https://www.cnblogs.com/leboop/p/9453756.html

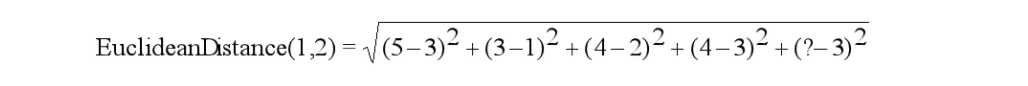
协同过滤基本思想：如果用户在过去有相同的偏好，那么他们在未来也会有相似的偏好。

协同过滤的基本思路：以用户为中心的话, 每个人都是由一个物品集表示, 相同爱好越多的人将被分为一个圈子，这个圈子里面和你都比较亲近，当然也有亲近的差异。通过为每个用户推荐圈子里面最受欢迎的物品(通过亲近程度加权而来)。

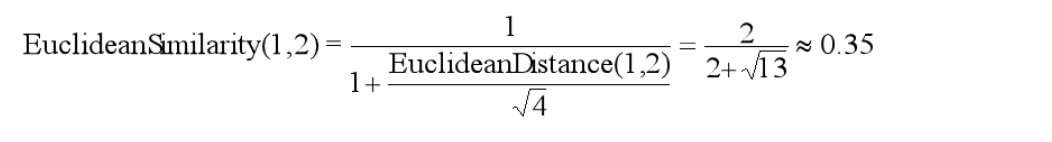
一、基于用户的协同过滤（你可能会喜欢 ）

1、用户相似度度量 GenericUserBasedRecommender

（1）、欧几里得空间距离

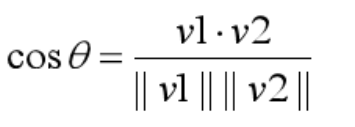


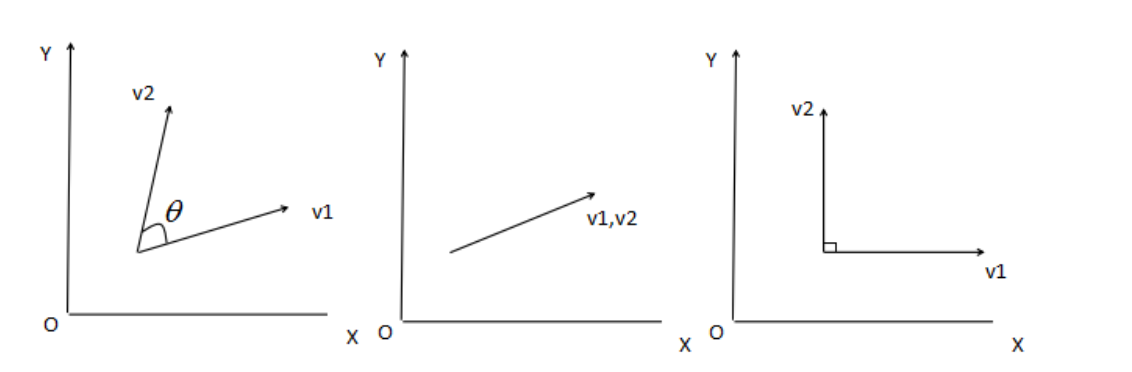
欧几里得空间相似度



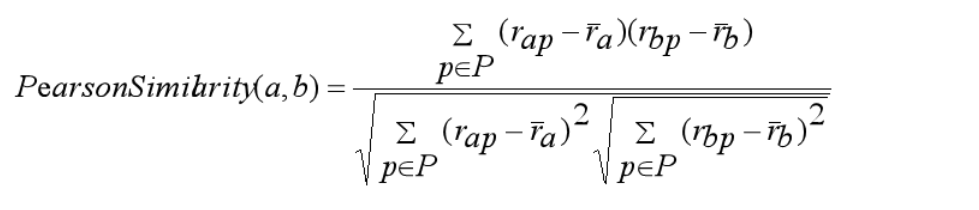
结果越高，相似度越高

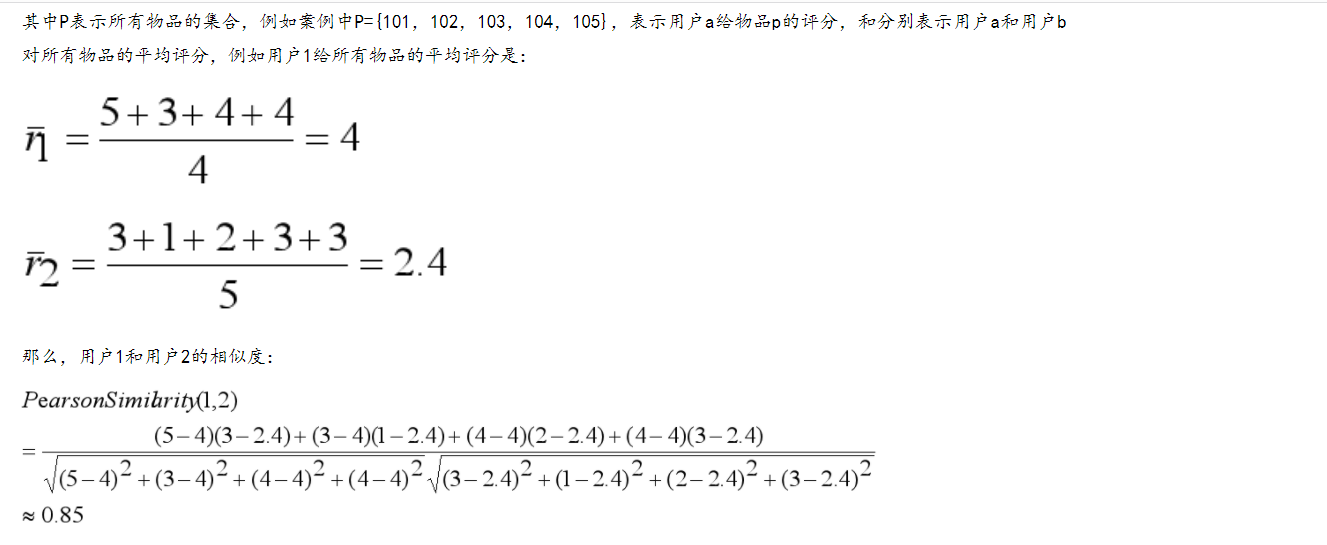
（2）、余弦夹角





（3）、皮尔森相关系数





2、用户的k-最近邻

根据建立的相似度计算方法，找到邻居用户。这里找邻居用户的方法根据前面我们介绍的，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：

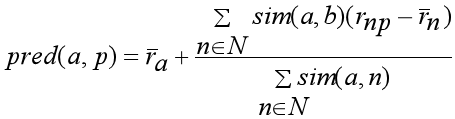
•NearestNUserNeighborhood：对每个用户取固定数量 N 的最近邻居

•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。

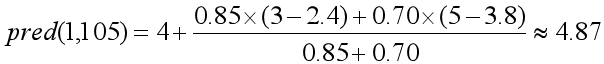
