数码类搜索引擎-实验报告

王世瑶 侯天孝 张紫曦 黄伊新

1 设计理念

数码产品是当代年轻人重要的消费方向，但是大多数消费者对于自己的目标产品定位比较模糊，不知道自己适合什么样价位的产品，也无法避开一些隐藏的消费陷阱。而爬虫技术可以整合全网的信息，分析处理并推荐给用户。我们致力于打通各大主要电商平台的壁垒，为那些对数码产品并不十分了解的新手推荐价位、性价比都合适的数码产品，构建用户友好的信息检索平台。

2 功能介绍

我们的网站整合了天猫、京东和苏宁三大国内主要的电商平台上数码分区的数据内容，并实现了基于这些数据的搜索功能。我们实现的具体搜索功能如下：

- 基础文字搜索：输入想要搜索的关键词，返回所有数据中与该关键词有相关度的商品。在搜索框中输入关键字时，网站会弹出历史搜索记录以及联想搜索关键词。
- 标签搜索：提供数码分区相关的大量关键词，包括品牌、平台、商品种类、特征等，用户可以通过勾选相关的关键词，并结合基础文字搜索来获得想要的搜索结果。
- 价格区间搜索：用户可以输入一个自己的心理价位区间，来获得满足自己预算情况的搜索结果。
- 图片搜索：用户可以上传图片，以在我们的数据库中寻找相似度高的商品。同样可以上传品牌的 logo 图片，进入该品牌的分区。

对于每一件商品，我们展示相关网站给出的图片、商品名称、价格、店铺、及其推荐值。同时商品名称上的超链接指向对应电商网站的商品详情页，而店铺上的超链接指向对应电商网站的店铺首页。同时对于搜索结果，我们提供了多种排序方式：

- 相关度排序：搜索结果按照与关键词的相关性程度排序，使得用户可以快速准确的定位自己想要的商品。
- 价格排序: 根据相关网站上给予的标价进行排序，支持升序与倒序，以便用户根据自己的需求获得适合自己价位的商品。
- 用户满意度排序：根据相关网站上的评论分析计算出该商品的满意度，优先展示用户满意度高的产品，可以帮助用户选择质量更加靠谱的商品。

- 综合排序：基于价格、用户满意度加权计算得到一个综合的推荐值，帮助用户买到真正物美价廉的商品。用户也可以通过调整“商品质量-性价比”的内心比重，来更改加权计算的权重。

3 网页抓取

我们从天猫、京东和苏宁三个电商平台抓取了约 15000 个数码类商品的商品信息。商品的图片、详情页链接和商品名等是由爬取搜索的目录页获得，而评论的具体内容则在商品详情页中具体获取。由于三个网站每个的具体布局都不同，爬取细节上会有偏差，下面仅以京东为代表进行说明。

3.1 目录页抓取

由于存在反爬虫，使用 selenium 动态抓取。在使用手机扫码登陆后，对关键词为数码的目录页进行遍历。其中翻页是结合了 js 和 xpath 使得翻页有关的元素可见，并使用 selenium 语法进行点击而实现的。

