懒惰啊我

自己动手写一个推荐系统

废话：

最近朋友在学习推荐系统相关，说是实现完整的推荐系统，于是我们三不之一会有一些讨论和推导，想想索性整理出来。

在文中主要以工程中做推荐系统的流程着手，穿插一些经验之谈，并对于推荐系统的算法的学术界最新的研究进展和流派作一些介绍。当然由于我做推荐系统之时还年幼，可能有很多偏颇甚至错误的见解，就当抛砖引玉，还请各位大大指点。

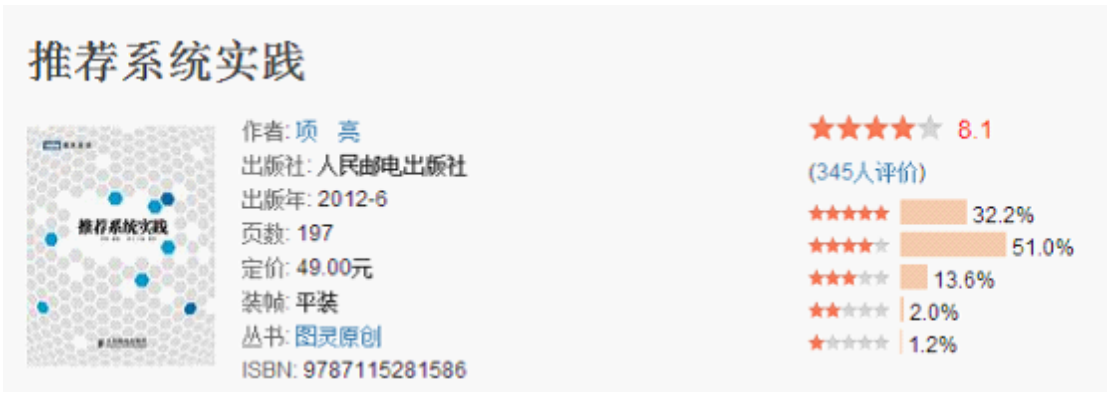
Reading lists

虽然很多人觉得作为AI的分支之一，推荐跟自然语言处理等问题的难度不可同日而语。但所谓磨刀不误砍柴工，我觉得，至少在开工前先应该阅读这样基本书，起码要看看目录，以对于推荐系统有个初步的了解。

中文书籍：

1.《推荐系统实践》项亮http://book.douban.com/subject/10769749/

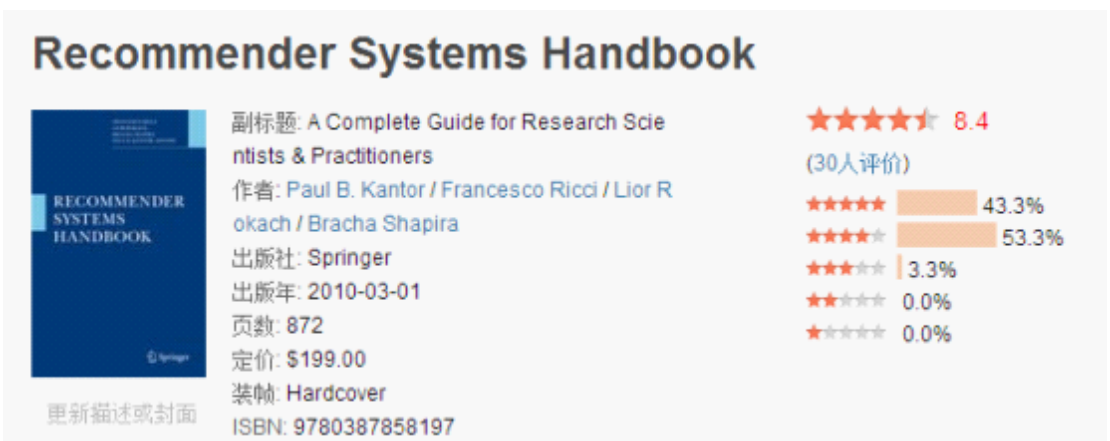
这本书你说他好吧，不如recommend handbook；你说他不好吧，确实又把很多基础而简单的问题讲的很详细，而且还是中文的。薄薄的一本书，分分钟可以翻完，总而言之，是一本入门的好书。



