协同过滤学习笔记

李向阳 d1142845997@gmail.com

目录

1 引入

这一次, 我们来谈谈推荐 (Recommendation) 算法.

我们知道, 在机器学习中, 大多数算法是解决分类、聚类问题的, 好像极少有机器学习的相关书籍讲到过推荐算法. 不过, 在很多数据挖掘相关的书籍中, 倒是很多提到过推荐算法, 比如《集体智慧编程》中的协同过滤算法等. 这也是我分成数据挖掘和机器学习两个系列的原因. 对于大的框架, 我个人的理解是数据分析、数据挖掘、机器学习、深度学习, 这并不是代表它们的优劣等级, 只是觉得越往后用到的数学知识越多, 比如机器学习和深度学习的很多算法中用到了大量的微积分、矩阵求导、优化算法之类的东西, 但是数据挖掘和数据分析的很多算法中就用的相对少些, 毕竟用 Excel 也能做一些基本的数据分析, 比如数据探索性分析、简单回归分析等.

回到推荐算法, 之前一直听说亚马逊和网易云音乐的算法非常牛逼, 所以知道了好的推荐算法的重要性. 常用的推荐算法有很多, 比如协同过滤算法、矩阵分解算法等. 本次主要介绍协同过滤 (Collaborative Filtering) 算法.

协同过滤算法有很多, 主要可分为两大类: 基于内存 (Memory-Based) 的协同过滤和基于模型 (Model-Based) 的协同过滤, 其中基于内存的协同过滤又可分为基于用户 (User-Based) 的协同过滤和基于物品 (Item-Based) 的协同过滤, 我们主要讲述这两种方法.

2 推荐算法模型框架

以推荐电影 (或音乐) 为例, 假设有 n 部电影作品 (Item), 不妨记为 $i_1, i_2, \cdots, i_n$(用 i 表示物品更有语义, 但我们一般又用 i 表示计数指标, 所以看的时候注意记号的含义即可), 一个人可能看过其中的几部, 也可能全都看过, 他对这些电影的喜好程度肯定是不一样的, 这可以用评分来表现.

现在假设样本集中有 m 个用户 (User), 不妨记为 $u_1, u_2, \cdots, u_m$, 设用户 u_j 对物品 i_k 的评分为 r_{jk}, 那么我们便可以得到一个评分矩阵 $R = (r_{jk})_{m\times n}$, 比如下表1

该表是《集体智慧编程》里电影推荐的数据, 原书里的数据是以 Python 中的字典形式给出的, 我把它转化成了通用的评分矩阵. 其中 u_a 表示书中的 Toby, 而 u_1 到 u_6 表示 Lisa Rose, Gene Seymour, Michael Phillips, Claudia Puig, Mick LaSalle 和 Jack Mattews, i_1 到 i_6 分别表示 Lady in the Water, Snakes on a Plane, Just My Luck, Superman Returns, You Me and Dupree 和 The Night Listener.

推荐算法的任务就是, 给定一个新人 u_a(当然也不一定是新人, 可以是

表 1: 评分矩阵的例子

用户 \ 物品	i_1	i_2	i_3	i_4	i_5	i_6
u_1	2.5	3.5	3.0	3.5	2.5	3.0
u_2	3.0	3.5	1.5	5.0	3.5	3.0
u_3	2.5	3.0	?	3.5	?	4.0
u_4	?	3.5	3.0	4.0	2.5	4.5
u_5	3.0	4.0	2.0	3.0	2.0	3.0
u_6	3.0	4.0	?	5.0	3.5	3.0
u_a	?	4.5	?	4.0	1.0	?

样本中的人), 我们知道它对一些电影的评分, 该如何向他推荐这 8 部电影中他没看过的一些电影呢?

这就是推荐算法模型的大致框架, 当然, 如果评分矩阵是一个完整的矩阵最好, 不完整的时候直接当做缺失值就可以了 (借用 R 语言的符号, 用 NA 表示缺失值).

3 相似性度量

如何度量两个样本之间的相似性呢?

