

推荐算法实现与优化

图灵：楼兰

六、推荐系统设计与改进

- 1、数据处理问题
- 2、推荐算法
- 3、算法实战
- 4、结果评估与模型优化

总结：

六、推荐系统设计与改进

这一章节回到我们的推荐系统问题。其实经过前面的总结，应该不难知道，推荐系统这样一个典型问题，其实按照他对于推荐规律的不同理解程度，可以分为以下四种设计方式：

- 基于统计学的推荐系统：

我们之前基于商品销量排行的推荐系统就是一种典型的基于统计学的推荐系统。这类推荐系统实现比较简单，但是千篇一律，所有人看到的東西都一样，不利于深入挖掘用户与产品之间的关系，更适用于传统购物场景。

- 基于规则的推荐系统：

可以通过一些简单的业务规则来指定推荐系统的实现方式。例如，一个电影院可以简单的把动画电影主推给青少年，把战争大片主推给年轻人，而把爱情片主推给中老年人。这种推荐系统，更多的是由业务来进行驱动，软件层面实施起来相对就会非常简单。比较适用于时间比较短，用户比较少，数据量规模也比较小的电商场景。

- 基于传统机器学习的推荐系统：

当电商的整体规模逐渐扩大，全方位的广告覆盖就会变得得不偿失。这个时候就需要更深度的挖掘历史数据，深入分析用户与产品之间的潜在关系，进行更据针对性，更有效的产品推荐。这时候，机器学习算法就能更好的帮我们寻找处理和计算这些历史数据的方式，并且通过建立数学模型，对未来的交易行为进行一定的预测。这也是我们会主要讨论的推荐系统。

- 基于深度学习的推荐系统：

但是，当电商的规模继续扩大，或者需要考虑同行业合作这种数据量更大的场景时，数据不光是体量增加，数据之间的关系也会变得更为复杂，这时基于数学计算的传统机器学习算法依然会表现得比较乏力，甚至于可以说人的理解能力也都会成为限制。这时就需要更为强力的基于神经网络的深度学习算法来加强机器学习的能力。这些深度学习算法其本质虽然依然是数学计算，但是他是在模拟人脑的结构以及处理问题的方式，可以理解为具备了一定的自我理解与自我演化的能力。认为是未来的推荐系统，目前也有很多大企业在使用像AlphaGo这样可以自我演化的推荐系统，但是在行业内还并没有形成规模。到时候，可能我们以前经常说的，你在淘宝买了块瑜伽垫，我马上敲门给你推荐减肥药这样的段子，也就成为常态。

我们这一章节主要讨论第三种，基于传统机器学习的推荐系统应该要如何进行核心的历史数据计算以及要如何设计系统的整体方案。

1、数据处理问题

用户画像：

关于推荐系统的数据要求，在之前已经简单做过介绍。推荐学习算法需要的数据是以(userId,ProductId,Rating)这样的向量数据的格式组成的一个稀疏矩阵。其中Rating表示用户对产品的评分。用来表示一个推荐的结果。可以是0和1这样的二分类结果集，也可以是一个有范围的正整数，表示用户对产品的评分结果集。而userId, productId, 这两个关键属性在机器学习算法中，只是作为一个与具体对象特征无关的一个代表值。这样确实也可以根据用户的历史行为数据做出一个比价好的推荐结果。但是，你可能会觉得，这样的推荐系统，跟用户或者产品的一些具体属性就脱节了。

所以，往往在对推荐系统进行业务优化的过程中，会将这个userId和productId替换成对用户特征有一定描述性的特征值。例如，对用户进行特征画像，比如将用户按照年龄阶段，划分为0-老年人、1-中年人和2-青少年三个值，将用户按照年收入情况划分为0-高产、1-中产以及2-低产三个值等等，这个过程就是用户画像的过程。用户画像往往是机器学习的基础。在前面的章节提到过，对机器学习来说，数据和特征就决定了机器学习效果的上限，所以，通过用户画像对机器学习的数据进行及时适当的梳理和调整，是整个机器学习的基础。

