利用社交网络数据

如何利用社交网络数据给用户进行个性化推荐，不仅讨论如何利用社交网络给用户推荐物品，而且将讨论如何利用社交网络给用户推荐好友。

获取社交网络数据的途径

电子邮件

用户注册信息

用户的位置数据

论坛和讨论组

即时聊天工具

社交网站

一种是通过人们之间的共同兴趣和信念形成的兴趣图谱；另一种社会群体则是由于人们之间的亲属关系，工作关系而形成的社会图谱。

社交网络数据简介

社交网络定义了用户之间的联系，因此可以用图定义社交网络。我们用图$G(V,E,w)$定义一个社交网络，其中 V 是顶点集合，每个顶点代表一个用户，E 是边集合，如果用户v_a和v_b有社交网络关系，那么就有一条边$e(v_a,v_b)$连接这两个用户，而$w(v_a,v_b)$定义了边的权重。

- **双向确认的社交网络数据**　这种社交网络一般可以通过无向图表示；
- **单向关注的社交网络数据**　这种社交网络中的用户关系是单向的，可以通过有向图表示；
- **基于社区的社交网络数据**　还有一种社交网络数据，用户之间并没有明确的关系，但是这种数据包含了用户属于不同社区的数据。

社交网络数据中的长尾分布

基于社交网络的推荐

- **好友推荐可以增加推荐的信任度**　好友往往是用户最信任的。用户往往不一定信任计算机的智能，但会信任好朋友的推荐。
- **社交网络可以解决冷启动问题**　可以在没有用户行为记录时就给用户提供较高质量的推荐结果，部分解决了推荐系统的冷启动问题。

最主要的就是很多时候并不一定能提高推荐算法的离线精度（准确率和召回率）。特别是在基于社交图谱数据的推荐系统中，因为用户的好友关系不是基于共同兴趣产生的，所以用户好友的兴趣往往和用户的兴趣并不一致。

基于邻域的社会化推荐算法

如果给定一个社交网络和一份用户行为数据集。其中社交网络定义了用户之间的好友关系，而用户行为数据集定义了不同用户的历史行为和兴趣数据。那么我们想到的最简单算法是给用户推荐好友喜欢的物品集合。即用户 u 对物品 i 的兴趣p_{ui}可以通过如下公式计算。

$$p_{ui} = \sum_{v \in out(u)} w_{uv} r_{vi}$$

熟悉度可以用用户之间的共同好友比例来度量。也就是说如果用户 u 和用户 v 很熟悉，那么一般来说他们应该有很多共同的好友。

$$familiarity(u,v) = \frac{|out(u) \cap out(v)|}{|out(u) \cup out(v)|}$$

除了熟悉程度，还需要考虑用户之间的兴趣相似度。在度量用户相似度时还需要考虑兴趣相似度（similarity），而兴趣相似度可以通过和 UserCF 类似的方法度量，即如果两个用户喜欢的物品集合重合度很高，两个用户的兴趣相似度很高：

$$similiarity(u,v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$$

```
def Recommend(uid, familiarity, similarity, train):
   rank = dict()
   interacted_items = train[uid]
   for fid,fw in familiarity[uid]:
      for item,pw in train[fid]:
         # if user has already know the item
         # do not recommend it
         if item in interacted_items:
            continue
         addToDict(rank, item, fw * pw)
   for vid,sw in similarity[uid]:
      for item,pw in train[vid]:
         if item in interacted_items:
            continue
         addToDict(rank, item, sw * pw)
   return rank
```

基于图的社会化推荐算法

在社交网站中存在两种关系，一种是用户对物品的兴趣关系，一种是用户之间的社交网络关系。

在定义完图中的顶点和边后，需要定义边的权重。其中用户和用户之间边的权重可以定义为用户之间相似度的α倍（包括熟悉程度和兴趣相似度），而用户和物品之间的权重可以定义为用户对物品喜欢程度的β倍，α和β需要根据应用的需求确定。如果我们希望用户好友的行为对推荐结果产生比较大的影响，那么就可以选择比较大的α。相反，如果我们希望用户的历史行为对推荐结果产生比较大的影响，就可以选择比较大的β。

一种社交网络关系称为 friendship，另一种社交网络关系称为 membership。如果要在前面提到的基于邻域的社会化推荐算法中考虑 membership 的社交关系，可以利用两个用户加入的社区重合度计算用户相似度，然后给用户推荐和他相似的用户喜欢的物品。但是，如果利用图模型，我们就很容易同时对 friendship 和 membership 建模。

实际系统中的社会化推荐算法

基于邻域的社会化推荐算法看起来非常简单，但在实际系统中却是很难操作的，这主要是因为该算法需要拿到用户所有好友的历史行为数据，而这一操作在实际系统中是比较重的操作。所以，如果一个算法再给一个用户做推荐时，需要他所有好友的历史行为数据，这在实际环境中是比较困难的。

- **两处截断**：第一处截断就是在拿用户好友集合时并不拿出用户所有的好友，而是只拿出和用户相似度最高的 N 个好友。此外，在查询每个用户的历史行为时，可以只返回用户最近 1 个月的行为，这样就可以在用户行为缓存中缓存更多用户的历史行为数据，从而加快查询用户历史行为接口的速度。此外，还可以牺牲一定的实时性，降低缓存中用户行为列表过期的频率。
- **重新设计数据库**：首先，为每个用户维护一个消息队列，用于存储他的推荐列表；当一个用户喜欢一个物品时，就将（物品 ID、用户 ID 和时间）这条记录写入关注该用户的推荐列表消息队列中；当用户访问推荐系统时，读出他的推荐列表消息队列，对于这个消息队列中的每个物品，重新计算该物品的权重。计算权重时需要考虑物品在队列中出现的次数，物品对应的用户和当前用户的熟悉程度、物品的时间戳。同时，计算出每个物品被哪些好友喜欢过，用这些好友作为物品的推荐解释。