    所谓某个用户的k-最近邻是指与该用户最相似的k个用户（不包括该用户本身）。例如我们前面形成了用户1的*4-最近邻*={用户3，用户2，用户4，用户5}。如果我们只选择两个最相似用户，那么就构成了用户1的*2-最近邻*={用户3，用户2}，从*2-最近邻*来看，前面的三种度量方式，结果是一致的。

3、预测得分

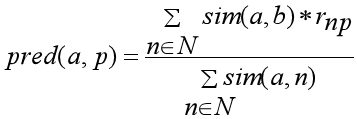
如何预测用户1给物品105的评分？这就关系到该重视哪些近邻的评分，如何重视？例如通过皮尔森相关系数，计算出的用户1与其他用户的相似度：用户1和用户2、用户3相似度分别为0.85和0.70，相关性最大，与用户4相似度值0，可以认为无关，与用户五的相关系数值是-0.79，是个负值，可以认为两个用户可能偏好截然相反。所以我们应当选择用户2和用户3作为用户1的2-最近邻来评分。下面公式考虑了用户a的N近邻与用户a平均评分的偏差，预测用户a对物品p的评分：



其中sim(a,b)表示用户a和用户b的相似度。所以用户1给物品105的预测评分是：



当然，Apache Mahout中并不是这样实现的，它未考虑平均评分，而是采用了如下简化的预测公式：



计算出的预测评分为



4、推荐

根据预测的评分设置阈值，考虑是否推荐

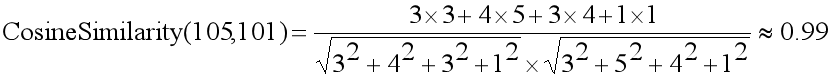


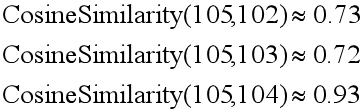
二、基于物品的协同过滤 GenericItemBasedRecommender

1、基本思想

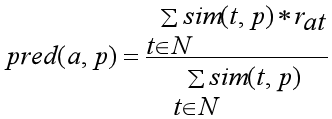
    尽管基于用户的协同过滤的方法已经成功应用在了不同领域，但在有着数以百万计甚至上亿用户和物品的大型电子商务网站（例如亚马逊Amazon）还是会存在很多严峻挑战。这种方法很难做到实时推荐。下面谈谈与基于用户协同过滤类似的另外一种推荐方法——基于物品的协同过滤推荐。

