1. 主要问题描述

互联网规模和覆盖面的迅速增长带来了信息超载（information overload）问题：过量信息同时呈现使得用户无法从中获取对自己有用的部分，信息使用效率反而降低。现有的很多网络应用，如门户网站、搜索引擎和专业数据索引本质上都是帮助用户过滤信息的手段。然而这些工具只满足主流需求，没有个性化的考虑，仍然无法很好地解决信息超载的问题。

对于搜索引擎来说，用户在搜索互联网中的信息时，需要在搜索引擎中输入“查询关键词”，搜索引擎根据用户的输入，在系统后台进行信息匹配，将与用户查询相关的信息展示给用户。但是，若用户无法想到准确描述自己需求的关键词，搜索引擎就变得无能为力。

推荐系统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来对用户的兴趣进行建模，从而主动给用户推荐可能满足他们兴趣和需求的信息。搜索引擎和推荐系统对用户来说是两个互补的工具。用户通过搜索引擎搜索自己感兴趣的东西（知道自己要什么），推荐系统又反过来利用搜索记录把用户可能感兴趣的东西推荐给用户（不知道自己要什么）。

比如亚马逊，就很好的利用了推荐系统，它利用所有用户在站点上的行为，根据不同数据的特点对他们进行处理，并分成不同区为用户推送推荐：基于社会化的推荐，amazon 会给出事实的数据，让用户信服，例如，购买此商品的用户百分之多少也购买了那个商品。基于物品本身的推荐，amazon 也会列出推荐的理由例如：因为你的购物车中有 A 物品，或者你买过 A 物品，所以给你推荐类似的 B 物品。

豆瓣也会给用户推荐电影，基于用户看过的电影。微博也会给用户推荐可能感兴趣的博主，QQ 也会推荐可能感兴趣的好友，比如同群而且有多个共同好友但是没有好友关系的两个用户。

所以推荐系统的应用十分广泛，是社交网络时代非常重要的一项技术。

2. 方法描述

推荐系统的概念：“它是利用电子商务网站向客户提供商品信息和建议，帮助用

户决定应该购买什么产品，模拟销售人员帮助客户完成购买过程”。推荐有 3 个组成要素：推荐候选对象、用户、推荐方法。用户可以向推荐系统主动提供个人偏好信息或推荐请求，或者用户不提供，而是推荐系统主动采集。推荐系统可以使用不同的推荐策略进行推荐，如将采集到的个性化信息和对象数据进行计算得到推荐结果，或者直接基于已建模的知识数据库进行推荐。推荐系统将推荐结果返回给用户使用。

2.1.用户建模

用户模型反映用户的兴趣偏好。用户兴趣的反馈可分为显性反馈和隐性反馈。

显性反馈包含两种方式：用户定制和用户评分。用户定制是指用户对系统所列问题的回答，如年龄、性别、职业等。评分又分为两级评分和多级评分。例如，在 YahooNews 中采用两级评分：喜欢（more like this）和不喜欢（less like this）。多级评分可以更详细地描述对某个产品的喜欢程度，如 GroupLens 中用户对新闻的喜好程度可评价为 1~5 分。 News Dude 支持用户的 4 级反馈：感兴趣、不感兴趣、已知道、想了解更多，然后进行归一化处理。

很多时候用户不能够准确地提供个人偏好或者不愿意显性提供个人偏好，更不愿意经常维护个人的偏好。隐性反馈往往能够正确地体现用户的偏好以及偏好的变化。

常用的隐性反馈信息有：是否点击、停留时间、点击时间、点击地点、是否加入收藏、评论内容（可推测用户的心情）、用户的搜索内容、社交网络、流行趋势、点击顺序等。

在协同过滤推荐方法中，常常把用户的隐性反馈转化为用户对产品的评分。例如，Google News 中用户阅读过的新闻记为喜欢，评分为 1；没有阅读过的评分为 0。Daily Learner 系统中用户点击了新闻标题评分为 0.8 分，阅读完全文则评分上升到 1 分；若用户跳过了系统推荐的新闻，则从系统预测评分中减去 0.2 分作为最终评分。

用户的兴趣可分为长期兴趣和短期兴趣：长期兴趣反映用户的真实兴趣，短期兴趣常与热点话题相关联且经常改变，从最近的历史行为中学习到的短期兴趣模型可快速反映用户兴趣的变化。