外文书籍：

1.《**Recommender Systems Handbook**》Paul B. Kantor http://book.douban.com/subject/3695850/

其实所有敢自称handbook的书都是神书。这本书写的非常细，而且非常全，如果你去看一些推荐系统的一些比较冷门的topic的paper，比如fighting spam的，甚至能发现很多paper就是直接引用这本书里相关章节的内容。可以说这本书是做推荐同学必备的枕边书，没看过这本书出去吹牛逼时你都不好意思说自己是做推荐的，不过说实在，真正看完的没几个，一般是用到哪里查哪里，我刚才就去豆瓣验证了，几个做推荐的都是在读，一群文艺青年都是想读。此书唯一的缺点是没有出新版，有些地方已经成为时代的眼泪了。



公告

昵称：懒惰啊我
园龄：4年7个月
粉丝：139
关注：4
+加关注

<	2013年3月					>
日	一	二	三	四	五	六
24	25	26	27	28	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

搜索

找找看

谷歌搜索

常用链接
我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签
数据挖掘(4)
推荐系统(2)
facebook(1)
读书笔记(1)
美国大选(1)

随笔档案
2015年1月 (1)
2013年3月 (1)
2012年12月 (4)
2012年11月 (3)

最新评论
1. Re:漫谈数据挖掘从入门到进阶
感谢博主的分享！刚接触数据挖掘不久，这篇文章对我来说很有参考价值！

--青紫随风

2. Re:漫谈数据挖掘从入门到进阶
对于我准备入门的程序员来说挺好，楼主谦虚，有建群么？

--晚来居上

3. Re:当推荐算法开源包多如牛毛，为什么我们还要专门的推荐算法工程师
作为一个搞算法的，我也曾经遇到过这种困惑！！

--Leo_羽佳

4. Re:自己动手写一个推荐系统
哈哈哈 楼主是大神，在滴滴年薪40+

--合唱团abc

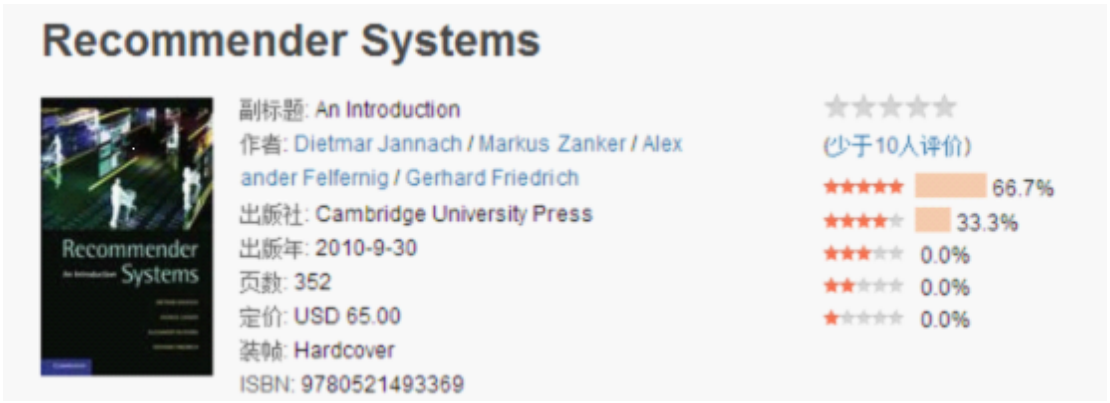
5. Re:当推荐算法开源包多如牛毛，为什么我们还要专门的推荐算法工程师
@南亭坡根据特征判断用户会不会进行操作（购买、点击），概率大的就进行推荐...

