# 电工导 C 大作业报告

陈一鸣　黄浩初　罗宇晗　章凌恺

January 2024

## 目录

# 1 任务概览

爬虫部分任务概览：本次爬虫使用了 selenium 进行爬取，并采用 beautifulSoup 解析页面，根据网页的结构，得到我们需要的信息。但需要注意，由于存在登录，验证码等网页拦截的问题，我们需要采取一些反爬措施。

索引部分的任务概览：使用 Lucene 创建索引 + 全文检索即可对爬虫部分的内容建立对应的索引。

前端部分的任务概览：利用 flask 框架，css 等渲染，得到最终的搜索，结果展示等界面。其中包含关键词搜索，品牌名称限制以及以图搜图等相关的服装搜索，结果展示中包含根据评论内容打分进行的排序，根据价格排序，根据综合相关程度进行排序的结果。

图像识别模块任务概览：根据我们已有的知识，完成以图搜图有以下方式选择：提取颜色直方图，SIFT 图像描述子匹配，pyTorch 深度学习提取图像特征后 CNN 算法匹配，LSH 算法预处理检索。在我们小组自制的搜索引擎中，以图搜索的部分有根据商品图片检索，基于 logo 的图像检索，因此我们需要权衡各算法利弊，找到最合适的算法完成以上任务。

评论情感分析及 AI 助手部分任务概览：情感分析模块中使用 cnsenti 库进行中文情感分析。将所有的评论分析情感倾向，以实现根据评论内容对商品进行筛选。AI 助手部分概览：使用 gpt4all 库，利用本地部署的 gpt4all falcon LLM 模型进行服装推荐。为用户提供更多元化，更方便的购物体验。

# 2 实现细节和核心代码解释

## 2.1 爬虫实现

我们爬取的电商网站包括京东，淘宝，天猫，爬取到的基本商品信息包括商品价格，名称，图片链接，商品链接，品牌的信息。由于发现各大电商网站的评论一般和商品价格等属性所在的页面有所不同，因而我们采用的策略是，先爬取对应的基本商品信息，再根据我们爬取到的商品信息的 url，爬取对应的商品评论信息。

爬取商品信息我们采用的是 selenium 用来模拟用户的行为，并采集到对应的商品的信息。由于淘宝和京东等电商网站的结构不相同，并且反爬机制也有不同，因此我们对不同电商网站采取不同的爬取策略。

### 2.1.1 淘宝爬虫

淘宝网站的反爬虫策略主要在于滑块验证码和登录操作，因此我们写了一段代码，模拟登录

```python
def login_to_taobao(driver, username, password):
    try:
        # 使用Selenium打开登录页面
        login_url = 'https://login.taobao.com/member/login.jhtml'
        driver.get(login_url)

        # 输入用户名和密码
        username_input = driver.find_element('id', 'fm-login-id')
        username_input.send_keys(username)

        password_input = driver.find_element('id', 'fm-login-password')
        password_input.send_keys(password)

        # 提交登录表单
        password_input.send_keys(Keys.RETURN)

        # 等待一段时间，确保登录成功
        time.sleep(5)
```

```
19
20          # 处理滑块验证
21          handle_slider_verification(driver)
22
23      except Exception as e:
24          print(f'Error during login: {e}')
```

后面发现登录还需要处理滑块验证，我们写了如下滑块验证代码，以保证成功登录：

```
1  def handle_slider_verification(driver):
2      try:
3          switch_to_slider_iframe(driver)
4
5          # 使用WebDriverWait等待滑块元素可见
6          slider = WebDriverWait
7          (driver, 10).until(EC.visibility_of_element_located
8          ((By.XPATH, '//*[@id="nc_1_n1z"]')))
9
10         # 创建ActionChains对象
11         actions = ActionChains(driver)
12
13         # 将鼠标移动到滑块
14         actions.move_to_element(slider).perform()
15
16         # 在滑块上按下左键
17         actions.click_and_hold().perform()
18
19         # 拖动鼠标
20         actions.move_by_offset(300, 0).perform()
21
22         # 释放鼠标
23         actions.release().perform()
24
25         # 切回主文档
26         driver.switch_to.default_content()
27
28     except Exception as e:
29         print(f'Error during slider verification: {e}')
```

后来发现这两个方法淘宝反爬虫的机制十分严格，这样子解决验证码的操作会验证失败。因此最终我们采用的是手动登录，发现这样子也可以实现，只要显式登录的过程中我们登录自己的账号，就不会出现这样的问题，因此我们关闭无头模式，进行登录，能够成功爬到对应的商品信息。核心的获取相应商品信息的代码如下：

```
1  def scrape_page(driver):
2      # 模拟滚动，加载更多的商品
3      for i in range(5):
4          driver.execute_script('window.scrollTo(0,
5  ␣␣␣␣␣␣␣␣document.body.scrollHeight);')
```

```
        time.sleep(2)  # 等待加载完成，以应对反爬虫

    # 获取完整的页面内容
    page_source = driver.page_source

    # 使用BeautifulSoup解析页面内容
    soup = BeautifulSoup(page_source, 'html.parser')

    # 提取标题、图片和超链接信息
    items = soup.find_all('div', class_='Card--mainPicAndDesc--wvcDXaK')

    # 使用列表存储标题、图片和超链接的对应关系
    title_image_url_list = []

    for item in items:
        title_text = item.find('div', class_='Title--title--jCOPvpf')
        .find('span').text.strip() if item
        .find('div', class_='Title--title--jCOPvpf') else ''
        image_url = item.find('div',
        class_='MainPic--mainPicWrapper--iv9Yv90')
        .find('img')['src'].strip() if item
        .find('div', class_='MainPic--mainPicWrapper--iv9Yv90') and item
        .find('div', class_='MainPic--mainPicWrapper--iv9Yv90')
        .find('img') else ''
        href = item
        .find_previous('a', class_='Card--doubleCardWrapper--L2XFE73')['href']
        .strip()
        if
        item.find_previous('a',
        class_='Card--doubleCardWrapper--L2XFE73') else ''
        price = item.find('div', class_='Price--priceWrapper--Q0Dn7pN')
        .find('span',class_='Price--unit--VNGKLAP').text.strip()
        +item.find('div', class_='Price--priceWrapper--Q0Dn7pN')
        .find('span',class_='Price--priceInt--ZlsSi_M').text.strip()
        +item.find('div', class_='Price--priceWrapper--Q0Dn7pN')
        .find('span',class_='Price--priceFloat--h2RR0RK').text.strip()
        href = urllib.parse.urljoin(url, href)
        # 存储到列表
        title_image_url_list.append((title_text, image_url, price, href))

    return title_image_url_list
```