3.2 评论信息抓取

评论信息是通过翻页遍历以及动态加载进行获取的。

首先是遍历目录页文件，将商品的详情页地址放在一个队列中等待抓取。由于可能发生多次中断，会在加入队列前判断该网页是否已被访问过。

图 1: 京东评论界面截图

对于每一个网页，首先对整个页面进行加载，获得第一页的评论内容。接着，在第一页的页面上依次获得上图中的好评度、商品评价标签以及评论统计信息，并存入和网页对应的文件中。接着，对于具体的评论内容，除了摘取评论的文字信息以外，同时也对其评价的星级进行了采集，为后续训练情感模型做准备。评论抓取部分主循环代码如下

```
while notLast() and cnt <= 100:
    for i in soup.findAll("p", {"class":"comment-con"}):
```

```
        starTag = i.previousSibling.previousSibling.get('class')[-1]
        if '-' not in starTag:
            f.write(f"{starTag}\n")
        f.write(i.get_text())
        f.write("\n")
    target = driver.find_element(By.CLASS_NAME, "ui-pager-next")
    driver.execute_script("arguments[0].scrollIntoView();", target)
    time.sleep(random.uniform(2,4))
    driver.execute_script("window.scrollBy(0,-150)")
    time.sleep(random.uniform(2,4))
    target.click()

    new_height = driver.execute_script(js)
    height = 0
    for i in range(new_height//1500):
        height += 1500
        driver.execute_script('window.scrollTo(0, %s)'%(height))
        time.sleep(0.5)
    prev_height = new_height
    new_height = driver.execute_script(js)
    while (new_height != prev_height):
        for i in range((new_height-prev_height)//1500):
            height += 1500
            driver.execute_script('window.scrollTo(0, %s)'%(height))
            time.sleep(0.5)
        prev_height = new_height
        new_height = driver.execute_script(js)
    content = driver.page_source
    soup = BeautifulSoup(content,features="lxml")
    cnt += 1
```

在爬取天猫的网页时，由于有天猫界面、天猫超市界面和天猫国际进口超市三种，首先需要对页面中的元素进行判断来确定具体页面类型，再进行有关抓取。而在爬取苏宁有关的界面时，出现了目录页显示的商品名称和详情页不匹配的情况，又考虑到在进行搜索时需要商品名称包含较多有效信息，故在抓取评论时也在详情页抓取了商品名和品牌名放在了文件的前两行，以供后续使用。

3.3 图片抓取

期望实现的功能中有“以图搜图”和“logo 搜索”等和图片相关的功能，所以需要先对图片进行爬取。初步的考虑是在进行商品比较时，只需要判断出上传商品的类型、品牌、型号即可，所以只需要保证图片覆盖商品规格比较完善即可，因此用到的图片都只用从京东上爬取下来的图片。

对于以图搜图功能的实现，我的考虑是在爬取图片的时候就将对应店铺对于商品的描述一并爬取下来，并且作为对应图片的命名，这样后续进行图片检索的时候，匹配到相似图片在之后只需要返回图片命名就可以知道对应商品信息（建立在相信京东数码店铺图片与描述契合度较高的前提下），进而进行对于商品描述的搜索，完成以图搜图功能的实现。

爬取图片所用部分代码如下：

```
import string
from bs4 import BeautifulSoup
import re
import urllib.request, urllib.error
from urllib.parse import quote
import xlwt
import ssl
import requests
import time


def main():
    keywords = input("请输入关键字：")
    page = input("请输入页数：")
    datalist = []
    savePath = "jd" + keywords + ".xls"
    baseUrl = "https://search.jd.com/Search?keyword=" + keywords + "&page="
    datalist = getData(baseUrl, page)
    time.sleep(1)
    saveData(datalist, savePath)


findImgSrc = re.compile(r'<img.*data-lazy-img="(.*?)"', re.S)
findPrice = re.compile(r'<i>(.*?)</i>', re.S)
findInfo = re.compile(r'<div class="p-name p-name-type-2">(.*?)<em>(.*?)</em>', re.S)
findTag = re.compile(r'<span(.*?)>(.*?)</span>', re.S)
findStore = re.compile(r'<span class="J_im_icon"><a.*?>(.*?)</a>', re.S)
findSupply = re.compile(r'<i class="goods-icons J-picon-tips J-picon-fix" data-idx="1" data-tips="京东自营，品质保障">(.*?)</i>',
                        re.S)


def getUrl(askUrl):
    head = {}
    head[
        "User-Agent"] = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36"
    s = quote(askUrl, safe=string.printable)
    request = urllib.request.Request(s, headers=head)

    html = ""

    try:
        response = urllib.request.urlopen(request)
        html = response.read().decode("utf-8")
```

```
    except Exception as e:
        if hasattr(e, "code"):
            print(e.code)
        if hasattr(e, "reason"):
            print(e.reason)
    return html


def getData(baseUrl, page):
    datalist = []
    for i in range(0, int(page)):
        url = baseUrl + str(i)
        html = getUrl(url)
        soup = BeautifulSoup(html, "html.parser")
        for item in soup.find_all("li", class_="gl-item"):
            data = []
            item = str(item)
            imgSrc = re.findall(findImgSrc, item)[0]
            imgSrc = imgSrc[2:]
            price = re.findall(findPrice, item)[0]
            data.append(imgSrc)
            data.append(price)
            info = re.findall(findInfo, item)[0]
            tmpTag = info[1]
            tag = re.findall(findTag, tmpTag)
            if len(tag) != 0:
                data.append(tag[0][1])
                tmpTag = re.sub(tag[0][1], '', tmpTag)
            else:
                data.append(' ')

            tmpTag = re.sub('<(.*?)>', '', tmpTag)
            tmpTag = re.sub('\n', '', tmpTag)
            tmpTag = re.sub('\t', '', tmpTag)
            data.append(tmpTag)
            try:
                imgSrc="https://"+imgSrc
                r=requests.get(imgSrc)
                imgname=imgSrc[2:].split("/")[-1]
                image=r.content
                print(tmpTag.split("/")[-1])
                with open("headset/"+tmpTag.split("/")[-1]+imgname
    [-4:],'wb') as f:
                    f.write(image)
            except Exception as err:
                time.sleep(1)
                print(err)
                print("产生未知错误，放弃保存")
                continue
    f.close()
```

```
    return

if __name__ == '__main__':
    main()
```

