

23 | 流程串联：数据处理和协同过滤串联进行内容推荐

2023-06-07 黄鸿波 来自北京

《手把手带你搭建推荐系统》



你好，我是黄鸿波。

在前面的课程中讲了很多召回算法，也讲了关于 Flask 和用户界面相关的内容，今天我们把所有的东西做一个流程串联。

今天主要会做下面五件事。

1. 将数据采集到协同过滤算法的召回中训练协同过滤算法。
2. 使用协同过滤算法训练出基于 Item 的协同过滤矩阵。
3. 利用协调过滤矩阵，将用户 ID 传入进去预测出每一个用户的 Item list。
4. 将预测出来的结果存入到 Redis 数据库。
5. 通过 WebService 做成接口。

接下来，我们针对上面的内容，看看怎么一步步实现。

训练协同过滤算法

要想把之前的那一套协同过滤算法跑起来，首先要做的是做好数据，并喂给协同过滤算法。

先来回顾一下在 [协同过滤](#) 那一节写的训练代码。

 复制代码

```
1 def cf_Item_train(self):
2     """
3
4     :return:相似度矩阵: {content_id:{content_id:score}}
5     """
6     print("start train")
7     self.Item_to_Item, self.Item_count = dict(), dict()
8
9     for user, Items in self.train.Items():
10        for i in Items.keys():
11            self.Item_count.setdefault(i, 0)
12            self.Item_count[i] += 1 # Item i 出现一次就加1分
13
14    for user, Items in self.train.Items():
15        for i in Items.keys():
16            self.Item_to_Item.setdefault(i, {})
17            for j in Items.keys():
18                if i == j:
19                    continue
20                self.Item_to_Item[i].setdefault(j, 0)
21                self.Item_to_Item[i][j] += 1 / (
22                    math.sqrt(self.Item_count[i] + self.Item_count[j])) # Item i 和 j 共现一次
23
24    # 计算相似度矩阵
25    for _Item in self.Item_to_Item:
26        self.Item_to_Item[_Item] = dict(sorted(self.Item_to_Item[_Item].Items(),
27                                              key=lambda x: x[1], reverse=True)[0:30])
```

在这段代码中，我们通过训练数据集可以取出 User 和所对应的 Item，然后最后生成一个相似度矩阵。矩阵的格式如下，这里面实际上是算出每一个内容与其他内容之间的相似度关系。

```
1 {content_id:{content_id:score}}
```

复制代码

我们继续分析这段代码，这段代码中需要的数据集就是代码中的 self.train，然后取里面的每一条内容（User 和 Items），然后再做共现矩阵。

知道这些内容能够让我们更好地去制作数据集。在数据集中，应该至少包含下面三个部分。

1. 用户 ID。
2. 内容 ID。
3. 用户对内容的评分。

我先给你一个简单的数据集参考格式。

```
1 1,2,6117caee32002fb435aab0e4
2 1,2,6117caef32002fb435aab22e
3 1,2,6117caef32002fb435aab231
4 1,3,6117caef32002fb435aab242
5 4,2,6117caef32002fb435aab23d
6 4,17,6117caef32002fb435aab232
7 4,15,6117caef32002fb435aab231
8 4,13,6117caef32002fb435aab22e
9 4,16,6117caef32002fb435aab23c
10 3,4,6117caef32002fb435aab231
```