2.2. 协同过滤算法

协同过滤技术是由 David Goldberg 在 1992 年提出的，是目前个性化推荐系统中应用最为成功和广泛的技术。

核心思想：如果两个用户对于一些项（item）的评分相似程度较高，那么一个用户对于一个新项的评分很有可能类似于另一个用户。

简介：通过在用户的一系列行为中寻找特定模式来产生用户特殊推荐

输入：仅仅依赖于惯用数据（例如评价、购买、下载等用户偏好行为）

注意：该算法不依赖于项的任何附加信息或者用户的任何附加信息（例如人口统计相关数据）。

该算法有两个类型：基于邻域的协同过滤，即直接根据用户对已有项的评分预测用户对新项的评分；基于模型的协同过滤，是指建立预测模型，让计算机学习模型（机器学习），然后将历史数据作为输入，得到预测的结果

2.3. 基于邻域的协同过滤算法

基于邻域的协同过滤算法可分为基于用户（UserCF）的和基于项（ItemCF）的两种。

基于用户的算法：适用于 item 更新频繁的应用。根据用户对物品的行为，找出兴趣爱好相似的一些用户，将其中一个用户喜欢的东西推荐给另一个用户。如，老张喜欢看的书有 A，B，C，D；老王喜欢看的书有 A，B，C，E。通过这些数据我们可以判断老张和老王的口味略相似，于是给老张推荐 E 这本书，同时给老王推荐 D 这本书。

基于项的算法：适用于 item 的增长速度远小于 user 的增长速度的情况。根据用户的喜好先找出相似的物品，如果同时喜欢两个物品的人比较多的话，就认为这两个物品相似。并给用户推荐和他原有喜好类似的物品。这种相似是基于项的共同出现几率（例如用户买了 X，同时时也买了 Y）。如，我们发现喜欢看《从一到无穷大》的人大都喜欢看《什么是数学》，那么如果你刚津津有味地看完《从一到无穷大》，我们就可以立马给你推荐《什么是数学》。

UserCF 更接近于社会化推荐，比较精准，适用于用户少，物品多，时效性较强

的场合，具有推荐新信息的能力，可以发现用户潜在但自己尚未察觉的兴趣爱好，能够推荐艺术品、音乐、电影等难以进行内容分析的产品。代价是运算量很大，而且对于新来的人(听得少，动作少)，也不太好用。

ItemCF 则更接近个性化推荐，适用于用户多，物品少的场合，比如豆瓣的豆瓣猜、豆瓣 FM，同时 ItemCF 还可以给出靠谱的推荐理由，例如豆瓣的「喜欢 OO 的人也喜欢 XX」和亚马逊的「买了 XX 的人也买了 OO」。

协同过滤算法最明显的一个缺点就是热门物品的干扰，即该算法经常会导致两个不同领域的最热门物品之间具有较高的相似度。如给喜欢《算法导论》的同学推荐《哈利波特》，显然是不科学的！要避免这种情况就得从物品的内容数据入手，也就是下面所说的基于模型的算法。

2.4. 基于模型的协同过滤算法

基于模型的方法会在使用评分去学习预测模型的基础上，去预测新项。一般的想法是使用机器学习算法建立用户和项的相互作用模型，从而找出数据中的模式。在一般情况下，基于模型的 CF 被认为是建立 CF 推荐系统的更先进的算法。有许多不同的算法可用于构建模型并基于这些模型进行预测，例如，贝叶斯网络、聚类、分类、回归、矩阵分解、受限玻尔兹曼机等等。

基于低秩矩阵分解的方法是最近一类成功的基于模型的方法。例如，SVD 和 SVD++ 将评价矩阵分解为 3 个低秩的矩阵，这 3 个矩阵的乘积能对原始矩阵进行某种程度的复原，从而可以评估出缺失值。另一种方法是非负矩阵分解（Non-negative matrix factorization，NMF），其不同之处在于，矩阵分解的结果不得出现负值。基于低秩矩阵分解的方法从评分矩阵中抽取一组潜在的（隐藏的）因子，并通过这些因子向量描述用户和物品。在电影领域，这些自动识别的因子可能对应一部电影的常见标签，比如风格或者类型（戏剧片或者动作片），也可能是无法解释的。

