实时推荐系统

目录

[实时推荐系统 1](#_Toc513733498)

[1实施推荐系统介绍与实例 1](#_Toc513733499)

[1.1算法 2](#_Toc513733500)

[1.1.1聚类技术 2](#_Toc513733501)

[1.1.2实时协同过滤算法 3](#_Toc513733502)

[1.2架构 3](#_Toc513733503)

[1.2.1基于Spark的方式 3](#_Toc513733504)

[1.2.2基于Kiji框架的方式 4](#_Toc513733505)

[1.2.3基于Storm的方式 4](#_Toc513733506)

[2评价指标 6](#_Toc513733507)

[2.1准确性指标 7](#_Toc513733508)

[2.2 非准确性指标 8](#_Toc513733509)

[2.2.1用户级非准确性指标 8](#_Toc513733510)

[2.2.2系统级的非准确性指标 9](#_Toc513733511)

推荐系统在垂直领域有了不少突破性进展，但是在全品类的电商、综合的广告营销上，进展还是缓慢，仍然有很多的工作需要探索。特别是在全品类的电商中，单个模型在母婴品类的效果还比较好，但在其他品类就可能很差，很多时候需要根据品类、推荐栏位、场景等不同，设计不同的模型。同时由于用户、SKU不停地增加，需要定期对数据进行重新分析，对模型进行更新，但是定期对模型进行更新，无法保证推荐的实时性，一段时间后，由于模型训练也要相当时间，可能传统的批处理的Hadoop的方法，无法再缩短更新频率，最终推荐效果会因为实时性问题达到一个瓶颈。

对于用户数量远远大于产品，并且产品相对稳定的电商系统，计算产品相似度计算量小，适用Item CF；反之，对于更新频繁，物品数量海量的新闻、博客、微博等系统，User CF效果更好。目前的商用推荐系统，当用户数和商品数达到一定数目时，推荐算法都面临严重的可扩展性问题，推荐的实效性变得非常差，如何在算法和架构上提高推荐速度是很多公司不得不思考的问题。说是实时推荐，其实更多的还是在离线的基础上做增量运算，也就是说实时推荐是和离线推荐完全分开的。

下面介绍一些实时推荐系统的实现架构与评价指标。

# 1实施推荐系统介绍与实例

一般来说实施推荐系统的关键点有两点，采集数据和推送结果。

采集数据其实不难，现在的实时数据采集系统不少，像最常用的flume。通过flume你可以将前端生成的日志做清洗转换然后存储下来。存储的介质有很多种，像hdfs、hbase等等，如果你想做实时推荐，那么就可以存在Kafka中。Kafka可以被理解为一个中间件，它从flume拿到数据，然后其他程序可以再拿去消费这些数据。Kafka就是最适合这种实时流计算场景的。

关于如何把计算结果推送出去，其实一般的机制都是被动触发式的。当接收到一个推荐的request时候，我们根据这个request信息去计算推荐列表，或者直接去redis里面取得已经计算好的推荐列表，返回这个结果就好了。只是这些行为都包裹在了一次网页请求中罢了。当然还有一种触发方式是主动推送，这种方式其实在调用数据方面和被动触发没什么区别，只不过请求方略有差别而已。

目前，在算法上主要通过引入聚类技术和改进实时协同过滤算法提高推荐速度；在架构上，目前实时推荐主要有基于Spark、Kiji框架和Storm的流式计算3种方法。

## 1.1算法

### 1.1.1聚类技术

在算法上，一般采用EM（Expectation-Maximization）、K-means、吉布斯（Gibbs Sampling）、模糊聚类等聚类技术提高推荐速度。因为使用聚类技术可以大大缩小用户或项目的最近邻居搜索范围，从而提高推荐的实时性，如下表所示。