当然，在推荐场景，一方面，要对画像的各种结果进行收敛，将所有的特征值整合成一个具有业务意义的userId，这样就可以用来进行机器学习了。比如，像支付宝那样，将用户的所有相关数据整合成一个蚂蚁信用分，然后用这样的信用分来进行学习，就能形成更具有明确业务意义的推荐系统。这样的结果，他的业务解释性也会更强。当然，这里需要注意一下用户区分度的问题，很多用户都可能有相同的分数。另一方面，其实画像也不仅仅针对用户，在推荐场景中对产品同样需要进行画像，这样形成的结果不光可以用来解释给一个特定的客户要推荐那些产品，反过来也可以解释给一个特定的产品要推荐给哪些用户。

数据来源：

前面提到了数据如何处理的问题，他需要从业务系统中整理出来。但是很多场景下，难的不是如何去清洗数据，而是压根就没有历史数据。这也就是所谓的冷启动问题。比如新上一个电影推荐系统，还完全没有历史业务数据，那要怎么开展基于机器学习的电影推荐呢？

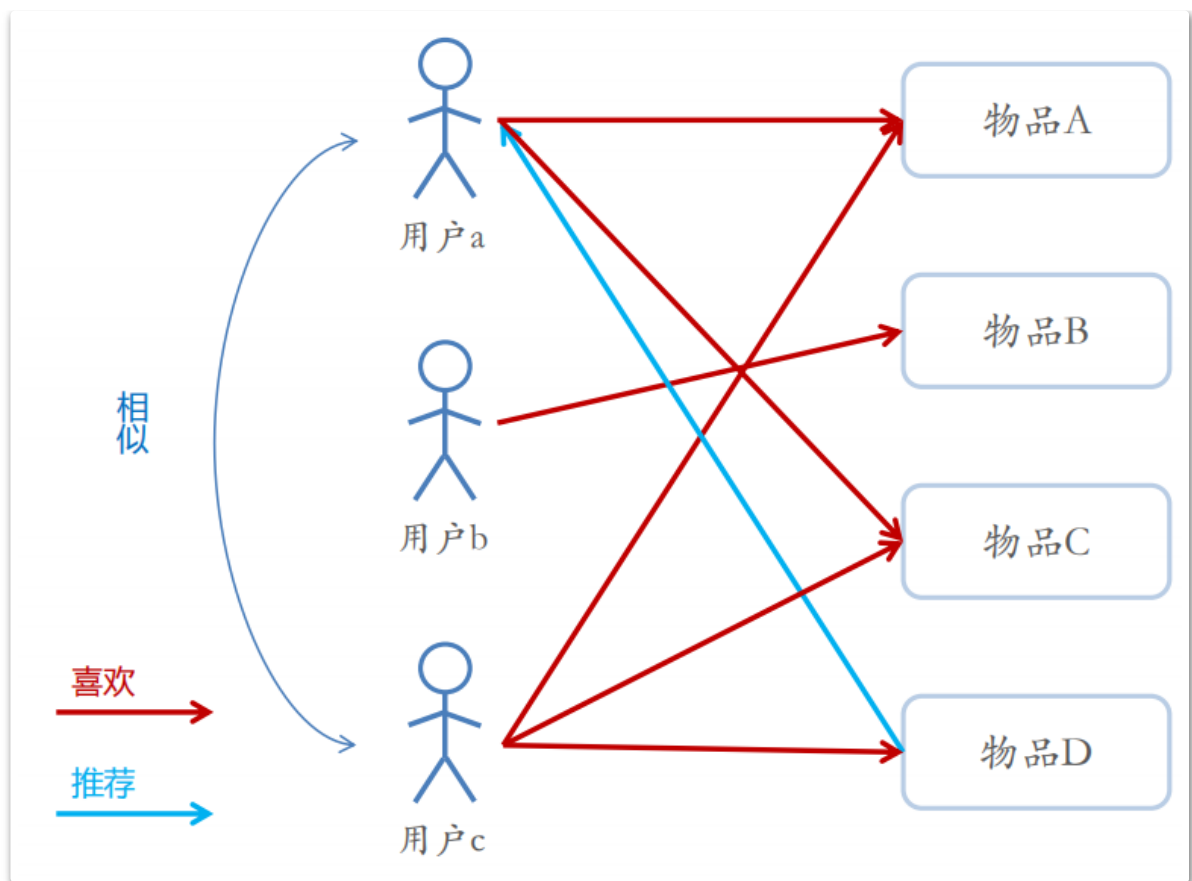
这个时候，往往会采取一些措施来人工产生一些数据。比如像豆瓣上对电影的评价，就是一些比价好的数据。这时有两种大的渠道。一种是邀请专家来评分，比如邀请专业影评人来进行评分。这种数据称为PGC(Professionally-generated Content)：专家数据。另一种就是广邀群众来进行评分。这种数据称为UGC(User-generated Content)：用户数据。比如豆瓣上的评分就是这种类型。这两种数据，在进行数据处理前，其实是可以分开进行处理的。往往PGC数据质量比较高，而UGC的数据质量则相对较低。这时，是可以对这些数据做不同处理的。例如对UGC，最好是去掉一些明显比较偏颇的数据。这样，整体的数据质量更高，机器学习的上限也会更高。另外，在评分的同时，往往也需要综合他们的评论。而对评论这样的文本数据要怎么进行特征抽取呢？这就是我们之前提到的特征工程了。

