Collaborative Filtering Recommendations (协同过滤，简称CF) 是目前最流行的推荐方法，在研究界和工业界得到大量使用。但是，工业界真正使用的系统一般都不会只有CF推荐算法，Content-based Recommendations (CB) 基本也会是其中的一部分。

      CB应该算是最早被使用的推荐方法吧，它根据用户过去喜欢的**产品**（本文统称为 **item**），为用户推荐和他过去喜欢的产品相似的产品。例如，一个推荐饭店的系统可以依据某个用户之前喜欢很多的烤肉店而为他推荐烤肉店。 CB最早主要是应用在信息检索系统当中，所以很多信息检索及信息过滤里的方法都能用于CB中。

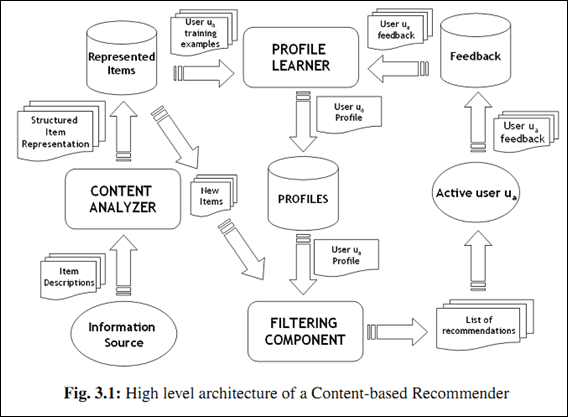
      CB的过程一般包括以下三步：

1. **Item Representation**：为每个item抽取出一些特征（也就是item的content了）来表示此item；

2. **Profile Learning**：利用一个用户过去喜欢（及不喜欢）的item的特征数据，来学习出此用户的喜好特征（profile）；

3. **Recommendation Generation**：通过比较上一步得到的用户profile与候选item的特征，为此用户推荐一组相关性最大的item。

[3]中对于上面的三个步骤给出一张很细致的流程图（第一步对应着Content Analyzer，第二步对应着Profile Learner，第三步对应着Filtering Component）：

[](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405273048.png)

       举个例子说明前面的三个步骤。对于个性化阅读来说，一个item就是一篇文章。根据上面的第一步，我们首先要从文章内容中抽取出代表它们的属性。常用的方法就是利用出现在一篇文章中词来代表这篇文章，而每个词对应的权重往往使用信息检索中的tf-idf来计算。比如对于本文来说，词“CB”、“推荐”和“喜好”的权重会比较大，而“烤肉”这个词的权重会比较低。利用这种方法，一篇抽象的文章就可以使用具体的一个向量来表示了。第二步就是根据用户过去喜欢什么文章来产生刻画此用户喜好的 profile了，最简单的方法可以把用户所有喜欢的文章对应的向量的平均值作为此用户的profile。比如某个用户经常关注与推荐系统有关的文章，那么他的profile中“CB”、“CF”和“推荐”对应的权重值就会较高。在获得了一个用户的profile后，CB就可以利用所有item与此用户profile的相关度对他进行推荐文章了。一个常用的相关度计算方法是cosine。最终把候选item里与此用户最相关（cosine值最大）的N个item作为推荐返回给此用户。

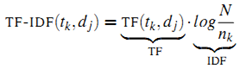
       接下来我们详细介绍下上面的三个步骤。

**一. Item Representation**

      真实应用中的item往往都会有一些可以描述它的属性。这些属性通常可以分为两种：结构化的（structured）属性与非结构化的（unstructured）属性。所谓结构化的属性就是这个属性的意义比较明确，其取值限定在某个范围；而非结构化的属性往往其意义不太明确，取值也没什么限制，不好直接使用。比如在交友网站上，item就是人，一个item会有结构化属性如身高、学历、籍贯等，也会有非结构化属性（如item自己写的交友宣言，博客内容等等）。对于结构化数据，我们自然可以拿来就用；但对于非结构化数据（如文章），我们往往要先把它转化为结构化数据后才能在模型里加以使用。真实场景中碰到最多的非结构化数据可能就是文章了（如个性化阅读中）。下面我们就详细介绍下如何把非结构化的一篇文章结构化。