### 1.1.2实时协同过滤算法

除此之外，实时协同过滤算法本身一直是人们研究的热点，早在2003年，Edward F. Harrington就第一次提出了基于感知器的实时协同过滤算法，但是这种方法需要所有用户的偏好，实用性较差；2010年，杨强等提出了实时进化的协同过滤算法，给予新得分更高的权重来增量更新User和Item的相似度；2011年，UC Berkeley的Jacob Abernethy等人提出了OCF-SGD算法，我们知道传统的矩阵分解把用户评分矩阵R分解成多个矩阵，比如R≈P\*Q，该方法提出当新来一个User到Item的得分，把更新R矩阵的问题转换成更新P和Q矩阵，从而达到实时协同过滤；近几年的RecSys大会上，实时协同过滤也是讨论的热点，OCF-SGD算法每次只考虑一个用户，忽略了用户之间的关系，Jialei Wang等人提出了基于多任务学习的实时协同过滤算法，把每一个用户当做一个任务，定义一个表示各个任务间相似性和交互程度的矩阵A，当新来一个User到Item的得分，通过矩阵A来更新其他用户的得分。

## 1.2架构

### 1.2.1基于Spark的方式

在架构上，第一种是使用Spark把模型计算放在内存中，加快模型计算速度，Hadoop中作业的中间输出结果是放到硬盘的HDFS中，而Spark是直接保存在内存中，因此Spark能更好地适用于数据挖掘与机器学习等需要迭代的模型计算，如下表所示。



### 1.2.2基于Kiji框架的方式

第二种是使用Kiji，它是一个用来构建大数据应用和实时推荐系统的开源框架，本质上是对HBase上层的一个封装，用Avro来承载对象化的数据，使得用户能更容易地用HBase管理结构化的数据，使得用户姓名、地址等基础信息和点击、购买等动态信息都能存储到一行，在传统数据库中，往往需要建立多张表，在计算的时候要关联多张表，影响实时性。Kiji与HBase的映射关系如下表所示。



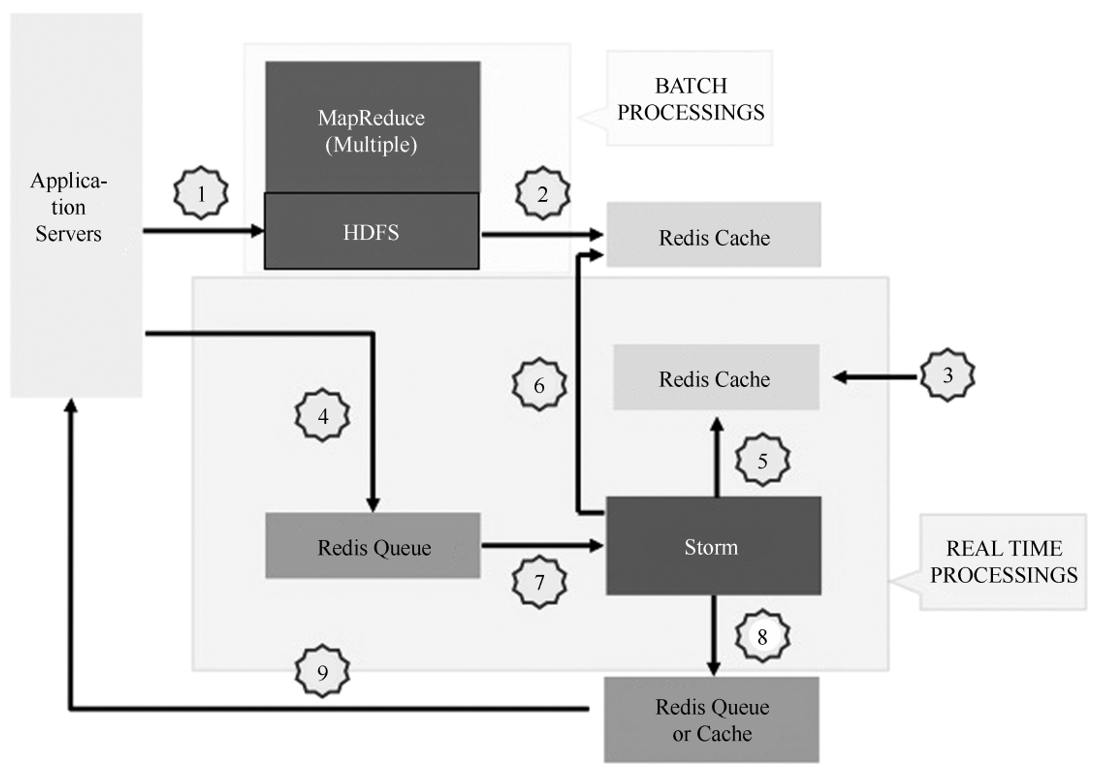
Kiji提供了一个KijiScoring模块，它可以定义数据的过期策略，如综合产品点击次数和上次的点击时间，设置数据的过期策略把数据刷新到KijiScoring服务器中，并且根据自己定义的规则，决定是否需要重新计算得分。如用户有上千万浏览记录，一次的行为不会影响多少总体得分，不需要重新计算，但如果用户仅有几次浏览记录，一次的行为，可能就要重新训练模型。Kiji也提供了一个Kiji模型库，使得改进的模型部署到生产环境时不用停掉应用程序，让开发者可以轻松更新其底层的模型。