2.5. 二部图网络推荐算法

二部图网络结构是复杂网络中一种重要的表现形式，具有--定的普遍性，是复杂科学中的研究热点。基于二部图网络的推荐算法以二部图网络中的节点及连边代

替传统推荐算法中的角色及其选择关系，并通过在网络结构上进行形式化的计算来发掘用户的兴趣点。二部图是的二分网络的数据结构表现形式，它是由两种不同类型的节点集以及这些节点间相连的边所组成的网状结构。设 G={V,E},是一个无向图，它的顶点集 V 包括两个子集V_1,和V_2，其满足以下条件:$V = V_1 \cup V_2$ && $V_1 \cap V_2 = \emptyset$ ， $\forall e = (u,v) \in E$ && $u \in V_1, v \in V_2$。我们就把他称为一个典型的二部图。

基于图的推荐算法由 Aggarwal 于 1999 年首次提出，并迅速成为了个性化推荐领域中新的研究热点。基于二部图网络的推荐算法以二部图中的节点及其连边代替传统推荐算法中的角色及其选择关系，并通过在网络结构上进行形式化的计算来发掘用户的兴趣点。当用户和项目之间发生选择关系时，在二部图结构中的相应节点间便出现了一一条代表这种选择关系的连边，我们认为在相连的节点之间存在着某种可传递的能力值。

算法的基本思想如下:在一个二部图网络中，推荐系统希望通过节点之间的选择关系，向目标用户进行推荐。当目标用户选择了某项目时，代表这个项目中必然存在着某种用户感兴趣的属性值，它代表一种可传递的能力值，根据二部图网络的关联结构，这个项目可以将自身的属性值传递给与其相连的项目节点。通过对目标用户所有选择过的项目，上的属性值进行传递及计算，便得到了用户感兴趣的属性在二部图，上各项目节点中的分布。

在二部图网络的基础上引入扩散动力学，实现了物质扩散(Mass Diffusion)及热传导(Heat Conduction)推荐算法。下面我们对两种算法分别进行分析。

物质扩散算法：

基于物质扩散(Mass Diffusion)的推荐算法假设目标用户选择过的所有项目都具有一定的其喜 爱的属性，并且可 以通过网络结构中节点之间的连边来传递这种属性，我们称这种属性为节点所拥有的资源值。拥有这种资源的项目节点会把资源更多的传递给目标用户喜爱的项目。

热传导算法：

推荐系统中的热传导过程类似于热量在用户一项目二部图中扩散的过程。基于热传导的推荐算法(HCBI)将物质扩散算法中代表推荐能力的资源命名为物质，

3. 算法流程

3.1. 基于邻域的协同过滤算法

基于用户的协同过滤算法：

首先用某一种相似性度量方法（余弦相似性或皮尔逊相关系数）计算用户的相似矩阵，即A_{ij}表示第 i 个用户和第 j 个用户的相似性。

然后选择一个需要进行推荐的用户 U，检查他有哪些商品没有买，同时筛选出 n 个与他最相似的用户（n 手动指定）。对于没有买的商品，我们取出所有 n 个用户对该商品的评分（如果未评分，就不取），然后按照他们相对于用户 U 的相似度求加权平均分数。举个例子，比如两个用户评分分别为 2，3，与用户 U 的相似度分别为 0.6,0.5,那么结果为$\frac{2*0.6+3*0.5}{0.6+0.5}$ 。如果这 n 个用户里没有人对该商品评了分，我们就预测为 0 分，因为与用户 U 相似的人一个都没有买，很可能用户 U 也不会买。最后按照得分高低，向用户推荐评分前 m 高的商品（m 手动指定）

基于项的协同过滤算法：

与基于用户的算法相似，首先用某一种相似性度量方法计算物品的相似矩阵。然后选择一个需要进行推荐的商品 I，检查哪些用户没有买这个商品，同时筛选出 n 个与商品 I 最相似的商品。对于没有买商品 I 的用户，我们取出所有 n 个商品被该用户评的分，然后按照他们相对于商品 I 的相似度求加权平均分数。如果这 n 个商品里该用户一个都没有买，我们就预测为 0 分。

3.2. 基于模型的协同过滤算法

首先假设有 n 个特征（手动设置）。有三个矩阵，D 是用户评分矩阵，U 是用户-特征矩阵，V 是商品-特征矩阵，然后建立损失函数(loss function)，比如下面这个损失函数：

$$
\begin{aligned}
L &= \sum_{i=1}^{m}\sum_{j=1}^{n}(D_{ij} - U_i V_j^T)^2 + \lambda(||U||^2 + ||V||^2) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n}(D_{ij} - U_i V_j^T)^2 + \lambda(\sum_{i=1}^{m}||U_i||^2 + \sum_{j=1}^{n}||V_j||^2) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n}(D_{ij} - \sum_{k} U_{ik} V_{jk})^2 + \lambda(\sum_{i=1}^{m}\sum_{k}||U_{ik}||^2 + \sum_{j=1}^{n}\sum_{k}||V_{jk}||^2)
\end{aligned}
$$

