项目 3

PROJECT 3

基于网页端的电影推荐

本项目基于 Movielens 的数据集，通过机器学习算法设计合适模型进行训练，根据相似度和用户的历史行为生成推荐列表，实现网站为用户精准推荐电影的功能。

3.1 总体设计

本部分包括系统整体结构和系统流程。

3.1.1 系统整体结构

系统整体结构如图 3-1 所示。

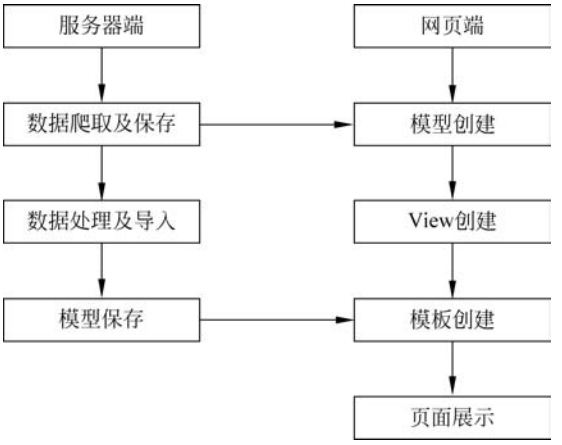

图 3-1 系统整体结构

3.1.2 系统流程

系统流程如图 3-2 所示。

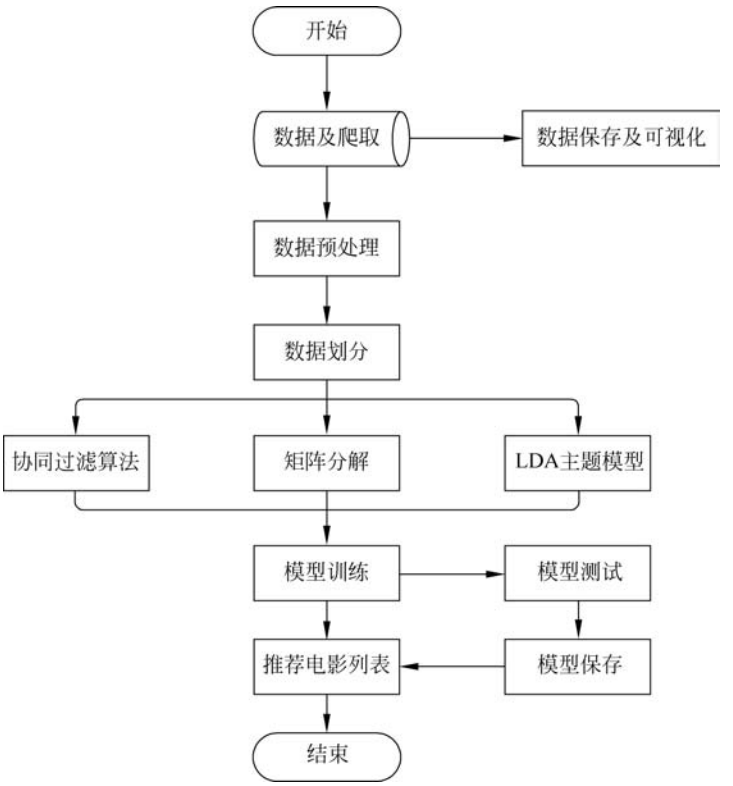

图 3-2　系统流程

3.2　运行环境

本部分包括 Python 环境、PyCharm 环境和数据库。

3.2.1　Python 环境

配置 Python 3.6 的环境，Windows 环境下载 Anaconda 完成 Python 所需配置，下载地址为 https://www.anaconda.com/。

3.2.2　PyCharm 环境

安装 PyCharm，下载地址为 https://www.jetbrains.com/pycharm/download/。新建 PyCharm 项目，打开 PyCharm，选择菜单项 File→New→New Project→Django。

Name 可自定义，Location 为项目保存的地址，Existing interpreter 为该项目使用的 Python 翻译器，需要 3.6 以上版本。单击 Create，完成新建项目。安装 Python 所需的第三

方安装包并导入,设置编译器。

3.2.3 数据库

数据库下载地址为 http://www.postgres.cn/index.php/v2/home。下载安装后用程序菜单名为 PostgreSQL11 文件夹下的 pgAdmin4 应用安装数据库。

3.3 模块实现

本项目包括 5 个模块:数据爬取及处理、模型训练及保存、接口实现、收集数据、界面设计。下面分别介绍各模块的功能及相关代码。

3.3.1 数据爬取及处理

在 Python 环境下执行命令,生成数据库表。

```
python  manage.py makemigrations
python  manage.py migrate --run-syncdb
```

安装所需第三方库。

```
python -m pip install -r requirement.txt
```

数据来源为爬取 Movielens 上相关电影数据、简介及评分。

```
python populate_ratings.py
python populate_movie.py
python populate_movie_description.py
python populate_ratings.py
```

