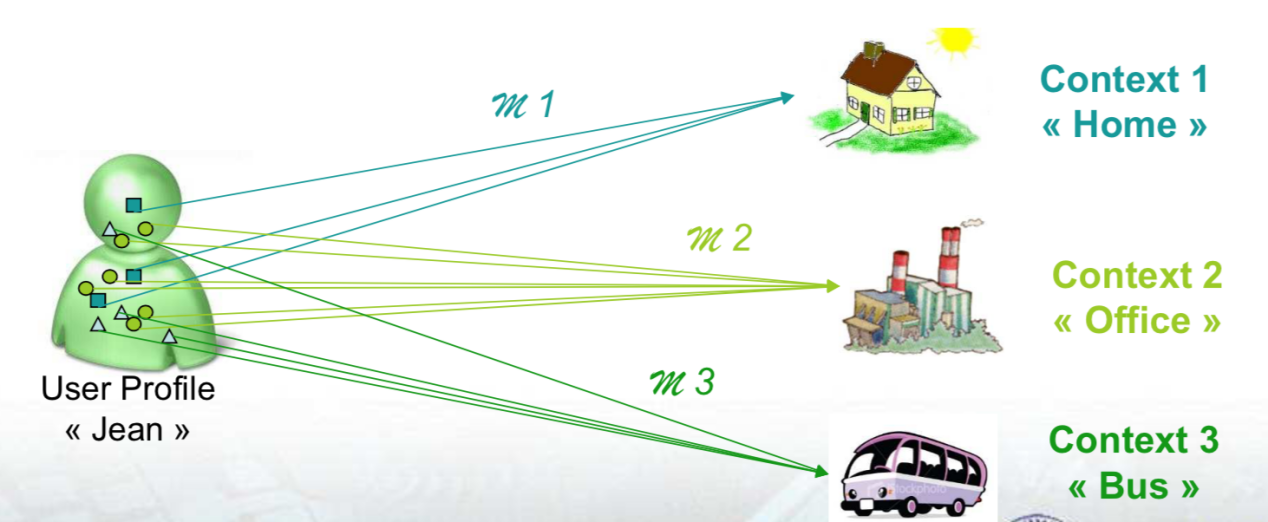
第八章 基于情景感知的推荐

# 8.1 背景

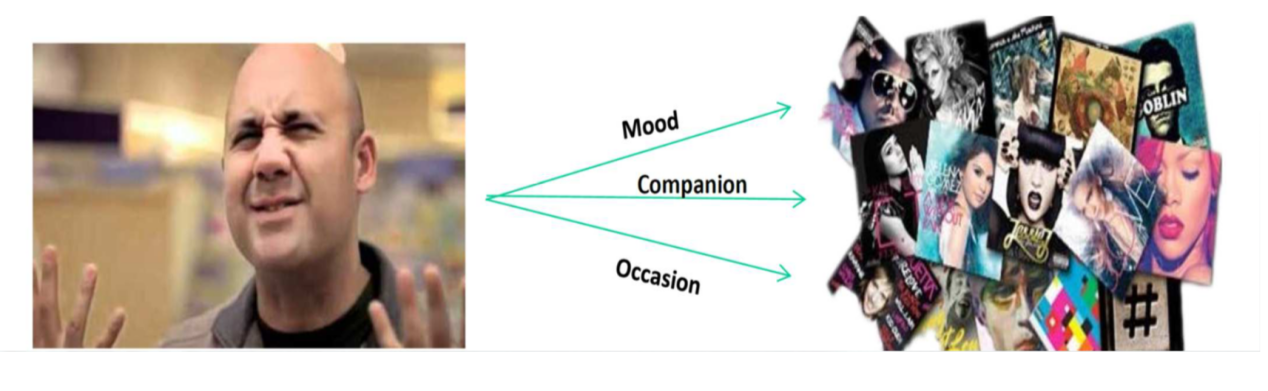
在日常生活中，随着科技地不断进步，信息量呈指数级增长，海量的数据带来了数据过载的问题。推荐系统能够有效克服数据过载，它通过分析用户的历史行为记录、挖掘用户兴趣爱好,构建出用户偏好模型,并通过向用户主动地推荐其最感兴趣的内容,帮助用户在海量的信息中快速地发现其所期望的信息,还能引导用户发现其潜在的偏好。传统的推荐系统主要利用用户和项目之间的关联性进行推荐,而忽略了用户所处的情境,因此推荐的结果并不能满足用户特定场景中的需要,导致推荐不够准确、用户满意度偏低等问题。情境感知推荐系统通过将情境信息融入到推荐过程,为用户提供了更为准确的推荐,并已成为推荐领域研究的重点和热点。