复制代码

我们要做的数据集就是上面的这种形式。这里面一共分成三列：第一列是用户的 ID、第三列是内容 ID，而第二列就是用户对内容的评分。

这里面还有一个小小的疑问，那就是中间的评分是怎么来的呢？

还记得 **加权热度** 吗？比如阅读加 1 分、点赞加 2 分、收藏加 2 分、评论加 3 分，今天我们就在这个基础上来再做一个加权，形成数据集的分数。

我们可以设置下面这么一个规则。

阅读加 1 分。

点赞加 2 分。

收藏加 3 分。

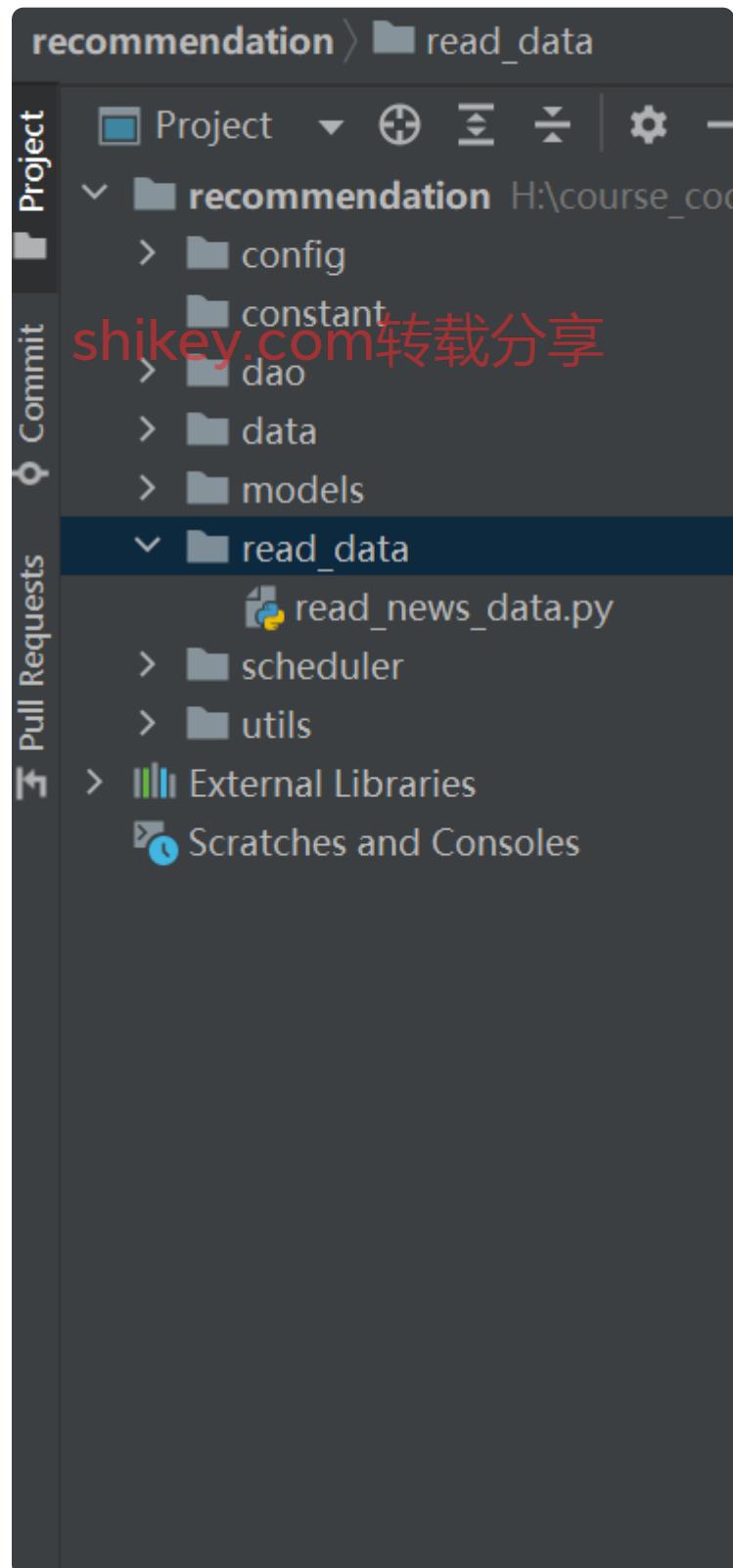
shikey.com转载分享

如果同时存在 2 项多加 1 分。

如果同时存在 3 项多加 2 分。

接下来从 MongoDB 数据库中读取需要的数据，再通过规则来进行加权。最后再把得分存入到一个 CSV 表中，供训练使用。

我们在 recommendation-class 项目里新建一个叫 read_data 的目录，以后读取数据都用这个目录，然后再在里面新建一个叫 read_news_data.py 的文件，新建之后目录格式如下。