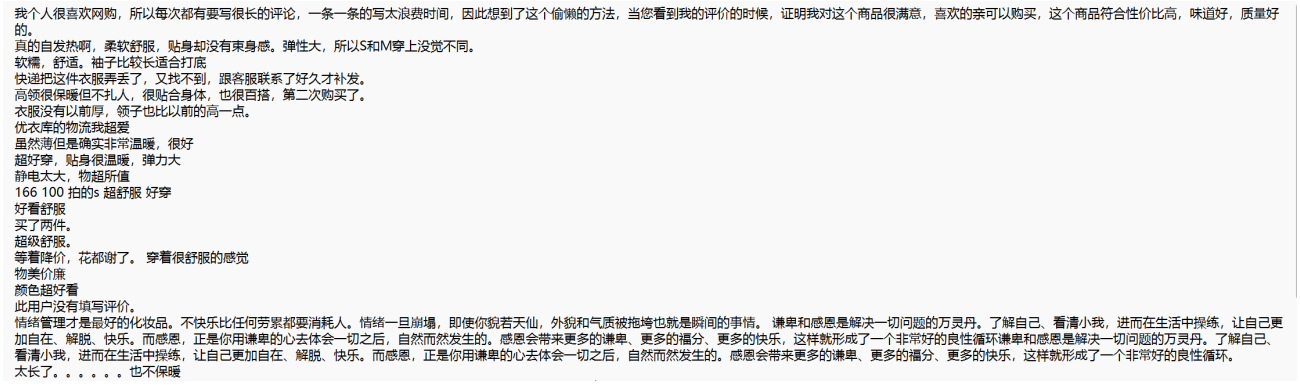
加入了页面的滚动，以适应淘宝中动态加载的机制。之后的一些代码可以参考 crawler_scroll.py 中的代码，主要是实现了不断下一页的爬取，以获得更多的商品信息。最终爬取到的结果如下：

Title: Theory西装连衣裙明星同款2023秋冬新品羊毛混纺双排扣长袖戗驳领
Image URL: https://img.alicdn.com/imgextra/i4/129970823/O1CN01Ru7ypD1HwyqO4LYdk !!0-saturn solar.jpg 460x460q90.jpg .webp
Brand: MANGO
Price: ¥538.00
Product URL: https://click.simba.taobao.com/cc im?p=MANGO&s=504358029&k=1069&e=m8%2B7wtkXcj%2FZrgL90a1DGXpIS8XKvwIYtXfkCyGbiMtBRdUTbPJgS%2FwuvpOVNiKKjw%
2B3A43hl5uuFHg5EttIDwA3bvAQgkomg2czn6qSt%2Fb6JPQMqY5lK%2FIUDJd9Hgc757N0ojhMKmoGBLB2AhkfDJFz29TneaLeQjB%2Furky1ZzPvVC8CUGK%
2FbO4MG6hzM0ktCoGPNJYYakytSgRwgTdYnB910Kl15H8xGAhOtTjoO1iHn0%2F6Lam7%2BYUUew0DCDkuF6VagRXnLuQSRuAYVQJPFjvMkx0tRFKco1zSpicmT3IYf1WZMhNAniyAjb3dh%2BbOcotn%
2FB6tEQ9QknikVtexcIOwo%2BwZ4BmU9n7DF%2FedBOXREn4IJeekTOuuBgpoRb5m5c3plAX6yk35XUrCRabTmHoHOoQcrrtpvbC%2FnWEdvAPRFQSMqaa4ljVmMq%2FnqAKKO73XLTu5cPl8vTdiEZg2D%
2F96Ta159YBj7F%2BCC6DlvRiVWejxCP3bTvwewPukgI5Pg1csBdq5GaEjnf6AlwV4A3pA%2B9zs%
2BcLSOhtXB6osufqBxGZ9MkEFeXmaYSLya0V8tNyXlal209sx8lVADTUpPRBkQyy1l8wkrW4OlTQNfRbw8ZmODv1Xhu1Aj%2Bw6lJOblgnWDykHUExQPBVNBlaKe0oQzsE5TFAjNstPZ2z%2BdhZbplPaaqfdBWmiXx%
2FfrVvgusTm4Z05kiHEfN8sY4panpRdKWEJ%2BLcMMt6lx2bstu8dFeroYgbS3%
2BtcSKDZcV5BjYgmdn48pl1ZOzkcLL9uSLWRfW8bbxwPHETDTst4KKZ3gQpWQNRiUfJ5rOXWKKNFuM0OseULpCuIpe79IJDTmwcVeijUW3qZMSXRfC3Mvrw0%2BzD6tIihVMi3x89LqWnukW2v%
2B6RnBFvdWIK2YilznCCUEMVzLVAmW4fB79IIC2gSDH%2FlpZ8AxM8jErGwjAWfvZMHWcYF8RcnsqQAFPBfr%2BTb8PoWQ0GsfDQpckOc28iIc1L1LGZGfE%2FR%2FwyxG8BuAYsU2v2dvlccyTbF40Y%
2FLJgg8q8lfrLTWBX0yJICQA%3D#detail

淘宝爬取评论内容是一个比较麻烦的事情，我们采取的方式是根据页面检查中的 network 模块，得到我们登录信息的 cookies 以及反爬虫所需要的 headers 和 parameters，利用以上三者信息，以及 requests.get 方法访问淘宝中 comment 所在的文件，即可得到对应的 json 文件内容，再根据网页中 comment 所在的位置，利用 json 文件的索引功能，找到对应的 comment 内容。具体代码结构如下：

```
cookies = {...}#从网页检查窗口中network位置得到
headers = {...}
param = {...}
response = requests.
get('https://rate.taobao.com/feedRateList.htm', params=params,
cookies=cookies, headers=headers)
# print(response.text)
json_match = re.search(r'\((.*)\)', response.text)
if json_match:
    json_data = json_match.group(1)
    # 将提取到的 JSON 数据转换为 Python 对象
    json_data = json.loads(json_data)
# json_data = response.json()
    print(json_data)
    for com in json_data['comments']:
        content = com['content']
        href = 'https://item.taobao.com/item.htm?id=706407719891&ns=
␣␣␣␣␣␣␣␣1&abbucket=9'
        validname = valid_filename(href)
        with open(f'comment/{validname}.txt', 'a', encoding='utf-8') as file:
            file.write(f'{content}\n')
else:
    print("No␣JSON␣data␣found␣in␣the␣response.")
```

代码中前面一部分用于处理 requests.get 到的内容中非 json 文件的部分，我们发现得到的内容会包含如同 jsonp[..] 的格式，不能利用这个格式的内容得到评论内容，因此我们利用正则表达式处理掉这部分内容，使我们能够成功爬取到对应评论。这么做是可行的，并且通过改变 id 的方式，我们可以爬取到更多的商品评论。爬取的结果如下：




但我们发现这个方式爬取到一定量的评论之后，并不能继续爬取了，似乎淘宝识别到了我的爬虫行为，并且禁止我的账号访问评论内容，这给我们带来了不小的冲击。因此在爬取到了一定数量的评论之后，我们尝试去京东网站爬取对应的信息。

