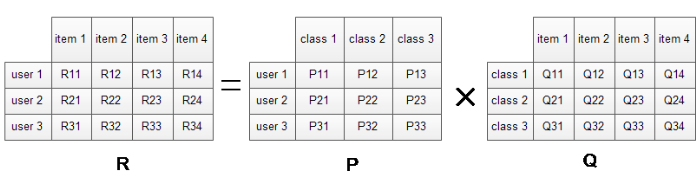
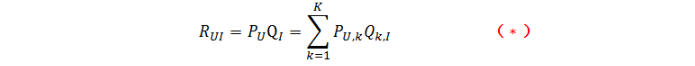
**一、代码理论模型**（参考书本《推荐系统实践》以及《机器学习》中理论内容，可跳过看后文具体思路和实现）

**1.LFM**

对于一个给定的用户行为数据集（数据集包含的是所有的user, 所有的item，以及每个user有过行为的item列表），使用LFM对其建模后，我们可以得到如下图所示的模型：（假设数据集中有3个user, 4个item, LFM建模的分类数为4）



R矩阵是user-item矩阵，矩阵值Rij表示的是user i 对item j的兴趣度，这正是我们要求的值。对于一个user来说，当计算出他对所有item的兴趣度后，就可以进行排序并作出推荐。LFM算法从数据集中抽取出若干主题，作为user和item之间连接的桥梁，将R矩阵表示为P矩阵和Q矩阵相乘。其中P矩阵是user-class矩阵，矩阵值Pij表示的是user i对class j的兴趣度；Q矩阵式class-item矩阵，矩阵值Qij表示的是item j在class i中的权重，权重越高越能作为该类的代表。所以LFM根据如下公式来计算用户U对物品I的兴趣度

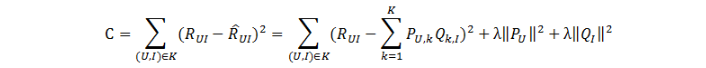


我们发现使用LFM后，

1. 我们不需要关心分类的角度，结果都是基于用户行为统计自动聚类的，全凭数据自己说了算。
2. 不需要关心分类粒度的问题，通过设置LFM的最终分类数就可控制粒度，分类数越大，粒度约细。
3. 对于一个item，并不是明确的划分到某一类，而是计算其属于每一类的概率，是一种标准的软分类。
4. 对于一个user，我们可以得到他对于每一类的兴趣度，而不是只关心可见列表中的那几个类。
5. 对于每一个class，我们可以得到类中每个item的权重，越能代表这个类的item，权重越高。

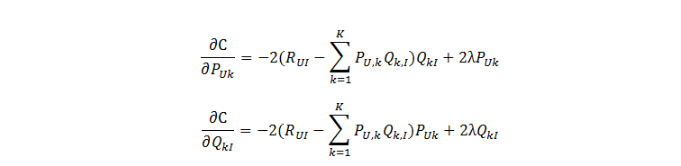
那么，接下去的问题就是如何计算矩阵P和矩阵Q中参数值。一般做法就是最优化损失函数来求参数。在定义损失函数之前，我们需要准备一下数据集并对兴趣度的取值做一说明。

数据集应该包含所有的user和他们有过行为的（也就是喜欢）的item。所有的这些item构成了一个item全集。对于每个user来说，我们把他有过行为的item称为正样本，规定兴趣度RUI=1，此外我们还需要从item全集中随机抽样，选取与正样本数量相当的样本作为负样本，规定兴趣度为RUI=0。因此，兴趣的取值范围为[0,1]。  
  
  
采样之后原有的数据集得到扩充，得到一个新的user-item集K={(U,I)}，其中如果(U,I)是正样本，则RUI=1，否则RUI=0。损失函数如下所示：

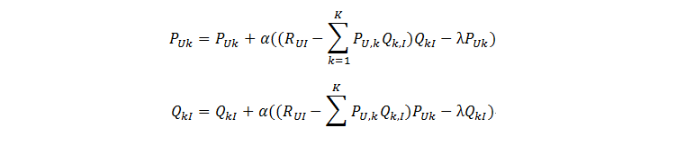


上式中的https://img-blog.csdn.net/20130812142056953是用来防止过拟合的正则化项，λ需要根据具体应用场景反复实验得到。损失函数的优化使用随机梯度下降算法：