实际上，推荐系统的目标是：在恰当的时间、恰当的地点、恰当的场合，通过恰当的媒介，给用户推荐能满足用户偏好、需求和意图的信息。例如：当我们知道了某个用户的概况，以及用户对《Jean》的打分，那么我们就可以预测其对其他物品的打分或者向其推荐类似的商品。



情境感知技术最早是由Schilit于1994年提出的。情境感知，就是通过智能手机或可穿戴设备应用软件等"感知"到当前的环境和情境，从而利用环境和情境信息为用户提供更为人性化的设计体验

有时候我们需要根据用户的动机来进行情景感知。用户兴趣并非一成不变的，而是随着所处环境而改变。例如在电影推荐中，所推荐的电影更容易受到用户同伴的影响，而用户与其情侣、朋友或父母感兴趣的电影主题很可能有区别；在新闻推荐中，用户在工作日早晨倾向于阅读新闻时事，晚上则更愿意了解股票信息，而在周末需要的可能是娱乐新闻或购物信息；而在电子商务推荐中，电商推荐会受用户购买意图的影响，不同的购物意图可能导致不同的购物行为，比如：购书以提升专业技能、买包或鲜花作为礼物、买电子产品满足生活需要等。



# 8.2 情境信息的定义

一般来说，情境信息指的是能对某些事情产生影响的条件和环境。目前，对情境信息没有形成统一的定义，这里我们将刻画情境环境的因素统称为情境信息。除”用户-项目”评分信息外，影响推荐系统且能辅助预测的所有因素。

常用的情境信息有：

物理：时间、地点（位置）、天气、温度、用户行为等；

社交：和什么人在一起（同伴）；

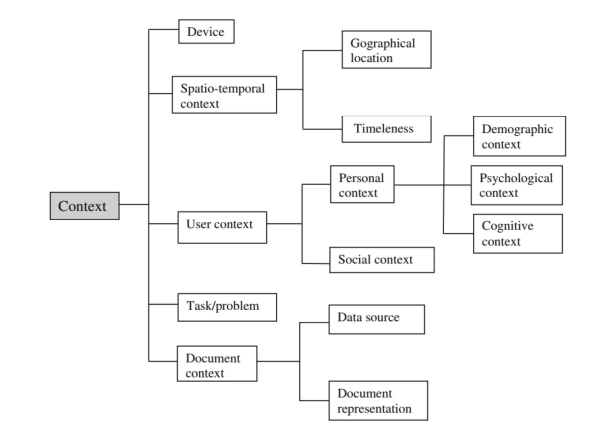
交互媒体：访问设备（PC、Pad）、正浏览的媒体类型（文本、图片、视频）；

情绪：用户当前的心情、用户意图（购买意图）、用户体验、认知

我们可以用如下公式来定义情境信息：

情境 = （内在）情感 + （外部）环境

一个完整的情境信息包含如下方面：



# 8.3 情境信息的获取

总的来说，情境信息的获取可以分为如下三种方式：

1、显式获取

通过直接接触相关人士和其他情境信息源或通过直接问问题或引导性的方式显式获得这些信息，例如：某些网站需要用户填写一些调查表后才能访问信息

2、隐式获取

隐式地从数据或环境中获得，例如：通过手机GPS获得用户位置信息；利用事务时间戳获得时间情境信息

3、推理获取

通过统计或机器学习方法推断出情境信息，例如：有线电视公司难以显式获得观众的家庭角色，但可以通过观看的电视节目较准确地推断出来。



综上三种情境获取的方式中，各有优缺点，显示获取最为直接简单，它能最大化的利用原始数据信息，所以显示获取的情境信息在三种方法中最为精确，但是在现实具体情境中，显示获取很难获取到更代表用户兴趣偏好的情境信息。因此，应用最普遍的隐式获取和推理获取也显得极为重要，需要三者互相弥补才能达到最好的效果。

# 8.4 情境信息的表达

我们如何通过具体的数值描述将情境信息表达出来呢？情境信息具有聚合特性，具体表现在一类情境因素可能包含多个子属性，且情境信息呈现出极为复杂的层次结构。假设在电影推荐中，我们考虑如下情境因素：剧院（Theater）、时间（Time）和同伴（Companion），那么这几个因素又可以分为以下几个多元组：