2、推荐算法

前面说了一些前置的数据处理问题，接下来进入推荐算法的正题。

协同过滤 Collaborative Filtering Recommendation：

我们之前提到，推荐算法的本质就是对推荐矩阵里的缺失值进行填充的过程。那要怎么填充才会比较合理呢？这就涉及到推荐算法的根基：协同过滤(Collaborative Filtering Recommendation)。



他的基础思想就是先使用一些统计学的方法，寻找与目标用户有相同喜好的邻居，然后根据目标用户的邻居的喜好产生向目标用户的推荐。例如在上图中，用户a和用户c，有比较相同的喜好，都喜欢物品A和物品C。那这样，用户c对物品D的喜好就比较适合推荐给用户a。

这种方式，就是以用户为基础的协同过滤，称为UCF(User-based Collaborative Filtering)。其实反过来，从物品的角度，也可以来寻找对应的用户，建立推荐关系。这就称为CCF(Content-based Collaborative Filtering)。

	多啦A梦	机器总动员	指环王	寂静之地	速度与激情
用户A	5	5	2	1	5
用户B	2	1		5	2
用户C	5	?	3	1	5
用户D	3	3	5	4	

比如我们看这样的一个推荐矩阵。我需要预测用户C对于机器总动员这部电影的喜爱程度。喜爱程度较高，就需要给用户C主动发送机器总动员电影上映的推荐信息。这个时候要怎么去填充?位置的值呢？首先从用户的角度，横向的来看，用户A与用户C有很多相似的爱好的。而对于机器总动员这部电影，用户A的评价很高。那我

们就有理由去预测，用户C对机器总动员电影的喜爱程度也会比较高，给他填个4或者5都是比较合理的，这样对用户C，就可以有针对性的主动发送机器总动员这部电影的推荐信息了。这就是基于UCF的一种推荐机制。另外，从电影的角度，纵向来，多啦A梦这部电影与机器总动员这部电影在用户群里中的评价也很相似，而对于多啦A梦这部电影，用户C的评价比较高，那同样我们也有理由去预测，用户C对机器总动员电影的喜爱程度会比较高，同样给他填个4或者5都是比较合理的。这样如果电影院要推广机器总动员这部电影，就可以主动向用户C发送推荐信息。这就是基于CCF的一种推荐机制。

隐语义模型： Latent Factor Model

有了协同过滤的这个基础的机制，那剩下就是如何去实现了。前面我们只是简单的通过直观判断分析出矩阵之中的一些相似性。但是这种相似性毕竟太过感性化。基于这种机制，通常可以再转为使用KNN算法，以基于K个邻居的历史偏好信息，为当前用户进行推荐。