相关代码如下:

```
#导入需要的库
import os
import urllib.request
import django
import datetime
import decimal
from tqdm import tqdm
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'Recs.settings')
django.setup()
from Analytics.models import Rating
#插入评分记录
defcreate_rating(user_id,content_id,rating,timestamp):ating = Rating(user_id = user_id,movie_
id = content_id, rating = decimal.Decimal(rating), rating_timestamp = datetime.datetime.
```

```
fromtimestamp(float(timestamp)))
        rating.save()
        return rating
    #爬取评分记录
    def download_ratings():
        URL = 'https://raw.githubusercontent.com/sidooms/MovieTweetings/master/latest/
ratings.dat'
        response = urllib.request.urlopen(URL)
        data = response.read()
        print('download finished')
        return data.decode('utf-8')
    #删除已有的评分数据
    def delete_db():
        print('truncate db')
        Rating.objects.all().delete()
        print('finished truncate db')
    def populate():
        delete_db()
        ratings = download_ratings()
        for rating in tqdm(ratings.split(sep="\n")):
            r = rating.split(sep="::")
            if len(r) == 4:
                create_rating(r[0], r[1], r[2], r[3])
    if __name__ == '__main__':
        print("Starting MovieRecs Population script...")
        populate()
    #python populate_movie.py 相关代码
    #导入需要的库
    import os
    import urllib.request
    from tqdm import tqdm
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'Recs.settings')
    import django
    django.setup()
    from Movies.models import Movie, Genre
    #创建每一条电影数据并保存
    def create_movie(movie_id, title, genres):
        movie = Movie.objects.get_or_create(movie_id=movie_id)[0]
        title_and_year = title.split(sep="(")
        movie.title = title_and_year[0]
        movie.year = title_and_year[1][:-1]
        if genres:
            for genre in genres.split(sep="|"):
                g = Genre.objects.get_or_create(name=genre)[0]
                movie.genres.add(g)
                g.save()
        movie.save()
```

```
        return movie
    #爬取电影数据
    def download_movies(URL = 'https://raw.githubusercontent.com/sidooms/MovieTweetings/
master/latest/movies.dat'):  #下载电影数据
        response = urllib.request.urlopen(URL)
        data = response.read()
        return data.decode('utf-8')
    #如果之前存在数据则先删除
    def delete_db():
        print('truncate db')
        movie_count = Movie.objects.all().count()
        if movie_count > 1:
            Movie.objects.all().delete()
            Genre.objects.all().delete()
        print('finished truncate db')
    def populate():
        movies = download_movies()
        if len(movies) == 0:
            print('The latest dataset seems to be empty. Older movie list downloaded.')
             print('Please have a look at https://github.com/sidooms/MovieTweetings/issues
and see if there is an issue')
            movies = download_movies(
    'https://raw.githubusercontent.com/sidooms/MovieTweetings/master/snapshots/100K/movies.dat')
        print('movie data downloaded')
        for movie in tqdm(movies.split(sep='\n')):
            m = movie.split(sep="::")
            if len(m) == 3:
                create_movie(m[0], m[1], m[2])
    if __name__ == '__main__':
        print("Starting MovieGeeks Population script...")
        delete_db()
        populate()
    #python populate_movie_description.py 相关代码
    #导入所需的库
    import os
    import django
    import json
    import pandas as pd
    import requests
    from tqdm import tqdm
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'Recs.settings')
    django.setup()
    from Recommender.models import MovieDecriptions
    #获取电影数据并保存
    from Analytics.models import Rating
    def get_descriptions_with_movieid(movie_id):
        url = "https://api.themoviedb.org/3/find/tt{}?external_source=imdb_id&api_key={}"
```

```
    api_key = get_api_key()
    format_url = url.format(movie_id, api_key)
    r = requests.get(format_url)
    for film in r.json()['movie_results']:
        md = MovieDecriptions.objects.get_or_create(movie_id = movie_id)[0]
        #保存电影名
        md.imdb_id = movie_id
        if 'title' in film:
            md.title = film['title']
        #保存电影简介
        if 'overview' in film:
            md.description = film['overview']
            #保存电影类型
        if 'genre_ids' in film:
            md.genres = film['genre_ids']
        if len(md.description) > 0:
            md.save()
            # print("{}: {}".format(movie_id, r.json()))
#如果之前存在数据则先删除
def delete_db():
    print('truncate db')
    MovieDecriptions.objects.all().delete()
    print('finished truncate db')
def get_api_key():              #获取 API 的键
    cred = json.loads(open(".prs").read())
    return cred['themoviedb_apikey']
#加载评分数据
def load_all_ratings():
    #提取相关列的数据
    columns = ['movie_id']
    ratings_data = Rating.objects.all().values(*columns)
    movie_ids = pd.DataFrame.from_records(ratings_data, columns = columns)
    movie_ids = movie_ids.drop_duplicates(subset = None, keep = 'first', inplace = False)
    movie_ids = movie_ids.reset_index()
    return movie_ids
if __name__ == '__main__':     #主函数
    print("Starting MovieRecs Population script...")
    delete_db()
    movie_ids = load_all_ratings()
    movie_ids = movie_ids.iloc[:, 1]
    for movie_id in tqdm(movie_ids.values):
        get_descriptions_with_movieid(movie_id)
```

爬取的数据保存在 pgAdmin4 中，如图 3-3 所示。

Python 命令行爬取数据成功，如图 3-4 所示。

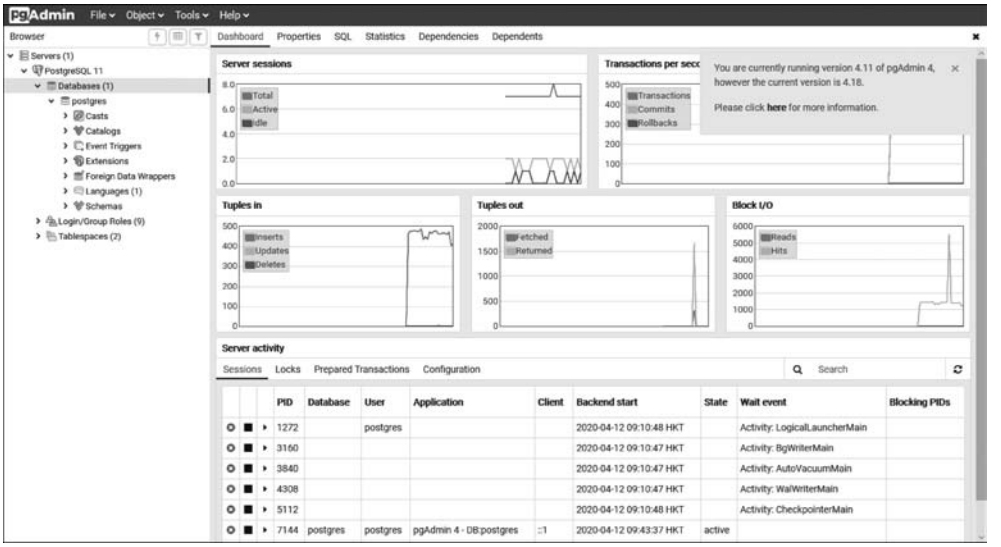

图 3-3　爬取数据展示

```
Exception ignored in: <function tqdm.__del__ at 0x000002078852E438>
Traceback (most recent call last):
  File "C:\Users\miaomiao\Anaconda3\lib\site-packages\tqdm\_tqdm.py", line 931, in __del__
    self.close()
  File "C:\Users\miaomiao\Anaconda3\lib\site-packages\tqdm\_tqdm.py", line 1133, in close
    self._decr_instances(self)
  File "C:\Users\miaomiao\Anaconda3\lib\site-packages\tqdm\_tqdm.py", line 496, in _decr_instances
    cls.monitor.exit()
  File "C:\Users\miaomiao\Anaconda3\lib\site-packages\tqdm\_monitor.py", line 52, in exit
    self.join()
  File "C:\Users\miaomiao\Anaconda3\lib\threading.py", line 1041, in join
```

图 3-4　爬取数据成功

3.3.2　模型训练及保存

输入命令构建模型并训练：

```
python -m Builder.item_similarity_calculator
python -m Builder.matrix_factorization_calculator
python -m Builder.lda_model_calculator
```