### 1.2.3基于Storm的方式

最后一种是基于 Storm 的实时推荐系统。在实时推荐上，算法本身不能设计的太复杂，并且很多网站的数据库是TB、PB级别，实时读写大表比较耗时。可以把算法分成离线部分和实时部分，利用Hadoop离线任务尽量把查询数据库比较多的、可以预先计算的模型先训练好，或者把计算的中间数据先计算好，比如，线性分类器的参数、聚类算法的群集位置或者协同过滤中条目的相似性矩阵，然后把少量更新的计算留给Storm实时计算，一般是具体的评分阶段。

#### 1.2.3.1一个例子

如下图所示，我们可以搭建这样一个实施推荐系统：用HBase或HDFS存储历史的浏览、购买行为信息，用Hadoop基于User CF的协同过滤，先把用户的相似度离线生成好，用户到商品的矩阵往往比较大，运算比较耗时，把耗时的运行先离线计算好，实时调用离线的结果进行轻量级的计算有助于提高产品的实时性。



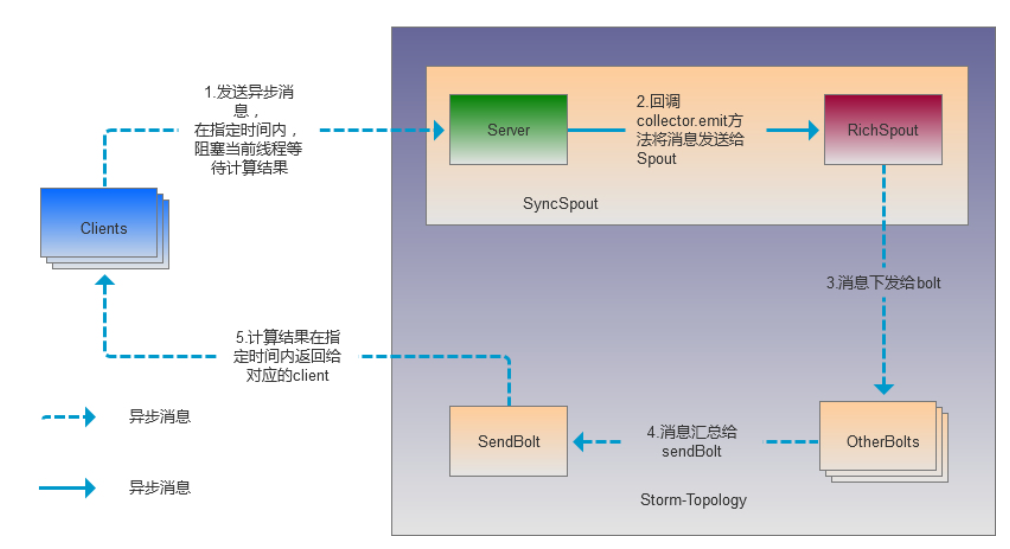
首先程序获取用户和产品的历史偏好，得到用户到产品的偏好矩阵，利用Jaccard相似系数（Jaccard coefficient）、向量空间余弦相似度（Cosine similarity）、皮尔逊相关系数（Pearson correlation coefficient）等相似度计算方法，得到相邻的用户（User CF）或相似商品（Item CF）。在User CF中，基于用户历史偏好的相似度得到邻居用户，将邻居用户偏好的产品推荐给该用户；在Item CF中，基于用户对物品的偏好向量得到相似产品，然后把这款产品推荐给喜欢相似产品的其他用户。