--sky88088

2.《**Recommender Systems - An Introduction**》 Dietmar Jannach

http://book.douban.com/subject/5410320/

跟上面一本差不多，学院派的综述型的书，厚厚一本，不看的时候当枕头用。



3.《**mahout in action**》Sean Owen http://book.douban.com/subject/4893547/

上面的一中一外的书都是理论基础的书，这本书就和工程有些沾边。如果你要用mahout框架来做推荐系统，那么是必须要扫一遍的；如果你不用mahout，看看其中的流程和代码组织也是有一定好处的。

Paper：

由于《Recommender Systems Handbook》很多地方已经成为了时代的眼泪，这里推荐一些综述，以便开阔眼界。

一篇是**physics report**上面的**recommend system**这篇综述，可以说是最新最全的综述了，看完后学术界都在折腾啥分分钟就清楚了。http://arxiv.org/pdf/1202.1112v1.pdf

比较经典的是**recommend system**的**state of art**这篇综述，很多老一辈的同志喜欢推荐的，说实在这篇因为年代久远，也已经成为时代的眼泪了。

开工：

上面给了一些reading lists，并不是说要读完所有的才能开工，只是说，先看完个目录，哪里不懂查哪里。在下面介绍的做推荐系统的流程中，我只是想给大家介绍个普通的推荐系统该怎么做，所以很多地方都有偷懒，还请大家见谅。而且由于我不是做的在线的推荐系统，而是属于隔天推荐的那种离线的，所以叙述工程的时候就是只叙述离线的推荐。

数据准备：

一般来说，做推荐系统的数据一般分两种，一种从在线的读取，比如用户产生一个行为，推荐系统就反应下（传说豆瓣fm就是这么做的？），还有一种就是从数据库里读。

我个人一般是这样做的：跟做反作弊的人打个招呼，让他们在记用户行为数据的时候顺便存到各个线上服务器上，再写个php脚本，从各个服务器上把我需要的日志抓过来，然后当日要的最新的数据就来了。

但是这种地方其实存储的数据是加了一些判断的，也就是说是分类记录的（因为很多记录是别人刷的，比如丢一个链接到qq群里让别人帮忙投票什么的），这里不细说，到后面fighting spam的地方再讨论。

阅读排行榜
1. 自己动手写一个推荐系统(28533)
2. 漫谈数据挖掘从入门到进阶(15505)
3. 当推荐算法开源包多如牛毛，为什么我们还要专门的推荐算法工程师(13890)
4. Weka中BP神经网络的实践（参数调整以及结果分析）(13821)
5. 谷歌中的数据挖掘应用->statisticians都在google干嘛(3101)

评论排行榜
1. 漫谈数据挖掘从入门到进阶(31)
2. 自己动手写一个推荐系统(16)
3. 当推荐算法开源包多如牛毛，为什么我们还要专门的推荐算法工程师(14)
4. 谷歌中的数据挖掘应用->statisticians都在google干嘛(10)
5. 数据挖掘和统计学的区别（guide to Intelligent data analysis学习笔记）(6)

推荐排行榜
1. 当推荐算法开源包多如牛毛，为什么我们还要专门的推荐算法工程师(23)
2. 漫谈数据挖掘从入门到进阶(15)
3. 自己动手写一个推荐系统(8)
4. 美国大选数据挖掘相关论文笔记（A 61-million-person experiment in social influence and political mobilization）(2)
5. Weka中BP神经网络的实践（参数调整以及结果分析）(2)

数据过滤：

当我们得到了每天产生的数据后，说实在这些数据实在是太多了，我们当然用不到这么多，就要写个过滤模块，把一些我们用不到的数据过滤掉。

我一般是这样做的：写个python的脚本，把过滤器放到一个单独的模块，要用的过滤器就到责任链里面注册下。这样别人和自己维护起来也方便点，顺便一说，过滤的东西一般来说有这样几种：一种是一个item只有一个user打过分的，而且以前没有人打分的，这样的数据放到推荐的模型里去跑虽然mahout会自动无视它，但其实按照power law来说是有不少的，内存能省就省吧；还有一种是有些黑名单的，有item和user各自的黑名单，这些也是事先要去掉的。

数据存储：

这个就是大家仁者见仁，智者见智的时候了，因为一般来说，是你选用的算法和算法具体的实现方式以及基础架构决定了你的存储方式，就不多说了。

我一般是这样做的：一般做成增量处理的方式，然后每天一计算，一周一回滚。由于算法实现的特殊性，每40个item user对存储在一起，有点类似于bitmap吧。

推荐系统算法部分：

这部分以前写过类似的小记录和心得笔记之类的东西，就直接贴了_(:3」∠)_

这里的推荐系统的核心算法主要用**mahout**实现。

各种算法对于推荐都有着自己的特定的假设，对于什么时候什么样的算法会有比较好的performance应该对于假设反复验证。说白了就是做实验。

然后我们一般用什么算法呢，看看mahout给的算法吧，花样繁多，什么item based，user based，slop-one，SVD等等，常用的都有了，那我们要用什么算法呢。

