文章编号:1671 - 6833(2017)05 - 0007 - 06

基于 Spark 的标准化 PCA 算法

董建华[1,2], 王国胤[1], 雍 熙[3], 史晓雨[1], 李庆亮[4]

(1. 中国科学院重庆绿色智能技术研究院 电子信息技术研究所,重庆 400714; 2. 中国科学院大学,北京 100049; 3. 水利部 水利信息中心,北京 100053; 4. 西昌卫星发射中心,海南 文昌 571300)

摘 要: 主成分分析(PCA)是数据挖掘中常用的数据降维技术,它将原来多个变量转化为少数的几个综合指标,介绍了主成分分析的原理、Spark 的分布式架构以及 Spark 的 MLlib 分布式矩阵 PCA 算法,通过改进设计提出了一种带有标准化处理的 SNPCA 算法,并在多核 CPU 环境下进行了测试验证,实验结果证明了该算法的有效性.

关键词: 主成分分析;Spark;分布式;标准化处理

中图分类号: TU316.4 **文献标志码:** A doi:10.13705/j.issn.1671 - 6833.2017.05.001

0 引言

主成分分析(principal component analysis, PCA)作为一种常用的多指标统计方法,是由 Karl parson[1] 1901 年提出的,它将原来多个变量转化为少数几个综合指标. 从数学的角度看,它是一种降维处理技术,能够最大限度地保留原始数据信息,对高维变量进行综合和简化,并客观地确定各个指标的权重,从而避免了降维过程中的主观随意性.

随着大数据量计算任务的需求,多核处理下的 PCA 并行计算逐渐成为数据挖掘中必不可少的分析方式,比较著名的方法是 Hadoop 的 Mahout[2] 和 Spark 的 MLlib 库[3] 实现的并行 PCA 算法. 近几年,以降低时间复杂度和通信复杂性为目的,在分布式平台上设计可扩展的 PCA 算法模型,成为多核处理下 PCA 算法研究的重点[4-5].

Apache Spark 于 2009 年诞生 AMPLab,其官方的定义为"Spark 是一个通用的大规模数据快速处理引擎"[6]. 2013 年推出的 Spark 0.8 在全面兼容 Hadoop[7](包括 Hadoop 的 HDFS 文件系统和存储数据库)的基础上,利用更多的内存处理机制大幅提高了系统性能,进而提高了在大数据环境下数据处理的实时性,同时保证了高容错性和高扩展性. 依靠 Scala 强有力的函数式编程、Actor 通信模式、闭包、容器、泛型,借助统一资源分配调度框架 Mesos 和 YARN[8],并融合了 MapReduce 和 Dryad[9],Spark 最后成了一个简洁、直观、灵活、高效的大数据分布式处理框架.

Hadoop/MapReduce 依靠 HDFS 来实现分布式环境下中间数据的交互,Spark 则通过共享的虚拟内存来实现中间数据的快速交换[10]. 相比而言,Spark 具有更好的扩展性,同时 Spark 需要更强的 CPU 以及苛刻的 I/O 等条件来支持它的运行环境. Spark 现有 PCA 存在的问题是:在 Spark 的 MLlib 组件中,分布式矩阵 PCA 算法的计算是直接对样本矩阵读入后转换为分布式矩阵来处理的,并没有对样本矩阵进行标准化操作. 笔者提出一种带标准化处理的 PCA 算法 SNPCA(Spark's normalized principal component analysis)来实现对 Spark 分布式矩阵的标准化操作,该算法借助 Spark 的分布式计算平台,提高了大数据环境下主成分分析的实时处理能力.

收稿日期:2017 - 04 - 10;**修订日期:**2017 - 07 - 11

基金项目:国家自然科学基金青年基金资助项目(61602434);三峡库区水生态环境感知系统及平台业务化运行(2014ZX07104-006);重庆市基础科学与前沿研究技术重点专项(cstc2015jcyjB0244);中国科学院青年创新促进会资助项目(No. 2017393)

作者简介:董建华(1987—),男,中国科学院博士研究生,主要从事云计算技术、数据挖掘、人工智能等研究,E-mail:928991386@qq.com.

