

大数据挖掘实验指导

—基于 DataOne 大数据实验平台

深圳大学计算机与软件学院

2021 年 3 月

目录

第一章 DataOne 大数据实验平台.....	1
1.1 DataOne 大数据实验平台简介.....	1
1.2 DataOne 大数据实验平台应用场景.....	1
1.2.1 数据仓库建设.....	1
1.2.2 离线批处理计算.....	1
1.2.3 实时数据处理.....	1
1.3 DataOne 大数据实验平台产品优势.....	1
1.3.1 降低门槛.....	1
1.3.2 便捷运维.....	2
1.3.3 智慧路由式数据集成.....	2
1.3.4 降低成本.....	2
1.3.5 强大的任务调度.....	2
第二章 Hadoop 集群计算基础.....	3
2.1 Hadoop 概述.....	3
2.1.1 Hadoop 简介.....	3
2.1.2 Hadoop 发展简史.....	3
2.1.3 Hadoop 的特征与优势.....	4
2.1.4 Hadoop 项目结构.....	4
2.2 Hadoop 安装部署.....	6
2.2.1 Hadoop 的安装模式.....	6
2.2.2 Linux 系统安装.....	7
2.2.3 Java 环境安装.....	7
2.2.4 Hadoop 安装.....	7
2.3 HDFS 分布式文件系统.....	8
2.3.1 HDFS 组成.....	8
2.3.2 HDFS 常用命令.....	9

2.4 MapReduce 分布式并行编程模型.....	11
2.4.1 MapReduce 架构	11
2.4.2 MapReduce 处理流程	12
2.4.3 MapReduce 编程实例	14
2.5 Hadoop Streaming.....	17
2.5.1 Hadoop Streaming 简介	17
2.5.2 Hadoop Streaming 应用实例.....	17
第三章 Spark 编程	19
3.1 Spark 概述	19
3.1.1 Spark 简介	19
3.1.2 Spark 架构	19
3.2 Spark 安装部署	21
3.2.1 Spark 下载	21
3.2.2 Spark 安装	21
3.2.3 测试 Spark 安装结果	22
3.3 Spark 编程基础	23
3.3.1 SparkSession 及其创建	23
3.3.2 RDD 简介	23
3.3.3 RDD 创建	24
3.3.4 RDD 操作	26
3.3.5 RDD 编程实例	27
3.3.6 Spark SQL 简介	28
3.3.7 DataFrame 创建	29
3.3.8 DataFrame 操作	30
第四章 机器学习库 MLlib	37
4.1 Spark MLlib 简介	37
4.2 基本统计	37
4.2.1 相关性.....	37
4.2.2 假设检验.....	39

4.3 特征处理.....	40
4.3.1 CountVectorizer.....	40
4.3.2 Word2Vec.....	41
4.3.3 TF-IDF	42
4.3.4 ChiSqSelector.....	45
4.3.5 PCA.....	46
4.4 分类与回归.....	47
4.4.1 逻辑斯蒂回归分类器.....	47
4.4.2 线性回归.....	49
4.5 聚类.....	51
4.5.1 K-means	51
4.5.2 GMM.....	52
第五章 教学实验案例.....	54
5.1 教学实验要求及内容.....	54
5.2 实验报告格式.....	55
5.3 教学实验案例.....	56
5.3.1 实验一 原材料采购策略制定	56
5.3.2 实验二 供应商聚类分析.....	70
5.3.3 实验三 客户反馈信息分析.....	87

第一章 DataOne 大数据实验平台

1.1 DataOne 大数据实验平台简介

大数据统一处理平台（DataOne）是基于开源社区，提供简单易用、一站式的大数据处理、计算、管理平台，覆盖集群部署与管理，数据同步，任务开发，任务管理开发，数据管理和数据运维等功能，使企业能轻松的完成数据采集、抽取、转换、建模、分析、挖掘、报表展示等数据处理的各个环节。DataOne 极大的降低了企业构建大数据中心的成本，使其能聚焦在业务层和数据价值变现上，占领商业先机。

1.2 DataOne 大数据实验平台应用场景

1.2.1 数据仓库建设

覆盖数据采集、存储、抽取、转换、建模、分析、挖掘、报表展现等数据处理的各个环节，可快速搭建企业数据仓库。

1.2.2 离线批处理计算

基于 Hadoop、Hive、Spark 等套件，提供对数据进行同步、抽取、转换、分析等离线数据处理功能，快速挖掘企业海量历史数据的商业价值。

1.2.3 实时数据处理

基于 Spark Streaming、Storm 等套件，实现对企业实时业务的风险监控与告警，比如：工业生产线的实时故障预警、网站的实时流量分析等应用场景。

1.3 DataOne 大数据实验平台产品优势

1.3.1 降低门槛

一站式、拖拽式的 IDE 开发，数据采集、抽取转换、分析、挖掘、报表展现等数据处理的各个环节，均可在 Web 式 IDE 上轻松完成开发。

1.3.2 便捷运维

管理和监控任务运行状态，可通过邮件、短信、云之家等方式及时告警，避免业务故障。

1.3.3 智慧路由式数据集成

支持 MySQL、Oracle、MongoDB、日志采集等各种数据源，且支持离线、实时、增量等各种方式的数据集成，数据集成工具由机器智慧选择。

1.3.4 降低成本

低门槛的数据集成、数据计算、数据分析和挖掘平台，业务专家也可以快速上手平台，从而降低企业人力成本。

1.3.5 强大的任务调度

支持多任务并发，支持小时、天、周、月等多种调度周期，支持小时、天、周、月的混合调度方式。

第二章 Hadoop 集群计算基础

2.1 Hadoop 概述

2.1.1 Hadoop 简介

Hadoop 是 Apache 软件基金会旗下的一个开源分布式计算平台，它为分布式环境提供了对海量数据进行处理的能力。Hadoop 是基于 Java 语言开发的，其具有很好的跨平台特性，并能部署在廉价的计算机集群上。Hadoop 的核心由两大部分组成，一是 HDFS（Hadoop Distributed File System）分布式文件系统，其用于分布式的文件存储；二是 MapReduce 分布式并行编程模型，其用于分布式的并行处理。

2.1.2 Hadoop 发展简史

Hadoop 起源于 2002 年的 Apache Nutch 项目（一个开源的网络搜索引擎），是 Apache Lucene 的子项目之一。最初，Hadoop 只是 Apache Lucene 项目的创始人 Doug Cutting 开发的一个文本搜索库；

在 2004 年，Apache Nutch 项目模仿 GFS（Google 文件系统）开发了自己的分布式文件系统 NDFS（Nutch Distributed File System），也就是 HDFS 的前身；

在 2004 年，谷歌公司在“操作系统设计与实现”（Operating System Design and Implementation, OSDI）会议上公开发表了题为 MapReduce: Simplified Data Processing on Large Clusters（Mapreduce: 简化大规模集群上的数据处理）的论文，阐述了 MapReduce 分布式编程思想；

在 2005 年，Apache Nutch 开源实现了谷歌的 MapReduce；

在 2006 年 2 月，由于 NDFS 和 MapReduce 在 Nutch 引擎中有着良好的应用，NDFS 和 MapReduce 开始独立出来，成为 Apache Lucene 项目的一个子项目，称为 Hadoop；

在 2008 年 1 月，Hadoop 正式成为 Apache 顶级项目，Hadoop 也逐渐开始被众多公司使用；

在 2008 年 4 月，Hadoop 打破世界纪录，成为最快排序 1TB 数据的系统，它采用一个由 910 个节点构成的集群进行运算，排序时间只用了 209 秒；

在 2009 年 5 月，Hadoop 更是把 1TB 数据排序时间缩短到 62 秒。Hadoop 从此名声大震，迅速发展成为大数据时代最具影响力的开源分布式开发平台，并成为事实上的大数据处理标准。

2.1.3 Hadoop 的特征与优势

Hadoop 是一个能够对大量数据进行分布式处理的软件框架，并且是以一种可靠、高效、可伸缩的方式进行处理的，它具有以下几个方面的特性：

（1）高可靠性：

集群中有多台机器，数据存储有多个备份，可以防止一个节点宕机而造成集群损坏。Hadoop 中还有备份机制和检验模式，会对出现问题的部分进行修复，或通过设置快照的方式在集群出现问题时回到之前的一个时间点。

（2）高扩展性：

Hadoop 是在可用的计算机集群间运行的，而为集群添加新的节点并不复杂，集群可以很容易通过节点的扩展来扩大集群。

（3）高效性：

Hadoop 能够在节点之间动态地移动数据，在数据所在节点进行并发处理，并保证各个节点的动态平衡，因此处理速度非常快。

（4）高容错性：

Hadoop 的分布式文件系统 HDFS 在存储文件时，会在多台机器上存储文件的备份副本。因此，当读取文件失败或某一台机器宕机时，系统会调用其他节点上的备份文件保证程序的顺利运行。

（5）成本低：

Hadoop 是免费的开源软件，不需要付费即可下载使用。

（6）可构建在廉价机器上：

Hadoop 的运行并不需要很高配置的机器，大部分的常用机器都可运行。

（7）支持多种编程语言：

Hadoop 中的 Streaming 框架能让任何编程语言编写的 MapReduce 程序在 Hadoop 集群上运行，从而使得 Hadoop 支持多种编程语言。

2.1.4 Hadoop 项目结构

Hadoop 的项目结构不断丰富发展，已经形成一个丰富的 Hadoop 生态系统，其包括了下图所示的各个组件。

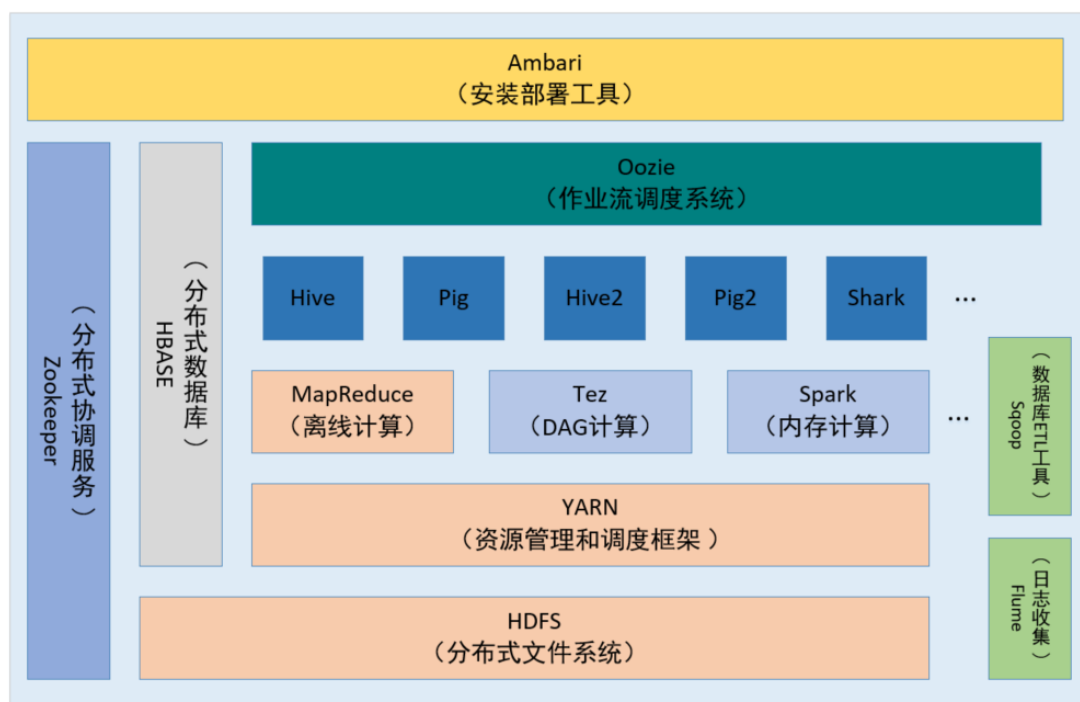


图 2.1 Hadoop 项目结构

不同的组件，提供的服务是不同的，下面我们对这些组件一一进行介绍。

(1) HDFS:

HDFS 是以分布式进行存储的文件系统，主要负责集群数据的存储和读取。

(2) YARN:

YARN 是负责资源管理和调度框架的，比如内存、CUP、带宽等等，为后面分布式并行处理提供基础。

(3) MapReduce:

MapReduce 是基于磁盘进行计算的、用于大规模数据集离线并行运算的编程模型，其包括 Map（映射）和 Reduce（规约）两部分。

(4) Tez:

Tez 是运行在 YARN 之上的下一代 Hadoop 查询处理框架。

(5) Spark:

Spark 是类似于 Hadoop MapReduce 的通用并行框架。它与 MapReduce 最大的不同在于 Spark 是基于内存进行计算的，而不是磁盘，它的处理速度比 MapReduce 快得多。

(6) Hive:

Hive 是建立在 Hadoop 上的数据仓库基础架构。它提供了一系列的工具，可存储、查询和分析存储在 Hadoop 中的大规模数据。其定义了一种类似于 SQL 的语言——HQL，用户可以通过简单的 HQL 语句将操作转化为复杂的 MapReduce。

(7) Pig:

Pig 是一个基于 Hadoop 的大规模数据分析平台，提供类似 SQL 的查询语言 Pig Latin。

(8) Oozie:

Oozie 是 Hadoop 上的工作流管理系统，可以调度 MapReduce、Pig、Hive 等。

(9) Zookeeper:

Zookeeper 提供了分布式协调一致性服务，可以解决分布式环境下的数据管理问题，如统一命名、状态同步、集群管理、配置同步等。

(10) HBase:

HBase 是 Hadoop 上的非关系型的分布式数据库。它提供了对大规模数据的随机、实时的读写操作，主要用于大数据量且有快速随机访问的需求，以及需要及时响应的需求等等。

(11) Flume:

Flume 是一个高可用的，高可靠的，分布式的海量日志采集、聚合和传输的系统。

(12) Sqoop:

Sqoop 是用于在 Hadoop 与传统数据库之间进行数据传递的。

(13) Ambari:

Ambari 是一个 Hadoop 快速部署工具，支持 Apache Hadoop 集群的供应、管理和监控。

2.2 Hadoop 安装部署

2.2.1 Hadoop 的安装模式

Hadoop 一共有三种安装模式：

(1) 单机模式:

只在一台机器上运行，存储是采用本地文件系统，没有采用分布式文件系统 HDFS。

(2) 伪分布式模式:

存储采用分布式文件系统 HDFS，但是，HDFS 的名称节点和数据节点都在同一台机器上。

(3) 完全分布式模式:

存储采用分布式文件系统 HDFS，而且，HDFS 的名称节点和数据节点位于不同机器上。

2.2.2 Linux 系统安装

Hadoop 的运行平台是 GNU/Linux, Hadoop 已在有 2000 个节点的 GNU/Linux 主机组成的集群系统上得到验证。当然, 也可以使用 Windows, 但是 Windows 平台是作为开发平台支持的。由于分布式操作尚未在 Win32 平台上充分测试, 所以还不作为一个生产平台被支持。

在这里, 我们可以选择在 Windows 系统下安装 Linux 虚拟机, 或者直接安装 Linux 双系统。如果电脑配置较好, 建议可以选择安装 Linux 虚拟机, 这样安装的操作会比较简单, 而且系统运行速度与双系统相差也不大。在此次项目中, 我们采用 Linux 虚拟机下的 Ubuntu18 (64 位) 作为系统环境。

2.2.3 Java 环境安装

因为 Hadoop 是用 Java 编写的, 所以 Hadoop 的运行必须依靠 Java 环境。在 Linux 系统中安装 Java 环境, 我们只需将下载到的 JDK 安装包解压到一个自己指定的文件目录下, 并在环境变量配置文件 .bashrc 中添加一些关于 Java 安装目录的环境变量即可。

在安装完成后, 我们可以在终端中输入 `java -version` 来检验是否安装成功。若安装成功, 终端会输出如下图所示的结果。

```
java version "1.8.0_162"
Java(TM) SE Runtime Environment (build 1.8.0_162-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.162-b12, mixed mode)
```

2.2.4 Hadoop 安装

(1) 单机模式与伪分布式模式的安装

首先, 我们先要去 Hadoop 官网下载 Hadoop 安装包, 将其解压到一个指定的文件目录下 (在这里, 我们的安装目录为 `/usr/local/`)。解压后, 在该目录中会得到一个文件夹, 这个文件夹名比较复杂, 还包含了详细版本号 (如 `hadoop-3.1.3`), 建议将其重命名为 `hadoop`, 这样可以方便未来我们输入执行命令。

接下来, 我们使用 `chown -R` 命令, 让当前用户拥有使用该 `hadoop` 文件的权限。至此, 如果是要以单机模式安装 Hadoop 的话已经安装完毕。我们可以在安装目录 (`/usr/local/hadoop`) 下执行 `./bin/hadoop version` 命令来检查 Hadoop 是否可用, 如果可用, 其会输出如下图所示的版本信息。

```
Hadoop 3.1.3
Source code repository https://gitbox.apache.org/repos/asf/hadoop.git -r ba631c436b806728f8ec2f54ab1e289526c90579
Compiled by ztang on 2019-09-12T02:47Z
Compiled with protoc 2.5.0
From source with checksum ec785077c385118ac91aadd5ec9799
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-3.1.3.jar
```

若要以伪分布式模式进行安装, 则还需要修改 `hadoop` 中的 2 个配置文件——`core-site.xml` 和 `hdfs-site.xml`, 并执行 NameNode 的格式化。

(2) 完全分布式模式的安装

而对于 Hadoop 完全分布式模式的安装，我们需要进行的工作大致如下：

- 1、选定一台机器作为 NameNode；
- 2、在 NameNode 上安装 Java 环境、安装 SSH 服务端、配置 SSH 无密码登陆；
- 3、在 NameNode 上安装 Hadoop，并完成配置；
- 4、在其他 DataNode 上安装 Java 环境、安装 SSH 服务端、配置 SSH 无密码登陆；
- 5、将 NameNode 上的“/usr/local/hadoop”目录（已安装并配置好的 Hadoop 文件）复制到其他 DataNode 上；
- 6、在 NameNode 上开启 Hadoop。

在本次项目中，我们所使用的是以完全分布式模式安装的 Hadoop。

2.3 HDFS 分布式文件系统

2.3.1 HDFS 组成

HDFS 是以分布式进行存储的文件系统，主要负责集群数据的存储和读取。HDFS 文件系统主要包括一个 NameNode（名称节点）、一个 Secondary NameNode（第二名称节点），和多个 DataNode（数据节点），其结构组成如下图所示。

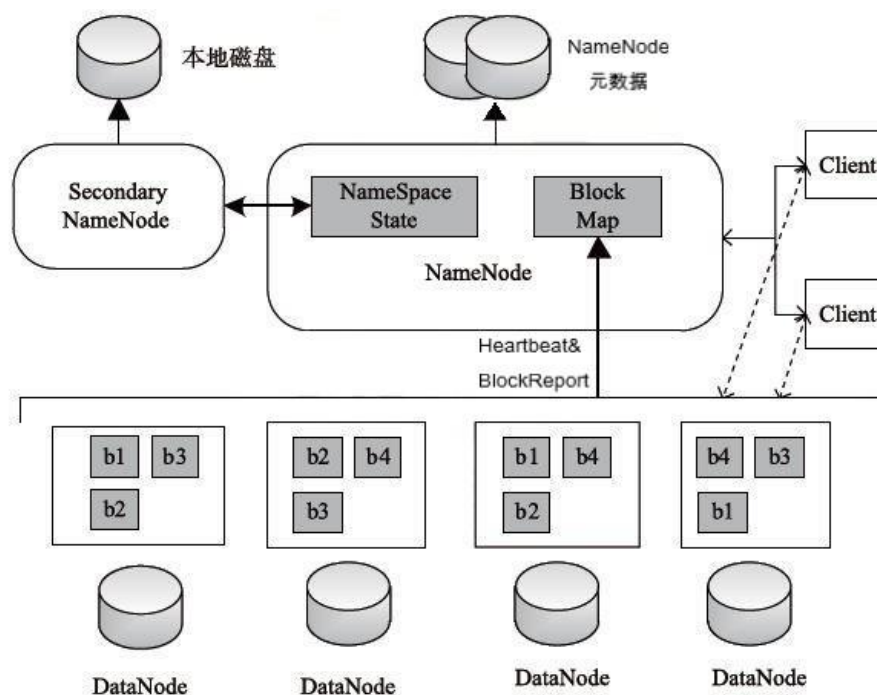


图 2.2 HDFS 结构组成

它们各自的用途如下：

- (1) NameNode 用于存储元数据以及处理客户端发出的请求；
- (2) Secondary NameNode 用于备份 NameNode 的数据；
- (3) DataNode 是真正存储数据的地方，在 DataNode 中，文件以数据块的形式进行存储。

2.3.2 HDFS 常用命令

在 Linux 系统中，我们可以通过使用 shell 命令来与 HDFS 进行交互。在这里，共有 3 种 shell 命令方式可供选择：

- 1、**hdfs dfs**（只能适用于 HDFS 文件系统）
- 2、**hadoop dfs**（只能适用于 HDFS 文件系统）
- 3、**hadoop fs**（适用于任何不同的文件系统，比如本地文件系统和 HDFS 文件系统）

在接下来的介绍中，我们统一使用 **hdfs dfs** 这一种方式。

在 **hadoop** 安装目录（**/usr/local/hadoop**）下执行命令 **./bin/hdfs dfs**，我们可以查看到 HDFS 所支持的所有命令。

```
[ -appendToFile <localsrc> ... <dst> ]
[ -cat [-ignoreCrc] <src> ... ]
[ -checksum <src> ... ]
[ -chgrp [-R] GROUP PATH... ]
[ -chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH... ]
[ -chown [-R] [OWNER][:[GROUP]] PATH... ]
[ -copyFromLocal [-f] [-p] [-l] [-d] [-t <thread count>] <localsrc> ... <dst> ]
[ -copyToLocal [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst> ]
[ -count [-q] [-h] [-v] [-t <storage type>]] [-u] [-x] [-e] <path> ... ]
[ -cp [-f] [-p | -p[topax]] [-d] <src> ... <dst> ]
[ -createSnapshot <snapshotDir> [<snapshotName>]]
[ -deleteSnapshot <snapshotDir> <snapshotName> ]
[ -df [-h] [<path> ...]]
[ -du [-s] [-h] [-v] [-x] <path> ... ]
[ -expunge ]
[ -find <path> ... <expression> ... ]
[ -get [-f] [-p] [-ignoreCrc] [-crc] <src> ... <localdst> ]
[ -getfacl [-R] <path> ]
[ -getfattr [-R] {-n name | -d} [-e en] <path> ]
[ -getmerge [-nl] [-skip-empty-file] <src> <localdst> ]
[ -head <file> ]
[ -help [cmd ...]]
[ -ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [-e] [<path> ...]]
[ -mkdir [-p] <path> ... ]
[ -moveFromLocal <localsrc> ... <dst> ]
[ -moveToLocal <src> <localdst> ]
[ -mv <src> ... <dst> ]
[ -put [-f] [-p] [-l] [-d] <localsrc> ... <dst> ]
[ -renameSnapshot <snapshotDir> <oldName> <newName> ]
[ -rm [-f] [-r|-R] [-skipTrash] [-safely] <src> ... ]
[ -rmdir [--ignore-fail-on-non-empty] <dir> ... ]
[ -setfacl [-R] [{-b|-k} {-m|-x <acl_spec>} <path>][--set <acl_spec> <path>]]
[ -setfattr {-n name [-v value] | -x name} <path> ]
[ -setrep [-R] [-w] <rep> <path> ... ]
[ -stat [format] <path> ... ]
[ -tail [-f] [-s <sleep interval>] <file> ]
[ -test [-defsz] <path> ]
[ -text [-ignoreCrc] <src> ... ]
```

同时,我们也可以在 hadoop 安装目录(/usr/local/hadoop)下执行命令./bin/hdfs dfs -help,来查看某一命令的具体使用方法。比如,我们可以使用以下命令来查看 put 命令的具体用法。

```
./bin/hdfs dfs -help put
```

```
-put [-f] [-p] [-l] [-d] <localsrc> ... <dst> :
Copy files from the local file system into fs. Copying fails if the file already
exists, unless the -f flag is given.
Flags:

-p Preserves access and modification times, ownership and the mode.
-f Overwrites the destination if it already exists.
-l Allow DataNode to lazily persist the file to disk. Forces
  replication factor of 1. This flag will result in reduced
  durability. Use with care.

-d Skip creation of temporary file(<dst>. COPYING ).
```

(1) HDFS 目录操作命令

在 Hadoop 安装完第一次使用时,我们首先要在 HDFS 中创建一个用户目录。比如当前系统的用户名为 SZU,则需要先在 HDFS 上创建一个名为 SZU 的用户。

创建用户目录的操作可以使用-mkdir 命令来完成,具体的命令如下:

```
./bin/hdfs dfs -mkdir -p /user/SZU
```

这里,“-p”表示如果是多级目录,则父目录和子目录一起创建,这里“/user/SZU”就是一个多级目录,因此必须使用参数“-p”,否则会出错。

要查看 HDFS 中的内容,我们可以使用-ls 命令来完成。比如,要查看 SZU 用户目录下的所有内容,可以采用以下命令:

```
./bin/hdfs dfs -ls /user/SZU
```

如果要列出 HDFS 上的所有目录,可以使用如下命令:

```
./bin/hdfs dfs -ls
```

要删除 HDFS 中的某个目录(或文件),我们可以使用-rm 命令来完成。比如,要删除 SZU 这一个用户目录,我们可以使用以下命令:

```
./bin/hdfs dfs -rm -r /user/SZU
```

这里,“-r”参数表示删除“/user/SZU”目录及其子目录下的所有内容。如果要删除的一个目录包含了子目录,则必须使用“-r”参数,否则会执行失败。

(2) HDFS 文件操作命令

要将本地的文件上传到 HDFS 文件系统中,我们可以使用-put 命令。比如,我们可以使用以下命令,将保存在本地/home/SZU 目录下的 data.csv 文件,上传到 HDFS 中名为 SZU 的用户目录:

```
./bin/hdfs dfs -put /home/SZU/data.csv /user/SZU
```

要查看 HDFS 文件系统中某一文件的内容,我们可以使用-cat 命令。比如,我们可以使

用以下命令，查看 HDFS 中 SZU 用户目录下的 data.csv 文件：

```
./bin/hdfs dfs -cat /user/SZU/data.csv
```

要将 HDFS 文件系统中的文件下载到本地，我们可以使用 -get 命令。比如，我们可以使用以下命令，将 HDFS 中 SZU 用户目录下的 data.csv 文件，保存到本地的 /home/SZU/download 目录下：

```
./bin/hdfs dfs -get /user/SZU/data.csv /home/SZU/download
```

2.4 MapReduce 分布式并行编程模型

2.4.1 MapReduce 架构

MapReduce 是基于磁盘进行计算的、用于大规模数据集离线并行运算的编程模型。MapReduce 体系结构主要由四个部分组成，分别是：Client、JobTracker、TaskTracker 以及 Task。

