

# 电子商务网站用户分析及推荐

小组成员：韩林洁

夏铭泽

相 铮

李晓波

# 电子商务网站用户行为分析及服务推荐

**摘要：**随着网络信息的迅速发展，电子商务等越来越普及，用户在海量信息中找到自己想要的信息会变得越来越困难，因此本文通过用户的历史行为进行分析，采用基于物品的协同算法计算物品的相似性，从而生成推荐表。

## 一、背景

随着互联网和信息技术的快速发展，电子商务、网上服务与交易等网络业务的越来越普及，大量的信息聚集起来，形成海量信息。用户想要从海量信息中快速准确地寻找到自己感兴趣的信息已经变得越来越困难，在电子商务领域这点显得更加突出。因此信息过载问题已经成为互联网技术中的一个重要难题。如今传统的搜索引擎已经无法满足用户的需求，与搜索引擎不同，推荐系统并不需要用户提供明确的需求，而是通过分析用户的历史行为，从而主动向用户推荐能够满足他们兴趣和需求的信息，在电子商务领域中推荐技术可以起到一下作用：

（1）帮助用户发现其感兴趣的物品，节省用户时间，提升用户体验。

（2）提高用户对电子商务网站的忠诚度，如果推荐系统能够准确地发现用户的兴趣点，并将合适的资源推荐给用户，用户就会对该电子商务网站产生依赖，从而建立稳定的企业忠实客户群。

## 二、问题描述

### 2.1. 数据准备

本次我们主要的研究对象是北京某家法律网站,它是一家电子商务类的大型法律咨询网站,致力于为用户提供丰富的法律信息与专业的咨询服务,并为律师与律师事务所提供卓有成效的互联网整合营销解决方案。 数据文件: 7law.zip

### 2.2. 模型建立

为实现较好的推荐效果,我们采用结合多种推荐方法将推荐结果进行组合,最后得出推荐结果。我们采用基于物品的协同过滤推荐系统对用户进行个性化推荐,以其推荐结果作为推荐系统的重要部分。基于物品的协同过滤算法主要分为两步:

- 1、计算物品之间相似度;

- 2、根据物品的相似度和用户的历史行为给用户生成推荐列表。关于物品相似度计算方法我们采用杰拉德(Jaccard)相似系数法。完成各个物品之间的相似度的计算之后,即可构成一个物品之间的相似度矩阵,通过采用相似度矩阵,推荐算法会给用户推荐与其物品最相似的K个物品。

## 三、数据探索

### 3.1 数据获取

本系统是某律师网站选取用户3个月的用户访问数据(2015-2-1—2015-

4-29)作为原始数据集,每个地区用户访问习惯和爱好存在差异性,本系统抽取广州的访问数据进行分析,共 837450 条数据,包括用户号、访问时间、来源网站、访问节目、页面标题、来源网页、标签、网友类别、关键词等属性。采用 MariaDB 开源数据库,读取 7law.sql 文件,因为数据比较大需要进行分块处理。

数据库访问代码:

```
[python]
1. import pandas as pd
2. from sqlalchemy import create_engine
3.
4. # 先自定义函数将表格写入数据库里,以备操作过程中有些数据要写入数据库
5. def savetosql(DF,tablename):
6.     import pandas as pd
7.     from sqlalchemy import create_engine
8.     yconnect = create_engine('mysql:mysql://root:@127.0.0.1:3306/jing?charset=utf8')
9.     pd.io.sql.to_sql(DF,tablename, yconnect, schema='jing', if_exists='append')
```

```
[python]
1. # python 访问数据库
2. engine = create_engine('mysql+pymysql://root:@127.0.0.1:3306/jing?charset=utf8')
3. sql = pd.read_sql('all_gzdata', engine, chunksize = 10000)
4. # '''
5. # 由于本人电脑用的主机Host名称为: 127.0.0.1, 端口3306, 使用的数据库名称为test,字符集为utf8, 用户名为root,密码为空,所以配置如上所示
6. # 用create_engine建立连接,连接地址的意思依次为“数据库格式(mysql)+程序名(pymysql)+账号密码@地址端口/数据库名(test)”,最后指定编码为utf8;
7. # all_gzdata是表名,engine是连接数据的引擎,chunksize指定每次读取1万条记录。这时候sql是一个容器,未真正读取数据。
```