情境信息的具体表达方式可以分为两种：树形结构的表达与数据立方体结构的表达。

## 8.4.1 树状表达

树形结构即对情境信息进行分层展示，将主要情境因素作为根节点的子节点，将各因素的属性作为其子节点，并以此类推。

图8.4.1

## 8.4.2 数据立方体

数据立方体则将情境信息进行三维展示，数据立方体的每个维都有一个关系表与之相关联，每类情境信息对应于数据立方体的一个维，情境信息则可以看成维表所有属性笛卡尔积的子集。

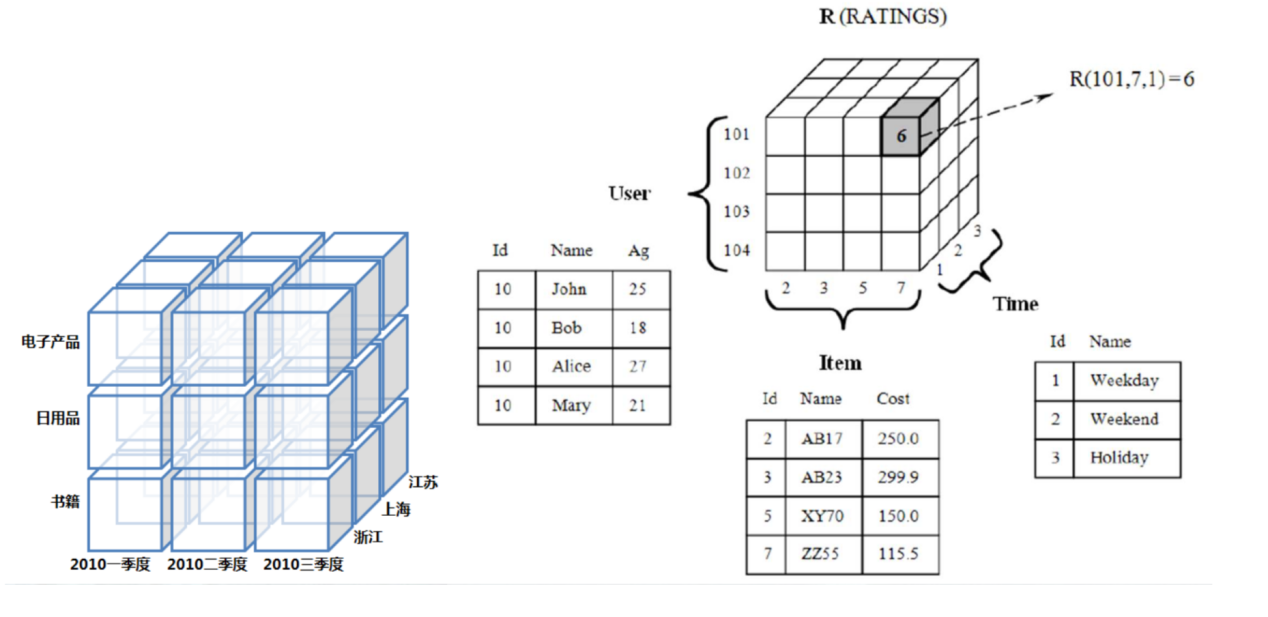


图8.4.2

# 8.5 融合情境信息的推荐模型

传统的推荐模型过程如图8.5.1所示：

（图8.5.1：传统的推荐模型过程。其中U：User I：Item R：Rating）

我们的原始数据为由User、Item、Rating构成的三维数据，然后通过2D推荐器，找到User、Item与Rating的内在联系，然后得到最终推荐结果，返回给用户。

然而，在不同的情景中，用户期望的推荐可能会有所不同，所以情景因素是我们在进行推荐计算时必须要考虑的因素，为此，我们提出本模型——融合情景信息的推荐模型。融合情景信息的推荐模型即是指基于情境信息的对用户进行偏好抽取和预测，然后给出推荐结果，即基于情境信息学习和勾勒用户兴趣。其评分函数定义为R:User × Item × Context →Rating，即User、Item、Context共同作用产生Rating结果

其中根据情境信息融入推荐生成过程可以有三种形式：a.情境预过滤 b.情境后过滤 c.情境化建模

## 8.5.1 情境预过滤

情境预过滤形式基本过程如图8.5.2所示：

Data

U\*I\*C\*R

C

Contextualized data

U\*I\*C\*R

2D recommender

U\*I->R

U

Contextual recommendations

i1,i2,i3...