### 2.1.2 京东爬虫

京东爬虫与淘宝爬虫类似，我们采用 selenium 进行京东网站的爬取，需要注意的是，我们需要将自动检测关闭，以消除反爬虫的影响：

```
    chrome_driver_path =
    'C:\\Users\\25458\\Downloads\\chromedriver-win64
␣␣␣␣\\chromedriver-win64\\chromedriver.exe'#这个路径是我chrome_drive
    r所在的路径
    chrome_options = ChromeOptions()
    # 这一部分将自动检测关闭，以保证爬虫的正常进行
    chrome_options.add_argument('--disable-blink-features=Automation
␣␣␣␣Controlled')
    chrome_options.add_argument('--disable-extensions')
    chrome_options.add_argument('--disable-dev-shm-usage')
    chrome_options.add_argument('--disable-gpu')
    chrome_options.add_argument('--no-sandbox')
    chrome_options.add_experimental_option('excludeSwitches',

    ['enable-automation'])
    chrome_options.add_experimental_option('useAutomationExtension', False)

    driver = webdriver.Chrome(service=Service(chrome_driver_path),

    options=chrome_options)
```

其他的包括翻页，网页结构分析得到对应的内容都与淘宝类似，这里不再赘述。

京东爬虫的评论获取：我们惊喜地发现，我们可以直接利用 selenium 访问商品链接，进而得到对应的商品评论信息，由于评论信息较多，爬取速度较慢，因此我们采取多线程爬取的形式，同时注意我们的爬虫文件中包含有 get_url.py 和 delete_txt.py 文件，作用分别是从 txt 文件中提取出商品 url 到新的 txt 文件中，便于进行评论爬取，同时 delete_txt.py 的作用就是将已经爬取过的网页在文件夹中进行删除。防止重复爬取浪费时间。

```
with ThreadPoolExecutor(max_workers=1) as executor:
    # 启动线程
```

```
    threads = []
    for _ in range(5):
        thread = threading.Thread(target=worker)
        thread.start()
        threads.append(thread)

    # 等待队列中的所有任务完成
    url_queue.join()

    # 停止线程
    for _ in range(5):
        url_queue.put(None)
    for thread in threads:
        thread.join()
```

每一个线程对应工作的 worker 函数如下：

```
def worker():
    driver = webdriver.Chrome(service=Service(chrome_driver_path),

    options=chrome_options)
    while True:
        try:
            url = url_queue.get()
            if url is None:
                break
            driver.get(url)
            get_fav_rate(driver,url)
            scrape_page(driver,url)
            for page in range(2, 11):  # 限制爬取的页数
                try:
                    next_page_button = driver
                    .find_element(By.CLASS_NAME,'ui-pager-next')
                    ActionChains(driver).move_to_element(next_page_button)
                    .perform()
                    time.sleep(2)
                    next_page_button.click()
                    scrape_page(driver,url)
                except Exception as err:
                    continue
            url_queue.task_done()
        except:
            continue
```

其中加入了 time.sleep() 等函数防止反爬虫机制的检测。其中 get_fav_rate 函数和 scrape_page 函数分别实现了获得好评率和得到评论内容的功能。具体的实现如下：

```
# 得到好评率信息
```

```
def get_fav_rate(driver, url):
    for i in range(2):
        driver.execute_script('window.scrollTo(0,
␣␣␣␣␣␣␣␣document.body.scrollHeight);')
    time.sleep(5)
    page_source = driver.page_source
    soup = BeautifulSoup(page_source, 'html.parser')
    element = WebDriverWait(driver, 15).until(
        EC.visibility_of_element_located((By.CLASS_NAME, "percent-con"))
    )
    fav_rate = soup.find('div', class_='percent-con')
    validname = valid_filename(url)
    with open(f'comment/{validname}.txt', 'w', encoding='utf-8') as file:
        file.write(f'好评率: {fav_rate.text}\n')

# 爬取一页中所有的评论信息
def scrape_page(driver, url):
    for i in range(1):
        driver.execute_script('window.scrollTo(0,
␣␣␣␣␣␣␣␣document.body.scrollHeight);')
    time.sleep(3)

    element = WebDriverWait(driver, 15).until(
        EC.visibility_of_element_located((By.CLASS_NAME, "comment-con"))
    )

    page_source = driver.page_source
    soup = BeautifulSoup(page_source, 'html.parser')
    comments = soup.find_all('p', class_='comment-con')

    for comment in comments:
        validname = valid_filename(url)
        with open(f'comment/{validname}.txt', 'a', encoding='utf-8') as file:
            file.write(f'{comment.text.strip()}\n')
```

分别采用了也采用了 time.sleep() 函数防止反爬虫，同时根据对应的标签的信息得到对应的内容。并且写入以对应的 url 为名的文件中。最终爬取的评论结果如下：

```
好评率: 92%
衣服很合身，挺满意的.
衣服很好，码偏小一点，各位亲注意喽!
布料质量好，穿着也舒服，值得推荐
站在消费者的角度说：卖家粗心大意，有缺陷的衣服也出货。（有图有真相，像素不好莫见怪）
尺码大小：有点大了，还好能穿
衣服偏短。。。。。。
不怎么样
一般吧
质量可以，穿起好帅
```