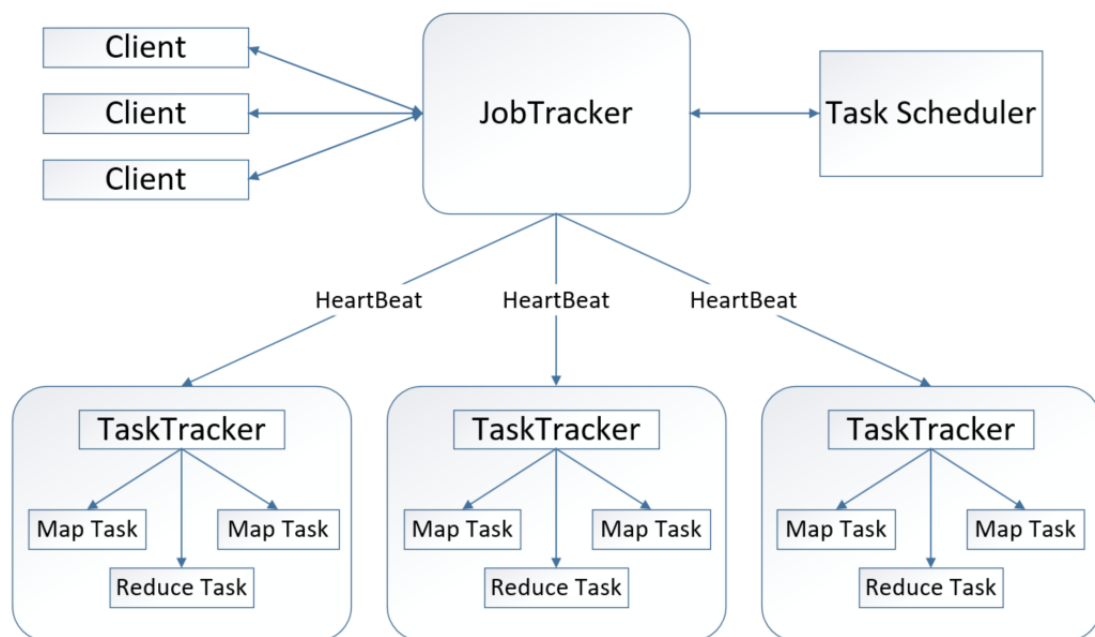


图 2.3 MapReduce 架构

(1) Client:

用户编写的 MapReduce 程序通过 Client，可以提交到 JobTracker 端。同时，用户也可通过 Client 提供的一些接口查看作业运行状态。

(2) JobTracker:

JobTracker 负责资源监控和作业调度。它会监控所有 TaskTracker 与 Job 的健康状况，以及跟踪任务的执行进度、资源使用量等信息，并将这些信息告诉任务调度器 (TaskScheduler)。

任务调度器（TaskScheduler）会在有资源出现空闲时，选择合适的任务分配给这些资源去执行。

（3）TaskTracker:

TaskTracker 会周期性地通过“心跳”将本节点上资源的使用情况和任务的运行进度汇报给 JobTracker，同时接收 JobTracker 发送过来的命令并执行相应的操作（如启动新任务、杀死任务等）。

（4）Task:

Task 分为 Map Task 和 Reduce Task 两种，均由 TaskTracker 启动。

2.4.2 MapReduce 处理流程

MapReduce 采用的是“分而治之”的处理策略，将复杂的、运行于大规模集群上的并行计算过程高度地抽象为两个函数：Map 和 Reduce。

在 MapReduce 的处理流程中，一个存储在分布式文件系统的大规模数据集，会被切分成许多独立的分片，这些分片可以被多个 Map 任务并行处理。而 Map 任务生成的结果会继续作为 Reduce 任务的输入，最终由 Reduce 任务输出最后的结果。MapReduce 大致的简化工作流程图如下所示：

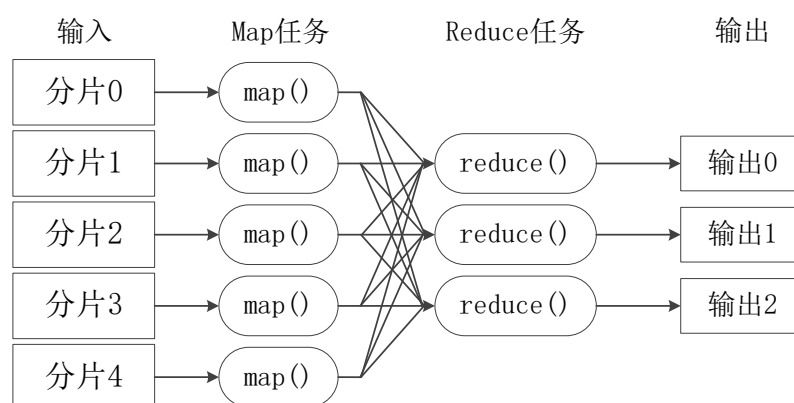


图 2.4 MapReduce 大致工作流程

在这里要特别注意，根据 MapReduce 的处理流程，不同的 Map 任务之间不会进行通信，不同的 Reduce 任务之间也不会发生任何信息交换。因此，我们使用 MapReduce 来处理某一个任务时，该任务必须要满足一个前提条件：**待处理的数据集要可以分解成多个小数据集，并且这些小数据集能完全并行的处理，其结果不受其他小数据集的影响。**

虽然 Hadoop 框架是用 Java 实现的，但是 MapReduce 应用程序不一定要用 Java 来写（也可以使用 Python、C++ 等等）。MapReduce 中 Map 和 Reduce 这两个函数的详细说明如下：

函数	输入	输出	说明
Map	$\langle k_1, v_1 \rangle$ 例如: $\langle 1, "a\ a\ b" \rangle$	$List(\langle k_2, v_2 \rangle)$ 例如: $\langle "a", 1 \rangle$ $\langle "a", 1 \rangle$ $\langle "b", 1 \rangle$	首先, 将小数据集进一步解析成一批 $\langle key, value \rangle$ 对, 即 $\langle k_1, v_1 \rangle$, 并输入 Map 函数中进行处理。 然后, Map 函数中每一个输入的 $\langle k_1, v_1 \rangle$ 会转化成一批 $\langle k_2, v_2 \rangle$ 进行输出, 即 $List(\langle k_2, v_2 \rangle)$ 。这些 $\langle k_2, v_2 \rangle$ 是 MapReduce 计算的中间结果。
Reduce	$\langle k_2, List(v_2) \rangle$ 例如: $\langle "a", \langle 1, 1 \rangle \rangle$	$\langle k_3, v_3 \rangle$ 例如: $\langle "a", 2 \rangle$	输入的中间结果 $\langle k_2, List(v_2) \rangle$ 中的 $List(v_2)$ 表示是一批属于同一个 k_2 的 value。 Reduce 函数会将这些中间结果 $\langle k_2, List(v_2) \rangle$ 中具有相同键的键值对以某种特定的方式组合起来, 输出组合后的最终结果 $\langle k_3, v_3 \rangle$ 。

MapReduce 的设计理念是“计算向数据靠拢”，而不是“数据向计算靠拢”。这样，就可以避免因移动数据而产生大量的网络传输开销。MapReduce 各个执行阶段的详细流程图如下所示：

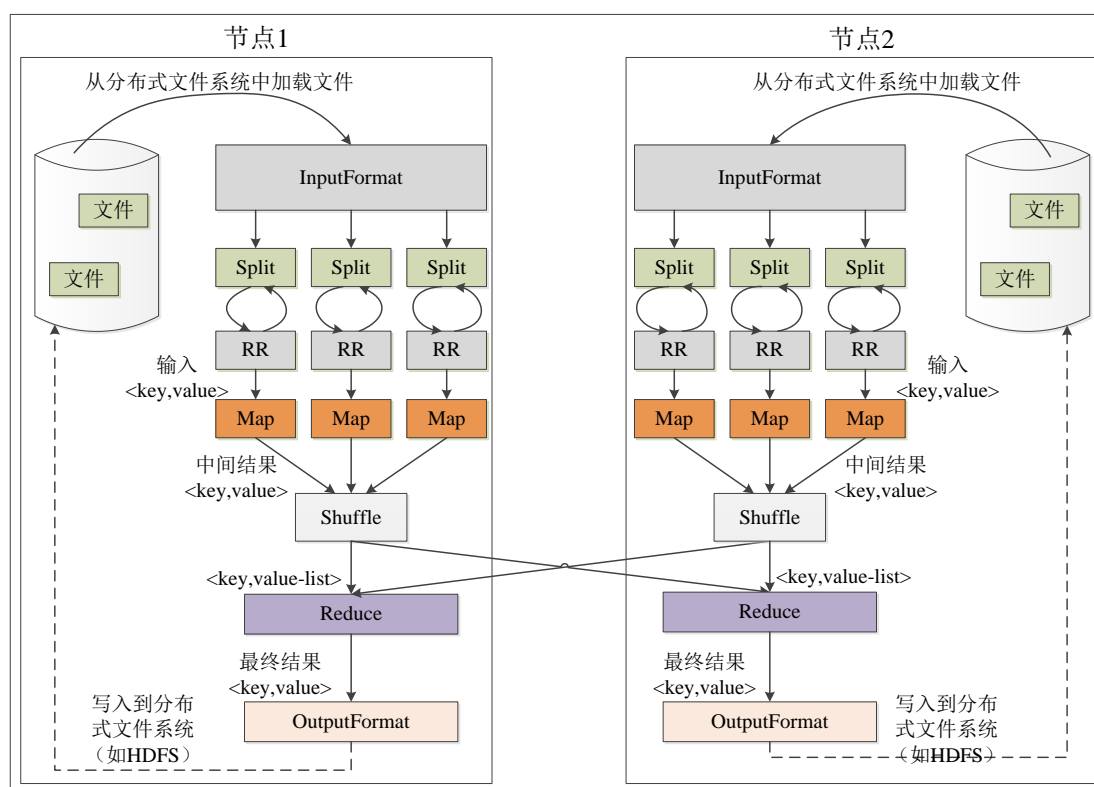


图 2.5 MapReduce 的完整执行流程

在这里，InputFormat 模块首先对输入文件的格式进行检验，检验通过后，会将文件逻辑切分成多个 Split（不是物理上实际的切分，只是记录了要处理的数据的位置和长度信息）。之后，RR（RecordReader）会根据 Split 中的信息来加载数据，并转化为适合 Map 任务读取的键值对 $\langle key, value \rangle$ ，输入给 Map 处理。

Map 处理完后输出的结果是无序的一堆<key, value>, 并不能直接作为 Reduce 的输入, 其会先通过 Shuffle 模块进行分区、排序、合并、归并等操作, 得到一系列<key, List(value)>形式的键值对, 再输入给 Reduce 处理。例如, Map 处理后得到的键值对为: <“a”,1>、<“b”,1>、<“a”,1>, 经过 Shuffle 模块处理后得到<“a”,<1,1>>、<“b”,1>, 再输入给 Reduce 处理。

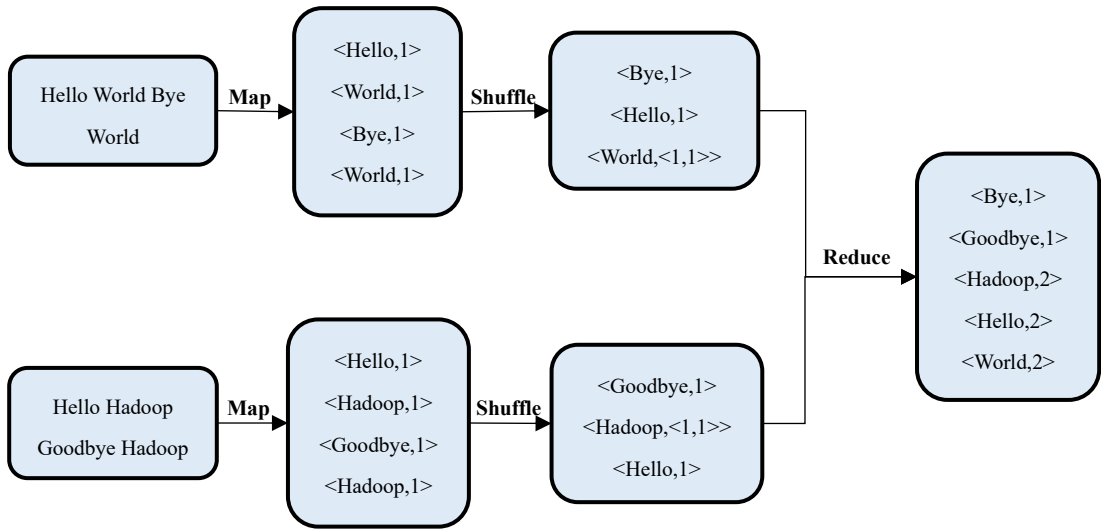
最后, OutoutFormat 模块会验证输出目录是否已经存在, 以及输出结果的类型是否符合要求等, 如果验证通过, 其会将 Reduce 的结果写入到 HDFS 中。

2.4.3 MapReduce 编程实例

在这里, 我们通过一个简单的单词词频统计程序作为样例, 来阐述 MapReduce 程序的编程思路与实现过程。该程序的输入输出如下所示:

输入	输出
text1.txt: Hello World Bye World text2.txt: Hello Hadoop Goodbye Hadoop	Bye 1 Goodbye 1 Hadoop 2 Hello 2 World 2

根据 MapReduce 处理流程的思想, 我们可以用以下方式来实现该单词词频统计程序。



该 MapReduce 程序 Java 实现的代码框架如下:

```

public class WordCount {
    //Map 模块
    public static class MyMapper extends Mapper<Object,Text,Text,IntWritable>{ .....
    }

    //Reduce 模块
    public static class MyReducer extends Reducer<Text,IntWritable,Text,IntWritable>{ .....
    }

    //Main 方法
    public static void main(String[] args) throws Exception{ .....
    }
}

```

其具体代码实现如下：

```

import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount{
    public static class MyMapper extends Mapper<Object,Text,Text,IntWritable>{
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(Object key, Text value, Context context)
            throws IOException,InterruptedException{
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()){
                word.set(itr.nextToken());
                context.write(word,one);
            }
        }
    }
}

```

```

public static class MyReducer extends Reducer<Text,IntWritable,Text,IntWritable>{
    private IntWritable result = new IntWritable();
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException,InterruptedException{
        int sum = 0;
        for (IntWritable val : values){
            sum += val.get();
        }
        result.set(sum);
        context.write(key,result);
    }
}

public static void main(String[] args) throws Exception{
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf,args).getRemainingArgs();
    if (otherArgs.length != 2){
        System.err.println("Usage: wordcount <in> <out>");
        System.exit(2);
    }
    Job job = new Job(conf,"word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(MyMapper.class);
    job.setReducerClass(MyReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job,new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job,new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true)?0:1);
}
}

```

要运行该 MapReduce 程序，我们需要进行以下几个步骤：

1、如果是在伪分布式或完全分布式下运行该 MapReduce 程序，首先要将保存在本地的输入文件（/home/SZU/input/text1.txt、/home/SZU/input/text2.txt）上传到 HDFS：

```
./bin/hdfs dfs -put /home/SZU/input/* ./input
```

2、编译 java 代码（WordCount.java）：

```
javac WordCount.java
```

3、将编译后得到的所有.class 文件打包为 jar 包（WordCount.jar）：

```
jar -cvf WordCount.jar *.class
```

4、启动 Hadoop 运行 MapReduce 程序：

```
./bin/hadoop jar WordCount.jar WordCount ./input ./output
```

当程序运行完成后，我们就可以在./output 目录下查看到程序运行的结果了。

2.5 Hadoop Streaming

2.5.1 Hadoop Streaming 简介

在前面我们有提及到，虽然 Hadoop 框架是用 Java 实现的，但是 MapReduce 应用程序不一定要用 Java 来写。这是因为 Hadoop 中含有 Streaming 这个工具，它能帮助用户创建和运行一类特殊的 MapReduce 作业，这些特殊的 MapReduce 作业是由一些可执行文件或脚本文件来充当 mapper 和 reducer 的。换句话说，也就是用户可以通过 Hadoop Streaming 来使用任意语言（如 python）编写、运行 MapReduce 作业。

2.5.2 Hadoop Streaming 应用实例

在 Hadoop Streaming 中，mapper 和 reducer 从标准输入 stdin 中按行读取输入，然后将输出发送到标准输出 stdout 中。比如，上面 MapReduce 编程实例中的 WordCount 单词词频统计程序，我们也可以通过 Hadoop Streaming 使用 Python 语言编程实现。

其中，Mapper 部分的代码（Mapper.py）如下：

```
#!/usr/bin/env python
import sys

for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print '%s\t%s' % (word, 1)
```

Reducer 部分的代码（Reducer.py）如下：

```
#!/usr/bin/env python
import sys

current_word = None
current_count = 0
word = None

for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    try:
        count = int(count)
    except ValueError:
        continue
    if current_word == word:
        current_count += count
    else:
        if current_word:
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

要使用 Streaming 运行该 MapReduce 程序，首先要用 `chmod` 命令为 `Mapper.py` 和 `Reducer.py` 这两个文件授权（否则会报错说权限不够）：

```
chmod 777 Mapper.py
```

```
chmod 777 Reducer.py
```

然后，再执行以下命令使用 Hadoop Streaming 运行该 Python 编写的 MapReduce 程序：

```
./bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-*.jar \  
-mapper Mapper.py -reducer Reducer.py -input Input/* -output Output/*
```

该命令中各项参数的含义如下：

- **-mapper Mapper.py** 是指定 `Mapper.py` 作为 mapper
- **-reducer Reducer.py** 是指定 `Reducer.py` 作为 reducer
- **-input Input/***是指定 `Input` 文件夹中的内容作为该程序的输入
- **-output Output/***是指定该程序的输出文件保存到 `Output` 文件夹中

当程序运行完成后，我们就可以在 `./output` 目录下查看到程序运行的结果了。

第三章 Spark 编程

3.1 Spark 概述

3.1.1 Spark 简介

Spark 是专为大规模数据处理而设计的快速通用的计算引擎。Spark 最初由美国加州伯克利大学的 AMP 实验室于 2009 年开发，是基于内存计算的大数据并行计算框架，可用于构建大型的、低延迟的数据分析应用程序。

Spark 在继承了 Hadoop MapReduce 的优点的同时，也完善了 Hadoop MapReduce 的一些不足。相比于 Hadoop MapReduce，Spark 主要有如下两大优势：

- Spark 提供了多种数据集操作类型，编程模式比 Hadoop MapReduce 更灵活；
- Spark 提供了内存计算，可将中间结果放到内存中，之后的迭代计算都可以直接使用内存中的中间结果进行运算，避免了从磁盘中频繁读取数据，对于迭代运算效率更高；

Spark 的生态系统主要包含了 Spark、Spark SQL、Spark Streaming、MLlib 和 GraphX 等组件，这些组件的适用场景如下：

Spark 生态系统中的组件	适用场景
Spark	复杂的批量数据处理
Spark SQL	基于历史数据的交互式查询
Spark Streaming	基于实时数据流的数据处理
MLlib	基于历史数据的数据挖掘
GraphX	图结构数据的处理

3.1.2 Spark 架构

Spark 运行架构包括驱动器(Driver)、集群资源管理器(Cluster Manager)、主进程(Master)运行作业任务的工作节点 (Worker Node)、和每个工作节点上负责具体任务的执行进程 (Executor)。在一个独立集群当中，其详细的运行框架图如下所示：

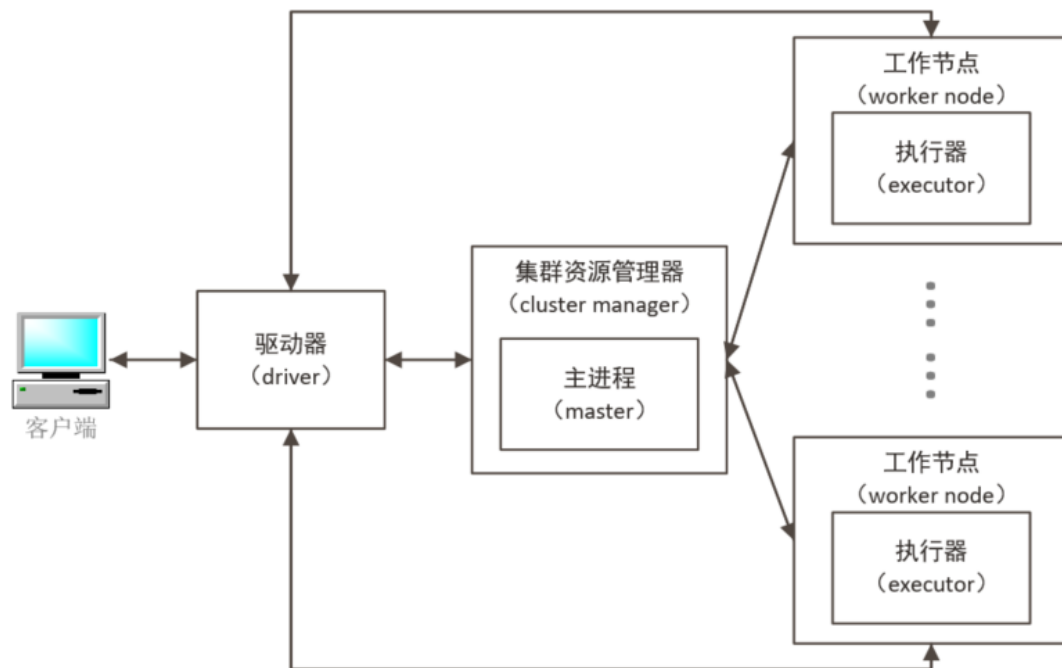


图 3.1 Spark 架构图

驱动器 (driver) 是 Spark 客户端用来提交应用的进程。在这里, 驱动器主要执行两项任务。第一个是创建 SparkSession 对象: SparkSession 对象代表 Spark 集群的一个连接, 在 Spark 2.0 版本后 SparkSession 成为了所有 Spark 程序的统一入口。第二个是确定应用的执行计划: 驱动器接收到应用处理作业作为输入后, 会根据应用程序要执行的所有操作, 创建出由节点组成的有向无环图 (DAG), 其中每个节点表示一个转化或者计算的步骤。

执行器(executor)是运行有向无环图(DAG)中描述的任务的进程, 这些执行器(executor)是运行在工作节点 (worker note) 上的。

主进程 (master) 是向集群申请资源, 并把资源交给驱动器的进程。主进程还会跟踪工作节点的状态, 以及监控它们的执行速度。

集群资源管理器 (cluster manager) 负责监控工作节点, 并在主进程发起请求时在工作节点上预留资源。当工作节点预留到资源时, 主进程就会把这些集群资源以执行器的形式交给驱动器使用。

主进程和集群资源管理器可以是分开的进程, 也可以合在一个进程中。比如上图中以独立集群模式运行 Spark 时, 主进程也会提供集群资源管理器的功能, 也就是主进程和集群资源管理器是合在一个进程中的。

3.2 Spark 安装部署

3.2.1 Spark 下载

在安装 Spark 之前，我们首先要具备 Java 和 Hadoop 环境。同时，因为我们后面 Spark 的编程都是基于 Python 语言的，所以我们还要对 Python 3 环境进行安装，Python 3 环境的安装非常简单，在此不再赘述。当以上环境都配置好了，我们就可以对 Spark 进行下载安装了。

我们进入 Spark 官方下载网站 (<http://spark.apache.org/downloads.html>)，在“Choose a Spark release”中选择好要安装的版本，并在“Choose a package type”需要选择“Pre-build with user-provided Apache Hadoop”（因为我们在前面已经安装好了 Hadoop，所以选择这个安装包类型）。最后，我们再点击“Download Spark”后面的超链接就可以进行下载了。

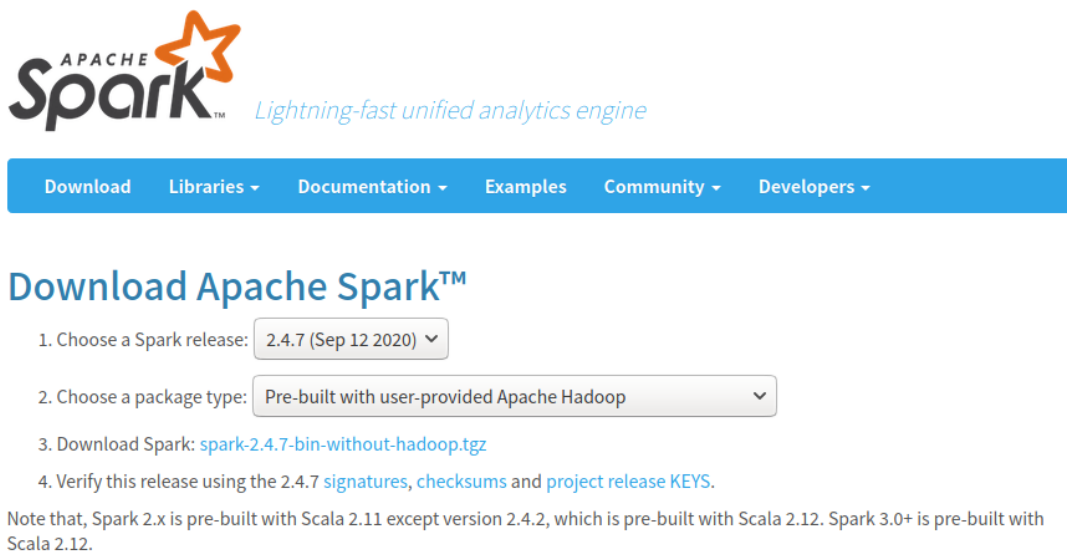


图 3.2 Spark 下载

3.2.2 Spark 安装

Spark 部署模式主要有四种：Local 模式（单机模式）、Standalone 模式（使用 Spark 自带的简单集群管理器）、YARN 模式（使用 YARN 作为集群管理器）和 Mesos 模式（使用 Mesos 作为集群管理器）。

在这里，我们以单机模式进行 Spark 的安装。具体的安装分为如下两个步骤：

（1）解压 Spark 安装包：

首先，我们将下载到的安装包解压到我们指定的安装目录下（在这里我们选择的安装目录为/usr/local/）。

```
sudo tar -zxf ~/下载/spark-2.4.0-bin-without-hadoop.tgz -C /usr/local/
```

然后，为了未来执行命令的方便，我们跳转至安装目录下，将文件夹 `spark-2.4.7-bin-without-hadoop` 重命名为 `spark`。

```
cd /usr/local
```

```
sudo mv ./spark-2.4.0-bin-without-hadoop/ ./spark
```

最后，我们再为授予用户（在这里 SZU 是用户名）该文件夹的操作权限。

```
sudo chown -R SZU:SZU ./spark
```

（2）修改 Spark 的配置文件 `spark-env.sh`：

在安装完成后，我们还需要修改 Spark 的配置文件 `spark-env.sh`，以便 Spark 能在 Hadoop 分布式文件系统 HDFS 中读写数据。

```
vim /usr/local/spark/conf/spark-env.sh
```

通过 vim 在该配置文件第一行添加以下配置信息：

```
export SPARK_DIST_CLASSPATH=$(/usr/local/hadoop/bin/hadoop classpath)
```

（3）设置必要的环境变量：

除此之外，我们还要设置必要的环境变量，

```
vim ~/.bashrc
```

通过 vim 在该配置文件开头添加以下配置信息：

```
export SPARK_HOME=/usr/local/spark
```

```
export PYTHONPATH=$SPARK_HOME/python:$SPARK_HOME/python/lib/py4j-0.10.4-  
src.zip:$PYTHONPATH
```

