基于Django的用户协同与物品协同过滤推荐系统

各种推荐系统

图书管理系统、电影推荐系统、在线选修课程系统、健康知识推荐系统等。

有开源的、现成的、可定制。

联系我

微信 **1257309054**

点我添加

一、环境

python版本3.7，Djangn版本3，Mysql版本5.7。

三、项目Demo

1、图书推荐系统

在线demo传送门1

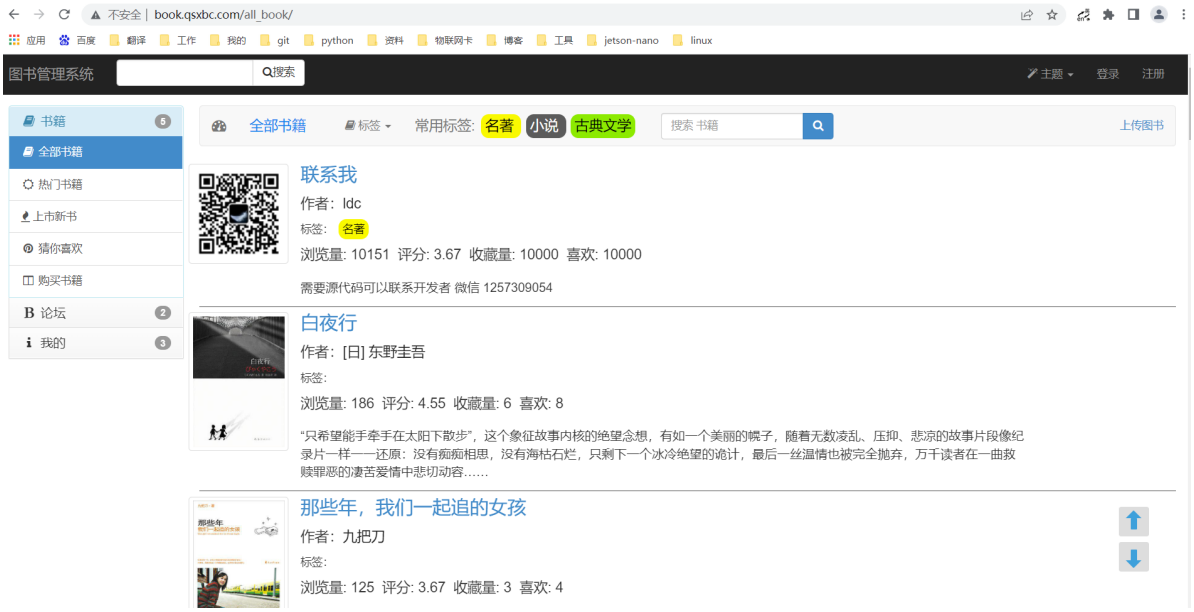
在线demo传送门2

详细讲解传送门

开源传送门

1.1 具体功能

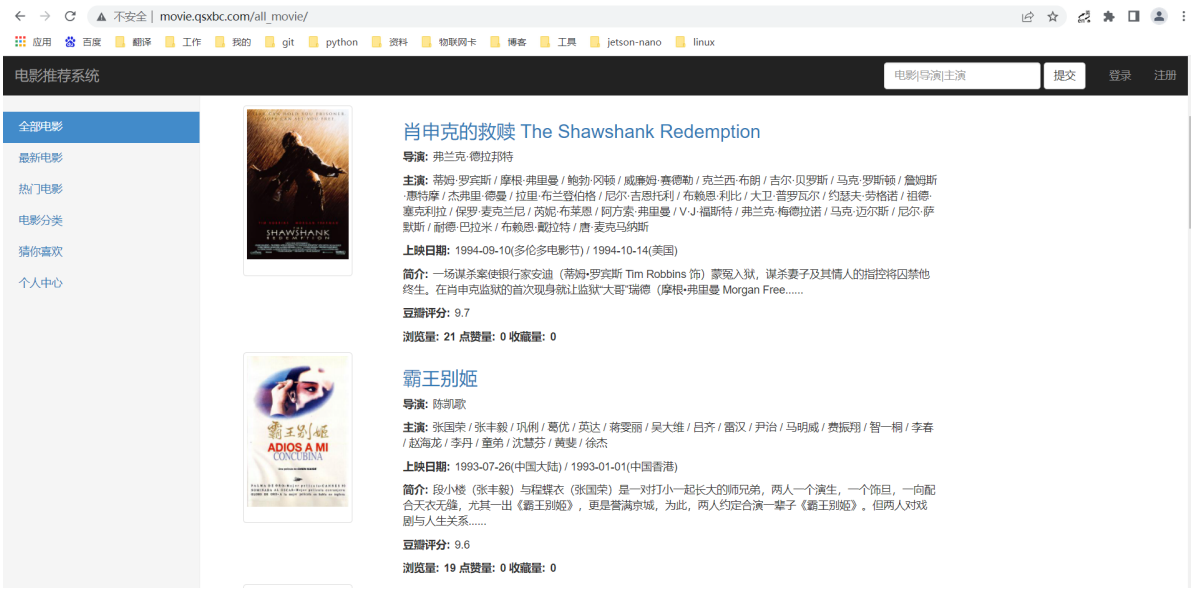
登录、注册、搜索、全部书籍、热门书籍、上市新书、书籍分类、点赞、评论、收藏、论坛、购买书籍、购物车、立即支付、图书借阅、个人中心、物流状态。