（图8.5.2：情境预过滤形式基本过程）

利用情景信息直接对原始数据进行过滤，过滤掉在此情景下不适用的“User-Item”评分数据，此时我们得到的“Contextualized data（上下文化数据）”，是符合此情境限定的，对于后续推荐真正有用的数据集。然后，以过滤后的数据集为输入，采用传统推荐算法产生结果。

举个例子，我们现在规定时间限定为情景信息，进行基于时间预过滤的推荐。我们利用时间限定来对数据做一处理，用D[Time=t](User, Item, Rating) 表示过滤后的评分数据集。这样一来，一时间段的不同为判定条件，将原始数据分成了不同的子集，在不同的时间内，我们用不同的书进行分析推荐，能够更好地分析在当段时间内的用户行为，为其做更精确的推荐。

通过一定的条件对数据集划分子集，这一过程我们将它称为：情景化分片（contextual segment）。在此形势下，情景可以泛化，例如：将周一晚上十点扩大到周一晚上，再到工作日晚上，每天晚上，最后扩大至任意时间，我们将特殊情景泛化为范围更大的，更为一般的情景，这时只需要将不同子集进行合并操作，即可得到新情境下的数据集。

## 8.5.2 情境后过滤

情境后过滤形式基本过程如图8.5.3所示：

C

Data

U\*I\*C\*R

Contextualized data

U\*I\*C\*R

2D recommender

U\*I->R

U

Contextual recommendations

i1,i2,i3...

（图8.5.3：情境后过滤形式基本过程）

情景后过滤形式是指在推荐生成阶段不考虑情境信息的影响，首先基于传统推荐模型生成Top-N 推荐列表，然后将得到的Top-N 推荐列表根据情景信息进行一定的调整，然后生成符合情景条件的最终推荐结果。

调整的方式有两种：在给定的情境下从Top-N 推荐列表中过滤掉无关的项目；基于给定的情境调整Top-N 推荐列表的排序。

情景后过滤的具体实现如图8.5.4所示：

**DATA**

U\*I\*C\*R

**Input User**

U

**Input Context**

C

Item Usage Patterns

**Traditional Recommendations**

I1,i2,i3...

**Contextual**

**Recommendations**

I1,i2,i3...

**Item Adjustments**

（图8.5.4：情景后过滤实现）

在经过传统推荐流程，得到初步推荐之后，我们将原有的U、I、C、R四维数据以及上下文情景信息与用户信息作为参数进行输入，得到项目调整规则，将已经得到的推荐进行调整，进而得出最终的符合上下文情景的更加切合实用的推荐条目。

# 8.6 情境化建模

与情境预过滤和情境后过滤不同，情境化建模指的是将情境信息融入推荐生成过程，对情境信息以一种计算机能够处理的方式进行定义和存储。在推荐生成过程中，直接在推荐函数中把情境信息作为预测用户评分的显式因素来考虑，生成的是真正的多维推荐函数。

与情境预过滤和情境后过滤相比，情境化建模需要处理高维数据，在三者之中最为复杂，同时其最能有效挖掘用户、项目、情境三者之间的关联关系。情境化建模适用于情境信息与用户偏好耦合度紧密的情况。

情境化建模具有两种形式：启发式方法和基于模型对方法。

## 8.6.1 启发式情境化建模

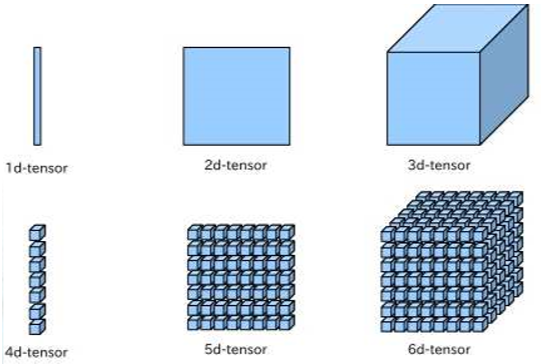
## 8.6.2 基于张量分解的推荐

目前的主要推荐算法包括基于内容、协同过滤和混合推荐算法等多种算法，其中基于用户、产品二元关系的矩阵分解是较有效的推荐方法。随着社会化标签的诞生，又产生了基于张量分解的推荐算法，将用户、产品的二元关系扩展为包含情景的用户、产品、情景（上下文）三元关系。