1. 通过求参数PUK和QKI的偏导确定最快的下降方向；

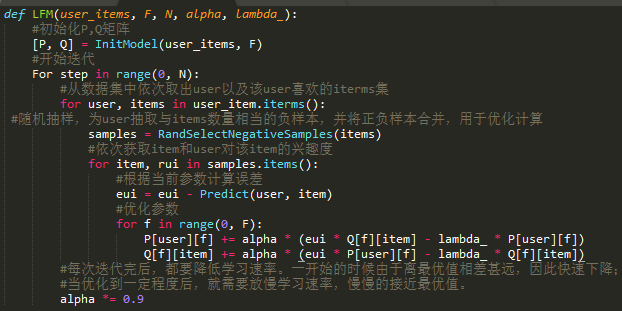


1. 迭代计算不断优化参数（迭代次数事先人为设置），直到参数收敛。



其中，α是学习速率，α越大，迭代下降的越快。α和λ一样，也需要根据实际的应用场景反复实验得到。

LFM的伪代码可以表示如下：



**2.Kmeans**

K-Means算法的基本思想是初始随机给定K个簇中心，按照最邻近原则把待分类样本点分到各个簇。然后按平均法重新计算各个簇的质心，从而确定新的簇心。一直迭代，直到簇心的移动距离小于某个给定的值。  
  
K-Means聚类算法主要分为三个步骤：  
(1)第一步是为待聚类的点寻找聚类中心  
(2)第二步是计算每个点到聚类中心的距离，将每个点聚类到离该点最近的聚类中去  
(3)第三步是计算每个聚类中所有点的坐标平均值，并将这个平均值作为新的聚类中心  
(4)反复执行(2)、(3)，直到聚类中心不再进行大范围移动或者聚类次数达到要求为止

