摘要：随着互联网的成长和普及，网络平台和论坛成为了股票投资者探讨与会商的主要平台，一些网络的评价信息对股票市场的影响日趋明显。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，利用大数据平台，通过深入挖掘股票社区中丰富的用户评论信息，通过分词、特征提取算法，筛选出相关的词汇，结合用户评论对股票进行分层打分；最后通过统计分析得出最好的股票或者行业，并对其做出推荐。

关键字：

绪论

1. 背景

随着数据库技术的发展以及人们获取数据的手段越来越多，我们拥有的数据也越来越多，根据美国GTE研究中心统计，全国范围内仅科研机构每天储存的新的数据就高达几TB。巨量的数据在给人们带来方便的同时也带来了很多的问题。比如过量的数据使得我们无法抓住其中的重点而难以消化和理解。传统的数据库也仅仅是对所获得的数据进行简单的进行存取等操作，然而人们对这些数据的理解也是非常有限的。在股票市场，这种情况也是非常严重的，每支股票所产生的数据也是繁多的，但股票市场的效益丰厚使得越来越多的投资者进入了股票市场。如果没有一种快速合理有效的处理方法对这些数据进行挖掘的话，投资者将会淹埋在巨大的数据中，无法对股市行情进行一个明确的判断和投资，给投资者带来巨大的损失。如果数据仅仅是表现为简单的储存而不经过处理那么这种数据我们可以认为这是毫无价值的，只有经过处理的数据才能体现出他的价值[1]。随着人们对有用数据的强烈需求数据挖掘技术应运而生，并迅速得到了发展。经过多年的研究和实践，数据挖掘技术吸收各种科学研究成果，形成了独具特色的研究方法，数据挖掘概念也逐步被大多数专家和学者所接受，研究的成果也得到大家的认可，同时也吸引了越来越的研究者，但目前的数据挖掘还是存在许多待研究和探索的问题需要专家和学者继续开发新的挖掘方法[2]。

1. 目的和意义
2. 国内外研究现状
3. 股票市场概述

1.中国股市简介

2.影响股市因素

2.1宏观因素

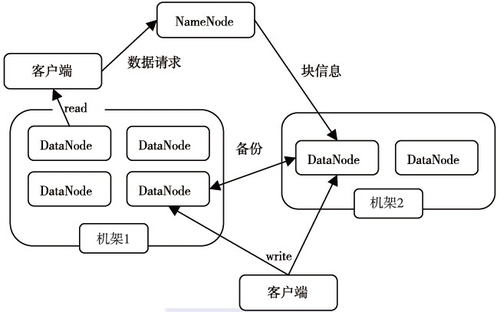
2.2微观因素

2.3市场因素

2.大数据技术概述

2.1 HDFS文件系统

2.1.1 HDFS概述



HDFS（Hadoop Distributed File System）是一个搭建在集群上的分布式文件系统，平常都用于大数据集上的程序，为程序提供一个高容错、高吞吐量和的数据访问平台，对客户端而言，HDFS就像传统的文件系统。可以使用新建、删除、移动或重命名等方式对文件进行修改。HDFS的架构是基于一组特定的节点构建的，这是由HDFS本身的特点掌控的。这些角色包括DataNode和NameNode，NamNode是HDFS里的存储单元，所有数据存放在其中；NameNode在 HDFS 内部管理元数据,记载着每个文件所处的位置信息。由于仅存在一个 NameNode，很容易造成单点故障，所以我们想出了另一种搭建集群的方式。搭建一个高度可靠的HDFS集群，具有两个NameNode，一个NameNode为Active（活跃）状态，一个NameNode为StandBy（待命）状态。

2.1.2 HDFS角色简介

NameNode:

Namenode起一个元数据管理的作用，记载了数据存储的节点信息和位置信息，用户向namenode发出访问指令来对数据的访问，namenode则返回给用户一个数据存储的DataNode的信息。Namenode中带有：文件名和数据块之间的联系（由edits和fsimage两个文件来实现），数据块和DataNode节点之间的联系。edits文件与fsimage文件是Namenode结点上的主要文件。