然后通过Kafka或者Redis队列，保存前端的最新浏览等事件流，在Storm的Topology中实时读取里面的信息，同时获取缓存中用户topN个邻居用户，把邻居用户喜欢的商品存到缓存中，前端从缓存中取出商品，根据一定的策略，组装成推荐商品列表。

当然除了相似性矩阵，其他模型大体实现也相似，比如实际的全品类电商中不同的品类和栏位，往往要求不同的推荐算法，如母婴产品，如果结合商品之间的序列模式和母婴年龄段的序列模式，效果会比较好，可以把模型通过Hadoop预先生成好，然后通过Storm实时计算来预测用户会买哪些产品。

#### 1.2.3.2基于Storm的syncspout

SyncSpout是上海华瑞银行（SHRB）大数据团队开发的，用来构造可交互的、同步的Storm拓扑的组件。我们在做实时推荐系统中，希望将Storm的并发性和分布式计算能力应用到“请求-响应”范式中， 比如客户的某次购买行为能够以消息的形式发送到storm拓扑中，storm在指定时间返回推荐结果，也就是说storm需要具有可交互性。基于这样的背景，大数据团队开发了SyncSpout组件， 该组件可以接收客户端异步的消息，经过Storm拓扑异步计算，在指定时间内返回给客户端。其架构图如下所示。



主要包括三个关键组件：

1）SyncSpout：继承storm的IRichSpout，用于接收客户端调用消息并将消息emit出去的Spout

2）SendBolt：拓扑中发送计算结果的bolt，该bolt将计算结果返回给客户端

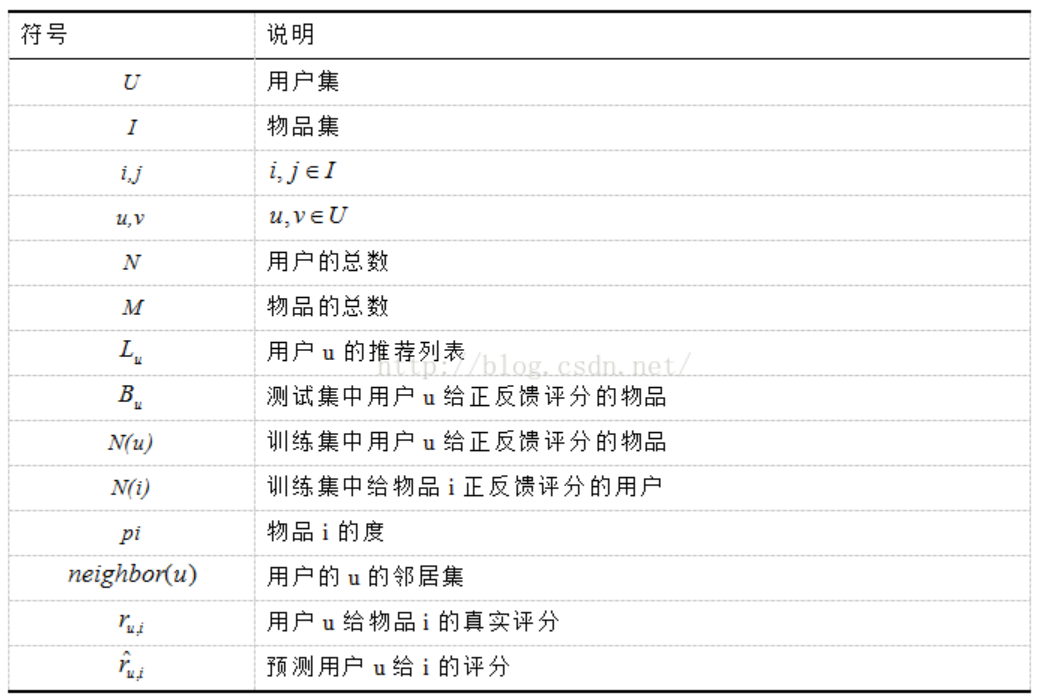
3）SyncSpoutClient：用于向SyncSpout发送同步消息，并在指定时间内获取结果