先简单说下**user based**的算法在**mahout**中的一些实现：

第一步应该先算出所有人的相似度矩阵W，再去对于item进行遍历，事实上mahout也是这样做的。

相似度矩阵也许可以保留下来，这样可以用来做谱聚类来验证。

UserSimilarity 封装了用户之间的相似性

UserSimilarity similarity = new PearsonCorrelationSimilarity (model);

UserNeighborhood封装了最相似的用户组

UserNeighborhood neighborhood = new NearestNUserNeighborhood (2, similarity, model);

总而言之，用DataModel来生成data model，用UserSimilarity生成User-user之间的相似度矩阵，用户的邻居的定义用UserNeighborhood定义，推荐引擎使用Recommender实现。

对于推荐的评判是使用evaluator来进行评判

double score = evaluator.evaluate(recommenderBuilder, null, model, 0.95, 0.05);

用95%的数据构建模型，用5%的数据进行test

Fixed-size neighborhoods

对于到底用多少人作为用户周围的一圈，我们事实上没有一个确定的值，就像做生物实验一样需要不断在特定的数据集里跑出来。

Threshold-based neighborhood

当然，我们也可以定义个threshold去找出最相似的用户群。

Threshold定义为-1到1（相似度矩阵返回的相似度就是在这个范围）

new ThresholdUserNeighborhood(0.7, similarity, model)

我们对于各个算法做个简单的比（吐）较（槽）：

（假设我们是要像亚马逊一样对商品做推荐，然后我们最终是要做top k的推荐）

Item based

一般来说item-based要跑得快一些，因为item比user少

Slop one

说实在话我觉得在对于一些个人品味比较看重的东西上这个算法不是很靠谱

但是它在GroupLens 10 million数据的结果是0.65

当然，对于股票系统这种还是可取的

这个算法的假设是对于不同item的preference有种线性关系

不过slope-one有个好处是它的online算的很快，并且它的性能不由用户的数量决定

在mahout中的调用方法是new SlopeOneRecommender(model)

这个方法提供了两种weight：weighting based on count and on standard deviation

count 是越多的user的给越多的权重，对出的是一个加权的平均数

standard deviation则是越低的deviation给予越高的权重

这两个weight是默认采用的，当然disable它们只会让结果变得稍微坏一点0.67

不过这个算法有个很明显的缺点是比较占内存

但是幸运的是我们可以把它存在数据库里：MySQLJDBCDataModel

Singular value decomposition–based recommenders

事实上，尽管**SVD**失去了一些信息，但是有时候它可以改善推荐的结果。

这个过程在有用的方式平滑了输入

new SVDRecommender(model, new ALSWRFactorizer(model, 10, 0.05, 10))

第一个参数10是我们的目标属性个数

第二个属性是lambda->regularization

最后一个参数是training step跑的次数

KnnItemBasedRecommender

囧，事实上这个是用的knn的方式来做的算法，和前面的选择一个threshold然后圈画用户的算法是比较像的

但是用knn代价是非常高的，因为它要去比较所有的items的对

ItemSimilarity similarity = new LogLikelihoodSimilarity(model);

Optimizer optimizer = new NonNegativeQuadraticOptimizer();

return new KnnItemBasedRecommender(model, similarity, optimizer, 10);

结果也还不算差，是0.76

Cluster-based recommendation

基于聚类的推荐可以说是基于用户的推荐的算法的变体中最好的一种思路

对于每一个聚类里面的用户进行推荐

这个算法在推荐里面是非常快的，因为什么都事先算好了。

这个算法对于冷启动还是挺不错的

感觉mahout里面用的聚类算法应该类似于Kmeans？

TreeClusteringRecommender

UserSimilarity similarity = new LogLikelihoodSimilarity(model);