Namenode中仅存储数据存储的位置信息，而关于数据存储的的位置信息则是由Datanode提交到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存储的是用户修改文件的log信息，在Namenode启动后用户对数据的增添，删除，修改等访问都会被记录到edits文件中，并不会同步到在fsimage中。就算是在关停或者kill掉Namenode节点的时候，也不会将edits与fsimage文件进行整合，整合发生在Namenode节点启动中的时候。若是Namenode在启动后产生的改变非常多，会致使edits文件变得非常大，之所以这么大，是因为与Namenode的更新频率有联系。如果文件过大，那么在下次Namenode启动中，读取了fsimage文件后，会调用这个无比大的edits文件，致使启动等待时间变长，而且不能控制。

DataNode:

提供真实文件数据的存储服务。

数据块（block）：HDFS中真正储存数据的节点。对文件内容而言，一个文件从文件的０偏移开始，按照指定的大小，依次对文件进行切分并分别编号，切分好的每个文件称一个数据块(block)。

HDFS默认Block大小是128MB，老版本中默认一个块的大小为64M，以一个512MB文件来说，共有512除以128等于4，所以这个文件有个4个Block。不同于传统文件系统的是，在HDFS中，若是一个文件小于一个数据块（block）的大小，即小于128M，并不会占用所有数据块的存储空间。(该设置可以减缓namenode压力，这是由于namonode维护的文件大小与数据块数目有关。数据块越多，记载的位置信息越多，维护的文件就越大。)

2.1.3 HDFS特性

存储在HDFS中的文件被按指定大小切分成块，然后将这些块备份到多个计算机中（DataNode），默认是三份，一份是本机上，一份是同一机架的不同机器上，一份是不同机架的不同机器上，这样就既能保证数据的快速访问，又能保证数据的容灾。块的大小默认为128M，低版本为64M，当然块的大小和备份的块数量可以在配置中进行修改。

2.1.4 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，比如一个数据块128m，文件有300m就会被切分成3个块，一个128M、一个128M、一个44M，然后请求第一个数据块该传输到哪些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上传第一个数据块，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）。第一台每传一个packet会放入一个应答队列等待应答。

7、当一个数据块传输完成以后，client再次向NameNode发出请求来获得上传第二个数据块的服务器具体信息。

读流程：



1、跟namenode通信查询元数据（数据块所在的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、客户端按照packet为单位一个个接收，先通过本地进行缓存，在写入目标文件，之后的数据块就依次的追加到前面的数据块，最后合并生成最终需要的文件。

2.2 Zookeeper协调服务

2.2.1 Zookeeper概述

Zookeeper是一种服务于分布式程序HBase，Kafka等所设计的高新能、高可用且一致的协调应用。Zookeeper的优点有：服务器进程之间的彼此排斥和协作、简单的分布式协调、序列化（按照特定的规则对数据进行编码）、有序的消息、原子性、可靠性。Zookeeper特征使它能够在分布式的程序当中得到使用。从Zookeeper可靠性来讲，它是没有单点故障的，Zookeeper必须存在2n+1个节点。从分布式协调来讲，它能够保证分布式程序中数据的一致性。所谓分布式数据一致性，即让集群中传递的数据保持一致。

2.2.2 Leader选举机制

1、当master的zookeeper启动时，通过zoo.cfg中配置的其他zookeeper机器进行通讯，发现只有一台zookeeper启动，不能提供服务。

2、当slave1的zookeeper启动时，通过zoo,cfg中配置的zookeeper机器进行通讯。发现此时已经启动了两台zookeeper。超过zookeeper集群的半数。Zookeeper间进行选举。

选举投票：

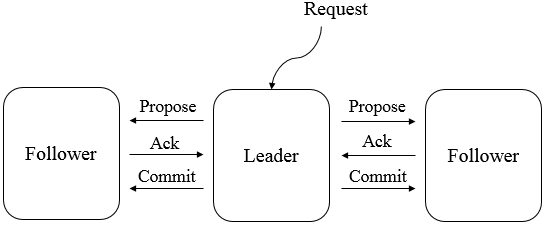
①每个zookeeper都投递出自己的myid。Master投出0，slave1投出1.

