摘要：随着互联网的发展和普及，网络平台和论坛成为了股票投资者交流与讨论的重要平台，然后一些网络的评论信息对股票市场的影响日益显著。中国的股票市场处于刚刚发展的阶段，理性投资者较少，投资的决策很容易收到网络上的消息以及传闻的影响。在网络发展的现在，用户的评论数据量也越来越大，传统数据处理技术遇到了瓶颈，大数据相关技术为数据处理这方面提供了良好的支持。本文针对股票网站系统中，由于互联网“信息过载”造成的难以精确定位用户兴趣并提供产品推荐的问题，利用大数据平台，通过深入挖掘股票社中丰富的用户评论信息，开发产品特征提取算法，建立用户兴趣偏好模型，结合用户评论打分来改善传统协同过滤推荐的推荐准确性；通过一系列的处理，准确的挖掘出有用的信息，并向用户进行推荐。

关键字：

绪论

1.背景和意义

2.现状分析

3.目的和主要内容

4.中国股市简介

5.影响股市因素

5.1宏观因素

5.2微观因素

5.3市场因素

1.大数据相关技术概述

1.1 hdfs文件系统

HDFS（Hadoop Distributed File System）是一个分布式文件系统，提供高吞吐量和一定高度的容错性的应用程序数据访问，适用于大数据集上的应用，对外部客户端而言，HDFS 类似于一个传统的文件系统。可以创建、删除、移动或重命名文件，等等。但是 HDFS 的架构是基于一组特定的节点构建的，这是由它自身的特点决定的。这些节点包括 NameNode，它在 HDFS 内部管理元数据；DataNode是HDFS里的存储块，所有数据存放在其中。由于仅存在一个 NameNode，因此很容易造成单点故障，但是可以搭建一个高可靠的HDFS集群，拥有两个NameNode，一个NameNode为Active状态，一个NameNode为StandBy状态。

存储在HDFS中的文件被分成块，然后将这些块复制到多个计算机中（DataNode）。这与传统的 RAID 架构大不相同。块的大小默认为128M，低版本为64M，当然块的大小和复制的块数量在创建文件时由客户机决定。

1.1.1 概述

1.1.2

1.1.3 特性

1.HDFS中的文件在物理上是分块存储（block），块的大小可以通过配置参数( dfs.blocksize)来规定，默认大小在hadoop2.x版本中是128M，老版本中是64M

2.HDFS文件系统会给客户端提供一个统一的抽象目录树，客户端通过路径来访问文件，形如：hdfs://namenode:port/dir-a/dir-b/dir-c/file.data

3.目录结构及文件分块信息(元数据)的管理由namenode节点承担——namenode是HDFS集群主节点，负责维护整个hdfs文件系统的目录树，以及每一个路径（文件）所对应的block块信息（block的id，及所在的datanode服务器）

4.文件的各个block的存储管理由datanode节点承担--- datanode是HDFS集群从节点，每一个block都可以在多个datanode上存储多个副本（副本数量也可以通过参数设置dfs.replication）

5.HDFS是设计成适应一次写入，多次读出的场景，且不支持文件的修改

1.2 zookeeper协调服务

1.2.1 概述

1.2.2 选举机制

1.2.3 运行机制

1.3 Kafka消息系统

1.3.1 Kafka基本介绍

Kafka最初是由Linkedin公司开发，是一个分布式，可分区，多副本，多消费者，基于Zookeeper管理的分布式消息系统，常作为流处理的高级数据来源。Kafka于2010年捐献给Apache基金会，并成为其顶级开源项目。

Kafka主要的设计目标：

1.以时间复杂度为O1的方式给消息的持久化提供强大的能力，就算是TB级别的数据也能保证正常的访问性能。

2.高吞吐量，即使在配置底下的商用机器上也能做到每秒100KB的消息传输能力

3.支持消息分区，消息多副本，以提供高可靠的消息传输。以及分布式消费数据，一个partition内的消息能保证顺序传输。

4.支持离线和实时两种数据处理。

1.3.2 Kafka的设计原理

一个原始Kafka集群应该包括多个Producer（生产者），多个Broker,多个Consumer（消费者）和一个Zookeeper集群。Zookeeper为Kafka管理集群配置。包括Kafka的Leader选举，在（Consumer Group）消费组发生变化时进行Rebalance。Produce（生产者）使用推的方式向Kafka的Broker发送消息，消费者（Consumer）使用拉的方式从Kafka的Broker消费消息。

Kafka术语：

* Broker：消息处理中间结点，一个Kafka节点就是一个Broker，多个Broker构成一个Kafka集群，每一个Broker的Broker.id都不相同。
* Topic：主题，存放着相同类型的消息，Kafka集群能够同时负责多个Topic的分发。
* Partition：Topic物理上的分组，一个Topic可以由多个Partition组成，每个Partition是一个有序的消息队列。
* Replication：副本数，每个Partition的备份数量。避免数据丢失。
* Segment：Partition在磁盘中存放是由多个Segment组成。
* offset：每个Partition都由一系列有序的、不可变的消息组成，这些消息被连续的追加到Partition中。Partition中的每个消息都有一个自己的序列号叫做offset，用于Partition唯一标识一条消息，也用于记录消费者消费的位置。
* Producer：负责生产发布消息到Kafka Broker。
* Consumer：消息消费者，向Kafka Broker消费消息的客户端。
* Consumer Group：每个Consumer属于一个特定的Consumer Group，不同消费组的消费者可以消费相同的消息。