然后使用梯度下降算法，求 L 对于 U 和 V 的偏导数，结果为：

$$
\begin{aligned}
\frac{\partial L}{\partial U} &= -2RV + 2\lambda U \\
\frac{\partial L}{\partial V} &= -2R^T U + 2\lambda V
\end{aligned}
$$

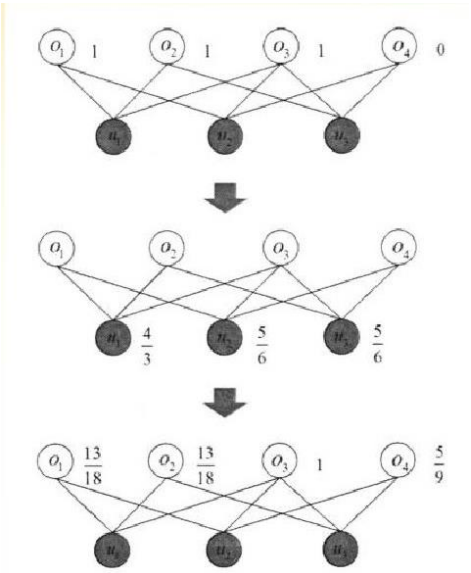
可以发现，这里的偏导数在经过数学运算之后变成了常见的矩阵运算，然后直接套用梯度下降算法，可以得到 U，V 的更新公式：其中α是学习率，值越大，更新幅度就会越大，但是不能太大，否则会发散，不能太小，因为会收敛很慢。

$$
\begin{aligned}
U &= U - \alpha * \frac{\partial L}{\partial U} \\
V &= V - \alpha * \frac{\partial L}{\partial V}
\end{aligned}
$$

3.3. 二部图网络推荐算法

物质扩散算法：

具体推荐过程分为三步:首先，为所有目标用户选择过的项目分配一个初始值，在这里我们将初始值设为 1，它代表了某种用户喜爱的属性。然后，根据用户和项目之间的选择关系，把项目节点上的初始值按照一定的方式传递给用户节点。最后，计算所有用户节点获得的分配值，并把它们按照同样的传递方式返回给项目节点，最终，每个项目节点都获得了一定的代表目标用户喜爱特性的属性值。算法中每个节点分配给对应节点的分配值都是通过其自身拥有的初始值除以节点度得到的。