②投票结果不统一。进行第二轮投票

③Master发现自己的id小于slave1的。则投递出1，Slave1发现自己的id大于master，则继续投递1

④投票结果统一，则leader为slave1.master身份为follower.

3、当slave3启动时，发现已经由leader存在了。则以follower身份与leader进行通讯（同步数据）。下图为leader与follower进行通讯，保证事物一致性。



2.3 Kafka消息系统

2.3.1 Kafka基本介绍

Kafka最初是由Linkedin公司发布，是一个分布式的多生产者、多消费者、可分区、多副本、基于Zookeeper管理的可横向拓展的分布式消息系统，常常作为流式数据处理程序的高级数据来源。Kafka最终被捐献给Apache基金会，并成为Apache的TOP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，当数据量达到TB级别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topic的分区，分区消息的多副本，以达到高可靠，高性能的消息传输。分布式的生产数据、消费数据。每个partition内的消息始终是保持着顺序传输，这是因为同一个partition的消息是以顺序存储。

4.支持离线和实时两种数据处理。

2.3.2 Kafka的设计原理

一个原始Kafka集群应当包括多个消费组、多个消费者、多个生产者、多个Broker和一个Zookeeper集群。Zookeeper为Kafka管理消费者消费数据的偏移量和一些配置。包括Kafka的Leader选举等。生产者利用推送消息的方式向Kafka的Broker推送消息，消费者使用拉取消息的方式从Kafka的Broker获取消息。

Kafka术语：

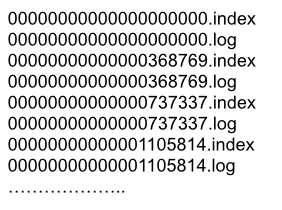
* Broker：消息处理中间节点，一个Kafka节点即一个Broker，多个Broker组成一个Kafka集群，每个Broker的Broker.id都不相同。
* Topic：主题，存储着相同类型的消息，Kafka在同一时刻能够负责多个Topic消息的的分发。
* Partition：Topic物理上的分组，一个Topic可以由多个分区组成，每个分区是一个有序的消息队列。
* Replication：副本数，每个分区的备份数量。避免数据丢失。
* Segment：分区在磁盘中存放是由多个Segment组成。
* offset：每个分区都由一系列有序的、不可变的消息组成，这些消息被连续的追加到分区中。分区中的每个消息都有一个自己的序列号叫做偏移量，用于分区唯一标识一条消息，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：每个消费者属于一个特定的消费组，不同消费组的消费者可以消费相同的消息。

2.3.3 Kafka消息存储

在Kafka中一个Topic通常存储的时同一类的消息，类似于在新闻中的某一类新闻，每个Topic是由多个分区组成，其中每个分区在磁盘中是以append log文件存放。

每个分区文件被切分为多个大小相等但消息数量不一定相等segment数据文件。这种存储方式让老的segment文件能快速被删除。每个分区只负责顺序读写文件，segment文件生命周期由服务端配置参数决定。

一个Segment的由两个文件组成，分别为index file和data file，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” 表示segment索引文件，“.log”表示segment的数据文件。segment文件命名规则：第一个segment从0开始，文件名也以0命名，之后的每个segment文件名为上一个segment文件中最后一条消息的offset值。文件名有19位数字字符长度，没有数字用0填充。