```
export PYSPARK_PYTHON=python3
```

```
export PATH=$HADOOP_HOME/bin:$SPARK_HOME/bin:$PATH
```

在上面的配置项中，`PYTHONPATH` 这一行有个 `py4j-0.10.4-src.zip`，这个 zip 文件的版本号一定要和“`/usr/local/spark/python/lib`”目录下的 `py4j-0.10.4-src.zip` 文件保持版本一致。

最后，再执行以下命令使环境变量立即生效：

```
source ~/.bashrc
```

至此，Spark 的单机模式已安装完成。

3.2.3 测试 Spark 安装结果

Spark 中自带有一个测试程序，我们可以通过以下命令运行该测试程序：

```
/usr/local/spark/bin/run-example SparkPi 2>&1 | grep "Pi is"
```

如果 Spark 安装成功，在终端会输出如下结果：

```
Pi is roughly 3.137555687778439
```

3.3 Spark 编程基础

3.3.1 SparkSession 及其创建

SparkSession 是 Spark 集群的一个连接，是所有 Spark 程序的统一入口，其在程序整个运行过程中都会使用。因此，我们在编写 Spark 程序时，要首先进行 SparkSession 的创建。

SparkSession 创建的代码如下：

```
from pyspark.sql import SparkSession

#SparkSession 的创建
SparkSession spark = SparkSession.builder
    .master("your_master")
    .appName("your_appName")
    .config("spark.some.config.option", "config-value")
    .getOrCreate()
```

在这里，master 指定了 Spark 主程序；appName 指定了应用名称；config 指定了一些运行参数。在接下来的示例中，spark 这个变量都特指这里的 SparkSession。

3.3.2 RDD 简介

RDD（Resilient Distributed Dataset）是 Spark 编程中最基本的数据对象，它是 Spark 应用中的数据集。一个 RDD 就是一个分布式对象集合，本质上是一个只读的分区记录集合，每个 RDD 可分成多个分区，每个分区就是一个数据集片段，并且一个 RDD 的不同分区可以被保存到集群中不同的节点上，从而可以在集群中的不同节点上进行并行计算。

RDD 的操作分为“转换”（Transformation）和“动作”（Action）两种类型。转换操作就是对现有的 RDD 进行操作来创建新的 RDD，动作操作就是在数据集上进行运算，返回计算值的操作。

要注意，RDD 提供了一种高度受限的共享内存模型，即 RDD 是只读的记录分区的集合，不能直接修改，只能基于稳定的物理存储中的数据集创建 RDD，或者通过在其他 RDD 上执行确定的转换操作（如 map、join 和 group by）而创建得到新的 RDD。因此，RDD 提供的转换接口都非常简单，都是类似 map、filter、groupBy、join 等粗粒度的数据转换操作，而不是针对某个数据项的细粒度修改。

表面上 RDD 的功能很受限、不够强大，实际上 RDD 已经被实践证明可以高效地表达许多框架的编程模型（比如 MapReduce、SQL、Pregel）。

RDD 典型的执行过程如下：

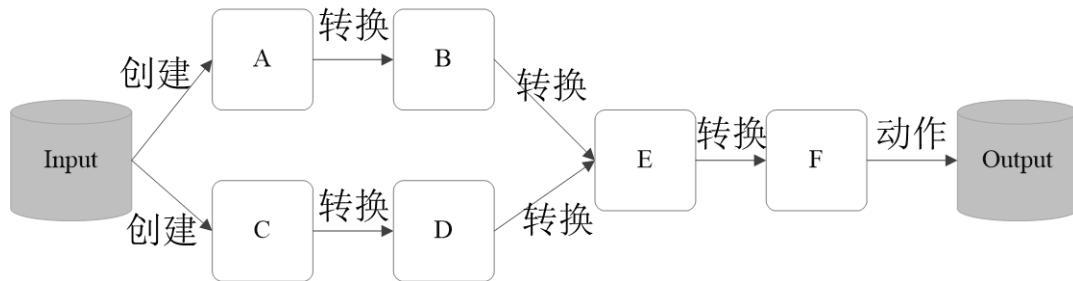


图 3.3 RDD 执行过程

- (1) 从外部数据源进行 RDD 的创建；
- (2) RDD 经过一系列的转换（Transformation）操作，每一次都会产生不同的 RDD，供给下一个转换操作使用；
- (3) 最后一个 RDD（Action）经过动作操作进行转换，并输出到外部数据源。

3.3.3 RDD 创建

RDD 是 Spark 编程中最基本的数据对象，要对某一数据集进行操作，我们应先用该数据集创建一个 RDD，然后再在此基础上进行后续实验操作。RDD 可以通过文件(包括 txt 文件、json 文件、csv 文件等)，或并行集合（列表）进行创建。

(1) 通过文件创建 RDD

Spark 可以采用 SparkSession 中 sparkContext 的 `textFile()` 方法来从文件中加载数据，并创建 RDD。该方法的参数为该文件的 URI，这个 URI 可以是本地文件系统的地址，也可以是 HDFS 分布式文件系统的地址等等。

```
#从本地文件创建 RDD
sc=spark.sparkContext
dataRDD=sc.textFile("file:///usr/home/SZU/data.txt")
```

```
#从 HDFS 中的文件创建 RDD
sc=spark.sparkContext
dataRDD=sc.textFile("hdfs:///user/SZU/data.txt")
```

(2) 通过并行集合（列表）创建 RDD

Spark 也可以采用 SparkSession 中 sparkContext 的 `parallelize()` 方法，利用一个已经存在的并行集合（列表）创建 RDD。

```
#通过并行集合（列表）创建 RDD
sc=spark.sparkContext
list = ["Hadoop","Spark","Hive","Spark"]
dataRDD = sc.parallelize(list)
```

虽然普通的 RDD 中可以包含任何类型的对象，但是其并不便于我们继续进行分析操作。因此，Spark 操作中经常会用到“键值对 RDD”（Pair RDD），用于完成聚合计算。普通 RDD 里面存储的数据类型是 Int、String 等，而“键值对 RDD”里面存储的数据类型是“键值对”（key-value）。

键值对 RDD 与 RDD 一样，也是通过文件(包括 txt 文件、json 文件、csv 文件等)，或并行集合（列表）进行创建的。

(1) 通过文件创建键值对 RDD

```
#从本地文件创建 RDD
sc=spark.sparkContext
dataRDD=sc.textFile("file:///usr/home/SZU/data.txt")
pairRDD = dataRDD.flatMap(lambda line : line.split(" ")).map(lambda word : (word,1))
```

```
#从 HDFS 中的文件创建 RDD
sc=spark.sparkContext
dataRDD=sc.textFile("hdfs:///user/SZU/data.txt")
pairRDD = dataRDD.flatMap(lambda line : line.split(" ")).map(lambda word : (word,1))
```

(2) 通过并行集合（列表）创建键值对 RDD

```
#通过并行集合（列表）创建 RDD
sc=spark.sparkContext
list = ["Hadoop","Spark","Hive","Spark"]
dataRDD = sc.parallelize(list)
pairRDD = dataRDD.map(lambda word : (word,1))
```

3.3.4 RDD 操作

RDD 被创建好以后，在后续使用过程中一般会发生两种操作。转换（Transformation）操作，就是基于现有的数据集创建一个新的数据集；行动（Action）操作，就是在数据集上进行运算，返回计算值。

（1）常见的转化（Transformation）操作有：

- `filter(func)`:
筛选出满足函数 `func` 的元素，并返回一个新的数据集
- `map(func)`:
将每个元素传递到函数 `func` 中，并将结果返回为一个新的数据集
- `flatMap(func)`:
与 `map()`相似，但每个输入元素都可以映射到 0 或多个输出结果
- `groupByKey()`:
应用于(K,V)键值对的数据集时，返回一个新的(K, Iterable)形式的数据集
- `reduceByKey(func)`:
应用于(K,V)键值对的数据集时，返回一个新的(K, V)形式的数据集，其中的每个值是将每个 `key` 传递到函数 `func` 中进行聚合

（2）常见的行动（Action）操作有：

- `count()`:
返回数据集中的元素个数
- `collect()`:
以数组的形式返回数据集中的所有元素
- `first()`:
返回数据集中的第一个元素
- `take(n)`:
以数组的形式返回数据集中的前 `n` 个元素
- `reduce(func)`:
通过函数 `func`（输入两个参数并返回一个值）聚合数据集中的元素
- `foreach(func)`:
将数据集中的每个元素传递到函数 `func` 中运行

(3) 常见的键值对转化操作有：

- `reduceByKey(func)`:
使用 `func` 函数合并具有相同键的值
- `groupByKey()`:
对具有相同键的值进行分组
- `keys()`:
会把键值对 RDD 中的 `key` 返回形成一个新的 RDD
- `values()`:
会把键值对 RDD 中的 `value` 返回形成一个新的 RDD
- `sortByKey()`:
返回一个根据键排序的键值对 RDD
- `mapValues(func)`:
对键值对 RDD 中的每个 `value` 都应用一个函数，但是，`key` 不会发生变化
- `join(pair RDD)`:
将两个键值对 RDD 内连接，和关系数据库中的内连接一样
- `leftOuterJoin(pair RDD)`:
将两个键值对 RDD 左外连接，和关系数据库中的左外连接一样
- `rightOuterJoin(pair RDD)`:
将两个键值对 RDD 右外连接，和关系数据库中的右外连接一样

3.3.5 RDD 编程实例

下面，我们就以一道简单的题目为例，展示 RDD 编程的思路。假设有一组键值对 ("spark",2)、("hadoop",6)、("hadoop",4)、("spark",6)，键值对的 `key` 表示图书名称，`value` 表示某天图书销量，现在我们要计算的是每个键对应的平均值，也就是计算每种图书每天的平均销量。

首先，我们调用 `parallelize()` 方法生成键值对 RDD，生成的 `rdd` 类型是 `RDD[(String, Int)]`。针对构建得到的 `rdd`，我们调用 `mapValues()` 函数，把 `rdd` 中的每个键值对 (`key,value`) 的 `value` 部分进行修改，把 `value` 转换成键值对 (`value,1`)，这里的数值 1 表示这个 `key` 在 `rdd` 中出现了 1 次。因为我们最终要计算每个 `key` 对应的平均值，所以，必须记住这个 `key` 出现了几次，最后用 `value` 的总和除以 `key` 的出现次数，就是这个 `key` 对应的平均值。

```

from pyspark.sql import SparkSession

#SparkSession 的创建
SparkSession spark = SparkSession.builder
    .master("your_master")
    .appName("your_appName")
    .config("spark.some.config.option", "config-value")
    .getOrCreate()

#调用 parallelize()方法生成键值对 RDD
rdd = sc.parallelize([("spark",2),("hadoop",6),("hadoop",4),("spark",6)])

# 把 value 转换成键值对(value,1),例如('spark',2)=>('spark',(2,1))
temp = rdd.mapValues(lambda x: (x,1))

# 合并具有相同键的值。比如, 合并('spark',(2,1))和('spark',(4,1))为('spark',(6,2))
temp1 = temp.reduceByKey(lambda x,y : (x[0]+y[0],x[1] + y[1]))

# 得到每种书的每天平均销量, 比如('spark',(6,2))=>('spark',6/2=3)
temp2 = temp1.mapValues(lambda x: x[0]/x[1])

# 打印结果
temp2.collect()

```

程序运行后最终打印出的结果为: `[('hadoop', 5.0), ('spark', 4.0)]`

3.3.6 Spark SQL 简介

Spark SQL 在 RDD 的基础上, 增加了 SchemaRDD (即带有 Schema 信息的 RDD), 使用户可以在 Spark SQL 中执行 SQL 语句, 数据既可以来自 RDD, 也可以来自 Hive、HDFS、Cassandra 等外部数据源, 还可以是 JSON 格式的数据。

Spark SQL 目前支持 Scala、Java、Python 三种语言, 支持 SQL-92 规范。从 Spark1.2 升级到 Spark1.3 以后, Spark SQL 中的 SchemaRDD 变为了 DataFrame, DataFrame 相对于 SchemaRDD 有了较大改变,同时提供了更多好用且方便的 API。

因此, DataFrame 的推出, 让 Spark 具备了处理大规模结构化数据的能力。其不仅比原有的 RDD 转化方式更加简单易用, 而且还拥有更高的计算性能。

DataFrame 是一种以 RDD 为基础的分布式数据集, 也就是分布式的 Row 对象的集合 (每个 Row 对象代表一行记录)。它提供了详细的结构信息, 也就是我们经常说的模式 (schema), 我们可以清楚地知道该数据集中包含哪些列、每列的名称和类型。DataFrame 与 RDD 的对比如下图所示:

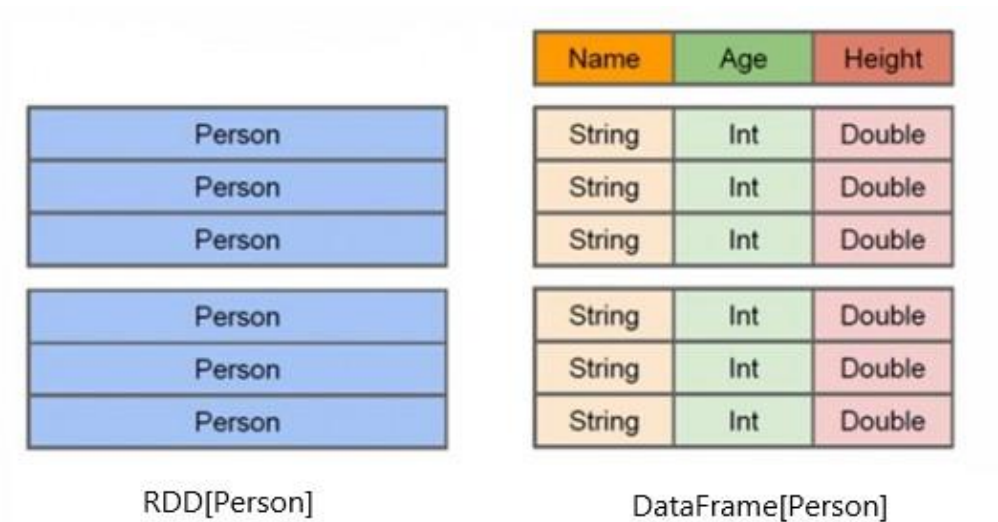


图 3.4 DataFrame 与 RDD 的对比

DataFrame 与 RDD 的执行机制是一样的，都是基于 DAG（有向无环图），并遵循惰性机制。

3.3.7 DataFrame 创建

与 RDD 的创建一样，DataFrame 的创建也是通过 SparkSession 来创建的。DataFrame 的创建方法有以下几种：

（1）通过现有 RDD 创建 DataFrame

Spark 可以采用 SparkSession 中的 `createDataFrame()` 方法，通过现有的 RDD 来创建 DataFrame。

```
#通过现有 RDD 创建 DataFrame（myrdd 是现有的 RDD）
df=spark.createDataFrame(myrdd)
```

（2）通过并行集合（列表）创建 DataFrame

Spark 也可以采用 SparkSession 中 `createDataFrame()` 方法，通过现有的并行集合（列表）来创建 DataFrame。在这里，我们需要两个列表，第一个是描述的是 DataFrame 里面的内容，第二个描述的是 DataFrame 中每一列内容的字段。

```
#通过并行集合（列表）创建 DataFrame。
df=spark.createDataFrame(
    [('Michael',20),('Andy',30),('Justin',18)],
    ["name","age"]
)
```

(3) 通过文件（txt 文件、json 文件、csv 文件）创建 DataFrame

Spark 还可以采用 SparkSession 中 read 的 text()方法、json()方法、csv()方法来从文件中加载数据，并创建 RDD。该方法的参数为该文件的 URI，这个 URI 可以是本地文件系统的地址，也可以是 HDFS 分布式文件系统的地址等等。

```
#从本地 txt 文件创建 RDD
df=spark.read.text("file:///usr/home/SZU/data.txt")
```

```
#从 HDFS 中的 txt 文件创建 RDD
df=spark.read.text("hdfs:///user/SZU/data.txt")
```

```
#从本地 json 文件创建 RDD
df=spark.read.json("file:///usr/home/SZU/data.json")
```

```
#从 HDFS 中的 json 文件创建 RDD
df=spark.read.json("hdfs:///user/SZU/data.json")
```

```
#从本地 csv 文件创建 RDD
df=spark.read.csv("file:///usr/home/SZU/data.csv")
```

```
#从 HDFS 中的 csv 文件创建 RDD
df=spark.read.csv("hdfs:///user/SZU/data.csv")
```

(4) 将 DataFrame 转化为 RDD

同时,我们也可以使用 DataFrame 自带的 rdd()方法,将现有的 DataFrame 转化为 RDD。

```
#将 DataFrame 转化为 RDD (df 是现有的 DataFrame)
myRDD=df.rdd()
```

3.3.8 DataFrame 操作

DataFrame 被创建好以后,和 RDD 一样,其各种变换操作也采用惰性机制,分为“转换”(Transformation)和“行动”(Action)两种类型。

在这里，我们假设有如下一个简单的 DataFrame，其名称为 df，其内容如下所示：

name(string)	age(int)
Michael	20
Andy	30
Justin	18

接下来，我们通过该 DataFrame 来对一些常用的 DataFrame 操作进行介绍：

(1) printSchema():

该方法用于打印 DataFrame 的模式信息。

```
df.printSchema()
```

```
root
|-- name: string (nullable = true)
|-- age: integer (nullable = true)
```

(2) columns:

该方法返回由 DataFrame 的列名组成的一个列表。

```
df.columns
```

```
['name', 'age']
```

(3) dtypes:

该方法返回由 DataFrame 的列名及其数据类型组成的一个二元列表。

```
df.dtypes
```

```
[('name', 'string'), ('age', 'int')]
```

(4) show(n, truncate):

该方法用于将 DataFrame 以表格的形式打印输出。参数 n (int 类型) 用于设置要打印多少行，默认 n=20；参数 truncate (bool 类型) 用于指定是否要截取过长的字符串并在单元格内右对齐，默认 truncate=True。

```
df.show()
```

```
+-----+
|  name | age |
+-----+
| Michael | 20 |
|   Andy  | 30 |
|  Justin | 18 |
+-----+
```

(5) **select(col):**

该方法用于选择指定的列（可以是多列）构造并返回一个新的 DataFrame。

```
df2=df.select(df.name)
df2.show()
```

```
+-----+
|  name |
+-----+
| Michael |
|   Andy  |
|  Justin |
+-----+
```

(6) **drop(col):**

该方法用于返回一个删除了 col 列的新 DataFrame。

```
df3=df.drop(df.age)
df3.show()
```

```
+-----+
|  name |
+-----+
| Michael |
|   Andy  |
|  Justin |
+-----+
```

(7) **filter(condition):**

该方法用于返回满足给定条件 condition（可以是多个条件）的那些行的新 DataFrame。

```
df4=df.filter(df.age>=20)
df4.show()
```

```
+-----+-----+
|   name | age |
+-----+-----+
| Michael | 20 |
|   Andy  | 30 |
+-----+-----+
```

(8) distinct():

该方法用于返回去除了 DataFrame 中重复的行的新 DataFrame。

```
no_distinct_df = spark.createDataFrame(
    [('Michael',20),('Justin',18),('Justin',18)],
    ["name", "age"]
)
no_distinct_df.show()
```

```
+-----+-----+
|   name | age |
+-----+-----+
| Michael | 20 |
|  Justin | 18 |
|  Justin | 18 |
+-----+-----+
```

```
distinct_df = no_distinct_df.distinct()
distinct_df.show()
```

```
+-----+-----+
|   name | age |
+-----+-----+
| Michael | 20 |
|  Justin | 18 |
+-----+-----+
```

(9) sort(order):

该方法用于返回对 DataFrame 中指定的列（可以是多列），进行升序(asc)/降序(desc)排序后的新 DataFrame。

```
#根据姓名升序对 DataFrame 进行排序
df6=df.sort(df.name.asc())
df6.show()
```

name	age
Andy	30
Justin	18
Michael	20

(10) **groupBy(col):**

该方法用于将输入的 DataFrame 按照参数 col 指定的列（可以是多列）进行分组，使用新的分组结果创建新的 DataFrame。

```
#统计每个年龄的人的个数
df7=df.groupBy('age').count()
#对结果根据年龄进行升序排序
df7=df7.sort(df7.age.asc())
df7.show()
```

age	count
18	1
20	1
30	1

```
#统计所有人的平均年龄
df8=df.groupBy().avg('age')
df8.show()
```

avg (age)
22.666666666666668

(11) **join(otherDataFrame, on, how):**

该方法用于将当前 DataFrame 与 otherDataFrame 进行连接操作，并将连接后的结果作为一个新 DataFrame 返回。其中，参数 otherDataFrame 是指要与当前 DataFrame 进行连接的另外一个 DataFrame；参数 on 是指一个列、一组列，或者一个表达式，用于连接操作的求值；how 参数是指定连接的类型（与关系型数据库的连接类型一样），包括 inner、left、right 和 full，默认 how="inner"。

```
#创建一个 DataFrame
DF = spark.createDataFrame(
    [('Alice',18,160),('Tom',20,180)],
    ["name","age","height"]
)
DF.show()
```

name	age	height
Alice	18	160
Tom	20	180

```
#创建另外一个 DataFrame
otherDF = spark.createDataFrame(
    [('Alice',45),('Tom',75)],
    ["name","weight"]
)
otherDF.show()
```

name	weight
Alice	45
Tom	75

```
#对这两个 DataFrame 进行内连接操作
joinedDF=DF.join(otherDF,['name'])
joinedDF.show()
```

name	age	height	weight
Tom	20	180	75
Alice	18	160	45

(12) write.csv(URI):

该方法用于将 DataFrame 中的数据写入到指定的 URI 中，这个 URI 可以是本地文件系统的地址，也可以是 HDFS 分布式文件系统的地址等等。

```
#将 DataFrame 中的数据写入到指定的 URI  
df.write.csv("hdfs:///home/SZU/saved_dataframe.csv")
```


第四章 机器学习库 MLlib

4.1 Spark MLlib 简介

在传统的机器学习算法中，因为受限于单机存储（存储空间小）以及单机运算（运算速度慢），往往只适用于小数据集上的机器学习。之后，随着 Hadoop HDFS 分布式文件系统和 Hadoop MapReduce 架构的出现，使得海量数据的存储和高速并行运算成为可能。

由于 MapReduce 是基于磁盘进行运算的，而且通常情况下机器学习算法的过程都是迭代计算的，即本次计算的结果要作为下一次迭代的输入。因此在这个过程中，如果使用 MapReduce，我们会不断地把中间结果存储磁盘，然后在下一次计算的时候又不断地从磁盘重新读取。这大量的磁盘 IO 开销，无疑使得 MapReduce 在进行分布式的机器学习算法非常耗时。这对于迭代频发的机器学习算法来说，显然是致命的性能瓶颈。

而 Spark 是立足于内存计算的，其天然的适应于迭代频发的机器学习算法。Spark MLlib（Machine Learning Library），就是 Spark 提供的一个机器学习库，它提供了很多常用机器学习算法的分布式实现。开发者只需对 Spark 编程，和机器学习算法的基本原理有所了解，就可以通过简单地调用 MLlib 中相应的 API（Application Programming Interface），来实现基于海量数据的分布式机器学习过程。

4.2 基本统计

4.2.1 相关性

计算两个数据系列之间的相关性是“统计”中的常见操作。在 `pyspark.ml.stat.Correlation` 中，目前所支持的相关统计方法有 Pearson 相关性和 Spearman 相关性。

下面是一个计算身高与体重的相关性的简单例子：

```

from pyspark.ml.linalg import DenseMatrix, Vectors
from pyspark.ml.stat import Correlation

#构造 DataFrame，每个特征向量中第一个元素是身高，第二个元素是体重。
dataset = [
    [Vectors.dense([149.0,81.0])],
    [Vectors.dense([150.0,88.0])],
    [Vectors.dense([153.0,87.0])],
    [Vectors.dense([155.0,99.0])],
    [Vectors.dense([160.0,91.0])],
    [Vectors.dense([155.0,89.0])],
    [Vectors.dense([160.0,95.0])],
    [Vectors.dense([150.0,90.0])]
]
dataset = spark.createDataFrame(dataset, ['features'])
dataset.show()

```

```

+-----+
|  features  |
+-----+
|[149.0, 81.0]|
|[150.0, 88.0]|
|[153.0, 87.0]|
|[155.0, 99.0]|
|[160.0, 91.0]|
|[155.0, 89.0]|
|[160.0, 95.0]|
|[150.0, 90.0]|
+-----+

```

```

#计算 pearson 相关性
pearsonCorr = Correlation.corr(dataset, 'features', method='pearson').collect()[0][0]
print(pearsonCorr)

```

```

DenseMatrix([[1.          , 0.61240306],
              [0.61240306, 1.          ]])

```

```

#计算 spearman 相关性
spearmanCorr = Correlation.corr(dataset, 'features', method='spearman').collect()[0][0]
print(spearmanCorr)

```

```

DenseMatrix([[1.          , 0.73951618],
              [0.73951618, 1.          ]])

```

4.2.2 假设检验

假设检验是一种强大的统计工具，可用来确定结果是否具有统计学意义，以及该结果是否为偶然发生。在 `pyspark.ml.stat.ChiSquareTest` 中，目前支持的假设检验方法有 Pearson 卡方检验，其可以用来进行适配度检测和独立性检测。

适配度检测：指验证一组观察值的次数分配是否异于理论上的分配，其 H_0 假设（虚无假设）为一个样本中已发生事件的次数分配会服从某个特定的理论分配。

独立性检测：指验证从两个变量抽出的配对观察值组是否互相独立，其 H_0 假设（虚无假设）为两个变量呈统计独立性。

下面我们通过一个对身高体重和性别之间进行独立性检验的简单例子，来阐述 Spark MLlib 中假设检验的使用。

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.stat import ChiSquareTest

#样本标签为性别（0 代表女性、1 代表男性）
#样本特征为身高（特征向量中第一个元素）和体重（特征向量中第二个元素）
dataset = [
    (0, Vectors.dense([149.0, 81.0])),
    (0, Vectors.dense([150.0, 88.0])),
    (1, Vectors.dense([153.0, 87.0])),
    (1, Vectors.dense([155.0, 99.0])),
    (1, Vectors.dense([160.0, 91.0])),
    (0, Vectors.dense([155.0, 89.0])),
    (1, Vectors.dense([160.0, 95.0])),
    (0, Vectors.dense([150.0, 90.0]))
]
df = spark.createDataFrame(dataset, ["label", "features"])
df.show()
```

```
+-----+-----+
|label|  features|
+-----+-----+
|    0| [149.0, 81.0]|
|    0| [150.0, 88.0]|
|    1| [153.0, 87.0]|
|    1| [155.0, 99.0]|
|    1| [160.0, 91.0]|
|    0| [155.0, 89.0]|
|    1| [160.0, 95.0]|
|    0| [150.0, 90.0]|
+-----+-----+
```

```
#进行 pearson 卡方检验
r = ChiSquareTest.test(df, "features", "label")
r.show(truncate=False)
```

pValues	degreesOfFreedom	statistics
[0.1991482734714558, 0.33259390259930843]	[4, 7]	[6.0, 8.0]