ClusterSimilarity clusterSimilarity =

new FarthestNeighborClusterSimilarity(similarity);

return new TreeClusteringRecommender(model, clusterSimilarity, 10);

注意的是两个cluster之间的相似度是用ClusterSimilarity来定义的

其中cluster之间的相似度还有NearestNeighborClusterSimilarity可选

吐槽：

对于算法的选择，其实我们是要和我们要推荐的目标挂钩的。为什么最近学术界搞SVD那一系的算法这么火，什么LDA，plsa各种算法层出不穷，事实上因为netflix的要求是要优化RMSE，在机器学习的角度来看，类似于回归问题，而工业界的角度来说，我们一般的需求是做top k的推荐，更加类似于分类问题。所以为什么相比用SVD系列的算法，用item based这种更加ad hoc的算法要表现的更好一些。当然2012年的KDD cup第一名的组用的是item based+SVD的算法，这是后话。

那么我就假设解我们这个top k的商品推荐的问题就用item based好了（速度快，结果又不算差），接下来就是确定相似度了。

相似度确定：

我们对于各个相似度做一些比（吐）较（槽）：

PearsonCorrelationSimilarity

Pearson correlation：

coeff = corr(X , Y);

function coeff = myPearson(X , Y)

% 本函数实现了皮尔逊相关系数的计算操作

%

% 输入：

% X：输入的数值序列

% Y：输入的数值序列

%

% 输出：

% coeff：两个输入数值序列X，Y的相关系数

%

if length(X) ~= length(Y)

  error('两个数值数列的维数不相等');

  return;

end

```
fenzi = sum(X .* Y) - (sum(X) * sum(Y)) / length(X);

fenmu = sqrt((sum(X .^2) - sum(X)^2 / length(X)) * (sum(Y .^2) - sum(Y)^2 / length(X)));

coeff = fenzi / fenmu;


end %函数myPearson结束
```

当两个变量的标准差都不为零时，相关系数才有定义，皮尔逊相关系数适用于：

(1)、两个变量之间是线性关系，都是连续数据。

(2)、两个变量的总体是正态分布，或接近正态的单峰分布。

(3)、两个变量的观测值是成对的，每对观测值之间相互独立。

1.没有考虑到用户偏好重合的items的数量

2,只有一个item是相交错的话是不能得到correlation的，对于比较稀疏或者小的数据集这是个要注意的问题。不过一般来说两个用户之间只有一个item重叠从直觉上来说也不那么相似

Pearson correlation一般出现在早期的推荐的论文和推荐的书里，不过并不总是好的。

在mahout中使用了一个增加的参数Weighting.WEIGHTED，适当使用可以改善推荐结果

EuclideanDistanceSimilarity

Return 1 / (1 + d)

CosineMeasureSimilarity

当two series of input values each have a mean of 0(centered)时和PearsonCorrelation是有相同结果的

所以在mahout中我们只要简单的使用PearsonCorrelationSimilarity就好

Spearman correlation

这个方法用rank的方式，虽然失去了具体的打分信息，不过却保留了item的order

Return的结果是-1和1两个值，不过和pearson一样，对于只有一个item重叠的也算不出。

而且这个算法比较慢，因为它要算和存储rank的信息。所以paper多而实际用的少，对于小数据集才值得考虑

CachingUserSimilarity

```
UserSimilarity similarity = new CachingUserSimilarity(

new SpearmanCorrelationSimilarity(model), model);
```

Ignoring preference values in similarity with the Tanimoto coefficient

TanimotoCoefficientSimilarity

如果一开始就有preference value，当数据中signal比noise多的时候可以用这个方法。

不过一般来说有preference信息的结果要好。

log-likelihood

Log-likelihood try to access how unlikely 这些重叠的部分是巧合

结果的值可以解释为他们的重合部分并不是巧合的概念

算法的结果可能比tanimoto要好，是一个更加聪明的metric

Inferring preferences

对于数据量比较小的数据，pearson很难handle，比如一个user只express一个preference

于是要估计相似度么.....

AveragingPreferenceInferrer

setPreferenceInferrer().