比如在评分矩阵中, 用户 u_i 对所有电影的评分构成一个向量, 用户 u_j 对所有电影的评分也构成一个向量, 所以其实我们计算的是两个向量之间的相似度.

度量相似度, 我们可以用距离, 比如欧氏距离、余弦距离等. 对于两个向量 $\boldsymbol{x} = (x_1, x_2, \cdots, x_n)^T$ 和 $\boldsymbol{y} = (y_1, y_2, \cdots, y_n)^T$ 来说, 计算它们之间的距离有很多方法, 下面我们稍微总结一下.

3.1 欧氏距离

最简单的莫过于欧氏距离了, 也就是 L_2 距离, 如下

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{(\boldsymbol{x} - \boldsymbol{y})^T (\boldsymbol{x} - \boldsymbol{y})} = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

3.2 余弦距离

在几何中, 夹角余弦可以衡量两个向量方向之间的差异, 所以数据挖掘中有时用它来表示两个样本向量之间的差异, 如下

$$\cos\theta = \cos(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2}}$$

夹角余弦取值范围为 $[-1, 1]$. 夹角余弦越大表示两个向量的夹角越小, 夹角余弦越小表示两向量的夹角越大.

3.3 相关系数

回顾一下, 在统计学中, 假设有两个变量 X 和 Y, 则总体的 Pearson 相关系数为

$$\rho_{X,Y} = \frac{Cov(X,Y)}{\sqrt{DX}\sqrt{DY}} = \frac{E(XY) - E(X)E(Y)}{\sqrt{E(X^2) - (EX)^2}\sqrt{E(Y^2) - (EY)^2}}$$

设样本观测数据为

$$\begin{pmatrix} X_1 & X_2 & \cdots & X_n \\ Y_1 & Y_2 & \cdots & Y_n \end{pmatrix}$$

则样本相关系数为

$$r = \frac{S_{XY}}{\sqrt{S_{XX}}\sqrt{S_{YY}}} = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}$$

进行适当化简可得

$$r = \frac{\sum_{i=1}^{n} X_i Y_i - \frac{\sum_{i=1}^{n} X_i \sum_{i=1}^{n} Y_i}{n}}{\sqrt{\sum_{i=1}^{n} X_i^2 - \frac{\left(\sum_{i=1}^{n} X_i\right)^2}{n}}\sqrt{\sum_{i=1}^{n} Y_i^2 - \frac{\left(\sum_{i=1}^{n} Y_i\right)^2}{n}}}$$

注意, 以上公式是针对两个变量 X 和 Y 计算的相关系数, 比如身高和体重, 做了 n 次观测, 得到了 n 个人的样本 $(X_1, Y_1), (X_2, Y_2), \cdots, (X_n, Y_n)$, 我们的样本是一个二元总体, 所得到的相关系数是身高和体重的相关系数, 相关系数越大, 则相关性越显著, 比如身高越高、体重越重.

我们这里是两个样本 (两个人), 一个用户样本是 $(x_1, x_2, \cdots, x_n)^T$, 另一个用户样本是 $(y_1, y_2, \cdots, y_n)^T$, 那么如何应用上面的相关系数呢?

事实上, 意义其实完全可以迁移过来, 用户一喜欢电影 i_k, 那么他对电影 i_k 的打分 x_k 就越高, 如果用户二与用户一的比较相似, 即兴趣差不多, 那么他对电影的 i_k 的打分 y_k 也就越高, 也就是说 x_k 越大, y_k 就越大, 相当于每一部电影都是对这两个用户的观测, 即电影 i_k 的观测是 $(x_k, y_k)^T$, 总共有 n 部电影, 相当于进行了 n 次观测 (考察了 n 次), 那用这 n 次观测计算出的相关系数不正是对这两个用户兴趣相似性大小的衡量吗?