先来看代码。

```
1 from dao.mongo_db import MongoDB  
2 import os
```

复制代码

```
3
4
5 class NewsData(object):
6     def __init__(self):
7         self.mongo = MongoDB(db='recommendation')
8         self.db_client = self.mongo.db_client
9         self.read_collection = self.db_client['read']
10        self.likes_collection = self.db_client['likes']
11        self.collection = self.db_client['collection']
12        self.content = self.db_client['content_labels']
13
14    """
15        阅读 1
16        点赞 2
17        收藏 3
18
19        如果同时存在2项 加 1分
20        如果同时存在3项 加 2分
21    """
22    def cal_score(self):
23        result = list()
24        score_dict = dict()
25        data = self.likes_collection.find()
26        for info in data:
27            #这里面做分数的计算
28            score_dict.setdefault(info['user_id'], {})
29            score_dict[info['user_id']].setdefault(info['content_id'], 0)
30
31            query = {"user_id": info['user_id'], "content_id": info['content_id']}
32
33            exist_count = 0
34            # 去每一个表里面进行查询，如果存在数据，就加上相应的得分
35            read_count = self.read_collection.find(query).count()
36            if read_count > 0:
37                score_dict[info['user_id']][info['content_id']] += 1
38                exist_count += 1
39
40            like_count = self.likes_collection.find(query).count()
41            if like_count > 0:
42                score_dict[info['user_id']][info['content_id']] += 2
43                exist_count += 1
44
45            collection_count = self.collection.find(query).count()
46            if collection_count > 0:
47                score_dict[info['user_id']][info['content_id']] += 2
48                exist_count += 1
49
50            if exist_count == 2:
51                score_dict[info['user_id']][info['content_id']] += 1
```

```

52         elif exist_count == 3:
53             score_dict[info['user_id']][info['content_id']] += 2
54         else:
55             pass
56
57         result.append(str(info['user_id']) + ',' + str(score_dict[info['user_'
58
59             self.to_csv(result, '../data/news_score/news_log.csv')
60
61     def rec_user(self):
62         data = self.read_collection.distinct('user_id')
63         return data
64
65     def to_csv(self, user_score_content, res_file):
66         if not os.path.exists('../data/news_score'):
67             os.mkdir('../data/news_score')
68         with open(res_file, mode='w', encoding='utf-8') as wf:
69             # info = "1,8,6145ec828451a2b8577df7b3"
70             for info in user_score_content:
71                 wf.write(info + '\n')

```

解释下这段代码，在项目的最开始，我们初始化了 MongoDB 数据库，并将数据库的各个集合存储到了相应的变量中。

接着我们定义了一个 cal_score() 函数，这个函数的主要作用就是计算每个用户对于每篇新闻的得分，该函数遍历点赞数据集 (likes_collection)，对每个用户和新闻计算其得分。计算该用户点赞该新闻的次数 (like_count)、阅读该新闻的次数 (read_count) 和收藏该新闻的次数 (collection_count)，对于每个存在的操作，可以给相应的新闻加分（点赞 +2，阅读 +1，收藏 +2）。如果同时存在 2 项操作，则额外加 1 分，如果同时存在 3 项操作，则额外加 2 分。然后将每个用户的得分和对应的新闻 ID 写入 CSV 文件中。

最后我们还实现了两个函数，一个是 rec_user() 函数，用来计算用户；另一个是 to_csv 函数，用来将计算好的矩阵与 CSV 格式存储。

训练出基于 Item 的协同过滤矩阵

有了数据之后，就可以使用协同过滤算法来进行计算了。

我们在讲 [协同过滤](#) 时讲了怎么训练和推理，现在补全读取数据这一部分。

我们进到 recommendation-class 这个项目，来到 models 目录下的 recall 目录，在这里找到之前写的 Item_base_cf.py 文件。

首先，这个文件应该是在一个大类里，所以要建立一个类为 ItemBaseCF，并在里面去写 init 函数，代码如下。

shikey.com转载分享

复制代码

```
1 class ItemBaseCF(object):
2     def __init__(self, train_file):
3         """
4             读取文件
5             用户和Item历史
6             Item相似度计算
7             训练
8             """
9         self.train = dict()
10        self.user_Item_history = dict()
11        self.Item_to_Item = dict()
12        self.read_data(train_file)
```