## 3.2 网页类型分析

### 3.2.1 统计各个网页类型所占的比例

```
[python]
1. counts1 = [ i['fullURLId'].value_counts() for i in sql] #逐块统计
2. counts1
```

```
Out[2]: [101003      4762
          1999001    2389
          107001     2143
          301001      177
          101002      166
          102002      163
          101001       51
          106001       37
          103003       26
          101009       18
          103003       18]
```

### 3.2.2 统计知识类型（107 类别）内部的点击情况：

```
[python]
1. # 因为只有107001一类，但是可以继续细分成三类：知识内容页、知识列表页、知识首页
2. def count107(i): #自定义统计函数
3.     j = i[['fullURL']][i['fullURLId'].str.contains('107')].copy() #找出类别包含107的网址
4.     j['type'] = None # 添加空列
5.     j['type'][j['fullURL'].str.contains('info/.+?/')]= u'知识首页'
6.     j['type'][j['fullURL'].str.contains('info/.+?/.+?')]= u'知识列表页'
7.     j['type'][j['fullURL'].str.contains('/\d+?_*\d+?\.\html')]= u'知识内容页'
8.     return j['type'].value_counts()
9. # 注意：获取一次sql对象就需要重新访问一下数据库(!!!)
10. engine = create_engine('mysql+pymysql://root:@127.0.0.1:3306/jing?charset=utf8')
11. sql = pd.read_sql('all_gzdata', engine, chunksize = 10000)
12.
13. counts2 = [count107(i) for i in sql] # 逐块统计
14. counts2 = pd.concat(counts2).groupby(level=0).sum() # 合并统计结果
15. counts2
```

```
Out[11]: 知识内容页      164243
          知识列表页      9656
          知识首页       9001
          Name: type, dtype: int64
```

### 3.2.3 统计带"?"问号网址类型统计

```
[python]    
1. # 注意获取一次sql对象就需要重新访问一下数据库  
2.  
3. def countquestion(i): #自定义统计函数  
4.     j = i[['fullURLId']][i['fullURL'].str.contains('?')].copy() #找出类别包含107的网址  
5.     return j  
6.  
7. engine = create_engine('mysql+pymysql://root:@127.0.0.1:3306/jing?charset=utf8')  
8. sql = pd.read_sql('all_gzdata', engine, chunksize = 10000)  
9.  
10. counts3 = [countquestion(i)['fullURLId'].value_counts() for i in sql]  
11. counts3 = pd.concat(counts3).groupby(level=0).sum()  
12. counts3
```

```
Out[13]: 101003      47  
         102002      25  
         107001     346  
        1999001    64718  
         301001     356  
         Name: fullURLId, dtype: int64
```

## 3.3 点击次数分析

目标: 点击次数分析: 统计分析原始数据用户浏览网页次数(以“真实IP”区分)的情况

### 3.3.1 统计点击次数

```
[python]    
1. counts1 = [i['realIP'].value_counts() for i in sql] # 分块统计各个IP的出现次数  
2. counts1 = pd.concat(counts1).groupby(level=0).sum() # 合并统计结果, level=0表示按照index分组
```

```
Out[2]: 82033      2  
        95502      1  
        103182      1  
        116010      2  
        136206      1  
        140151      1  
        155761      1  
        158601      1
```

```
[python]
1. counts1_ = DataFrame(counts1)
2. counts1_[1]=1 # 添加1列全为1
3. a = counts1_.groupby('realIP').sum()#统计各个“不同点击次数”分别出现的次数# 也可以使用counts1_['realIP'].value_counts()功能
4. a.columns=[u'用户数']
5. a.index.name = u'点击次数'
6. a[u'用户百分比'] = a[u'用户数']/a[u'用户数'].sum()*100
7. a[u'记录百分比'] = a[u'用户数']/a.index/counts1_['realIP'].sum()*100
8. a.sort_index(inplace = True)
9. b = a.iloc[:,7:]
10. c = b.T
11. c
```

Out[3]:

点击次数	1	2	3	4	5	6	7
用户数	132119.000000	44175.000000	17573.000000	10156.000000	5952.000000	4132.000000	2632.000000
用户百分比	57.405854	19.194087	7.635488	4.412793	2.586151	1.795359	1.143607
记录百分比	15.776345	10.549884	6.295182	4.850916	3.553645	2.960416	2.200012

### 3.3.2 对浏览一次的用户行为进行分析

```
[python]
1. # 获取浏览一次的所有数据
2. f = counts1_[counts1_['realIP']==1]
3. del f[1]
4. f.columns = [u'点击次数']
5. f.index.name = 'realIP'
6. # g = [pd.merge(f,i[['fullURLId','fullURL','realIP']],right_on = 'realIP',left_index=True,how = 'left') for i in sql]
7. g = [i[['fullURLId','fullURL','realIP']] for i in sql]
8. g = pd.concat(g)
9. h = pd.merge(f,g,right_on = 'realIP',left_index=True,how = 'left')
10. h
```

Out[24]:

	点击次数	fullURLId	fullURL	realIP
5978	1	101003	http://www.lawtime.cn/ask/question_7882607.html	95502
0	1	101003	http://www.lawtime.cn/ask/question_7174864.html	103182
1083	1	101003	http://www.lawtime.cn/ask/question_8246285.html	136206
6727	1	107001	http://www.lawtime.cn/info/gongsi/slbfgs/2011-	140151
1804	1	101003	http://www.lawtime.cn/ask/question_5951952.html	155761

## 3.4 网页排名分析

目标：网页排名分析 获得各个网页点击率排名以及类型点击率排名：统计分析原始数据用户浏览网页次数（以“真实 IP”区分）

### 3.4.1 获取网页点击排名

```
[python]
1. engine = create_engine('mysql+pymysql://root:@127.0.0.1:3306/jing?charset=utf8')
2. sql = pd.read_sql('all_gzdata', engine, chunksize = 10000)
3.
4. def clickfreq(i): #自定义统计函数
5.     j = i[['fullURL', 'fullURLId', 'realIP']][i['fullURL'].str.contains('\.html')]
6.     return j
7.
8. counts1 = [clickfreq(i) for i in sql] # 分块统计各个I的出现次数
9. counts1 = pd.concat(counts1)
10.
11. counts1_ = counts1['fullURL'].value_counts()
12. counts1_ = DataFrame(counts1_)
13.
14. counts1_.columns = [u'点击次数']
15. counts1_.index.name = u'网址'
16. a = counts1_.sort_values(u'点击次数', ascending=False).iloc[:20,:]
17. a
```

网址	点击次数
http://www.lawtime.cn/faguizt/23.html	6503
http://www.lawtime.cn/info/hunyin/lhlawlhxy/20110707137693.html	4938
http://www.lawtime.cn/faguizt/9.html	4562
http://www.lawtime.cn/info/shuifa/slb/2012111978933.html	4495
http://www.lawtime.cn/faguizt/11.html	3976
http://www.lawtime.cn/info/hunvin/lhlawlhxy/20110707137693_2.html	3305

### 3.4.2 获取网页点击排名数筛选出点击次数>50 的有 html 结尾的网址

```
[python]
1. b = counts1_.reset_index()
2. c = b[b[u'点击次数']>50][b[u'网址'].str.contains('\d+?\d+?.html')]
3. c.set_index(u'网址', inplace=True)
4. c.sort_index(inplace = True)
5. # savetosql(c, 'count355')# 并保存到数据库中
6. c
```

网址	点击次数
http://www.lawtime.cn/ask/exp/10573.html	61
http://www.lawtime.cn/ask/exp/10668.html	128
http://www.lawtime.cn/ask/exp/11563.html	58
http://www.lawtime.cn/ask/exp/12201.html	61
http://www.lawtime.cn/ask/exp/12947.html	60