2、电影推荐系统

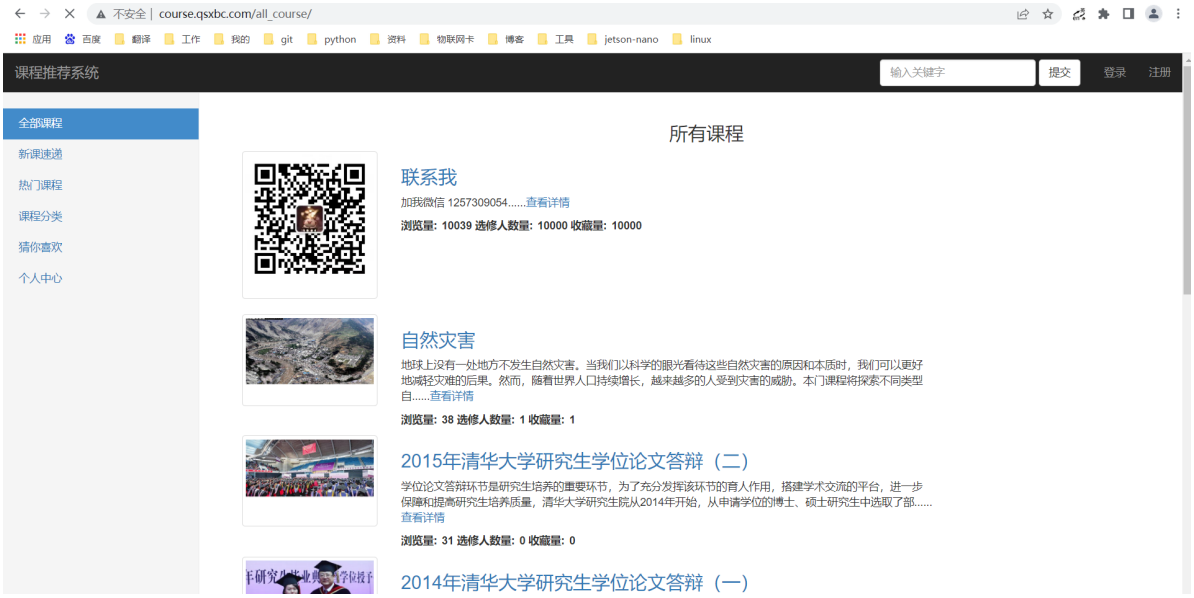
传送门

登录、注册、搜索、全部电影、最新电影、热门电影、电影分类、电影观看、猜你喜欢、个人中心、点赞、评论、收藏。



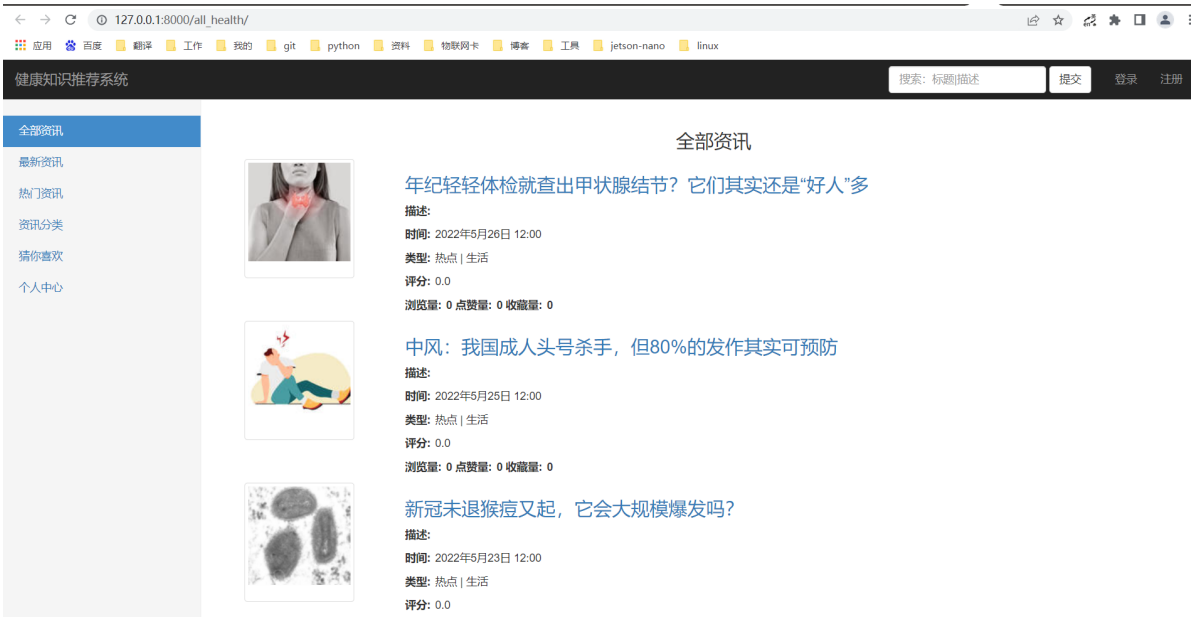
3、在线选修课程推荐系统

登录、注册、搜索、全部课程、最新课程、热门课程、课程分类、课程选修、猜你喜欢、个人中心、点赞、评论、收藏。



4、健康知识推荐系统

登录、注册、搜索、全部资讯、最新资讯、热门资讯、资讯分类、猜你喜欢、个人中心、点赞、评论、收藏。



二、用户协同过滤推荐

[算法讲解传送门]

```python
import os
import django
import operator
from health.models import *
from math import sqrt, pow

os.environ["DJANGO_SETTINGS_MODULE"] = "health.settings"
django.setup()


