



中科院计算培训中心

Part 4 大数据推荐及其应用开发

协同过滤与推荐

- 杨文川



- 1) 推荐系统的模型
- 2) 基于内容的推荐
- 3) 协同过滤
- 4) 电影推荐案例



推荐系统是什么

推荐系统是用于预测偏好模型的系统，用于发现新的、称心的事物

- 人们每天都会对喜欢的、不喜欢的、甚至不关心的事情有很多观点。
- 在推荐系统应用当中,存在两类元素
 - 一类称为用户(user),另一类称为项(item)或物品
 - 用户会偏爱某些项,这些偏好信息必须要从数据中梳理出来



两大类推荐算法

- 两大类推荐算法：
 - “user-based”和“item-based”
- user-based
 - 也可由其他人的明显偏好，计算出类似你喜欢的items
- item-based
 - 为发现喜欢的items，可能去寻找有相同兴趣的人。



推荐系统的使用

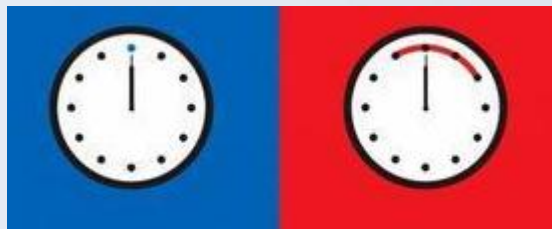
- 推荐系统是将用户对选项的喜好，进行预测的Web应用
- 推荐系统可以实现：
 - (1)基于对用户兴趣的预测结果，为在线读者提供新闻报道；
 - (2)基于顾客过去的购物和/或商品搜索历史，为在线零售商的顾客，推荐可能想要买的商品；



协同过滤 vs 基于内容的推荐

- “协同过滤” -- 基于user和item关系生成推荐
 - 该技术不需要item自己的属性信息。
 - 不需要关心“item”是书、花或人，因为其任何属性都没有作为输入数据。

•



- “基于内容”的推荐，是基于item属性
 - 例如，如果一个朋友推荐一本书给你，因为这是一本图灵出版的书籍，这基于内容推荐的事情。
 - 推荐是基于书籍的属性：出版社。



推荐系统的应用

- 1)产品推荐
 - Product Recommendations
- 2)电影推荐
 - Movie Recommendations
- 3)新闻报道推荐
 - News Recommendations



1)产品推荐

- 产品推荐是一种重要的推荐系统，应用于在线零售商。
 - Amazon等厂商，尽力以建议的方式为每个老用户，展示他们可能想购买的商品的。
 - 这些建议是基于相似用户的购买决定，或者其他一些技术



2) 电影推荐

- Netflix会为其用户推荐他们可能喜欢的电影。
 - 这些推荐基于用户所提供的评级结果，这里的评级结果和效用矩阵样例中的评级十分类似。
- 由于精确预测评级的重要性很高，Netflix提供过一个百万美元大奖，奖励给第一个超过其现有推荐系统10%的算法。
 - 在三年竞赛之后，该奖金颁给一个名叫“Bellkor’s Pragmatic Chaos”的研发团队



3)新闻报道推荐

- 新闻服务机构，会基于读者过去所阅读的文章，识别读者的兴趣：
 - 这里的相似度，可能基于文档中的重要词之间的相似度，或者基于具有类似阅读品味的读者所阅读的文章。
 - 相同的原则也适用于，从数百万个博客中进行博客推荐，或者YouTube上的视频推荐，或者其他定期提供内容的网站上的内容推荐。



推荐系统成为主流

- 推荐系统在Amazon或Netflix上成功应用：
 - 基于浏览和交易历史，页面将产生一个认为对你有吸引力的商品列表。
- 对推荐技术的需求在增加，尤其是开源实现的系统。
 - 随着理解的深入和计算资源越来越廉价，推荐系统变得越来越容易接近和广泛使用



推荐系统的模型及效用矩阵

- 数据本身会表示成一个效用矩阵
 - 该矩阵中每个“用户-项”对，所对应的元素值代表的是当前用户对当前项的喜好程度。
 - 下面介绍基于喜好效用矩阵的推荐系统模型



一个效用矩阵的例子

- 下面矩阵代表用户对电影的评级结果
 - 分(1~5)级，其中5是最高级。空白表示用户目前没有对当前电影评分
- 电影的名字分别是
 - **Harry Potter** 《哈利波特》 **Twilight** 《暮光之城》， **Star Wars** 《星球大战》，
 - 用户分别用大写字母A、B、C和D表示。

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3



相似度度量

- 可以通过和其它朋友共同喜欢某个或某类影片，来确定用户相似
 - 通常是通过“距离”来表示相似
 - 例如：欧几里得距离、皮尔逊相关度、曼哈顿距离、Jaccard系数等等。

	Item 101	Item 102	Item 103
User 1	5.0	3.0	2.5
User 2	2.0	2.5	5.0
User 3	2.5	-	-
User 4	5.0	-	3.0
User 5	4.0	3.0	2.0



欧几里得距离

- 欧几里德距离（Euclidean Distance），最初用于计算欧几里得空间中两个点的距离
- 在二维空间中，就是我们熟悉的两点间的距离， x 、 y 表示两点，维度为 n ：

$$d(x, y) = \sqrt{(\sum_i^n (x_i - y_i)^2)}$$

- 相似度：

$$sim(x, y) = \frac{1}{1 + d(x, y)}$$



皮尔逊相关度

- 皮尔逊相关度（Pearson Correlation Coefficient），用于判断两组数据，与某一直线拟合程度的一种度量，取值在[-1,1]之间。
- 当数据不是很规范的时候（如偏差较大），皮尔逊相关度会给出较好的结果。

$$p(x, y) = \frac{\sum x_i y_i - n \bar{x} \bar{y}}{(n-1) S_x S_y} = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n \sum x_i^2 - (\sum x_i)^2} \sqrt{n \sum y_i^2 - (\sum y_i)^2}}$$