以上 3 行命令分别代表基于协同过滤、矩阵分解、LDA 主题模型。

1. 协同过滤

相关代码如下：

```
#导入相关的库
import os
from tqdm import tqdm
```

```
from datetime import datetime
import pandas as pd
import psycopg2
from scipy.sparse import coo_matrix, csr_matrix
import numpy as np
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "Recs.settings")
import django
django.setup()
#获取电影评分
from Analytics.models import Rating
from Recs import settings
class ItemSimilarityMatrixBuilder(object):
def __init__(self, min_overlap=15, min_sim=0.2):
#同时对项目1和项目2有过评分的最小用户数
self.min_overlap = min_overlap
#最小相似度
self.min_sim = min_sim
self.db = settings.DATABASES['default']['ENGINE']
#ratings: 评分数据。save: 是否保存到数据库,默认保存
def build(self, ratings, save=True):
print("calculating similarities ... using {} ratings".format(len(ratings)))
start_time = datetime.now()
print("Creating ratings matrix")
ratings['rating'] = ratings['rating'].astype(float)
#计算每个user_id的平均评分,并做归一化处理
ratings['avg'] = ratings.groupby('user_id')['rating'].transform(lambda x: normalize(x))
#把user_id、movie_id转为pandas的类别,以便去重
ratings['avg'] = ratings['avg'].astype(float)
ratings['user_id'] = ratings['user_id'].astype('category')
ratings['movie_id'] = ratings['movie_id'].astype('category')
#构建稀疏评分矩阵,没有评分的数据全部用0填充
coo = coo_matrix((ratings['avg'].astype(float),
                  (ratings['movie_id'].cat.codes.copy(),
                  ratings['user_id'].cat.codes.copy())))
#计算两个项目间的重叠个数,同时统计项目1和项目2有过评分的用户数
print("Calculating overlaps between the items")
overlap_matrix = coo.astype(bool).astype(int).dot(coo.transpose().astype(bool).astype(int))
#重叠部分大于min_overlap的项目数量
number_of_overlaps = (overlap_matrix > self.min_overlap).count_nonzero()
print("Overlap matrix leaves {} out of {} with {}".format(number_of_overlaps, overlap_matrix.
count_nonzero(),self.min_overlap))
print("Rating matrix (size {}x{}) finished, in {} seconds".format
(coo.shape[0],coo.shape[1],datetime.now() - start_time)) sparsity_level = 1 - (ratings.
shape[0] / (coo.shape[0] * coo.shape[1]))
print("Sparsity level is {}".format(sparsity_level))
start_time = datetime.now()
#初始化一个为0的相似度矩阵
```

```
print("Calculating similarity between the items")
cor = self.calculating_similarity(coo)
#cor = cosine_similarity(coo, dense_output = False)
#print(type(cor))
# print(cor)
#对相似度大于最小相似度的元素,进行对应位置相乘
cor = cor.multiply(cor > self.min_sim)
#对相似度大于最小重叠度的元素,进行对应位置相乘
cor = cor.multiply(overlap_matrix > self.min_overlap)
print(cor)
movies = dict(enumerate(ratings['movie_id'].cat.categories))
print('Correlation is finished, done in {} seconds'.format(datetime.now() - start_time))
if save:
start_time = datetime.now()
print('save starting')
if self.db == 'django.db.backends.postgresql':
self.save_similarity(cor, movies)
print('save finished, done in {} seconds'.format(datetime.now() - start_time))
return cor, movies
#计算相似度优化算法,从 sklearn 得到启发
def calculating_similarity(self, coo):
#稀疏矩阵转 Numpy 数组
data_array = coo.toarray()
data_array = check_array(data_array)
#爱因斯坦求和约定,即对两个矩阵按元素位置对应相乘,按行求和
#[[1 2 3]  [[1 2 3]     [[1 4 9]  [14, 14]
#[1 2 3]]   [1 2 3]]    [1 4 9]]
norms = np.einsum('ij,ij->i', data_array, data_array)
np.sqrt(norms, norms)
norms[norms == 0.0] = 1.0
data_array /= norms[:, np.newaxis]
#运算之后把 numpy 库的多维数组或矩阵转为 SciPy 库的稀疏矩阵进行计算,否则汇报内存溢出
array_sparse = csr_matrix(data_array)
sim_matrix = array_sparse @ array_sparse.transpose()
return sim_matrix
def save_similarity(self, sim_matrix, index, created = datetime.now()):
#设置开始时间
start_time = datetime.now(
print('truncating table in {} seconds'.format(datetime.now() - start_time))
sims = []
no_saved = 0
start_time = datetime.now()
print('instantiation of coo_matrix in {} seconds'.format(datetime.now() - start_time))
#计算相似度矩阵
coo = coo_matrix(sim_matrix)
csr = coo.tocsr()
query = "insert into similarity (created,source,target,similarity) values %s;
```

```
conn = self.get_connect()
cur = conn.cursor()
cur.execute('truncate table similarity')
print('{} similarities to save'.format(coo.count_nonzero()))
#初始化相似度矩阵
xs, ys = coo.nonzero()
for x, y in tqdm(zip(xs, ys), leave = True):
if x == y:
continue
sim = csr[x, y]
#寻找相似度最高的用户
if sim < self.min_sim:
Continue
if (len(sims)) == 500000:
psycopg2.extras.execute_values(cur, query, sims)
sims = []
print("{} saved in {}".format(no_saved, datetime.now() - start_time))
#创建相似度矩阵
new_similarity = (str(created), index[x], index[y], sim)
no_saved += 1
sims.append(new_similarity)
psycopg2.extras.execute_values(cur, query, sims, template = None, page_size = 1000)
conn.commit()
print('{} Similarity items saved, done in {} seconds'.format(no_saved, datetime.now() - start_
time))
@staticmethod
#获取用户名和密码
def get_connect():
ifsettings.DATABASES['default']['ENGINE'] == 'django.db.backends.postgresql':
dbUsername = settings.DATABASES['default']['USER']
dbPassword = settings.DATABASES['default']['PASSWORD']
dbName = settings.DATABASES['default']['NAME']
#用户名和密码校验
conn_str = "dbname = {} user = {} password = {}".format(dbName, dbUsername, dbPassword)
conn = psycopg2.connect(conn_str)
return conn
#检查数据类型
def check_array(array, dtype = "numeric", order = None):
array_orig = array
dtype_numeric = isinstance(dtype, str) and dtype == "numeric"
dtype_orig = getattr(array, "dtype", None)
if dtype_numeric:
if dtype_orig is not None and dtype_orig.kind == "O":
#如果输入为一个对象,则转换为浮点型
dtype = np.float64
else:
dtype = None
```

```
if np.may_share_memory(array, array_orig):
array = np.array(array, dtype = dtype, order = order)
return array
#归一化
def normalize(x):
x = x.astype(float)
#计算数值的和
x_sum = x.sum()
#计算大于 0 的元素
x_num = x.astype(bool).sum()
#计算均值
x_mean = 0
if x_num > 0:
x_mean = x_sum / x_num
if x_num == 1 or x.std() == 0:
return 0.0
return (x - x_mean) / (x.max() - x.min())
#加载评分数据
def load_all_ratings(min_ratings = 1):
#提取相关列的数据
columns = ['user_id', 'movie_id', 'rating', 'type']
ratings_data = Rating.objects.filter(user_id__range = (0, 30000)).values( * columns)
ratings = pd.DataFrame.from_records(ratings_data, columns = columns)
#通过用户 ID 分类,统计每个用户 ID 评分过的项目数量
user_count = ratings[['user_id', 'movie_id']].groupby('user_id').count()
user_count = user_count.reset_index()
#取出评分项目数量超过 min_ratings 的所有用户 ID
user_ids = user_count[user_count['movie_id'] > min_ratings]['user_id']
#取出用户 ID 的评分数据记录
ratings = ratings[ratings['user_id'].isin(user_ids)]
#将评分数据转换成浮点类型
ratings['rating'] = ratings['rating'].astype(float)
return ratings
def main():
print("Calculation of item similarity")
all_ratings = load_all_ratings()
ItemSimilarityMatrixBuilder().build(all_ratings)
if __name__ == '__main__':
main()
```