所以, 如下公式 (其实就是把上面公式中的大写改为小写) 计算的相关系数也可以衡量相似性

$$
\begin{aligned}
r &= \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \\
&= \frac{\sum_{i=1}^{n} x_i y_i - \frac{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n}}{\sqrt{\sum_{i=1}^{n} x_i^2 - \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n}}\sqrt{\sum_{i=1}^{n} y_i^2 - \frac{\left(\sum_{i=1}^{n} y_i\right)^2}{n}}}
\end{aligned}
$$

其实以上默认都是计算用户的相似度, 主要用于基于用户的协同过滤. 在基于物品的协同过滤中, 我们需要计算两个物品的相似度, 那么可否利用相关系数呢?

当然也是可以的, 比如我们想计算两部电影的相似度, 如果这两部电影比较相似, 那么同一个人对它们的评分应该是相近的 (因为要么都喜欢, 要么都不喜欢). 设用户对电影一的评分为 x, 他对电影二的评分为 y, 若两部电影相似, 那么 x 越大, y 应该也越大, 也就是说同一个人对这两部电影的评分相当于做了一次观测 (x, y), 现在有 m 个人, 相当于进行了 m 次观测, 即 $(x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m)$, 那用这 m 组数据计算出的相关系数不也正是对这两部电影相似度大小的衡量吗?

而且, 这时也和统计学中的二元总体很像 (两部电影相当于身高和体重的关系), 也就是说样本是 (x, y), 然后进行了 m 次观测, 得到了 m 个样本,

计算相关系数如下

$$
\begin{aligned}
r &= \frac{\sum_{i=1}^{m}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{m}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{m}(y_i-\bar{y})^2}} \\
&= \frac{\sum_{i=1}^{m}x_iy_i - \frac{\sum_{i=1}^{m}x_i\sum_{i=1}^{m}y_i}{m}}{\sqrt{\sum_{i=1}^{m}x_i^2-\frac{\left(\sum_{i=1}^{m}x_i\right)^2}{m}}\sqrt{\sum_{i=1}^{m}y_i^2-\frac{\left(\sum_{i=1}^{m}y_i\right)^2}{m}}}
\end{aligned}
$$

当然, 实际计算时我们可以把评分矩阵取个转置, 把上述公式理解成计算两个向量 $\boldsymbol{x}=(x_1,x_2,\cdots,x_m)^T$ 和 $\boldsymbol{y}=(y_1,y_2,\cdots,y_m)^T$ 的相关性, 这两个向量分别就是两部电影的评分向量, 即以物品 (电影) 为主题, 看看人们对我的评分, 然后构成的向量. 但是完整的理解, 最好加上统计的背景.

4 基于用户的协同过滤

所谓基于用户的协同过滤, 简单来说, 就是以人为本, 通过定义用户之间的相似关系, 找出和目标用户最相似的用户, 然后从该用户看过的电影 (而目标用户没有看过) 中挑几部推荐给目标用户. 在亚马逊中, “喜欢这个商品的人, 也喜欢某某某”就是基于用户的协同过滤.

以上面的电影推荐为例, 我们来说明具体的计算过程.

先看欧氏距离, 距离越近, 偏好越相似. 不过, 我们一般构造一个值, 偏好越相似, 这个值越大. 为此, 对于得到的欧氏距离, 我们可以取个倒数, 不过为了避免分母为 0, 我们将距离加上 1 后再取其倒数, 即书中的 sim_distance 函数

$$\text{sim\_distance}=\frac{1}{1+\sqrt{d^2}}$$

注意计算相似度时, 我们是忽略那些不是共同电影的评分, 即如果有一个人没有看过某部电影 (他对该电影的评分为 NA), 那么这部电影在计算中直接忽略, 特别的, 如果两个人没有公共的电影, 直接定义两人的相似度为 0.

比如计算用户 u_3 和 u_4 的相似度,u_3 和 u_4 对电影的评分向量为

$$\boldsymbol{x}=(2.5,3.0,NA,3.5,NA,4.0)^T$$
$$\boldsymbol{y}=(NA,3.5,3.0,4.0,2.5,4.5)^T$$

那么则有

$$\boldsymbol{x}-\boldsymbol{y}=(NA,-0.5,NA,0.5,NA,0.5)^T$$

所以他们的相似度为

$$\frac{1}{1+\sqrt{(-0.5)^2+0.5^2+0.5^2}}=0.5358984$$

再看 Pearson 相关系数, 计算公式可见上面, 注意同欧氏距离一样, 我们也是只对两个人都评价过的物品做计算, 否则忽略. 书中的 sim_pearson 函数便是实现了这一功能. 由于 Pearson 相关系数的范围是 $[-1,1]$, 所以我们也就不再变换了. 注意 R 语言中直接使用 cor 函数便可计算相关系数.