曼哈顿距离

- 曼哈顿距离（Manhattan distance），就是在欧几里得空间的固定直角坐标系上，两点所形成的线段，对轴产生的投影的距离总和。

$$d(x, y) = \sum \|x_i - y_i\|$$



Jaccard系数

- Jaccard系数，也称为Tanimoto系数，是Cosine相似度的扩展，也多用于计算文档数据的相似度。
- 通常应用于x为布尔向量，即各分量只取0或1的时候。
- 此时，表示的是x,y的公共特征的占x，y所占有的特征的比例：

$$T(x, y) = \frac{x \bullet y}{\|x\|^2 + \|y\|^2 - x \bullet y} = \frac{\sum x_i y_i}{\sqrt{\sum x_i^2} + \sqrt{\sum y_i^2} - \sum x_i y_i}$$

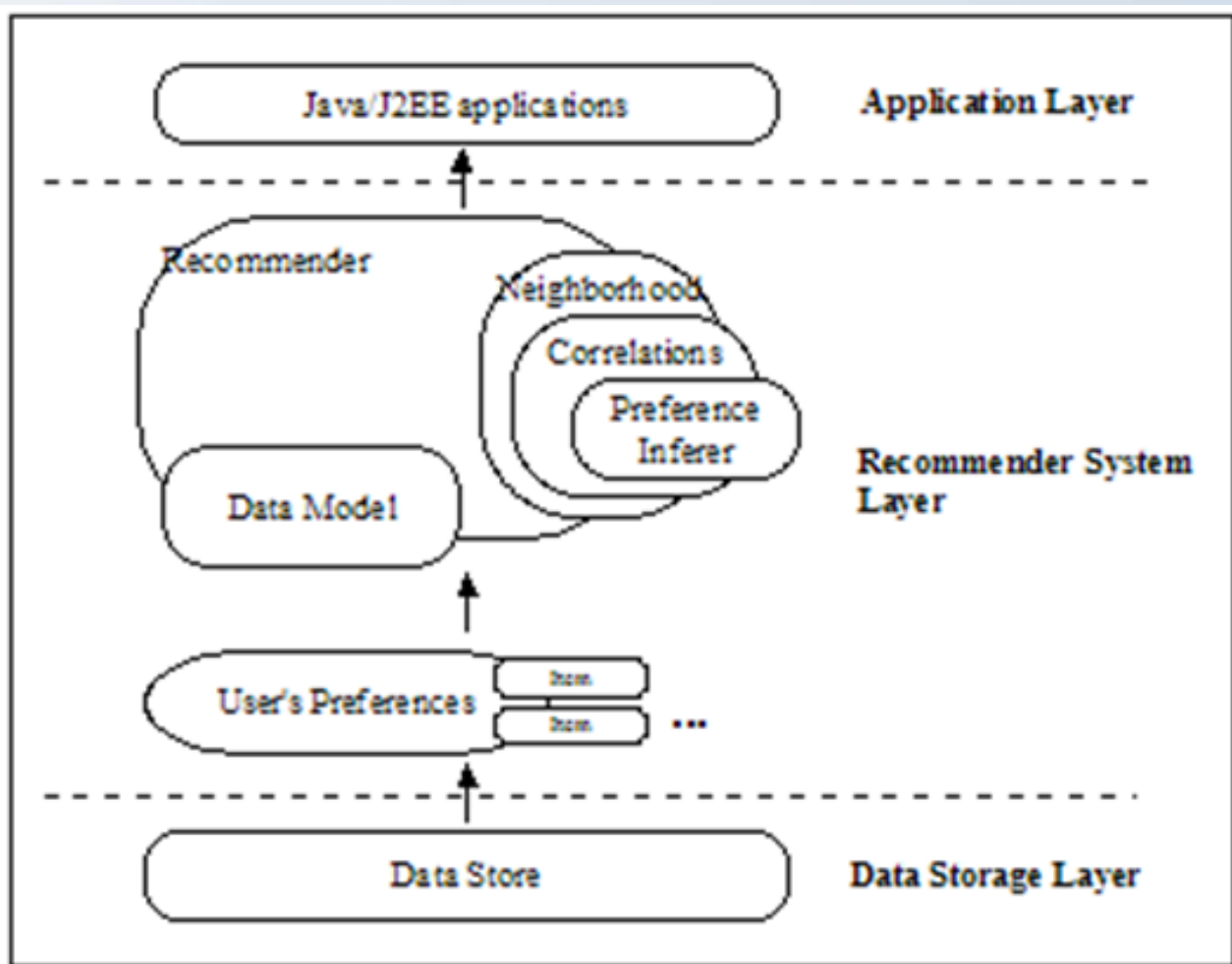


一个推荐系统样例

- Mahout包含了开源的推荐系统
 - 有很多类型，源于传统的user-based和item-based推荐系统。
 - 还包含了基于“slope-one”技术的实现，一个新的、有效的方法。