## 2.2 索引建立及情感分析

### 2.2.1 搜索引擎

使用 Lucene 创建索引 + 全文检索。通过将搜索得到的每个商品的各种信息打包成 dict 进行排序。排序方法包括相关度，好评率，综合评价，价格升序和价格降序。其中综合评价是在相关度的基础上，通过好评、差评数量占比进行修正，从而得到更加全面的评价分数。将检索到的信息排序好再打包发送到页面。对于 Lucene 中文分词性能欠佳的问题，我们使用了 jieba 中文分词库，利用 Lucene 总是会依照空格分词，在两个中文词汇之间插入空格，以提高 Lucene 的中文分词性能。而这会造成在进行 site-search 时，Lucene 识别进行 site-search 的命令会被分词库切割的问题。比如：输入：img_url: https://... 分词库会把 img_url: 这个本该被 Lucene 整体识别的指令切割，导致 Lucene 报错。所以我们采用对搜索结果进行过滤的方式，实现针对品牌等属性的搜索。具体建立索引代码实现如下：

```
def indexDocs(self, root, writer):
        t1 = FieldType()
        t1.setStored(True)
        t1.setTokenized(False)
        t1.setIndexOptions(IndexOptions.NONE)  # Not Indexed

        t2 = FieldType()
        t2.setStored(False)
        t2.setTokenized(True)
        # Indexes documents, frequencies and positions.
        t2.setIndexOptions(IndexOptions.DOCS_AND_FREQS_AND_POSITIONS)


        for root, dirnames, filenames in os.walk(root):
            for filename in filenames:
                if not filename.endswith('.txt'):
                    continue
                print("adding", filename)
                path = os.path.join(root, filename)
                file = open(path, encoding='utf-8', errors='ignore')
                contents = file.read()
                file.close()
                doc = Document()
                doc.add(StringField("name", filename, Field.Store.YES))
                doc.add(StringField("path", path, Field.Store.YES))
                if len(contents) > 0:
                    title = self.getTxtAttribute(contents, 'Title')
                    title = jieba.cut(title)
                    title = " ".join(title) # put space between
                    segmented words
                    doc.add(TextField('title', title, Field.Store.YES))
                    # title of the product
                    brand = self.getTxtAttribute(contents, 'Brand')
                    brand = jieba.cut(brand)
                    brand = " ".join(brand) # put space between
                    segmented words
```

```
                    doc.add(TextField('brand', brand, Field.Store.YES))
                    # brand of the product
                    img_url = self.getTxtAttribute(contents, 'Image URL')
                    doc.add(TextField('img_url', img_url, Field.Store.YES))
                    # image url of the product
                    price = self.getTxtAttribute(contents, 'Price')
                    doc.add(TextField('price', price, Field.Store.YES))
                    # price of the product
                    prd_url = self.getTxtAttribute(contents, 'Product URL')
                    doc.add(TextField('prd_url', prd_url, Field.Store.YES))
                    # product url of the product
                    pos = self.getTxtAttribute(contents, 'pos')
                    doc.add(TextField('pos', pos, Field.Store.YES))
                    # number of positive comments
                    neg = self.getTxtAttribute(contents, 'neg')
                    doc.add(TextField('neg', neg, Field.Store.YES))
                    # number of negative comments
                    neu = self.getTxtAttribute(contents, 'neu')
                    doc.add(TextField('neu', neu, Field.Store.YES))
                    # number of neutral comments
                    pos_rate = self.getTxtAttribute(contents, 'pos_rate')
                    doc.add(TextField('pos_rate', pos_rate, Field.Store.YES))
                    # positive rate of the product
                    neg_rate = self.getTxtAttribute(contents, 'neg_rate')
                    doc.add(TextField('neg_rate', neg_rate, Field.Store.YES))
                    # negative rate of the product
                else:
                    print("warning: no content in %s" % filename)
                writer.addDocument(doc)
```

有关于搜索的代码如下:

```
def filter_by_brand(products, brand_range):
    # products: a list of products
    # brand_range: a list of brands that the user wants to search
    # filter the products by the brands the user wants to search

    if brand_range == []:
        return products
    filtered_products = []
    for product in products:
        for brand in brand_range:
            if brand == product['brand']:
                filtered_products.append(product)
                break
    return filtered_products

```

```
16 def filter_by_price(products, price_range):
17     # products: a list of products
18     # price_range: a list of two numbers, the price range
19     # filter the products by the price range
20 
21     if price_range == []:
22         return products
23     filtered_products = []
24     for product in products:
25         if float(product['price']) >= price_range[0]
26         and float(product['price']) <= price_range[1]:
27             filtered_products.append(product)
28     return filtered_products
29 
30 def get_comprehensive_score(score, pos, neg, neu, penalty = 3):
31     # score: search score
32     # pos: positive comment count
33     # neg: negative comment count
34     # neu: neutral comment count
35     # penalty: penalty per negative comment
36     # calculate the comprehensive score of a product
37 
38     if pos + neg + neu == 0:
39         return score
40     comment_score = float(pos - neg * penalty) / (pos + neg + neu)
41     comprehensive_score = score * (1 + comment_score)
42     return comprehensive_score
43 # this is only a half-guessing method to evaluate the product
44 # you can change the penalty to get a better result
45 
46 def sort(products, sort_method = 1):
47     # sort_method: 1. search score
48     #              2. pos_rate
49     #              3. comprehensive_score
50     #              4. price up
51     #              5. price down
52     # sort the result by the sort method
53 
54     if sort_method == 1:
55         products.sort(key = lambda x: x['search_score'], reverse = True)
56     elif sort_method == 2:
57         products.sort(key = lambda x: x['pos_rate'], reverse = True)
58     elif sort_method == 3:
59         products.sort(key = lambda x: x['comprehensive_score']
60         , reverse = True)
61     elif sort_method == 4:
```

```
62          products.sort(key = lambda x: x['price'], reverse = False)
63      elif sort_method == 5:
64          products.sort(key = lambda x: x['price'], reverse = True)
65      else:
66          print('Error: sort method not found!')
67          return
68      return products
69
70  def run(searcher, analyzer, command):
71      # while True:
72      if deletespace(command) == '':
73          print('Error: empty command!')
74          return
75
76      products = []
77      score_threshold = 0 # threshold for score of each token
78
79      print()
80      print ("Searching for:", command)
81
82      command = jieba.cut(command)
83      command = list(command)
84      cmd_tokens = len(command) # number of tokens in command
85      command = " ".join(command) # put space between segmented words
86      query = QueryParser("title", analyzer).parse(command)
87      scoreDocs = searcher.search(query, 50000).scoreDocs
88      print ("%s total matching documents." % len(scoreDocs))
89
90      for i, scoreDoc in enumerate(scoreDocs):
91          product = {}
92          if scoreDoc.score < score_threshold * cmd_tokens:
93              continue
94          doc = searcher.doc(scoreDoc.doc)
95          product['title'] = deletespace(doc.get("title"))
96          product['prd_url'] = doc.get("prd_url")
97          product['price'] = doc.get("price").rstrip('\n')
98          product['img_url'] = doc.get("img_url").rstrip('\n')
99          product['brand'] = doc.get("brand").rstrip('\n')
100         product['file_path'] = doc.get("path").rstrip('\n')
101         product['search_score'] = scoreDoc.score
102         product['pos_rate'] = doc.get("pos_rate").rstrip('\n')
103         product['comprehensive_score'] =
104         get_comprehensive_score(scoreDoc.score,
105         int(doc.get("pos").rstrip('\n')),
106
107         int(doc.get("neg").rstrip('\n')),
```

```
108
109             int(doc.get("neu").rstrip('\n')))
110             print ('title:', deletespace(doc.get("title")), 'score:',
111             scoreDoc.score)
112             products.append(product)
113                 # print 'explain:', searcher.explain(query, scoreDoc.doc)
114         print("Search finished!")
115
116         # prd_url = prodcut url
117         # img_url = image url
118         # there's no sales volume cuz we didn't crawl it
119
120         # there're 4 ways to sort the result
121         # 1. sort by search score
122         # 2. sort by pos_rate
123         # 3. sort by comprehensive_score
124         # 4. sort by price
125         products = sort(products, 3)
126
127         return products # all return in a list, each element is a dict
```