在返回的结果中，各值的含义如下：

- (1) **pValues**: 其含义为各个样本特征与样本标签卡方检验的值（卡方值）。越大（越接近 1）代表该特征列越无意义，对标签的区分作用越低；反之，越小（越接近 0）越有区分价值。
- (2) **degreeOfFreedom**: 其含义为各个样本特征的自由度，**degreeOfFreedom+1** 等价于该特征值的种类。
- (3) **statistics**: 可以认为是越大分类价值越高，越小分类价值越低。

4.3 特征处理

接下来的这一节，我们将介绍 MLlib 中与特征处理相关的算法，其大体分为以下三类：

- (1) **特征提取**: 根据原始数据，提取出原本没有的新特征
- (2) **特征选取**: 从大量现有的特征集里，选取出一些有价值的特征
- (3) **特征转化**: 对特征的维度进行变化，或对特征进行修改

4.3.1 CountVectorizer

CountVectorizer 算法是一个用于特征提取的算法，旨在通过计数来将一个文档转换为向量。CountVectorizer 首先会从文档集合中提取出词汇表并生成 CountVectorizerModel，然后再通过 CountVectorizerModel 将文本向量转换成词频向量。同时，它会把频率高的单词排在前面。

```
#先以 DataFrame 的形式构造一个文档集合
df = spark.createDataFrame([
    (0, ['a', 'b', 'c']),
    (1, ['a', 'a', 'a', 'c']),
    (2, ['b', 'a', 'a', 'c', 'b', 'b'])
]).toDF('labels', 'features')
df.show(truncate=False)
```

labels	features
0	[a, b, c]
1	[a, a, a, c]
2	[b, a, a, c, b, b]

```
from pyspark.ml.feature import CountVectorizer

#创建 CountVectorizer 模型
cv = CountVectorizer(inputCol="features", outputCol="vectors")
#训练 CountVectorizerModel 模型
cvmodel = cv.fit(df)
#获取模型训练得到的词汇表
cvmodel.vocabulary
```

```
['a', 'b', 'c']
```

```
#通过 CountVectorizerModel 模型得到词频向量
result = cvmodel.transform(df)
result.show(truncate=False)
```

labels	features	vectors
0	[a, b, c]	(3, [0, 1, 2], [1.0, 1.0, 1.0])
1	[a, a, a, c]	(3, [0, 2], [3.0, 1.0])
2	[b, a, a, c, b, b]	(3, [0, 1, 2], [2.0, 3.0, 1.0])

在这里，词频向量 `vectors` 是一个稀疏向量（`SparseVector`）。其第一个元素表示该文档集合的词汇表中共有 3 个词，而第二个元素与第三个元素是一一对应的，表示词汇表中的第 i 个词在该文档中出现了 n 次。

4.3.2 Word2Vec

Word2Vec 是一种著名的词嵌入（Word Embedding）方法，它可以计算每个单词在其给定语料库环境下的分布式词向量（Distributed Representation，也可直接称为词向量）。如果词的语义相近，它们的词向量在向量空间中也相互接近，因此词向量表示可以在一定程度上刻画每个单词的语义。

```
#先以 DataFrame 的形式构造一个文档集合
df = spark.createDataFrame([
    ("Hi I heard about Spark".split(" "), ),
    ("I wish Java could use case classes".split(" "), ),
    ("Logistic regression models are neat".split(" "), )
], ["text"])
df.show(truncate=False)
```

```
+-----+
|text|
+-----+
|[Hi, I, heard, about, Spark]|
|[I, wish, Java, could, use, case, classes]|
|[Logistic, regression, models, are, neat]|
+-----+
```

```
from pyspark.ml.feature import Word2Vec

#创建 Word2Vec 模型
#vectorSize 为特征向量的维度
#minCount 为单词在 word2vec 模型的词汇表中出现的最少次数
w2v = Word2Vec(vectorSize=3, minCount=0, inputCol="text", outputCol="result")
#训练 Word2Vec 模型
model = w2v.fit(df)
#通过 Word2Vec 模型将文档向量化
result = model.transform(documentDF)
result.show(truncate=False)
```

```
+-----+-----+
|text|result|
+-----+-----+
|[Hi, I, heard, about, Spark]| [0.0044096939265727995, -0.05489438772201538, -0.057969415187835695]|
|[I, wish, Java, could, use, case, classes]| [0.026761331728526523, 0.04510576943201678, 0.055212869708027155]|
|[Logistic, regression, models, are, neat]| [-0.02019107732921839, -0.02984520643949509, 0.035833410127088426]|
+-----+-----+
```

可以看到，所有文档都被转变为了一个 3 维的特征向量，这些特征向量可以被应用到相关的机器学习方法中。

4.3.3 TF-IDF

TF-IDF（词频—逆向文件频率）是一种在文本挖掘中广泛使用的特征向量化方法，它可以体现一个文档中词语在语料库中的重要程度。

TF（term frequency）指词频，其是一词语 ω 出现的次数除以该文件的总词语数。

$$TF = \frac{\text{词语}\omega\text{在该文档中出现的次数}}{\text{该文档的总词语数}}$$

IDF (inverse document frequency) 指逆向文件频率，它的主要思想是如果包含词语 ω 的文档越少，IDF 越大，则说明词语具有很好的类别区分能力。某一特定词语的 IDF，可以由总文档数目除以包含该词语的文档数加一，再将得到的商取对数得到（这里之所以要+1，是为了避免分母为 0 这种情况）。

$$IDF = \lg \left(\frac{\text{总文档数目}}{\text{包含词语}\omega\text{的文档数} + 1} \right)$$

TF-IDF 就是 TF 与 IDF 的乘积。

$$TF - IDF = TF \times IDF$$

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的 TF-IDF。因此，TF-IDF 倾向于过滤掉常见的词语，保留重要的词语。

```
#先以 DataFrame 的形式构造一个文档集合
sentenceData = spark.createDataFrame([
    (0, "I heard about Spark and I love Spark"),
    (0, "I wish Java could use case classes"),
    (1, "Logistic regression models are neat")
]).toDF("label", "sentence")
sentenceData.show(truncate=False)
```

```
+-----+
|label|sentence|
+-----+
|0    |I heard about Spark and I love Spark|
|0    |I wish Java could use case classes |
|1    |Logistic regression models are neat |
+-----+
```

```
from pyspark.ml.feature import Tokenizer

#在得到文档集合后，用 tokenizer 对句子进行分词
tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
wordsData = tokenizer.transform(sentenceData)
wordsData.show(truncate=False)
```

label	words
0	[i, heard, about, spark, and, i, love, spark]
0	[i, wish, java, could, use, case, classes]
1	[logistic, regression, models, are, neat]

```
from pyspark.ml.feature import HashingTF
```

#得到分词后的文档序列，即可使用 HashingTF 的 transform()方法把句子哈希成特征向量，这里设置最大特征维数 numFeatures 为 20

#或者，也可以使用上面 4.3.2 提到的 CountVectorizer 获得词频向量

```
hashingTF = HashingTF(inputCol="words", outputCol="rawFeatures", numFeatures=20)
```

```
featurizedData = hashingTF.transform(wordsData)
```

```
featurizedData.show(truncate=False)
```

label	rawFeatures
0	(20, [0, 5, 9, 13, 17], [1.0, 2.0, 2.0, 1.0, 2.0])
0	(20, [2, 7, 9, 13, 15], [1.0, 1.0, 3.0, 1.0, 1.0])
1	(20, [4, 6, 13, 15, 18], [1.0, 1.0, 1.0, 1.0, 1.0])

```
from pyspark.ml.feature import IDF
```

#创建 IDF 模型

```
idf = IDF(inputCol='rawFeatures', outputCol='features')
```

#训练 IDF 模型，得到 IDFModel 模型

```
idfModel = idf.fit(featurizedData)
```

#通过 IDFModel 模型得到文档中每一个单词对应的 TF-IDF 度量值

```
rescaledData = idfModel.transform(featurizedData)
```

```
rescaledData.select("features").show(truncate=False)
```

label	features
0	(20, [0, 5, 9, 13, 17], [0.6931471805599453, 1.3862943611198906, 0.5753641449035617, 0.0, 1.3862943611198906])
0	(20, [2, 7, 9, 13, 15], [0.6931471805599453, 0.6931471805599453, 0.8630462173553426, 0.0, 0.28768207245178085])
1	(20, [4, 6, 13, 15, 18], [0.6931471805599453, 0.6931471805599453, 0.0, 0.28768207245178085, 0.6931471805599453])

4.3.4 ChiSqSelector

ChiSqSelector（卡方选择器）是基于 Chi-Squared feature selection（卡方特征选择）的一个特征选取模型。它可以在特征向量中选择出那些“优秀”的特征，组成新的、更“精简”的特征向量的过程。它在高维数据分析中十分常用，可以剔除掉“冗余”和“无关”的特征，提升学习器的性能。

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import ChiSqSelector

#构造一个含有 4 个特征的数据集
df = spark.createDataFrame([
    (0, Vectors.dense([0.0, 0.0, 18.0, 1.0]), 1.0),
    (1, Vectors.dense([0.0, 1.0, 12.0, 0.0]), 0.0),
    (2, Vectors.dense([1.0, 0.0, 15.0, 0.1]), 0.0),
], ["id", "features", "label"])
df.show()
```

id	features	label
0	[0.0, 0.0, 18.0, 1.0]	1.0
1	[0.0, 1.0, 12.0, 0.0]	0.0
2	[1.0, 0.0, 15.0, 0.1]	0.0

```
#创建卡方选择器模型（numTopFeatures 参数设置要提取的特征个数）
selector1 = ChiSqSelector(
    numTopFeatures=1,
    featuresCol="features",
    outputCol="selectedFeatures",
    labelCol="label")

#训练模型，并通过得到的模型提取出和标签关联性最强的一个特征
result = selector1.fit(df).transform(df)
result.show()
```

id	features	label	selectedFeatures
0	[0.0, 0.0, 18.0, 1.0]	1.0	[18.0]
1	[0.0, 1.0, 12.0, 0.0]	0.0	[12.0]
2	[1.0, 0.0, 15.0, 0.1]	0.0	[15.0]

#创建卡方选择器模型（numTopFeatures 参数设置要提取的特征个数）

```
selector2 = ChiSqSelector(
    numTopFeatures=2,
    featuresCol="features",
    outputCol="selectedFeatures",
    labelCol="label")
```

#训练模型，并通过得到的模型提取出和标签关联性最强的两个特征

```
result = selector2.fit(df).transform(df)
result.show()
```

id	features	label	selectedFeatures
0	[0.0, 0.0, 18.0, 1.0]	1.0	[18.0, 1.0]
1	[0.0, 1.0, 12.0, 0.0]	0.0	[12.0, 0.0]
2	[1.0, 0.0, 15.0, 0.1]	0.0	[15.0, 0.1]

4.3.5 PCA

PCA 训练模型是基于 PCA（principal components analysis，主成分分析）的一个特征转化模型，它可以将高维向量投影到低维空间中，主要运用于特征的降维处理。下面的示例显示了如何将 5 维特征向量投影到 3 维主成分中。

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import PCA

#创建一个含有 5 维特征的数据集
data = [(Vectors.dense([5.0, 1.0, 7.0, 7.0, 7.0])),
        (Vectors.dense([2.0, 0.0, 3.0, 4.0, 5.0])),
        (Vectors.dense([4.0, 0.0, 0.0, 6.0, 7.0]))]
df = spark.createDataFrame(data, ["features"])
df.show(truncate=False)
```

```

+-----+
| features |
+-----+
| [5.0, 1.0, 7.0, 7.0, 7.0] |
| [2.0, 0.0, 3.0, 4.0, 5.0] |
| [4.0, 0.0, 0.0, 6.0, 7.0] |
+-----+

```

```

#创建 PCA 模型（k=3 设置主成分数量，即将特征向量降到 k 维）
pca = PCA(k=3, inputCol="features", outputCol="pcaFeatures")
#训练模型，并通过得到的模型将特征向量从 5 维降到 3 维
result = pca.fit(df).transform(df)
result.show(truncate=False)

```

```

+-----+-----+
| features | pcaFeatures |
+-----+-----+
| [5.0, 1.0, 7.0, 7.0, 7.0] | [-10.204757082790582, 7.728739778252358, 1.858739491810498] |
| [2.0, 0.0, 3.0, 4.0, 5.0] | [-4.7076290412191915, 4.765366144272079, 1.8587394918105025] |
| [4.0, 0.0, 0.0, 6.0, 7.0] | [-3.1182608087717996, 9.06349464650268, 1.8587394918104994] |
+-----+-----+

```

4.4 分类与回归

4.4.1 逻辑斯蒂回归分类器

逻辑斯蒂回归（logistic regression）是统计学习中的经典分类方法，属于对数线性模型。逻辑斯蒂回归的因变量可以是二分类的，也可以是多分类的。MLlib 中的 LogisticRegression 模块对逻辑斯蒂回归的二分类和多分类问题都有实现，在这里，我们通过一个二分类问题对 LogisticRegression 模块的使用进行说明。

```

#利用 spark 提供的样例数据(data/mllib/sample_libsvm_data.txt)创建数据集
df = spark.read.format("libsvm").load("sample_libsvm_data.txt")
#选取数据集的 70%作为模型的训练集，30%作为模型的测试集
training,testing = df.randomSplit([0.7,0.3])

```

```

training.show()

```

label	features
0.0	(692, [95, 96, 97, 12...
0.0	(692, [98, 99, 100, 1...
0.0	(692, [100, 101, 102...
0.0	(692, [122, 123, 124...
0.0	(692, [123, 124, 125...
0.0	(692, [124, 125, 126...
0.0	(692, [124, 125, 126...
0.0	(692, [124, 125, 126...
0.0	(692, [124, 125, 126...
0.0	(692, [125, 126, 127...
0.0	(692, [126, 127, 128...
0.0	(692, [126, 127, 128...
0.0	(692, [126, 127, 128...
0.0	(692, [126, 127, 128...
0.0	(692, [126, 127, 128...
0.0	(692, [126, 127, 128...
0.0	(692, [126, 127, 128...
0.0	(692, [127, 128, 129...
0.0	(692, [127, 128, 129...
0.0	(692, [150, 151, 152...

only showing top 20 rows

testing.show()

label	features
0.0	(692, [121, 122, 123...
0.0	(692, [122, 123, 148...
0.0	(692, [123, 124, 125...
0.0	(692, [123, 124, 125...
0.0	(692, [124, 125, 126...
0.0	(692, [124, 125, 126...
0.0	(692, [126, 127, 128...
0.0	(692, [127, 128, 129...
0.0	(692, [128, 129, 130...
0.0	(692, [129, 130, 131...
0.0	(692, [152, 153, 154...
0.0	(692, [153, 154, 155...
1.0	(692, [123, 124, 125...
1.0	(692, [123, 124, 125...
1.0	(692, [124, 125, 126...
1.0	(692, [124, 125, 126...
1.0	(692, [125, 126, 127...
1.0	(692, [125, 126, 153...
1.0	(692, [126, 127, 128...
1.0	(692, [126, 127, 128...

only showing top 20 rows

```

from pyspark.ml.classification import LogisticRegression

#创建逻辑斯蒂回归模型
lr = LogisticRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
#训练逻辑斯蒂回归模型
lrModel = lr.fit(training)
#使用得到的逻辑斯蒂回归模型对测试集的数据进行分类
Predictions=lrModel.transform(testing)
Predictions.show()

```

label	features	rawPrediction	probability	prediction
0.0	(692, [121, 122, 123...	[0.49204188896929...	[0.62058733109898...	0.0
0.0	(692, [122, 123, 148...	[0.44378780556689...	[0.60916121795649...	0.0
0.0	(692, [123, 124, 125...	[1.02391510390286...	[0.73573451235695...	0.0
0.0	(692, [123, 124, 125...	[0.32450276517164...	[0.58042121686027...	0.0
0.0	(692, [124, 125, 126...	[0.88269416076316...	[0.70738020614250...	0.0
0.0	(692, [124, 125, 126...	[0.49594434149390...	[0.62150576411237...	0.0
0.0	(692, [126, 127, 128...	[1.00631080237218...	[0.73229754695247...	0.0
0.0	(692, [127, 128, 129...	[0.92265185288749...	[0.71558213073428...	0.0
0.0	(692, [128, 129, 130...	[0.86514893639971...	[0.70373529097957...	0.0
0.0	(692, [129, 130, 131...	[0.79056557839003...	[0.68795275832754...	0.0
0.0	(692, [152, 153, 154...	[0.15186287144325...	[0.53789292091132...	0.0
0.0	(692, [153, 154, 155...	[0.05812685942535...	[0.51452762467438...	0.0
1.0	(692, [123, 124, 125...	[-1.2695403452666...	[0.21933594755213...	1.0
1.0	(692, [123, 124, 125...	[-1.1051996575813...	[0.24876690350317...	1.0
1.0	(692, [124, 125, 126...	[-1.3126672501999...	[0.21204085892597...	1.0
1.0	(692, [124, 125, 126...	[-1.2183553704405...	[0.22822600432055...	1.0
1.0	(692, [125, 126, 127...	[-1.1603660543588...	[0.23860077753514...	1.0
1.0	(692, [125, 126, 153...	[-1.2150577460704...	[0.22880736472746...	1.0
1.0	(692, [126, 127, 128...	[-1.3228322953935...	[0.21034745928270...	1.0
1.0	(692, [126, 127, 128...	[-1.2381464700288...	[0.22475878423261...	1.0

only showing top 20 rows

从中我们可以看到，最左边的 label 一列是测试集真实的类别，最右边的 prediction 一列是模型分类的结果，它们几乎都是一样的，分类的结果还是比较好的。

4.4.2 线性回归

```

#利用 spark 提供的样例数据(data/mllib/sample_linear_regression_data.txt)创建训练集
training = spark.read.format("libsvm").load("sample_linear_regression_data.txt")
training.show()

```

label	features
-9.490009878824548	(10, [0, 1, 2, 3, 4, 5, ...
0.2577820163584905	(10, [0, 1, 2, 3, 4, 5, ...
-4.438869807456516	(10, [0, 1, 2, 3, 4, 5, ...
-19.782762789614537	(10, [0, 1, 2, 3, 4, 5, ...
-7.966593841555266	(10, [0, 1, 2, 3, 4, 5, ...
-7.896274316726144	(10, [0, 1, 2, 3, 4, 5, ...
-8.464803554195287	(10, [0, 1, 2, 3, 4, 5, ...
2.1214592666251364	(10, [0, 1, 2, 3, 4, 5, ...
1.0720117616524107	(10, [0, 1, 2, 3, 4, 5, ...
-13.772441561702871	(10, [0, 1, 2, 3, 4, 5, ...
-5.082010756207233	(10, [0, 1, 2, 3, 4, 5, ...
7.887786536531237	(10, [0, 1, 2, 3, 4, 5, ...
14.323146365332388	(10, [0, 1, 2, 3, 4, 5, ...
-20.057482615789212	(10, [0, 1, 2, 3, 4, 5, ...
-0.8995693247765151	(10, [0, 1, 2, 3, 4, 5, ...
-19.16829262296376	(10, [0, 1, 2, 3, 4, 5, ...
5.601801561245534	(10, [0, 1, 2, 3, 4, 5, ...
-3.2256352187273354	(10, [0, 1, 2, 3, 4, 5, ...
1.5299675726687754	(10, [0, 1, 2, 3, 4, 5, ...
-0.250102447941961	(10, [0, 1, 2, 3, 4, 5, ...

only showing top 20 rows

```
from pyspark.ml.regression import LinearRegression
```

```
#创建线性回归模型
```

```
lr = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)
```

```
#训练线性回归模型
```

```
lrModel = lr.fit(training)
```

```
#输出模型的系数和截距
```

```
print("Coefficients: %s" % str(lrModel.coefficients))
```

```
print("Intercept: %s" % str(lrModel.intercept))
```

```
Coefficients: [0.0, 0.32292516677405936, -0.3438548034562218, 1.9156017023458414, 0.05288058680386263, 0.765962720459771, 0.0, -0.1510539266918668
2, -0.21587930360904642, 0.22025369188813426]
Intercept: 0.1598936844239736
```

```
#总结训练集上的模型并打印出一些指标
```

```
trainingSummary = lrModel.summary
```

```
print("numIterations: %d" % trainingSummary.totalIterations)
```

```
print("objectiveHistory: %s" % str(trainingSummary.objectiveHistory))
```

```
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
```

```
print("r2: %f" % trainingSummary.r2)
```

```
trainingSummary.residuals.show()
```

```

numIterations: 7
objectiveHistory: [0.49999999999999994, 0.4967620357443381, 0.4936361664340463, 0.4936351537897608, 0.4936351214177871, 0.49363512062528014,
0.4936351206216114]
RMSE: 10.189077
r2: 0.022861
+-----+
| residuals |
+-----+
|-9.889232683103197|
| 0.5533794340053554|
|-5.204019455758823|
|-20.566686715507508|
|-9.4497405180564|
|-6.909112502719486|
|-10.00431602969873|
| 2.062397807050484|
| 3.1117508432954772|
|-15.893608229419382|
|-5.036284254673026|
| 6.483215876994333|
| 12.429497299109002|
|-20.32003219007654|
|-2.0049838218725005|
|-17.867901734183793|
| 7.646455887420495|
|-2.2653482182417406|
|-0.10308920436195645|
|-1.380034070385301|
+-----+
only showing top 20 rows

```

4.5 聚类

4.5.1 K-means

K-means 是一个迭代求解的聚类算法，它首先会创建 K 个划分，然后迭代地将样本从一个划分转移到另一个划分来改善最终聚类的质量。MLlib 中的 `KMeans` 方法对其进行了实现，下面我们通过一个例子来对其用法进行说明。

```

#利用 spark 提供的样例数据(data/mllib/sample_kmeans_data.txt)创建数据集
dataset = spark.read.format("libsvm").load("sample_kmeans_data.txt")
dataset.show(truncate=False)

```

```

+-----+-----+
| label | features |
+-----+-----+
| 0.0   | (3, [], []) |
| 1.0   | (3, [0, 1, 2], [0.1, 0.1, 0.1]) |
| 2.0   | (3, [0, 1, 2], [0.2, 0.2, 0.2]) |
| 3.0   | (3, [0, 1, 2], [9.0, 9.0, 9.0]) |
| 4.0   | (3, [0, 1, 2], [9.1, 9.1, 9.1]) |
| 5.0   | (3, [0, 1, 2], [9.2, 9.2, 9.2]) |
+-----+-----+

```

```

from pyspark.ml.clustering import KMeans

#创建 KMeans 模型，setK 设置簇的个数为 2，setSeed 设置随机数种子为 1
kmeans = KMeans().setK(2).setSeed(1)
#训练 KMeans 模型
model = kmeans.fit(dataset)

#利用该模型将数据集聚成两个簇
result = model.transform(dataset)
result.show(truncate=False)

#显示这两个簇的中心
centers = model.clusterCenters()
for center in centers:

```

label	features	prediction
0.0	(3, [], [])	0
1.0	(3, [0, 1, 2], [0.1, 0.1, 0.1])	0
2.0	(3, [0, 1, 2], [0.2, 0.2, 0.2])	0
3.0	(3, [0, 1, 2], [9.0, 9.0, 9.0])	1
4.0	(3, [0, 1, 2], [9.1, 9.1, 9.1])	1
5.0	(3, [0, 1, 2], [9.2, 9.2, 9.2])	1

```

[0.1 0.1 0.1]
[9.1 9.1 9.1]

```

4.5.2 GMM

GMM（Gaussian Mixture Model，高斯混合模型）是一种概率式的聚类方法，也被称为软聚类。MLlib 中的 GaussianMixture 方法对 GMM 进行了实现，它的用法与 MLlib 中的 KMeans 方法完全一样。

```

#利用 spark 提供的样例数据(data/mllib/sample_kmeans_data.txt)创建数据集
dataset = spark.read.format("libsvm").load("sample_kmeans_data.txt")
dataset.show(truncate=False)

```


label	features
0.0	(3, [], [])
1.0	(3, [0, 1, 2], [0.1, 0.1, 0.1])
2.0	(3, [0, 1, 2], [0.2, 0.2, 0.2])
3.0	(3, [0, 1, 2], [9.0, 9.0, 9.0])
4.0	(3, [0, 1, 2], [9.1, 9.1, 9.1])
5.0	(3, [0, 1, 2], [9.2, 9.2, 9.2])

```

from pyspark.ml.clustering import GaussianMixture

#创建 GMM 模型，setK 设置簇的个数为 2，setSeed 设置随机数种子为 538009335
gmm = GaussianMixture().setK(2).setSeed(538009335)
#训练 GMM 模型
model = gmm.fit(dataset)

#利用该模型将数据集聚成两个簇
result = model.transform(dataset)
result.show(truncate=False)

```

label	features	prediction	probability
0.0	(3, [], [])	0	[0.9999999999999979, 2.093996169658831E-15]
1.0	(3, [0, 1, 2], [0.1, 0.1, 0.1])	0	[0.9999999999999999, 9.891337521299578E-16]
2.0	(3, [0, 1, 2], [0.2, 0.2, 0.2])	0	[0.9999999999999979, 2.093996169657856E-15]
3.0	(3, [0, 1, 2], [9.0, 9.0, 9.0])	1	[2.0939961696573796E-15, 0.9999999999999979]
4.0	(3, [0, 1, 2], [9.1, 9.1, 9.1])	1	[9.89133752130383E-16, 0.9999999999999999]
5.0	(3, [0, 1, 2], [9.2, 9.2, 9.2])	1	[2.0939961696583838E-15, 0.9999999999999979]

和 K-means 等硬聚类方法不同的是，我们在它的返回结果中，不仅可以得到每个样本的簇归属，还可以得到每个样本属于各个簇的概率，这个概率在结果的“probability”一列中。

第五章 教学实验案例

5.1 教学实验要求及内容

一、适用专业及年级

高等学校计算机及相关专业的本科高年级学生和研究生。

二、课程目标与基本要求

通过本课程的学习，帮助学生了解大数据技术的基本概念、存储、处理、分析和应用；掌握 Hadoop HDFS、Hadoop MapReduce，以及 Spark 的框架与应用；熟悉 Spark MLlib 机器学习库的使用；拥有运用大数据技术对现实中的一些问题进行分析和求解的能力。

通过在 DataOne 大数据平台上完成若干典型任务的分布式编程分析，掌握 Hadoop、Spark 分布式大数据框架程序的设计、实现、调试和优化。本课程全面系统地介绍现代大数据技术编程、开发方法、语言、环境和工具等，从大量现实案例入手，渐进地展开大数据技术的各个技术层面。

三、主要仪器设备

DataOne 大数据实验平台。

四、实验项目及教学安排

五、考核方式及成绩评定

考核方式：

(1)平时成绩：

- 实验考勤：每次考勤：出勤（2分）；请假、迟到、早退（1分）；旷课（0分）。
- 预习报告：要求写明实验目的、主要实验设备名称、实验原理和内容：优秀（4分）；良好（3分）；中等（2分）；及格（1分）；不及格（0分）。
- 实验报告：要求写明实验设备名称和型号、实验步骤、实验分析及注意事项：优秀（4分）、良好（3分）、中等（2分）、及格（1分）和不及格（0分）记分。