得到所有人与 u_a 的相似度后, 接下来我们采用 $K-$ 近邻法, 即选取与目标用户 u_a 最相似的前 K 个用户, 然后用他们对某个 u_a 没看过的电影的加权评分作为 u_a 对这个电影的"预期"评分, 这个权重便是相似度, 最后按照评分高低就可以对 u_a 推荐电影了.

对于本例, 除 u_a 外只有 6 个人, 取 $K=5$, 用 Pearson 相关度计算, 可得用户的相似度矩阵如下表2.

表 2: 用户相似度矩阵

用户	u_1	u_2	u_3	u_4	u_5	u_6	u_a
u_1	1	0.3961	0.4045	0.5669	0.5941	0.747	0.9912
u_2	0.3961	1	0.2046	0.315	0.4118	0.9638	0.3812
u_3	0.4045	0.2046	1	1	-0.2582	0.1348	-1
u_4	0.5669	0.315	1	1	0.5669	0.0286	0.8934
u_5	0.5941	0.4118	-0.2582	0.5669	1	0.2113	0.9245
u_6	0.747	0.9638	0.1348	0.0286	0.2113	1	0.6628
u_a	0.9912	0.3812	-1	0.8934	0.9245	0.6628	1

可得与 u_a 最相似的依次是 u_1,u_5,u_4,u_6 和 u_2(只排出了 u_3), 用户 u_a 没看的电影是 i_1,i_3 和 i_6, 列如下表格计算用户 u_a 对这三部电影的预期评分,

具体解释一下, 用户 u_1 与 u_a 的相似度为 0.99, 而他对电影 i_6 的评分为 3.0, 于是可认为 u_a 对电影 i_6 的评分为 $3.0\times0.99=2.97$, 光用用户 u_1 不太好, 我们把表中的 5 个人的加权评分作为用户 u_a 对 i_6 的预期评分, 也就是 3.35 分. 注意这其中也可能有缺失值, 比如计算用户 u_a 对电影 i_1 的预期评分时, 本来要用 5 个人加权计算, 可是用户 u_4 也没看过电影 i_1, 这里我们也是直接忽略之, 所以计算出来是 2.83.

表 3: 为 u_a 推荐

用户	相似度	i_6	$a.i_6$	i_1	$a.i_1$	i_3	$a.i_3$
u_1	0.99	3.0	2.97	2.5	2.48	3.0	2.97
u_2	0.38	3.0	1.14	3.0	1.14	1.5	0.57
u_4	0.89	4.5	4.02			3.0	2.68
u_5	0.92	3.0	2.77	3.0	2.77	2.0	1.85
u_6	0.66	3.0	1.99	3.0	1.99		
总计			12.89		8.38		8.07
Sim.Sum			3.84		2.95		3.18
总计/Sim.Sum			3.35		2.83		2.53

用这种方法计算了用户 u_a 对没看过的电影的预期评分之后, 便可以推荐了, 从表中看出, 最应该向他推荐电影 i_6, 其次是 i_1 和 i_3($3.35 > 2.83 > 2.53$).

这便是基于用户的协同过滤.

