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

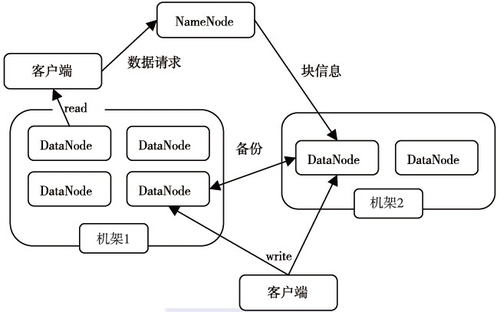
5.2微观因素

5.3市场因素

1.大数据相关技术概述

1.1 HDFS文件系统

1.1.1 HDFS概述



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。

1.1.2 HDFS角色简介

NameNode:

Namenode起一个管理元数据的作用，用户通过namenode来请求对其他数据的访问和操作。Namenode包含：目录与数据块之间的关系（靠fsimage和edits来实现），数据块和节点之间的关系。fsimage文件与edits文件是Namenode结点上的核心文件。

Namenode中仅仅存储数据位置信息，而关于BLOCK的位置信息则是从各个Datanode上传到Namenode上的。Namenode的数据位置信息就是物理的存储在fsimage这个文件中的，当Namenode启动的时候会首先读取fsimage这个文件，将位置信息装载到内存中。而edits存储的是日志信息，在Namenode启动后所有对数据的的增加，删除，修改等操作都会记录到edits文件中，并不会同步的记录在fsimage中。而当Namenode结点关闭的时候，也不会将fsimage与edits文件进行合并，合并的过程发生在Namenode启动的过程中。如果Namenode在启动后发生的改变过多，会导致edits文件变得非常大，大得程度与Namenode的更新频率有关系。那么在下一次Namenode启动的过程中，读取了fsimage文件后，会应用这个无比大的edits文件，导致启动时间变长，并且不可控。

DataNode:

提供真实文件数据的存储服务。

文件块（block）：HDFS中最基本的存储单位。对于文件内容而言，一个文件的长度大小是size，那么从文件的０偏移开始，按照固定的大小，顺序对文件进行划分并编号，划分好的每一个块称一个Block。

HDFS默认Block大小是128MB，以一个256MB文件，共有256/128=2个Block. 不同于普通文件系统的是，HDFS中，如果一个文件小于一个数据块的大小，并不占用整个数据块存储空间。(这样设置可以减轻namenode压力，因为namonode维护者文件的大小与数据块数量有关)

Replication。多复本。默认是三个。（hdfs-site.xml的dfs.replication属性）

1.1.3 HDFS特性

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。

1.1.4 HDFS读写流程

写流程：



1、根namenode通信请求上传文件，namenode检查目标文件是否已存在，父目录是否存在

2、namenode返回是否可以上传

3、client会先对文件进行切分，比如一个blok块128m，文件有300m就会被切分成3个块，一个128M、一个128M、一个44M请求第一个 block该传输到哪些datanode服务器上

4、namenode返回datanode的服务器

5、client请求一台datanode上传数据（本质上是一个RPC调用，建立pipeline），第一个datanode收到请求会继续调用第二个datanode，然后第二个调用第三个datanode，将整个pipeline建立完成，逐级返回客户端

6、client开始往A上传第一个block（先从磁盘读取数据放到一个本地内存缓存），以packet为单位（一个packet为64kb），当然在写入的时候datanode会进行数据校验，它并不是通过一个packet进行一次校验而是以chunk为单位进行校验（512byte），第一台datanode收到一个packet就会传给第二台，第二台传给第三台；第一台每传一个packet会放入一个应答队列等待应答

7、当一个block传输完成之后，client再次请求namenode上传第二个block的服务器。

读流程：



1、跟namenode通信查询元数据（block所在的datanode节点），找到文件块所在的datanode服务器

2、挑选一台datanode（就近原则，然后随机）服务器，请求建立socket流

3、datanode开始发送数据（从磁盘里面读取数据放入流，以packet为单位来做校验）

4、客户端以packet为单位接收，先在本地缓存，然后写入目标文件，后面的block块就相当于是append到前面的block块最后合成最终需要的文件。

1.2 Zookeeper协调服务

1.2.1 Zookeeper概述

Zookeeper是一种为分布式应用HBase，Kafka等所设计的高可用、高性能且一致的开源协调服务。它提供一项基本服务：分布式锁服务。Zookeeper现在版本也提供集群的配置维护、分布式消息队列、分布式通知/协调、负载均衡，命名服务，分布式锁等功能。Zookeeper性能上的特点决定了它能够用在大型的、分布式的系统当中。从集群可靠性方面来说，它并没有单点故障。从分布式协调服务来说，它能够保证分布式数据的一致性。所谓分布式数据一致性，就是保证集群中数据传递的一致性。