这段代码的 init 函数下面有几个变量，我们分别来看一下。

首先在 init 函数中传入一个 train_file 文件（也就是训练文件），这个训练文件就是在前面保存到 CSV 中的文件。

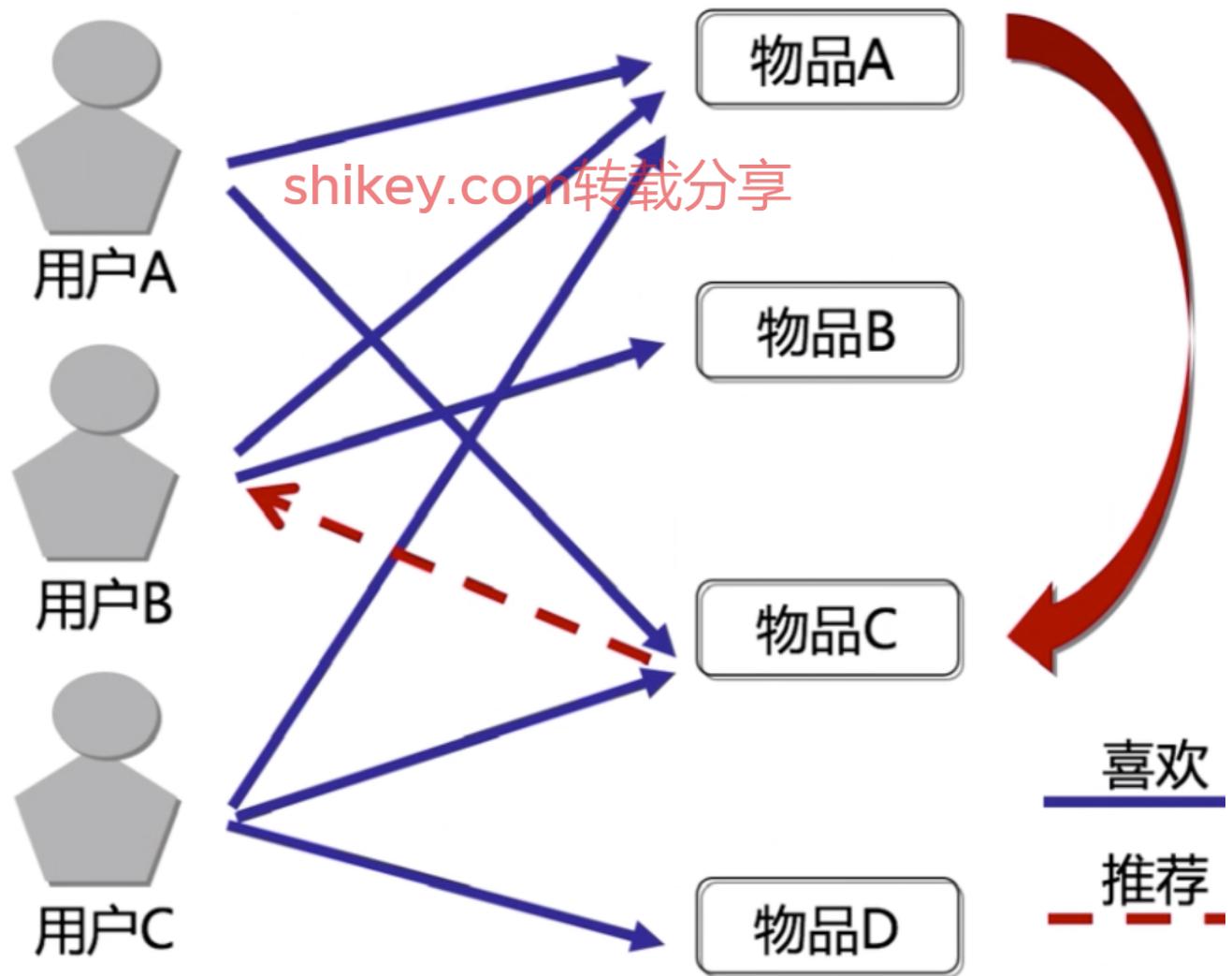
然后我们定义了以下三个变量。

self.train 是一个字典，主要用来保存训练数据。读取出来的数据最终要保存成一个字典，然后从字典中获取相应的数据再进行训练。

self.user_Item_history 变量是用来保存用户读取内容的一个字典，这个字典的格式是 self.user_Item_history[“user_id”] 中用户所阅读过的所有内容列表。

self.Item_to_Item 变量实际上是一个内容的相似度矩阵，这个矩阵是整个协同过滤的核心所在。在 self.Item_to_Item 中会存放每一个内容与其他内容的相似度，这个相似度一般是对称的。