社会化推荐系统和协同过滤推荐系统

社会化推荐系统的效果往往很难通过离线实验评测，因为社会化推荐的优势不在于增加预测准确度，而是在于通过用户的好友增加用户对推荐结果的信任度，从而让用户单击那些很冷门的推荐结果。此外，很多社交网站（特别是基于社交图谱的社交网站）中具有好友关系的用户并不一定有相似的兴趣。因此，利用好友关系有时并不能增加离线评测的准确率和召回率。因此，很多研究人员利用用户调查和在线实验的方式评测社会化推荐系统。

信息流推荐

信息流推荐算法 EdgeRank，该算法综合考虑了信息流中每个会话的时间、长度与用户兴趣的相似度。将其他用户对当前用户信息流中的会话产生过行为的行为称为 edge，而一条会话的权重定义为：

$$\sum_{\text{edges e}} u_e w_e d_e$$

如果一个会话被你熟悉的好友最近产生过重要的行为，它就会有比较高的权重。不过，EdgeRank 算法的个性化因素仅仅是好友的熟悉度，它并没有考虑帖子内容和用户兴趣的相似度。所以 EdgeRank 仅仅考虑了“我”周围用户的社会化兴趣，而没有重视“我”个人的个性化兴趣。

- **会话的长度**　越长的会话包括越多的信息。
- **话题相关性**　度量了会话中主要话题和用户兴趣之间的相关性。
- **用户熟悉程度**　主要度量了会话中涉及的用户（比如会话的创建者、讨论者等）和当前用户的熟悉程度。计算熟悉度的主要思想是考虑用户之间的共同好友数等。

给用户推荐好友

好友关系是社会化网站的重要组成部分，如果用户的好友很稀少，就不能体验到社会化的

好处。因此好友推荐是社会化网站的重要应用之一。好友推荐系统的目的是根据用户现有的好友、用户的行为记录给用户推荐新的好友，从而增加整个社交网络的稠密程度和社交网站用户的活跃度。

基于内容的匹配

利用内容信息计算用户的相似度和前面讨论的利用内容信息计算物品的相似度类似。

基于共同兴趣的好友推荐

用户往往不关心对于一个人是否在现实社会中认识，而只关心是否和他们有共同的兴趣爱好。也可以根据用户在社交网络中的发言提取用户的兴趣标签，来计算用户的兴趣相似度。

基于社交网络图的好友推荐

在社交网站中，我们会获得用户之间现有的社交网络图，然后可以基于现有的社交网络给用户推荐新的好友。基于好友的好友推荐算法可以用来给用户推荐他们在现实社会中互相熟悉，而在当前社交网络中没有联系的其他用户。

- 对于用户 u 和用户 v，我们可以用共同好友比例计算他们的相似度：

$$w_{out}(u,v) = \frac{|out(u) \cap out(v)|}{\sqrt{|out(u)||out(v)|}}$$

```
def FriendSuggestion(user, G, GT):
   suggestions = dict()
   friends = G[user]
   for fid in G[user]:
      for ffid in GT[fid]:
         if ffid in friends:
            continue
         if ffid not in suggestions:
            suggestions[ffid] = 0
         suggestions[ffid] += 1
   suggestions = {x: y / math.sqrt(len(G[user]) * len(G[x])
                     for x,y in suggestions}
```

- 有向图中，引入

$$w_{in}(u,v) = \frac{|in(u) \cap in(v)|}{\sqrt{|in(u)||in(v)|}}$$

```
def FriendSuggestion(user, G, GT):
   suggestions = dict()
   for fid in GT[user]:
      for ffid in G[fid]:
         if ffid in friends:
            continue
         if ffid not in suggestions:
            suggestions[ffid] = 0
         suggestions[ffid] += 1
```

```
    suggestions = {x: y / math.sqrt(len(GT[user]) * len(GT[x])
                        for x,y in suggestions}
```

- 用户 u 关注的用户中，有多大比例也关注了用户 v。但是，这个相似度有一个缺点，就是在该相似度的定义下所有人都和名人有很大的相似度。这是因为这个相似度在分母的部分没有考虑|in(v)|的大小。

$$w_{out,in}(u,v) = \frac{|out(u) \cap in(v)|}{|out(u)|}, w'_{out,in}(u,v) = \frac{|out(u) \cap in(v)|}{\sqrt{|out(u)||in(v)|}}$$

```
def FriendSuggestion(user, G, GT):
    suggestions = dict()
    for fid in GT[user]:
        for ffid in G[fid]:
            if ffid in friends:
                continue
            if ffid not in suggestions:
                suggestions[ffid] = 0
            suggestions[ffid] += 1
    suggestions = {x: y / math.sqrt(len(GT[user]) * len(GT[x])
                        for x,y in suggestions}
```

在实际系统中我们需要在自己的数据集上对比不同的算法，找到最适合自己数据集的好友推荐算法。

基于用户调查的好友推荐算法对比

- **InterestBased** 给用户推荐和他兴趣相似的其他用户作为好友。
- **SocialBased** 基于社交网络给用户推荐他好友的好友作为好友。
- **Interest+Social** 将 InterestBased 算法推荐的好友和 SocialBased 算法推荐的好友按照一定权重融合。
- **SONA** SONA 是 IBM 内部的推荐算法，该算法利用大量用户信息建立了 IBM 员工之间的社交网络。