基于物品的协同过滤推荐主要思想是利用物品间相似度，而不是用户间相似度来计算预测值。我们看用户-物品-评分矩阵的某一列，得到物品101对应的向量v1=（5，3，4，3，1）和物品105对应的向量v5=（？，3，5，4，1），舍弃向量的第一个分量，通过余弦相似度计算出他们的相似度如下：

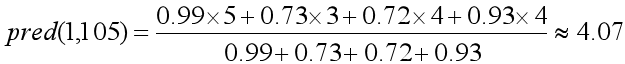




我们通过计算用户1对所有与物品105相似物品的加权评分总和来预测用户1对物品105的评分，公式如下：



N表示物品p的k-最近邻，同用户的k-最近邻类似，实际中需要选择k值，sim(t,p)表示物品t与物品p的相似度，表示用户a给物品t的评分。Apache Mahout实现时，N采用与物品p相似的所有物品。所以，用户1对物品105的预测评分为：



七、协同过滤推荐基本步骤

1、基于用户的协同过滤

（1）采集用户与物品之间的关联数据，如浏览、购买或交易记录，形成初始数据；

（2）分析用户与物品的关联数据形成用户-物品-评分数据；

（3）依据用户-物品-评分数据计算所有用户间的相似度；

（4）选择与当前用户最相似的k个用户，也就是用户k-最近邻。

（5）将这k个用户加权评分最高且当前用户没有浏览过的n个物品推荐给当前用户。

2、基于物品的协同过滤

（1）采集用户与物品之间的关联数据，如浏览、购买或交易记录，形成初始数据；

（2）分析用户与物品的关联数据形成用户-物品-评分数据；

（3）依据用户-物品-评分数据计算所有物品间的相似度；

（4）对当前用户没浏览过的某个物品，选择最相似的k个物品（k-最近邻）

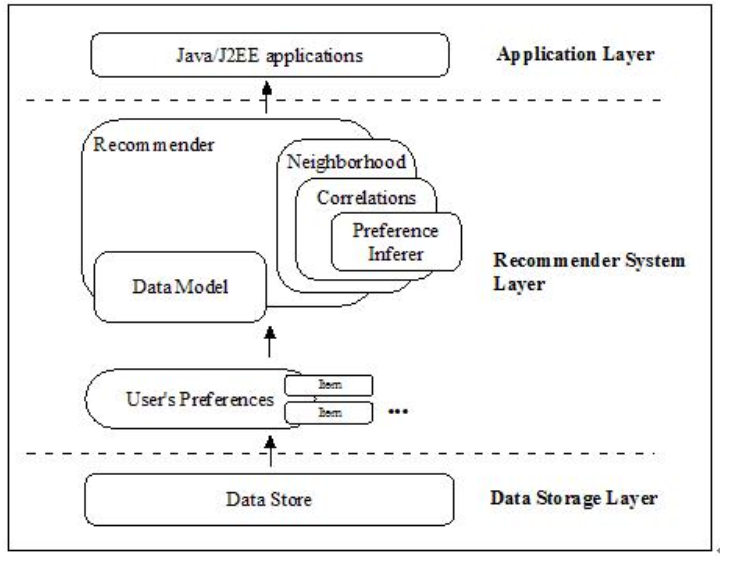
（5）基于这k个物品评分预测当前物品评分；

（6）将评分最高的n个物品推荐给当前用户。

**Mahout下个性化推荐引擎Taste**

**Taste是Apache Mahout提供的一个个性化推荐引擎的搞笑实现，基于java，可扩张性强**

**核心组件**



DataModel：DataModel是用户喜好信息的抽象接口，它的具体实现支持从指定类型的数据源抽取用户喜好信息。在Mahout0.5中，Taste 提供 JDBCDataModel 和 FileDataModel两种类的实现，分别支持从数据库和文件文件系统中读取用户的喜好信息。对于数据库的读取支持，在Mahout 0.5中只提供了对MySQL和PostgreSQL的支持，如果数据存储在其他数据库，或者是把数据导入到这两个数据库中，或者是自行编程实现相应的类。

  UserSimilarit和ItemSimilarity：前者用于定义两个用户间的相似度，后者用于定义两个项目之间的相似度

UserNeighborhood：在基于用户的推荐方法中，推荐的内容是基于找到与当前用户喜好相似的“邻居用户”的方式产生的，该组件就是用来定义与目标用户相邻的“邻居用户”。所以，该组件只有在基于用户的推荐算法中才会被使用。

 Recommender：Recommender是推荐引擎的抽象接口，Taste 中的核心组件。利用该组件就可以为指定用户生成项目推荐列表。

Mahout初探

Preference类: 表示每个用户或者物品的属性, 如{1: {2: 1.0, 3: 2.0}} 表示用户2和3分别对1进行了评价, 评分分别为1.0和2.0

    DataModel类: 表示所有的数据集合, 即Preference的聚合. DataModel可以是代码输入或者HTTP参数获取, 也可以从文件或者数据库中读取.