2.3.4 Kafka副本策略

数据同步

Kafka在0.8版本才开始提供分区的副本机制，之前的机制中如果Kafka Broker节点宕机，宕机的Broker节点上的全部分区都无法提供服务，而分区中又没有备份的数据，致使数据可靠性极低。所以0.8后提供了副本机制来保证Kafka Broker节点的故障转移。

引入副本策略以后，一个分区会有多个副本，而这时需要在这些副本之间选举出一个Leader，生产者和消费者只与这个Leader交互，其它副本作为Follower从Leader中拉取数据。

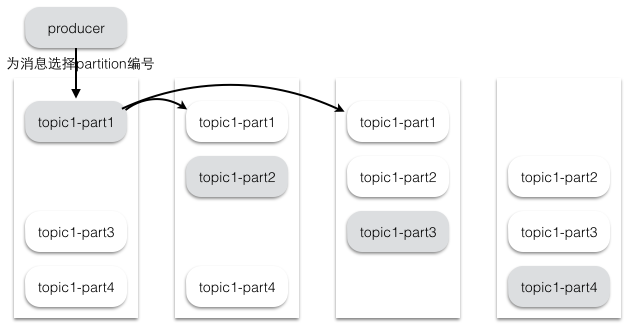


副本放置策略

为了做出更好的负载平衡，Kafka尽可能的将分区的副本均匀分配到集群的各个节点上。Kafka的副本分配算法如下：

* 将所有存活的Brokers和分区排序。
* 将第i个分区分配到第（i mod n）个Broker上面，这个分区的第一个副本存在存放分区的Broker上面。并会作为这个分区的优先副本（Leader）。
* 将第i个分区的第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个分区,每个分区拥有3个副本，那么分配情况如下图：



同步策略

生产者在发布消息到某个分区时，先访问ZooKeeper获取到该分区的Leader，生产者只会将该消息发送给该分区的Leader。Leader会更新本地的data数据文件和index数据文件。每个Follower都会从Leader中去拉取消息数据。这种方式就可以保证Follower中消息的存储的数据顺序与Leader中消息存储的顺序保持一致。Follower在拉取到该消息并写入文件以后，就会向Leader发送一条ACK。一旦Leader收到了ISR中的全部副本的ACK以后，就会认为该消息已经被commit了，Leader将提高HW的值并且向生产者发送ACK，消费者消费数据只能消费到HW（高水位）。

为了提升性能，每一个Follower在拉去到消息数据后就直接向Leader发送ACK，而不是在数据写入文件以后。因此，对已经被commit的消息，Kafka只能保证它被存于多个副本节点的内存当中，而不能保证它们以及被持久化写入到磁盘中，就不能完全保证这条条消息一定能被消费者消费到。

2.3.5 Kafka消息消费原理

一个Topic的一条消息只能被同一个消费组的一个消费者消费，不同消费组的不同消费者可以消费同一条消息。



作为一个消息系统，Kafka遵循了传统消息系统发送消息和消费消息的方式，选择由生产者向Broker 推送消息并由消费组从Broker拉取消息。

推送模式很难适应消费速率不同的消费者，因为消息发送速率控制权在Broker上。推送模式的目的是尽量以最快传输速度传递消息，这样就很容易造成消费者来不及处理消息，然后造成数据丢失和网络拥塞。而拉取模式的消费速率可以由消费者本身的消费能力来自主控制。

对于Kafka消费者而言，拉取的模式更合适。拉取模式可降低Broker的工作量，消费消息的速率由消费者自己来决定，同时消费者可以自己控制消费方式，可批量消费也可逐条消费。

2.4 SparkStreaming流处理

2.4.1 概述

Spark Streaming 作为Spark API核心的拓展，支持实时数据流并提供容错的、高吞吐量数据的处理。支持从多种数据源获取数据，包括Kafka、Flume、Twitter等。从数据源获取数据后，可以使用join、map、reduce、second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分布式文件系统、数据库等。