syncspout使普通的storm应用可交互，storm应用重启后，客户端可自动重连。且对storm应用几乎没有侵入，对业务没有侵入。storm集群返回的计算结果能够准确的返回给指定客户端的某次调用，客户端可发送任意类型的消息给storm应用；storm应用可返回任意类型的消息给客户端，客户端可在指定时间内同步获取storm应用返回的计算结果。除此之外，syncspout支持高并发，在单机环境下1000并发量基本在100毫秒内返回。

# 2评价指标

对推荐系统的研究一个重要的环节是如何评价一个推荐算法的好坏。评价方法分为离线评估，用户调查，在线评估。由于用户调查和在线评估代价要求高，目前大多数的研究采用的是离线测试。本文的研究工作采用了几种离线评估指标，将其归纳为准确性指标和非准确性指标。下面分别给予简单介绍。

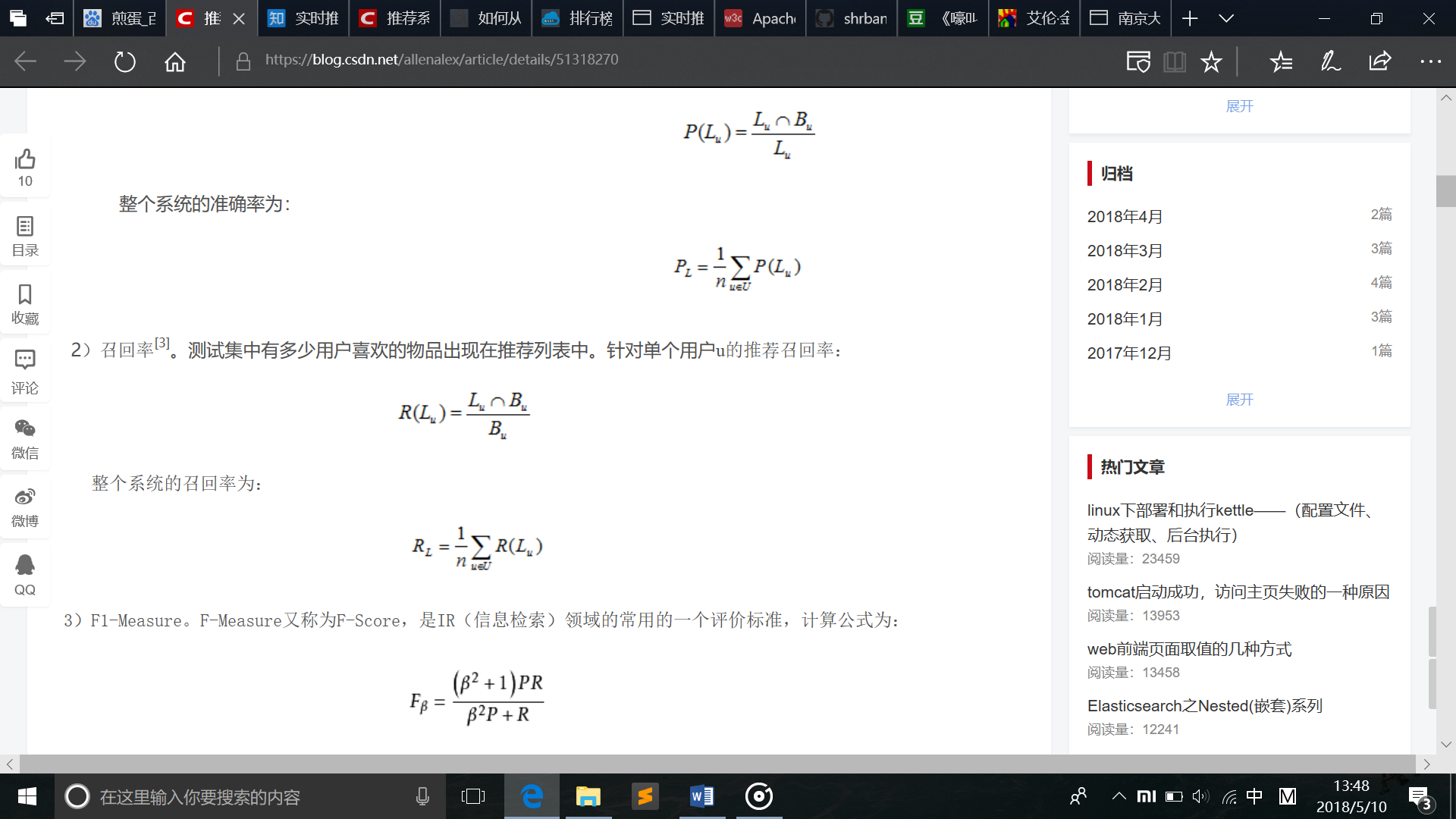
为了方便描述评价指标，下表对后面会用到的符号进行简单说明。



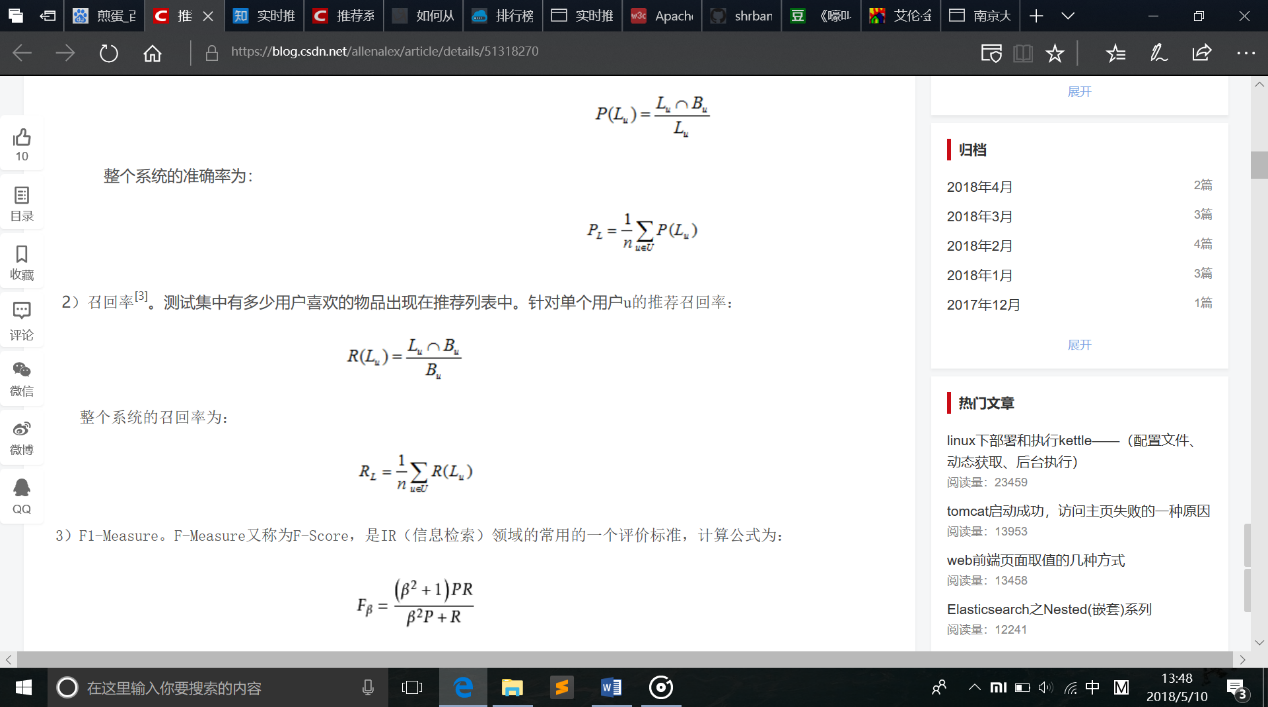