这其中关于核心代码的解释已经放在了注释当中，这里不再赘述。

### 2.2.2 评论内容情感分析

这一部分我们使用 cnsenti 库进行中文情感分析。将所有的评论分析情感倾向，如果 positive > negative 则将其识别为好评。类似地，将所有评论识别为 positive, negative, neutral 中的一种之后，按照 positive 和 negative 占比计算好评率和差评率。将结果写入原始数据以便于建立索引。由于从网页上爬取的评论包含大量的 unicode 表情符号，所以在进行情感分析之前，对评论文件进行了清洗，使用 emoji 库，遍历所有文件，去除了所有的表情符号。由于原理较为简单，在这里不做赘述。具体代码实现如下：

```
1  def analysis(comment_path, result_path):
2      # args: comment_path: the path of the comment files
3      #       result_path: the path of the analysis result files
4      # process all the files in the comment_path, and write the emo-analysis
5
6      # result to the result_path
7
8      for root, dirs, files in os.walk(comment_path):
9          for file in files:
10             print("processing file: ", file)
11             com_path = os.path.join(root, file)
12             res_path = os.path.join(result_path, file)
13             com_file_lines = open(com_path, 'r', encoding='utf-8').readlines()
14             res_file = open(res_path, 'w', encoding='utf-8')
15             # open the comment file and the result file
16
17             for c in com_file_lines:
```

```
18                  print(">>> " + c)
19                  emot = senti.sentiment_calculate(c)
20                  if emot['pos'] > emot['neg']: # positive
21                      emot = 1
22                  elif emot['pos'] < emot['neg']: # negative
23                      emot = -1
24                  else: # neutral
25                      emot = 0
26                  res_file.write(str(emot) + '\n')
27                  print(emot, " written to file")
28                  # write the analysis result to the result file
29                  in the form of 1, -1, 0
30
31  def conclude(analysis_path, result_path):
32      # args: analysis_path: the path of the analysis result files
33      #       result_path: the path of the conclusion files
34      # process all the files in the analysis_path, and write the
35      # conclusion to the result_path
36
37      for root, dirs, files in os.walk(analysis_path):
38          for file in files:
39              print("processing file: ", file)
40              com_path = os.path.join(root, file)
41              com_file_lines = open(com_path, 'r', encoding='utf-8').readlines()
42              # open the analysis result file
43
44              pos = 0
45              neg = 0
46              neu = 0
47              for c in com_file_lines:
48                  if c == '1\n':
49                      pos += 1
50                  elif c == '-1\n':
51                      neg += 1
52                  else:
53                      neu += 1
54              print("pos: ", pos, "neg: ", neg, "neu: ", neu)
55              # count the number of positive, negative and neutral comments
56
57              res_path = os.path.join(result_path, file)
58              res_file = open(res_path, 'w', encoding='utf-8')
59              res_file.write("pos: " + str(pos) + '\n')
60              res_file.write("neg: " + str(neg) + '\n')
61              res_file.write("neu: " + str(neu) + '\n')
62              total = pos + neg + neu
63              # write the number of positive, negative and neutral
```

```
64              # comments to the conclusion file
65
66              if total != 0:
67                  pos_rate = float(pos) / (pos + neg + neu)
68                  neg_rate = float(neg) / (pos + neg + neu)
69              else:
70                  pos_rate = 0
71                  neg_rate = 0
72              res_file.write("pos_rate: " + str(pos_rate) + '\n')
73              res_file.write("neg_rate: " + str(neg_rate) + '\n')
74              # write the rate of positive and negative comments
75              # to the conclusion file
76
77              res_file.close()
```

## 2.3 前端搭建

这一部分主要进行前端部分实现的展示以及核心代码的呈现和解释。

### 2.3.1 主页设计

Taobao goods Search.html 实现了主页的设计，使用了 style.css 作为样式，核心代码的实现如下：

```
1  <div id="header">
2      <div id="menu">
3          <ul>
4              <li class="current_page_item"><a href="/home">Home</a></li>
5              <li><a href="/about">Brand</a></li>
6              <li><a href="/contact">Kinds</a></li>
7          </ul>
8      </div>
9      <div id="search">
10         <form id="searchform" method="get" action="/result">
11             <fieldset>
12             <input id="s" type="text" name="keyword" value="" class="text">
13             <input id="x" type="submit" value="Search" class="button">
14             </fieldset>
15         </form>
16     </div>
17 </div>
18 <div id="logo">
19 </div>
```

### 2.3.2 品牌分类页

about.html 为品牌分类页，提供了 logo 搜索，以图搜图，部分品牌介绍等功能，使用了 style2.css 作为样式，具体前端展示代码如下：

```
<div id="pageBrand">
    <div id="contentBrand">
        <div class="post">
            <h1 class="title">Brand</h1>
            <div class="entryBrand">
                <h1>logo搜索</h1>
                <form method="POST" action="/logo" enctype="multipart/form-data">
                    <input type="file" name="image" accept="image/*">
                    <input type="submit" value="上传图片">
                </form>
                <h1>以图搜图</h1>
                <form method="POST" action="/pic" enctype="multipart/form-data">
                    <input type="file" name="image" accept="image/*">
                    <input type="submit" value="上传图片">
                </form>
                <h1 class="name">常见品牌推荐</h1>
                    <ul>
                        <li>
                            <h2><a href="{{ url_for('result', keyword='李宁') }}">李宁</a></h2>
                            <p>李宁，是"体操王子"李宁在1990年创立的专业体育品牌。
                                李宁公司拥有完善的品牌营销、研发、设计、制造、经销及销售能力，以经营李宁品牌专业及休闲运动鞋、服装、器材
                        </li>
                        <li>
                            <h2><a href="{{ url_for('result', keyword='ANTA') }}">ANTA</a></h3>
                            <p>'安踏始创于一九九一年，而安踏体育用品有限公司（股份代号：2020（港币柜台）及82020（人民币柜台））在二零零
```

### 2.3.3 衣服种类页

contact.html 为衣服种类页，提供衣服穿搭 gpt 推荐，按衣服适用人群场合，具体分类搜索等功能，使用了 style2.css 作为样式，具体实现代码如下：

```
<div id="pageBrand">
    <div id="contentBrand">
        <div class="post">
            <h1 class="title">Kinds</h1>
            <div id="search-g">
                <form id="searchform" method="get" action="/gpt">
                    <fieldset>
                    <input id="s-g" type="text" name="question" value="" class="text">
                    <input id="x-g" type="submit" value="Ask GPT For Advice" class="button">
                    </fieldset>
                </form>
            </div>
            <h1>Kinds of Clothes</h1>
            <div class="kinds-container">
                <!-- Contact info start -->
                    <div class="kind-item">
                        <a href="{{ url_for('result', keyword='男') }}">
                            <img src="{{ url_for('static', filename='images/men.jpg') }}" alt='男' title = '男装'>
                        </a>
                    </div>
                    <div class="kind-item">
                        <a href="{{ url_for('result', keyword='女') }}">
                            <img src="{{ url_for('static', filename='images/women.jpg') }}" alt='女' title = '女装'>
                    </a>
                    </div>
                        <div class="kind-item">
                            <a href="{{ url_for('result', keyword='童') }}">
                                <img src="{{ url_for('static', filename='images/kid.jpg') }}" alt='童' title = '童装'>
```