2.4.2 现状

当Hadoop第一次出现时，它提供了一个存储PB级数据的平台，并对这些数据执行批量查询以收集洞察力。该模型适用于许多用例，例如分析大量客户数据以获取有趣的模式。但是，并非所有数据都可以等待批量查询执行。Spark Streaming提供流式计算，这意味着数据实时处理是通过源进行流式处理的，从而可以执行许多新的用例。

2015年是流式处理的一年。大家考虑用Spark，多数原因是因为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Spark R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。

2.4.3 特性

SparkStreaming是一个对实时数据流进行高吞吐量、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。

2.4.3 执行流程

(1) Application运行于StreamigContext和SparkContext之上。

(2) 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。

(3) Receiver接受输入源并拆分为块（离散化），这些块其实就是RDD。

(4) 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。

(5) 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。

3.股评文本数据采集概述

3.1 网络爬虫概述

3.2 Java爬虫Jsoup技术

3.2.1

3.2.2

4.文本数据分析相关技术与算法概述

4.1 文本分词

4.1.1 中文分词概述

中文分词(Chinese Word Segmentation) 指的是将一个汉字序列或一句话切分成一个个词。分词便是将连续的字序列按照一定的规则重新组合成词序列的过程。在英文的语言中，单词之间是由空格作为分界符的，而中文只是字、句和段能通过明显的分界符来简单划界，就是词语没有一个形式上的分界符，即使英文语言也同样存在短语的划分问题，不过在词划分这一层面上，中文比之英文要复杂得多、困难得多。中文分词是文本挖掘的根本，对输入的一段中文进行中文分词，可以达到电脑自动识别语句含义的效果。

4.1.2 Ansj中文分词概述

4.2 分词数据清洗

4.2.1 数据清洗概述

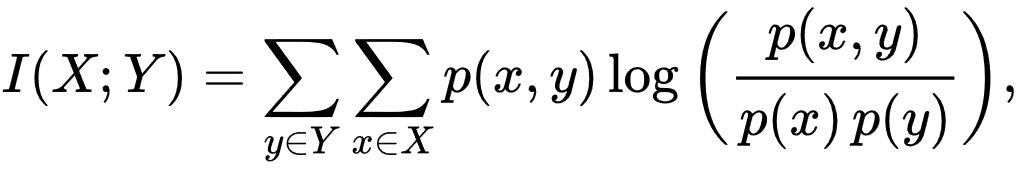
数据清洗指的是讲原来的数据通过我们自己定义的规则来进行筛选，清洗。

数据清洗从名字上来看就是对数据中的脏数据进行清洗，然后把无用的数据丢弃，只留下我们觉得好的数据，或则有用的数据。那什么样的数据脏数据，什么样的数据优势好的数据呢，这个是由项目中需要的数据类型，或者数据于该项目的相关性来决定的。比如本文是针对股票评论信息的研究，那么在这个项目中我们进行数据清理后的数据都必须能与股票相关联。否者就会被视为脏数据进行丢弃。当然数据清洗也包括清除掉重复的数据。

4.2.2 互信息过滤算法概述

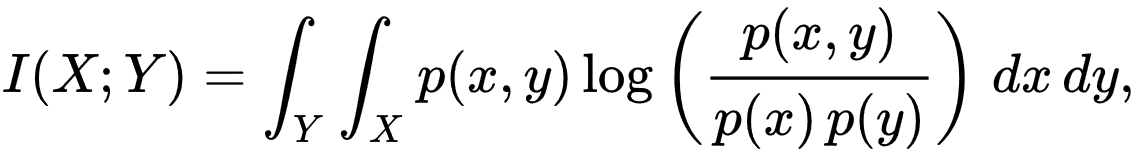
在概率论和信息论中，两个随机变量的互信息（Mutual Information，简称MI）或转移信息（Transinformation）是变量间相互依赖性的量度。不同于相关系数，互信息并不局限于实值随机变量，它更加一般且决定着联合分布 p(X,Y) 和分解的边缘分布的乘积 p(X)p(Y) 的相似程度。互信息是点间互信息（PMI）的期望值。互信息最常用的单位是bit。