1 主成分分析方法

主成分分析的目的是减少数据集的维数,把原有高维的数据变量集重新组合为相互无关的综合变量集,这些不相关的综合变量的数量小于或等于原始变量的数量,综合变量所表征"信息量"的大小用方差来衡量. 通常情况下,主成分分析算法可以看作是一种揭露数据的内部结构、从而更好地解释其综合变量之间关系的方法,主要计算步骤如下[11].

(1)建立原始变量观测数据构成的矩阵.

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}. \tag{1}$$

(2)对原始变量数据构成的矩阵进行标准化处理来消除不同量纲的影响.

$$x_{ij}^{*} = \frac{x_{ij} - \bar{x}_j}{\sqrt{\mathrm{var}(x_j)}}, \tag{2}$$

式中:$\bar{x}_j$ 为第 j 个列向量的均值,即

$$\bar{x}_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}.$$

$\mathrm{var}(x_j)$ 为第 j 个列向量的方差,即

$$\mathrm{var}(x_j) = \frac{1}{n-1}\sum_{1}^{n} (x_{ij} - \bar{x}_j)^2.$$

(3)计算标准化数据的相关系数矩阵,求相关系数矩阵 $\boldsymbol{R}$ 的特征值$(\lambda_1,\lambda_2,\cdots,\lambda_p)$ 和相应的特征向量$\boldsymbol{a}_i = (a_{i1},a_{i2},\cdots,a_{ip})$,$i = 1,2,\cdots,p$,用雅克比方法、SVD 分解法[12]等.

(4)选择重要的主成分分量. 主成分分量的方差贡献率 c_i 定义如下:

$$c_i = \frac{\lambda_i}{\sum_{i=1}^{p}\lambda_i}. \tag{3}$$

第 i 个主成分分量 F_i 的累积贡献率为:

$$\delta_i = c_1 + c_2 + \cdots + c_i. \tag{4}$$

通常选择使累积贡献率达到 85% 的主成分变量,计算出主成分变量的表达式为:

$$\begin{cases} F_1 = a_{11}x_1 + a_{12}x_2 + \cdots + a_{1p}x_p \\ F_2 = a_{21}x_1 + a_{22}x_2 + \cdots + a_{2p}x_p \\ \vdots \\ F_p = a_{p1}x_1 + a_{p2}x_2 + \cdots + a_{pp}x_p \end{cases}, \tag{5}$$

式中:x_1、x_2、…、x_p 表示变量矩阵经过标准化处理的值. 主成分变化后新的主成分变量彼此不相关,且方差依次递减.

2 Spark Mlib 库中的分布式矩阵 PCA 算法

本节将以 Spark 中 RowMatrix 类型的分布式矩阵为例来研究 MLlib 中的分布式 PCA 算法,Spark 中,其它两个分布式矩阵类型 indexedRowMatrix、CoordinateMatrix 与之类似,在此从略. RowMatrix 中求解 PCA 的函数方法为:

computePrincipalComponents(k: Int): Matrix[6].

其具体过程是通过先求 RowMatrix 类型矩阵(标识为矩阵 $\boldsymbol{A}$)的格拉姆矩阵,进而求得矩阵 $\boldsymbol{A}$ 的协方差矩阵,再使用 ScalaNLp 的 SVD 方法求解特征值、特征向量,得到分布式矩阵 $\boldsymbol{A}$ 的主成分分量,其算法流程如图 1 所示.

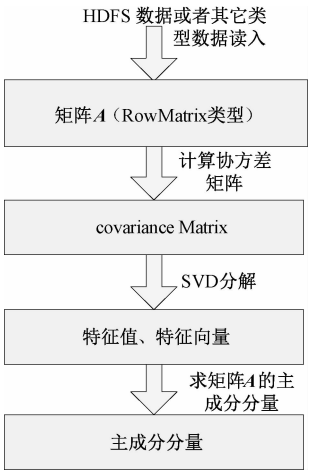

图 1 Spark 的 MLlib 组件中 PCA 算法流程

Fig.1 Flow chart of PCA from Spark's MLlib component

在 Spark MLlib 库中的分布式矩阵 PCA 算法中,通过对矩阵 $\boldsymbol{A}$ 进行不断迭代来实现协方差矩阵求解,其余部分则是串行算法(在提交的计算节点上串行处理完成). 协方差矩阵的求解是直接对矩阵 $\boldsymbol{A}$ 进行的操作,通过将读入的数据转换为 RowMatrix 类型矩阵来实现,并没有对样本矩阵进行标准化操作,鉴于此,我们将引入 SNPCA 算法来实现对 Spark 分布式矩阵的标准化操作.