Mahout推荐实现组件图





图书推荐的例子

1,101,5.0
1,102,3.0
1,103,2.5

2,101,2.0
2,102,2.5
2,103,5.0
2,104,2.0

3,101,2.5
3,104,4.0
3,105,4.5
3,107,5.0

4,101,5.0
4,103,3.0

4,104,4.5
4,106,4.0

5,101,4.0
5,102,3.0

5,103,2.0
5,104,4.0

5,105,3.5
5,106,4.0

- 数据格式：“用户 书 打分”
- 用户1和5，具有相同的兴趣。他们都喜欢101这本书，对102的喜欢弱一些，对103的喜欢更弱
- 用户1和4，具有相同的兴趣，他们都喜欢101和103，没有信息显示用户4喜欢102。
- 用户1和2，兴趣好像正好相反，用户1喜欢101，但用户2讨厌101，用户1喜欢103而用户2正好相反。
- 用户1和3，交集很少，只有101这本书显示了他们的兴趣



分析

1,101,5.0
1,102,3.0
1,103,2.5

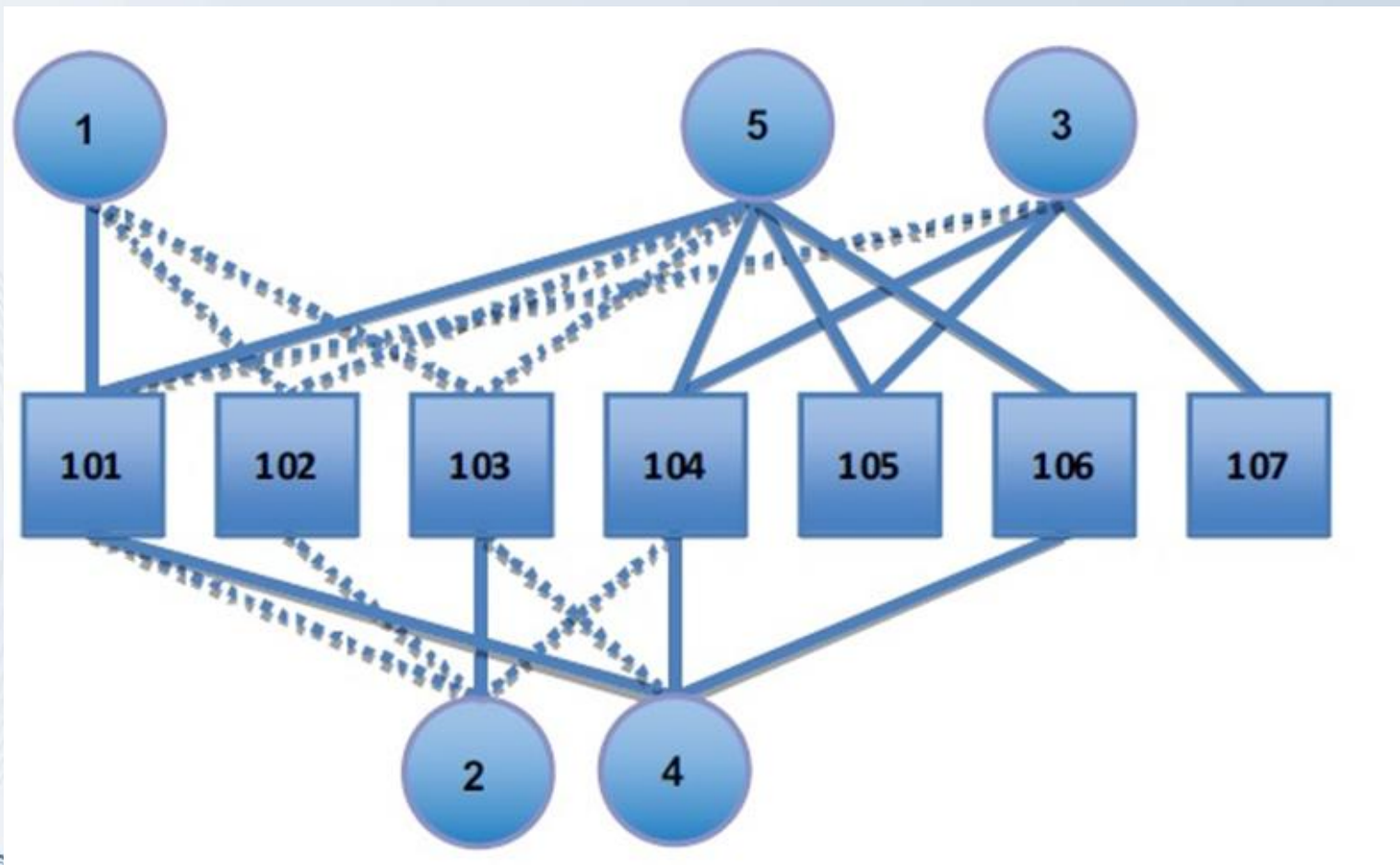
2,101,2.0
2,102,2.5
2,103,5.0
2,104,2.0

3,101,2.5
3,104,4.0
3,105,4.5
3,107,5.0

4,101,5.0
4,103,3.0

4,104,4.5
4,106,4.0

5,101,4.0
5,102,3.0
5,103,2.0
5,104,4.0
5,105,3.5
5,106,4.0





创建推荐系统

1,101,5.0
1,102,3.0
1,103,2.5

2,101,2.0
2,102,2.5
2,103,5.0
2,104,2.0

3,101,2.5
3,104,4.0
3,105,4.5
3,107,5.0

4,101,5.0
4,103,3.0

4,104,4.5
4,106,4.0

5,101,4.0
5,102,3.0
5,103,2.0
5,104,4.0
5,105,3.5
5,106,4.0

- 那么应该给用户1推荐哪本书？不是101, 102或者103，因为用户已购买，推荐系统需要发现新的事物。
- 直觉上，用户4、5与用户1类似，所以推荐一些用户4和5喜欢的书籍，给用户1是不错的。
- 这样使得104、105和106成为可能的推荐。
- 整体上看，104是最有可能的一个推荐，这基于104的4.5和4.0的偏好打分。