## 2.1准确性指标

准确性指标是推荐系统中最重要的指标。最常使用的准确性指标包括准确率和召回率。

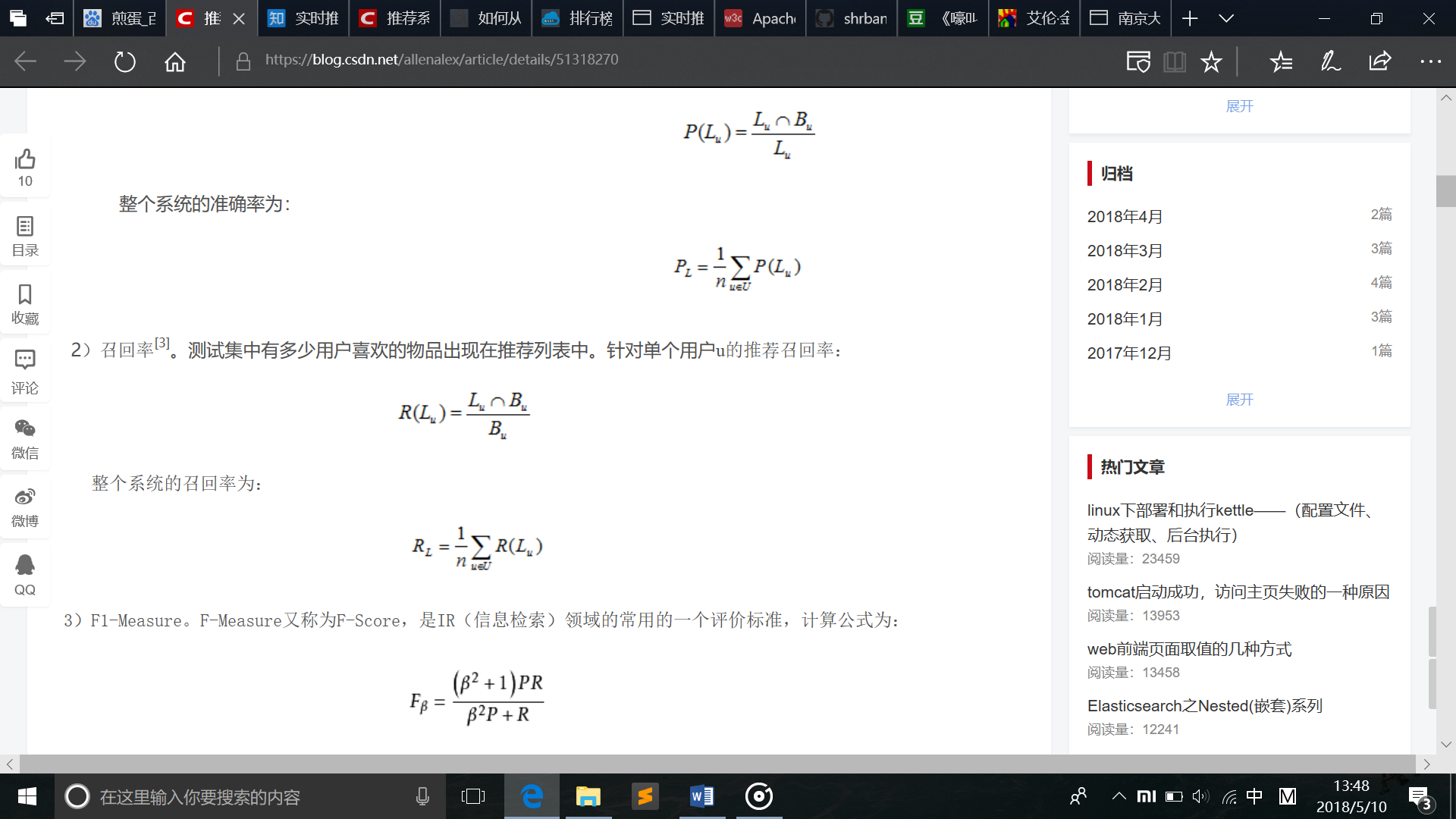
1. 准确率。推荐列表中用户喜欢的物品所占的比例。针对单个用户u的推荐准确率：



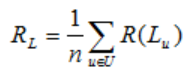
整个系统的准确率为：



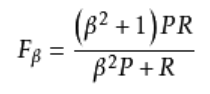
2）召回率。测试集中有多少用户喜欢的物品出现在推荐列表中。针对单个用户u的推荐召回率：



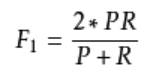
整个系统的召回率为：



3）F1-Measure。F-Measure又称为F-Score，是IR（信息检索）领域的常用的一个评价标准，计算公式为：

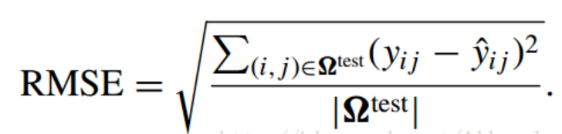


其中β是参数，P是精确率(Precision)，R是召回率(Recall)。当参数β=1时，就是最常见的F1-Measure了：