不过这种办法在实际中并不是有用的，只是在很早的paper中mention到

通过现在的信息来估计并不能增加什么东西，并且很大的降低了计算速度

最终我们要通过实验来比较上面的相似度，一般来说是用准确率，召回率，覆盖率评测。

这里有一篇**Building Industrial-scale Real-world Recommender Systems**

http://vdisk.weibo.com/s/rMSj-

写netflex的，非常好，我就不献丑多说了，所以上面只是吐槽下常见的算法和相似度。

其实算法按流派分是分为下面这些类别的，大家有兴趣也可以了解下，我这里就不多做介绍：

Similarity-based methods

Dimensionality Reduction Techniques

Dimensionality-based methods

Diffusion-based methods

Social fltering

Meta approaches

我上面所说的相似度和推荐算法，只能算是Similarity-based methods和 Dimensionality Reduction Techniques里的非常小的一小部分，只当抛砖引玉了。

Ps：刚在豆瓣上问了下，他们说就用前两种，事实上我也是觉得协同过滤+SVD 偶尔做做 topic model就够了 如果没事干再上点social trusted的东西

增加规则：

记得hulu在做presentation的时候说过“不能做规则定制的推荐系统不是一个好的推荐系统”（好像是这样吧.....）事实上我们做出来的推荐的结果是需要做很多处理再展现给用户的，这时候就是需要增加规则的时候了。

1.改善推荐效果：有些协同过滤的算法，做出来的结果不可避免的产生一些让人啼笑皆非的结果，比如用户什么都买，导致你有可能跟姑娘们推荐的时候在佛祖下面推荐泳装什么的（真实的故事）。这时候就有两种办法，一种就是调整模型，一种就是增加规则做一定的限制。再举个常见的例子就是有时候会在推荐冬装的时候出现夏装什么的，这时候一般我的处理方式是给这种季节性商品做一个随时间的衰退。

2.增加广告和导向：插入广告，我们的最爱，这个就不多说了，靠规则。而所谓用户喜欢的，其实不一定就是最好的，比如用户一般来说就喜欢便宜的，还有什么韩流爆款之流，如果你推出来的东西都是这样的，整个系统就变成洗剪吹大集合了，非常影响定位，这时候也要上规则。

3.做一些数据挖掘和fighting spam的工作：这个在fighting spam的地方细说

可视化参数调整：

做完上面的工作，一般来说推荐系统的基础架构就差不多了，但是往往各个算法以及你自己上的规则都有多如牛毛的参数要调整，这时候一般要自己写个测试脚本，将调整的结果可视化下一下，我个人推荐的是highchart，看参数以及比较各个指标非常清爽， 还可以自己做一些比如是取log之类的定制，很是方便。http://www.highcharts.com/

调整参数以及上线：

上线前有两个事要做，一般来说就是离线测试和AB test。

离线测试就是把数据抽样一部分，分为train data和test data，然后评估一些准确率，召回率以及coverage之类的指标，用上面的可视化工具观测比较下，感觉差不多了把pm叫过来让她给小编们看看，看看审美和效果之类的。这是比较粗糙的，有的地方做的比较过细，还有用户调研和请一些人来实际使用，这是后话。

AB test也是大家最喜欢的地方了。因为说实在，评估推荐系统学术界是看准确率那一些东西，但是工业界还是看pv uv转化率这种实打实带来效益的东西，而AB test就是评估这些的。我个人是比较推荐这种方法，说不上好，只是抛砖引玉，就是按一般的做法先空跑一个星期，然后再把系统上线实际做算法pk，然后选取的实验用户一半的几率进入原来的算法的推荐，一半的几率进入你的算法的推荐，每天看看转化率之间的比较，顺便还可以调下参数做做实验。如果算法稳定表现较好，就差不多了。

Fighting spam :

俗话说，有人的地方就有江湖，有推荐的地方就有人刷。刷子一般分三种类型的：average random和nuke。一般来说，average和random比较好对付，只要你的系统鲁棒性好点，基本影响不大。但是nuke的就非常烦，一般来说有两种思路解决，一种是提高系统鲁棒性，另外就是上规则。我们从这张图看看两种思路分布对应的解决效果：

