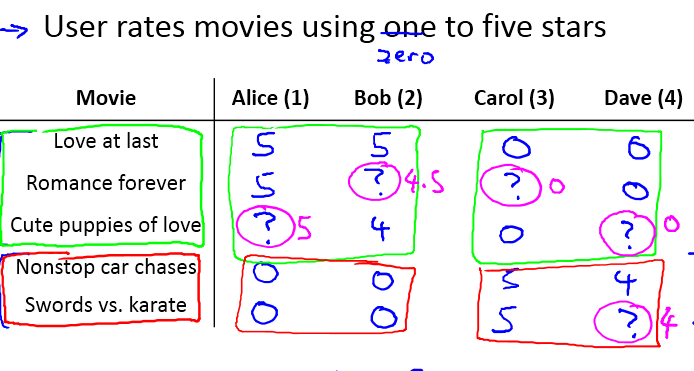
# 推荐系统

1. 题目是要我们写个像豆瓣那样的推荐系统。根据已有的用户对电影的评分去预测此用户将会对其他电影给出的评分。



已知一些用户（Alice、Bob、Carol、Dave）和一些电影（Love at last、Romance forever…），用户对某些看过的电影评了分（0-5分），把它们列在表上（打问号的表明用户尚未对此电影评分）。我们的任务是预测表中的？处该填的数值。

1. 输入为： nu表示用户数

nm表示电影数

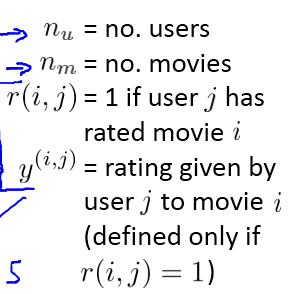
r(i, j)表示j用户是否已经对i电影评分

比如r(Love at last, Bob)=1 表明Bob已经对Love at last评了分

r(Cute puppies of love, Alice)=0 表明Alice没有对Cute puppies of love评分

y(i, j)表示j用户给电影i的评分

比如y(Love at last, Bob)=5



输出为： a(i, j)=我们所预测的用户j对电影i的评分

1. 那么怎么实现评分的预测呢？（中间省略n个步骤，直接跳到协同过滤算法）

电影可以分很多种类吧，比如说恐怖片、爱情片、动作片balabala。对每个电影我们用一个x向量来表示它的各种类型电影元素的含量。比如说：

X(3)=[0.99; 0]

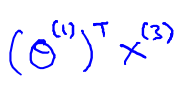
表明第3部电影中爱情片的元素为0.99，而动作片的元素几乎没有。

同样的，对每个用户，我们可以设一个theta，表明该用户对某种类型电影元素的喜爱程度。比如说：

Ɵ(1)=[5; 0]

表明第1个用户对爱情片的元素的喜爱程度为5，而几乎不喜欢看动作片元素。

那么 第1个用户对第3部电影的评分可以预测为：



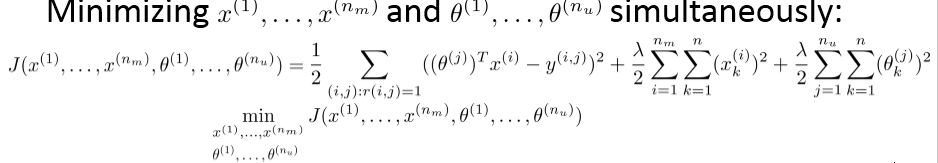
（声明一下，**实际上X中的元素很难找到它的具体意义**，这里只是打个比方说他们分别代表爱情片、动作片神马的）

1. 那么现在问题来了：怎么求出Ɵ和X？

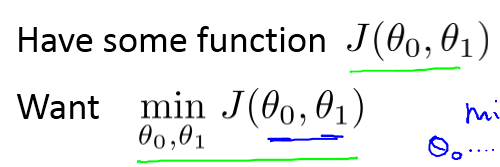
对每个用户我们有Ɵ(1)、Ɵ(2)…Ɵ(nu)

对每部电影我们有X(1)、X(2)…X(nm)