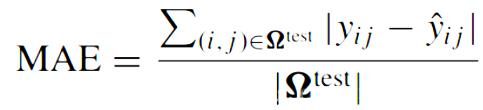
以上的准确性指标是只从整个推荐列表的推荐用户是否喜欢这种二值情况考虑的。除此之外，针对评分（比如0-5分制）的评估主要还有：

1）RMSE （均方根误差）；

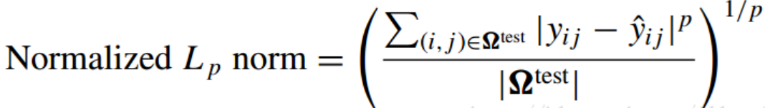


其中Ωtest为测试集；yij为用户i对物品j的实际评分，ij为预测评分。

2）MAE （平分绝对误差）：



3）Normalized Lp norm:



显然，RMSE,MAE是 Normalized Lp norm的特例。

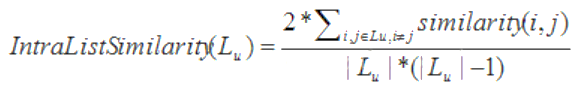
准确性指标局限性也有它的局限性，比如不适合评估排序性能，而且离线的准确率的提高并不意味着实际线上系统效果的提高。

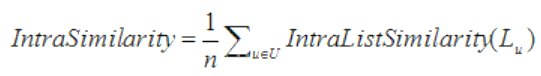