1. **思路分析和遇到的问题**

在开始正式讲代码之前，先来说说对几个细节问题的解决思路：

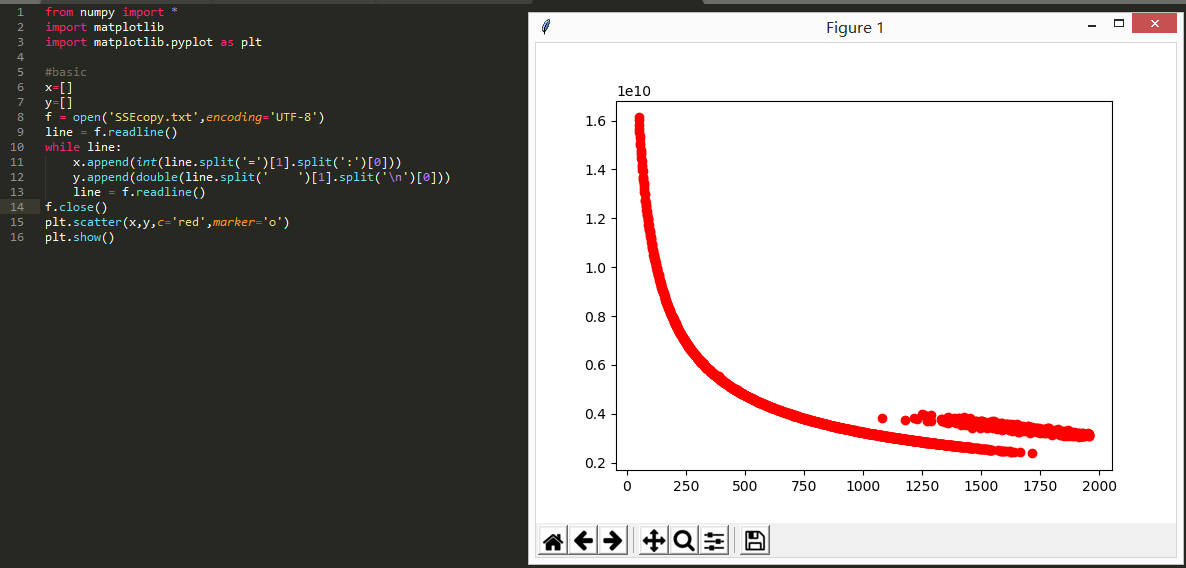
1.根据数据集初始化P和Q矩阵

这个初始化其实耗费了巨量的写代码时间，我的思路是

1. 对于P矩阵，对每一个用户计算其自己评分的平均值，然后用户对每个class的评分就以平均值为基础进行正态分布的随机值取值。
2. 对于Q的初始化前前后后做了很多工作：一开始就是以所有评分的平均值进行正态随机值（显然这和理论不符合，理论上Q是概率矩阵，也就是某个item所有class加起来要为1，但一开始就先简单赋予初值了）。后来结合itemAtrribute的文件，首先对这个文件里的item进行聚类（使用kmean），对于聚类结果我的利用方式有两种：
3. 一种是根据聚类结果把item分为150个class（为什么是150后文有解释），那么对于Q矩阵的初始化就可以按照我们的理论进行初始化了：某个item根据聚类结果属于某个class（又出现在itemAttribute最好，没有也可以和150个中心点算距离得到类别），将这个class对应的数设为某个参数（需要实验，可以说取决于你的class聚类的可信度，可信度高就概率给的大接近1，低就小），当然还是进行正态处理得到p，对于这个item其他class的值，则是（1-p）/149再分别进行正态处理，为了保证和为1最后那个p得重新赋值1-sum（其他）。
4. 第二种想法其实相当于增加数据集，根据聚类结果可以将一些同class的item在相同user那的评分给个差不多的值（同样正态处理），这样就能减少过拟合的风险，当然效果也要取决于聚类的可信度。

2.确定4个参数：分类数F，迭代次数N，学习速率α，正则化参数λ。

其中α，λ都是实验出来的选择了0.015和0.01，N是一开始20次，之后可以看结果继续决定需不需要继续。对于F，这边结合itemAtrribute的聚类结果，计算其SSE(误差平方和)来定下。因为随着聚类数k的增大，样本划分会更加精细，每个簇的聚合程度会逐渐提高，那么误差平方和SSE自然会逐渐变小。并且，当k小于真实聚类数时，由于k的增大会大幅增加每个簇的聚合程度，故SSE的下降幅度会很大，而当k到达真实聚类数时，再增加k所得到的聚合程度回报会迅速变小，所以SSE的下降幅度会骤减，然后随着k值的继续增大而趋于平缓，也就是说SSE和k的关系图是一个手肘的形状，而这个肘部对应的k值就是数据的真实聚类数。所以我们如下图所示选择150这个数作为F



正式讲代码之前想说一下过程中碰到的一些问题：

1. 首先第一个关于代码的速度问题，python的方便快捷确实让我在写代码过程中十分愉悦，但是大数据量下的长时间需要我们考虑并行等加速手段，这时候就有点苦恼了，尝试自己写了几个多线程来加速但是发现相对于串行真的没有提升，看到了python官方的一些调用其自己的多线程的东西还没加进去。一些C++中常用的方式比如mpi openmp等我也还没具体查阅python的用法，这点还需要后续补充。
2. 第二个关于迭代中对于参数优化梯度下降也有一些问题，一开始简单初始化就很随意按照公式优化参数，没有问题，准确率也是正常上升。但是后边真正认真初始化后反而有些问题凸显出来了，首先由于和为1的限制导致Q矩阵实际上很多数特别小（相对于P矩阵，我把两者都放在0~1的范围内了，所以简单初始化两者差不多），所以真正按照公式对两者参数优化就会发现，优化个一轮大量出现predict\_score几百的现象（predict\_score是函数，用来算损失函数时候算预测值的，就是把对应user和item的150个class加起来乘以100就是score），一开始我一直想不明白为啥Q和为1以后会超出100，后来想到了因为优化的尺度问题，一边全是0.1精度的数，一遍大部分0.001，用同样的alpha和lamda就有问题，对于P矩阵很好但是Q矩阵的话，只要一轮和为1的条件就被大大破坏，结果自然而然十分滑稽。
3. **具体代码分析**