### 8.6.2.1 张量的概念

张量是一个线性映射的概念。向量是基于一个参考系下的具有方向的量，而张量则吸收了参考系的概念，如果这个量在不同的参考系下按照某种特定的法则进行变换, 就是张量了。这种概念的好处在于统一不同参考系下的量和表示，使张量的使用脱离开参考系的约束，即张量包含参考系基的信息。

以上理解主要从物理角度出发，而几何代数中定义的张量是基于向量和矩阵的推广，如果按照较为通俗的理解方式，我们可以将标量视为零阶张量，矢量视为一阶张量，那么矩阵就是二阶张量。



### 8.6.2.2 张量分解

1）矩阵分解的由来

矩阵补全是为了估计矩阵中缺失的部分（不可观察的部分），可以看做是用矩阵X近似矩阵M，然后用X中的元素作为矩阵M中不可观察部分的元素的估计。

矩阵分解指用 A\*B 来近似矩阵M，那么 A\*B 的元素就可以用于估计M中对应不可见位置的元素值，而A\*B可以看做是M的分解。

因为协同过滤本质上是考虑大量用户的偏好信息（协同），来对某一用户的偏好做出预测（过滤），那么当我们把这样的偏好用评分矩阵M表达后，这即等价于用M其他行的已知值（每一行包含一个用户对所有商品的已知评分），来估计并填充某一行的缺失值。若要对所有用户进行预测，便是填充整个矩阵，这是所谓“协同过滤本质是矩阵填充”。

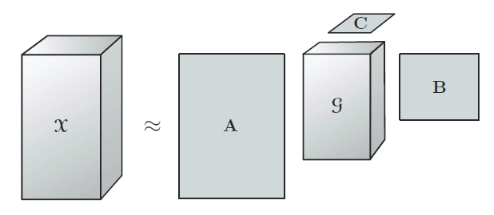
那么，这里的矩阵填充如何来做呢？矩阵分解是一种主流方法。这是因为，协同过滤有一个隐含的重要假设，可简单表述为：如果用户A和用户B同时偏好商品X，那么用户A和用户B对其他商品的偏好性有更大的几率相似。这个假设反映在矩阵M上即是矩阵的低秩。极端情况之一是若所有用户对不同商品的偏好保持一致，那么填充完的M每行应两两相等，即秩为1。

所以这时我们可以对矩阵M进行低秩矩阵分解，用U\*V来逼近M，以用于填充。如果我们使用一个m\*r的矩阵U和一个r\*n的矩阵V来近似M，即使用秩为r的矩阵U\*V近似m\*n的M达到预测效果。即利用了M的低秩性。

以下这种方法和思想将从二维的矩阵拓展到张量空间。

2）张量的Tucker分解

张量的Tucker分解是一种张量的分解方法，这种方法是PCA和张量CP分解的延申和高阶推广，如果核张量的各个维数相同并且是对角的,则Tucker分解就退化成了CP分解，如果固定两个因子矩阵，则Tucker 分解就退化成了普通的PCA。

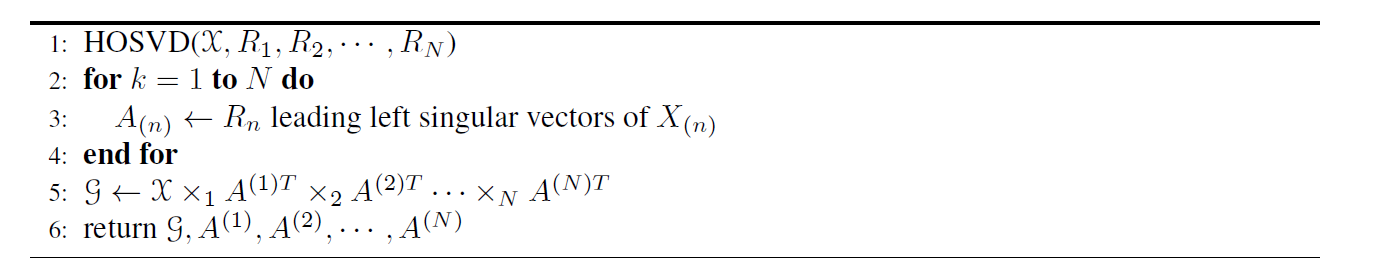


对于一个三阶张量, 由Tucker分解可以得到, , 三个因子矩阵和一个核张量 ，每个模上的因子矩阵称为张量在每个模上的基矩阵或者是主成分，故该分解方法被称为高阶PCA。