3 基于 Spark 的标准 PCA 算法—SNPCA

在数据预处理阶段,通常将数据进行标准化(normalization)处理后再进行数据分析. 数据的标准化是指将数据按比例缩放,调节不同的尺度测量值到一个公共的范围,使数据值落入一个小的特定区间(将不同范围的数据统一到同一个区

间参考系下进行比较才有意义[13]). 数据的标准化在处理某些指标的比较和评价中也经常涉及到,通过去除数据的范围和单位量纲的限制,以便不同量级或单位的指标能够进行比较和加权,在许多情况下使用标准化后的变量进行主成分分析会显著地改善分析效果. 对数据进行标准化处理的常见方法有"按小数定标标准化"、"Z-score 标准化"和"min—max 标准化"等[14].

文献[15]的研究表明,对于样本矩阵($n \times p$ 型,p 远小于 n),计算 PCA 的最好方法是先计算出所有的统计量(时间复杂度为 $O(p^2n)$),再对相关系数矩阵或协方差矩阵($p \times p$ 型)进行 SVD 分解[16](时间复杂度为 $O(p^3)$). 在第 1 章求解 PCA 的步骤中,依照公式(2)中数据标准化进行求解需要对样本数据读入的矩阵进行 3 次扫描,即求出列向量的均值和方差值再代入公式(2)进行标准化. 在大数据量的情况下,事物的观察数据量动辄上万,这种操作显然是不可接受的,要尽量提高数据的扫描效率,才能适应快速分析大数据任务的需要. 鉴于此,我们以 Spark 的 MLlib 库和开源的数值计算库 ScalaNLP 为基础,设计改进了基于 Spark 的标准化 PCA 算法—SNPCA 算法来实现对大数据样本矩阵进行标准化的 PCA 操作,算法的的主要步骤如图 2 所示.