2. 矩阵分解

相关代码如下：

```
#导入需要的包
import numpy as np
import pandas as pd
```

```
import os
import psycopg2
from tqdm import tqdm
from datetime import datetime
from scipy.sparse import coo_matrix
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "Recs.settings")
import django
django.setup()
#导入评分数据
from Analytics.models import Rating
from Recs import settings
class MatrixFactorization(object):
#创建评分矩阵
def __init__(self, min_sim=0.1):
self.min_sim = min_sim
self.db = settings.DATABASES['default']['ENGINE']
def train(self, c_ui, factors = 50, regularization = 0.01, iterations=15):
print("calculating Matrix ... using {} ratings".format(len(c_ui)))
start_time = datetime.now()
print("Creating ratings matrix")
c_ui['rating'] = (c_ui['rating'] - c_ui['rating'].min()) / (c_ui['rating'].max() - c_ui['rating'].
min())
c_ui['rating'] = c_ui['rating'].astype(float)
#计算每个 user_id 的平均评分,并做归一化处理
#c_ui['avg'] = c_ui.groupby('user_id')['rating'].transform(lambda x: normalize(x))
#把 user_id、movie_id 转为 pandas 的类别,以便去重
#c_ui['avg'] = c_ui['avg'].astype(float)
  c_ui['user_id'] = c_ui['user_id'].astype('category')
  c_ui['movie_id'] = c_ui['movie_id'].astype('category')
  #构建稀疏评分矩阵,没有评分的数据全部用 0 填充
  coo = coo_matrix((c_ui['rating'].astype(float),
                    (c_ui['movie_id'].cat.codes.copy(),
                     c_ui['user_id'].cat.codes.copy())))
users, items = coo.shape
print("Ratings matrix finished,in{} seconds".format(datetime.now() - start_time))
start_time = datetime.now()
print("Calculating ALS....")
#随机初始化两个隐语义矩阵 X 和 Y
X = np.random.rand(users, factors) * 0.01
Y = np.random.rand(items, factors) * 0.01
cui, ciu = coo.tocsr(), coo.T.tocsr()
 for iteration in range(iterations):
 self.least_squares_cg(cui=cui,X=X,Y=Y, regularization=regularization,)
 self.least_squares_cg(cui=ciu,X=Y,Y=X, regularization=regularization,)
 print("Rating matrix (size {}x{}) finished, in {} seconds".format(coo.shape[0],
 coo.shape[1], datetime.now() - start_time))
 #用户的相似度计算
```

```
 sim = np.dot(X, Y.T)
 movies_ = dict(enumerate(c_ui['movie_id'].cat.categories))
 users_ = dict(enumerate(c_ui['user_id'].cat.categories))
 self.save_similarity(sim_matrix = sim, movies = movies_, users = users_)
 #print(sim)
 #self.rmse(coo, sim)
 return X, Y
 #ALS 算法/共轭梯度法
 #创建三元组
 def least_squares_cg(self, cui, X, Y, regularization, cg_steps = 3):
#用户因子
users, factors = X.shape
YtY = Y.T.dot(Y) + regularization * np.eye(factors)
for u in range(users):
#基于用户历史
x = X[u]
#计算残差 r = (YtCuPu - (YtCuY.dot(Xu),不计算 YtCuY
r = -YtY.dot(x)
for i, confidence in self.nonzeros(cui, u):
r += (confidence - (confidence - 1) * Y[i].dot(x)) * Y[i]
p = r.copy()
rsold = r.dot(r)
 for it in range(cg_steps):
 #计算 Ap = YtCuYp - 并非实际计算 YtCuY
Ap = YtY.dot(p)
for i, confidence in self.nonzeros(cui, u):
Ap += (confidence - 1) * Y[i].dot(p) * Y[i]
#更新 CG 标准
            alpha = rsold / p.dot(Ap)
            x += alpha * p
            r -= alpha * Ap
            rsnew = r.dot(r)
            p = r + (rsnew / rsold) * p
            rsold = rsnew
        X[u] = x
#返回 CSR 矩阵非零元素的索引和值
def nonzeros(self, m, row):
    """ returns the non zeroes of a row in csr_matrix """
    for index in range(m.indptr[row], m.indptr[row + 1]):
        yield m.indices[index], m.data[index]
def rmse(self, coo, sim):
    #取出评分大于 0 的数据
    start_time = datetime.now()
    print('instantiation of coo_matrix in {} seconds'.format(datetime.now() - start_time))
    csr = coo.tocsr()
    print('Calculating rmse....')
    #计算最小均方误差
```

```
    mse = 0.0
    xs, ys = coo.nonzero()
    number = len(coo.data)
    for x, y in tqdm(zip(xs, ys), leave = True):
        y_r = csr[x, y]
        if y_r > 0:
            y_hat = sim[x][y]
            square_error = (y_r - y_hat) ** 2
            mse += square_error
    print('RMSE {}'.format((mse / number) ** 0.5))
@staticmethod
#用户连接登录
def get_connect():
if settings.DATABASES['default']['ENGINE'] == 'django.db.backends.postgresql':
#获取用户名和密码
        dbUsername = settings.DATABASES['default']['USER']
        dbPassword = settings.DATABASES['default']['PASSWORD']
        dbName = settings.DATABASES['default']['NAME']
#用户名和密码校验
        conn_str = "dbname = {} user = {} password = {}".format(dbName,
                                                    dbUsername,
                                                    dbPassword)
        conn = psycopg2.connect(conn_str)
    return conn
#用户相似度的计算和保存
def save_similarity(self, sim_matrix, movies, users, created = datetime.now()):
    start_time = datetime.now()
print('truncating table in {} seconds'.format(datetime.now() - start_time))
    sims = []
    no_saved = 0
    start_time = datetime.now()
print('instantiation of coo_matrix in {} seconds'.format(datetime.now() - start_time))
    query = "insert into similarity_mf (created, user_id, movie_id, similarity) values %s;"
    conn = self.get_connect()
    cur = conn.cursor()
    cur.execute('truncate table similarity_mf')
    print('{} similarities to save'.format(len(sim_matrix)))
    #用户相似度匹配
    row, column = sim_matrix.shape
    for i in tqdm(range(row)):
        for j in range(column):
            sim = sim_matrix[i][j]
            if sim < self.min_sim:
continue
if (len(sims)) == 500000:
psycopg2.extras.execute_values(cur, query, sims)
sims = []
```

```
print("{} saved in {}".format(no_saved,datetime.now() - start_time))
#用户评分相似度矩阵创建
new_similarity = (str(created), users[j], movies[i], sim)
no_saved += 1
sims.append(new_similarity)
psycopg2.extras.execute_values(cur, query, sims, template = None, page_size = 1000)
conn.commit()
print('{} Similarity items saved, done in {} seconds'.format(no_saved, datetime.now() - start_
time))
#获取评分数据
def load_all_ratings(min_ratings = 1):
columns = ['user_id','movie_id','rating', 'type', 'rating_timestamp']
ratings_data = Rating.objects.all().values( * columns)
ratings = pd.DataFrame.from_records(ratings_data, columns = columns)
user_count = ratings[['user_id', 'movie_id']].groupby('user_id').count()
user_count = user_count.reset_index()
user_ids = user_count[user_count['movie_id']> min_ratings]
['user_id']
#获取评分高的相应用户名
ratings = ratings[ratings['user_id'].isin(user_ids)]
ratings['rating'] = ratings['rating'].astype(float)
return ratings
if __name__ == '__main__':
all_ratings = load_all_ratings()
model = MatrixFactorization(min_sim = 0.1)
X, Y = model.train(c_ui = all_ratings, factors = 50, regularization = 0.01, iterations = 1)
```