先来看下面这张图。



这张图是一张典型的基于 Item 的 **协同过滤图**，左侧是用户，右侧是物品（物品可以理解为推荐系统中的内容）。

用户 A 买了物品 A 和 C，用户 B 买了物品 A 和 B，我们假设物品 A 和物品 C 的相似度很高，那么这个时候，就会把物品 C 推荐给用户 B。这个推荐实际上就是利用了基于 Item 的协同过滤，再把这四个物品转换成一个表格来表示相似度，就会看到下面这样一个表格。

	物品A	物品B	物品C	物品D
物品A	1	0.6	0.8	0.5
物品B	0.6	1	0.7	0.2
物品C	0.8	0.7	1	0.1
物品D	0.5	0.2	0.1	1



这个表只是一个例子，在这个例子中，两个相同的物品相似度一定为 1，那么可以看到 A 对 B 和 B 对 A 的相似度一定是一样的，同理 A 对 C 和 C 对 A 的相似度也一定是一样的，以此类推可以发现，这个表有一个很明显的特性，就是对称性。也就是说在实际的协同过滤算法中，我们会只计算其中的一半数据，这样可以减少计算量。在存储到字典中时，就会减少一半数据的大小，使得推理速度加快。

最后一个变量实际上是一个读取数据的函数，来看下这部分代码应该怎么来写。

复制代码

```

1 def read_data(self, train_file):
2     """
3         读文件，并生成数据集（用户、分数、新闻，user,score,Item）
4         :param train_file: 训练文件
5         :return: {"user_id": {"content_id": predict_score}}
6     """
7     with open(train_file, mode='r', encoding='utf-8') as rf:
8         for line in tqdm(rf.readlines()):
9             user, score, Item = line.strip().split(",")
10            self.train.setdefault(user, {})
11            self.user_Item_history.setdefault(user, [])
12            self.train[user][Item] = int(score)
13            self.user_Item_history[user].append(Item)

```

这段代码将前面生成的 CSV 数据集传入进来，然后使用 with open 方法打开。打开之后，逐行读取文件中的每一行数据，并使用 split 进行分隔，这样就可以把用户 ID、分数、文章 ID

分开并分别赋值。

在这里，我们使用 `setdefault` 方法将用户 ID、评分和新闻 ID 添加到字典 `self.train` 中。如果用户 ID 不存在，新建一个空字典；如果用户已存在，则直接添加。用户读取内容的历史操作也是同理。

shikey.com转载分享

接着，将评分转换为整数类型，并将用户 ID 和新闻 ID 添加到 `self.train` 字典中。然后把新闻 ID 添加到 `self.user_Item_history` 字典中，表示该用户已经浏览过该新闻。

最终，这段代码返回一个字典，其中键为用户 ID，值为字典，表示该用户浏览过的新闻及评分。另外，这段代码还把每个用户浏览过的新闻 ID 添加到 `self.user_Item_history` 字典中，以便后续使用。

搭建整个运行流程

前面的步骤相当于已经把相关的数据跑通了，下一步就是搭建整个流程，把它们串起来。

我们在 `scheduler` 目录下新建一个 `sched_rec_news.py` 文件，主要用来跑通整个协同过滤从数据进入到将结果存入到数据库的流程，需要做下面这四个步骤。

1. 知道要推荐给谁，也就是要先计算一下推荐用户的列表，分成冷启动和有推荐记录两种，只需要计算有阅读记录的人。
2. 通过训练，得到协同过滤矩阵。
3. 做推荐。
4. 把推荐的结果写到数据库里面，以备后面应用。