如左图所示，下为用户，上为商品。比如要给 1 号用户推荐，首先把买过的 1，2，3 号商品权值赋值为 1，然后假设他们按等概率流回用户结点。

1 号商品，就会拿 1/2 给 1 号用户，1/2 给 2 号用户。

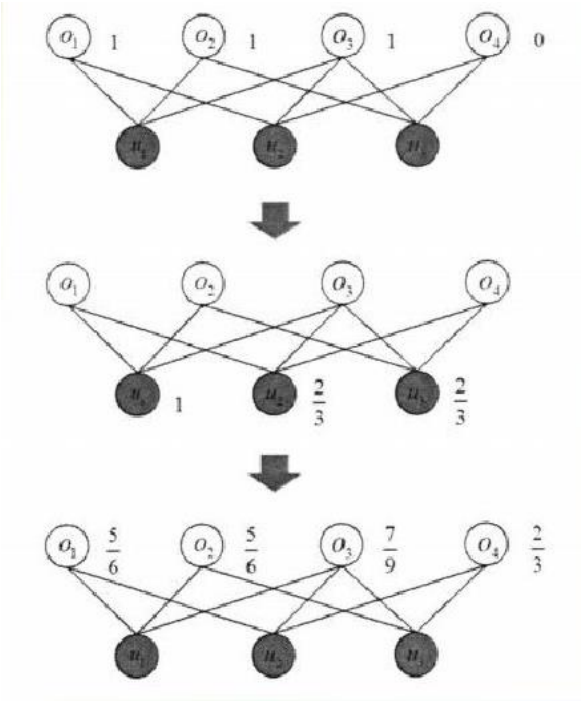
2 号商品各拿 1/2 给 1 号和 3 号用户。

3 号商品各拿 1/3 给 1，2，3 号用户

用户的权值按照相同的方式回流，就得到了最终结果，权值越高，就表示 1 号用户可能越喜欢。

热传导算法：

热量的传递同样通过三步来完成:首先，我们同样为所有目标用户选择过的项目分配一个初始值 I，它代表了某种用户喜爱的属性。然后根据选择关系，把项目节点上的初始值按照一定的方式传递给用户节点。最后，计算所有用户节点获得的分配值，并把它们按照同样的传递方式返回给项目节点，通过计算，每个项目节点最终都获得了一定的代表目标用户喜爱特性的属性值。在这里热量传递的方式为每个节点所得到的所有相连节点分配给它的值的和除以这个节点的度。



仍然假设给 1 号用户推荐，1，2，3 号商品初值为 1.