## 2.2 非准确性指标

推荐系统中，除了推荐准确性外，还有其他一些重要的指标。包括推荐的多样性、新颖性、惊喜度和覆盖率等等。本文将这些指标统称为非准确性指标。其中非准确性指标又可以分为用户级非准确性指标和系统级非准确性指标。

### 2.2.1用户级非准确性指标

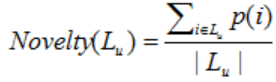
1）个体多样性。用户的推荐列表列内的所有物品的平均相似度:



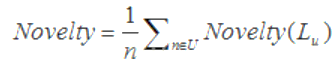


IntraSimilarity值越大，说明用户的推荐列表内的物品之间总体平均相似度越高，也就是系统整体的个体多样性越低。

1. 新颖性。评估新颖性最简单的方法是计算推荐列表中物品的平均流行度：

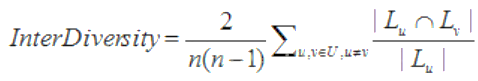


整个系统的新颖性为：



### 2.2.2系统级的非准确性指标

1）整体多样性。采用推荐列表间的相似度，也就是用户的推荐列表间的重叠度来定义整体多样性。



2）覆盖率。覆盖率测量的是推荐系统推荐给所有用户的物品数占总物品数的比例。