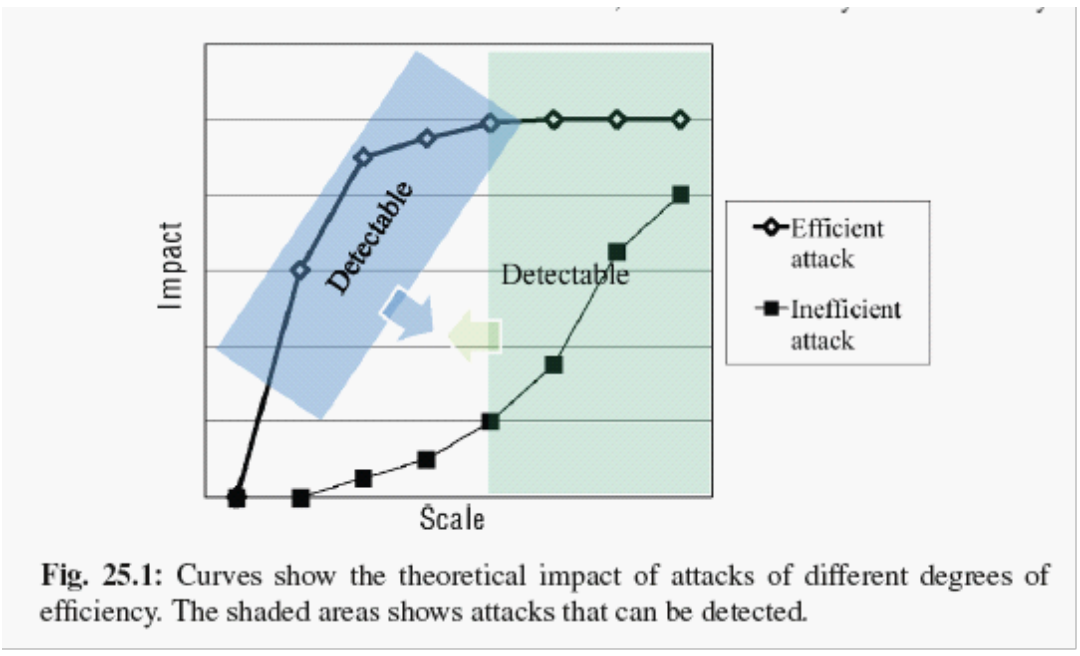


Fig. 25.1: Curves show the theoretical impact of attacks of different degrees of efficiency. The shaded areas shows attacks that can be detected.

其实鲁棒性的系统做的就是把efficient attack的曲线降低，其实效果不算太好。

规则就是做到提前检测，将危险扼杀在摇篮之中，就是做的蓝色的那块detectable的部分。

Fighting spam是个博大精深的问题，俗话说，与人斗，其乐无穷，就是说的这个意思。

我们从规则说来，一般来说规则可以放到最前面的数据收集和过滤的阶段，比如在收集数据的时候看看这个人是否是多个账号但是是一个ip，或者有些人用户名或者注册邮箱有群体相似性质，或者有没有出现pv等不正常的情况。有时候我们也可以从推荐的结果里上规则来查，比如有些人刷的过火了，导致推荐结果出现一些问题，我们就可以用迭代的方式按照先验的刷子的比例来排出刷子排行榜之类的东西。这些都是经验之谈，上不了台面，大家也可以自己摸索。

结束：

上面啰嗦了半天，大体上做一个完整的简单推荐系统就是涉及到上面这些步骤。一些地方说的不够详细，有些是因为我懒，有些是不方便说。大家觉得我说的不对或者不妥的地方，可以直接在下面跟帖喷，或者有大大指导我也是很欢迎的，大家多交流经验。我的邮箱是

flclain@gmail.com 豆瓣是http://www.douban.com/people/45119625/ 有什么问题也可以豆瓣或者邮件交流。

其实我是觉得，要是有个大大说“你个傻缺，写的狗屁不通，让我来教你我是怎么做推荐的”就更好了。

标签: 推荐系统

懒惰啊我
关注 - 4
粉丝 - 139
+加关注

8 2

« 上一篇：Weka中BP神经网络的实践（参数调整以及结果分析）
» 下一篇：当推荐算法开源包多如牛毛，为什么我们还要专门的推荐算法工程师

posted @ 2013-03-03 12:53 懒惰啊我 阅读(28532) 评论(16) 编辑 收藏

评论列表

#1楼 2013-03-03 14:29 Jany_zhang
理论较多。。。。没看到实战。。。
支持(0) 反对(0)