第一轮传播：

1 号用户的权值是$\frac{1+1+1}{3}=1$

2，3 号用户的权值是$\frac{1+1+0}{3}=\frac{2}{3}$

第二轮传播：

1 号商品权值 $\frac{1+\frac{2}{3}}{2}=\frac{5}{6}$

2 号商品权值 $\frac{1+\frac{2}{3}}{2}=\frac{5}{6}$

3 号商品权值$\frac{1+\frac{2}{3}+\frac{2}{3}}{3}=\frac{7}{9}$

4 号商品权值$\frac{\frac{2}{3}+\frac{2}{3}}{2}=\frac{2}{3}$

4. 核心算法代码

4.1. 基于邻域的协同过滤算法

```
//计算用户之间的相似矩阵
public void getSimMatrix_UserCF(){
    sim = Init.initArray2(userNum,userNum,-1.0);
    for(int i=0;i<userNum;++i){
        sim.get(i).set(i,1.0);//自己和自己的一定是 1
        for(int j=i+1;j<userNum;++j){
            //使用余弦方式计算相关系数
            double xy=0,y2=0,x2=0;
            for(int k=0;k<movieNum;++k){
                xy += rating.get(i).get(k)*rating.get(j).get(k);
                y2 += MyMath.sqr(rating.get(j).get(k));
                x2 += MyMath.sqr(rating.get(i).get(k));
            }
            double ans = xy/Math.sqrt(y2*x2);
            sim.get(i).set(j,ans);
            sim.get(j).set(i,ans);
        }
    }
}

//给某一个用户推荐商品
public Integer[] recommend_UserCF(int userID){
    Integer[] v = new Integer[userNum];
    for(int i=0;i<userNum;++i) v[i]=i;
    //按照相似度排序，记得处理时要去除自己
    Arrays.sort(v, (o1, o2) -> -
sim.get(userID).get(o1).compareTo(sim.get(userID).get(o2)));

    predict = Init.initArray2(userNum,movieNum,-1.0);
    for(int i=0;i<movieNum;++i)
        if(true || rating.get(userID).get(i)==-1.0) {//枚举没买的
            double fz = 0, fm = 0;//定义分子分母
            for (int j = 1; j < Math.min(userNum, 1 + mostSimNum);
++j) {//枚举用户
                int user2 = v[j];
```

```
                if (rating.get(user2).get(i) == -1.0) continue;
                fz += sim.get(userID).get(user2) * rating.get(user2
).get(i);
                fm += sim.get(userID).get(user2);
            }
            if (fm == 0) continue;
            predict.get(userID).set(i,fz/fm);
        }
    Integer[] movie = new Integer[movieNum];
    for(int i=0;i<movieNum;++i) movie[i]=i;
    Arrays.sort(movie, (o1, o2) -> -
predict.get(userID).get(o1).compareTo(predict.get(userID).get(o2)))
;
    return movie;
}

//给所有用户推荐商品
public void recommendForAll_UserCF(){
    for(int i=0;i<userNum;++i){
        System.out.printf("正在为第%d 位用户推荐:",i+1);
        Integer[] rec = recommend_UserCF(i);
        for(int j=0;j<mostRecNum;++j){
            System.out.printf("%d-%.1f, ",rec[j]+1,predict.get(i).g
et(rec[j]));
        }
        System.out.println();
    }

    //评估部分
    double sumMSE=0;
    for(int i=0;i<userNum;++i){
        System.out.printf("第%d 位用户的均方差:",i);
        double MSE=0;
        for(int j=0;j<movieNum;++j){
            MSE += MyMath.sqr(rating.get(i).get(j)-
predict.get(i).get(j));
        }
        sumMSE += MSE/movieNum;
        System.out.println(MSE/movieNum);
    }
    System.out.println("总均方差:"+sumMSE/userNum);
}

//计算商品之间的相似性
```

```
public void getSimMatrix_ItemCF(){
    sim = Init.initArray2(movieNum,movieNum,-1.0);
    for(int i=0;i<movieNum;++i){//枚举两个 item 计算相关度
        System.out.printf("给第%d 个项计算相似度\n",i);
        sim.get(i).set(i,1.0);
        for(int j=0;j<movieNum;++j){
            double xy=0,x2=0,y2=0;
            for(int k=0;k<userNum;++k){
                xy += rating.get(k).get(i)*rating.get(k).get(j);
                x2 += MyMath.sqr(rating.get(k).get(i));
                y2 += MyMath.sqr(rating.get(k).get(j));
            }
            double ans=xy/Math.sqrt(x2*y2);
            sim.get(i).set(j,ans);
            sim.get(j).set(i,ans);
        }
    }
}

//将某个商品推荐给用户
public Integer[] recommend_ItemCF(int movieID,Double[] predict){
    Integer[] movie = new Integer[movieNum];
    for(int i=0;i<movieNum;++i) movie[i]=i;
    Arrays.sort(movie, (o1, o2) -> -
sim.get(o1).get(movieID).compareTo(sim.get(o2).get(movieID)));

    for(int i=0;i<userNum;++i) if(rating.get(i).get(movieID)==-
1.0){
        double fm=0,fz=0;
        for(int j=1;j<Math.min(movieNum,1+mostSimNum);++j){
            int movie2=movie[j];
            if(rating.get(i).get(movie2)==-1.0) continue;
            fz+=sim.get(movie2).get(movieID)*rating.get(i).get(movi
e2);
            fm+=sim.get(movie2).get(movieID);
        }
        if(fm==0) continue;
        predict[i] = fz/fm;
    }

    Integer[] user = new Integer[userNum];
    for(int i=0;i<userNum;++i) user[i]=i;
    Arrays.sort(user, (o1, o2) -> -
predict[o1].compareTo(predict[o2]));
```

```
    return user;
}

//将所有商品推荐给用户
public void recommendForAll_ItemCF(){
    try {
        PrintStream ps = new PrintStream("D:/log.txt");
        PrintStream out = System.out;
        for (int i = 0; i < movieNum; ++i) {
            System.setOut(out);
            System.out.printf("%d 电影处理中\n",i);
            System.setOut(ps);
            System.out.printf("%d 电影被推荐给: ", i);
            Double[] predict = new Double[userNum];
            Arrays.fill(predict,-1.0);
            Integer[] rec = recommend_ItemCF(i, predict);
            for (int j = 0; j < mostRecNum; ++j) {
                System.out.printf("%d-%.1f, ", rec[j], predict[rec[
j]]);
            }
            System.out.println();
        }
    }catch (Exception e){
        e.printStackTrace();
    }
}
```

4.2. 基于模型的协同过滤

由于涉及矩阵运算，所以使用了 matlab 完成，同时使用均方差评估

```
clear all;clc;
data=importdata('C:\Users\QWsin\Desktop\文档\课程相关\数据挖掘课设\课程资料_推荐系统\ml-latest-small\ratings_disc2.txt');
u=max(data(:,1));%用户数
m=max(data(:,2));%电影数
D=-ones(u,m);
[r,c]=size(data);
for i=1:r
    D(data(i,1),data(i,2))=data(i,3);
end

n=5;%特征维数
```

```
alpha=0.0001;%学习率
lambda=0.1;

U=randn(u,n);
M=randn(m,n);
predict=zeros(u,m);
Q=ones(u,m);
Up=U;
Mp=M;
tmp = find(D==-1);
Q(tmp)=0;
for j=1:3000
    predict=U*M'.*Q;
    R=D-predict;
    for i=1:u
        tmp=find(Q(i,:)==1);
        M_=M(tmp,:);
        Up(i,:)=(U(i,:)*M_'-D(i,tmp))*M_+lambda*U(i,:);
    end
    U=U-alpha*Up;
    for i=1:m
        tmp=find(Q(:,i)==1);
        U_=U(tmp,:);
        Mp(i,:)=(U_*M(i,:)'-D(tmp,i))'*U_+lambda*M(i,:);
    end
    M=M-alpha*Mp;
    %U=U-2*alpha*(-R*M+lambda*U);
    %M=M-2*alpha*(-R'*U+lambda*M);
    L=sum(sum(R.^2))+lambda*(sum(sum(U.^2))+sum(sum(M.^2)));
    fprintf("j=%d L=%.6f\n",j,L);
end

for i=1:20
    fprintf("%.4f ",predict(1,i));
end

[r,c]=size(find(D==-1));
MSE=sum(sum((predict.*Q-D).^2))/(u*m-r);
```