## 四、数据清洗

在这个数据库中，中间页面的网址、咨询发布成功页面、律师登录助手等页面与分析数据无关，从而进行数据清洗。

代码如下：

```
#数据清洗
for i in sql:
    d = i[['realIP', 'fullURL', 'pageTitle', 'userID', 'timestamp_format']].copy() # 只要网址列
    d['fullURL'] = d['fullURL'].str.replace('\?.*', '') # 网址中间号后面的部分
    d = d[(d['fullURL'].str.contains('\.html')) & (d['fullURL'].str.contains('lawtime')) & (d['fullURL'].str.contains('midques_') == False)] #
    # 保存到数据库中
    d.to_sql('cleaned_one', engine, index = False, if_exists = 'append')

for i in sql:
    d = i[['realIP', 'fullURL', 'pageTitle', 'userID', 'timestamp_format']].copy() # 只要网址列
    d['pageTitle'] = d['pageTitle'].fillna('空', inplace=True)
    d = d[(d['pageTitle'].str.contains('快车-律师助手') == False) & (d['pageTitle'].str.contains('咨询发布成功') == False) & \
          (d['pageTitle'].str.contains('免费发布法律咨询') == False) & (d['pageTitle'].str.contains('法律快搜') == False)]
    d = d.copy()
    # 保存到数据库中
    d.to_sql('cleaned_two', engine, index = False, if_exists = 'append')
```

## 五、数据变换

由于是推荐系统对输入数据的需要，可以对处理后的数据进行属性规约，提取所需要的属性，本数据需要的数据属性是用户和用户访问的网页

```
#属性归约
for i in sql:
    zixun = i[['userID', 'fullURL']][i['fullURL'].str.contains('(ask)|(askzt)')].copy()
    l1 = len(zixun) + 11
    hunyin = i[['userID', 'fullURL']][i['fullURL'].str.contains('hunyin')].copy()
    l2 = len(hunyin) + 12
    zixun.to_sql('zixunformodel', engine, index=False, if_exists = 'append')
    hunyin.to_sql('hunyinformodel', engine, index=False, if_exists = 'append')
```

## 六、模型选取

本系统为推荐程序，存在长尾网页丰富，用户个性化需求强烈、推荐结果的实时变化，原始数据中网页数明显小于用户数，故而采用基于物品的协同过滤推

荐系统对用户进行个性化推荐，以推荐系统结果作为系统结果的重要部分。它的主要过程是：分析用户和物品的数据集，通过对项目浏览喜好找到相似的物品，然后根据用户历史喜好，推荐相似的项目给目标用户。

这就需要计算物品直接的相似度。这个本系统采用 jaccard 相似系统法计算，这是因为用户行为是 0-1 型。根据相似度和用户历史行为为用户生成推荐列表。在计算之间的相似度后会产生相似度矩阵利用  $P = SIM * R$  度量，其中 P 为用户的感兴趣程度，R 为用户对物品的兴趣（0-1），SIM 表示所有物品之间的相似度。评测模型的方法采取随机打乱数据的方法：随机函数打乱原始数据，然后将用户行为数据集按照均匀反驳分成 M 份，1 份为测试集，M-1 份为训练集，在训练集上建立模型，在测试集进行评测。原因：随机可以保证系统的随机性得到稳定可靠的结果。

## 七、实验结果

本实验选取的个性化推荐算法为基于物品的协同过滤算法，算法步骤及结果如下：

## 7.1 协同推荐算法定义协同推荐函数：



```
[python] 1. def Jaccard(a,b): #自定义杰卡德相似系数函数，仅对0-1矩阵有效
2.     return 1.0*(a*b).sum()/(a+b-a*b).sum()
3.
4. class Recommender():
5.     sim = None # 相似度矩阵
6.     def similarity(self, x, distance): # 计算相似度矩阵的函数
7.         y = np.ones((len(x), len(x)))
8.         for i in range(len(x)):
9.             for j in range(len(x)):
10.                 y[i,j] = distance(x[i], x[j])
11.         return y
12.
13.     def fit(self, x, distance = Jaccard): # 训练函数
14.         self.sim = self.similarity(x, distance)
15.         return self.sim
16.
17.     def recommend(self, a): # 推荐函数
18.         return np.dot(self.sim, a) * (1-a)
```



```
[python] 1. len(data['fullURL'].value_counts()) # 4339
2. len(data['realIP'].value_counts()) # 10333
3. # 由题意知 网址数 < 用户数--> 建立基于物品的协同过滤推荐
```