分析输出

- 用IDE编译运行，请求一个推荐结果，运行程序的输出是：
 - **RecommendedItem[item:104, value:4.257081]**
-
- 系统将书104推荐给用户1，因为104的打分高一些。系统得出用户1对书104的偏好是4.3，这是最高打分。
 - 输出结果包含了一个用户1喜欢104的评估值
 - 介于用户4和5的打分(4.0和4.5)的一个值



评价推荐系统

- 大部分的推荐系统，通过给item评价打分来实现。
 - 评价推荐系统的一种方式，是评价它的评估偏好值的质量
 - 评价评估偏好和实际偏好的匹配度。





训练集和打分

- 计算评估值和实际值之间的平均距离，分值越低越好。
 - 0.0表示非常好的评估，这说明评估值和实际值根本没有差距

	Item 1	Item 2	Item 3
Actual	3.0	5.0	4.0
Estimate	3.5	2.0	5.0
Difference	0.5	3.0	1.0
Average difference	$= (0.5 + 3.0 + 1.0) / 3 = 1.5$		
Root-mean-square	$= \sqrt{((0.5^2 + 3.0^2 + 1.0^2) / 3)} = 1.8484$		



运行RecommenderEvaluator 并得出 评估结果

- RecommenderEvaluator将数据划分为训练集和测试集，创建一个新的训练DataModel和推荐系统测试，比较评估结果和实际结果。
 - 大部分的操作发生在evaluate()这个方法中。
 - 调用者必须提供一个RecommenderBuilder对象用于从DataModel创建Recommender





电影推荐引擎示例

- 下载数据 “MovieLens 1M - Consists of 1 million ratings from 6000 users on 4000 movies.”
- 这个数据文件夹下有三个文件：
 - movies.dat, ratings.dat和users.dat
 - 这些文件包含来自6040个MovieLens用户在2000年对约3900部电影的1,000,209个匿名评分信息
 - 数据形式如下所示



movies.dat

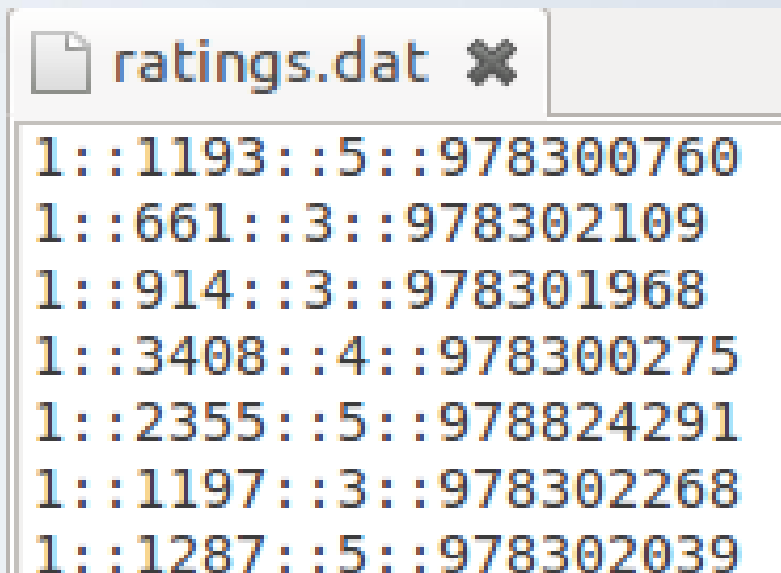
- movies.dat的文件描述是
- 电影编号::电影名::电影类别

```
movies.dat ✕  
1::Toy Story (1995)::Animation|Children's|Comedy  
2::Jumanji (1995)::Adventure|Children's|Fantasy  
3::Grumpier Old Men (1995)::Comedy|Romance  
4::Waiting to Exhale (1995)::Comedy|Drama  
5::Father of the Bride Part II (1995)::Comedy  
6::Heat (1995)::Action|Crime|Thriller  
7::Sabrina (1995)::Comedy|Romance  
8::Tom and Huck (1995)::Adventure|Children's  
9::Sudden Death (1995)::Action  
10::GoldenEye (1995)::Action|Adventure|Thriller
```