我们用梯度下降法（Gradient Descent）求：



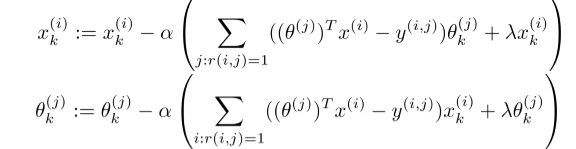
回想一下Gradient Descent是怎么求最原始的线性回归的：



我们首先随便初始化Ɵ0、Ɵ1，然后再根据J的偏导来梯度下降。

这里的原理是一模一样的——随便初始化Ɵ(1)、Ɵ(2)…Ɵ(nu)、X(1)、X(2)…X(nm)到一个[-Ƹ, Ƹ]的范围内（Ƹ很小），再一步步用J的偏导梯度下降即可。

协同过滤算法的J的偏导在此：



当然我们也可以用一些advanced optimization algorithms。

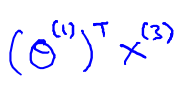
1. 那么怎么把协同过滤算法应用到分布式计算中呢？

在每走一步梯度下降的过程中，我们可以在map步骤让不同的计算机分布的求Ɵ(1)、Ɵ(2)…Ɵ(nu)、X(1)、X(2)…X(nm)的新值，然后再集中reduce。

1. 难怪大家会一直围着Ɵ(1)、Ɵ(2)…Ɵ(nu)、X(1)、X(2)…X(nm)转，我忘记强调最后一步了。

在用Gradient Descent求出Ɵ(1)、Ɵ(2)…Ɵ(nu)、X(1)、X(2)…X(nm)后，我们就可以用它们来进行预测评分了。具体的方法如前文中第3点所说：

**第1个用户对第3部电影的评分可以预测为：**

****

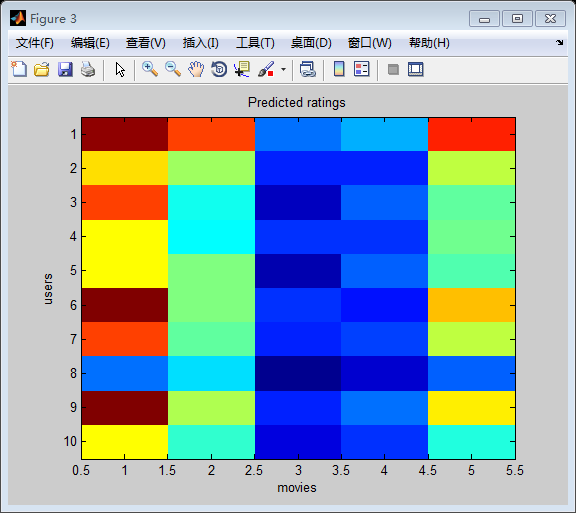
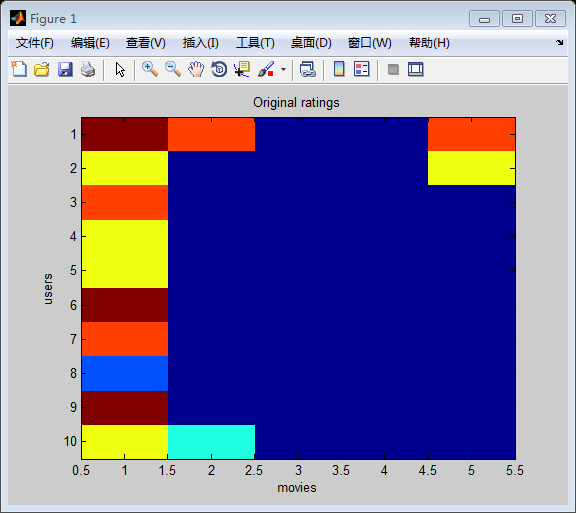
以此类推，对于第i个用户对第j部电影的评分，我们可以预测为：

Ɵ(i)T X(j)

这样整个预测评分任务就解决了。

1. 我们看看实际效果怎么样：

下面是用Matlab实现的原型输出的两幅图——



两张图（前者代表原始评分，后者代表预测的评分）中的**每一个色块表示某个用户对某一部电影的评分**（如左下角就是第一个用户对第一个电影评分的可视化表示），颜色相似表明值也相似，第一幅图中成块的深蓝色区域是指0，即用户尚未对这些电影评分。注意到**第二张图中与第一张图的色块颜色相似（除去第一张图中蓝色的未评分部分）**，这表明我们用Ɵ(1)、Ɵ(2)…Ɵ(nu)、X(1)、X(2)…X(nm)预测出来的值和原来的值及其相似，一定程度上反映了此算法的有效性。