### 2.3.4 搜索结果页

search.html 为搜索结果页，采用白色背景以避免信息过多影响阅读，提供排序方式选择，价格区间，品牌筛选，分页展示等多种搜索结果筛选形式，实现了以图片为主要呈现形式的商品展示方式，使用了 style2.css 作为样式，核心代码如下：

```
<div class="product-container">
                {% for i in range(0,length)%}
                <div class="product-item">
                    <a href="{{u[i]}}" target="_blank">
                        <img src={{img[i]}} alt={{t[i]}} title = "{{t[i]}}">
                    </a>
                    <p>商品名称<p>
                    <p><strong></strong>{{t[i]}}</strong></p>
                    <p>品牌 : {{brand[i]}}</p>
                    <p>好评率 : {{pos[i]}}</p>
                    <p class="price">综合评价指数 : {{com[i]}}</p>
                    <p class="price">价格 : {{p[i]}}</p>
                </div>
                {%endfor%}
            </div>
```

### 2.3.5 搜索结果页

logoSearch.html 为 logo 搜索结果页，提供 logo 搜索结果与相关的商品，以相对商品指数进行排序，使用了 style2.css 作为样式，具体代码如下：

```
<div id="content">
        <div class="post">
            <h1 class="title">Search Results</h1>
            <h1 class="name">upload logo might be {{brand[0]}}</h1>
            <div class = "entry">
                <h2 style="color:purple;">Reupload your logo picture
                :</h2>
                <form method="POST" action="/logo"
                enctype="multipart/form-data">
                    <input type="file" name="image"
                    accept="image/*">
                    <input type="submit" value="上传图片">
                </form>
            <h2 style="color:purple;">
            Displaying {{ length }}results for your search:</h2>
                <!-- List of search results start -->
            <div class="product-container">
                {% for i in range(0,length)%}
                <div class="product-item">
                    <a href="{{u[i]}}" target="_blank">
                        <img src={{img[i]}} alt={{t[i]}} title = "{{t[i]}}">
```

```
                    </a>
                    <p>商品名称<p>
                    <p><strong></strong>{{t[i]}}</strong></p>
                    <p>品牌 : {{brand[i]}}</p>
                    <p>好评率 : {{pos[i]}}</p>
                    <p class="price">综合评价指数 : {{com[i]}}</p>
                    <p class="price">价格 : {{p[i]}}</p>
                </div>
                {%endfor%}
            </div>
                <br/>
                <br/>
                <!-- More search results can be added here -->
            </div>
        </div>
    </div>
```

### 2.3.6 以图搜图结果展示页面

picSearch.html 为以图搜图搜索结果页，提供数据库中与上传图像相似程度较高的商品，使用了 style2.css 作为样式，核心代码的实现如下：

```
<div id="content">
        <div class="post">
            <h1 class="title">Search Results</h1>

            <div class = "entry">
                <h2 style="color:purple;">Reupload your picture :</h2>
                <form method="POST" action="/pic"
                enctype="multipart/form-data">
                    <input type="file" name="image" accept="image/*">
                    <input type="submit" value="上传图片">
                </form>
            <h2 style="color:purple;">Displaying {{ length }}
            results for your search:</h2>
                <!-- List of search results start -->
            <div class="product-container">
                {% for i in range(0,length)%}
                <div class="product-item">
                    <a href="{{u[i]}}" target="_blank">
                        <img src={{img[i]}} alt={{t[i]}} title = "{{t[i]}}">
                    </a>
                    <p>商品名称<p>
                    <p><strong></strong>{{t[i]}}</strong></p>
                    <p>品牌 : {{brand[i]}}</p>
                    <p>好评率 : {{pos[i]}}</p>
                    <p class="price">综合评价指数 : {{com[i]}}</p>
```

```
                    <p class="price">价格 : {{p[i]}}</p>
                </div>
                {%endfor%}
            </div>
                <br/>
                <br/>
                <!-- More search results can be added here -->
            </div>
        </div>
    </div>
```

### 2.3.7 chatbot 聊天返回结果界面

gpt.html 为 Chatbot 返回结果页，提供 bot 推荐的五个搜索关键词，并实现一键搜索功能，使用 style2.css 作为样式，核心代码实现如下：

```
<div id="contentBrand">
        <div class="post">
            <h1 class="title">Kinds</h1>
            <div class="kinds-contrainer">
                <!-- Contact info start -->
                <h1>GPT's recommend for your question</h1>
                    <ul>
                        <li><a href="{{ url_for('result',
                                keyword=command[0]) }}" style="font-size:
                                2em;" target="_blank">{{command[0]}}</a></li>
                        <li><a href="{{ url_for('result',
                                keyword=command[1]) }}" style="font-size:
                                2em;" target="_blank">{{command[1]}}</a></li>
                        <li><a href="{{ url_for('result',
                                keyword=command[2]) }}" style="font-size:
                                2em;" target="_blank">{{command[2]}}</a></li>
                        <li><a href="{{ url_for('result',
                                keyword=command[3]) }}" style="font-size:
                                2em;" target="_blank">{{command[3]}}</a></li>
                        <li><a href="{{ url_for('result',
                                keyword=command[4]) }}" style="font-size:
                                2em;" target="_blank">{{command[4]}}</a></li>
                    </ul>
                <!-- Contact info end -->
            </div>
        </div>
    </div>
```

### 2.3.8 渲染界面

app.py 主要是使用 flask 进行渲染，将搜索函数返回的结果进行进一步处理之后通过变量传递进入 html 之中进行进一步处理，具体代码如下：

主页代码：

```
@app.route('/home', methods=['POST', 'GET'])
def bio_data_form():
    if request.method == "POST":
        keyword = request.form['keyword']
        return redirect(url_for('result', keyword=keyword, method = 3))
return render_template("Taobao goods Search.html")
```

搜索页：

```
@app.route('/result', methods=['POST','GET'])
def result():
    t = []
    u = []
    p = []
    img = []
    b = []
    com = []
    pos = []
    name = ['搜索指数','好评率','综合评价指数','价格']

    STORE_DIR = "result/index"
    jieba.load_userdict("userdict.txt")
    print ('lucene', lucene.VERSION)
    #base_dir = os.path.dirname(os.path.abspath(sys.argv[0]))
    vm_env.attachCurrentThread()
    directory = SimpleFSDirectory(File(STORE_DIR).toPath())
    searcher = IndexSearcher(DirectoryReader.open(directory))
    # set a new similarity computing method
    searcher.setSimilarity(SimpleSimilarity()) # simplesimilarity func
    analyzer = StandardAnalyzer()#Version.LUCENE_CURRENT)
    if 'keyword' in request.args:
        keyword = request.args.get('keyword')
    elif 'keyword' in request.form:
        keyword = request.form.get('keyword')
    else:
        return redirect(url_for('bio_data_form'))

    method = int(request.args.get('method',3))
    selected_brands = []
    if 'brands' in request.form:
        selected_brands = request.form.getlist('brands')
        print(selected_brands)
```

```
    price = []
    if 'price1' in request.args:
        price1 = int(request.args.get('price1'))
        price2 = int(request.args.get('price2'))
        price.append(price1)
        price.append(price2)
    if keyword == '':
        return redirect(url_for('bio_data_form'))
    keyword = ' '.join(jieba.cut(keyword))

    products = run(searcher, analyzer,keyword,method)
    products = filter_by_price(products,price)
    products = filter_by_brand(products,selected_brands)
    for i in range(len(products)):
        t.append(products[i]['title'])
        u.append(products[i]['prd_url'])
        p.append(products[i]['price'])
        img.append(products[i]['img_url'])
        b.append(products[i]['brand'])
        com.append(round(products[i]['comprehensive_score'],4))
        pos.append(round(float(products[i]['pos_rate']),4))
    page = int(request.args.get('page', 1))
    # 当前页，默认为第一页面
    per_page = 50
    # 每页显示的结果数量，可以自己设定
    start = (page - 1) * per_page
    end = start + per_page
    paged_results = t[start:end]                    # 取当前页面的结果
    paged_urls = u[start:end]
    paged_prices = p[start:end]

    del searcher
    length = min(len(t),len(u))
    length = min(length,len(p))
    total_results = length
    total_pages = (total_results + per_page - 1) // per_page
    current_t = len(paged_results)

    return render_template("search.html", keyword=keyword,
    length=current_t, img = img, brand = b,price = price,
                          u=paged_urls, t=paged_results,
                          p=paged_prices, name = name, com = com, pos = pos,
                          page=page, total_pages=total_pages,
                          total_results = total_results, method = method)
```