3. LDA 主题模型

相关代码如下：

```
#导入需要的包
import os
from tqdm import tqdm
import psycopg2
from datetime import datetime
from scipy.sparse import coo_matrix
os.environ.setdefault("DJANGO_SETTINGS_MODULE", "Recs.settings")
import django
from Recs import settings
import numpy as np
django.setup()
from nltk.tokenize import RegexpTokenizer
from stop_words import get_stop_words
from gensim import corpora, models, similarities
from  Recommender.models import MovieDecriptions, LdaSimilarity
#建立主题模型
```

```
class LdaModel(object):
    def __init__(self, min_sim = 0.1):
        self.min_sim = min_sim;
        self.db = settings.DATABASES['default']['ENGINE']
   def train(self, data = None, docs = None):
   #数据准备
        if data is None:
            data, docs = load_data()
        NUM_TOPICS = 10
        self.build_lda_model(data, docs, NUM_TOPICS)
    def build_lda_model(self, data, docs, n_topics = 5):
        texts = []
         #英文分词
        tokenizer = RegexpTokenizer(r'\w + ')
        for d in tqdm(data):
        raw = d.lower()
        tokens = tokenizer.tokenize(raw)
         #去除停用词
            stop_tokens = self.remove_stopwords(tokens)
            stemmed_tokens = stop_tokens
            texts.append(stemmed_tokens)
         #构建词典
        dictionary = corpora.Dictionary(texts)
         #生成语料库
        corpus = [dictionary.doc2bow(text) for text in texts]
lda_model = models.ldamodel.LdaModel(corpus = corpus, id2word = dictionary, num_topics = n_
topics)
        index = similarities.MatrixSimilarity(corpus)
       self.save_similarities_with_postgresql(index, docs)
        return dictionary, texts, lda_model
    @staticmethod
    def remove_stopwords(tokenized_data):
     #去除停用词
        en_stop = get_stop_words('en')
        stop_tokens = [token for token in tokenized_data if token not in en_stop]
        return stop_tokens
        #保留相似度
        def save_similarities_with_postgresql(self, index, docs, created = datetime.now()):
        start_time = datetime.now()
        print(f'truncating table in {datetime.now() - start_time} seconds')
        sims = []
        no_saved = 0
        start_time = datetime.now()
        #创建稀疏矩阵
        coo = coo_matrix(index)
        csr = coo.tocsr()
        print(f'instantiation of coo_matrix in {datetime.now() - start_time} seconds')
```

```
            query = "insert into lda_similarity (created, source, target, similarity) values %s;"
            conn = self.get_conn()
            cur = conn.cursor()
            #cur.execute('drop table lda_similarity')
            #cur.execute('ALTER TABLE lda_similarity ADD COLUMN similarity decimal(8, 7) NOT NULL')
            cur.execute('truncate table lda_similarity')
            print(f'{coo.count_nonzero()} similarities to save')
            #相似度对比
            xs, ys = coo.nonzero()
            for x, y in zip(xs, ys):
                if x == y:
                    continue
                sim = float(csr[x, y])
                x_id = str(docs[x].movie_id)
                y_id = str(docs[y].movie_id)
                     #取出评分 sim 数量超过 min_sim 的所有 sim
                if sim < self.min_sim:
                    continue
                if len(sims) == 100000:
                    psycopg2.extras.execute_values(cur, query, sims)
                    sims = []
                    print(f"{no_saved} saved in {datetime.now() - start_time}")
                new_similarity = (str(created), x_id, y_id, sim)
                no_saved += 1
                sims.append(new_similarity)
            psycopg2.extras.execute_values(cur, query, sims, template = None, page_size = 1000)
            conn.commit()
            print('{} Similarity items saved, done in {} seconds'.format(no_saved, datetime.now() -
start_time))
        #获取用户名和密码
        @staticmethod
        def get_conn():
            dbUsername = settings.DATABASES['default']['USER']
            dbPassword = settings.DATABASES['default']['PASSWORD']
            dbName = settings.DATABASES['default']['NAME']
             #用户名和密码校验
            conn_str = "dbname = {} user = {} password = {}".format(dbName,
                                                    dbUsername,
                                                    dbPassword)
            conn = psycopg2.connect(conn_str)
            return conn
#获取电影数据
def load_data():
    docs = list(MovieDecriptions.objects.all())
    data = ["{}, {}, {}".format(d.title, d.genres, d.description) for d in docs]
    if len(data) == 0:
        print("No descriptions were found, run populate_sample_of_descriptions")
```

```
    return data, docs
if __name__ == '__main__':
    print("Calculating lda model...")
    data, docs = load_data()
    lda = LdaModel()
    lda.train(data, docs)
```