```

```
class UserCf:
    # 基于用户协同算法来获取推荐列表
    """
    利用用户的群体行为来计算用户的相关性。
    计算用户相关性的时候我们就是通过对比他们对相同物品打分的相关度来计算的

    举例：

    --------+--------+--------+--------+--------+
            |   X    |    Y   |    Z   |    R   |
    --------+--------+--------+--------+--------+
        a   |   5    |    4   |    1   |    5   |
    --------+--------+--------+--------+--------+
        b   |   4    |    3   |    1   |    ?   |
    --------+--------+--------+--------+--------+
        c   |   2    |    2   |    5   |    1   |
    --------+--------+--------+--------+--------+

    a用户给X物品打了5分，给Y打了4分，给Z打了1分
    b用户给X物品打了4分，给Y打了3分，给Z打了1分
    c用户给X物品打了2分，给Y打了2分，给Z打了5分

    那么很容易看到a用户和b用户非常相似，但是b用户没有看过R物品，
    那么我们就可以把和b用户很相似的a用户打分很高的R物品推荐给b用户，
    这就是基于用户的协同过滤。
    """

    # 获得初始化数据
    def __init__(self, data):
        self.data = data

    # 通过用户名获得资讯列表，仅调试使用
    def getItems(self, username1, username2):
        return self.data[username1], self.data[username2]

    # 计算两个用户的皮尔逊相关系数
    def pearson(self, user1, user2):  # 数据格式为：资讯id，浏览次数
        print("user message", user1)
        sumXY = 0.0
        n = 0
        sumX = 0.0
        sumY = 0.0
        sumX2 = 0.0
        sumY2 = 0.0
        for health1, score1 in user1.items():
            if health1 in user2.keys():  # 计算公共的资讯浏览次数
                n += 1
                sumXY += score1 * user2[health1]
                sumX += score1
                sumY += user2[health1]
                sumX2 += pow(score1, 2)
                sumY2 += pow(user2[health1], 2)
        if n == 0:
            print("p氏距离为0")
            return 0
        molecule = sumXY - (sumX * sumY) / n
        denominator = sqrt((sumX2 - pow(sumX, 2) / n) * (sumY2 - pow(sumY,
2) / n))
```

```
        if denominator == 0:
            print("共同特征为0")
            return 0
        r = molecule / denominator
        print("p氏距离:", r)
        return r

    # 计算与当前用户的距离，获得最临近的用户
    def nearest_user(self, username, n=1):
        distances = {}
        # 用户，相似度
        # 遍历整个数据集
        for user, rate_set in self.data.items():
            # 非当前的用户
            if user != username:
                distance = self.pearson(self.data[username],
self.data[user])
                # 计算两个用户的相似度
                distances[user] = distance
        closest_distance = sorted(
            distances.items(), key=operator.itemgetter(1), reverse=True
        )
        # 最相似的N个用户
        print("closest user:", closest_distance[:n])
        return closest_distance[:n]

    # 给用户推荐资讯
    def recommend(self, username, n=1):
        recommend = {}
        nearest_user = self.nearest_user(username, n)
        for user, score in dict(nearest_user).items():  # 最相近的n个用户
            for health_id, scores in self.data[user].items():  # 推荐给用户的
资讯列表
                # 如果推荐用户评分低于3分，则表明用户不喜欢此资讯，则不推荐给别的用户
                rate_rec = RateHealth.objects.filter(health_k_id=health_id,
user__username=user) # 推荐用户的评分
                if rate_rec and rate_rec.first().score < 3:
                    continue
                # 如果用户已评分过，则不推荐给用户
                rate_obj = RateHealth.objects.filter(health_k_id=health_id,
user__username=username) # 用户的评分
                if rate_obj :
                    continue

                if health_id not in recommend.keys():  # 添加到推荐列表中
                    recommend[health_id] = scores
        # 对推荐的结果按照资讯浏览次数排序
        return sorted(recommend.items(), key=operator.itemgetter(1),
reverse=True)


def recommend_by_user_id(user_id, health_id=None):
    # 通过用户协同算法来进行推荐
    current_user = User.objects.get(id=user_id)
    # 如果当前用户没有打分 则返回空列表
    if current_user.ratehealth_set.count() == 0:
        return []

```

```
    users = User.objects.all()
    all_user = {}
    for user in users:
        rates = user.ratehealth_set.all()
        rate = {}
        # 用户有给资讯打分
        if rates:
            for i in rates:
                rate.setdefault(str(i.health_k.id), i.score)
            all_user.setdefault(user.username, rate)
        else:
            # 用户没有为资讯打过分，设为0
            all_user.setdefault(user.username, {})

    print("this is all user:", all_user)
    user_cf = UserCf(data=all_user)
    recommend_list = user_cf.recommend(current_user.username, 3)
    return recommend_list
```