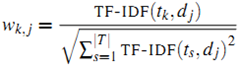
       如何代表一篇文章在信息检索中已经被研究了很多年了，下面介绍的表示技术其来源也是信息检索，其名称为向量空间模型（Vector Space Model，简称VSM）。

      记我们要表示的所有文章集合为 [Image(1)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405286886.png)，而所有文章中出现的词（对于中文文章，首先得对所有文章进行分词）的集合（也称为词典）为[Image(2)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405287932.png)。也就是说，我们有N篇要处理的文章，而这些文章里包含了n个不同的词。我们最终要使用一个向量来表示一篇文章，比如第j篇文章被表示为[Image(3)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405286263.png)，其中[Image(4)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405296754.png)表示第1个词[Image(5)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405303623.png)在文章j中的权重，值越大表示越重要；[Image(6)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405313524.png)中其他向量的解释类似。所以，为了表示第j篇文章，现在关键的就是如何计算[Image6](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405315933.png)各分量的值了。例如，我们可以选取[Image4](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/20120410140532294.png)为1，如果词[Image5](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405332703.png)出现在第 j 篇文章中；选取为0，如果[Image5](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405343161.png)未出现在第j篇文章中。我们也可以选取[Image4](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405358078.png)为词[Image5](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405364947.png)出现在第 j 篇文章中的次数（frequency）。但是用的最多的计算方法还是信息检索中常用的**词频-逆文档频率**（term frequency–inverse document frequency，简称tf-idf）。第j篇文章中与词典里第k个词对应的tf-idf为：

[](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405364881.png)

其中[Image(8)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405373976.png)是第k个词在文章j中出现的次数，而[Image(9)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405374815.png)是所有文章中包括第k个词的文章数量。

      最终第k个词在文章j中的权重由下面的公式获得：

[](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405376942.png)

做归一化的好处是不同文章之间的表示向量被归一到一个量级上，便于下面步骤的操作。

**二. Profile Learning**

       假设用户u已经对一些item给出了他的喜好判断，喜欢其中的一部分item，不喜欢其中的另一部分。那么，这一步要做的就是通过用户u过去的这些喜好判断，为他产生一个模型。有了这个模型，我们就可以根据此模型来判断用户u是否会喜欢一个新的item。所以，我们要解决的是一个典型的有监督分类问题，理论上机器学习里的分类算法都可以照搬进这里。

      下面我们简单介绍下CB里常用的一些学习算法：

**1. 最近邻方法（k-Nearest Neighbor，简称kNN）**

      对于一个新的item，最近邻方法首先找用户u已经评判过并与此新item最相似的k个item，然后依据用户u对这k个item的喜好程度来判断其对此新item的喜好程度。这种做法和CF中的item-based kNN很相似，差别在于这里的item相似度是根据item的属性向量计算得到，而CF中是根据所有用户对item的评分计算得到。

      对于这个方法，比较关键的可能就是如何通过item的属性向量计算item之间的两两相似度。[2]中建议对于结构化数据，相似度计算使用欧几里得距离；而如果使用向量空间模型（VSM）来表示item的话，则相似度计算可以使用cosine。

**2. Rocchio算法**

[Rocchio算法](http://en.wikipedia.org/wiki/Rocchio_algorithm)是信息检索中处理相关反馈（Relevance Feedback）的一个著名算法。比如你在搜索引擎里搜“苹果”，当你最开始搜这个词时，搜索引擎不知道你到底是要能吃的水果，还是要不能吃的苹果，所以它往往会尽量呈现给你各种结果。当你看到这些结果后，你会点一些你觉得相关的结果（这就是所谓的相关反馈了）。然后如果你翻页查看第二页的结果时，搜索引擎可以通过你刚才给的相关反馈，修改你的查询向量取值，重新计算网页得分，把跟你刚才点击的结果相似的结果排前面。比如你最开始搜索“苹果”时，对应的查询向量是{“苹果” : 1}。而当你点击了一些与Mac、iPhone相关的结果后，搜索引擎会把你的查询向量修改为{“苹果” : 1, “Mac” : 0.8, “iPhone” : 0.7}，通过这个新的查询向量，搜索引擎就能比较明确地知道你要找的是不能吃的苹果了。Rocchio算法的作用就是用来修改你的查询向量的：{“苹果” : 1}  --> {“苹果” : 1, “Mac” : 0.8, “iPhone” : 0.7}。