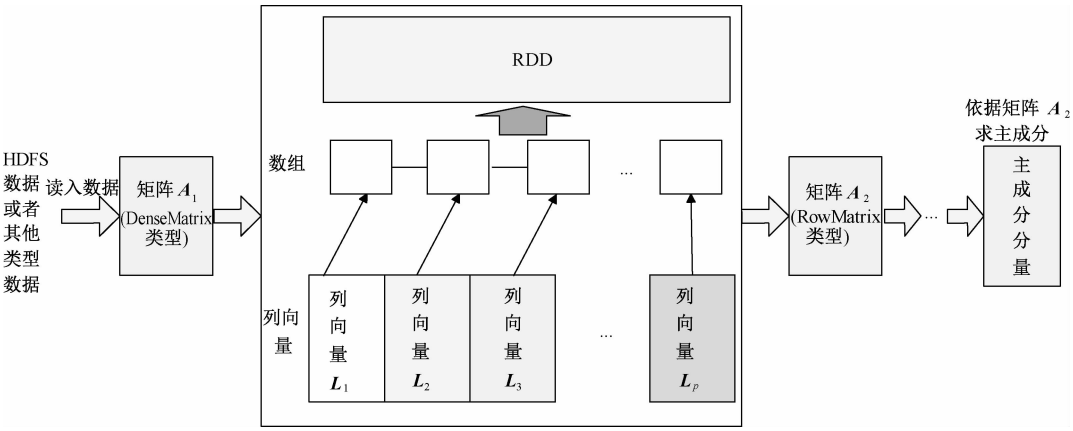

图 2　改进的 SNPCA 算法流程图

Fig. 2　Flow chart of improved SNPCA algorithm

在 SNPCA 算法中,读入的数据将转换为 DenseMatrix 类型的矩阵 $\boldsymbol{A}_1$,矩阵 $\boldsymbol{A}_1$ 的各个列向量 $\boldsymbol{L}_1$, $\boldsymbol{L}_2$, …, $\boldsymbol{L}_p$ 链接在一个数组上从而构造成"向量数组"结构,再放入 RDD 中对每个列向量并行地求解公式(2)所表征的列向量均值、方差等统计量,进而对每一个列向量进行标准化处理,这些存在于 RDD 中的列向量将会被重构成 RowMatrix 类型的矩阵 $\boldsymbol{A}_2$,再依照图 1 的流程求解相关系数矩阵、特征值、特征向量,最后根据这些统计参数和各个列向量 $\boldsymbol{L}_1$, $\boldsymbol{L}_2$, …, $\boldsymbol{L}_p$ 求出主成分分量. 下面列出算法的具体步骤及其所需时间.

(1)从 Spark 的 nameNode 节点驱动程序读取数据文件/数据库创建 DenseMatrix 类型的矩阵 $\boldsymbol{A}_1$. $\boldsymbol{A}_1$ 存在于 Spark 驱动程序所在的 nameNode 节点上.

(2)把矩阵 $\boldsymbol{A}_1$ 的列向量 $\boldsymbol{L}_i$($i = 1, 2, \cdots, p$)"挂载"到长度为 p 的数组上,使用该数组转化为 Spark 的 RDD(可以通过 Sparkcontext 类的 def parallelize[T](seq: Seq[T], numSlices: Int): RDD[T]方法),借助该 RDD 来并行计算列向量的均值、方差,进而对每一个列向量进行标准化处理. 假设资源池可以提供 q 个 core,在 Spark 并行环境下,每个 core 对矩阵中每个元素进行运算平均时间消耗为 T_i. 则该步骤整体需要的计算时间为 $3(pn/q)T_i$.

(3)计算列向量转换成的 RowMatrix 矩阵 $\boldsymbol{A}_2$ 的格拉姆矩阵,进而计算 $\boldsymbol{A}_2$ 的相关系数矩阵. 对于矩阵 $\boldsymbol{A}$,求其格拉姆矩阵的公式为 $\boldsymbol{Q} = \boldsymbol{A}^{\mathrm{T}} * \boldsymbol{A}$[17]. 由于 Spark 的 MLlib 提供的 API 是以并行迭代的方式计算格拉姆矩阵,假设 $\boldsymbol{A}_2$ 被分成 d 个分区,nameNode 和 dataNode 可以提供 q 个 core,格拉姆矩阵被并行处理需要的时间为 $(p^2n/q)\mathrm{T}_i$. 注意:Spark 在逻辑上是基于分区数并行的,实际的并行是基于核数的,并行度为 $\min\{d, q\}$,通常 d 为核数 q 的 2.3 倍,所以并行度就是核数 q. 由格拉姆矩阵计算相关系数矩阵(表示为 ρ)[18],设 Spark 串行运算条件下,每个 core 对矩阵中每个元素进行运算的平均时间消耗为 T_c,则需要时间为 $(p^2)T_c$.

(4)在 Spark 的 NameNode 上对相关系数矩阵进行 SVD 分解. 依据 SVD 公式 $[U,S,V]=\mathrm{SVD}(\rho)$,设 SVD 分解后由特征值组成的矩阵为 $\boldsymbol{S}$,由特征向量组成的矩阵为 $\boldsymbol{U}$[19],该 SVD 分解需要时间 $(p^3)T_c$.

(5)求主成分. 根据公式(5)求主成分分量 $\boldsymbol{F}=\boldsymbol{X}*\boldsymbol{A}$,则得到主成分分量构成的矩阵 $\boldsymbol{F}$ 为 $\boldsymbol{F}=\boldsymbol{A}_2*\boldsymbol{U}$(当然也可以根据特征值的贡献率进行筛选主成分分量),该过程所需时间为 $(p^2n/q)T_i$.