(2)考试成绩：上机考试，满分为 100 分。

(3)成绩评定：总评成绩=平时成绩×50%+考试成绩×50%。

5.2 实验报告格式

深 圳 大 学 实 验 报 告

课程名称:

实验名称:

姓 名:

学 号:

班 级:

实验日期:

一、实验目的:

二、实验环境:

三、实验内容:

四、程序代码及说明:

根据实验内容，写出程序的完整源代码，并对源代码的关键部分，进行必要的说明。

五、实验结果及分析：

根据实验内容，记录具体的实验数据或程序运行结果，阐述对程序性能改进的具体方法及主要调试过程。

六、实验结论：

5.3 教学实验案例

5.3.1 实验一 原材料采购策略制定

1、实验目的

- (1) 掌握 Spark 集群的使用；
- (2) 掌握 Spark 数据统计的基本方法；
- (3) 了解 Spark MLlib 的基本用法；
- (4) 掌握 Spark MLlib 求解线性回归模型的方法。

2、实验环境

使用 DataOne 大数据实验平台与 Python 语言，在 spark 集群上编写、运行和调试 spark 程序。

3、实验背景

Spark 是基于内存计算的大数据并行计算框架，可用于构建大型的、低延迟的数据分析应用程序。其具有运行速度快、容易使用、运行模式多样等多种特点。

在企业的运营过程中，库存管理是十分重要的一个环节，而原材料采购策略的制定又是实现有效库存管理的关键一步。如果采购量太少，则无法提供充足的原材料来进行产品的生产，从而损失了大量盈利的机会；如果采购量太大，则会造成库存的堆积，提升了库存的成本，并且过剩的原材料也可能会腐败变质。因此，如果能尽可能准确地预测未来一段时间各种产品的销量，并以此作为根据来进行原材料采购策略的制定，就能实现有效的库存管理。

在本次实验中，我们要对一个食品企业的各项数据进行挖掘，尽可能准确地预测未来一段时间各种产品的销量，并以此作为根据进行原材料采购策略的制定。

4、实验内容

- (1) 数据表的选取与 HDFS 中文件的上传
- (2) 通过使用 dataframe 对数据进行统计，统计出每种产品每月的销售额

(3) 通过作图（时间-销量），得出每种产品销量的大体趋势

(4) 通过 LinearRegression 分别对每种产品每月的销售额进行线性回归模型的训练，分析训练得到的模型的准确度，并用该模型来预测未来半年内产品的销售额

(5) 根据预测的结果，给出未来半年合理的原材料采购方案

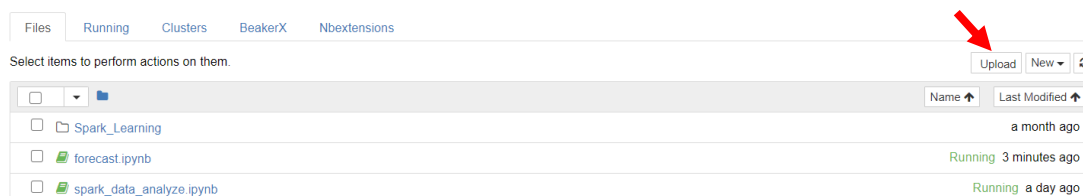
5、参考代码

(1) 数据表的选取与 HDFS 中文件的上传

经过观察与分析，我们得知所要用到的表有：“销售定制单”、“配方成本表-面包”、“配方成本表-烘焙”、“配方成本表-蛋糕”。

通过“销售定制单”，我们可以得到在某一个时间点，卖出了几件产品。通过“配方成本表”我们可以获得每一种产品生产所需要的原材料种类及数量。

首先，我们要在主页面点击“Upload”，将这 4 个 csv 文件上传到 jupyter notebook 上。



上传完成后，点击“New”，新建一个控制台。然后在控制台窗口中输入以下命令，将 jupyter notebook 本地上的文件上传到 Hadoop 分布式文件系统上（这里的-f是可选项，如果加了-f，上传时遇到同名文件会进行覆盖；如果没有加-f，上传时遇到同名文件会跳过）。

```
hdfs dfs -put -f /home/xuyuming/. /user/xuyuming/.
```

(2) 通过使用 dataframe 对数据进行统计，统计出每种产品每月的销售额

```

from pyspark.sql import SparkSession
from pyspark.sql.functions import unix_timestamp,date_format

#创建 sparksession
spark=SparkSession.builder.getOrCreate()

#读取 csv 文件，并生成 dataframe
df0=spark.read.format('com.databricks.spark.csv').options(header='true',
inferschema='true').load('/user/xuyuming/xuyuming/销售订制单.csv')

#将 string 类型的日期转换为 timestamp 类型的日期
df0=df0.withColumn("下单时间", df0.下单时间.cast('timestamp'))

#选择出需要用到列
df0=df0.select('商品名称','数量','下单时间')
#提取出年-月，并将其新建为 dataframe 的一列，命名为‘时间’
df0=df0.select('*',date_format('下单时间','yyyy-MM').alias('时间'))

```

调用 df0.show(), 我们可以观察到此时 df0 的结构如下:

商品名称	数量	下单时间	时间
提香	1	2016-01-01 07:58:31	2016-01
芒香	1	2016-01-01 08:33:09	2016-01
芒香	1	2016-01-01 08:20:09	2016-01
提香	1	2016-01-01 07:28:38	2016-01
黄金彼岸	1	2016-01-01 07:09:13	2016-01
提子面包	1	2016-01-01 08:22:44	2016-01
米露提子吐司	1	2016-01-01 07:39:21	2016-01
提子面包	1	2016-01-01 08:39:52	2016-01
温泉吐司	1	2016-01-01 07:18:25	2016-01
冠军芒果	1	2016-01-01 07:50:47	2016-01
干酪火腿	1	2016-01-01 07:07:28	2016-01
提子面包	1	2016-01-01 07:52:58	2016-01
传统长棍	1	2016-01-01 08:01:49	2016-01
米露提子吐司	1	2016-01-01 08:55:07	2016-01
米露提子吐司	1	2016-01-01 07:50:43	2016-01
法式起司	1	2016-01-01 08:43:45	2016-01
黑糖桂圆欧包	1	2016-01-01 08:34:34	2016-01
米露提子吐司	1	2016-01-01 07:28:46	2016-01
温泉吐司	1	2016-01-01 07:27:57	2016-01
传统长棍	1	2016-01-01 08:14:36	2016-01

```

#按月份分别统计每种商品的销量
df0=df0.groupBy([df0['商品名称'],df0['时间']]).sum('数量').orderBy(df0['商品名称'].asc(),df0['时间'].asc())

```

调用 df0.collect(), 我们可以观察到此时 df0 中的数据如下:

```

[Row(商品名称='DIY蛋糕亲子烘焙', 时间='2016-01', sum(数量)=25),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2016-05', sum(数量)=5),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2016-06', sum(数量)=5),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2016-07', sum(数量)=2),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2016-10', sum(数量)=2),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2016-11', sum(数量)=31),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2016-12', sum(数量)=33),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2017-01', sum(数量)=39),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2017-02', sum(数量)=19),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2017-03', sum(数量)=3),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2017-04', sum(数量)=4),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2017-05', sum(数量)=13),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2017-06', sum(数量)=5),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2017-07', sum(数量)=7),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2017-08', sum(数量)=9),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2017-09', sum(数量)=8),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2017-10', sum(数量)=57),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2017-11', sum(数量)=51),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2017-12', sum(数量)=52),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2018-01', sum(数量)=78),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2018-02', sum(数量)=27),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2018-04', sum(数量)=16),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2018-06', sum(数量)=22),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2018-08', sum(数量)=5),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2018-09', sum(数量)=41),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2018-10', sum(数量)=60),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2018-11', sum(数量)=160),
Row(商品名称='DIY蛋糕亲子烘焙', 时间='2018-12', sum(数量)=59),
Row(商品名称='DIY饼干亲子烘焙', 时间='2016-01', sum(数量)=59),
Row(商品名称='DIY饼干亲子烘焙', 时间='2016-02', sum(数量)=6),
Row(商品名称='DIY饼干亲子烘焙', 时间='2016-03', sum(数量)=10),
Row(商品名称='DIY饼干亲子烘焙', 时间='2016-04', sum(数量)=1),
Row(商品名称='DIY饼干亲子烘焙', 时间='2016-05', sum(数量)=33),
Row(商品名称='DIY饼干亲子烘焙', 时间='2016-06', sum(数量)=32),
Row(商品名称='DIY饼干亲子烘焙', 时间='2016-07', sum(数量)=32),
Row(商品名称='DIY饼干亲子烘焙', 时间='2016-09', sum(数量)=23),
Row(商品名称='DIY饼干亲子烘焙', 时间='2016-10', sum(数量)=37),
Row(商品名称='DIY饼干亲子烘焙', 时间='2016-11', sum(数量)=111),
Row(商品名称='DIY饼干亲子烘焙', 时间='2016-12', sum(数量)=129),
Row(商品名称='DIY饼干亲子烘焙', 时间='2017-01', sum(数量)=131),
Row(商品名称='DIY饼干亲子烘焙', 时间='2017-02', sum(数量)=75),
Row(商品名称='DIY饼干亲子烘焙', 时间='2017-03', sum(数量)=17),
Row(商品名称='DIY饼干亲子烘焙', 时间='2017-04', sum(数量)=16),
Row(商品名称='DIY饼干亲子烘焙', 时间='2017-05', sum(数量)=25),
Row(商品名称='DIY饼干亲子烘焙', 时间='2017-06', sum(数量)=35),
Row(商品名称='DIY饼干亲子烘焙', 时间='2017-07', sum(数量)=24),
Row(商品名称='DIY饼干亲子烘焙', 时间='2017-08', sum(数量)=43),

```

(3) 通过作图（时间-销量），得出每种产品销量的大体趋势

```

import matplotlib.pyplot as plt

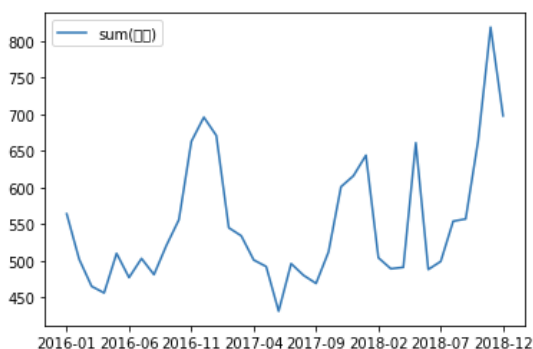
pddf0=df0.toPandas()

good_names=['芒香',
            '芒果蛋糕',
            '秘密花园',
            '暗香',
            '提香',
            '黄金彼岸',
            '榴莲千层',
            '榴莲双拼',
            '心语心愿',
            '红妆',
            '牛轧糖原料',
            '椰蓉粉',
            '南瓜原料套装',
            'DIY 饼干亲子烘焙',
            'DIY 蛋糕亲子烘焙',
            '亲子手工烘焙入门',
            '安佳黄油',
            'kiri 奶油奶酪',
            '蛋挞套装',
            '蛋黄酥原料套餐',
            '提子面包',
            '传统长棍',
            '法式起司',
            '米露提子吐司',
            '温泉吐司',
            '黑糖桂圆欧包',
            '冠军芒果',
            '干酪火腿',
            '布里欧',
            '芥末培根']

#作图（时间-销量）
for good_name in good_names:
    print(good_name)
    pddf0[pddf0.商品名称==good_name].plot(x='时间',y='sum(数量)',kind='line')
    plt.rcParams['font.sans-serif']=['SimHei']
    plt.show()

```


得到的时间-销量曲线如下：



(产品“芒香”的时间-销量曲线)

(4) 通过 **LinearRegression** 分别对每种产品每月的销售额进行线性回归模型的训练，分析训练得到的模型的准确度，并用该模型来预测未来半年内产品的销售额

```
#构造向量 (goods_list 嵌套列表)，用于预测销量。
goods_list=[]

for good_name in good_names:
    g1=pddf0[pddf0.商品名称==good_name]
    gg1=g1['sum(数量)'].tolist()
    goods_list.append(gg1)

#创建 libsvm 格式的文本文档，用来后面训练模型
for i in range(30):
    string = ""
    k = 1
    for num in goods_list[i]:
        string = string + str(num) + ' 1:' + str(k) + '\n'
        k = k + 1
    filename = '/home/xuyuming/销量' + str(i) + '.txt'
    file=open(filename,'w')
    file.write(string)
    file.close()
```

这段代码运行后，在 jupyter notebook 的文件页面将会产生用于模型训练的 30 个 libsvm 格式的文本文档（销量 0.txt ~ 销量 29.txt）。

<input type="checkbox"/>	销量0.txt	40 minutes ago
<input type="checkbox"/>	销量1.txt	40 minutes ago
<input type="checkbox"/>	销量10.txt	40 minutes ago
<input type="checkbox"/>	销量11.txt	40 minutes ago
<input type="checkbox"/>	销量12.txt	40 minutes ago
<input type="checkbox"/>	销量13.txt	40 minutes ago
<input type="checkbox"/>	销量14.txt	40 minutes ago
<input type="checkbox"/>	销量15.txt	40 minutes ago
<input type="checkbox"/>	销量16.txt	40 minutes ago
<input type="checkbox"/>	销量17.txt	40 minutes ago

销量0.txt ✓ 40 minutes ago

	File	Edit	View	Language
1	564	1:1		
2	502	1:2		
3	465	1:3		
4	456	1:4		
5	510	1:5		
6	477	1:6		
7	503	1:7		
8	481	1:8		
9	521	1:9		
10	556	1:10		
11	663	1:11		
12	696	1:12		
13	671	1:13		

因为这 30 个新创建的文本文件都是存储在本地的，所以我们在此要用前面的方法，将这些文件先上传到 Hadoop 分布式文件系统中，才能继续进行后面的工作。

```
hdfs dfs -put -f /home/xuyuming/. /user/xuyuming/.
```

```

from pyspark.ml.regression import LinearRegression

#存储系数 k
coeff = []
#存储截距 b
inter = []

#线性回归
for index in range(30):
    print('=====')
    print(good_names[index])
    # 加载训练数据
    string = '/user/xuyuming/xuyuming/销量' + str(index) + '.txt'
    training = spark.read.format("libsvm").load(string)

    lr = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

    # 训练模型
    lrModel = lr.fit(training)

    # 输出并保存得到的结果
    print("Coefficients: %s" % str(lrModel.coefficients))
    print("Intercept: %s" % str(lrModel.intercept))
    coeff.append(lrModel.coefficients)
    inter.append(lrModel.intercept)

    # 输出模型的准确度等信息
    trainingSummary = lrModel.summary
    print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
    print("r2: %f" % trainingSummary.r2)

```

这段代码运行后的输出如下：

```

=====
芒香
Coefficients: [3.0977853661583]
Intercept: 492.96874850384927
RMSE: 80.173750
r2: 0.140700
=====
芒果蛋糕
Coefficients: [4.458629545129518]
Intercept: 113.70979785954836
RMSE: 65.199318
r2: 0.338035
=====
秘密花园
Coefficients: [4.518793631786534]
Intercept: 127.70787336750466
RMSE: 64.090462
r2: 0.251012
=====

```

```

#用于存储预测得到的每种产品未来 6 个月的销量
sales_forecast=[]
sub_sales_forecast=[]

for i in range(30):
    print(good_names[i]+'： ')
    sub_sales_forecast=[]
    if inter[i]>0:
        print('sales = '+str(coeff[i][0])+'*t'+str(inter[i]))
    elif inter[i]<0:
        print('sales = '+str(coeff[i][0])+'*t'+str(inter[i]))
    else:
        print('sales = '+str(coeff[i][0])+'*t')
    print('未来 6 个月销量预测： ')
    for k in range(6):
        num = coeff[i][0]*(len(goods_list)+k)+inter[i]
        sub_sales_forecast.append(num)
        print(num)
    sales_forecast.append(sub_sales_forecast)
    print('=====')

```

这段代码运行后的输出如下：

```

芒香：
sales = 3.0977853661583*t+492.96874850384927
未来6个月销量预测：
585.9023094885982
589.0000948547565
592.0978802209149
595.1956655870732
598.2934509532315
601.3912363193898
=====
芒果蛋糕：
sales = 4.458629545129518*t+113.70979785954836
未来6个月销量预测：
247.4686842134339
251.9273137585634
256.3859433036929
260.84457284882245
265.30320239395195
269.7618319390815
=====
秘密花园：
sales = 4.518793631786534*t+127.70787336750466
未来6个月销量预测：
263.2716823211007
267.7904759528872
272.30926958467376
276.8280632164603
281.34685684824683
285.86565048003337
=====
暗香：

```

其中，得到的嵌套列表 sales_forecast 的内容如下：

[[585.9023094885982, 589.0000948547565, 592.0978802209149, 595.1956655870732, 598.2934509532315, 601.3912363193898], [247.4686842134339, 251.9273137585634, 256.3859433036929, 260.84457284882245, 265.30320239395195, 269.7618319390815], [263.2716823211007, 267.7904759528872, 272.30926958467376, 276.8280632164603, 281.34685684824683, 285.86565848003337], [513.8876628488803, 516.8948026135172, 519.9019423781541, 522.909082142791, 525.916221907428, 528.923361672065], [234.27639394219034, 238.4574330289585, 242.63847211571135, 246.81951120247186, 251.00055028923236, 255.18158937599287], [99.79324796522417, 99.93468981727266, 100.07613166932113, 100.21757352136962, 100.35901537341809, 100.50045722546656], [78.68261345225118, 79.26612573312326, 79.84963801399533, 80.43315029486742, 81.0166625757395, 81.60017485661159], [61.77935359064462, 62.62248578693256, 63.4656179832285, 64.30875017950844, 65.15188237579638, 65.99501457208432], [212.40837594463784, 214.7840578997645, 217.1597398548911, 219.53542181001774, 221.91110376514436, 224.28678572027098], [141.7370563459746, 143.13180390011541, 144.52655145425626, 145.92129900839709, 147.31604656253793, 148.71079411667876], [1180.761596681689, 1186.6128466346863, 1192.4640965876833, 1198.3153465406806, 1204.1665964936776, 1210.0178464466749], [943.29860873497186, 947.411529727955, 951.5244521061913, 955.6373744844277, 959.7502968626642, 963.86321924090005], [627.9930357457707, 630.6735895787364, 633.3541434117019, 636.0346972446675, 638.7152510776331, 641.3958049105987], [112.990179876217, 115.99645308984377, 119.00272630347054, 122.00899951709731, 125.01527273072408, 128.02154594435086], [63.461351146360784, 65.62475628944397, 67.78816143252715, 69.95156657561033, 72.11497171869351, 74.27837686177669], [27.236371961710418, 27.86294602869514, 28.489520095679858, 29.116094162664574, 29.74266822964929, 30.369242296634006], [105.5086426735149, 105.99490145671909, 106.48116023992328, 106.96741902312748, 107.45367780633167, 107.93993658953586], [143.913501805231, 143.80694640665203, 143.7003910080731, 143.59383560949414, 143.4872802109152, 143.38072481233624], [278.2056906437765, 282.9626703874833, 287.71965013119006, 292.4766298748969, 297.23360961860374, 301.9905893623105], [224.599999982531, 228.6858155846245, 232.77163118671803, 236.85744678881153, 240.94326239090506, 245.02907799299857], [1939.51247025092, 1962.3541343307102, 1985.1957984105002, 2008.0374624902904, 2030.8791265700806, 2053.7207906498707], [2074.815250697592, 2097.1639198403777, 2119.5125889831634, 2141.8612581259495, 2164.209927268735, 2186.5585964115207], [1748.791712642282, 1772.3678035966832, 1795.9438945510847, 1819.519985505486, 1843.0960764598872, 1866.6721674142887], [1704.792191334177, 1727.9238794695161, 1751.0555676048552, 1774.1872557401944, 1797.3189438755333, 1820.4506320108724], [1721.8148447694323, 1746.8929472131508, 1771.9710496568696, 1797.0491521005883, 1822.127254544307, 1847.2053569880259], [407.57762167343816, 411.3404100315149, 415.1031983895916, 418.8659867476684, 422.62877510574515, 426.39156346382185], [395.84029203817107, 399.59935124921975, 403.35841046026843, 407.11746967137177, 410.87652888236585, 414.6355880934145], [411.69541493057807, 415.63752830618387, 419.5796416817897, 423.5217550573995, 427.4638684330014, 431.4059818086072], [662.5453985191455, 674.066280011581, 685.5871615040163, 697.1080429964517, 708.6289244888872, 720.14980859813225], [624.9435927925758, 635.788209602026, 646.6328264114293, 657.477443220856, 668.3220600302827, 679.1666768397095]]

（5）根据预测的结果，给出未来半年合理的原材料采购方案

首先，我们要打开文件'配方成本表-蛋糕.csv'、'配方成本表-烘焙.csv'、'配方成本表-面包.csv'，并从中读取每种产品的各种原材料的名称（保存嵌套列表 `cai_liao`），以及每种产品所需的各种原材料的数量（保存嵌套列表 `shu_liang`）。

```

import csv
import codecs

#这三张表中第一条数据的商品名称
first_good=['芒香','南瓜原料套装','提子面包']

#读取每种产品的各种原材料的名称
f1 = codecs.open('配方成本表-蛋糕.csv', 'r', 'UTF-8')
f2 = codecs.open('配方成本表-烘焙.csv', 'r', 'UTF-8')
f3 = codecs.open('配方成本表-面包.csv', 'r', 'UTF-8')
r1 = csv.reader(f1)
r2 = csv.reader(f2)
r3 = csv.reader(f3)
r_list=[r1,r2,r3]

cai_liao=[]
sub_cai_liao=[]

for i in range(3):
    flag=1 #因为第一行是表头， flag 用来剔除这一行不处理
    name=new_name=first_good[i]
    for row in r_list[i]:
        if flag==1:
            flag=0
            continue
        new_name=row[1]
        if name==new_name:
            sub_cai_liao.append(row[4])
        else:
            cai_liao.append(sub_cai_liao)
            sub_cai_liao=[]
            sub_cai_liao.append(row[4])
            name=new_name
        cai_liao.append(sub_cai_liao)
    sub_cai_liao=[]

```

```
#读取每种产品的各种原材料的数量
f1 = codecs.open('配方成本表-蛋糕.csv', 'r', 'UTF-8')
f2 = codecs.open('配方成本表-烘焙.csv', 'r', 'UTF-8')
f3 = codecs.open('配方成本表-面包.csv', 'r', 'UTF-8')

r1 = csv.reader(f1)
r2 = csv.reader(f2)
r3 = csv.reader(f3)
r_list=[r1,r2,r3]

shu_liang=[]
sub_shu_liang=[]

for i in range(3):
    flag=1
    name=new_name=first_good[i]
    for row in r_list[i]:
        if flag==1:
            flag=0
            continue
        new_name=row[1]
        if name==new_name:
            sub_shu_liang.append(row[6])
        else:
            shu_liang.append(sub_shu_liang)
            sub_shu_liang=[]
            sub_shu_liang.append(row[6])
            name=new_name
    shu_liang.append(sub_shu_liang)
    sub_shu_liang=[]
```

得到的 `cai_liao`、`shu_liang` 这两个嵌套列表的内容如下（这里 `cai_liao[i][j]` 表示第 i 种产品中第 j 种原材料的名称，`shu_liang[i][j]` 表示第 i 种产品所需第 j 种原材料的数量）：

```
[['全脂牛奶', '奶油', '细砂糖', '芒果', '面粉', '鸡蛋', '黄油'], ['全脂牛奶', '大豆油', '奶油', '泡打粉', '细砂糖', '芒果', '面粉', '鸡蛋'], ['大豆油', '奶油', '小麦粉', '巧克力', '细砂糖', '芒果', '草莓', '食盐', '鸡蛋'], ['可可粉', '奶油', '小麦粉', '巧克力', '细砂糖', '草莓', '鸡蛋'], ['大豆油', '奶油', '小麦粉', '牛乳', '细砂糖', '葡萄干', '鸡蛋'], ['大豆油', '奶油', '小麦粉', '果酱', '牛乳', '细砂糖', '芒果', '蓝莓', '鸡蛋'], ['奶油', '小麦粉', '巧克力', '榴莲', '细砂糖', '鸡蛋'], ['全脂牛奶', '奶油', '细砂糖', '面粉', '鸡蛋', '黄油'], ['奶油', '小麦粉', '细砂糖', '芒果', '草莓', '蓝莓', '鸡蛋'], ['全脂牛奶', '可可粉', '小麦粉', '干酪', '细砂糖', '草莓', '鸡蛋'], ['小麦粉', '细砂糖', '黄油', '不粘派盘'], ['黄油', '面粉', '可可粉', '果酱'], ['面粉', '全脂牛奶', '黄油', '鸡蛋', '食盐', '细砂糖'], ['奶油', '鸡蛋', '小麦粉', '巧克力', '细砂糖', '可可粉'], ['安佳黄油'], ['kiri奶油奶酪'], ['全脂牛奶', '鸡蛋', '蛋挞皮', '细砂糖'], ['面粉', '黄油', '香咸蛋黄', '豆沙', '黑芝麻'], ['牛轧糖'], ['面粉', '细砂糖', '食盐', '搅子干', '酵母'], ['面粉', '食盐', '酵母'], ['面粉', '全脂牛奶', '鸡蛋', '食盐', '细砂糖', '酵母'], ['面粉', '细砂糖', '食盐', '鸡蛋', '黄油'], ['面粉', '小麦粉', '酵母', '黑糖', '食盐', '桂圆肉', '黄油'], ['面粉', '细砂糖', '全脂牛奶', '酵母', '食盐', '鸡蛋', '芒果', '黄油'], ['面粉', '鸡蛋', '全脂牛奶', '食盐', '细砂糖', '酵母', '黄油', '火腿', '奶酪'], ['面粉', '食盐', '细砂糖', '黄油', '全脂牛奶', '鸡蛋'], ['面粉', '黄油', '食盐', '细砂糖', '芥末', '培根', '面粉']]

[['100.00', '120.00', '50.00', '50.00', '35.00', '50.00', '15.00'], ['150.00', '50.00', '100.00', '10.00', '200.00', '400.00', '160.00', '200.00'], ['100.00', '100.00', '300.00', '100.00', '50.00', '200.00', '100.00', '10.00', '250.00'], ['50.00', '200.00', '70.00', '90.00', '50.00', '80.00', '50.00'], ['50.00', '240.00', '85.00', '120.00', '70.00', '50.00', '100.00'], ['40.00', '160.00', '50.00', '20.00', '80.00', '30.00', '50.00', '30.00', '100.00'], ['130.00', '40.00', '50.00', '60.00', '30.00', '50.00'], ['130.00', '150.00', '70.00', '100.00', '150.00', '50.00'], ['120.00', '30.00', '20.00', '30.00', '50.00', '20.00', '50.00'], ['100.00', '35.00', '80.00', '25.00', '50.00', '80.00', '100.00'], ['500.00', '150.00', '100.00', '2.00'], ['100.00', '200.00', '50.00', '100.00'], ['400.00', '300.00', '150.00', '300.00', '30.00', '50.00'], ['150.00', '200.00', '300.00', '100.00', '50.00', '40.00'], ['20.00'], ['33.00'], ['100.00', '150.00', '5.00', '50.00'], ['500.00', '50.00', '3.00', '250.00', '30.00'], ['180.00'], ['50.00'], ['120.00', '6.00', '3.00', '50.00', '15.00'], ['150.00', '10.00', '10.00'], ['60.00', '50.00', '100.00', '15.00', '20.00', '3.00'], ['200.00', '4.00', '2.00', '50.00', '3.00', '20.00'], ['150.00', '2.00', '30.00', '23.00', '3.00', '50.00', '20.00'], ['200.00', '2.00', '50.00', '8.00', '10.00', '100.00', '100.00', '10.00'], ['100.00', '50.00', '50.00', '2.00', '5.00', '2.00', '10.00', '10.00', '10.00'], ['100.00', '2.00', '5.00', '10.00', '20.00', '50.00'], ['10.00', '10.00', '3.00', '4.00', '2.00', '10.00', '100.00']]
```