## 7.2 将所有数据转成 0-1 矩阵：



```
[python] 1. # 大概运行时间是40秒左右
2. import time
3. start0 = time.clock()
4. data.sort_values(by=['realIP','fullURL'],ascending=[True,True],inplace=True)
5. realIP = data['realIP'].value_counts().index
6. realIP = np.sort(realIP)
7. fullURL = data['fullURL'].value_counts().index #
8. fullURL = np.sort(fullURL)
9. D = DataFrame([], index = realIP, columns = fullURL )
10.
11. for i in range(len(data)):
12.     a = data.iloc[i,0] # 用户名
13.     b = data.iloc[i,1] # 网址
14.     D.loc[a,b] = 1
15. D.fillna(0,inplace = True)
16. end0 = time.clock()
17. usetime0 = end0-start0
18. print '转成0、1矩阵所花费的时间为'+ str(usetime0) +'s! '#34.5123141125s!
19. # D.shape=10333 rows x 4339 columns
20. #保存的表命名格式为“3_1_k比表功能名称”，是本小节生成的第1张表格，功能为zero_one: 整个数据集计算得到的0-1矩阵
21. D.to_csv('3_1_1zero_one.csv')
```



## 7.3 交叉验证方法验证推荐:

```
[python]    
1. # 步骤: 解决采用一次验证的方法, 再创建十折交叉验证循环 (此处只采用了一次具体十折交叉方法见3_2_10-fold cross-validation.py)  
2. # 由于是基于物品(网址)的推荐, 所以测试集需包含所有网址(全集), 选择0.9*总用户数用户记录来进行训练模型  
3. # 注意: 将数据随机打乱  
4.  
5. # 随机打乱数据  
6. # 注意 每次打乱数据, 下面的都会改变  
7. df = D.copy()  
8.  
9. simplifier = np.random.permutation(len(df))  
10. df = df.take(simplifier)# 打乱数据  
11.  
12. train = df.iloc[:int(len(df)*0.9), :]  
13. test = df.iloc[int(len(df)*0.9):, :]  
14.  
15. df = df.as_matrix()  
16.  
17. df_train = df[:int(len(df)*0.9), : ]# 前90%为训练集len(df_train) = 9299  
18. df_test = df[int(len(df)*0.9):, : ]# 后10%为测试集len(df_test) = 103
```


```
[python]    
1. #由于基于物品的推荐, 对于矩阵, 根据上面的推荐函数, index为网址, 因此需要进行转置  
2. df_train = df_train.T  
3. df_test = df_test.T  
4.  
5. print df_train.shape # (4339L, 9299L)  
6. print df_test.shape # (4339L, 1034L)
```

## 7.4 建立相似矩阵, 训练模型:

```
[python]    
1. print df_train.shape # (4339L, 9299L)  
2. import time  
3. start1 = time.clock()  
4. r = Recommender()  
5. sim = r.fit(df_train)# 计算物品的相似度矩阵  
6. end1 = time.clock()  
7.  
8. a = DataFrame(sim) # 保存相似度矩阵  
9. usetime1 = end1-start1  
10. print u'建立相似矩阵耗时'+str(usetime1)+'s!' #1981.60760257s!  
11.  
12. print a.shape # (4339L, 9299L)  
13.  
14. # 将所有数据保存  
15. a.index = train.columns  
16. a.columns = train.columns  
17.  
18. #保存的表命名格式为"3_1_k此表功能名称", 是本小节生成的第25张表格, 功能为similarityMatrix: 计算训练集的相似度矩阵  
19. a.to_csv('3_1_2similarityMatrix.csv')  
20. a.head(20)
```

[python]  

```
1. print df_train.shape # (4339L, 9299L)
2. import time
3. start1 = time.clock()
4. r = Recommender()
5. sim = r.fit(df_train)# 计算物品的相似度矩阵
6. end1 = time.clock()
7.
8. a = DataFrame(sim) # 保存相似度矩阵
9. usetime1 = end1-start1
10. print u'建立相似矩阵耗时'+str(usetime1)+'s!' #1981.60760257s!
11.
12. print a.shape # (4339L, 9299L)
13.
14. # 将所有数据保存
15. a.index = train.columns
16. a.columns = train.columns
17.
18. #保存的表命名格式为“3_1_k此表功能名称”，是本小节生成的第25张表格，功能为similarityMatrix: 计算训练集的相似度矩阵
19. a.to_csv('3_1_2similarityMatrix.csv')
20. a.head(20)
```