在步骤(1)~(5)中,算法的并行是在 $\boldsymbol{A}_1$ 求列和、列向量方差、标准化,以及由矩阵 $\boldsymbol{A}_2$ 求解格拉姆矩阵和最后求解主成分分量这3个过程中. 由于 p 远小于 n,所以在整个计算步骤中,算法的整体复杂度在性能方面的瓶颈还取决于算法的并行阶段. 下面将对整体算法的效率进行分析. 一般地,程序加速比的定义[20]如下:

$$S_N=\frac{T_1}{T_N}, \tag{6}$$

式中:S_N为加速比;T_1 为单处理器下程序运行的时间;T_N为有 N 个处理器并行运行时的时间. 由于 Spark 的运行调度是以核为粒度的,所以在 Spark 运行环境下,对应的加速比公式中 T_1 是串行算法运行时间消耗,T_N是 N 核时并行运行所消耗的时间. SNPCA 算法的时间消耗是前面(1)~(5)过程中时间消耗的总和,所以核数为 q 时 SNPCA 总时间消耗为:

$$T_q=3(pn/q)T_i+(p^2n/q)T_i+(p^2)T_c+(p^3)T_c+(p^2n/q)T_i+k=2(p^2n/q)T_i+3(pn/q)T_i+(p^2)T_c+(p^3)T_c+k.$$

当 q 为1时,SNPCA 串行运行的时间消耗为:

$$T_1=2(p^2n)T_i+3(pn)T_i+(p^2)T_c+(p^3)T_c+t.$$

所以核数为 q 时并行加速比为:

$$S_q=(2(p^2n)T_i+3(pn)T_i+(p^2)T_c+(p^3)T_c+t)/(2(p^2n/q)T_i+3(pn/q)T_i+(p^2)T_c+(p^3)T_c+k). \tag{7}$$

因此,当核数 q 比较大时,算法的并行优势会比较明显.

4 仿真实验

4.1 仿真实验环境及数据集

4.1.1 SNPCA 算法时间性能实验

在 Spark 分布式运行环境下,Spark 的分区数一般为总核数的2~3倍,读取的数据文件一般来源于 HDFS,在读取 HDFS 数据文件的时候每一个 Block(128 M)为生成的 RDD 的一个分区. 考虑到虚拟机集群中通信和调度等因素影响,实验效果会很差,为验证算法的并行效率,我们选择在一个 DELL 服务器(4 核)环境下做仿真实验,模拟的随机数据构成 240 000×200 的样本矩阵(大小有 286 MB),通过控制不同的核数来看作业处理的仿真效果. 另外,为了增强对比试验,使用 ScalaNLP 的线性代数包构建的 PCA 串行算法(Breeze PCA,BPCA)和 Spark1.3.0 本身提供的 PCA 算法(Spark's PCA,SPCA,该算法没有进行数据的标准化处理)一并进行试验对比. 在分配核数为 q 的时候,算法 SPCA 的时间消耗为 $(p^2n/q)T_i+O(p^2)T_c+(p^3)T_c+k$. 由于 p 远小于 n,所以其加速比为:

$$S_q=\frac{T_1}{T_q}\approx q. \tag{8}$$

DELL 服务器配置如下:

4 Cores

Inte(R) Core(TM) i7-4770

RAM: 16 G

OS: win8, 64bit

frequency: 3.4 GhZ

Dise size: 250 GB

Shared memory:2 559 MB

SNPCA、BPCA、SPCA 算法在上述仿真实验环境下的运行时间对比如图3所示.

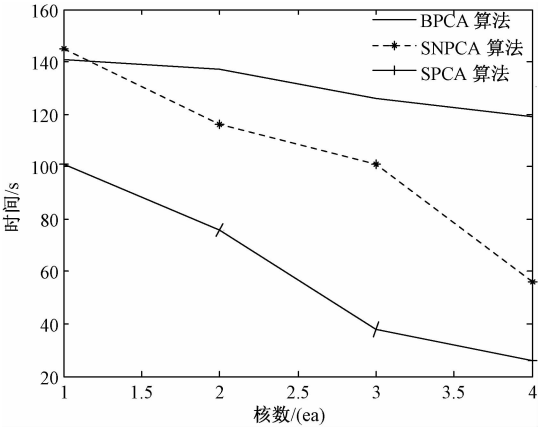

图3 SNPCA 等3种算法的运行时间对比

Fig. 3 Time consumption of the three Algorithms

4.1.2 SNPCA 算法与 SPCA 算法内存消耗量比较

一般地,分析某一程序在 Spark 运行环境下的空间消耗是比较困难的,因为程序的运行受分布式集群的运行环境、资源共享、JVM GC 效率、SWAP 分区等因素的影响. 通常 Spark 集群中 Driver 所在的节点会具有较高的系统性能,而