三、物品协同过滤推荐

[算法讲解传送门]

```
class ItemCf:
    # 基于物品协同算法来获取推荐列表
    '''
    1.构建用户->物品的对应表
    2.构建物品与物品的关系矩阵(同现矩阵)
    3.通过求余弦向量夹角计算物品之间的相似度，即计算相似矩阵
    4.根据用户的历史记录，给用户推荐物品
    '''
    def __init__(self, user_id, health_id=None):
        self.health_id = health_id  # 资讯id
        self.user_id = user_id  # 用户id

    def get_data(self):
        # 获取用户评分过的资讯
        rate_healths = RateHealth.objects.filter()
        if not rate_healths:
            return False
        datas = {}
        for rate_health in rate_healths:
            user_id = rate_health.user_id
            if user_id not in datas:
                datas.setdefault(user_id,{})
                datas[user_id][rate_health.health_k.id] = rate_health.score
            else:
                datas[user_id][rate_health.health_k.id] = rate_health.score

        return datas

    def similarity(self, data):
        # 1 构造物品：物品的共现矩阵
        N = {}  # 喜欢物品i的总人数
        C = {}  # 喜欢物品i也喜欢物品j的人数
```

```
        for user, item in data.items():
            for i, score in item.items():
                N.setdefault(i, 0)
                N[i] += 1
                C.setdefault(i, {})
                for j, scores in item.items():
                    if j != i:
                        C[i].setdefault(j, 0)
                        C[i][j] += 1
        print("---1.构造的共现矩阵---")
        print('N:', N)
        print('C', C)
        # 2 计算物品与物品的相似矩阵
        W = {}
        for i, item in C.items():
            W.setdefault(i, {})
            for j, item2 in item.items():
                W[i].setdefault(j, 0)
                W[i][j] = C[i][j] / sqrt(N[i] * N[j])
        print("---2.构造的相似矩阵---")
        print(W)
        return W

    def recommand_list(self, data, W, user, k=15, N=10):
        '''
        # 3.根据用户的历史记录，给用户推荐物品
        :param data: 用户数据
        :param W: 相似矩阵
        :param user: 推荐的用户
        :param k: 相似的k个物品
        :param N: 推荐物品数量
        :return:
        '''

        rank = {}
        for i, score in data[user].items():  # 获得用户user历史记录，
            for j, w in sorted(W[i].items(), key=operator.itemgetter(1),
reverse=True)[0:k]:  # 获得与物品i相似的k个物品
                if j not in data[user].keys():  # 该相似的物品不在用户user的记录
里
                    rank.setdefault(j, 0)
                    rank[j] += float(score) * w  # 预测兴趣度=评分*相似度
        print("---3.推荐----")
        print(sorted(rank.items(), key=operator.itemgetter(1), reverse=True)
[0:N])