3.3.3　接口实现

在定义模型架构和训练保存后，电影推荐系统接口实现如下。

1. 流行电影推荐

相关代码如下：

```
#导入需要的包
from decimal import Decimal
from Collector.models import Log
from django.db.models import Count
from django.db.models import Q
from django.db.models import Avg
from Recsmodel.baseModel import baseModel
#流行度推荐
class Popularity(baseModel):
    def predict_score(self, user_id, item_id):
        return None
    def recommend_items(self, user_id, num=6):
        return None
    @staticmethod
    #推荐6部流行度最高的电影
    def recommend_items_from_log(num=6):
        items = Log.objects.values('content_id')
        items = items.filter(event='like').annotate(Count("user_id"))
        sorted_items = sorted(items, key=lambda item: -float(item['user_id__count']))
        return sorted_items[:num]
```

2. 相邻用户推荐

相关代码如下：

```
#导入需要的包
from Recsmodel.baseModel import baseModel
from Analytics.models import Rating
from django.db.models import Q
import time
from decimal import Decimal
from Recommender.models import Similarity
class NeighborhoodRecs(baseModel):
```

```
    def __init__(self, neighborhood_size = 10, min_sim = 0.1):
        #最近邻个数,最小相似度,最大候选集个数
        self.neighborhood_size = neighborhood_size
        self.min_sim = min_sim
        self.max_candidates = 100
    def recommend_items(self, user_id, num = 6):
        #取出用户做出过的评分信息
        active_user_items = Rating.objects.filter(user_id = user_id).order_by('-rating')
[0: self.max_candidates]
        #print(user_id, active_user_items.values())
        return self.recommend_item_by_ratings(active_user_items.values(), num)
     #推荐
    def recommend_item_by_ratings(self, active_user_items, num = 6):
        #如果没有评过分的则返回空
        if len(active_user_items) == 0:
            return {}
        #标记时间
        start = time.time()
        movie_ids = {movie['movie_id']: movie['rating'] for movie in active_user_items}
        #用户平均评分
        user_mean = sum(movie_ids.values()) / len(movie_ids)
        candidate_items = Similarity.objects.filter(Q(source__in = movie_ids.keys())& ~
Q(target__in = movie_ids.keys())& Q(similarity__gt = self.min_sim))
        #print(candidate_items)
        candidate_items = candidate_items.order_by('-similarity')[:self.max_candidates]
        recs = dict()
        for candidate in candidate_items:
            target = candidate.target
            pre = 0
            sim_sum = 0
              rated_items = [i for i in candidate_items if i.target == target][:self.
neighborhood_size]
            #print(rated_items)
            if len(rated_items) > 0:
                for sim_item in rated_items:
                    r = Decimal(movie_ids[sim_item.source] - user_mean)
                    pre += sim_item.similarity * r
                    sim_sum += sim_item.similarity
      #取出相似度最高的所有项目
      if sim_sum > 0:
     recs[target] = {'prediction': Decimal(user_mean) + pre / sim_sum,
                     'sim_items': [r.source for r in rated_items]}
     #对筛选出来的项目进行分类
     sorted_items = sorted(recs.items(), key = lambda item: -float(item[1]['prediction']))
[:num]
        return sorted_items
    #评分预测
```

```
def predict_score(self, user_id, item_id):
user_items = Rating.objects.filter(user_id = user_id)
user_items = user_items.exclude(movie_id = item_id).order_by('-rating')[:100]
        movie_ids = {movie.movie_id: movie.rating for movie in user_items}
        return self.predict_score_by_ratings(item_id, movie_ids)
    def predict_score_by_ratings(self, item_id, movie_ids):
        top = Decimal(0.0)
        bottom = Decimal(0.0)
        ids = movie_ids.keys()
        mc = self.max_candidates
#候选电影名单
candidate_items = (Similarity.objects.filter(source__in = ids)
                                        .exclude(source = item_id)
                                        .filter(target = item_id))
candidate_items = candidate_items.distinct().order_by('-similarity')[:mc]
        if len(candidate_items) == 0:
            return 0
        for sim_item in candidate_items:
            r = movie_ids[sim_item.source]
            top += sim_item.similarity * r
            bottom += sim_item.similarity
        return Decimal(top/bottom)
```

3. 相似内容推荐

相关代码如下：

```
#导入需要的包
from decimal import Decimal
from django.db.models import Q
from Analytics.models import Rating
from Recommender.models import MovieDecriptions,LdaSimilarity
from Recsmodel.baseModel import baseModel
#建立基本推荐模型
class ContentBasedRecs(baseModel):
    def __init__(self, min_sim = 0.1):
        self.min_sim = min_sim
        self.max_candidates = 100
    #基于用户内容的协同过滤
    def recommend_items(self, user_id, num = 6):
active_user_items = Rating.objects.filter(user_id = user_id).order_by('-rating')[:100]
return self.recommend_items_by_ratings(user_id, active_user_items.values(), num)
    def recommend_items_by_ratings(self,user_id,active_user_items,num = 6):
        if len(active_user_items) == 0:
            return {}
        movie_ids = {movie['movie_id']: movie['rating'] for movie in active_user_items}
        user_mean = sum(movie_ids.values()) / len(movie_ids)
```

```
        #计算用户内容的相似度
        sims = LdaSimilarity.objects.filter(Q(source__in = movie_ids.keys())
                                    &~Q(target__in = movie_ids.keys())
                                    &Q(similarity__gt = self.min_sim))
        print(active_user_items)
        sims = sims.order_by('-similarity')[:self.max_candidates]
        recs = dict()
        targets = set(s.target for s in sims if not s.target == '')
        for target in targets:
            pre = 0
            sim_sum = 0
            rated_items = [i for i in sims if i.target == target]
            if len(rated_items) > 0:
                for sim_item in rated_items:
                    r = Decimal(movie_ids[sim_item.source] - user_mean)
                    pre += sim_item.similarity * r
                    sim_sum += sim_item.similarity
                    if sim_sum > 0:
        recs[target] = {'prediction': Decimal(user_mean) + pre / sim_sum,
                                'sim_items': [r.source for r in rated_items]}
        return sorted(recs.items(), key = lambda item: -float(item[1]['prediction']))[:num]
    def predict_score(self, user_id, item_id):
        return None
```