但是，转为使用KNN后，有什么不好呢？KNN是基于特征值+目标值的数据结构，那整个数据的结构就需要调整。所以对于已经准备好的推荐矩阵的挖掘理解实际上是不够的。那有没有一种更好的算法实现方式呢？这就可以考虑隐语义模型。


隐语义模型的基础思想是这样的，如果可以找到两个矩阵，其中一个矩阵与推荐矩阵有相同的行(用户)，另一个矩阵与推荐矩阵有相同的列(产品)，然后他们两个矩阵的乘积正好满足推荐矩阵。那就可以用这两个矩阵来计算出完整的推荐矩阵。

这样就把一个充满玄学的感性问题，转换成了一个数学求解的问题了。这个求解过程是比较复杂的，我们这里就不再去讨论这样的数据问题了。但是我们从工程上理解一下，如果真的找到了这样的两个矩阵，实际上就代表我们找到了一种UCF与CCF之间的关联。



例如通过这个示例，我们通过一个不完整的用户评分矩阵 R' ，找到了一个用户特征矩阵 P 和电影特征矩阵 Q ， $P \cdot Q$ 能够满足已有的评分矩阵 R' ，并且他们两个矩阵相乘后，就得到了一个新的完整的用户评分矩阵 R 。我们可以认为有一些隐藏的因素，影响了用户对电影的评分。这其中 f_1, f_2, f_3, f_4 就代表了某种潜在的关联特征，比如电影的演员、题材、年代或者是一些人所不能理解的隐藏因子。这种感觉就很像我们经常提到的"直觉"或者"玄学"。这也就是我们所说的隐语义模型。但是事实上，我们并不需要真正去理解这些隐藏因子是什么，通过这些隐藏因子，就可以找到用户与电影的关联关系，就可以推测用户是否会喜欢某一部没有看过的电影。

	Movie1	Movie2	Movie3	Movie4
user1	1		2	
user2		5	1	
user3	2		0	
user4		4		2
user5	1		2	
user6	3			5
user7	5		1	
user8		1		0
user9	2		3	



	Movie1	Movie2	Movie3	Movie4
user1	1	4	2	5
user2	3	5	1	1
user3	2	4	0	5
user4	1	4	0	2
user5	1	3	2	1
user6	3	1	2	5
user7	5	1	1	2
user8	2	1	0	0
user9	2	2	3	4

接下来，应该要知道，这样的用户特征矩阵和电影特征矩阵应该是不止一个解的，所以，还需要有一个算法来对计算的结果进行评测，寻找一组最优化的解。并且，在机器学习中，由于有过拟合问题的存在，最优解一般来说并不是最正确的解。有一些小的误差，往往更能适合对实际问题进行预测。

这个过程同样会有几种方法，而在Spark中我们集成了其中一种方法。最小二乘法：ALS(Alternating Least Square :最小交替二乘法)。他的基础思想是对 P, Q 两个矩阵，先固定其中第一个矩阵，去求第二个矩阵的最优解，接下来再固定第二个矩阵，转过来求第一个矩阵的最优解。通过多次交替迭代，最终得到最佳的 P, Q 组合解。这是一种梯度下降，逼近最优解的过程。

3、算法实战

- 1 Spark Demo:
- 2 基于DF的ml实现
- 3 1、org.apache.spark.examples.ml.JavaALSExample
- 4 基于RDD的mllib实现
- 5 1、org.apache.spark.examples.mllib.JavaRankingMetricsExample
- 6 2、org.apache.spark.examples.mllib.JavaRecommendationExample

其中，Spark推荐的是基于RDD的mllib实现。其中的核心代码就是创建ALS的过程：

```
1 MatrixFactorizationModel model = ALS.train(JavaRDD.toRDD(ratings), rank,
    numIterations, 0.01);
```

对于ALS.train方法，第一个参数就是推荐向量集合。第二个参数rank就是隐语义的个数。第三个参数就是交替执行的次数。第四个参数是正则化系数，你可以认为是用来优化结果的一个参数。

