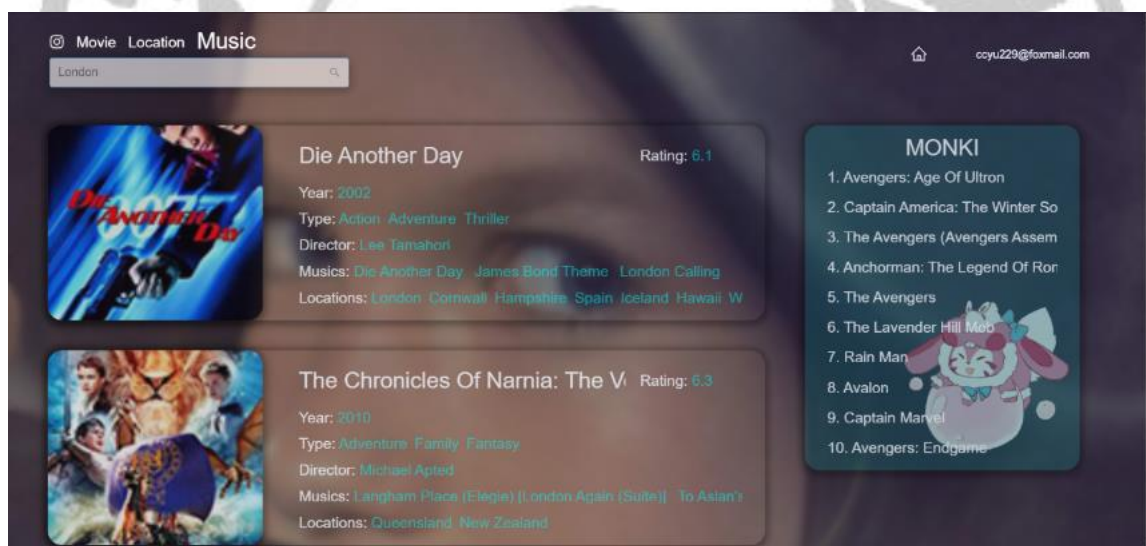
搜索结果优先地点时，搜索结果如下：



搜索结果优先电影时，搜索结果如下：



搜多结果优先配乐时，搜索结果如下：



4. 深度搜索：基于多种形式的推荐算法

4.1 原理和实现

4.1.1 特征向量

每一部电影都有它自己的特征。

电影自身附带了很多属性，但是在做推荐时，我们考虑的属性的范围可以适当收窄。

通常情况下，电影的导演、电影类型、电影名称和电影介绍可以成为电影的特征。考虑到电影的介绍本身是一大段文本，因此将特征分为两部分组成。

第一部分为电影导演、电影类型和电影名称，第二部分为电影的介绍。

将第一部分的特征连接成文本，特征文本和描述文本分别形如下图。

```
In[2]: meta
Out[2]: 'Directors: Curtis Hanson. Genre: Drama, Music. Name: 8 Mile '

In[3]: text
Out[3]: 'Description: The setting is Detroit in 1995. The city is divided by 8 Mile, a road that splits the town in half along
```

如此处理之后，一部电影的特征被概括为两个字符串，姑且称它们为 meta 和 text。

为了达到计算电影之间的相似度，我们需要计算出电影的特征向量。结合服务器性能考虑，在这里使用 Bert-base-cased 的预训练模型（以下简称 Bert）。对于 Bert，每一个句子都会被 encode 成一个张量

$$Tensor_{13 \times 1 \times token_len \times 768}$$

其中 13 代表 Bert 的 1 个初始 embedding 层和 12 个 hidden 层，总共 13 个层的输出；1 代表 batch_size，由于 Bert 的预训练只进行在单句和两句输入上，而

没有进行长文本输入的训练，此处也没有下游任务可供 Fine Tuning，这里我们统一将段落分割成单句一一处理；

对于 token_len，这和 Bert 的工作机制有关。Bert 在 Encode 前首先需要将句子切割成 token 的列表，列表其中的元素为单词或者单词碎片，token_len 即为列表的长度；最后的 768 位 base 版的 Bert 固定输出长度。

宏观来看，每一个单词碎片都被每一个 hidden 层编码成了一个 768 维的向量，因此每一个 token 就对应了一个的张量

$$Matrix_{13 \times 768}$$

对于如此多的 hidden 层输出，一般情况下不需要将它们全部用于计算。通常取倒数 4 层的平均值作为此 token 的向量。如此，我们便将一个 token 映射到了一个 768 维的向量 V 上

$$V \in R^{768}$$

对于长文本，应该有 token_len 个单词向量，在没有下游任务可供训练的情况下，我们取所有词向量的平均值为此长文本的特征向量。如此，一个长文本便被编码为了一个 768 维的向量 V

$$V \in R^{768}$$

按照这样的逻辑，电影的 meta 和 text 都被映射成 768 维的向量。最后，我们需要将两个向量按照一定的权重加起来。如果有训练数据，这个权重完全可以通过监督学习训练。但即使经过了非常努力的查询，我们仍然没有获取到任何数据有品质的可供训练的数据。故只能暂时规定，meta 的向量占比 0.3，text 的向量占比 0.7。