最后，我们求出未来 6 个月各种原材料的采购策略（在所有商品数据中，最后一条数据的日期都是 2018/12）。

```
purchase={  
    'kiri 奶油奶酪':0,  
    '不粘派盘':0,  
    '全脂牛奶':0,  
    '可可粉':0,  
    '培根':0,  
    '大豆油':0,  
    '奶油':0,  
    '奶酪':0,  
    '安佳黄油':0,  
    '小麦粉':0,  
    '巧克力':0,  
    '干酪':0,  
    '提子干':0,  
    '果酱':0,  
    '桂圆肉':0,  
    '椰蓉粉':0,  
    '榴莲':0,  
    '泡打粉':0,  
    '火腿':0,  
    '炼乳':0,  
    '牛乳':0,  
    '牛轧糖':0,  
    '细砂糖':0,  
    '芒果':0,  
    '芥末':0,  
    '草莓':0,  
    '葡萄干':0,  
    '蓝莓':0,  
    '蛋挞皮':0,  
    '豆沙':0,  
    '酵母':0,  
    '面粉':0,  
    '食盐':0,  
    '香咸蛋黄':0,  
    '鸡蛋':0,  
    '黄油':0,  
    '黑糖':0,  
    '黑芝麻':0  
}
```



```

#第 i 年的原材料采购策略
for i in range(6):
    #第 j 种产品
    for j in range(30):
        #第 j 种产品的第 k 种原材料
        for k in range(len(cai_liao[j])):
            purchase[cai_liao[j][k]]=purchase[cai_liao[j][k]]+sales_forecast[j][i]*eval(shu_liang[j][k])
print('=====')
print('【2019 年'+str(i+1)+'月的原材料采购策略】: ')
for n in purchase:
    print(str(n)+'-'+str(purchase.get(n)))
    #初始化 purchase
    purchase[n]=0

```

最后得到原材料的采购策略输出如下：

```

=====
【2019年1月的原材料采购策略】：
kiri奶油奶酪:898.8002747364438
不粘派盘:2361.523193363378
全脂牛奶:457931.60912427335
可可粉:82339.71767708774
培根:6249.435927925759
大豆油:54406.15205850024
奶油:358286.2754646013
奶酪:4116.95414930578
安佳黄油:1269.2270229272158
小麦粉:797220.4982571193
巧克力:87810.20654874356
干酪:3543.426408649365
提子干:96975.623512546
果酱:96325.72569427635
桂圆肉:12227.328650203144
椰蓉粉:11229.999999126549
榴莲:4720.95680713507
泡打粉:2474.686842134339
火腿:4116.95414930578
炼乳:51654.44534308297
牛乳:36096.62711028077
牛乳糖:50077.02431587977
细砂糖:476195.3287905386
芒果:231882.86850444105
芥末:1249.8871855851517
草莓:89397.56456493035
葡萄干:11713.819697109517
蓝莓:7241.964957849482
蛋挞皮:527.5432133675745
豆沙:35978.37545130775
酵母:68858.0669933948
面粉:2163066.6034061965
食盐:92050.55061060288
香咸蛋黄:431.74050541569295
鸡蛋:908031.1144193297
黄油:423312.09331547475
黑糖:9374.285298489078
黑芝麻:4317.4050541569295
=====
【2019年2月的原材料采购策略】：
kiri奶油奶酪:919.4772189469396
不粘派盘:2373.2256932693726
全脂牛奶:461806.3268687016

```

5.3.2 实验二 供应商聚类分析

1、实验目的

- (1) 了解 hive 数据仓库的使用;
- (2) 掌握利用主成分分析法 (PCA) 对数据进行降维的方法;
- (3) 掌握 K-means 和 DBSCAN 两种常用的聚类分析算法;
- (4) 掌握 Python 图表绘制的方法;
- (5) 掌握基本的数据统计方法。

2、实验环境

对 hive 数据仓库中的供应商数据, 在 DataOne 大数据实验平台中, 利用 Python 语言进行分析。

3、实验背景

对于工业企业来说，选择合适的供应商，有效控制企业成本，是企业提升盈利水平，持续健康发展的重中之重。在大数据时代下，伴随着供应链全球化的趋势日益增强，供应商的数量愈加巨大，以往对供应商的选择方式已经不适用于企业科学有效地选择供应商需求。

现借助大数据挖掘技术，通过研究某工业企业现有的供应商评价指标分值，将供应商进行分类，挖掘供应商评价指标分值的分布特点及规律，构建适合工业企业的科学、规范、高效的供应商选择体系。

评价指标关系如下：

一级评价指标	二级评价指标
企业素质	领导素质
	员工素质
	经营理念
	技术设备
企业能力	年营业额
	设备更新期限
	揽货能力
	固定资产
	员工数量
服务环境	子公司、营业网点数量
	地理位置
	信息传递及时率
	服务价格
交易能力	技术水平
	交货期
	退货率
	长期客户数量
物流能力	业务覆盖率
	送货频率
	车辆数量
	信息系统应用覆盖率
	物流驻点
售后服务	仓库数量
	物流设施的数量
	客户满意度
	次品率
	售后维护点
	客户维护期限

本数据集包括某工业企业 199 个供应商的评价指标分值数据，数据通过量化、脱敏处理，样本示例如下：

	src_id	src_leadership_quality	src_staff_quality	src_management_idea	src_technical_equipment	src_volume	src_equipment_update_period	src_freight_quality	src_fixed_assets
1	id	leadership_quality	staff_quality	management_idea	technical_equipment	volume	equipment_update_period	freight_quality	fixed_assets
2	SUP007749	6.7	7.2	7.1	7.9	6.7	6.8	7.6	6.2
3	SUP008992	7.2	6.2	7.7	7.3	6	7.1	7.1	7.7
4	SUP010988	6.9	6.6	6.3	6.6	7.9	7.9	7.7	7.2
5	SUP019294	6.1	6.6	7	6.2	6.4	6.2	7.7	7.2
6	SUP019907	4.8	4.3	4.2	5.8	4	4.6	4.4	5.5
7	SUP022376	6.1	6.4	7	7.1	6.7	6.6	7.3	6.8
8	SUP034800	6.9	6.4	7.3	7	6.8	8	7.4	7.5
9	SUP045015	7.2	7.4	6.7	7.4	6.5	7.3	7.3	8
10	SUP047176	9.8	9.8	8.9	8.6	9.8	8.9	8.7	9.9
11	SUP050048	5	4.6	4.6	4.2	5.7	5.6	5.7	4.8
12	SUP050549	4.6	5.5	5	4.3	4.1	4.2	4.9	5.9
13	SUP054721	4.4	5.3	4.9	4.9	5.4	4.1	4.3	5.1
14	SUP057902	6.7	6.7	7	6.7	6	8	6	6.6
15	SUP060319	6.6	6.8	7.9	7.4	6.1	7.8	7.5	7.9

数据特征描述对应如下：

id -> 供应商编号

leadership_quality -> 领导素质

staff_quality -> 员工素质

management_idea -> 经营理念

technical_equipment -> 技术设备

volume -> 年营业额

equipment_update_period -> 设备更新期限

freight_quality -> 揽货能力

fixed_assets -> 固定资产

staff_number -> 员工数量

organization_number -> 子公司、营业网点数量

location -> 地理位置

information_timely_rate -> 信息传递及时率

service_price -> 服务价格

technological_level -> 技术水平

delivery_date -> 交货期

refund_rate -> 退货率

longterm_customer_number -> 长期客户数量

business_coverage -> 业务覆盖率

delivery_frequency -> 送货频率

vehicle_number -> 车辆数量

information_system_coverage -> 信息系统应用覆盖率

logistics_point -> 物流驻点

warehouse_number -> 仓库数量

logistics_facility_number -> 物流设施的数量

customer_satisfaction -> 客户满意度

defective_percentage -> 次品率

aftersales_point -> 售后维护点

customer_maintenance_period -> 客户维护期限

4、实验内容

- (1) 连接 hive 数据仓库并读取供应商数据;
- (2) 使用主成分分析 (PCA) 对数据的特征值进行降维处理;
- (3) 利用 K-means 算法对供应商进行聚类, 并使用 Calinski-Harabasz 准则评价模型的拟合度;
- (4) 利用 DBSCAN 算法对供应商进行聚类, 并使用 Calinski-Harabasz 准则评价模型的拟合度;
- (5) 挑选出存在特项异常的供应商, 并找出他们的专长项。

5、参考代码

(1) 连接 hive 数据仓库并读取数据

首先, 我们先要使用 Python 的 pyhive 库与 hive 数据仓库进行连接, 获取到数据表的内容并用其生成一个 dataframe。

```
from pyhive import hive
import pandas as pd

#连接 hive 中的数据库
conn = hive.Connection(host='192.168.94.206', port=10000, username='hive', database='szu')
cursor = conn.cursor()
#获取数据表
cursor.execute("select * from supplier_data")
#构造 dataframe
result = cursor.fetchall()
data = pd.DataFrame(data=list(result[1:]), columns=list(result[0]))
#关闭连接
cursor.close()
conn.close()
```

得到的 dataframe 的大致内容如下所示:

	id	leadership_quality	staff_quality	management_idea	technical_equipment	volume	equipment_update_period	freight_quality	fixed_assets
0	SUP007749	6.7	7.2	7.1	7.9	6.7	6.8	7.6	6.2
1	SUP008992	7.2	6.2	7.7	7.3	6	7.1	7.1	7.7
2	SUP010988	6.9	6.6	6.3	6.6	7.9	7.9	7.7	7.2
3	SUP019294	6.1	6.6	7	6.2	6.4	6.2	7.7	7.2
4	SUP019907	4.8	4.3	4.2	5.8	4	4.6	4.4	5.5
...
194	SUP972756	6.9	7.3	7.8	6.5	6.2	6.9	8	6.6
195	SUP980578	6.8	7.7	6.3	8	7.6	7.8	6.8	6.4
196	SUP983470	5.4	4	5.1	4.3	4.5	5.8	5.2	5.5
197	SUP987060	4.3	5.4	4.4	5.5	5.2	4.8	5	4.6
198	SUP988696	7.8	7.6	7.8	7.2	6.3	7.9	6.4	6.1

199 rows x 10 columns

(2) 使用 PCA 对数据进行降维处理

因为数据集中样本的特征值非常多，将近 30 个，也就是将近 30 维，所以我们要先对样本特征进行降维处理。在这里，我们采用主成分分析（PCA）来对数据进行降维处理。

在进行主成分分析之前，我们先对数据集中一些无关的数据进行剔除，包括"Unnamed: 0"、"id"、"20201207"这 3 列数据，并构造一个训练集 train。然后，再将数据集中的“供应商 ID”，和训练集中的“特征”提取出来，方便后面分析时使用。

```
#删除无用的列，构造训练集 train
train=data.drop(["Unnamed: 0","id","20201207"], axis=1)

#提取供应商 ID
sup_id=data['id']
#构造特征向量
feature=list(train.columns)
```

首先，我们先要对数据集的 eigenvalue 进行分析，看看要降成几维比较合适。

```
from sklearn.decomposition import PCA

#使用主成分分析 PCA 进行降维。因为目前我们不知道要降成多少维，为了分析其 eigenvalue 我们先将
维数设置为 28 维（数据集总共 28 个特征）
pca = PCA(n_components=28)
pca.fit_transform(train)
#分析其 eigenvalue，看看要降成几维
eigenvalue=pca.explained_variance_
eigenvalue
```

得到的 eigenvalue 结果如下：

```
array([31.86366621, 14.08570188,  1.22656257,  0.58124522,  0.5594265 ,
        0.53061075,  0.5203859 ,  0.46620109,  0.4569424 ,  0.44604706,
        0.42779386,  0.4157626 ,  0.39070124,  0.34531282,  0.33192607,
        0.32112006,  0.30737716,  0.28239017,  0.28083059,  0.27725171,
        0.26111925,  0.25330198,  0.23606832,  0.22664027,  0.21712987,
        0.21477544,  0.16583797,  0.1456473 ])
```

因为 eigenvalue 前两项值很大，后面的值相比都非常小，所以我们将维度降为 2 维。

```
#因为 eigenvalue 前两项值很大，后面的值相比都非常小，故将维度降为 2 维
pca = PCA(n_components=2)
#将结果生成一个 dataframe
data_pca = pd.DataFrame(pca.fit_transform(train))
```

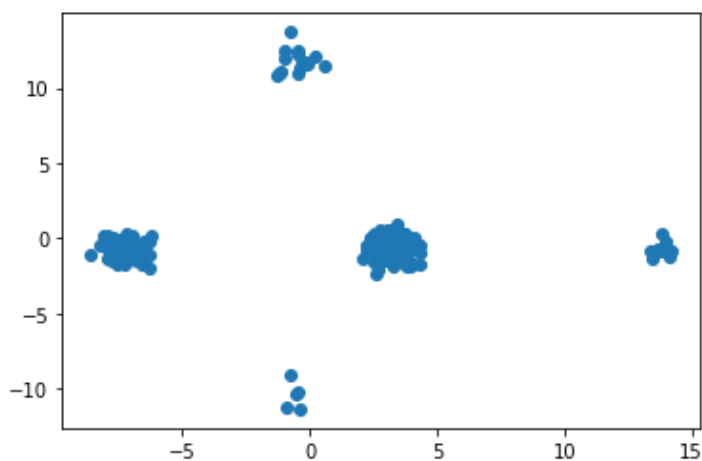
(3) 利用 K-means 算法进行聚类分析

首先，我们先对降维后的数据进行散点图的绘制，观察数据的大致分布情况。

```
import matplotlib.pyplot as plt

#绘制散点图查看数据点大致情况
plt.scatter(data_pca[0],data_pca[1])
```

得到的散点图如下：



通过观察，我们可以大致知道这些供应商应该被分为 5 类。因此，我们接下来利用 K-means 算法将供应商分成 5 类。

```

from sklearn.cluster import KMeans

#根据上面散点图的分布情况，我们将数据点分类为 5 类
kmmodel = KMeans(n_clusters=5) #创建模型
kmmodel = kmmodel.fit(train) #训练模型
ptarget = kmmodel.predict(train) #对原始数据进行标注

#查看各个供应商所分到的类别
print("供应商"+"\\t"+"类别")
print("-----")
for i in range(199):
    print(sup_id[i]+"\\t"+str(ptarget[i]))

```

分类的结果如下（部分结果截图）：

供应商	类别
SUP007749	0
SUP008992	0
SUP010988	1
SUP019294	0
SUP019907	2
SUP022376	1
SUP034800	0
SUP045015	0
SUP047176	3
SUP050048	2
SUP050549	2
SUP054721	2
SUP057902	0
SUP060319	0
SUP071582	2
SUP078872	0
SUP081317	2
SUP096359	2
SUP101540	4
SUP106315	0
SUP113097	0
SUP128347	2
SUP142100	0
SUP153234	2
SUP155475	0
SUP156980	3
SUP157193	0
SUP161238	1
SUP166241	2
SUP166893	0
SUP170506	0
SUP179916	2

为了更便于我们对结果的观察，我们对分类的结果进行交叉表和散点图的绘制。

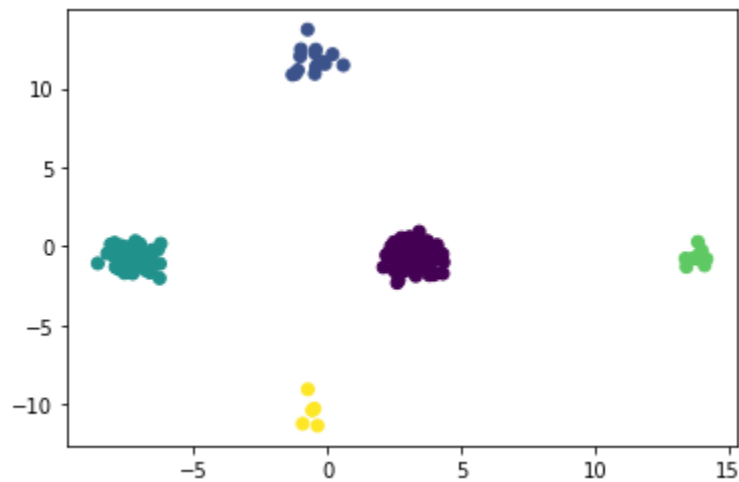

```
#交叉表查看各个类别数据的数量
pd.crosstab(ptarget,ptarget)

#查看聚类的分布情况
plt.scatter(data_pca[0],data_pca[1],c=ptarget)
```

得到的交叉表如下：

col_0	0	1	2	3	4
row_0					
0	107	0	0	0	0
1	0	15	0	0	0
2	0	0	63	0	0
3	0	0	0	9	0
4	0	0	0	0	5

得到的散点图如下：



最后，使用 Calinski-Harabasz 准则评价模型的拟合度。

```
from sklearn.metrics import calinski_harabasz_score

#使用 Calinski-Harabasz 准则评价模型的拟合度
CH_score_KMeans=calinski_harabasz_score(data_pca, ptarget)
CH_score_KMeans
```

得到该聚类的 Calinski-Harabasz 评价得分为 3404.7563。

(4) 利用 DBSCAN 算法进行聚类分析

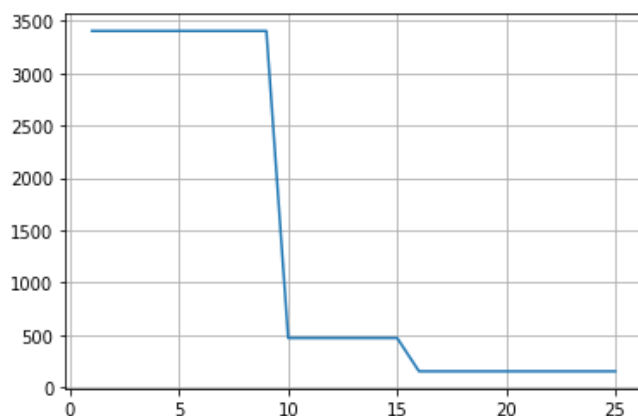
在 DBSCAN 中，我们要确定 2 个参数，一个是邻域的距离阈值 ϵ ，一个是样本点要成为核心对象所需要的样本数阈值 MinPts 。通过对散点图的观察，我们大致可以确定邻域的距离阈值 $\epsilon=5$ 。对于样本点要成为核心对象所需要的样本数阈值 MinPts ，我们使用遍历的方式，在 1~25 这一区间内逐个尝试，对各个 MinPts 下聚类结果的评分进行折线图绘制，挑选出 Calinski-Harabasz 准则评分最高的一个。

```
from sklearn.cluster import DBSCAN

#参数选取，固定  $\epsilon=5$ ， $\text{MinPts}=[1:25]$ 
MinPts_list=[]
CH_score_DBSCAN_list=[]
for i in range(25):
    #邻域的距离阈值
     $\epsilon = 5$ 
    #样本点要成为核心对象所需要的样本数阈值
    MinPts = i+1
    MinPts_list.append(MinPts)
    #模型训练
    model = DBSCAN( $\epsilon=\epsilon$ , min_samples=MinPts)
    #对训练集的数据进行分类
    Type = model.fit_predict(data_pca)
    #使用 Calinski-Harabasz 准则评价模型的拟合度
    CH_score_DBSCAN=calinski_harabasz_score(data_pca, Type)
    CH_score_DBSCAN_list.append(CH_score_DBSCAN)

#对各个  $\text{MinPts}$  下聚类结果的评分进行折线图绘制
plt.plot(MinPts_list,CH_score_DBSCAN_list)
plt.grid(True)
```

得到的折线图如下：



通过对该折线图的观察，我们发现当 MinPts 的取值为 1-9 时，聚类效果最佳，且聚类效果一样。所以接下来我们就取 $\epsilon=5$ ，MinPts=3 来进行 DBSCAN 聚类。

```
#邻域的距离阈值
e = 5
#样本点要成为核心对象所需要的样本数阈值
MinPts = 3
#模型训练
model = DBSCAN(eps=e, min_samples=MinPts)
#对训练集的数据进行分类
Type = model.fit_predict(data_pca)

#查看各个供应商所分到的类别
print("供应商"+"\\t"+"类别")
print("-----")
for i in range(199):
    print(sup_id[i]+"\\t"+str(Type[i]))
```

聚类得到的结果如下（部分结果截图）：

供应商	类别
SUP007749	0
SUP008992	0
SUP010988	1
SUP019294	0
SUP019907	2
SUP022376	1
SUP034800	0
SUP045015	0
SUP047176	3
SUP050048	2
SUP050549	2
SUP054721	2
SUP057902	0
SUP060319	0
SUP071582	2
SUP078872	0
SUP081317	2
SUP096359	2
SUP101540	4
SUP106315	0
SUP113097	0
SUP128347	2
SUP142100	0
SUP153234	2
SUP155475	0

为了更便于我们对结果的观察，我们对分类的结果进行交叉表和散点图的绘制。

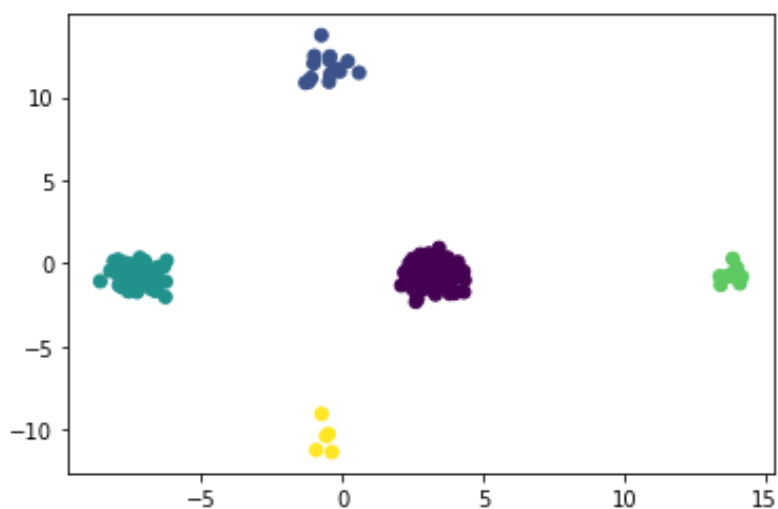
```
#交叉表查看各个类别数据的数量
pd.crosstab(Type,Type)

#查看聚类的分布情况
plt.scatter(data_pca[0], data_pca[1], c=Type)
```

得到的交叉表如下：

col_0	0	1	2	3	4
row_0					
0	107	0	0	0	0
1	0	15	0	0	0
2	0	0	63	0	0
3	0	0	0	9	0
4	0	0	0	0	5

得到的散点图如下：



最后，使用 Calinski-Harabasz 准则评价模型的拟合度。

```
#使用 Calinski-Harabasz 准则评价模型的拟合度
CH_score_DBSCAN=calinski_harabasz_score(data_pca, Type)
CH_score_DBSCAN
```

得到该聚类的 Calinski-Harabasz 评价得分为 3404.7563。该得分与 K-means 算法聚类的结果是一样的。

(5) 挑选出存在特项异常的供应商，并找出他们的专长项

在这里，我们主要分为两步进行。第一步是找出特项异常供应商，第二步是分析各个特项异常供应商的专长项。

首先，我们先对各个供应商的各项评分的方差进行分析，以确定哪些供应商是特项异常的，哪些供应商是各项评分正常的（各项评分差距不大）。

```
import numpy as np

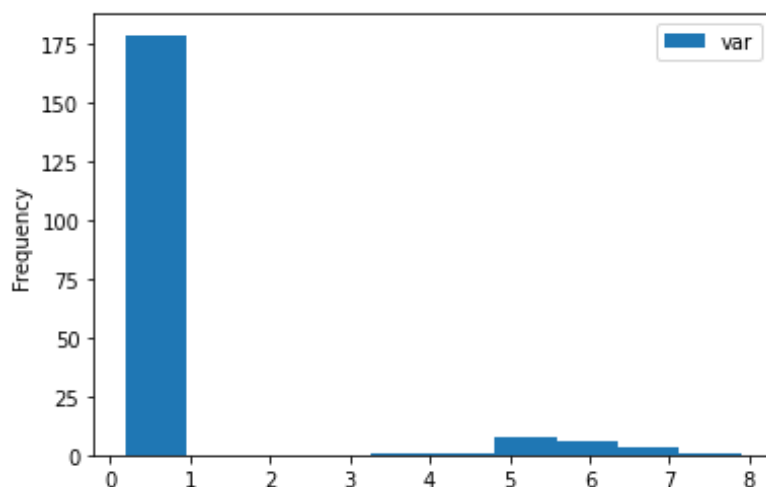
#获取各个供应商的各项评分
var_list=[]
for i in range(199):
    var = np.var(np.array(train[i:i+1]))
var_list.append(var)

#构造 dataframe
var_df=pd.DataFrame(data=var_list,columns=['var'])

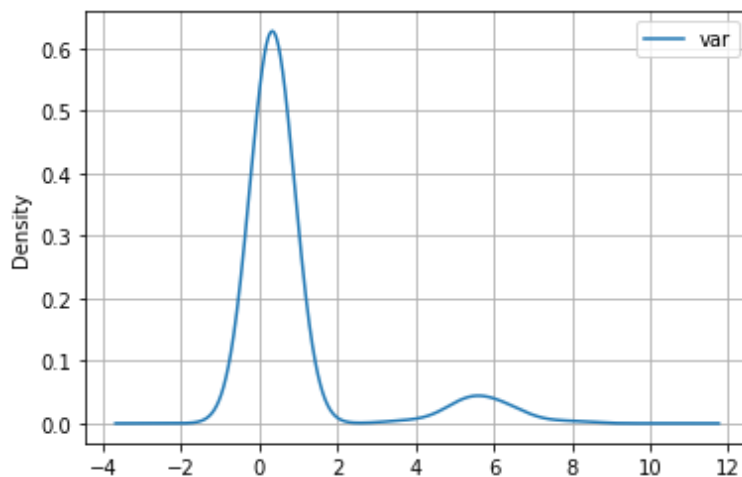
#绘制直方图
var_df.plot(kind='hist')
#显示图表
plt.show()

#绘制密度图
var_df.plot(kind='kde',grid=True)
#显示图表
plt.show()
```

得到的“各个供应商各项评分方差”直方图如下：



得到的“各个供应商各项评分方差”密度图如下：



通过观察分析，我们发现大部分供应商的各项评分方差都是集中在 0~1 之间，只有极少一部分分布在大于 1 的地方。所以，我们令各项评分方差>1 的供应商，为存在特项异常的供应商。而各项评分方差≤1 的供应商，为正常的供应商。