一般地，两个离散随机变量 X 和 Y 的互信息可以定义为：



其中 p(x,y) 是 X 和 Y 的联合概率分布函数，而 p(x) p(x)分别是 X 和 Y 的边缘概率分布函数。

在连续随机变量的情形下，求和被替换成了二重定积分：

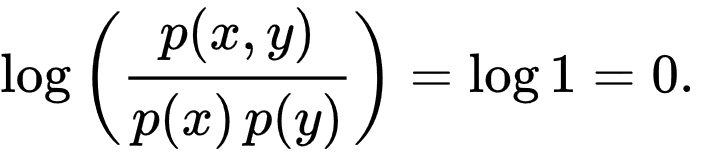


其中 p(x,y) 当前是 X 和 Y 的联合概率密度函数，而 p(x) 和 p(y) 分别是 X 和 Y 的边缘概率密度函数。

如果对数以 2 为基底，互信息的单位是bit。

直观上，互信息度量 X 和 Y 共享的信息：它度量知道这两个变量其中一个，对另一个不确定度减少的程度。例如，如果 X 和 Y 相互独立，则知道 X 不对 Y 提供任何信息，反之亦然，所以它们的互信息为零。在另一个极端，如果 X 是 Y 的一个确定性函数，且 Y 也是 X 的一个确定性函数，那么传递的所有信息被 X 和 Y 共享：知道 X 决定 Y 的值，反之亦然。因此，在此情形互信息与 Y（或 X）单独包含的不确定度相同，称作 Y（或 X）的熵。而且，这个互信息与 X 的熵和 Y 的熵相同。（这种情形的一个非常特殊的情况是当 X 和 Y 为相同随机变量时。）

互信息是 X 和 Y 联合分布相对于假定 X 和 Y 独立情况下的联合分布之间的内在依赖性。 于是互信息以下面方式度量依赖性：I(X; Y) = 0 当且仅当 X 和 Y 为独立随机变量。从一个方向很容易看出：当 X 和 Y 独立时，p(x,y) = p(x) p(y)，因此：



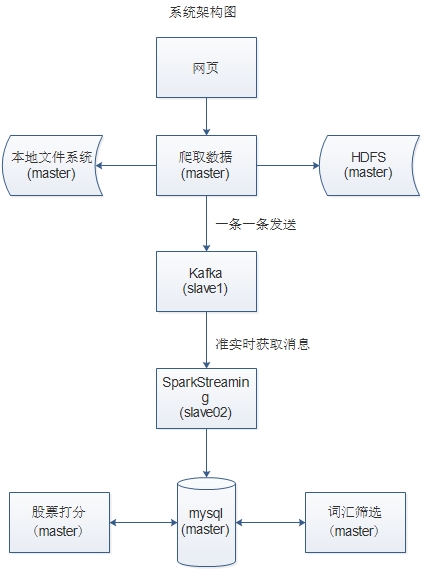
此外，互信息是非负的（即 I(X;Y) ≥ 0; 见下文），而且是对称的（即 I(X;Y) = I(Y;X)）。