然后，构建出来的这个model就是训练得到的一个推荐模型。通过这个推荐模型，就很容易构建出完整的推荐系统。

```
1 //拿到所有用户的推荐产品ID。也就是UCF的结果
2 RDD<Tuple2<Object, double[]>> userFeatures = model.userFeatures();
3 //拿到所有产品的推荐用户ID，也就是CCF的结果
4 RDD<Tuple2<Object, double[]>> productFeatures = model.productFeatures();
5 //使用模型来预测用户与产品之间的推荐分数。预测分数与推荐分数是存在差异的，他们之间的均方
  差等指标也是评价模型的一个指标。
6 RDD<Rating> predictRDD = model.predict(JavaRDD.toRDD(userProducts))
7 //给所有用户推荐10个产品。 包含原始矩阵中的记录
8 RDD<Tuple2<Object, Rating[]>> productsForUsers =
  model.recommendProductsForUsers(10);
9 //给所有产品推荐10个用户。 同样包含原始矩阵中的记录
10 RDD<Tuple2<Object, Rating[]>> usersForProducts =
  model.recommendUsersForProducts(10);
```

4、结果评估与模型优化

从Spark提供的示例当中，应该要注意到，对于推荐矩阵的求解问题，实现的API是非常简单的。但是，示例中有很大部分的工作都是在从各个角度对推荐矩阵训练出来的模型进行评估。

所以，我们在构建一个推荐系统时，对于最核心的推荐算法，往往需要通过调整不同的超参数(例如推荐系统中的隐语义个数、迭代次数、正则化参数)，形成多个推荐模型。然后对这些推荐模型进行多个角度的综合排名。然后选取其中排名比较靠前的多个推荐算法，作为候选算法。

接下来需要在后面的实际业务中，不断观察这些模型的真实预测表现，对这些模型的业务效果进行衡量。而衡量推荐系统效果的方式，主要有三种

- 1、基于常识的评判标准。

2、基于业务的评判标准。

3、基于机器学习的评判标准。

这些在最开始介绍推荐系统时已经给大家介绍过。其中第一点和第二点，还是比较好理解的。但是对于第三点，在开始的时候并没有说得太清楚。而现在，我们看到Spark示例当中的这些指标都可以作为实际的评判标准。

最后，推荐系统的模型并不是一成不变的，并不是说训练出了一个最好的模型，那这个模型就永远是最好的了。所以，在后续业务过程中，还需要结合新的业务数据，对模型继续进行计算、评估、优化这样的过程。

总结：

到这里，我们整个课程就告一段落。在这个课程中，我们提到了推荐算法的一种实现，ALS。并且在推荐系统之外，还带大家了解了机器学习常用的一些方法、工具以及算法。通过这几节课程，希望大家能够对机器学习有一个初步的理解，对于机器学习中常用的这些术语有一个基础的理解，并且能够使用简单的API来解决一些基础的机器学习问题。其实大家也会了解到，对于机器学习领域，工程化实现往往并不难，而且往往随意用机器学习算法一尝试，就能达到一个不错的效果。而难的是对算法的选择以及对模型的调优。

当然，我们这几节课是尽量绕过了复杂的理论知识，只是从工程实现的角度带大家完成快速的入门，还是有很多不足的。一方面，整个推荐系统实施的工程基础还是比较薄弱的。像Hadoop、Spark这些典型的大数据组件还没有了解。另一方面，如果要真正深入把握机器学习的这些方法，这些简单的API背后的理论知识，各种算法的原理，还是绕不开的。最后，在传统机器学习的基础上，还有更强大的基于神经网络的深度学习方法也还没有接触。所以，对于人工智能这个领域，我们还真的只是在门口看了看热闹。在后续的课程中，将会逐步带大家继续深入接触人工智能的精彩。

文档：C4_推荐算法实现与调优.md

链接：<http://note.youdao.com/noteshare?id=b615beca2626549673874c21631eb34a&sub=A5AEB88F4A9046509632012FE906BC06>