ratings.dat

- ratings.dat的文件描述是
- 用户编号::电影编号::电影评分::时间戳

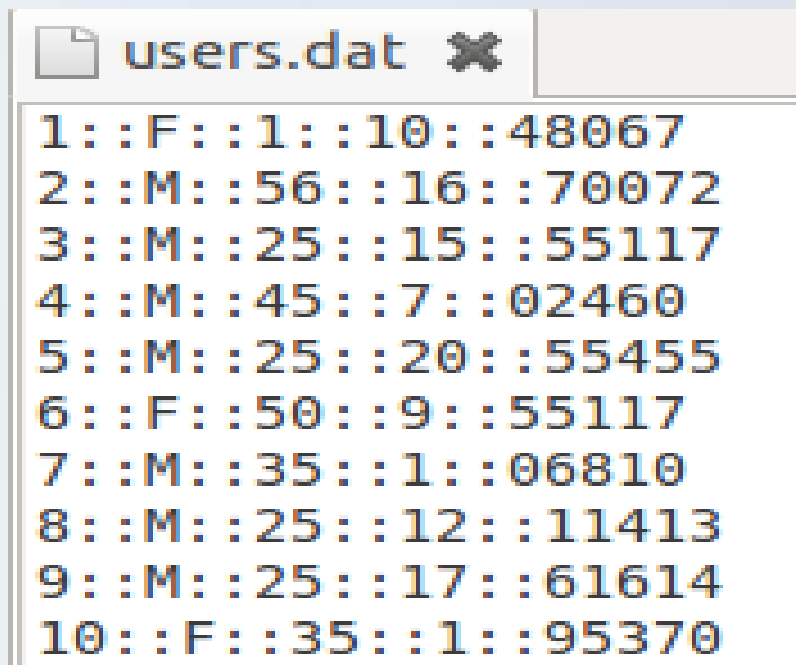


```
ratings.dat ✕  
1::1193::5::978300760  
1::661::3::978302109  
1::914::3::978301968  
1::3408::4::978300275  
1::2355::5::978824291  
1::1197::3::978302268  
1::1287::5::978302039
```



users.dat

- users.dat的文件描述是
- 用户编号::性别::年龄::职业::Zip-code



```
users.dat ✕  
1::F::1::10::48067  
2::M::56::16::70072  
3::M::25::15::55117  
4::M::45::7::02460  
5::M::25::20::55455  
6::F::50::9::55117  
7::M::35::1::06810  
8::M::25::12::11413  
9::M::25::17::61614  
10::F::35::1::95370
```



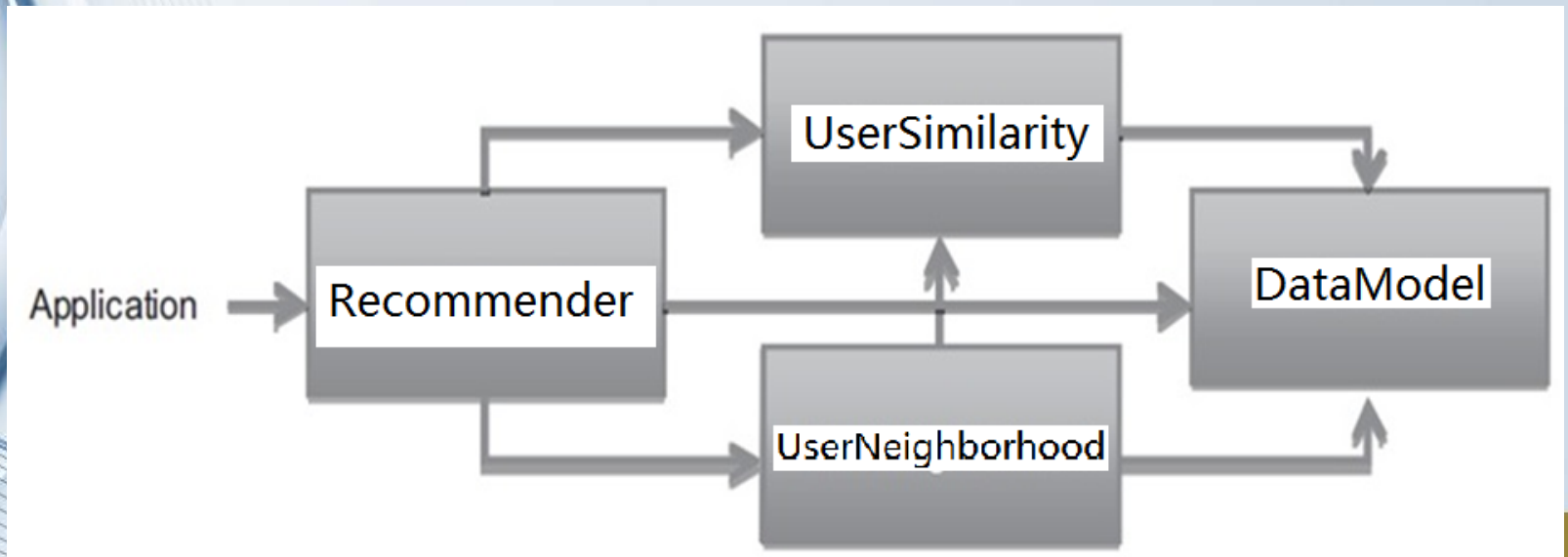
推荐引擎实现

- 本例使用movie.dat文件作为数据源
- 实现了多种方式的推荐引擎：
 - a)基于用户相似度的推荐实现
 - b)基于内容相似度的推荐实现
 - c)其他推荐实现



a) 基于用户相似度的推荐实现

- Mahout 中提供了 DataModel, UserNeighborhood 和 UserSimilarity,
- 构建 GenericUserBasedRecommender, 从而实现基于用户的推荐策略。





相似度的计算

- Mahout 中提供了相似度计算，实现了 UserSimilarity 接口，以实现用户相似度的计算，
- 包括下面这些常用的相似度计算：
 - PearsonCorrelationSimilarity:
 - 基于皮尔逊相关系数计算相似度
 - 表示两个数列对应数字一起增大或减小的可能性。
 - EuclideanDistanceSimilarity:
 - 基于欧几里德距离计算相似度
 - TanimotoCoefficientSimilarity:
 - 基于 Tanimoto 系数计算相似度
 - 也称Jaccard 相似度