#评分方差大于 1 的供应商的各项评分散点图（特项异常供应商）

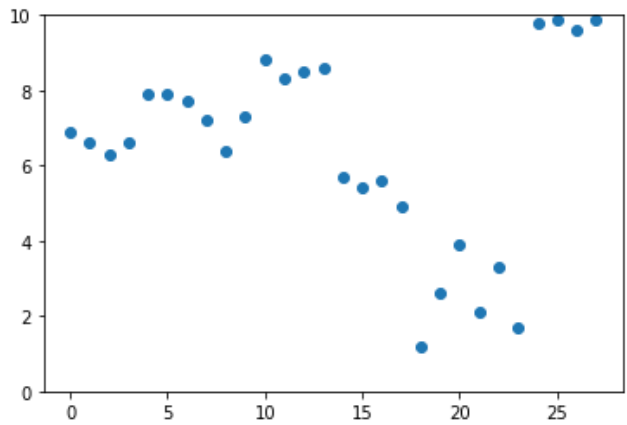
```
special_sid=[]
for i in range(199):
    var = np.var(np.array(train[i:i+1]))
    if var>1:
        special_sid.append(i)
        print("供应商"+str(i))
        print("方差:"+str(var))
        mean = np.mean(np.array(train[i:i+1]))
        print("均值:"+str(mean))
        plt.scatter(range(len(feature)),np.array(train[i:i+1]))
        plt.ylim(0,10)
        plt.show()
```

#评分方差大于等于 1 的供应商的各项评分散点图（正常供应商）

```
for i in range(199):
    var = np.var(np.array(train[i:i+1]))
    if var<=1:
        print("供应商"+str(i))
        print("方差:"+str(var))
        mean = np.mean(np.array(train[i:i+1]))
        print("均值:"+str(mean))
        plt.scatter(range(len(feature)),np.array(train[i:i+1]))
        plt.ylim(0,10)
        plt.show()
```

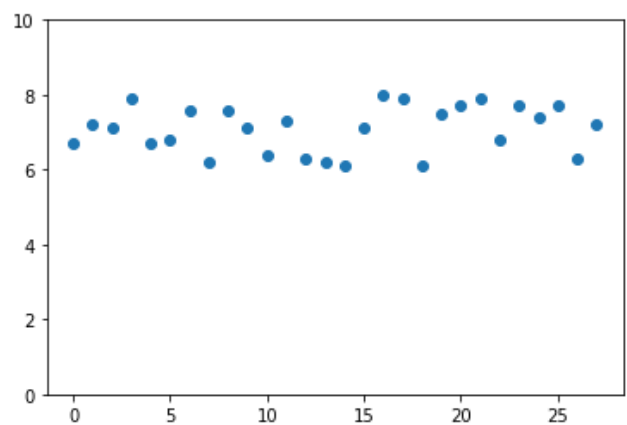
特项异常供应商的各项评分散点图如下（以供应商 2 为代表，其各项评分差异大）：

供应商2
方差:6.254642857142858
均值:6.450000000000001



正常供应商的各项评分散点图如下（以供应商 0 为代表，其各项评分差异小）：

供应商0
方差:0.3752423469387756
均值:7.0892857142857135



在分析出了哪些供应商是特项异常的供应商后，接下来我们分析各个特项异常供应商的专长项。在这里，如果一个特项异常供应商某一项的评分排在所有供应商前 5%的话，我们就认为该供应商的该项是它的特项专长。

```

#保存异常的供应商的专长项
special_suppliers_dictionary={}
for i in special_sid:
    special_suppliers_dictionary[i]=[]
print(special_suppliers_dictionary)

#对各项评分指标一一进行分析
for f in feature:
    print("\n-----")
    print(f+"\n")
    arr=np.array(train[f])
    var = np.var(arr)
    print("方差:"+str(var))
    mean = np.mean(arr)
    print("均值:"+str(mean))

    #Top10
    D=train[f].to_dict()
    sorted_D=sorted(D.items(), key = lambda kv:(kv[1], kv[0]), reverse=True)
    sorted_D=sorted_D[0:10]
    print("\nTop10:")
    print(sorted_D)

    #Top10 in special supplier
    sorted_special_D=[]
    for key_value in sorted_D:
        if (key_value[0] in special_sid):
            sorted_special_D.append(key_value)
    print("\nTop10 in special suppliers:")
    print(sorted_special_D)

    #append
    for kv in sorted_special_D:
        special_suppliers_dictionary[kv[0]].append(f)

#散点图
plt.scatter(range(199),arr)
plt.ylim(0,10)
plt.show()

```



```
#输出特项异常的供应商，及其专长项
for key in special_suppliers_dictionary:
    print("\n 供应商"+str(key)+"的专长项: ")
    print(special_suppliers_dictionary[key])
```

最终，得到各个特项异常供应商的专长项如下：

供应商 2 的专长项:

```
['customer_satisfaction', 'defective_percentage', 'aftersales_point', 'customer_maintenance_period']
```

供应商 5 的专长项:

```
['location', 'information_timely_rate', 'customer_satisfaction', 'aftersales_point']
```

供应商 18 的专长项:

```
['vehicle_number', 'information_system_coverage', 'logistics_point', 'warehouse_number', 'logistics_facility_number']
```

供应商 27 的专长项:

```
['location', 'information_timely_rate', 'service_price', 'customer_maintenance_period']
```

供应商 33 的专长项:

```
['information_timely_rate', 'customer_satisfaction']
```

供应商 52 的专长项:

```
['delivery_frequency', 'vehicle_number', 'information_system_coverage', 'logistics_point', 'warehouse_number', 'logistics_facility_number']
```

供应商 60 的专长项:

```
['information_timely_rate', 'service_price', 'technological_level', 'customer_satisfaction', 'defective_percentage']
```

供应商 63 的专长项:

['delivery_frequency', 'vehicle_number', 'logistics_point', 'warehouse_number']

供应商 67 的专长项:

['location', 'information_timely_rate', 'defective_percentage', 'aftersales_point']

供应商 71 的专长项:

['service_price', 'technological_level', 'defective_percentage', 'customer_maintenance_period']

供应商 75 的专长项:

['technological_level']

供应商 81 的专长项:

['location', 'technological_level', 'customer_maintenance_period']

供应商 116 的专长项:

['location', 'information_timely_rate', 'service_price', 'technological_level', 'aftersales_point', 'customer_maintenance_period']

供应商 125 的专长项:

['aftersales_point']

供应商 128 的专长项:

['location', 'service_price']

供应商 134 的专长项:

['location', 'information_timely_rate', 'service_price', 'customer_satisfaction', 'defective_percentage', 'aftersales_point']

供应商 149 的专长项:

['volume', 'location', 'information_timely_rate', 'service_price', 'technological_level',

'customer_maintenance_period']

供应商 151 的专长项:

['delivery_frequency', 'vehicle_number', 'information_system_coverage', 'logistics_point', 'warehouse_number']

供应商 164 的专长项:

['information_system_coverage', 'logistics_point', 'logistics_facility_number']

供应商 178 的专长项:

['location', 'service_price', 'technological_level', 'customer_maintenance_period']

5.3.3 实验三 客户反馈信息分析

1、实验目的

- (1) 掌握爬取手机 APP 数据的方法;
- (2) 掌握使用 Python 读取 json 文件的方法;
- (2) 掌握使用 jieba 库进行中文文本分词的方法;
- (3) 掌握使用 wordcloud 库进行词云绘制的方法;
- (4) 掌握文本向量化的 TF-IDF 算法;
- (5) 掌握高维数据降维的 t-SNE 算法;
- (6) 掌握 K-means 聚类算法;
- (7) 掌握 Python 的基本绘图方法。

2、实验环境

使用 Airtest 自动化测试工具和 MuMu Android 模拟器，对“美团 APP”中的客户评价进行爬取。对客户评价数据，在 DataOne 大数据实验平台中，利用 Python 语言进行分析。

3、实验背景

对于商家来说，客户的反馈信息是十分重要的，通过对客户的反馈信息进行分析，我们能从中获取到非常多有用的信息。这些信息能帮助商家了解当前的销售状况，为商家未来决策的制定提供指导。

在以往,对于客户评价的分析往往只停留在定性的层面,这样不仅缺乏严谨性、准确性,而且无法对大数据集进行分析处理。现借助大数据挖掘技术,以美团 APP 中的“幸福西饼蛋糕(深圳店)同城配送”门店为例,对客户的反馈信息进行统计、可视化、聚类分析,建立一套完整、科学的客户反馈信息分析方法。

在此次实验中,我们对两组评价数据分别进行分析。一组是我们自己利用 Airstest 在美团 APP 中直接爬取到的评价数据——“外卖评价_仅评论.txt”、“外卖评价_仅评分.txt”、“到店评价_仅评论.txt”、“到店评价_仅评分.txt”。另一组是已有的 json 格式评价数据——“幸福西饼生日蛋糕门店美团评论数据集_1_5k.json”。

其中,利用 Airstest 直接能爬取到的评价数据规模较小(百数量级),且分为“外卖评价”和“到店评价”两大类;而 json 格式的评价数据规模较大(千数量级),且没有“外卖评价”和“到店评价”的区分。

另外,为了方便剔除评价中如:“啊”、“呀”、“并且”、“因为”、“和”等停用词,我们将网上收集到的常用停用词汇汇总在了“stop_words.txt”中。

4、实验内容

(1) 爬取美团 APP “幸福西饼蛋糕(深圳店)同城配送”门店的外卖评价与到店评价。

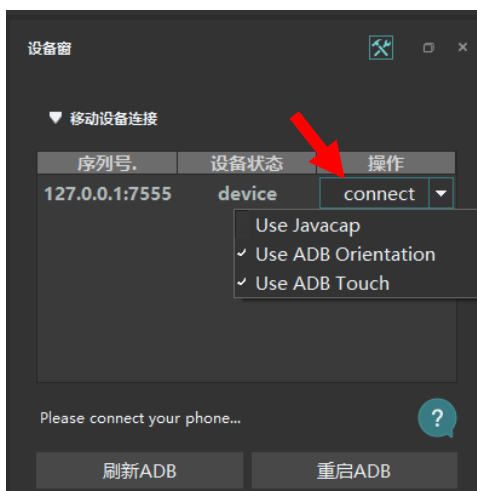


(2) 从“幸福西饼生日蛋糕门店美团评论数据集_1_5k.json”中提取出评价数据。

(3) 筛选抓取的客户评价,找出评价的高频关键词,并绘制如下所示的词云。




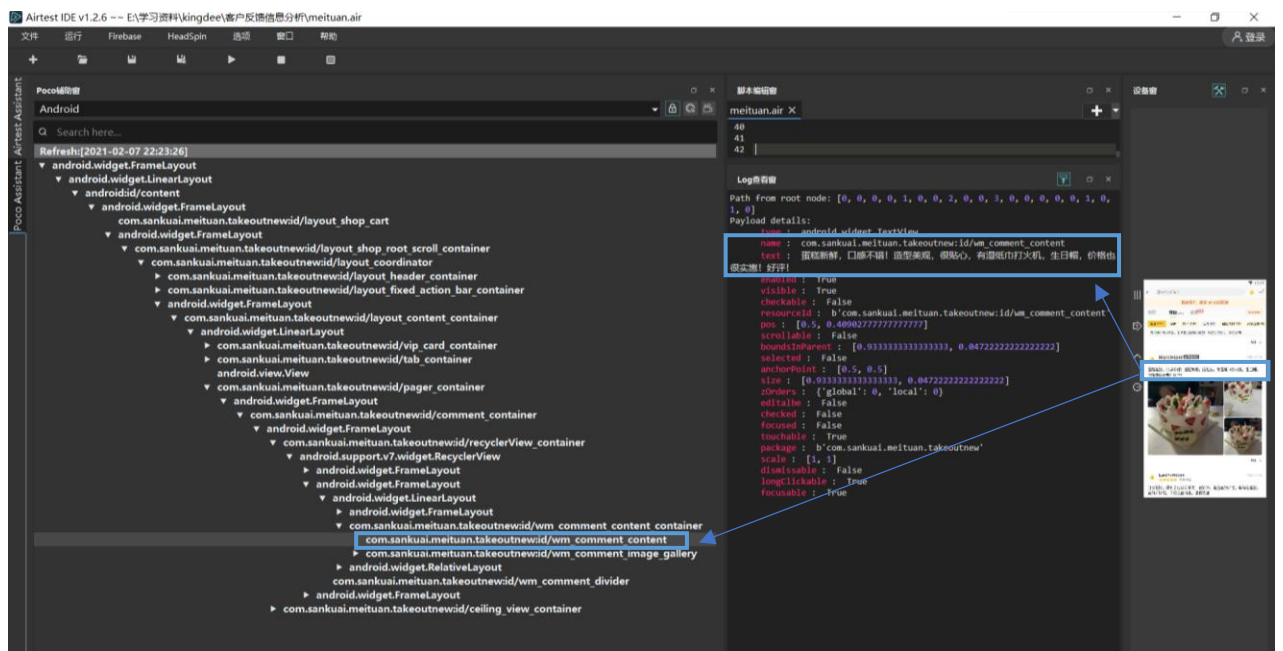
连接成功后，在“移动设备连接”会出现我们的设备信息，这时我们点击“connect”即可完成设备的连接。



此时，“设备窗”中会同步显示设备屏幕上的内容，且在“设备窗”中进行的操作也会同步到设备中。



在这里，我们是通过 APP 中的 UI 部件来获取评价信息的。我们先在设备中进入到美团 APP“幸福西饼蛋糕（深圳店）同城配送”门店的外卖评价页面，然后在“Poco 辅助窗”中选择“Android”，并点击右上方的锁形按钮将界面锁定。之后我们点击选中评论的部分，就可以在“Poco 辅助窗”中查看到当前 UI 部件在整个 UI 的树形层次结构中的位置，以及在“Log 查看窗”查看其详细内容。



通过了上面的分析，我们可以确定评价中评分、评论在整个 UI 的树形层次结构中的位置，及其内容。于是，我们就可以由此写出爬取该门店“外卖评价”的脚本代码如下（爬取该门店“到店评价”的脚本代码是一样的，只要将设备的页面切换到“到店评价”即可）：


```

from airtest.core.api import *
from poco.drivers.android.uiautomation import AndroidUiautomationPoco
poco = AndroidUiautomationPoco(use_airtest_input=True, screenshot_each_action=False)
auto_setup(__file__)

#打开文件，该文件用于保存爬取到的数据
f=open("D:\meituan_data.txt","w")

for i in range(3000):
    try:
        #获取评价
result=poco("android:id/content").child("android.widget.FrameLayout").offspring("com.sankuai.meituan.takeoutnew:id/layout_coordinator").offspring("com.sankuai.meituan.takeoutnew:id/layout_content_container").offspring("com.sankuai.meituan.takeoutnew:id/pager_container").offspring("com.sankuai.meituan.takeoutnew:id/comment_container").offspring("android.support.v7.widget.RecyclerView").child("android.widget.FrameLayout")[1].child("android.widget.LinearLayout")

        #获取评价中的评分
score=result.child(name="android.widget.FrameLayout").child(name="android.widget.LinearLayout")[1].child(name="com.sankuai.meituan.takeoutnew:id/wm_comment_score_text").get_text()

        #获取评价中的评论
comment=result.child(name="com.sankuai.meituan.takeoutnew:id/wm_comment_content_container").child(name="com.sankuai.meituan.takeoutnew:id/wm_comment_content").get_text()

        #输出与保存数据
print(str(i)+"\t"+score+"\t"+comment.replace("\n", " "))
f.write(score+"\t"+comment.replace("\n", " "))
f.write("\n")

        #暂停 1ms
time.sleep(1)

        #向下滑屏幕以获取新的一页评价内容
poco.swipe([0.5,0.8],[0.5,0.2])

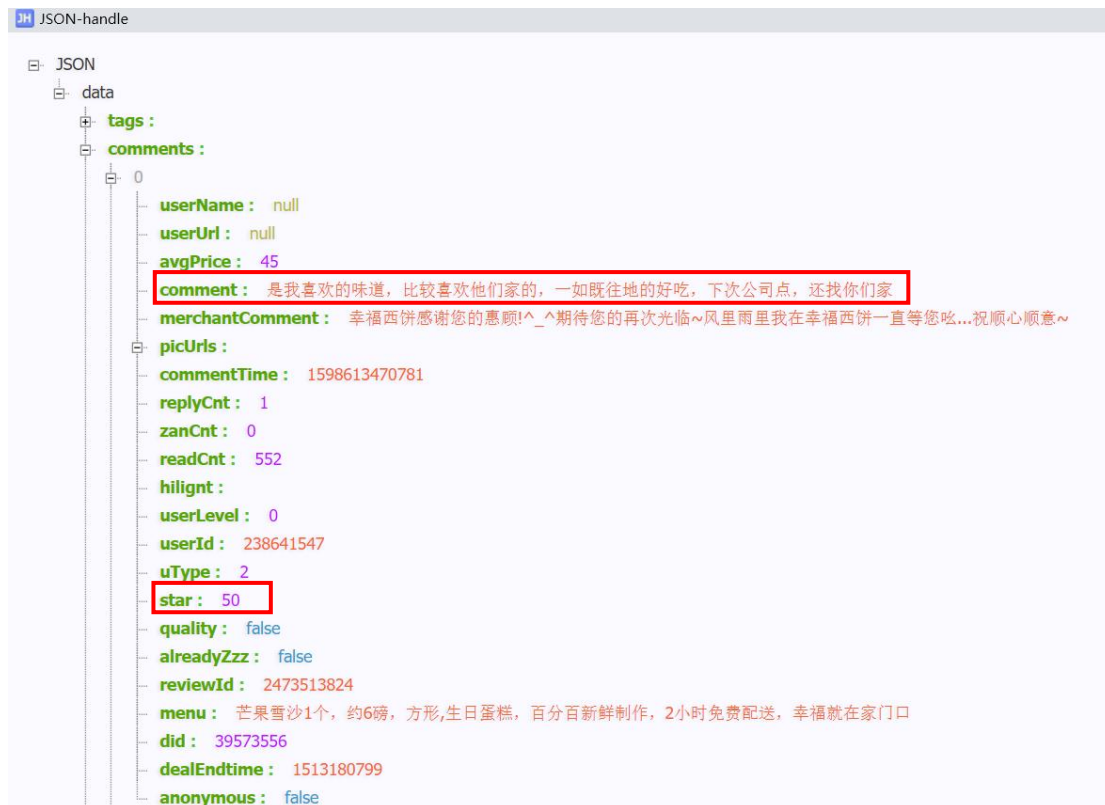
    except:
        #如果在当前页面中没有完整的评价内容，则无法成功获取到数据。这时我们直接向下滑动，跳过此页内容。
        poco.swipe([0.5,0.8],[0.5,0.2])

#关闭文件
f.close()

```

(2) 从“幸福西饼生日蛋糕门店美团评论数据集_1_5k.json”中提取出评价数据

首先，我们先使用 json 文件的查看工具对该 json 文件的结构及其内容进行大致查看。从中我们观察得出，用户评价的位置在“data”-“comments”。在“comments”一项下，共有 5000 条记录，每条记录的结构都是相似的。其中，每条记录中的“comment”一项是评论内容，“star”一项是评分（50 对应于非常满意，40 对应于满意，30 对应于一般，20 对应于差，10 对应于非常差）。



接下来，我们就可以使用 Python 中的 json 库对其进行读取了，读取的代码如下：

```
import json

#打开 json 文件
file=open("幸福西饼生日蛋糕门店美团评论数据集_1_5k.json", "r", encoding="utf-8")
#读取 json 文件的内容，生成一个字典
dic=json.load(file)
#关闭 json 文件
file.close()

#从字典中获取每一条评论的内容
for item in dic.get("data").get("comments"):
    print(item.get("comment"))
```

(3) 筛选抓取的客户评价，找出评价的高频关键词，并绘制词云

在这里，我们可以先使用 jieba 库对评论进行分词操作，再使用 wordcount 库来进行词云的绘制。但是注意两点，一是对评论中的特殊字符进行剔除，只保留中英文和数字；二是要对评论中的停用词进行剔除，如：“啊”、“呀”、“并且”、“因为”、“和”等词。

其中，对 Airstest 爬取到的“外卖评价”进行词云绘制的代码如下：

```
import jieba
import re
from wordcloud import WordCloud
import matplotlib.pyplot as plt

#构造停用词列表
stopwords=[]
stopwordsfile=open("stop_words.txt","r",encoding="utf-8")
for word in stopwordsfile.readlines():
    #去除换行符
    word=word.strip('\n')
    stopwords.append(word)

# 读入数据
waimai = open("外卖评价_仅评论.txt", "r", encoding="utf-8")
text_waimai = waimai.read()

#去除标点，只保留中英文和数字
text_waimai=re.sub('[^\u4e00-\u9fa5^a-z^A-Z^0-9]', '', text_waimai)
waimai.close()

# 结巴中文分词，生成字符串，默认精确模式，如果不通过分词，无法直接生成正确的中文词云
cut_text_waimai = jieba.cut(text_waimai,cut_all=False)

# 必须给个符号分隔开分词结果来形成字符串,否则不能绘制词云
result_waimai = " ".join(cut_text_waimai)

#去除停用字
for w in stopwords:
    result_waimai = re.sub(r"\b%s\b" %(w), "", result_waimai)
```

```
# 生成词云图，这里需要注意的是 WordCloud 默认不支持中文，所以这里需已下载好的中文字库
wc = WordCloud(
    # 设置字体，不指定就会出现乱码
    font_path="simfang.ttf",
    # 设置背景色
    background_color='white',
    # 设置背景宽
    width=250,
    # 设置背景高
    height=175,
    # 最大字体
    max_font_size=50,
    # 最小字体
    min_font_size=10,
    # 避免重复统计词语
    collocations=False,
    mode='RGBA'
)

# 产生词云
wc.generate(result_waimai)

# 以图片的形式显示词云
plt.imshow(wc)
# 关闭图像坐标系
plt.axis("off")
plt.show()

# 保存图片
wc.to_file(r"wordcloud_waimai.png")
```

对于 Airtest 爬取到的“到店评价”、以及“幸福西饼生日蛋糕门店美团评论数据集_1_5k.json”绘制词云的关键代码跟这个是一样的，只是读入数据那部分代码不同。

运行代码后，得到的 Airtest 爬取到的“外卖评价”的词云如下：



Airtest 爬取到的“到店评价”的词云如下：



“幸福西饼生日蛋糕门店美团评论数据集_1_5k.json”中评价的词云如下：



(4) 对抓取的客户评价有效信息进行向量化，采用二维平面对客户评价进行可视化处理

在这里，我们用 TF-IDF 算法来对客户评价进行向量化，用 t-SNE 算法来对评价进行可视化处理。对于 TF-IDF 与 t-SNE 算法的实现，我们可以直接调用 Python 的 sklearn 库。

下面是对 Airtest 爬取到的“外卖评价”进行向量化与可视化处理的代码实现：

首先，我们先剔除评论中的特殊字符，并用 jieba 库完成分词。

```

import jieba
import re
import numpy as np
from sklearn import feature_extraction
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer

#corpus 用于存储分词后的评论，其元素为分词后的一条评论
corpus=[]
f=open("外卖评价_仅评论.txt","r",encoding="utf-8")
for line in f.readlines():
    #去除标点，只保留中英文和数字
    line=re.sub('[^\u4e00-\u9fa5^a-zA-Z^0-9]', "", line)
    #分词
    cut_line=jieba.cut(line,cut_all=False)
    #用空格将分出的各个词语连接成字符串
    result=" ".join(cut_line)
    #append
    corpus.append(result)

```

得到的列表 `corpus` 的内容如下：

```

['经典 四重奏 生日蛋糕',
 '泰式 下午茶 16 杯 木瓜 椰奶 8 块 赠 红茶 和 层 架 抹 茶 木瓜 等 口感 都 不好 买回来 同事 们 基本 都 不吃 木瓜 怀疑 新鲜 程度 还有 品
质 把 控',
 '准时 送达 开心 过 生日蛋糕 很 美味',
 '好次 购买 分量 足 速度 快 干净 卫生 味道 非常 正宗',
 '味道 很 好',
 '很 好',
 '好吃',
 '香颂 草莓 生日蛋糕 配送 时间 为 23 小时 平台 默认 配送 时间 有误 蛋糕 跟 图片 上 的 一样 草莓 也 很 新鲜 蛋糕 很 好吃',
 '太 感谢 了 很 喜欢 幸福 西饼 的 蛋糕 每 年 都 会 买 希望 生意 兴隆',
 '还 不错 蛋糕 完好 无损 就是 餐具 比较 少 忘记 多买 一套 蛋糕 八 个 人 吃 刚刚 好',

```

在完成了分词操作后，我们就对词频进行统计，并计算每个词在每条评论中的 TF-IDF 得分。

```

#该类会将文本中的词语转换为词频矩阵，矩阵元素 a[i][j] 表示 j 词在 i 评论下的词频
vectorizer=CountVectorizer()
#该类会统计每个词语的 tf-idf 权值
transformer=TfidfTransformer()

#将文本转为词频矩阵
frequency=vectorizer.fit_transform(corpus)
frequency_array=frequency.toarray()
#行数（文本个数）
r=frequency_array.shape[0]
#列数（词语个数）
c=frequency_array.shape[1]

#计算 tf-idf
tfidf=transformer.fit_transform(frequency)

#获取词袋模型中的所有词语，生成一个列表 words
words=vectorizer.get_feature_names()
#将 tf-idf 矩阵抽取出来，元素 weight[i][j]表示 j 词在 i 这条评论中的 tf-idf 权重
weight=tfidf.toarray()

#获取词频为 1 的词语（这些词在后面要进行剔除）
sum_c_frequency=np.sum(frequency_array,axis=0)
once_words=[]
for i in range(c):
    if sum_c_frequency[i]==1:
        once_words.append(words[i])

```

接下来，我们就根据评论中的词语及其 TF-IDF 得分，构造一个 dataframe，并对该 dataframe 中一些无意义的部分进行剔除。

```

import pandas as pd

#构造 dataframe 字段名为词袋里的词，内容为每条评论中、各个词的 TF-IDF 得分
train=pd.DataFrame(data=weight, columns=words)

#去除词频为 1 的词语
for w in words:
    if w in once_words:
        train=train.drop([w], axis=1)

#更新词袋
words=list(train.columns)

#构造停用词列表
stopwords=[]
stopwordsfile=open("stop_words.txt","r",encoding="utf-8")
for word in stopwordsfile.readlines():
    #去除换行符
    word=word.strip('\n')
    stopwords.append(word)

#去除 dataframe 中的停用词
for w in words:
    if w in stopwords:
        train=train.drop([w], axis=1)

#更新词袋
words=list(train.columns)

```

此时我们对列表 `words` 中的词进行查看，其内容如下：


```
['10',  
'11',  
'12',  
'120g',  
'200g',  
'320g',  
'380g',  
'390g',  
'400g',  
'68',  
'900g',  
'一半',  
'一口',  
'一块',  
'一如既往',  
'一家',  
'一点',  
'一点点',  
'一般般',  
'三个',  
'下午茶',
```