对于图片搜索相似度对比，使用了三种哈希函数来进行相似度比对，以确保返回的匹配图片确实是在多个维度都和目标图片相一致，在此略去哈希函数比对，列出部分主函数实现：

```
from PIL import Image
from multiprocessing import Process
import histogram as htg
import aHash as ah
import pHash as ph
import dHash as dh
import os

def imagesearh(file_name)
    flag=0
    tmp=0
    name=""
    img1 = Image.open(file_name)
    for filepath,dirnames,filenames in os.walk(r'dataset'):
        for filename in filenames:
            img2 = Image.open(filepath+"/"+filename)
            img1_htg = htg.regularizeImage(img1)
            img2_htg = htg.regularizeImage(img2)
            hg1 = img1_htg.histogram()
            hg2 = img2_htg.histogram()
            sub_thread = Process(target=htg.drawHistogram, args=(hg1,
    hg2,))
            sub_thread.start()
            res=htg.calMultipleHistogramSimilarity(img1_htg, img2_htg)
    *64+ah.calaHashSimilarity(img1, img2)+ph.calpHashSimilarity(img1,
    img2)+dh.caldHashSimilarity(img1, img2)
            print(res)
            if(res==256.0):
                name=filename
                flag=1
                break
            if(flag==1):
                break
            elif(res>tmp):
                tmp=res
                name=filepath+"/"+filename
            else:
                continue
    return name
```

4 评论情感分析

本处情感分析由 snownlp 重新训练获得。训练数据包括京东 5000 个网页的评论加上原本的 snownlp 的数据。首先根据用户给的评分信息进行初步分类，接着考虑到存在大量的打高分写差评的情况，对含有部分关键词的评论进行了删除，如“垃圾”，“太差”等。接着将数据导入到 snownlp 中进行训练，并进行替换即可获得数码类电商铲平的情感分析模型。

接着为了方便后续使用，提前获得了每个网页的情感分数并存入文档。在京东和苏宁的部分，由于其网页版包含了评论数量的因素故也对其进行了考虑。本处认为评论数越多，情感的指向应该更加强烈，而晒图和视频的评论则一般包含了更多的内容，本处认为晒图的影响力是 *2，视频的影响力是 *3，并将最后总的评论有效数取 log 获得最后的影响力放大的因数。若后面有方法通过手机端爬取天猫的网页并获得其评论数的信息则可改为使用这种考虑评论数量的评论情感分数，但本处为了跨平台的兼容性，最后还是选用了仅对评论情感正向程度进行考量的评论情感分数。