[python]  

```
1. # 使用测试集进行预测
2. print df_test.shape # (4339L, 1034L)
3. start2 = time.clock()
4. result = r.recommend(df_test)
5. end2 = time.clock()
6.
7. result1 = DataFrame(result)
8. usetime2 = end2-start2
9. print u'推荐函数耗时'+str(usetime2)+'s!' # 推荐函数耗时2.60267174653s!
10.
11. result1
```

[python]  

```
1. # 将推荐结果表格中的对应的网址和用户名对应上
2. result1.index = test.columns
3. result1.columns = test.index
4. #保存的表命名格式为“3_1_k此表功能名称”，是本小节生成的第25张表格，功能为recommedresult: 显示推荐的结果
5. result1.to_csv('3_1_3recommedresult.csv')
6. result1
```

# 7.5 协同推荐结果展现

```
1. # 定义展现具体协同推荐结果的函数，K为推荐的个数，recomMatrix为协同过滤算法算出的推荐矩阵的表格化
2. # type(K):int, type(recomMatrix):DataFrame
3.
4. def xietong_result(K, recomMatrix ):
5.     recomMatrix.fillna(0.0,inplace=True)# 将表格中的空值用0填充
6.     n = range(1,K+1)
7.     recommends = ['xietong'+str(y) for y in n]
8.     currentemp = DataFrame([],index = recomMatrix.columns, columns = recommends)
9.     for i in range(len(recomMatrix.columns)):
10.         temp = recomMatrix.sort_values(by = recomMatrix.columns[i], ascending = False)
11.         k = 0
12.         while k < K:
13.             currentemp.iloc[i,k] = temp.index[k]
14.             if temp.iloc[k,i] == 0.0:
15.                 currentemp.iloc[i,k:K] = np.nan
16.                 break
17.             k = k+1
18.
19.     return currentemp
20.
21. start3 = time.clock()
22. xietong_result = xietong_result(3, result1)
23. end3 = time.clock()
24. print '按照协同过滤推荐方法为用户推荐3个未浏览过的网址耗时为' + str(end3 - start3)+'s!' #29.4996622053s!
25.
26. #保存的表命名格式为“3_1_k此表功能名称”，是为本小节生成的第4张表格，功能为xietong_result: 显示协同过滤推荐的结果
27. xietong_result.to_csv('3_1_4xietong_result.csv')
28.
29. xietong_result # 结果中出现了全空的行，这是冷启动现象
```

	xietong1	xietong2	xietong3
3931892151	http://www.lawtime.cn/info/hunyun/jhsy/ertaizh...	http://www.lawtime.cn/info/hunyun/jiechutongji...	http://www.lawtime.cn/info/hunyun/jiehun/xieyi...
647913072	http://www.lawtime.cn/info/hunyun/jichengfagui...	http://www.lawtime.cn/info/hunyun/fangqi/20110...	NaN
2352137592	http://www.lawtime.cn/info/hunyun/jiehun/hunji...	http://www.lawtime.cn/info/hunyun/jiehun/hunji...	http://www.lawtime.cn/info/hunyun/jiehundengji...
4224096014	http://www.lawtime.cn/info/hunyun/lihunshouxu/...	http://www.lawtime.cn/info/hunyun/lhlavlhxy/20...	http://www.lawtime.cn/info/hunyun/lhlavlhxy/20...
1303280398	http://www.lawtime.cn/info/hunyun/lhlavlhxy/20...	NaN	NaN
377341560	http://www.lawtime.cn/info/hunyun/lhlavlhxy/20...	http://www.lawtime.cn/info/hunyun/lhlavlhxy/20...	http://www.lawtime.cn/info/hunyun/lhlavlhxy/20...

## 参考文献

[1] Python 数据分析与挖掘实战

[2] 数据挖掘概念与技术：第三版