4.3. 二部图推荐算法

```
//对某个用户进行扩散推荐，heat 表示是否是热传导方式
public void diffusion(int userID, Double[] val,boolean heat){
    Arrays.fill(val,0.0);
    for(int u=userNum;u<userNum+movieNum;++u) val[u]=0.0;//先全部当作
没买过
    for(int v : graph.to(userID)) {
        val[v] = rating.get(userID).get(v - userNum)*4;//设置初值
//            System.out.println(v+" "+val[v]);
    }
    for(int clk=0;clk<1;++clk) {
        for (int u = userNum; u < userNum + movieNum; ++u)
            if (Math.abs(val[u]) > 1e-9) {//枚举商品结点
                for (int v : graph.to(u)) {
                    if(heat) val[v] += val[u] / graph.getDegree(v);
                    else val[v] += val[u] / graph.getDegree(u);
                }
                val[u] = 0.0;//归零
            }
        for (int u = 0; u < userNum; ++u)
            if (Math.abs(val[u]) > 1e-9) {
                for (int v : graph.to(u)) {
                    if(heat) val[v] += val[u] / graph.getDegree(v);
                    else val[v] += val[u] / graph.getDegree(u);
                }
                val[u] = 0.0;
            }
    }
}

//对所有用户进行扩散算法，heat 表示是不是热传导，因为两个算法差别非常非常小。
public void diffusionAll(boolean heat){
    Double[] val = new Double[userNum+movieNum];//记录权值

    double sumrho=0;
    for(int i=0;i<userNum;++i){
        diffusion(i,val,heat);
        double[] x = new double[movieNum];
        for(int j=0;j<movieNum;++j){
            x[j] = val[j+userNum];
        }
        double rho=0,xMean=MyMath.mean(x),yMean=MyMath.mean(rating.
get(i));

        for(int j=0;j<x.length;++j){
```

```
            rho+=(x[j]-xMean)*(rating.get(i).get(j)-yMean);
        }
        double fm = 0;
        for(int j=0;j<x.length;++j){
            fm += (x[j]-xMean)*(x[j]-xMean);
            fm += MyMath.sqr(rating.get(i).get(j)-yMean);
        }
        rho /= fm;
        sumrho += rho;
        System.out.printf("第%d 个用户的相关系数:%.6f\n",i,rho);
    }
    System.out.println("总相关系数:"+sumrho/userNum);
}
```

5. 实验结果截图及分析

5.1. 基于领域的协同过滤算法

基于用户的协同过滤：参数设置为找 3 个最相似的用户，推荐 3 部电影

部分推荐结果示例：格式为电影编号-分数

```
正在为第 1 位用户推荐:1-5.0, 5-5.0, 8-5.0,
正在为第 2 位用户推荐:239-5.0, 253-5.0, 21-4.5,
正在为第 3 位用户推荐:79-4.5, 86-4.5, 35-4.0,
正在为第 4 位用户推荐:179-5.0, 180-5.0, 185-5.0,
正在为第 5 位用户推荐:1-5.0, 8-5.0, 29-5.0,
正在为第 6 位用户推荐:29-5.0, 30-5.0, 233-5.0,
正在为第 7 位用户推荐:16-5.0, 69-5.0, 71-5.0,
正在为第 8 位用户推荐:1-5.0, 43-5.0, 8-4.3,
…
第 0 位用户的均方差:0.7022830111065406
第 1 位用户的均方差:0.07489201974496092
第 2 位用户的均方差:0.06442821883998355
第 3 位用户的均方差:0.49917729329494037
第 4 位用户的均方差:0.10160427807486631
```