5 建立搜索引擎

5.1 使用工具

- lucene 版本：lucene 8.6.1
- 分词器：org.apache.lucene.analysis.cjk.CJKAnalyzer

5.2 建立索引

使用 lucene 建立索引。对于每一个搜索目录页，遍历其中的每一个商品，抓取其中的信息并放入文档中，文档中的信息包括：商品名字，商品详情页网址，商品图片链接，商品价格，商品所在店铺名称，店铺链接，商品的情感评分，商品所在平台。其中商品的名称与店铺的名称是可索引对象 TextField，其余的皆为不同分析的对象 StringField。

```
if len(contents) > 0 and 'jd' in root :
    soup = BeautifulSoup(contents)
    for i in soup.find_all("div",{"class":"gl-i-wrap"}):
        url=i.find("div",{"class":"p-img"}).find("a").get("href")
        if url in product_crawled:
            continue
        else:
            product_crawled.append(url)
        img=i.find("img").get("src")
        if not img:
            img=i.find("img").get("data-lazy-img")
        try:
            price=i.find("div",{"class":"p-price"}).text[3:]
            price=price[:price.index('.')+3]
            price=str(price)
```

```
        except:
            continue
        name=i.find("div",{'class':re.compile("name")}).find('a').text
        name=re.sub('\n','',name)
        try:
            shop=i.find("div",{'class':"p-shop"}).text
            shopurl=i.find("div",{'class':"p-shop"}).find('a').get('
    href')
        except:
            shop="商家未找到"
            shopurl=''
        shop=re.sub('\n','',shop)
        score_file=open("score/html_jd_score/https{}#comment".format(
    valid_filename(url)), encoding='utf8')
        score = score_file.readline()
        score = float(score)
        score = str(score)
        score_file.close()
        doc=Document()
        doc.add(StringField('url',url,Field.Store.YES))
        doc.add(StringField('img',img,Field.Store.YES))
        doc.add(StringField('price',price,Field.Store.YES))
        doc.add(StringField('score', score, Field.Store.YES))
        doc.add(TextField('name', name, Field.Store.YES))
        doc.add(TextField('shop',shop,Field.Store.YES))
        doc.add(TextField('shopurl',shopurl,Field.Store.YES))
        doc.add(StringField('platform','jd',Field.Store.YES))
        writer.addDocument(doc)
```

IndexFiles.py(对京东的搜索页建立索引部分)

5.3 查询索引

基于布尔搜索 BooleanQuery 方法实现对于索引的查询。首先分析关键词的语句，建立一个字典储存输入的关键字的结构，其中 name 存储用户搜索的关键词，platform 表示搜索的平台，price_range 后的两个参数表示搜索结果的上下界，而 sort 后的参数表示查询结果的排序方式。其中 name 和 platform 相互之间可以实现组合搜索，而 price_range 和 sort 需要对搜索结果进行进一步整理。其相互之间的优先级为：name>price_range>sort。

因为要对搜索结果进行排序，所以需要将查询得到的结果重新整理为一个可以排序的数组，然后使用 sorted () 函数根据排序要求进行重新整理，值得注意的是价格和评分在文档中都以 str 的类型进行存储，需要进行类型转换。

```
command_dict = parseCommand(command)
querys = BooleanQuery.Builder()
for k,v in command_dict.items():
    if(k!='sort' and k!='price_range'):
        query = QueryParser(k, analyzer).parse(v)
        querys.add(query, BooleanClause.Occur.MUST)
```

```
7
8  scoreDocs1 = searcher.search(querys.build(),500).scoreDocs
9  scoreDocs = [i for i in scoreDocs1]
10 if(command_dict.get('price_range', '')!=''):
11     l,h=command_dict.get('price_range', '').split(',')[:2]
12     h=float(h)
13     l=float(l)
14     i=0
15     while(i<len(scoreDocs)):
16         price=float(searcher.doc(scoreDocs[i].doc).get("price"))
17         if price>h or price<l:
18             scoreDocs.pop(i)
19         else:
20             i+=1
21
22 if('score' in command_dict.get('sort','')):
23     scoreDocs = sorted(scoreDocs, key=lambda x: float(searcher.doc(x.
       doc).get("score")), reverse=True)
24 if('price_d' in command_dict.get('sort', '')):
25     scoreDocs=sorted(scoreDocs,key=lambda x:float(searcher.doc(x.doc).
       get("price")),reverse=True)
26 if('price_i' in command_dict.get('sort', '')):
27     scoreDocs=sorted(scoreDocs,key=lambda x:float(searcher.doc(x.doc).
       get("price")))
```