SNPCA 算法在 Driver 端程序运行的空间复杂度是一个多项式,下面将针对 SNPCA 算法在 Spark 集群的单个计算节点上的内存消耗进行试验,并与 SPCA 算法进行比较. 可以通过获取在操作系统上运行产生的 RES (resident memory usage)与 SHR (shared memory) 之差的平均值来计算 SNPCA 算法所消耗的内存. 仿真数据为 200 维、条数依次为 [1,2,3,4,5,6,7,8,9,10] 万条的 10 个数据集,大小依次为 [11.9, 23.7, 35.9, 47.7, 59.7, 71.4, 83.4, 95.1, 108.1, 119.2, 131.1] MB. 在 Spark 运行环境下,SPCA 算法和 SNPCA 算法在处理不同大小数据集时单个计算节点上所消耗的内存量对比情况如图 4 所示.

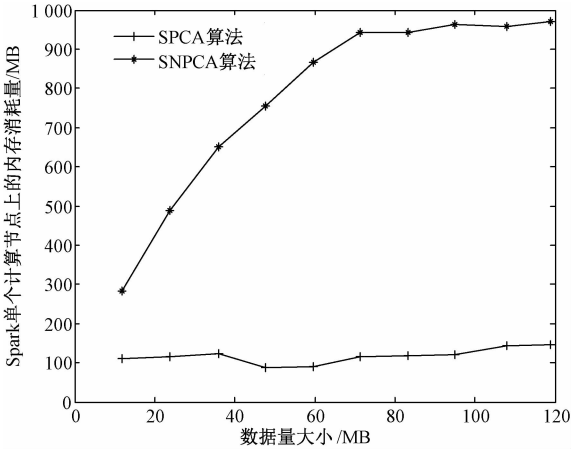

图 4　SPCA 算法和 SNPCA 算法在集群的单个工作节点上内存消耗量对比

Fig. 4　Memory consumption comparison between SPCA and SNPCA on single worker node

4.2　仿真实验结果及分析

在仿真实验中,BPCA 算法是串行算法,SPCA 算法是 Spark 提供的没有进行数据标准化的 PCA 算法. 从图 3 我们发现,Spark 1.3.0 提供的 SPCA 算法符合公式(8)的加速比,且基于 Spark 编程模型的 SNPCA 算法随着核数的增加也实现了快速处理的目的. 算法 BPCA 对多核表现并不敏感,因为它本身就是一个串行算法. 另外,可以看出 SNPCA 算法的性能介于 BPCA 算法和 SPCA 算法之间,SNPCA 算法的性能通常比 BPCA 算法好,除非在单核的时候,因为 Spark 的分布式机制决定了即使在单核的时候仍然对数据进行了分区,在不同分区上的迭代运算产生不可避免的交互运算、各个分区与资源管理器等的通信引起时间消耗. 总体来看,我们仍然可以推断出,SNPCA 算法不仅实现了带标准化的 PCA 功能,并且表现出不错的算法性能,适用于大型数据集的 PCA 运算.

从图 4 容易看出,SNPCA 算法在 Spark 集群的单个节点上的内存消耗量要远大于 SPCA 算法,这是因为 SNPCA 算法涉及了几个 collect 的操作,消耗了因分布式数据传递引起的大量内存开销,但是考虑到随着数据量上升开销幅度上升得不快,其算法仍在一定程度上提高了大数据下主成分分析的处理能力.

5　结论

提出了在 Spark 下具有数据标准化处理的 SNPCA 算法,通过“空间换时间”,实现了在时间上的节约,为 Spark 运行环境下进行大数据的 PCA 处理提供了解决途径. 另外,如何继续改进算法使得在分布式 RDD 中直接进行标准化处理(而不是通过 collect 机制)来进一步提高效率,这一点还有待于进一步研究.

参考文献:

[1] KARL P. On lines and planes of closest fit to systems of points in space[J]. Philosophical magazine, 1901, 2(6): 559 – 572.

[2] SEAN O, ROBIN A, TED D, et al. Mahout in action [M]. Vivgimia: Manning Publications, 2011.