**1.kmeans聚类**

具体到kmeans的实现，这里数据量太大，所以借用Python的scikit-learn 提供了MiniBatchKMeans算法，大致思想就是对数据进行抽样，每次不使用所有的数据来计算，这就会导致准确率的损失。

Mini Batch K-Means算法是K-Means算法的变种，采用小批量的数据子集减小计算时间，同时仍试图优化目标函数，这里所谓的小批量是指每次训练算法时所随机抽取的数据子集，采用这些随机产生的子集进行训练算法，大大减小了计算时间，与其他算法相比，减少了k-均值的收敛时间，小批量k-均值产生的结果，一般只略差于标准算法。

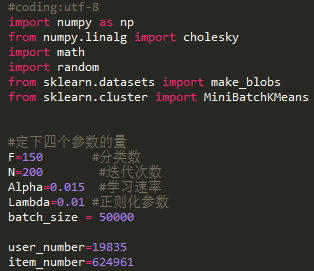
该算法的迭代步骤有两步：  
1：从数据集中随机抽取一些数据形成小批量，把他们分配给最近的质心  
2：更新质心

与K均值算法相比，数据的更新是在每一个小的样本集上。对于每一个小批量，通过计算平均值得到更新质心，并把小批量里的数据分配给该质心，随着迭代次数的增加，这些质心的变化是逐渐减小的，直到质心稳定或者达到指定的迭代次数，停止计算Mini Batch K-Means比K-Means有更快的 收敛速度，但同时也降低了聚类的效果，但是在实际项目中却表现得不明显

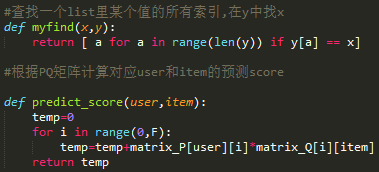


Kmeans.py代码部分不多，调用MiniBatchKMeans通过改变不同的class分类结果的数量来计算SSE，结果也展示过了，得到结果F=250时候聚类效果最好。这个结果为我们的正式的要运行的test.py里的参数F=250提供支持，在test.py里开头也会有Kmeans的相关代码（和这里差不多，定下F=250），为的是把item分好类为后边初始化做准备。

**2.test.py里的主代码讲解**



一开始定义了需要的参数变量



这里是后边会用到的两个函数，myfind是用来查找一个list里某个值得所有索引的，predict\_score是用来预测某个用户对某个item的分数的

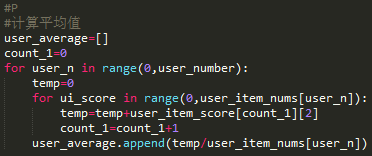


这里是对train.txt的读取，制作成[userid，itemid，score]的list方便后面使用。注意的是这里加入了几个特别的操作，首先是对每个user评论过的·item会随机抽取一个作为测试集的东西，其他作为训练集。第二，对于评论数太少的user（小于500），借助itemattribute的聚类结果会加入一些item和分数来减少矩阵的稀疏。

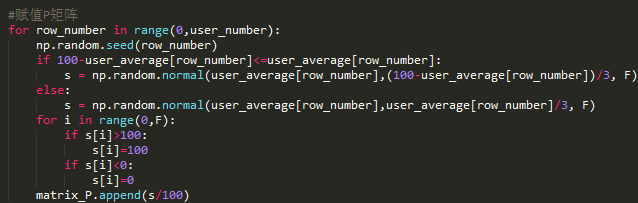


**关于kmeans以及上文说过不再赘述，接下来进入核心代码部分：**

1.首先对P的初始化，大致思路就是用某个user对某个类的评分高低计算其评分的平均值进行初始化



首先对所有的user计算其自己评过分数的item的平均值

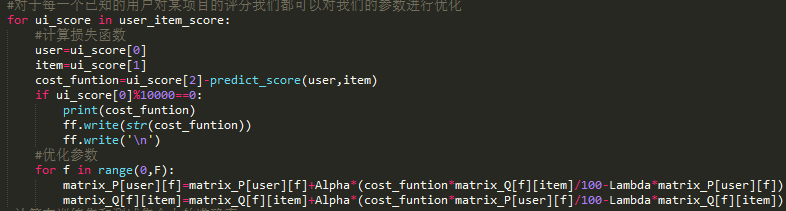
接下来逐行对P矩阵进行初始化，想法也很简单，每一行就是每个user对每个class的评分，初始化就以这个user的平均值为基准用np.random.normal用平均值做均值，（100-均值）/3或者（均值-100）/3做方差（目的是3西格玛原则不要跳出0到100分这个范围），然后规整到0到1赋予P矩阵每个值。