3.3.4 收集数据

电影推荐系统需要收集用户行为,完成相应预测和推荐。

```
unction add_log(user_id, event_type, content_id, session_id, csrf_token) {
    $.ajax({
        type:'POST',
        url: '/collect/log/',
   #收集用户数据
        data: {
            "csrfmiddlewaretoken": csrf_token,
            "event_type": event_type,
            "user_id": user_id,
            "content_id": content_id,
            "session_id": session_id
        },
        fail: function () {
            console.log('log failed(' + event_type + ')')
        }
    })
}
```

3.3.5　界面设计

对网页显示的方式、大小、格式、布局及每个组件的颜色、位置进行设计，不同页面对应不同的功能。

在 views 文件中定义视图函数，当浏览器向服务器发送 http 请求时，这些函数被调用，在 views 中导入数据库，创建 HTML 模板，将电影推荐列表呈现给用户。在应用包中创建 templates 和 index.html 文件，html 文件中代码用于测试。相关代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% load static %}
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>MovieRes</title>
    <!-- Bootstrap -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/css/bootstrap.min.
css" rel="stylesheet">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/css/bootstrap-theme.
min.css" rel="stylesheet">
     <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
    <!--[if lt IE 9]>
      <script src="https://cdn.bootcss.com/html5shiv/3.7.3/html5shiv.js"></script>
      <script src="https://cdn.bootcss.com/respond.js/1.4.2/respond.min.js"></script>
    <![endif]-->
    <script src="https://cdn.bootcss.com/html5shiv/3.7.3/html5shiv.js"></script>
    <script src="https://cdn.bootcss.com/respond.js/1.4.2/respond.min.js"></script>
    <script src="https://cdn.bootcss.com/jquery/1.12.2/jquery.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@3.3.7/dist/js/bootstrap.min.js">
</script>
    <script src="{% static 'js/collector.js' %}"></script>
    <script>
        function get_url(movieid) {
            return 'https://api.themoviedb.org/3/find/tt' + movieid + '?external_source=
imdb_id&api_key={{ api_key }}'
        }
    </script>
    <style type="text/css">
      .bg-navbar {
        background: rgb(0,127,246);
          width: 100%;
    }
      .navbar-nav>li>a:hover{
```

```
        text-decoration: underline;
        background: none;
    }
    .container-fluid{
        margin-top: 50px;
        padding-top: 12px;
        padding-left: 100px;
        padding-right: 100px;
    }
    .nav-sidebar{
        background-color: white;
    }
    .well{
        background: white;
        border: none;
    }
    .form-control{
        height: 30px;
        border: none;
    }
    .input-group-addon{
        height: 30px;
        border: none;
    }
    .btn-primary{
        border: none;
        background: rgb(0,127,246);
    }
    .line-clamp{
        overflow: hidden;
        font-size: 14px;
        height: 40px;
    }
    .line-clamp-rating{
        overflow: hidden;
        text-overflow: ellipsis;
        font-size: 8px;
        height: 30px;
    }
    .pagination-bottom{
        text-align: center;
    }
    .right-content {
        overflow: hidden;
    }
    [class* ="col-"] {
        margin-bottom: -99999px;
```

```
            padding - bottom: 99999px;
        }
    </style>
    {% block head %}{% endblock %}
</head>
<body>
    <div class="container-fluid">
        <nav class="navbar navbar-transparent navbar-expand-xl bg-navbar navbar-fixed-
top">
            <div class="navbar-header">
                <button type="button" class="navbar-toggle"
                        data-toggle="collapse" data-target=".navbar-collapse">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                </button>
                <div class="navbar-left">
                    <ul class="nav navbar-nav">
                        <li><a class="navbar-brand" style="color: white" href="/">
MovieRes</a></li>
                        <li><a style="color: white" href="/analytics/user/{{user_
id}}/">User: {{ user_id }}</a></li>
                    </ul>
                </div>
            </div>
            <!-- Search -->
            <div class="nav nav-pills pull-right">
                <form class="navbar-form"  action="/movies/search/">
                    <div class="input-group">
                        <input type="search" name="q" class="form-control" placeholder=
"Search" style="background-color:white;" maxlength="40" />
                        <span class="input-group-btn">
                            <button class="input-group-addon" style="width: 40px;
background: rgb(242,242,242)">
                                <span class="glyphicon glyphicon-search"></span>
                            </button>
                        </span>
                    </div>
                </form>
            </div>
        </nav>
        <!-- end of top -->
        <div class="row row-fluid">
            {% block content %}{% endblock content %}
        </div>
        </div>
    </div>
    <script>
```