我们直接来看代码。

 复制代码

```
1 from read_data import read_news_data
2 from models.recall.Item_base_cf import ItemBaseCF
3 import pickle
4 from dao import redis_db
```

```
5
6
7 class SchedRecNews(object):
8     def __init__(self):
9         self.news_data = read_news_data.NewsData()
10        self.Redis = redis_db.Redis()
11
12    def schedule_job(self):
13        """
14            首先我们要知道要推荐给谁，也就是说，我们要先计算一下推荐用户的列表，分成冷启动、有推
15            2、我们通过训练，得到协同过滤矩阵
16            3、做推荐
17            4、把推荐的结果写到数据库里面，以备后面应用
18        :return:
19        """
20
21        user_list = self.news_data.rec_user()
22        # self.news_data.cal_score()
23        self.news_model_train = ItemBaseCF("../data/news_score/news_log.csv")
24        self.news_model_train.cf_Item_train()
25        # 模型固化
26        with open("../data/recall_model/CF_model/cf_news_recommend.m", mode='wb')
27            pickle.dump(self.news_model_train, article_f)
28        for user_id in user_list:
29            self.rec_list(user_id)
30
31    def rec_list(self, user_id):
32        recall_result = self.news_model_train.cal_rec_Item(str(user_id))
33        recall = []
34        scores = []
35        for Item, score in recall_result.Items():
36            recall.append(Item)
37            scores.append(score)
38        data = dict(zip(recall_result, scores))
39        self.to_redis(user_id, data)
40        print("Item_cf to redis finish...")
41
42    def to_redis(self, user_id, rec_content_score):
43        rec_Item_id = "rec_Item:" + str(user_id)
44        res = dict()
45        for content, score in rec_content_score.Items():
46            res[content] = score
47
48        if len(res) > 0:
49            data = dict({rec_Item_id: res})
50            for Item, value in data.Items():
51                self.Redis.redis.zadd(Item, value)
52
53 if __name__ == '__main__':
```

shikey.com转载分享

```
54     sched = SchedRecNews()  
55     sched.schedule_job()  
56
```

解释下这段代码。

- 解释下这段代码。
1. 创建一个 SchedRecNews 类对象，其中包含新闻数据类对象和 Redis 数据库类对象。
2. 调用 NewsData 类中的 rec_user 函数，获取需要进行推荐的用户列表。
3. 创建一个 ItemBaseCF 类对象并传入新闻评分数据的文件路径，然后调用 ItemBaseCF 类中的 cf_Item_train 函数，训练基于物品的协同过滤模型。
4. 将训练好的模型固化保存到本地文件中。
5. 对于每个用户调用 rec_list 函数进行推荐。
6. 在 rec_list 函数中，调用 ItemBaseCF 类中的 cal_rec_item 函数获取每个用户的推荐列表（包括推荐内容和推荐得分），然后将推荐列表和相应的推荐得分保存到 Redis 数据库中。
7. 完成任务，程序结束。

做成 WebService 服务

接下来就是把协同过滤算法的输出和推荐接口进行对接，从而完成整个流程。

在这个阶段，我们要用的是 recommendation-service 这个项目，要做的有以下两件事。

1. 在 Redis 数据库中进行查询，然后把数据返回给前端进行展示。
2. 如果查询结果是空，还是走之前的接口，可以将它理解成为一个冷启动。

首先回忆一下之前的推荐接口代码。

复制代码

```
1 @app.route("/recommendation/get_rec_list", methods=['POST'])  
2 def get_rec_list():  
3     if request.method == 'POST':
```

```

4     req_json = request.get_data()
5     rec_obj = json.loads(req_json)
6     page_num = rec_obj['page_num']
7     page_size = rec_obj['page_size']
8
9     try:
10         data = page_query.get_data_with_page(page_num, page_size)
11         print(data)
12         return jsonify({"code": 0, "msg": "请求成功", "data": data})
13     except Exception as e:
14         print(str(e))
15         return jsonify({"code": 2000, "msg": "error"})

```

这段代码实际上是请求一个冷启动的翻页代码。也就是说用户进来之后不管是谁，都会按照时间倒序进行推荐。但现在有了协同过滤，我们应该将协同过滤的结果引入进来，需要做下面这么两个更改。