[3] XIANGRUI M, JOSEPH B, BURAK Y, et al. Mllib: machine learning in apache spark[J]. JMLR, 2016, 17(34): 1 – 7.

[4] TAREK E, MAYSAM Y, ASHRAF A, et al. SPCA: scalable principal component analysis for big data on distributed platforms [C]//Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM, 2015: 79 – 91.

[5] SIDDHARTH MALHOTRA B. SPCA: scalable principle component analysis for big data[C]//Qatar Foundation Annual Research Conference. 2014 (1): ITSP0517.

[6] SPARK. APA CHE Spark[EB/OL]. [2016-06-11]. http://spark.apache.org/.

[7] 蔡金收,陈铁军,郭丽. 一种基于投票极限学习机的人脸识别混合算[J]. 郑州大学学报(工学版), 2016, 37(2): 37 – 41.

[8] BENJAMIN H, ANDY K, MATEI Z, et al. Mesos: a platform for fine-grained resource sharing in the data center[C]. NSDI. 2011(11): 22 – 22.

[9] MICHAEL I, MIHAI B, YUAN Y, et al. Dryad: Distributed data-parallel programs from sequential building blocks[C]. ACM SIGOPS Operating Systems Review. ACM, 2007, 41(3): 59 – 72.

[10] TAREK E, MOHAMED H. Analysis of PCA algo-

rithms in distributed environments[R]. Qatar foundation: Qatar computing research instistue, 2015.
[11] 江政,周勇.基于FPGA的电能质量监测装置设计[J]. 郑州大学学报(工学版), 2016, 37(2): 29-33.
[12] 王敏, 周树道, 叶松. 基于小波变换方向信息的奇异值图像去噪研究[J]. 郑州大学学报(工学版), 2012, 33(3): 121-124.
[13] 马立平. 现代统计分析方法的学与用(三):统计数据标准化——无量纲化方法[J]. 北京统计,2000(3): 34-35.
[14] CHAOSIMPLE. [EB/OL]. [2016-04-19]. http://www.cnblogs.com/chaosimple/archive/2013/07/31/3227271.
[15] CARLOS O. Statistical model computation with UDFs [J]. Knowledge and data engineering, 2010, 22(12): 1752-1765.
[16] CARLOS O, NAVEEN M, CARLOS G-A, et al. Fast PCA computation in a DBMS with aggregate UDFs and LAPACK[C]. Proceedings of the 21st ACM international conference on Information and knowledge management. ACM, 2012: 2219-2223.
[17] GRAMIAN M. http://en.wikipedia.org/wiki/Gramian_matrix. Accessed July 2016
[18] CARLOS O, NAVEEN M, CARLOS G-A. PCA for large data sets with parallel data summarization[J]. Distributed and parallel databases, 2014, 32(3): 377-403.
[19] JAN J G. On the relationships between SVD, KLT and PCA[J]. Pattern recognition, 1981, 14(1): 375-381.
[20] JACK J D, IAIN S D, DANNY C S, et al. Numerical linear algebra for high-performance computers[M]. Siam, 1998.

Normalized PCA Algorithm Based on Spark

DONG Jianhua[1,2], WANG Guoyin[1], YONG Xi[3], SHI Xiaoyu[1], LI Qingliang[4]

(1. Institute of Electronic Information & Technology, Chongqing Institutes of Green and Intelligent Technology, Chinese Academy of Sciences, Chongqing 400714, China; 2. University of Chinese Academy of Sciences, Beijing 100049, China; 3. Water Information Center, Ministry of Water Resources, Beijing 100053, China; 4. Xichang Satellite Launch Center, Wenchang 571300, China)

Abstract: Principal Component Analysis (PCA) is a well known model for dimensionality reduction in data mining, it transforms the original variables into a few comprehensive indices. In this paper, we study the principle of PCA, the distributed architecture of Spark and PCA algorithm of distributed matrix from spark's MLlib, then improved the design and present a new algorithm named SNPCA (Spark's Normalized Principal Component Analysis), this SNPCA algorithm computes principal components together with data normalization process. We carried out benchmarking on multicore CPUs and the results demonstrate the effectiveness of SNPCA.

Key words: PCA; Spark; distributed; normalization