```
    </script>
</body>
</html>
```

3.4 系统测试

整体效果如图 3-5 所示。

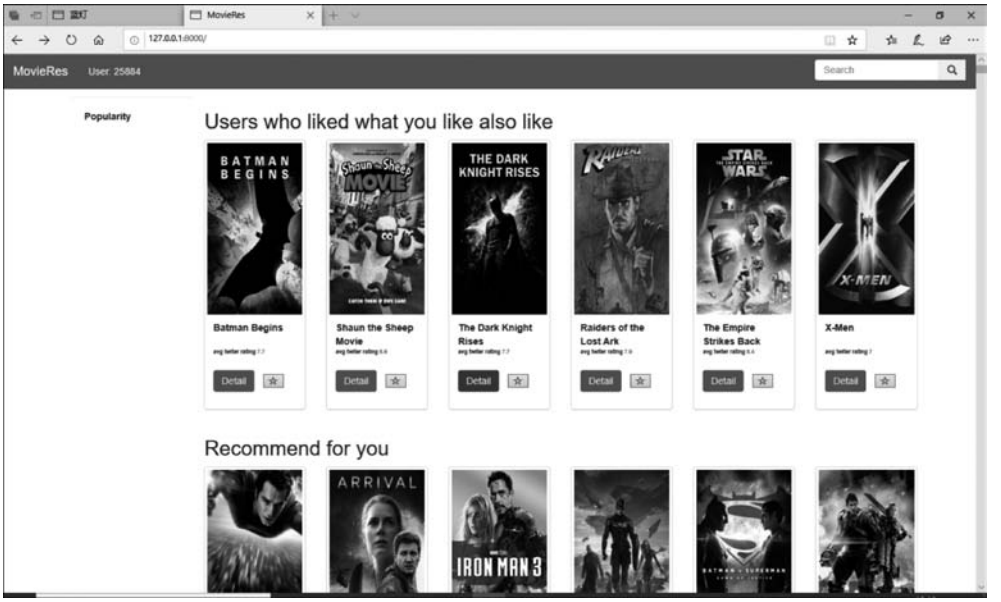

图 3-5 整体效果

网站电影推荐分 3 部分展示,如图 3-6～图 3-8 所示。网站电影详情页如图 3-9 所示。

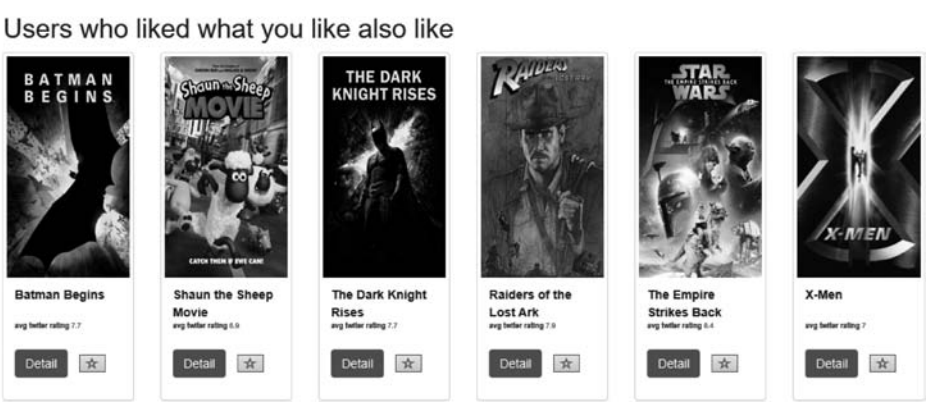

图 3-6 网站第 1 部分效果

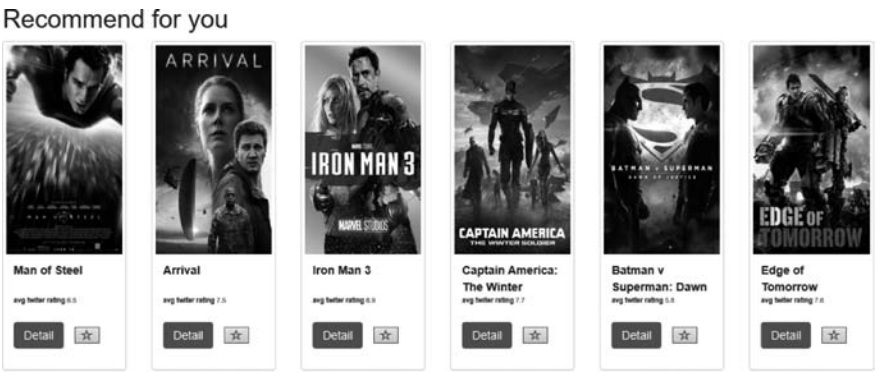

图 3-7　网站第 2 部分效果

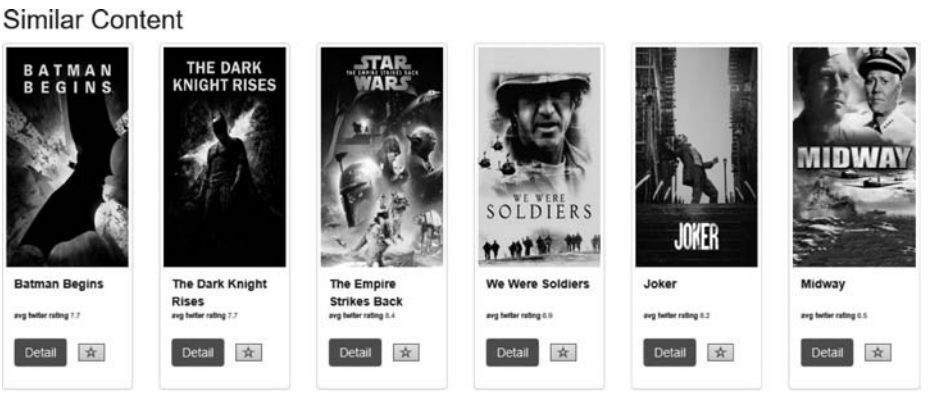

图 3-8　网站第 3 部分效果

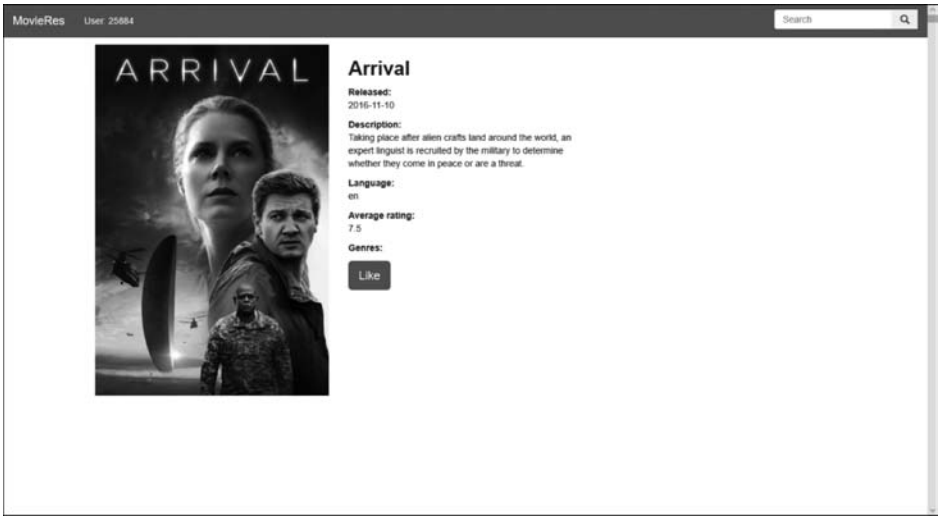

图 3-9　网站电影详情页展示