### 2.3.9 两个 style 函数的编写

对页面各部分元素的字体大小，背景颜色，子元素间距等等进行了规范，对交互精细度进行了美化，比如搜索按钮在 hover 时会有放大效果，图片链接在鼠标悬停时会缩放，并显示提示框提示点击后下一步的商品名或者商品种类名，在开源网站设计了背景图和项目主页图，设计了 slogan。

## 2.4 图像识别

### 2.4.1 根据商品图片进行检索

基本思路：考虑到实际情况中用户输入商品图片的目的是检索同类商品，我们排除了提取颜色直方图算法以及对颜色匹配敏感的 LSH 算法，而 CNN 算法需要建立我们已有一万余张图片的图像特征库，这不符合搜索引擎的速度要求。因此我们选择 SIFT 算法，并利用 threading 多线程方法加快搜索速度。

代码实现：search_by_picture.py 主要功能分为三个函数，url_to_img 函数用于打开爬虫数据库中图片的 URL，读入图片信息并转化为三维或二维的数组形式。由于存在 URL 网址无法打开的情况，我们加入异常处理机制，防止程序报错中止。

```
def url_to_img(img_url):
    try:
        resp = urllib.request.urlopen(img_url)
        image = np.asarray(bytearray(resp.read()), dtype="uint8")
        image = cv2.imdecode(image, cv2.IMREAD_COLOR)
        return image
    except:
        return np.array([])
```

process_folder 函数用于对已打开的文件夹遍历爬到的 txt 文件数据，并提取爬虫数据库中图片的关键点和描述符，将其与用户输入的商品图片进行比较匹配，对于满足比例要求的好的匹配点计数，当数量达到一定阈值时，可以认为该图片与用户输入图片相似度高，并用好的匹配点个数模拟相似度评分返回结果。这一函数单独存在的意义是方便各个线程单独独立工作。核心代码如下

```
kp2, des2 = sift.detectAndCompute(goods, None)
matches = flann.knnMatch(des1, des2, k=2)
good_matches = [m for m,n in matches if m.distance < 0.75*n.distance]
if(len(good_matches) >= 20):
    with lock:
        product = getProductInfo(data, filename, len(good_matches)/40)
        products.append(product)
        results.append(products)
        if len(results) >= 50:
            stop_event.set()
```

picture_detection 函数用于遍历文件夹中的各个子文件夹，创建子线程对它们完成上述 process_folder 函数，其中也完成了创建 FLANN 匹配器帮助匹配的功能。核心代码如下

```
# 创建FLANN匹配器
    FLANN_INDEX_KDTREE = 1
    index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
    search_params = dict(checks=50)
    flann = cv2.FlannBasedMatcher(index_params,search_params)
```

```
    # 遍历文件夹中各商品图片
    results = []
    lock = threading.Lock()
    stop_event = threading.Event()
    threads = []
    for brand in os.listdir(folder_path):
        thread = threading.Thread(target=process_folder,
        args=(brand, folder_path, img, sift, flann, des1, results,
        lock, stop_event))
        threads.append(thread)
        thread.start()
    for thread in threads:
        thread.join()
```

### 2.4.2 基于 logo 的图像检索

基本思路：同任务 2.1，考虑到实际情况中用户输入 logo 图片的目的是限定商品类别，我们排除了提取颜色直方图算法以及对颜色匹配敏感的 LSH 算法。并且由于商品图片中 logo 部分所占比例小且模糊，与用户输入的 logo 图片匹配精度低，因此我们选择从网上爬取我们已有品牌数据的 12 张 logo 图片作比较。比较次数缩小到 12 次，我们采用的是 CNN 算法和 SIFT 算法双重评价标准，得到结果的精确度得到显著提升。

代码实现：search_by_logo.py 主要部分分为一个类和三个函数，完成准备工作的 getProductInfo 函数实现提取匹配好的商品信息的功能。部分代码如下

```
def getProductInfo(data, filename, score):
    product = dict()
    pos, neg, neu = 0, 0, 0
    for content in data:
        if(re.match("Title:", content)):
            content = content.lstrip("Title:").rstrip('\n')
            product["title"] = content
            continue
        if(re.match("Product URL:", content)):
            content = content.lstrip("Product URL").rstrip('\n')
            product["prd_url"] = content
            continue
```

MyCBIR 类通过已有的训练模型提取图像特征并比较两张图片的相似度。在相似度计算方面，我们采用的方法是计算两特征向量夹角的余弦值。核心代码如下

```
# 图像预处理
    def preprocess(self, img):
        input_image = default_loader(img)
        input_image = trans(input_image)
        input_image = torch.unsqueeze(input_image, 0)
        #print(input_image)
        return input_image

```

```
    # 抽取图像特征
    def features(self, x):
        x = model.conv1(x)
        x = model.bn1(x)
        x = model.relu(x)
        x = model.maxpool(x)
        x = model.layer1(x)
        x = model.layer2(x)
        x = model.layer3(x)
        x = model.layer4(x)
        x = model.avgpool(x)
        return x

    # 计算相似度
    def compare(self, other):
        v1 = self.features
        v2 = other.features
        normalized_v1 = v1 / np.linalg.norm(v1)
        normalized_v2 = v2 / np.linalg.norm(v2)
        return 1 - np.sqrt(np.sum((normalized_v1-normalized_v2)**2))
```