5.系统设计概述

5.1 系统架构

5.1.1 系统架构概述

系统分为四个模块：爬取数据模块，实时分词模块，词汇筛选模块,股票打分模块。这四个运行在三台Linux Centos 7虚拟机上。三台虚拟机分别为master、slave1、slave2。按照传统搭建系统的方式，爬取数据，将数据传给后面的代码进行处理是我们容易理解的一种方式，但是在大数据量的情况下，程序间高度的耦合会造成不可挽回的局面，当前一步执行失败之后，程序后面的执行过程全部会跟着一起失败，这是我们一直想避免的，所以在程序设计上，我们追求的是高聚合、低耦合。在本系统中，采用了消息系统中间件，很好的达到了程序间的解耦，如果在消息的上一层出现了程序中的错误，并不会带向消息系统的下一层。当消息系统的下一层出现了错误，并不会将错误返回给消息系统的上一层，消息系统只负责了系统中数据的传输和存储。系统中同时提供了多种存储方式接口，爬取好的数据可以通过调用相关的接口，存入相关的文件系统中。当系统传输的数据量和需要处理的数据量很多的时候。系统中HDFS、Kafka和spark组件都可以进行横向扩展来增加系统的处理能力和处理速度。系统的架构图以及其分布如下：



5.1.2 系统流程简要概述

该系统通过从网页爬取股票的文本信息，在实时爬取的时候，将数据实时的发送给Kafka消息系统，Kafka消息系统的数据被SparkStreaming组建实时消费。爬取的数据也可以调用存本地系统和存分布式文件系统HDFS的接口。分词后的数据存入MYSQL中，存入MYSQL的数据被词汇筛选组件获取，通过一些过滤算法对词进行筛选，筛选过后调用股票打分组件，通过对词汇的打分，最终得出股票在各个层次的分数。

5.2 系统模块概述

5.2.1爬取数据模块

该模块通过Jsoup解析网页爬取相关股票评论信息，在对每一条信息进行处理、筛选之后。将该条信息发送给Kafka。在处理完并发送完所有数据后，可调用不同的接口将数据放入不同的文件系统中。

5.2.2 实时分词模块

该模块是一直运行在Linux Centos 7上的程序，程序一直保持活跃，当Kafka消息系统中有数据之后，就会从其中拉取消息下来进行分词处理。排除网络IO的影响，基本上和爬取数据保持同时进行。分词后的数据存入MYSQL数据库。

5.2.3 词汇筛选模块

该模块从MYSQL获取数据，进行两轮筛选，首先通过词汇模板进行筛选，将每支股票词汇模板前8的相关词汇进行保留，在通过使用互信息的方式将分词数据进行筛选。最终将筛选的数据存入MYSQL数据库。

5.2.4 股票打分模块

该模块从MYSQL获取数据，对词汇进行打分，打分的依据为该词汇的语句中是否存在褒义或者贬义词，最终获得每个词的最终分数，再将这些词汇通过手动划分层次，划分为三层。最后通过加权平均获得每支股票每个层面的得分，将数据存入MYSQL数据库。

5.3 数据库设计

5.3.1 MYSQL数据库概述

MySQL是一个关系型数据库管理系统。它的开发公司是瑞典AB。现在它属于甲骨文公司的产品之一。目前,世界上最流行的关系数据库管理系统之一是MySQL,而MySQL也是Web应用中ROBMS(关系数据库管理系统)的最佳应用。不同于将所有数据放在同一个大仓库中的是,MySQL中的数据将通过关系数据库存储在不同的表中,以提高速度并提高灵活性。MySQL所使用的SQL语言是用于访问数据库的最常用的标准化语言。它的软件有两种:社区版和商业版,即双重授权策略。MySQL是大多数中小型网站选择的,因为MySQL具有速度快、体积小、成本低、开发源代码等优点。

5.3.2 数据字典说明

stock表：

主要字段有：主键ID，股票代码，股票名称，行业名称，爬取标志，筛选标志，打分标志

comment表:

主要字段有：主键ID，股票代码，评论内容，创建时间

stock\_term表：

主要字段有：主键ID，股票代码，评论ID，句子ID，词号，词，词性，创建时间

screen\_term表：

主要字段有：主键ID，股票代码，评论ID，句子ID，词号，词，词性，创建时间

stock\_score表：

主要字段有：主键ID，股票代码，宏观分数，微观分数，市场分数，创建时间

5.3.3 数据库ER图设计

6.基于大数据平台的股评信息文本挖掘实现

6.1 分布式文件系统搭建

6.1.1 JDK的安装

在官网下载JDK1.8版本，下载完成后上传JDK文件至虚拟机，通过解压命令tar -zxvf解压文件。在环境变量中配置Java的环境变量，配置好之后执行 source /etc/profile应用配置，通过命令行输入java如果出现java相关的内容说明JDK安装成功。