$$V_{movie} = Meta * 0.3 + Text * 0.7$$

编码后，我们再用这个向量除以它自身的模长，做一次单位化。

$$V' = \frac{V}{|V|}$$

如此，每一个电影被编码成了一个 768 维线性空间的一个单位向量。

此外，在计算完所有电影的特征向量后，我们会将它们保存到数据库中。重复使用时只需要查询即可完成特征向量的提取。

4.1.2 召回

召回算法完全基于电影的关键词和特征，借助 ES 搜索引擎的算法，我们为每一个用户选取 50 条电影作为召回算法的结果。

系统每天定时维护召回部分的数据。召回的结果存在 MySQL 的表 t_recommend 中。

召回算法的实现机制如下：

首先选取用户过去 10 次的搜索记录，过去的 10 次点击记录。

在 ES 中对于每一条搜索记录重新执行搜索，取每个搜索结果的前 3 条记录，可以获得 30 条记录。

对于每一次点击记录，在 ES 中重新执行搜索，取每个搜索结果的前 2 条记录，可以获得 20 条记录。

如果用户的行为不足 10 条搜索记录或者 10 条点击记录，系统会用随机选取的方式补足到 50 条。

4.1.3 重排

重排的函数是按照可以在线上调用，也可以在线下调用的需求来编写的。

重排算法要求在数据库中，此用户拥有召回算法的 50 条推荐记录，并且至少拥有 1 条最近的点击记录。两者中的任何一个条件不被满足，重排算法失效。

重排时，首先取出用户近 10 次点击记录所点击的电影，并将它们转变为张量，即得 768*10 的矩阵，记为

$$Click_{768 \times 10} = [\alpha_{c'_1}, \alpha_{c'_2}, \dots, \alpha_{c'_{10}}]$$

$$\text{where } \alpha_{c'_i} \in R^{768}, i \in \{1, 2, \dots, 10\}$$

再将召回步骤中，对用户推荐的 50 部影片取出转换成矩阵，得到 50 * 768 的矩阵，记为

$$Recall_{50 \times 768} = [\alpha_{c_1} \quad \alpha_{c_2} \quad \dots \quad \alpha_{c_{50}}]^T$$

$$\text{where } \alpha_{c_i} \in R^{768} \text{ and } i \in \{1, 2, \dots, 50\}$$

接下来计算各个电影之间的相似度

$$Similarity_{50 \times 10} = Recall_{50 \times 768} \cdot Click_{768 \times 10} =$$

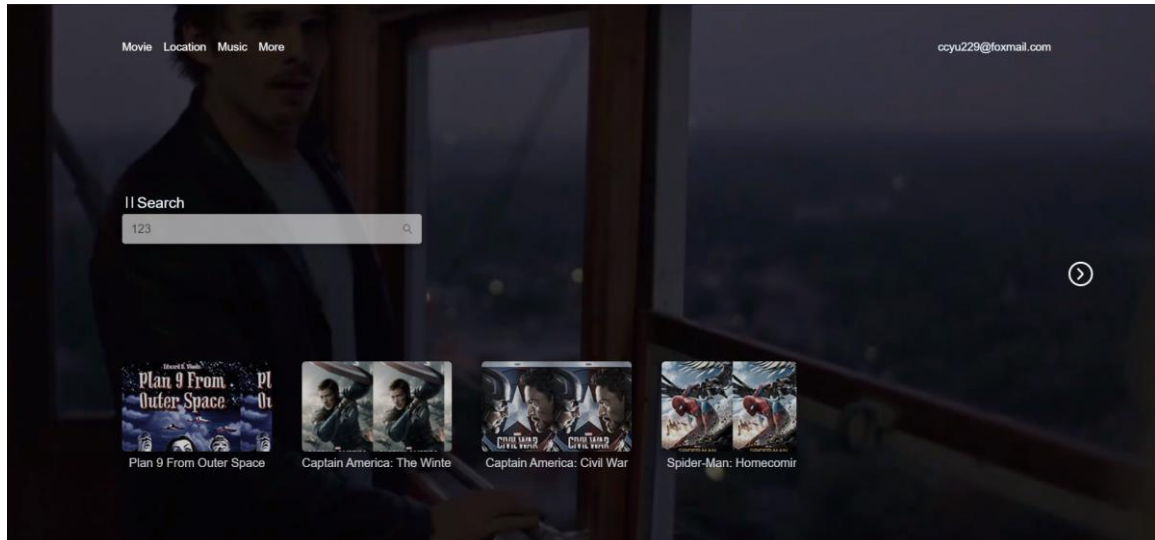
$$\begin{bmatrix} \alpha_{c_1, c'_1} & \alpha_{c_1, c'_2} & \dots & \alpha_{c_1, c'_{10}} \\ \alpha_{c_2, c'_1} & \alpha_{c_2, c'_2} & \dots & \alpha_{c_2, c'_{10}} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \alpha_{c_{50}, c'_1} & \alpha_{c_{50}, c'_2} & \dots & \alpha_{c_{50}, c'_{10}} \end{bmatrix}$$

$$\text{where } \alpha_{i,j} = \alpha_{c_i} \cdot \alpha_{c'_j}, \alpha_{c_i} \in Recall, \alpha_{c'_j} \in Click, \alpha_{i,j} \in [0, 1]$$

接着将矩阵拉直，按照相关度由高到低排序，再去掉重复，即可得到推荐给此用户的电影列表。

4.2 效果展示

根据用户点击的推荐：



根据大数据的推荐：