#2楼[楼主] 2013-03-03 14:42 懒惰啊我
@ Jany_zhang
其实要是帖代码的话我项目的源代码都有 光过滤器这个模块就可以帖10几页 但是总觉得推荐系统这种还是方法论多一些 所谓code is cheap, show me the method 说的就是这类学科
支持(2) 反对(0)

#3楼 2013-03-03 17:46 *深海
学习了，有空好好交流交流
支持(0) 反对(0)

#4楼 2013-03-05 09:55 sunlovesea
如果发源代码，就是我们程序猿的福利了
支持(0) 反对(0)

#5楼 2013-05-27 14:38 mlw2000
我想知道类似京东里的：买了xx东西的用户还买了xx。。。。这个是哪个推荐：
1. 获取所有相似的物品，然后计算每个的相似度后排序得出;
2. 对物品聚类，然后计算同一个簇中的所有物品的相似度后排序；
3. 其他？
支持(0) 反对(0)

#6楼 2013-11-21 10:19 huofootball
楼主好，我初学SVD，有个小疑问：既然要分解矩阵式个稀疏的，有很多缺失的评分，又怎么能进行奇异值分解呢？
支持(0) 反对(0)

#7楼 2014-02-19 15:15 目 留
@ huofootball

有几种方法：

1，缺失的全部填平均值。

2，用其他方法预测出缺失值的最可能值，然后填充。

3，梯度下降法
支持(0) 反对(0)

#8楼 2014-02-19 15:17 目 留
@ mlw2000
有几种可能的方法：

1，user-based或者item-based

2，就你这个问题而言，每个item维护了一个user list，user list里的每个user都购买了这个

item。然后在user的item-list里随便找一个item，就可以达到你的目的了：买了xx东西的用户还买了xx。。。。

支持(0) 反对(0)

#9楼 2014-02-19 15:18 目 留

@ sunlovesea

源码贴出来其实非常占用篇幅

其次，学推荐系统，算法、模型、数据是非常重要的，代码反而是其次。再加上有mahout，在理解了相应模型的情况下，只要你会java，写出代码，毫无压力。

应该追求对模型的透彻理解以及业务需求的把握，也就是根据业务不同，选择不同的合适的算法和模型。这些能力，更重要。

支持(1) 反对(0)

#10楼 2014-10-24 18:08 李永辉

@ mlw2000

买了还买是基于关联规则的算法。

浏览了还买是相似

支持(0) 反对(0)

#11楼 2014-10-24 18:09 李永辉

@ 目 留

支持

支持(0) 反对(0)

#12楼 2015-01-08 20:09 火星十一郎

求源代码 不要光说不做

支持(0) 反对(0)

#13楼[楼主] 2015-01-08 20:26 懒惰啊我

@ DM张朋飞

在公司做的，还有我也没有开源的习惯

支持(0) 反对(0)

#14楼 2015-09-09 11:22 lazy6

item-based,计算完相似性之后的部分呢，具体怎么推荐。。。

支持(0) 反对(0)

#15楼 2015-10-28 09:10 火星十一郎

求源代码 不要光说不做

支持(0) 反对(0)

#16楼 2016-08-18 15:58 合唱团abc

哈哈哈 楼主是大神，在滴滴年薪40+

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

最新IT新闻:

· 淘宝商家“大招”够狠：买家好评率不够拒绝下单

· 郭台铭：亚马逊苹果将和鸿海共同出资竞购东芝半导体业务

· iOS 11今晚正式发布！苹果送开发者服饰大礼包

· 京东为五年以上员工医药费买单！章泽天：京东向来以人文关怀至上

· 小米贷款ABS首尝备案制 30亿元专项计划获上交所确认

» 更多新闻...

最新知识库文章:

· 程序员的工作、学习与绩效
· 软件开发为什么很难
· 唱吧DevOps的落地，微服务CI/CD的范本技术解读
· 程序员，如何从平庸走向理想？
· 我为什么鼓励工程师写blog

» 更多知识库文章...