找到邻居用户

- 根据建立的相似度计算方法，找到邻居用户
 - 找邻居用户的方法，也包括两种：“固定数量的邻居”和“相似度门槛邻居”计算方法
- Mahout 提供对应的实现：
 - NearestNUserNeighborhood:
 - 对每个用户取固定数量 N 的最近邻居
 - ThresholdUserNeighborhood:
 - 对每个用户基于一定的限制，取落在相似度门限内的所有用户为邻居。



部分用户相似度推荐源码

- `public class MyUserBasedRecommender {`
- `public List<RecommendedItem> userBasedRecommender(long userID,int size) {`
- `// step:1 构建模型 2 计算相似度 3 查找k紧邻 4 构造推荐引擎`
- `List<RecommendedItem> recommendations = null;`
- `try {`
- `DataModel model = MyDataModel.myDataModel();`
- `//构造数据模型， Database-based`
- `UserSimilarity similarity = new`
- `PearsonCorrelationSimilarity(model);`
- `//用PearsonCorrelation 算法计算用户相似度`
- `UserNeighborhood neighborhood = new`
- `NearestNUserNeighborhood(3, similarity, model);`
- `//计算用户的“邻居”，这里将与该用户最近距离为3的用户设置为该用户的“邻居”。`



程序首页

- 提供三个输入：
- 用户id，推荐电影的数目（默认为25），推荐策略

The screenshot shows a web browser window with a dark title bar containing window control icons. The address bar shows the URL "st:8080/MyRecommender/". The main content area has a light beige background. It contains the following elements:

- A label "Enter UserID" followed by a text input field.
- A label "Enter Size" followed by a text input field containing the value "25".
- Three radio buttons for selection: "User Based" (which is selected), "Item Based", and "Slope One".
- A "submit" button located below the radio buttons.



基于用户相似度的推荐结果

- 编号为10的用户,

Your movies				Recommend movies			
Movie Name	Published Year	Movie Type	Score	Movie Name	Published Year	Movie Type	Score
Christmas Vacation	1989	Comedy	5.0	Almost Famous	2000	Comedy, Drama	5.0
Winnie the Pooh and the Blustery Day	1968	Animation, Children's	5.0	Silence of the Lambs, The	1991	Drama, Thriller	5.0
Sixth Sense, The	1999	Thriller	5.0	Rebel Without a Cause	1955	Drama	5.0
Cinderella	1950	Animation, Children's, Musical	5.0	Honey, I Blew Up the Kid	1992	Children's, Comedy, Sci-Fi	5.0
Independence Day (ID4)	1996	Action, Sci-Fi, War	5.0	Saving Private Ryan	1998	Action, Drama, War	5.0
Gone with the Wind	1939	Drama, Romance, War	5.0	Waterboy, The	1998	Comedy	5.0
Grease	1978	Comedy, Musical, Romance	5.0	Girlfight	2000	Drama	5.0
Hero	1992	Comedy, Drama	5.0	Eraserhead	1977	Drama, Horror	5.0
Life Is Beautiful (La Vita ? bella)	1997	Comedy, Drama	5.0	Total Recall	1990	Action, Adventure, Sci-Fi, Thriller	5.0
Out of Africa	1985	Drama, Romance	5.0	One Flew Over the Cuckoo's Nest	1975	Drama	5.0
Shaggy Dog, The	1959	Children's, Comedy	5.0	Requiem for a Dream	2000	Drama	5.0
Wizard of Oz, The	1939	Adventure, Children's, Drama, Musical	5.0	Best in Show	2000	Comedy	5.0
Rain Man	1988	Drama	5.0	Gladiator	2000	Action, Drama	5.0
Trekkies	1997	Documentary	5.0	Billy Madison	1995	Comedy	5.0
Mary Poppins	1964	Children's, Comedy, Musical	5.0	Kingpin	1996	Comedy	5.0
That Thing You Do!	1996	Comedy	5.0				



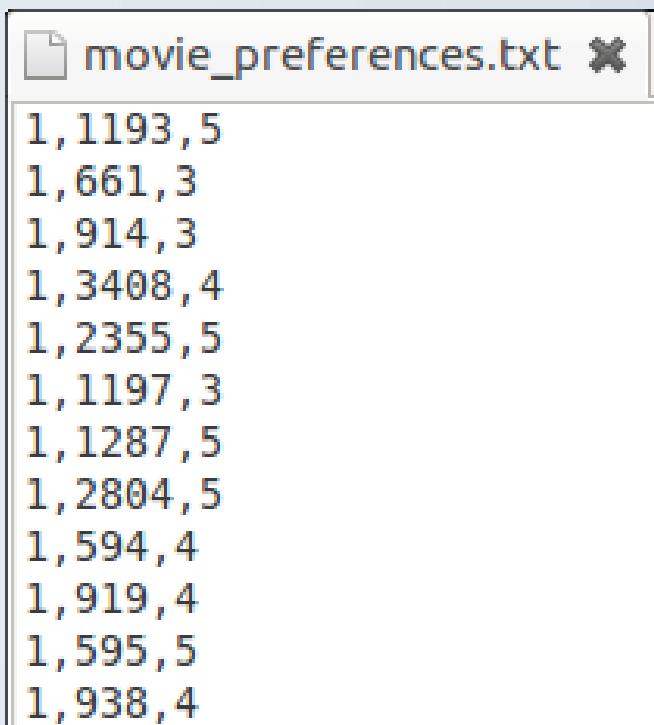
b)基于内容相似度的推荐实现

- 基于内容相似度的推荐引擎类似，较为简单。
- `public class MyItemBasedRecommender {`
- `public List<RecommendedItem>`
- `myItemBasedRecommender(long userID,int size){`
- `List<RecommendedItem> recommendations = null;`
- `try {`
- `DataModel model = new FileDataModel(new`
- `File("/home/lab466/movie_preferences.txt"));`
- `//构造数据模型， File-based`