2. 然后就是Q矩阵的初始化，Q矩阵是item属于某个类的概率，所以我们结合聚类的结果来进行初始化

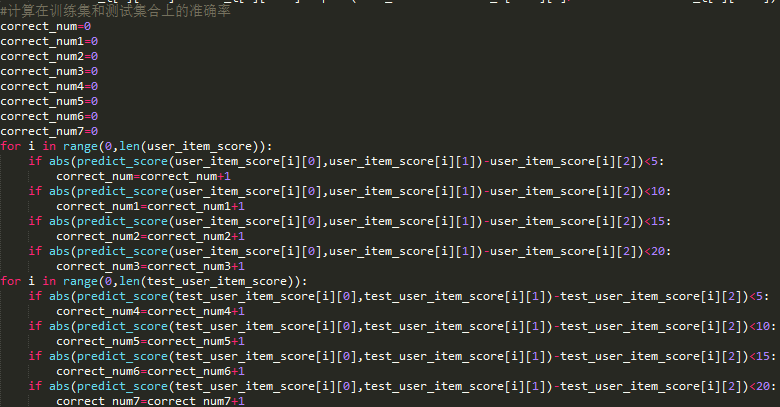


这里也有个需要一直实验再修改的参数，就是聚类正确的概率是多少，我们假设为0.1.所以对于每一个item的250个class，根据kmeans的结果得到的聚类结果把对应的item赋予0.1（当然还是normal一下），然后剩下的249个就分剩余的概率，集体操作不说了保证加起来为1即可。最后记得转置一下。

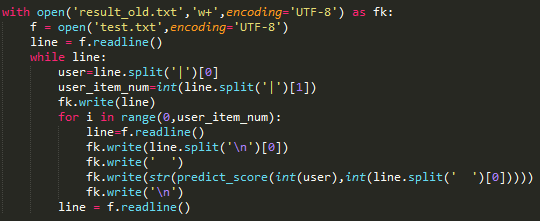
3.正式进入学习迭代阶段



迭代的外层循环以及一些小操作就不放出来了，核心部分其实很简单，根据理论部分的内容进行梯度下降的参数优化。所以要做的就是在每次迭代过程中对每个训练集的user item计算其预测值和实际分数对比算出损失函数，然后再优化参数即可。



这就是代码中计算准确率的部分，分别统计在train和test集和正确答案相距5、10、15、20的个数除以总数。



最后迭代轮次结束达到一定的精度以后就可以把test.txt的内容意义predict\_score输出了。

**四、结果展示**

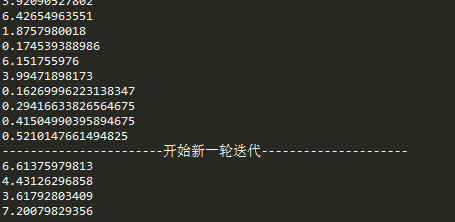
本次代码结果所用的评判标准是准确率，具体来说分为八个值。训练集和测试集各四个。

分别为对对应user和item预测值和实际值的差值绝对值大小在5分以内、10分以内、15分以内和20分以内的概率。



每一次迭代在窗口都会给出固定的一些user-item项的预测实际分数差距，以及非常重要的8个准确率供我们查看。到迭代后期我将其设计成每次迭代结束不是继续而是询问是否continue并且手动输入alpha等参数值，这样虽然需要人为输入但是可以更具情况来进行参数选择和迭代。当最后我们发现test集的准确率无法上升的时候我们可以知道再在训练集训练下去将会出现过拟合的情况，所以这时可以选择不再continue，程序将用这时候的矩阵来为我们预测需要的user item分数。

迭代过程的所有记录都在evidence.txt中



最后的结果在result.txt中