1.2.2 Zookeeper的特点

顺序一致性：从同一个客户端发起的事务请求，会验证按照其发起的顺序应用到Zookeeper中

原子性：所有事物请求在整个集群中的所有机器上的应用情况是一致的，即，整个集群中所有机器要么都成功应用了某一事务，要么都没有应用某一事物，不会出现集群中部分机器应用了该事务，另外一部分没有应用的情况。

单一视图：无论客户端连接的是哪台Zookeeper服务器，其看到的服务端数据模型都是一致的。

可靠性：如果服务端成功的执行了一个事务，并完成对客户端的响应，那么该事务所引起的服务端的状态变更将会一直保留下来，直到下一个事务又对其进行了更改。

1.2.3 Leader选举机制

(1) 每个Server发出一个投票。由于是初始情况，Server1和Server2都会将自己作为Leader服务器来进行投票，每次投票会包含所推举的服务器的myid和ZXID，使用(myid, ZXID)来表示，此时Server1的投票为(1, 0)，Server2的投票为(2, 0)，然后各自将这个投票发给集群中其他机器。

(2) 接受来自各个服务器的投票。集群的每个服务器收到投票后，首先判断该投票的有效性，如检查是否是本轮投票、是否来自LOOKING状态的服务器。

(3) 处理投票。针对每一个投票，服务器都需要将别人的投票和自己的投票进行PK，PK规则如下:

　　　· 优先检查ZXID。ZXID比较大的服务器优先作为Leader。

　　　· 如果ZXID相同，那么就比较myid。myid较大的服务器作为Leader服务器。

　　对于Server1而言，它的投票是(1, 0)，接收Server2的投票为(2, 0)，首先会比较两者的ZXID，均为0，再比较myid，此时Server2的myid最大，于是更新自己的投票为(2, 0)，然后重新投票，对于Server2而言，其无须更新自己的投票，只是再次向集群中所有机器发出上一次投票信息即可。

(4) 统计投票。每次投票后，服务器都会统计投票信息，判断是否已经有过半机器接受到相同的投票信息，对于Server1、Server2而言，都统计出集群中已经有两台机器接受了(2, 0)的投票信息，此时便认为已经选出了Leader。

(5) 改变服务器状态。一旦确定了Leader，每个服务器就会更新自己的状态，如果是Follower，那么就变更为FOLLOWING，如果是Leader，就变更为LEADING。

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。

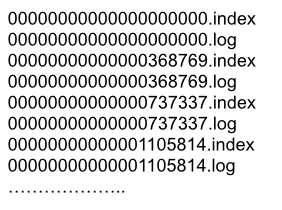
1.3.3 Kafka消息存储

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。

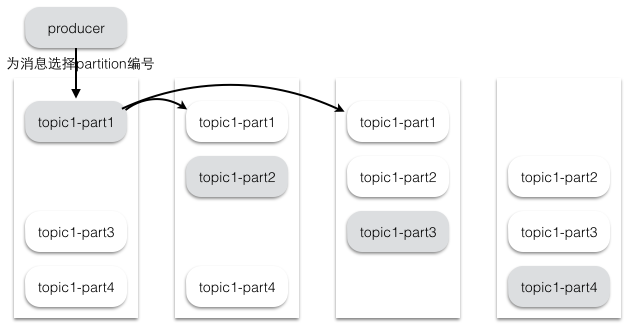


副本放置策略

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

Spark Streaming 是Spark核心API的一个扩展，可以实现高吞吐量的、具备容错机制的实时流数据的处理。支持从多种数据源获取数据，包括Kafk、Flume、Twitter、ZeroMQ、Kinesis 以及TCP sockets，从数据源获取数据之后，可以使用诸如map、reduce、join和window等高级函数进行复杂算法的处理。最后还可以将处理结果存储到文件系统，数据库和现场仪表盘。在“One Stack rule them all”的基础上，还可以使用Spark的其他子框架，如集群学习、图计算等，对流数据进行处理。