以上过程, 实际上是把评分矩阵中 u_a 的那一行向量“补全了”, 即如下

表 4: 对 u_a 推荐之后的评分矩阵

用户 \ 物品	i_1	i_2	i_3	i_4	i_5	i_6
u_1	2.5	3.5	3.0	3.5	2.5	3.0
u_2	3.0	3.5	1.5	5.0	3.5	3.0
u_3	2.5	3.0	?	3.5	?	4.0
u_4	?	3.5	3.0	4.0	2.5	4.5
u_5	3.0	4.0	2.0	3.0	2.0	3.0
u_6	3.0	4.0	?	5.0	3.5	3.0
u_a	2.83	4.5	2.53	4.0	1.0	3.35

我们对谁推荐, 就要把对应的那一行评分向量补全, 如果要对所有人推荐, 相当于补全整个评分矩阵. 此外, 补全的过程其实可以通过矩阵乘法实现, 比如要计算用户 u_a 对电影 i_6 的预期评分, 上面的结果是 3.35, 它实际上不正是用户 u_a 的相似度向量 (也就是用户相似度矩阵中对应 u_a 的那一行) 与电影 i_6 的评分向量 (即评分矩阵中对应 i_6 的那一列) 的加权乘积吗?

再说的清楚一点, 用户 u_a 的相似度向量是

$$\boldsymbol{x} = (0.9912, 0.3812, -1, 0.8934, 0.9245, 0.6628, 1)$$

而电影 i_6 的评分向量是

$$\boldsymbol{y} = (3.0, 3.0, 4.0, 4.5, 3.0, 3.0, 0)$$

以上计算时, 由于取的 $K = 5$, 所以删去了一个分量, 同时由于用户 u_a 没看过电影 i_6, 可理解为评分为 0, 因此计算时对应的分量也要删除. 以上计算相当于计算向量 $\boldsymbol{y}$ 的加权平均值, 而权重向量便是 $\boldsymbol{x}$.

其实, 大多数推荐算法的思想便是"补全"这个评分矩阵, 因为有了"预期"评分, 自然可以利用评分的高低进行推荐了.

5 基于物品的协同过滤

所谓基于物品的协同过滤, 简单来说, 就是以物为本, 直接建立各物品之间的相似度关系, 然后从目标用户没有看多的电影中挑选几部和他看过电影最相似的几部, 然后推荐给目标用户. 在亚马逊中, "买了这个商品的人, 也买了某某某"就是基于物品的协同过滤.

还是以上面的电影推荐为例子, 我们来说明一下计算过程.

前面计算相似度时已经提到过, 计算两个物品的相似度, 其实也是计算人们对它们俩的评分向量来实现的, 相当于把评分矩阵取了转置.

比如计算电影 i_2 和 i_1 的相似度, 它们的评分向量分别为

$$\boldsymbol{x} = (2.5, 3.0, 2.5, NA, 3.0, 3.0, NA)^T$$
$$\boldsymbol{y} = (3.5, 3.5, 3.0, 3.5, 4.0, 4.0, 4.5)^T$$

这里我们采用欧氏距离计算相似度, 仍然忽略缺失值, 可得

$$d^2 = (2.5-3.5)^2 + (3.0-3.5)^2 + (2.5-3.0)^2 + (3.0-4.0)^2 + (3.0-4.0)^2 = 3.5$$

按理说计算得到的相似度应该是

$$\frac{1}{1+\sqrt{d^2}} = \frac{1}{1+\sqrt{3.5}} = 0.34833$$

不过书上却写的是 0.22222, 实际上是书错了, 书中的代码是对 d^2 开了根号, 但是给出的结果没有开根号, 即书上计算的相似度实际为

$$\frac{1}{1+d^2} = \frac{1}{1+3.5} = 0.22222$$

这算是书里的失误, 其勘误可见http://www.oreilly.com/catalog/erratasunconfirmed.csp?isbn=9780596529321 (还有一处吐槽的地方, 就是书中在讲基于物品的协同过滤推荐之前, 就把评分矩阵取了转置, 美其名曰"匹配商品", 我觉得没有必要讲, 直接放到基于物品的推荐里就行了) , 为了便于解释 (此处偷懒), 我们就采用书上的数据 (即计算相似度时 d^2 不开根号). 所有物品的相似度矩阵为表5.

表 5: 物品相似度矩阵

物品	i_1	i_2	i_3	i_4	i_5	i_6
i_1	1	0.2222	0.2222	0.0909	0.4	0.2857
i_2	0.2222	1	0.1053	0.1667	0.0513	0.1818
i_3	0.2222	0.1053	1	0.0645	0.1818	0.1538
i_4	0.0909	0.1667	0.0645	1	0.0533	0.1026
i_5	0.4	0.0513	0.1818	0.0533	1	0.1481
i_6	0.2857	0.1818	0.1538	0.1026	0.1481	1

计算完相似度之后怎么办呢?