第 5 位用户的均方差:0.6753393665158371

第 6 位用户的均方差:0.3071781160016454

…

总均方差: 0.36320280294272234

根据这个均方差来看，平均差值大约在 0.6 左右，比较准确。

基于项的协同过滤：

参数同上，部分推荐结果展示

0 电影被推荐给: 572-5.0, 19-5.0, 55-5.0, 第 0 电影均方差 0.429720

1 电影被推荐给: 0-5.0, 41-5.0, 491-5.0, 第 1 电影均方差 0.096281

2 电影被推荐给: 13-5.0, 30-5.0, 51-5.0, 第 2 电影均方差 0.299058

3 电影被推荐给: 4-5.0, 6-5.0, 10-5.0, 第 3 电影均方差 0.581152

4 电影被推荐给: 21-5.0, 29-5.0, 38-5.0, 第 4 电影均方差 0.674305

5 电影被推荐给: 38-5.0, 552-5.0, 167-4.5, 第 5 电影均方差 0.125797

6 电影被推荐给: 274-4.0, 509-4.0, 602-4.0, 第 6 电影均方差 0.031218

…

总均方差: 0.01685993

这个均方差的结果是非常优秀的，可能是因为电影相对用户来说很多。

5.2. 基于模型的协同过滤算法

部分输出展示：其中 j 是迭代次数，L 是 loss function 的值

j=1 L=7600786.543373

j=2 L=7373897.737997

j=3 L=7343461. 026954

j=4 L=7363816.221405

j=5 L=7389069.303737

j=6 L=7353915.884586

j=7 L =7265729.471042

j=8 L=7154812.698199

j=9 L=7055203.649555

j=10 L =6940022. 643163

j=11 L =6854166. 760648

j=12 L =6798204.600289

j=13 L=6801059.461798

j=14 L=6734015.004711

…

均方差：0.5805

取了 1 号用户对前 10 个电影的评分：

电影	1	2	3	4	5	6	7	8	9	10
预测	4.7859	3.7559	2.9648	3.4778	3.6138	4.1807	4.0308	3.9741	3.8646	2.6997
实际	4	4	4	5	5	3	5	4	5	5

5. 3. 基于二部图的协同过滤算法

物质扩散结果：使用的是皮尔逊相关系数，部分结果展示如下

第 0 个用户的相关系数:0.277264

第 1 个用户的相关系数:0.285422

第 2 个用户的相关系数:0.415560

第 3 个用户的相关系数:0.270558

第 4 个用户的相关系数:0.288195

第 5 个用户的相关系数:0.300953

第 6 个用户的相关系数:0.267371

…

总相关系数:0.267607

热传导结果：使用的同样是皮尔逊相关系数，部分结果展示如下

第 0 个用户的相关系数:0.141265

第 1 个用户的相关系数:0.163556
第 2 个用户的相关系数:0.347432
第 3 个用户的相关系数:0.176542
第 4 个用户的相关系数:0.139776
第 5 个用户的相关系数:0.243175
第 6 个用户的相关系数:0.121066
第 7 个用户的相关系数:0.136899
第 8 个用户的相关系数:0.211071
…
总相关系数:0.151642

在运行这两个算法的时候，我发现了一些有趣的事情：①扩散和传导可以做多次，但是如果做多次的话，相关性会下降，猜测是因为传导多次之后，各结点的权值就越发趋于相似，和现实生活中的情况也是一样的。②对于初值，相比直接赋评分（没看为 0 分）的方式，在评分上面乘上一个系数，可能会使相关系数更大，经测试，4~6 大概是一个峰值，比 4 小的时候，增加系数会使相关系数增加，大于 6 之后，增加系数反而会使相关系数下降。但是总的来说，相关系数都不是很大

6. 总结

五种算法分析比较：

算法	正确率	时间
基于用户的协同过滤算法	较高	较短
基于项的协同过滤算法	较高	较长
基于模型的协同过滤算法	较高	较长
物质传播模型	较低	较短
热传导模型	较低	较短

协同过滤算法的表现比较优异，同时，由于 MovieLens 数据中，电影（项）多达 9724 个，所以使用基于项的协同过滤算法就会很慢，这时候就应灵活选择选择

基于用户的算法。

这次的算法理解难度和数学要求高一点，涉及到了机器学习常用的梯度下降算法，自然也就涉及到，求偏导数，矩阵运算等，需要一定的算式推导过程。本次报告的某些算法需要运行不短的时间，比如基于模型的协同过滤算法和基于项的协同过滤算法，有必要对代码进行优化或者对数据做一些处理（比如离散化）来加快速度。同时不少地方也需要调整参数，比如α学习率的调整和进行基于模型的协同过滤算法时，应该设置有多少维特征等。有较强的综合性。