6.1.2 HDFS安装

在官网下载Hadoop 2.6.5版本，下载完成后上传文件至虚拟机，通过解压命令tar -zxvf解压文件。本文中使用伪分布式安装，也就是只需要一台主机，但是可以提交相应的出口给其他程序访问。启动后三个角色都在同一台虚拟机上。

在对外提供访问地址时，一般不建议用IP地址，而是用主机名。一般情况在网络中，IP会出现冲突的问题，如果有很多软件在该主机上，当IP发生改变之后，所有用到IP的地方都需要修改。如果用主机名，只需要修改主机名和IP的映射关系就可以了。打开/etc/hosts文件在其中添加映射即可。

配置Hadoop：进入Hadoop配置文件目录，在hadoop-env.sh文件中，修改JAVA\_HOME为的JDK安装目录；创建入口：在core-site.xml文件中加入fs.defaultFS属性；修改副本数：在hdfs-site.xml文件中添加dfs.replication属性；配置环境变量；配置完成，首次启动之前要先对hadoop执行格式化，格式命令为:hdfs namenode -format;格式化之后通过启动命令：start-dfs.sh启动HDFS。查看进程，如果Namenode、DataNode、SecondaryNameNode三个角色都成存在的话，说明HDFS安装成功。对应文件的配置如下：

表5-1 HDFS相关配置

|  |  |  |
| --- | --- | --- |
| 文件名 | 属性名 | 属性值 |
| core-site.xml | fs.defaultFS | <https://localhost:9000> |
| hdfs-site.xml | dfs.replication | 1 |

6.1.3 HDFS优化配置

HDFS默认是将数据文件存放在/tmp目录下，所以有可能会数据丢失，所以我们在core-site.xml加上hadoop.tmp.dir属性的配置，由于目录格式发生了改变，所以需要重新对HDFS格式化。对应的优化配置如下：

表5-2 HDFS优化配置

|  |  |  |
| --- | --- | --- |
| 文件名 | 属性名 | 属性值 |
| core-site.xml | hadoop.tmp.dir | /data |

当我们启动hdfs时，会让我们输入三次登录密码，当我们的主机数很多时，会出现输多次密码的情况，所以一般情况下，为了操作方便，会采用免密。在任意目录下执行：ssh-keygen -t dsa -P '' -f ~/.ssh/id\_dsa。然后将公钥放在指定文件下：cat ~/.ssh/id\_dsa.pub >> ~/.ssh/authorized\_keys。

hdfs默认会给我们提供一个WEB界面，可以通过windows浏览器进行访问，但一般情况下，我们一般是访问不了的，因为我们没有关闭Linux防火墙，执行命令systemctl stop firewalld关闭防火墙。

6.2 消息系统中间件的搭建

6.3 数据爬取模块

6.3.1 网站以及地址解析

6.3.2 Jsoup提取网页关键数据

6.3.3 Kafka生产者接口

6.3.4 评论数据存本地文件接口

6.3.5 评论数据存HDFS接口

6.3.6 评论数据存MYSQL接口

6.4 大数据平台实时分词

6.4.1 Kafka消费者配置

6.4.2 Ansj分词

6.4.2 分词存MYSQL接口

6.5 词汇数据清洗

6.5.1 词汇模板过滤

6.5.2 互信息过滤

6.5.3 过滤词存MYSQL接口

6.6 股票打分

6.6.1 分层词汇打分

6.6.2 各个层面加权平均分

6.6.3 分数存数据库接口

6.7 结果分析

6.7.1 股票推荐

6.7.2 行业推荐

结 论

附 录

参考文献

致 谢