比如电影 i_2 和 i_1 的相似度是 0.222, 用户 u_a 是看过电影 i_2 的, 其评分是 4.5, 那么我们就把他对电影 i_1 的"预期"评分当做 $4.5 \times 0.222 = 0.999$, 当然, 仅用这一部电影不合理, 所以我们用 u_a 看过的所有电影做加权平均, 权重还是相似度, 比如用户 u_a 还看过电影 i_4 和 i_5, 电影 i_4 和 i_1 的相似度是 0.091, $4.0 \times 0.091 = 0.363$, 电影 i_5 和 i_1 的相似度是 0.4, $1.0 \times 0.4 = 0.4$, 所以总的来说, 用户对电影 i_1 的预期评分是

$$\frac{0.999 + 0.363 + 0.4}{0.222 + 0.091 + 0.4} = 2.473$$

实际上, 这也可以通过矩阵计算来实现, 以上过程实际上就是电影 i_1 的相似度向量 (即物品相似度矩阵的第一行) 与用户 u_a 的评分向量 (即评分矩阵中对应 u_a 的那一行) 的加权乘积.

说的清楚一点, 电影电影 i_1 的相似度向量为

$$\boldsymbol{x} = (1, 0.2222, 0.2222, 0.0909, 0.4, 0.2857)$$

用户 u_a 的评分向量为

$$\boldsymbol{y} = (0, 4.5, 0, 4.0, 1.0, 0)$$

计算本质上是对向量 $\boldsymbol{y}$ 的非零分量求加权平均值, 而权重向量为 $\boldsymbol{x}$.

对用户没看过的电影 i_6 和 i_3 可同样计算, 结果如下表6.

表 6: 对 u_a 提供基于物品的推荐

影片	评分	i_6	$a.i_6$	i_1	$a.i_1$	i_3	$a.i_3$
i_2	4.5	0.182	0.818	0.222	0.999	0.105	0.474
i_4	4.0	0.103	0.412	0.091	0.363	0.065	0.258
i_5	1.0	0.148	0.148	0.4	0.4	0.182	0.182
总计	0.433	1.378	0.713	1.762	0.352	0.914	
平均值			3.183		2.473		2.598

从表中可以看出, 最应该对用户 u_a 推荐电影 i_6, 其次是 i_3 和 i_1($3.183 > 2.598 > 2.473$), 这和之前基于用户的推荐顺序略有不同. 当然, 该方法的实质也是补全了评分矩阵.

6 关于协同过滤的补充

6.1 协同过滤推荐和基于内容的推荐的区别

还是举电影推荐的例子, 假设有 6 部电影, 命名为 $A, B, \cdots, F$, 用户 u_1 喜欢电影 A, B, C, 用户 u_2 喜欢电影 A, C, E, F, 用户 u_3 喜欢电影 B, D, 我们要解决的问题是对不对用户 u_1 推荐 F 这部电影?

所谓基于内容的推荐, 就是要分析电影 F 的特征和 u_1 喜欢的电影 A, B, C 的特征, 比如知道了信息 A(战争片)、B(战争片)、C(剧情片), 如果 F 是战争片, 那么我们很大程度上可以把 F 推荐给 u_1, 这就是基于内容的推荐. 可以看到, 要基于内容推荐, 就要知道用户和物品之间的关系, 我们需要对物品进行特征建立和建模分析 (比如这里电影的类型).

而上面我们已经看到, 协同过滤可以完全忽略对物品的建模, 我们不必知道 $A, B, \cdots, F$ 哪些是战争片, 哪些是剧情片, 我们只需要知道用户 u_1 和 u_2 按照物品可用向量表示, 他们的相似度比较高, 那么我们就可以把 u_2 喜欢的 F 这部电影推荐给 u_1.

以上就是协同过滤推荐和基于内容的推荐的区别.