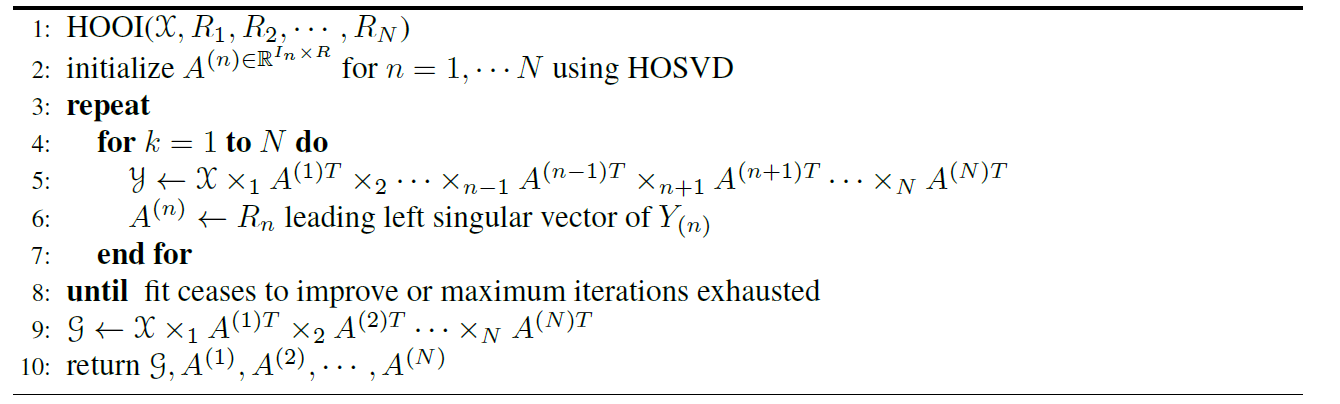
其中核张量衡量了不同维特征的关联程度，而特征矩阵（基矩阵）表明了可理解的不同维特征（用户、产品、情景等标签）和核向量的关联。往往在分解最后还要加上高斯噪声项，即

3）Tucker分解的运算

对于固定的n秩张量，Tucker分解的唯一性不能保证，一般加上一些约束，如分解得到的因子单位正交约束等。比如HOSVD(High Order SVD)求解算法,它通过张量的每一个模上做SVD分解对各个mode上的因子矩阵进行求解,最后计算张量在各个模上的投影之后的张量作为核张量。它的算法过程如下图所示。



虽然利用SVD对每个模做一次Tucker1分解,但是HOSVD 不能保证得到一个较好的近似，但HOSVD的结果可以作为一个其他迭代算法（如HOOI）的很好的初始化。



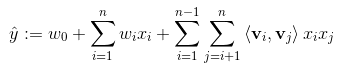
## 8.6.3 基于因子分解机的推荐

1）因子分解机的优势

因子分解机是一种基于矩阵分解的机器学习算法。对于因子分解机FM来说，最大的特点是对于稀疏的数据具有很好的学习能力。现实中稀疏的数据很多。

2）因子分解机模型

对于度为2的因子分解机FM的模型为：



其中，参数，，，表示的是两个大小为的向量和向量的点积：

其中，表示的是系数矩阵的第i维向量，且称为超参数。在因子分解机FM模型中，前面两部分是传统的线性模型，最后一部分将两个互异特征分量之间的相互关系考虑进来。

因子分解机FM也可以推广到高阶的形式，即将更多互异特征分量之间的相互关系考虑进来。

3）推荐模型

如下图所示，该图表示的是一个推荐模型。每一条购买记录都可以使用一个向量表示，上图中x1、x2、x3都是代表了同一个用户，但是看了三个不同的电影，后面还有些其他特征等等。最后每一个记录有个对应的类标签y。但是这样的数据表示很稀疏，但是每一个特征都要考虑而且特征之间很有可能有关联，所以在实际建模中还要考虑特征之间的交互。交互项一般用一个新的权重w和项目之间的乘积表示，以二阶交互为例，需要两个交互项都是非零的情况下才能产生一个非零的交互项。这就导致了数据更加稀疏。为了解决稀疏性，可以借助矩阵分解的思想。矩阵分解会讲一个巨大的稀疏矩阵分解成两个隐矩阵，通常隐矩阵的维度要远小于原来矩阵的维度，因此可以有效的降低稀疏性。