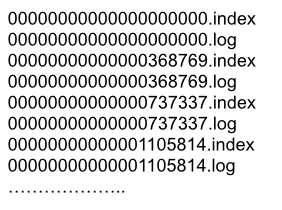
1.3.3 Kafka消息存储

在Kafka中同一个Topic通常存储的时同一类的消息，每个Topic由多个partition组成，每个partition在磁盘中时append log文件。

每个Partition（目录）文件被切分为多个大小相等segment数据文件。但每个segment文件中的消息数量不一定相等，这种特性方便老的segment文件能快速被删除。每个Partition只需要支持顺序读写就行了，segment文件生命周期由服务端配置参数决定。

segment文件由两部分组成，分别为index file（索引文件）和data file（数据分见），这两个文件一一对应，有一个data文件，就一定有一个index文件。后缀为“.index”和“.log”分别表示为segment索引文件、数据文件。

segment文件命名规则：Partition全局的第一个segment从0开始，后续每个segment文件名为上一个segment文件中最后一条消息的offset值。数值最大为64位long大小，19位数字字符长度，没有数字用0填充。



1.3.4 Kafka副本策略

数据同步

Kafka在0.8版本才开始提供Partition的Replication机制，之前的机制中如果Broker宕机，宕机的Broker中的所有Partition就都无法提供服务，而Partition中没有备份数据，数据可靠性就很低。所以0.8后提供了Replication机制来保证Broker的故障转移。

引入Replication之后，一个Partition会有多个Replication，而这时需要在这些Replication之间选举出一个Leader，Producer（生产者）和Consumer（消费者）只与这个Leader交互，其它Replica作为Follower从Leader中拉取数据。

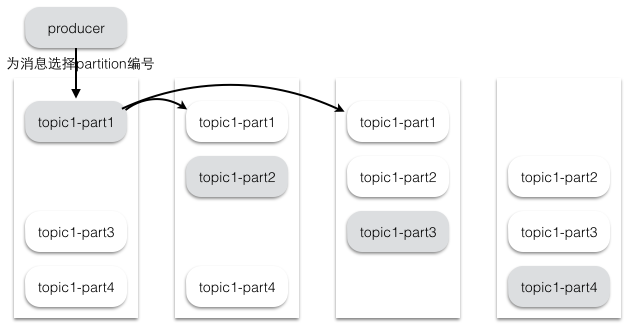


副本放置策略

为了更好的做负载均衡，Kafka尽可能的将Partition的Replication均匀分配到集群上。Kafka的Replication分配算法如下：

* 将所有存活的Brokers和Partitions排序。
* 将第i个Partition分配到第（i mod n）个Broker上面，这个Partition的第一个Replication存在存放Partition的Broker上面。并会作为这个Partition的优先副本（Leader）。
* 将第i个Partition的第j个副本分配到第（（i+j）mod n）个Broker上。

假设集群一共有4个Broker,一个Topic拥有4个Partition,每个Partition拥有3个副本，那么分配情况如下图：



同步策略

Producer在发布消息到某个Partition时，先通过ZooKeeper找到该Partition的Leader，Producer只会将该消息发送到该Partition的Leader。Leader会更新本地的data文件和index文件。每个Follower都从Leader拉取数据。这种方式上，Follower中消息的存储的数据顺序与Leader保持一致。Follower在收到该消息并写入文件后，向Leader发送ACK。一旦Leader收到了ISR中的所有Replica的ACK后，就说明该消息就被认为已经commit了，Leader将增加HW并且向Producer发送ACK，消费者消费数据只能消费到HW。

为了提高性能，每个Follower在接收到数据后就向Leader发送ACK，而非等到数据写入文件中。因此，对于已经commit的消息，Kafka只能保证它被存于多个Replica的内存当中，而不能保证它们被持久化写入到磁盘中，所以不能完全保证异常发生后该条消息一定能被Consumer（消费者）消费。

1.3.5 Kafka消息消费原理

一个Topic的一条消息只能被同一个Consumer Group(消费组)的一个Consumer（消费者）消费，不同Consumer Group（消费组）的不同Consumer（消费者）可以消费同一条消息。



作为一个消息系统，Kafka遵循了传统发送消息和消费的方式，选择由Producer向broker push(推送)消息并由Consumer从broker pull(拉取)消息。

push(推送)模式很难适应消费速率不同的Consumer，因为消息发送速率控制权在broker。push模式的目标是尽可能以最快速度传递消息，但是这样很容易造成Consumer来不及处理消息，然后造成数据堵塞和网络拥塞。而pull（拉取）模式的消费速率可以由Consumer自身的消费能力来自主控制。

对于Kafka而言，pull模式更合适。pull模式可降低Broker的工作量，消费消息的速率由Consumer自己来决定，同时Consumer可以自己控制消费方式，可批量消费也可逐条消费。

1.4 SparkStreaming流处理

1.4.1 概述

1.4.2 特性

1.4.3 执行流程

2.股评文本数据采集概述

2.1 数据爬虫概述

2.2 爬虫技术概述

2.3 Java爬虫Jsoup技术

2.3.1

2.3.2

3.文本数据分析相关技术与算法概述

3.1 文本数据分词

3.2 词汇数据清洗

4.系统设计概述

4.1 系统模块概述

4.2 数据库设计概述

5.基于大数据平台的股评信息文本挖掘实现

5.1 股评数据爬取

2.2 消息系统中间键的搭建

5.2 大数据平台实时分词

5.3 词汇筛选

5.4 股票打分

5.5 结果处理以及股票推荐

结 论

附 录

参考文献

致 谢