数据文件格式

- 在推荐引擎中，使用的是File-based Datamodel
- 如下图



```
movie_preferences.txt ✕  
1,1193,5  
1,661,3  
1,914,3  
1,3408,4  
1,2355,5  
1,1197,3  
1,1287,5  
1,2804,5  
1,594,4  
1,919,4  
1,595,5  
1,938,4
```




基于内容相似度的推荐结果

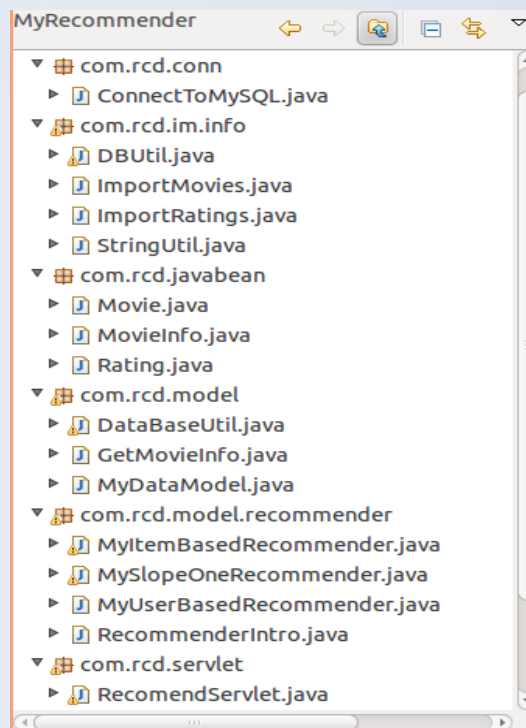
- 编号为10的用户

Your movies				Recommend movies			
Movie Name	Published Year	Movie Type	Score	Movie Name	Published Year	Movie Type	Score
Christmas Vacation	1989	Comedy	5.0	Loser	1991	Comedy	5.0
Winnie the Pooh and the Blustery Day	1968	Animation, Children's	5.0	Modulations	1998	Documentary	5.0
Sixth Sense, The	1999	Thriller	5.0	Dangerous Game	1993	Drama	4.769230842590332
Cinderella	1950	Animation, Children's, Musical	5.0	Crude Oasis, The	1995	Romance	4.75
Independence Day (ID4)	1996	Action, Sci-Fi, War	5.0	I'll Never Forget What's 'is Name	1967	Comedy, Drama	4.714285850524902
Gone with the Wind	1939	Drama, Romance, War	5.0	Show, The	1995	Documentary	4.666666507720947
Grease	1978	Comedy, Musical, Romance	5.0	Mamma Roma	1962	Drama	4.625
Hero	1992	Comedy, Drama	5.0	First Love, Last Rites	1997	Drama, Romance	4.56131649017334
Life Is Beautiful (La Vita ? bella)	1997	Comedy, Drama	5.0	Talk of Angels	1998	Drama	4.545454502105713
Out of Africa	1985	Drama, Romance	5.0	Callej?n de los milagros, El	1995	Drama	4.538461685180664
Shaggy Dog, The	1959	Children's, Comedy	5.0	Gay Deceivers, The	1969	Comedy	4.5
Wizard of Oz, The	1939	Adventure, Children's, Drama, Musical	5.0	Second Jungle Book: Mowgli & Baloo, The	1997	Adventure, Children's	4.488247394561768
Rain Man	1988	Drama	5.0	City of the Living Dead (Paura nella citt? dei morti viventi)	1980	Horror	4.480743885040283
Trekkies	1997	Documentary	5.0	Just the Ticket	1999	Comedy, Romance	4.479945659637451
Mary Poppins	1964	Children's, Comedy, Musical	5.0	It's in the Water	1998	Comedy	4.451860427856445
That Thing You Do!	1996	Comedy	5.0				



程序演示

- 项目开发环境是Ubuntu，IDE是MyEclipse，文件目录如下





NetFlix竞赛

- NetFlix竞赛对推荐系统的研究有重大促进
 - NetFlix为第一个设计出比他们当前推荐系统CineMatch，精度高10%的系统的人或团队，设立了100万美元的大奖。
 - 在3年多的工作之后，该奖金于2009年9月颁发



- NetFlix竞赛中包含一个公开数据集
 - 内容包括50万用户对17000部电影的评分数据，该数据来自一个更大的数据集
 - 提出的算法要在一个非公开测试集中经受预测能力的测试。
 - 公开数据集中每个(用户，电影)对的数据包括1到5之间的一个评级以及评级的日期。



- 算法的效果通过RMSE指标来度量。
 - CineMatch的RMSE大概是0.95，预测评级和真实评级相差差不多整整一颗星。
 - 为了赢取奖金，提交算法的RMSE必须比CineMatch，至少快10%。



利用MLlib进行推荐

- 协同过滤(Collaborative Filtering, 简称CF)
- MLlib中的协同过滤，常应用于推荐系统。
 - 利用某兴趣相投、拥有共同经验之群体的喜好，来推荐使用者感兴趣的资讯，补充用户-商品(User-Item)效用矩阵中所缺失的部分
 - MLlib当前支持基于模型的协同过滤，其中用户和商品通过一小组隐语义因子进行表达，并且这些因子也用于预测缺失的元素。
 - 为此，MLlib实现了交替最小二乘法(ALS)来学习这些隐性语义因子。
- 目前可用的Spark协同过滤的算法很多，例如 ALS