1.4.2 现状

2015年是流式处理的一年。大家考虑用Spark，主要也是因为Spark Streaming。这是一个流处理的时代，一切数据如果与流式处理不相关的话，都是无效的数据。Spark之所以强悍的一个重要原因在于，它的流式处理可以在线使用图计算、机器学习或者SparkR的成果，这得益于Spark一体化、多元化的基础架构设计。也就是在Spark Streaming中可以调用其它子框架，无需任何设置。这是Spark的无可匹敌之处，也是Spark Streaming必将一统天下的根源。但Spark的应用中，Spark Streaming也是最容易出问题的。

Spark Streaming与其它子框架不同之处在于，它更像是Spark Core之上的一个应用程序。所以如果要做Spark的定制开发，Spark Streaming则提供了最好的参考。你想掌握Spark Streaming，但你不去精通Spark Core的话，那是不可能的。所以我们选择Spark Streaming来提升自己，是找到了关键点。

1.4.3 特性

SparkStreaming是一个对实时数据流进行高通量、容错处理的流式处理系统，可以对多种数据源（如Kdfka、Flume、Twitter、Zero和TCP 套接字）进行类似Map、Reduce和Join等复杂操作，并将结果保存到外部文件系统、数据库或应用到实时仪表盘。

计算流程：

Spark Streaming是将流式计算分解成一系列短小的批处理作业。这里的批处理引擎是Spark Core，也就是把Spark Streaming的输入数据按照batch size（如1秒）分成一段一段的数据（Discretized Stream），每一段数据都转换成Spark中的RDD（Resilient Distributed Dataset），然后将Spark Streaming中对DStream的Transformation操作变为针对Spark中对RDD的Transformation操作，将RDD经过操作变成中间结果保存在内存中。整个流式计算根据业务的需求可以对中间的结果进行叠加或者存储到外部设备。下图显示了Spark Streaming的整个流程。

容错性：

对于流式计算来说，容错性至关重要。首先我们要明确一下Spark中RDD的容错机制。每一个RDD都是一个不可变的分布式可重算的数据集，其记录着确定性的操作继承关系（lineage），所以只要输入数据是可容错的，那么任意一个RDD的分区（Partition）出错或不可用，都是可以利用原始输入数据通过转换操作而重新算出的。对于Spark Streaming来说，其RDD的传承关系如下图所示，图中的每一个椭圆形表示一个RDD，椭圆形中的每个圆形代表一个RDD中的一个Partition，图中的每一列的多个RDD表示一个DStream（图中有三个DStream），而每一行最后一个RDD则表示每一个Batch Size所产生的中间结果RDD。我们可以看到图中的每一个RDD都是通过lineage相连接的，由于Spark Streaming输入数据可以来自于磁盘，例如HDFS（多份拷贝）或是来自于网络的数据流（Spark Streaming会将网络输入数据的每一个数据流拷贝两份到其他的机器）都能保证容错性，所以RDD中任意的Partition出错，都可以并行地在其他机器上将缺失的Partition计算出来。这个容错恢复方式比连续计算模型（如Storm）的效率更高。

实时性：

对于实时性的讨论，会牵涉到流式处理框架的应用场景。Spark Streaming将流式计算分解成多个Spark Job，对于每一段数据的处理都会经过Spark DAG图分解以及Spark的任务集的调度过程。对于目前版本的Spark Streaming而言，其最小的Batch Size的选取在0.5~2秒钟之间（Storm目前最小的延迟是100ms左右），所以Spark Streaming能够满足除对实时性要求非常高（如高频实时交易）之外的所有流式准实时计算场景。

扩展性与吞吐量：

Spark目前在EC2上已能够线性扩展到100个节点（每个节点4Core），可以以数秒的延迟处理6GB/s的数据量（60M records/s），其吞吐量也比流行的Storm高2～5倍，图4是Berkeley利用WordCount和Grep两个用例所做的测试，在Grep这个测试中，Spark Streaming中的每个节点的吞吐量是670k records/s，而Storm是115k records/s。

1.4.3 执行流程

Application运行于StreamigContext和SparkContext之上。

SparkStreaming为输入源启动接收器Receiver，Receiver以任务Task的形式运行在应用的执行器Executor中。

Receiver接受输入源并拆分为块（离散化），这些块其实就是RDD。

Receiver收集到输入数据后会把数据复制到另一个Executor中来保障容错性。

SparkContext会周期性的（由开发者设置的时间间隔决定）在内存中运行job来处理这些RDD数据。

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

5.2 分布式文件系统搭建

5.3 消息系统中间键的搭建

5.4 大数据平台实时分词

5.5 词汇筛选

5.6 股票打分

5.7 结果处理以及股票推荐

结 论

附 录

参考文献

致 谢