        return sorted(rank.items(), key=operator.itemgetter(1),
reverse=True)[0:N]

    def recommendation(self):
        """
        给用户推荐相似资讯
        """
        data = self.get_data()
        if not data or self.user_id not in data:
            # 用户没有评分过任何资讯，就返回空列表
            return []

        W = self.similarity(data)  # 计算物品相似矩阵
```

```
        sort_rank = self.recommand_list(data, W, self.user_id, 15, 10)  # 推荐
        return sort_rank
```

四、混合推荐

为了能够将这两种传统的协同过滤算法各自的优势充分结合起来，既使得 推荐系统能够对产生的预测或推荐结果做出比较合理的解释，又使得推荐系统 产生的预测或推荐结果具有比较强的新颖性，这里提出一种基于用户和物品的 加权型混合协同过滤算法。引入一个权重因子 w（其中 0 ≤ w ≤ 1），通过对基于 用户的和基于物品的协同过滤算法计算得出的预测兴趣偏好度求加权和来计算 活动用户对目标物品的综合兴趣偏好度。

`推荐列表 = w*P_cu + (1-w)* P_cf`

```
def recommend_by_mixture(user_id, health_id=None):
    # 混合推荐算法
    # 推荐列表 = w*P_cu + (1-w)* p_cf
    cu_list = recommend_by_user_id(user_id) # 用户协同过滤得到的推荐列表
    cf_list = ItemCf(user_id).recommendation() # 物品协同过滤得到的推荐列表
    if not cu_list:
        # 用户协同过滤推荐列表为空
        if not cf_list:
            # 物品协同过滤列表也为空，则按用户注册时选择的资讯类型各返回10本
            category_ids = []
            us = UserSelectTypes.objects.get(user_id=user_id)
            for category in us.category.all():
                category_ids.append(category.id)
            health_list = HealthKnowledge.objects.filter(category__in=category_ids).exclude(id=health_id).order_by("-like_num")
            return health_list
        # 返回物品协同过滤列表中的书籍
        health_list = HealthKnowledge.objects.filter(id__in=[s[0] for s in cf_list]).order_by("-like_num")[:3]
        return health_list
    else:
        if not cf_list:
            # 物品协同过滤列表为空，则返回用户协同过滤列表中的书籍
            health_list = HealthKnowledge.objects.filter(id__in=[s[0] for s in cu_list]).exclude(id=health_id).order_by("-like_num")[:3]
            return health_list

        # 混合推荐
        # 权重因子w
        w = 0.8
        rank = {}
        for book_id, distance in cu_list:
            cf_d = 0
            # 找到物品协同过滤列表中同一本书籍的兴趣度
            for book_id_cf, value in cf_list:
                if book_id == book_id_cf:
                    cf_d = value
                    break

            rank[book_id] = w*distance + (1-w) * cf_d
```

```
38          rank_list = sorted(rank.items(), key=operator.itemgetter(1),
    reverse=True)[:3]
39
40          health_list = HealthKnowledge.objects.filter(id__in=[s[0] for s in
    rank_list]).exclude(id=health_id).order_by("-like_num")[:3]
41          return health_list
```

五、数据爬虫

查看同级目录爬虫(spider.py)代码。