SearchFiles.py(查询以及排序部分)

5.4 综合排序算法

除以上简单的排序方法之外，我们实现了一个考虑到搜索结果的相关度，用户满意度以及价格的综合评分算法，用来进行综合的排序。这个算法还允许用户自定义“质量”与“性价比”两个因素在心里的权重，从而获取个性化的搜索结果。算法具体为

$$eval = rel\_score + (0.5 - n)\log_5(\frac{price}{\bar{price}} + 1) + n * score$$

其中 n 是一个 0 到 1 中间的数字，该数字越大，说明用户认为性价比更重要，而该数字越小，说明用户认为产品的质量更重要。在这里我们假设价格是产品质量的量度，而用户满意度评分反映了用户消费的体验是否达到预期，可以作为性价比的量度。我们使用对数函数对价格进行归一化处理，这里其实是考虑到了高消费存在的边际效应。可以看到算法中的后面两项 «rel_score, 因此可以认为这个评分很大程度上是相关度优先的。而在优先度接近的情况下，当 n=0.5 时我们只考虑用户满意度这单一因素，而当 n>0.5 时，价格与评分呈负相关，比较便宜的物品优先展示，满足比如学生党等低消费人群需求；当 n<0.5 时，价格与评分呈正相关，相对价格较高的物品优先展示，满足高消费人群的需要。同时用户满意度评分与综合评分始终正相关，这与正常用户的消费逻辑也是吻合的。

6 前端开发

整体上我们是用了 css 进行前端页面的美化，样式主要从京东和天猫的界面上汲取灵感，具体设计如下。

6.1 主界面

图 2: 首页搜索框

我们商城的主要颜色是红色，所以搜索框和商城名称都用红色进行了展示。同时，在首页搜索框我们还加入了历史记录的功能。若点击搜索框且搜索框内没有输入字符，则会在下拉框中展示你的历史记录。并且，也可以通过点击最下面的“删除历史记录”来清楚先前的搜索内容。本部分是通过 html 中 jquery 实现的，将历史搜索的内容存在 localStorage 的有关词项中，在后续若触发了显示事件则将其放入 div 进行显示。

图 3: 联想功能展示

另外，我们的搜索框也有词语的联想功能。本处使用 jquery，通过访问淘宝的关键词联想 api 获得有关联想词，将其经过处理以后放入我们的网页形成现在的样式。本处借鉴了 https://github.com/ziyuandi/TaobaoSearch 的使用方法。

图 4: 搜索方式及排序方式选择

至于其他的搜索方式我们则在首页以标签选择的形式进行了呈现，可以更加方便用户的使用。其中的自定义排序则是使用了滑动条，方便用户对二者的比例进行分配，更方便搜到符合自己需求的商品。

6.2 结果页

图 5: 结果页部分截图

在结果页可以按照选定的排序方式对商品进行排序，而在每个商品的显示界面上则有商品来自哪个平台、价格、商家名等信息，点击即可前往。同时，为了方便使用，我们还在页面右下角设置了返回首页以及滑到顶端的功能，更方便用户使用。

7 分工、总结与思考

经过一个学期的学习，我们最终完成了这样一个数码商城的检索平台的项目。这个项目运用了这个学期所学到的所有知识，包括使用 BeautifulSoup 解析 html 文件，使用 lucene 建立搜索引擎，使用 cnnfeatures 提取图片特征完成图片配对检索，使用 flask 实现前端网页等。同时我们也实现了很多课堂上不曾讲到的功能，比如使用 selenium 对反爬虫的网页进行动态抓取，使用 snownlp 训练模型分析评论的情感分数，运用 js 对网站进行渲染美化等。在完成这个项目的过程中，小组成员精诚合作，不断交流想法并且互相帮助解决技术问题，最终呈现了“鼠马商城”这个平台，其功能基本实现了我们在中期整合时制定的计划。