6.2 同现矩阵

在有些协同过滤的推荐资料中, 会发现是利用同现矩阵计算的. 这和上面我们说的方法有什么不同呢? 下面我们就来说一说这个同现矩阵 (co-occurrence matrix).

还是以电影推荐为例, 其实主要思路都是一样的, 只是相似度计算上有所不同罢了. 上面我们主要讲了欧氏距离和 Pearson 相关系数, 下面再介绍

一个.

我们在上面给出的是评分矩阵, 即用户对物品的具体评分高低, 有时给出的信息可能不是那么详细, 可能就只有用户看没看过这部电影, 或者用户喜不喜欢这个商品, 比如我们把上面的评分矩阵简单的看成评过分就代表看过或者说喜欢, 即计算相似度时忽略具体评分的高低, 只看喜不喜欢. 这种计算相似度的方法就是 Jaccard 系数.

对于两个用户 u 和 v, 用 $N(u)$ 和 $N(v)$ 分别表示用户 u 和 v 喜欢的物品集合, 那么 Jaccard 用如下公式简单计算用户 u 和 v 的相似度

$$w_{uv} = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$$

当然, 也可以类比余弦距离, 换成余弦相似度

$$w_{uv} = \frac{|N(u) \cap N(v)|}{\sqrt{|N(u)| \cdot |N(v)|}}$$

比如计算表1中用户 u_3 和 u_4 的相似度, 我们先列出他们喜欢的物品集合

$$N(u_3) = \{i_1, i_2, i_4, i_6\}$$
$$N(u_4) = \{i_2, i_3, i_4, i_5, i_6\}$$

因此有 $|N(u_3) \cap N(u_4)| = |\{i_2, i_4, i_6\}| = 3$, 再除以分母就行了, 有些资料中并不除以分母, 看的时候要注意.

如果用这这种方法对两两用户都计算相似度, 那么时间复杂度是 $O(m^2)$, 比较耗时. 事实上, 很多用户相互之间没有对同样的物品产生过行为, 即很多时候 $|N(u) \cap N(v)| = 0$, 所以我们可以先计算出 $|N(u) \cap N(v)| \neq 0$ 的用户对 (u, v), 然后再除以分母即可.

为此, 我们定义一个用户的同现矩阵, 阶数为 $m \times m$, 其 (i, j) 元为用户 u_i 与用户 u_j 喜欢的共同物品的个数, 如果没有则赋值为 0.

比如对于上面的电影评分矩阵, 我们可以求出对应的用户的同现矩阵为

当然, 这肯定是一个对称矩阵. 其实, 归一化 (除以分母后) 的同现矩阵就相当于相似度矩阵. 而从上面我们已经知道, 相似度矩阵乘以评分向量就可以得到预测的评分. 这也是为什么一些资料中用同现矩阵乘以评分向量来得到预测评分的原因.

表 7: 用户的同现矩阵

用户	u_1	u_2	u_3	u_4	u_5	u_6	u_a
u_1	6	6	4	5	6	5	3
u_2	6	6	4	5	6	5	3
u_3	4	4	4	3	4	4	2
u_4	5	5	3	5	5	4	3
u_5	6	6	4	5	6	5	3
u_6	5	5	4	4	5	5	3
u_a	3	3	2	3	3	3	3

7 总结

7.1 参考资料

(1)《集体智慧编程》, 看的中文版

(2)《推荐系统实践》, 项亮编写的, 感觉一般般, 不过同现矩阵参考了它.

(3) 博客:http://blog.fens.me/rhadoop-mapreduce-rmr/, 用 Hadoop 实现了协同过滤算法, 基于同现矩阵.

参考文献

[1] 李荣华. 偏微分方程数值解法. 高等教育出版社 (2010)

[2] Zhilin Li,Zhonghua Qiao,Tao Tang.*Numerical Solutions of Partial Differential Equations-An Introduction to Finite Difference and Finite Element Methods.*(2011)

[3] 孙志忠. 偏微分方程数值解法. 科学出版社 (2011)

[4] 陆金甫关治. 偏微分方程数值解法. 清华大学出版社 (2004)

附录