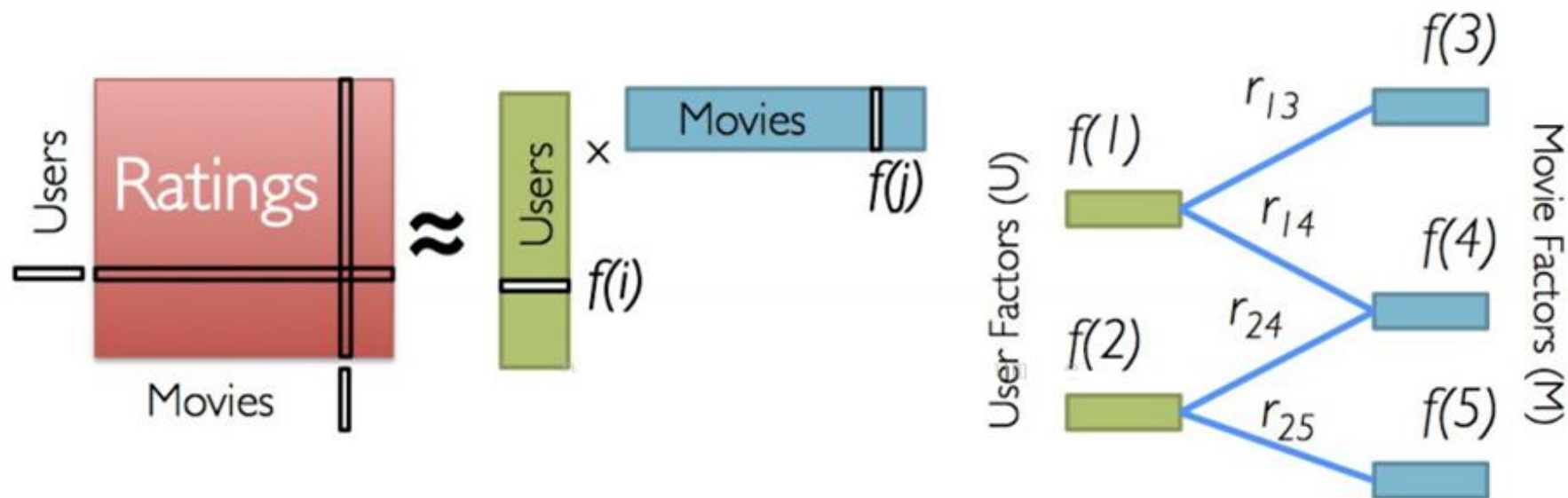
协同过滤例子

- 在下面的例子中，导入的训练集中，数据每一行由一个用户、一个商品和相应的评分，
- 下面使用默认的ALS.train()方法
 - 例子SparkMovieLensALS
- 通过计算预测出的评分的均方差，来评估这个推荐模型。



ALS

Low-Rank Matrix Factorization:



Iterate:

$$f[i] = \arg \min_{w \in \mathbb{R}^d} \sum_{j \in \text{Nbrs}(i)} (r_{ij} - w^T f[j])^2 + \lambda ||w||_2^2$$



实现参数解释

- 在MLlib中的实现有如下的参数：
 - numBlocks 是用于并行化计算的分块个数（设置为 - 1 为自动配置）。
 - rank 是模型中隐性因子的个数。
 - iterations 是迭代的次数。
 - lambda 是 ALS 的正则化参数。
 - implicitPrefs 决定了是用显性反馈 ALS 的版本还是用适用隐性反馈数据集的版本。
 - alpha 是一个针对于隐性反馈 ALS 版本的参数，这个参数决定了偏好行为强度的基准



中科院计算培训中心

程序实例

- 参见Scala代码中的MovieLensALS源码
- 注意： Rating是ML内置的数据类型



运行结果

Training: 602252, validation: 198919, test: 199049

RMSE (validation) = 0.9552578279170443 for the model trained with rank = 8, lambda = 0.1, a
RMSE (validation) = 0.9510527170784763 for the model trained with rank = 8, lambda = 0.1, a
RMSE (validation) = 0.8858844389573322 for the model trained with rank = 8, lambda = 10.0,
RMSE (validation) = 0.8826441136754684 for the model trained with rank = 8, lambda = 10.0,
RMSE (validation) = 1.0201507007524229 for the model trained with rank = 12, lambda = 0.1,
RMSE (validation) = 1.0247487950144085 for the model trained with rank = 12, lambda = 0.1,
RMSE (validation) = 0.8869216672500277 for the model trained with rank = 12, lambda = 10.0,
RMSE (validation) = 0.8837305066623388 for the model trained with rank = 12, lambda = 10.0,
The best model was trained with rank = 8 and lambda = 10.0, and numIter = 20, and its RMSE
0242288.

The best model improves the baseline by 20.80%.

Movies recommended for you:

- 1: Raiders of the Lost Ark (1981)
- 2: Matrix, The (1999)
- 3: Star Wars: Episode IV - A New Hope (1977)
- 4: Terminator 2: Judgment Day (1991)
- 5: Sixth Sense, The (1999)
- 6: Gladiator (2000)
- 7: Braveheart (1995)
- 8: Star Wars: Episode V - The Empire Strikes Back (1980)
- 9: Die Hard (1988)
- 10: Terminator, The (1984)



中科院计算培训中心

谢 谢