MySIFT 函数实现思路与任务 2.1 相同，不加以赘述。

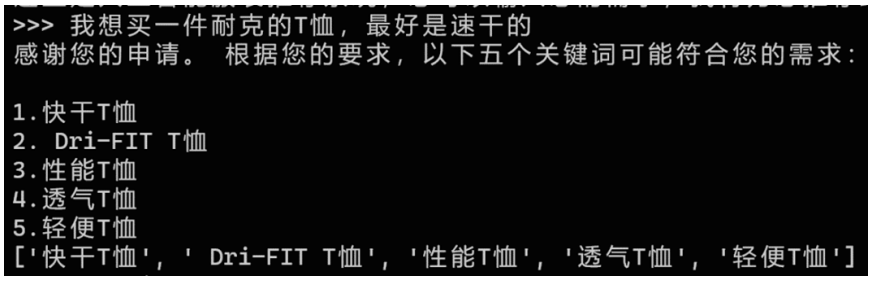
logo_detection 为两种算法的整合函数，用于遍历文件夹中的各个子文件夹，对每个品牌子文件夹中的 logo 图片分别用上述两种算法同用户输入的 logo 图片进行匹配，找到最佳匹配品牌。在两种算法结合的方式上，我们考虑的是 CNN 相似度评分 +SIFT 好的匹配点个数/1000 作为最终评分的机制。这样的匹配结果经多次实验证明效果较好。核心代码如下

```
# 遍历文件夹中各品牌的logo图片
    for brand in os.listdir(folder_path):
        match_score[brand] = 1
        for filename in os.listdir(os.path.join(folder_path, brand)):
            if (filename.endswith(".jpg") or filename.endswith(".png")):
                path = os.path.join(folder_path, brand, filename)
                print("Processing {}".format(filename))

                # 用CBIR算法计算相似度
                logo_tool = MyCBIR(path)
                match_score[brand] *= target_tool.compare(logo_tool)
                # print(match_score)

                # 用SIFT算法计算相似度
                match_score[brand] += MySIFT(target, path)/10
                # print(match_score)

    # 对比匹配结果，找到最合适匹配
    match_score = sorted(match_score.items(), key=lambda x : x[1],
    reverse=True
```

## 2.5 其他创新功能实现

### 2.5.1 AI 智能服装推荐助手

使用 gpt4all 库，利用本地部署的 gpt4all falcon LLM 模型进行服装推荐。由于在建立页面内聊天窗功能时遇到技术困难，我们最后选择针对用户输入的需求，进行单次推荐。通过适当的 prompt，让模型输出符合需求的 5 种服装类别，然后再分别进行搜索，达到推荐的效果。GPT4ALL Falcon 下载地址：https://gpt4all.io/models/gguf/gpt4all-falcon-newbpe-q4_0.gguf 由于本地部署的 gpt 性能有限，中文语言能力较差，因此使用了 translate 库进行输入和输出的翻译，以实现更好的效果。i/o 样例：




返回一个包含 5 个建议的列表，然后搜索就可以得到推荐的商品。由于容器没有安装 GLIBCXX_3.4.26 库，导致无法在容器内运行 gpt，且经过多次尝试无法安装，所以我们采用文件 i/o 的方式，通过在容器外提前运行 gpt，通过读取和写入 txt 文件来进行输入输出，既解决了容器内无法运行 gpt 的问题，又有效规避了 gpt 预热带来的延迟。人工智能 bot（gpt）提供购买建议的实现代码如下：

```
gpt 建议生成代码
def chat(model, hints, max_tokens=200, temp=0):
    # args: model: the gpt4all model
    #       hints: the prompts
    #       max_tokens: the max tokens of the output, higher
    # max_tokens means longer output
    #       temp: the temperature of the output,
    # higher temp means more randomness
    # return: the attributions of the request

    CACHE_FILE_PATH = "container/samples4/Group_Task/cache/chat_input.txt"
    OUTPUT_FILE_PATH = "container/samples4/Group_Task/cache/chat_output.txt"
    ATTRS_FILE_PATH = "container/samples4/Group_Task/cache/chat_attrs.txt"
    # cache file. read from it pre 1 second. if has content, use it
    as input, and clear it.

    with model.chat_session():
        # prompts
        for hint in hints:
            outp = model.generate(hint, max_tokens, n_batch=256, temp=temp)
        file = open(CACHE_FILE_PATH, 'w', encoding='utf-8')
        file.write('')
        file.close()
        # pre-clear the cache file
```

```
25
26          print("这里是人工智能服装推荐系统，您可以输入您的需求，
27 ␣␣␣␣␣␣␣␣我将为您推荐最适合您的服装。")
28
29          while(True):
30              file = open(CACHE_FILE_PATH, 'r', encoding='utf-8',
31              errors='ignore')
32              output_file = open(OUTPUT_FILE_PATH, 'w', encoding='utf-8')
33              attrs_file = open(ATTRS_FILE_PATH, 'w', encoding='utf-8')
34              command = file.read()
35              while(command == ''):
36                  sleep(1)
37                  print("waiting␣for␣input...")
38                  file = open(CACHE_FILE_PATH, 'r', encoding='utf-8',
39                  errors='ignore')
40                  command = file.read()
41              # detect contents in the input cache file
42
43              print("command:␣" + command)
44              hint = command
45              file = open(CACHE_FILE_PATH, 'w', encoding='utf-8')
46              file.write('')
47              file.close()
48              if hint == 'exit':
49                  break
50
51              outp = model.generate('request:␣' + translate(hint,
52              src='zh', tgt='en'), max_tokens, n_batch=256, temp=temp)
53                   # llm generate suggestions
54              outp = translate(outp)
55              output_file.write(outp + '\n')
56              attributions = search_attributions(outp + '\n')
57              attrs_file.writelines(attributions)
58              # write the output and attributions to the cache files
59
60              print(outp)
61              print(attributions)
62          print("chat␣session␣ended!")
```

具体的代码解析在注释中已经比较清楚，总体上实现了人机的交互。

## 3 可以改进之处

我们开发此次项目用了不少的时间，但仍有许多可以改进的地方：

1. 本次爬虫爬取时忘记爬取销量这一重要信息，并且由于电商网站网页不断地变更，我们的代码可能不一定一直适用，因而如果想要爬取更多有效的信息，应该爬取更多的服装网站信息，并且增强代码的鲁棒性，稳定性，以适应

网页更多的变化。

2. 本次项目没有引入数据库工作，如果可以的话，我们应该进一步实现根据用户的搜索记录来得到更加优化的搜索结果展示。

3. 前端部分前期经验不足，没有充分利用 javascript，后面开始学习想要使用的时候已经写好的代码量已经救不回去了，框架已定，导致许多功能实现比较复杂。因此后续可以通过 javascript 对页面进行优化，对代码进行简化等操作。

4.style 设计较为杂乱，查询函数较为困难，容易出现素材复用难度大的问题。因此后续应该进行素材的高效利用以及整合包装。

# 4 项目分工

陈一鸣（522030910055）：爬虫实现及网页异常处理

黄浩初（522030910076）：前端搭建及对接工作

罗宇晗（522030910023）：以图搜图及 logo 匹配工作

章凌恺（522030910056）：索引建立以及创新功能实现