1. 接收的参数增加 user_id 这个字段。
2. 拿到 user_id 去 Redis 数据库中进行查询，如果查到了就把里面的推荐列表给到前端；如果查不到，继续走冷启动（也就是按照时间排序进行推荐）。

先把从 Redis 中查找是否存在可推荐数据的方法写出来。我们在 utils 下面建立一个叫 redis_query.py 的文件，然后在里面写入下面的内容。

 复制代码

```

1 from dao import redis_db
2
3
4 class RedisQuery:
5     def __init__(self):
6         self.redis_client = redis_db.Redis()
7
8     def check_key_exist(self, key):
9         return self.redis_client.redis.exists(key)
10
11 这段代码很简单，就是把redis导入进去之后，使用redis.exists()命令查看key是不是存在，如果存在就
12 紧接着，我们在这段代码中增加一个函数：
13     def get_data_with_redis(self, user_id, page_num, page_size):
14         redis_key = "rec_Item:" + str(user_id)
15         if self.check_key_exist(redis_key):
16             start_index = (page_num - 1) * page_size

```

```
17     result = self.redis_client.redis.zrange(redis_key, start_index, end_i
18
19     lst = list()
20     for x in result:
21         info = self._redis.get("news_detail:" + x)
22         lst.append(info)
23     return lst
```

shikey.com转载分享

这个函数就是从 Redis 中取数据，然后获得内容的 ID 列表，再从 Redis 的 “news_detail:” 中获取相应的数据，返回给前端。

最后把推荐接口的代码再合入进来，变成如下代码即可。

复制代码

```
1 @app.route("/recommendation/get_rec_list", methods=['POST'])
2 def get_rec_list():
3     if request.method == 'POST':
4         req_json = request.get_data()
5         rec_obj = json.loads(req_json)
6         user_id = rec_obj['user_id']
7         page_num = rec_obj['page_num']
8         page_size = rec_obj['page_size']
9
10    try:
11        redis_key = "rec_Item:" + str(user_id)
12        if redis_query.check_key_exist(redis_key):
13            data = redis_query.get_data_with_redis(user_id, page_num, page_si
14            print(data)
15            return jsonify({"code": 0, "msg": "请求成功", "data": data})
16        else:
17            data = page_query.get_data_with_page(page_num, page_size)
18            print(data)
19            return jsonify({"code": 0, "msg": "请求成功", "data": data})
20    except Exception as e:
21        print(str(e))
22        return jsonify({"code": 2000, "msg": "error"})
```

总结

到目前为止，整个流程就已经串起来了，接下来我对这节课做一个总结。

首先，训练基于 Item 的协同过滤矩阵的一般步骤：确定推荐对象、计算推荐矩阵、进行推荐、记录推荐结果。

其次，你应该熟悉如何将数据和协同过滤算法串联起来，并存入到 Redis 数据库。

最后，熟悉推荐系统的流程，shike.com转载分享

1. 确定推荐对象：需要确定推荐系统的用户群体，并对他们的阅读记录进行收集和分析。同时，针对不同的用户群体，还需要选择不同的推荐算法和模型。
2. 计算推荐矩阵：得到用户的阅读记录之后，通过训练生成协同过滤矩阵（它反映了用户与文章之间的关联度），能够告诉我们哪些文章与用户更相关。
3. 进行推荐：基于得到的协同过滤矩阵就可以对用户进行推荐了。根据不同的算法和模型，我们可以选择不同的推荐方式，如基于用户的协同过滤、基于物品的协同过滤、基于内容的推荐等。
4. 记录推荐结果：把推荐的结果写到数据库里并实时更新，这样可以为后续应用提供支持和展示。同时，记录推荐结果也可以反哺推荐算法的迭代，提升推荐效果。

课后练习

这节课学完了，给你留两道课后题。

1. 实现上面的代码。
2. 把前端和这个推荐的结果串联起来。

期待你的分享，如果今天的内容让你有所收获，也欢迎你推荐给有需要的朋友！

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言 (1)



peter

2023-06-08 来自北京

请问：“共现矩阵”就是“协同过滤矩阵”吗？

作者回复：是的。



shikey.com转载分享