    Similarity类: 表示所有Preference两者之间的相似度, 所有的用户或者物品都通过Similarty建立起了联系, 所以这个是非常重要的, Similarity的选择和数据的类型有很大关系. Neighberhood记录每个用户或者物品相似的物品集, 方便后期调用.

    Recommender类: 表示推荐算法, 整个系统的核心。 如何使用前面的Preference, DataModel, Similarity以及Neigherhood都将结合在Recommender来来使用。

推荐系统的难点在于各种参数、算法的选择，以及推荐系统整体架构的测试

推荐引擎：

推荐引擎利用特殊的信息过滤（IF，Information Filtering）技术，将不同的内容（例如电影、音乐、书籍、新闻、图片、网页等）推荐给可能感兴趣的用户。通常情况下，推荐引擎的实现是通过将用户的个人喜好与特定的参考特征进行比较，并试图预测用户对一些未评分项目的喜好程度。参考特征的选取可能是从项目本身的信息中提取的，或是基于用户所在的社会或社团环境。

根据如何抽取参考特征，我们可以将推荐引擎分为以下四大类：

• 基于内容的推荐引擎：它将计算得到并推荐给用户一些与该用户已选择过的项目相似的内容。例如，当你在网上购书时，你总是购买与历史相关的书籍，那么基于内容的推荐引擎就会给你推荐一些热门的历史方面的书籍。

• 基于协同过滤的推荐引擎：它将推荐给用户一些与该用户品味相似的其他用户喜欢的内容。例如，当你在网上买衣服时，基于协同过滤的推荐引擎会根据你的历史购买记录或是浏览记录，分析出你的穿衣品位，并找到与你品味相似的一些用户，将他们浏览和购买的衣服推荐给你。

• 基于关联规则的推荐引擎：它将推荐给用户一些采用关联规则发现算法计算出的内容。关联规则的发现算法有很多，如 Apriori、AprioriTid、DHP、FP-tree 等。

• 混合推荐引擎：结合以上各种，得到一个更加全面的推荐效果。

c) 基于Slop One的推荐引擎

基于用户和基于内容是最常用最容易理解的两种推荐策略，但在大数据量时，它们的计算量会很大，从而导致推荐效率较差。因此 Mahout 还提供了一种更加轻量级的 CF 推荐策略：Slope One。

Slope One 是有 Daniel Lemire 和 Anna Maclachlan 在 2005 年提出的一种对基于评分的协同过滤推荐引擎的改进方法，下面简单介绍一下它的基本思想。

假设系统对于物品 A，物品 B 和物品 C 的平均评分分别是 3，4 和 4。基于 Slope One 的方法会得到以下规律：

•用户对物品 B 的评分 = 用户对物品 A 的评分 + 1

•用户对物品 B 的评分 = 用户对物品 C 的评分

基于以上的规律，我们可以对用户 A 和用户 B 的打分进行预测：

•对用户 A，他给物品 A 打分 4，那么我们可以推测他对物品 B 的评分是 5，对物品 C 的打分也是 5。

•对用户 B，他给物品 A 打分 2，给物品 C 打分 4，根据第一条规律，我们可以推断他对物品 B 的评分是 3；而根据第二条规律，推断出评分是 4。当出现冲突时，我们可以对各种规则得到的推断进行就平均，所以给出的推断是 3.5。

这就是 Slope One 推荐的基本原理，它将用户的评分之间的关系看作简单的线性关系：

Y = mX + b;

当 m = 1 时就是 Slope One，也就是我们刚刚展示的例子。

public class MySlopeOneRecommender {

public List<RecommendedItem> mySlopeOneRecommender(long userID,int size){

List<RecommendedItem> recommendations = null;

try {

DataModel model = new FileDataModel(new File("/home/huhui/movie\_preferences.txt"));//构造数据模型

Recommender recommender = new CachingRecommender(new SlopeOneRecommender(model));//构造推荐引擎

recommendations = recommender.recommend(userID, size);//得到推荐结果

} catch (Exception e) {

// TODO: handle exception

e.printStackTrace();

}

return recommendations;

}