      在CB里，我们可以类似地使用Rocchio算法来获得用户u的profile[Image(11)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/20120410140538779.png)：

[Image(12)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405391270.png)

其中[Image(13)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405399285.png)表示item j的属性，[Image(14)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405396187.png)与[Image(15)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/2012041014054025.png)分别表示已知的用户u喜欢与不喜欢的item集合；而[Image(16)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405414942.png)与[Image(17)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405424320.png)为正负反馈的权重，它们的值由系统给定。

      在获得[Image11](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405425890.png)后，对于某个给定的item j，我们可以使用[Image(11)[1]](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405424777.png)与[Image(13)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405432236.png)的相似度来代表用户u对j的喜好度。

      Rocchio算法的一个好处是[Image(11)[1]](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405431331.png)可以根据用户的反馈实时更新，其更新代价很小。

      正如在本节开头所说，本节要解决的是一个典型的有监督分类问题。所以各种有效的分类机器学习算法都可以用到这里，下面列举几个常用的分类算法：

**3. 决策树算法（Decision Tree，简称DT）**

      当item的属性较少而且是结构化属性时，决策树一般会是个好的选择。这种情况下决策树可以产生简单直观、容易让人理解的结果。而且我们可以把决策树的决策过程展示给用户u，告诉他为什么这些item会被推荐。但是如果item的属性较多，且都来源于非结构化数据（如item是文章），那么决策树的效果可能并不会很好。

**4. 线性分类算法（Linear Classifer，简称LC）**

      对于我们这里的二类问题，线性分类器（LC）尝试在高维空间找一个平面，使得这个平面尽量分开两类点。也就是说，一类点尽可能在平面的某一边，而另一类点尽可能在平面的另一边。

      仍以学习用户u的分类模型为例。[Image(13)[5]](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405434122.png)表示item j的属性向量，那么LC尝试在[Image(13)[5]](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405448233.png)空间中找平面[Image(18)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/20120410140544152.png)，使得此平面尽量分开用户u喜欢与不喜欢的item。其中的[Image(19)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405446215.png)就是我们要学习的参数了。最常用的学习[Image19](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405452594.png)的方法就是梯度下降法了，其更新过程如下：

[Image(20)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405458973.png)

其中的上角标t表示第t次迭代，[Image(21)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/20120410140545543.png)表示用户u对item j的打分（例如喜欢则值为1，不喜欢则值为-1）。[Image(22)](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405461001.png)为学习率，它控制每步迭代变化多大，由系统给定。

     和Rocchio算法一样，上面更新公式的好处就是它可以以很小的代价进行实时更新，实时调整用户u对应的[Image19](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405476267.png)。

     说到这里，很多童鞋可能会想起一些著名的线性分类器：Logistic Regression和Linear SVM等等，它们当然能胜任我们这里的分类任务。[2]中提到Linear SVM用在文本分类上能获得相当不错的效果:)。

     如果item属性[Image(13)[5]](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405472330.png)的每个分量都是0/1取值的话（如item为文章，[Image(13)[5]](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405472854.png)的第k个分量为1表示词典中第k个词在item j中，为0表示第k个词不在item j中），那么还有一种很有意思的启发式更新[Image19](http://images.cnblogs.com/cnblogs_com/breezedeus/201204/201204101405473693.png)的算法：[Winnow算法](http://en.wikipedia.org/wiki/Winnow_(algorithm))。[4]中就是使用Winnow算法来获得user profile的。