我们发现，一些词如'10', '11', '12', '120g'等也是没有意义的，故我们也将进行剔除。

```
#去除 dataframe 中没有意义的词  
drop_words_list=[  
    '10',  
    '11',  
    '12',  
    '120g',  
    '200g',  
    '320g',  
    '380g',  
    '390g',  
    '400g',  
    '68',  
    '900g']  
train=train.drop(drop_words_list, axis=1)  
  
#更新词袋  
words=list(train.columns)
```

因为我们对 dataframe 中的一些词进行了剔除,这可能会导致有一些评论其所有词的 TF-IDF 得分总和=0 (即 dataframe 中的一行 TF-IDF 得分和为 0)。这时，我们认为该条评论是没有意义的，并将其剔除。

```
#如果一条评论中，其所有词的 TF-IDF 得分总和=0，我们认为该条评论是没有意义的，将其剔除
sum_r_tfidf=np.sum(np.array(train),axis=1)

zero_list=[]
for i in range(len(sum_r_tfidf)):
    if sum_r_tfidf[i]==0:
        zero_list.append(i)

train=train.drop(zero_list)
```

最终得到的 dataframe 内容如下，其大小为 289 rows × 262 columns:

No	~一半	~一口	~一块	~一如既往	~一家	~一点	~一点点	~一般般	~三个	~下午茶	~下单	~下次	~不上	~不值	~不到	~
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0.11606	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0.15147	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0.43162	0	0	0	0
13	0	0	0	0	0	0.19252	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0.15585	0	0	0	0	0	0	0.27571	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0.22836	0	0	0.18917	0	0	0	0
17	0	0	0	0	0	0.2386	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0.3501769	0	0	0	0	0.18158	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0.51662	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	0	0	0	0	0	0	0	0.32356	0	0	0	0	0	0	0	0

接着，我们将上面得到的 dataframe 作为数据集，使用 t-SNE 算法将其降为二维。因为每次使用 t-SNE 进行降维的结果是有所不同的，尽管我们设置的参数一样，所以我们在这里进行 10 次 t-SNE 降维，将其 Kullback-Leibler 离散度作为衡量指标，取 KL 离散度最小的一次作为降维的结果。

```

from sklearn.manifold import TSNE
import numpy as np

#进行 10 次，取 KL 离散度最小的
result_list=[]
min_kl=100
for i in range(10):
    #使用 t_sne 降成 2 维
    tsne=TSNE(n_components=2, perplexity=50)
    result=tsne.fit_transform(np.array(train))
    result_list.append(result)
    #计算 KL 离散度（越小越好）
    kl=tsne.kl_divergence_
    print(kl)
    if kl<min_kl:
        min_kl=kl
        best_id=i

#构造 dataframe
best_result=result_list[best_id]
tsne_train=pd.DataFrame({"feature1":list(best_result[:, 0]), "feature2":list(best_result[:, 1])})

```

得到的降维结果如下：

	feature1	feature2
0	-3.250170	-1.876212
1	-0.237814	0.170697
2	-0.742413	-3.986148
3	-0.709729	-1.282788
4	1.595189	2.856571
...
284	2.518790	-0.930225
285	1.710810	-0.181974
286	-2.201175	2.693723
287	1.831212	-1.681897
288	-1.630664	-2.935716

为了能在二维图形中显示该评论属于好评还是差评，我们根据每条评论中的评分（非常满意、满意、一般、差、非常差）构造一个颜色列表，其与 tf-idf 的 dataframe 逐行对应，其颜色所代表的含义为：深绿（非常满意）、绿（满意）、黄（一般）、橙（差）、红（非常差）。

```

#构造评分列表，其与 tf-idf 的 dataframe 逐行对应
label_list=[]
f=open("外卖评价_仅评分.txt","r",encoding="utf-8")
for score in f.readlines():
    score=re.sub('[^\u4e00-\u9fa5^a-z^A-Z^0-9]', "", score)
    label_list.append(score)

#剔除评论中所有词的 TF-IDF 得分总和为 0 的评论得分
t=0
for i in zero_list:
    index=i-t
    del label_list[index]
t=t+1

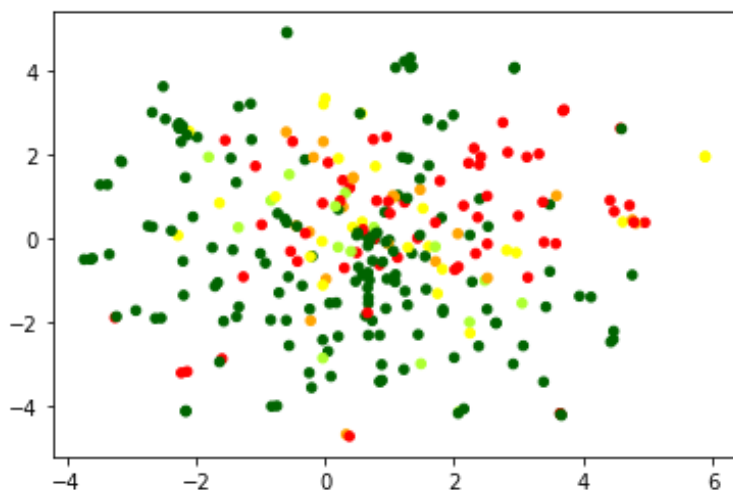
#设置颜色：深绿（非常满意）、绿（满意）、黄（一般）、橙（差）、红（非常差）
label_color=[]
for i in label_list:
    if i=="非常满意":
        label_color.append("DarkGreen")
    elif i=="满意":
        label_color.append("GreenYellow")
    elif i=="一般":
        label_color.append("Yellow")
    elif i=="差":
        label_color.append("Orange")
    elif i=="非常差":
        label_color.append("Red")

import matplotlib.pyplot as plt

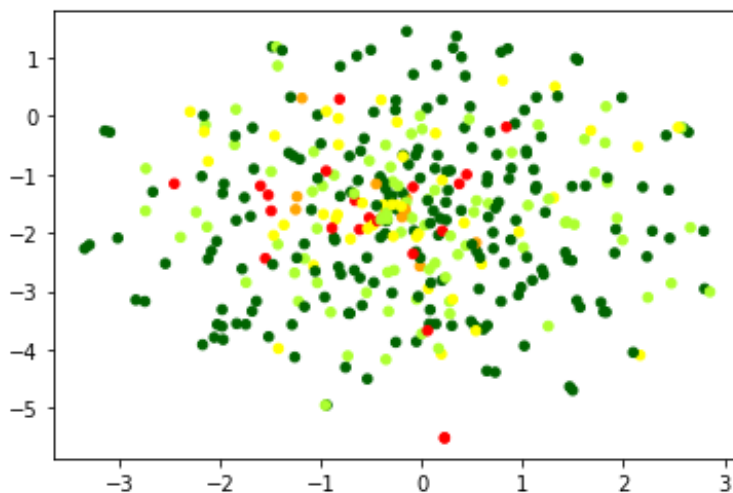
#将评论可视化，并将不同评分的评论用不同颜色进行标注
plt.scatter(best_result[:, 0], best_result[:, 1], 20, np.array(label_color))
plt.show()

```

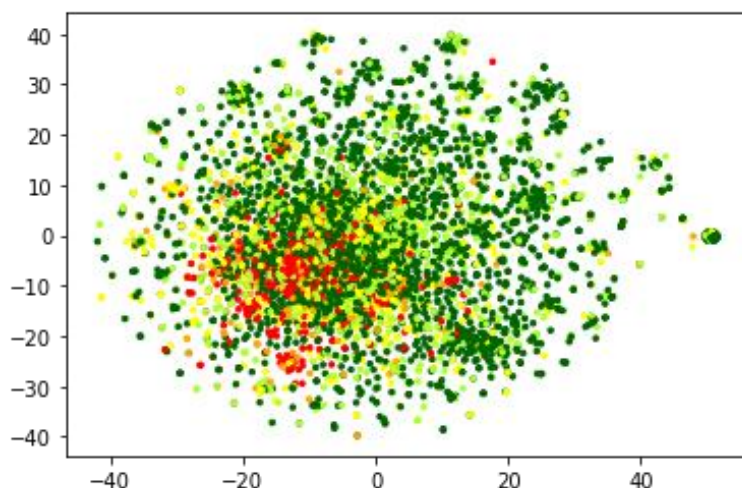
最终得到的 Airtest 爬取到的“外卖评价”的可视化结果如下：



对 Airtest 爬取到的“到店评价”进行向量化和可视化的核心代码和前面的是一样的，只是读入数据的那部分代码不同，其可视化的结果如下：



对“幸福西饼生日蛋糕门店美团评论数据集_1_5k.json”的评论进行向量化和可视化的核心代码和前面的也是一样的，也只是读入数据的那部分代码不同，其可视化的结果如下：



（5）对客户评价进行聚类分析，并采用二维平面对聚类结果进行可视化处理

对于客户评价聚类，我们可以直接对上面 t-SNE 降维后的结果用 K-means 算法进行聚类。对于聚类个数的选取，我们采用轮廓系数作为衡量指标，作出轮廓系数与聚类个数的关系曲线，取轮廓系数最大时的聚类个数。

下面是对 Airstest 爬取到的“外卖评价”进行聚类分析的代码实现：

```

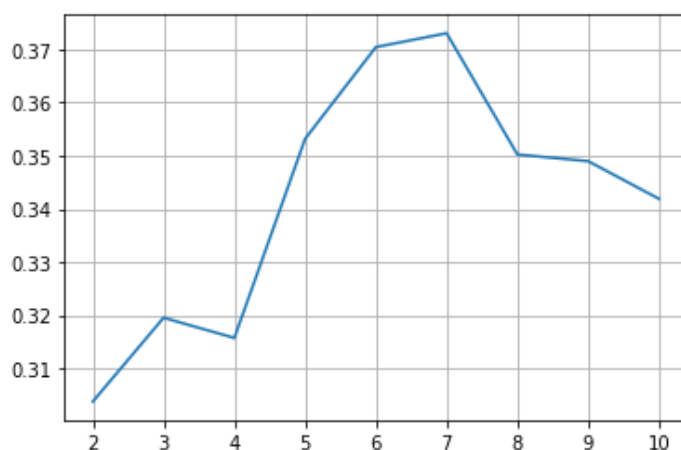
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

X=np.array(tsne_train)
S_list=[]
target_list=[]
#K-means 聚类(2-10)
for k in list(range(2,11)):
    #创建模型
    kmmodel = KMeans(n_clusters=k)
    #训练模型
    kmmodel = kmmodel.fit(tsne_train)
    #对原始数据进行标注
    target=kmmodel.predict(tsne_train)
    target_list.append(target)
    #使用轮廓系数评价模型的拟合度
    S_score=silhouette_score(tsne_train, target)
    S_list.append(S_score)

#对各个聚类结果的轮廓系数进行折线图绘制
plt.plot(list(range(2,11)),S_list)
plt.grid(True)
plt.show()

```

作出的轮廓系数与聚类个数的关系曲线如下：



之后我们根据轮廓系数选取出得分最高时的那个索引，以这时的结果作为聚类的结果。

```

#获取得分最高的索引
id=0
max_score=0
for s in S_list:
    if s>max_score:
        max_score_id=id
        max_score=s
    id=id+1

#选取得分评价最高的那个结果
target = target_list[max_score_id]

```

在得到聚类结果后，我们可以使用交叉表来查看各个簇中成员的数量，并用散点图对其进行展示。

```

#交叉表查看各个类别数据的数量
pd.crosstab(target,target)

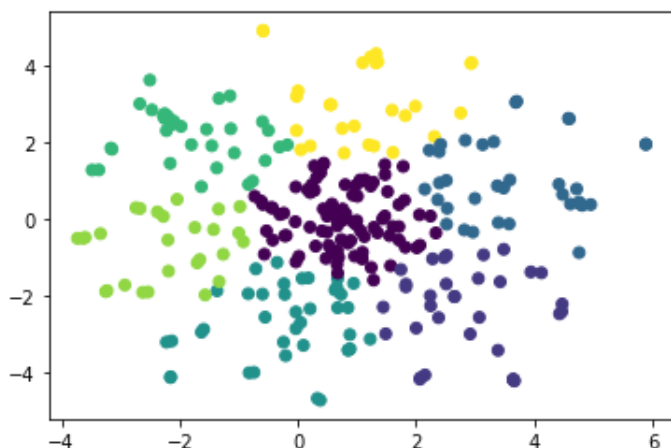
```

col_0	0	1	2	3	4	5	6
row_0							
0	92	0	0	0	0	0	0
1	0	34	0	0	0	0	0
2	0	0	35	0	0	0	0
3	0	0	0	39	0	0	0
4	0	0	0	0	35	0	0
5	0	0	0	0	0	27	0
6	0	0	0	0	0	0	27

```

#查看聚类的分布情况
plt.scatter(tsne_train["feature1"],tsne_train["feature2"],c=target)

```

最后，我们在聚类得到的每一类中，选取 TF-IDF 指标总和最大的那 3 个词作为该类的主题。

```
#簇的个数
num_of_cluster=max_score_id+2

#保存各个簇中成员的 id
cluster_dictionary={}
for i in range(num_of_cluster):
    cluster_dictionary[i]=[]

id=0
for i in target:
    cluster_dictionary[i].append(id)
    id=id+1

#在聚类得到的每一类中，取 TF-IDF 指标总和最大的那 3 个词作为该类的主题
for i in range(num_of_cluster):
    cluster_tfidf=train.iloc[cluster_dictionary[i]]
    sum_c_tfidf=np.sum(np.array(cluster_tfidf),axis=0)
    clister=pd.Series(sum_c_tfidf,words)
    clister=clister.sort_values(ascending=False)
    theme=clister.index.to_list()
    print("第"+str(i+1)+"个类的主题: "+theme[0]+" "+theme[1]+" "+theme[2])
```

得到的结果如下：

第 1 个类的主题：蛋糕 好吃 幸福

第 2 个类的主题：满意 不错 包装

第 3 个类的主题：电话 商家 难吃

第 4 个类的主题：好吃 美味 蛋糕

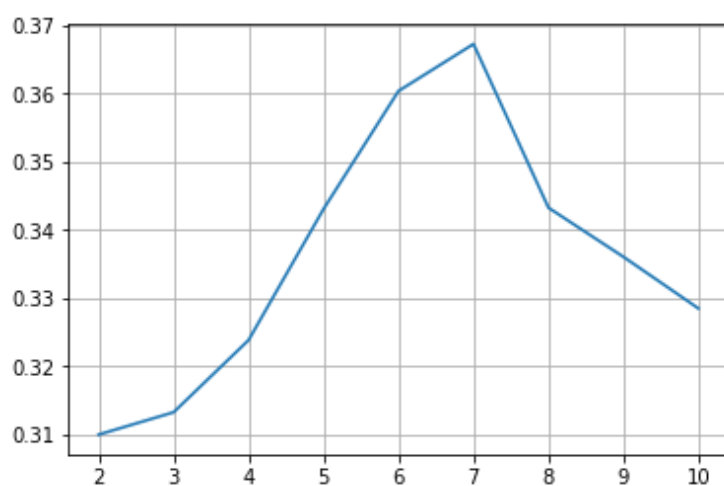
第 5 个类的主题：榴莲 千层 英寸

第 6 个类的主题：生日蛋糕 感谢 莓莓

第 7 个类的主题：味道 全心全意 奶油

对 Airstest 爬取到的“到店评价”进行聚类分析和聚类结果可视化的核心代码和前面的是一样的，只是读入数据的那部分代码不同。

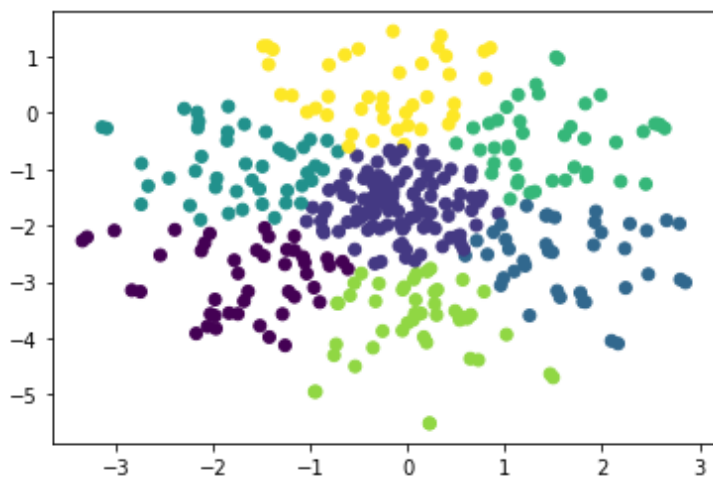
其轮廓系数与聚类个数的关系曲线如下：



其聚类结果的交叉表如下：

col_0	0	1	2	3	4	5	6
row_0							
0	45	0	0	0	0	0	0
1	0	111	0	0	0	0	0
2	0	0	40	0	0	0	0
3	0	0	0	45	0	0	0
4	0	0	0	0	42	0	0
5	0	0	0	0	0	46	0
6	0	0	0	0	0	0	41

其聚类结果的二维散点图如下：



其每个类的主题如下：

第 1 个类的主题：幸福 西饼 蛋糕

第 2 个类的主题：蛋糕 好吃 不错

第 3 个类的主题：好吃 蛋糕 喜欢

第 4 个类的主题：味道 太甜 蛋糕

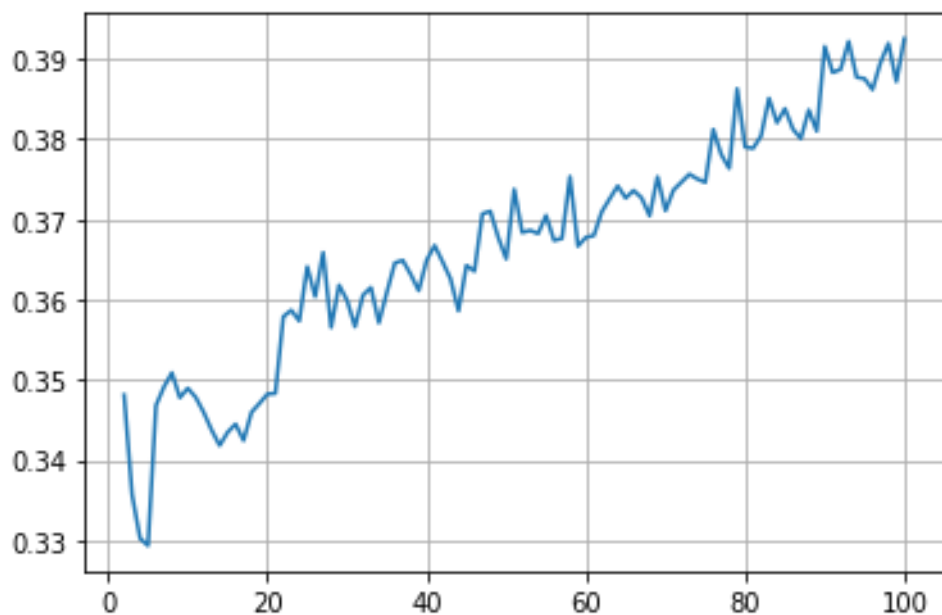
第 5 个类的主题：口味 很快 送货

第 6 个类的主题：榴莲 千层 芒果

第 7 个类的主题：味道 不错 准时

对“幸福西饼生日蛋糕门店美团评论数据集_1_5k.json”的评论进行聚类分析和聚类结果可视化的核心代码和前面的是一样的，只是读入数据的那部分代码不同。

其轮廓系数与聚类个数的关系曲线如下：

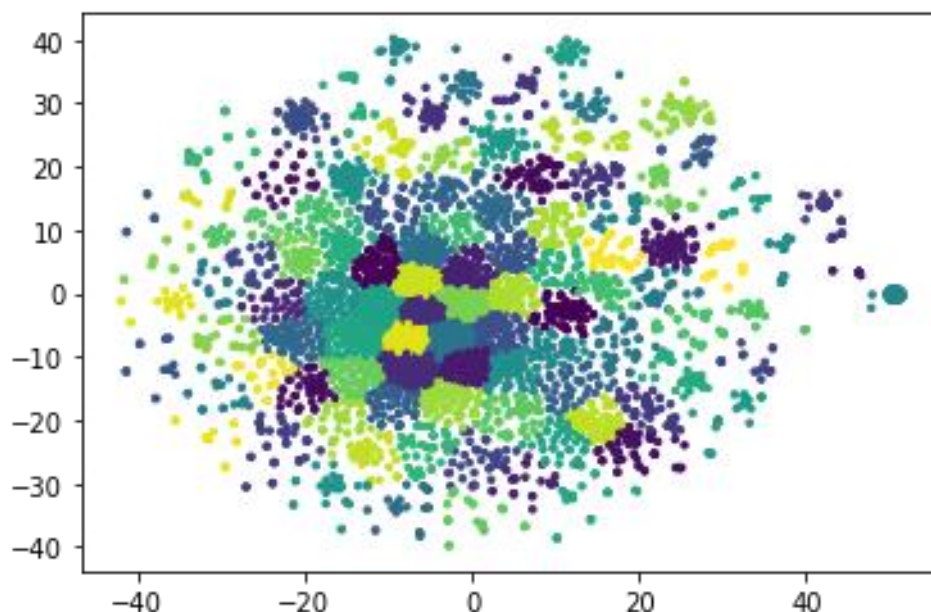


虽然其轮廓系数总体是随着聚类个数的增加而增加,但有时聚类个数过多并没有什么实际意义。因为此时数据规模为 5000 左右,所以我们就选取聚类个数为 2-100 时轮廓系数最大的那个结果,即聚 100 个类。

其聚类结果的交叉表如下:

col_0	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94	95	96	97	98	99
row_0																					
0	69	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	74	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	58	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	45	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	69	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...
95	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	27	0	0	0	0
96	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	17	0	0	0
97	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	28	0	0
98	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	45	0
99	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	33

其聚类结果的二维散点图如下:



其每个类的主题如下：

- 第 1 个类的主题：口感 不错 四个
- 第 2 个类的主题：榴莲 下午茶 公司
- 第 3 个类的主题：推荐 值得 不错
- 第 4 个类的主题：电话 不通 家门口
- 第 5 个类的主题：幸福 西饼 蛋糕
- 第 6 个类的主题：好吃 榴莲 太贵
- 第 7 个类的主题：喜欢 味道 不错
- 第 8 个类的主题：芒果 蛋糕 榴莲
- 第 9 个类的主题：蛋糕 好吃 齐全
- 第 10 个类的主题：感觉 味道 包装
- 第 11 个类的主题：回头客 老客户 幸福
- 第 12 个类的主题：好像 不想 聚会
- 第 13 个类的主题：蛋糕 过生日 喜欢
- 第 14 个类的主题：好吃 时间 美团
- 第 15 个类的主题：四种 口味 味道
- 第 16 个类的主题：很赞 味道 这定
- 第 17 个类的主题：送货 准时 不错

第 18 个类的主题：幸福 西饼 每次
第 19 个类的主题：不错 好吃 蛋糕
第 20 个类的主题：两个 吃不完 送人
第 21 个类的主题：好看 好吃 优惠活动
第 22 个类的主题：越来越 蛋糕 西饼
第 23 个类的主题：还行 想象 说好
第 24 个类的主题：挺好吃 不错 蛋糕
第 25 个类的主题：朋友 生日 蛋糕
第 26 个类的主题：活动 划算 涨价
第 27 个类的主题：服务 小哥 妈妈
第 28 个类的主题：习惯 可口 不错
第 29 个类的主题：支持 一如既往 很多年
第 30 个类的主题：炒鸡 继往 一如
第 31 个类的主题：蛋糕 依然 现做
第 32 个类的主题：西饼 幸福 蛋糕
第 33 个类的主题：满意 好吃 不腻
第 34 个类的主题：蜡烛 生日 生日蛋糕
第 35 个类的主题：口味 服务 准时
第 36 个类的主题：男朋友 送货员 态度
第 37 个类的主题：好吃 蛋糕 真心
第 38 个类的主题：棒棒 好吃 味道
第 39 个类的主题：蛋糕 生日 不错
第 40 个类的主题：朋友 家里人 很足
第 41 个类的主题：小朋友 喜欢 他家
第 42 个类的主题：这家 蛋糕 每次
第 43 个类的主题：芝士 半熟 送过来
第 44 个类的主题：可惜 麻烦 卡片
第 45 个类的主题：物美价廉 回购 蛋糕店
第 46 个类的主题：好吃 不到 口味

第 47 个类的主题：新鲜 水果 好吃
第 48 个类的主题：顾客 全家 好吃
第 49 个类的主题：服务 下次 好吃
第 50 个类的主题：芒果 榴莲 好吃
第 51 个类的主题：千层 榴莲 芒果
第 52 个类的主题：草莓 送到 时间
第 53 个类的主题：太甜 一块 不错
第 54 个类的主题：购买 不错 值得
第 55 个类的主题：四重奏 首选 蛋糕
第 56 个类的主题：好评 一如既往 不错
第 57 个类的主题：味道 不错 新鲜
第 58 个类的主题：送货 送到 好吃
第 59 个类的主题：幸福 西饼 蛋糕
第 60 个类的主题：客服 打电话 蛋糕
第 61 个类的主题：粉丝 忠实 西饼
第 62 个类的主题：性价比 米苏 提拉
第 63 个类的主题：价格 公司 精致
第 64 个类的主题：便宜 信赖 好吃
第 65 个类的主题：下次 味道 真的
第 66 个类的主题：开心 漂亮 反正
第 67 个类的主题：好好 哈哈 赞赞
第 68 个类的主题：姐姐 即往 男票
第 69 个类的主题：速度 好吃 食材
第 70 个类的主题：还会 下次 光顾
第 71 个类的主题：优惠 价格 团购
第 72 个类的主题：美味 还来 下次
第 73 个类的主题：送达 准时 不错
第 74 个类的主题：一般般 榴莲 很浓
第 75 个类的主题：榴芒 双拼 蛋糕

第 76 个类的主题：服务周到 想到 味道
第 77 个类的主题：小时 蛋糕 承诺
第 78 个类的主题：挺不错 小孩 喜欢
第 79 个类的主题：分量 味道 价格
第 80 个类的主题：味道 不错 失望
第 81 个类的主题：太腻 准确 好不好
第 82 个类的主题：完美 祝福语 蜡烛
第 83 个类的主题：超级 好吃 第二次
第 84 个类的主题：一如既往 好吃 味道
第 85 个类的主题：不好 拿破仑 蛋糕
第 86 个类的主题：退款 小时 蛋糕
第 87 个类的主题：服务态度 送货上门 特别
第 88 个类的主题：不错 味道 款式
第 89 个类的主题：实惠 孩子 价格
第 90 个类的主题：幸福 西饼 蛋糕
第 91 个类的主题：难吃 一块 好食
第 92 个类的主题：好吃 榴莲 好看
第 93 个类的主题：几次 强烈推荐 买过
第 94 个类的主题：小贵 好几个 一年
第 95 个类的主题：蛋糕 送到 时间
第 96 个类的主题：还好 味道 购价
第 97 个类的主题：发票 感谢 ok
第 98 个类的主题：好腻 店里 客户
第 99 个类的主题：每次 生日 蛋糕
第 100 个类的主题：榴莲 喜欢 小孩子