侯天孝同学主要负责搜索引擎与排序的代码实现，他表示：“这个部分主要是复用了上半学期的组合搜索的相关代码，再辅以对每一个文档相关值的排序，实现过程中并没有遇到太大的障碍。其实主要的难度在于功能设计方面，需要考虑到实际搜索的应用场景。因此我创造性的设计了自定义搜索的算法，可以让用户自己决定各个因素在自己心中的权重占比，这其实最大程度上发挥了我们这个爬虫网站整合信息的能力。很可惜的是由于时间上的问题以及能力的原因，在标签搜索这一方面我们没有做的很精细，我们采用了将标签放入搜索关键词的搜索方式，而不是在文档中建立一些特征的域，所以导致标签搜索的结果并不很精确。同时标签的组织方式和收录数量都不甚理想，显得有些草率。但是作为一个短时间完成的项目，可以反映我们的整体设计思路。”

黄伊新同学主要负责前期网页的爬取、评论情感分析以及输入框的功能实现，她认为她任务的难度在于较大的数据量以及购物网页的反爬虫措施，前期花费了较多精力在调整爬取的代码上，爬取的过程也耗费了较多时间，但最后也是成功爬取到了指定数量的网页。评论情感分析方面，由于有过多打高分写差评的现象，并不能保证训练集内百分百都是标定正确的语料，虽然能保证基本的准确性，但还有一定的提升的空间。在输入框的设计中，由于涉及到了 JavaScript 的使用，代码的书写可能没有那么的规范，同时由于受 taobao 的 api 限制，联想功能只能对单个的字进行联想，和真正的联想还有一定的差距。

张紫曦同学主要负责前后端交互、html 网页的编写、css 样式的设计和翻页功能的实现。她表示：“上半学期的 lab 中涉及的前后端交互较为简单，实现的前端样式也比较简陋，所以在完成大作业的过程中我遇到了许多挑战，尤其是在设计 css 样式时，如何让元素样式美观排布合理花费了我很长时间，但是最终网页的实现效果还是较为理想的，不使用任何前端框架完成样式设计的过程也让我对前端基础知识有了更深入的了解。不过由于时间和能力的限制，我在设计样式时没有兼顾不同浏览器兼容性的问题，导致网页在非 Chrome 的其他浏览器上的呈现效果不尽人意。底部页码栏固定为 10 页，导致当搜索到的商品数量少于 500 个时会存在空白页。搜索结果页左侧“全部结果”条目中返回的关键词为后端 lucene 搜索时的关键词，可读性差。这些都是有待改进的地方，但是总体来说实现了预期目标。”

王世瑶同学主要负责图片检索功能的实现，包括图片的爬取、比较，以及在 flask 中上传图片等功能的实现。在实现的过程中，图片的爬取是课程前半段所学的知识，图片的搜索是课程后半段所学，在图片搜索的过程中出现了索引较慢的现象，为此曾尝试使用 vgg 去训练搜索模型，可是由于开始时间较晚没能最终应用到本次的大作业中。在实现的过程中一些过程看似简单，实际操作起来却会在一些细节的地方碰壁。比如从本地读入图片这一个小操作，在 flask 中可能两个函数就可以实现，但是对于一次都没应用过的情况下对于这一操作就显得有些茫然。其次就是对于大型项目代码的整合，在这一次作业中得到了很大的提升，如何和同学去进行沟通合作，让大家的代码在同一个项目中可以正常运行，这要求我们不仅要实现自己的功能，更要对自己的代码进行封装与 readme 的良好解释，最后在这次大作业中能够对之前的所学进行整合，有一个小作品，是很令人鼓舞的一件事！

8 参考网络素材

- https://github.com/isnowfy/snownlp
- https://github.com/ziyuandi/TaobaoSearch