**5. 朴素贝叶斯算法（Naive Bayes，简称NB）**

      NB算法就像它的简称一样，牛逼！NB经常被用来做文本分类，它假设在给定一篇文章的类别后，其中各个词出现的概率相互独立。它的假设虽然很不靠谱，但是它的结果往往惊人地好。再加上NB的代码实现比较简单，所以它往往是很多分类问题里最先被尝试的算法。我们现在的profile learning问题中包括两个类别：用户u喜欢的item，以及他不喜欢的item。在给定一个item的类别后，其各个属性的取值概率互相独立。我们可以利用用户u的历史喜好数据训练NB，之后再用训练好的NB对给定的item做分类。NB的介绍很多，这里就不再啰嗦了，有不清楚的童鞋可以参考[NB Wiki](http://en.wikipedia.org/wiki/Naive_Bayes_classifier)，或者[1-3]。

**三. Recommendation Generation**

      如果上一步Profile Learning中使用的是分类模型（如DT、LC和NB），那么我们只要把模型预测的用户最可能感兴趣的n个item作为推荐返回给用户即可。而如果Profile Learning中使用的直接学习用户属性的方法（如Rocchio算法），那么我们只要把与用户属性最相关的n个item作为推荐返回给用户即可。其中的用户属性与item属性的相关性可以使用如cosine等相似度度量获得。

     下面说说CB的优缺点。

**CB的优点**：

1. 用户之间的独立性（User Independence）：既然每个用户的profile都是依据他本身对item的喜好获得的，自然就与他人的行为无关。而CF刚好相反，CF需要利用很多其他人的数据。CB的这种用户独立性带来的一个显著好处是别人不管对item如何作弊（比如利用多个账号把某个产品的排名刷上去）都不会影响到自己。

2. 好的可解释性（Transparency）：如果需要向用户解释为什么推荐了这些产品给他，你只要告诉他这些产品有某某属性，这些属性跟你的品味很匹配等等。

3. 新的item可以立刻得到推荐（New Item Problem）：只要一个新item加进item库，它就马上可以被推荐，被推荐的机会和老的item是一致的。而CF对于新item就很无奈，只有当此新item被某些用户喜欢过（或打过分），它才可能被推荐给其他用户。所以，如果一个纯CF的推荐系统，新加进来的item就永远不会被推荐:( 。

**CB的缺点**：

1. item的特征抽取一般很难（Limited Content Analysis）：如果系统中的item是文档（如个性化阅读中），那么我们现在可以比较容易地使用信息检索里的方法来“比较精确地”抽取出item的特征。但很多情况下我们很难从item中抽取出准确刻画item的特征，比如电影推荐中item是电影，社会化网络推荐中item是人，这些item属性都不好抽。其实，几乎在所有实际情况中我们抽取的item特征都仅能代表item的一些方面，不可能代表item的所有方面。这样带来的一个问题就是可能从两个item抽取出来的特征完全相同，这种情况下CB就完全无法区分这两个item了。比如如果只能从电影里抽取出演员、导演，那么两部有相同演员和导演的电影对于CB来说就完全不可区分了。

2. 无法挖掘出用户的潜在兴趣（Over-specialization）：既然CB的推荐只依赖于用户过去对某些item的喜好，它产生的推荐也都会和用户过去喜欢的item相似。如果一个人以前只看与推荐有关的文章，那CB只会给他推荐更多与推荐相关的文章，它不会知道用户可能还喜欢数码。

3. 无法为新用户产生推荐（New User Problem）：新用户没有喜好历史，自然无法获得他的profile，所以也就无法为他产生推荐了。当然，这个问题CF也有。

       CB应该算是第一代的个性化应用中最流行的推荐算法了。但由于它本身具有某些很难解决的缺点（如上面介绍的第1点），再加上在大多数情况下其精度都不是最好的，目前大部分的推荐系统都是以其他算法为主（如CF），而辅以CB以解决主算法在某些情况下的不精确性（如解决新item问题）。但CB的作用是不可否认的，只要具体应用中有可用的属性，那么基本都能在系统里看到CB的影子。组合CB和其他推荐算法的方法很多（我很久以后会写一篇博文详细介绍之），最常用的可能是用CB来过滤其他算法的候选集，把一些不太合适的候选（比如不要给小孩推荐偏成人的书籍）去掉。