文章编号:1671 - 6833(2016)05 - 0047 - 10

基于 Spark 的并行极速神经网络

邓万宇, 李 力, 牛慧娟

(西安邮电大学 计算机学院,陕西 西安 710121)

摘 要: 随着数据规模的快速膨胀,基于单机的串行神经网络结构面临着巨大的计算挑战,难以满足现实应用中的扩展需求. 在极速学习机(extreme learning machine,ELM)基础上,基于 Spark 并行框架提出一种并行的极速神经网络学习方法,以 Spark 平台特有的 RDD 高效数据集管理机制对其进行封装,并将大规模数据中的高复杂度矩阵计算进行并行化,实现 ELM 加速求解,仅需一组 Map 和 Reduce 操作即可完成算法的训练. 在大量真实数据集上的实验结果表明,基于 Spark 的并行 ELM 算法相较于串行 ELM 获得了显著的性能提升.

关键词: 极速学习机;神经网络;并行化 ELM 算法;Spark

中图分类号: TP389.1 **文献标志码**: A doi:10.3969/j.issn.1671 - 6833.2016.05.010

0 引言

极速神经网络(extreme learning machine, ELM)已被广泛应用于数据挖掘、模式识别等众多领域[1]. 相较于应用更为广泛的单隐藏层前馈 BP 神经网络,ELM 具有隐藏层输入权及阈值可随机生成,有效克服了 BP 迭代次数过于频繁,计算量、内存开销及耦合性过大等问题,仅需要一次学习即可完成训练的优势. 更重要的是,传统基于梯度下降的 BP 神经网络算法,以反向传播误差的方式对参变量进行迭代调整和网络训练,导致计算过程难以并行化,无法集成和发挥分布式计算平台的并行计算优势. 因此,人们正努力构建各种更为高效的并行化计算框架或平台,通过汇集计算机节点与计算资源,将大规模数据进行并行化处理. MapReduce 是大家熟知的一种高容错分布式计算框架,然而 MapReduce 计算过程中因重复加载数据导致的大量磁盘 I/O 操作非常耗时低效,使其难以成为构建高性能并行机器学习算法的开发平台[2-3]. 最近,Spark 在解决磁盘 I/O 问题上被视为 MapReduce 的升级与替代者,Spark 是一种基于内存的集群式计算平台,允许机器将数据缓存在内存中,从而避免了反复的磁盘 I/O 操作[4]. 此外,Spark 采用一种全新的弹性分布式数据集(RDD)来解决容错问题,大大降低了容错不足导致的风险. 单机串行 ELM 已经得到了深入研究,然而单机平台的容量与承载能力在面对大规模数据膨胀问题时正面临极大的挑战,为此笔者基于 Spark 提出一种并行神经网络极速学习方法,以 Spark 平台特有的 RDD 高效数据集管理机制对极速学习机进行封装,将大规模数据中的高复杂度矩阵计算进行并行化,实现 ELM 的并行加速求解.

1 Spark 系统

传统的 MapReduce 框架在进行信息处理过程中需要对数据进行大量的磁盘 I/O 操作,并行效率较为低下. 基于内存的分布式计算框架 Spark 很好的解决了频繁的磁盘 I/O 问题. Spark 是一种高效的基于内存的多功能集群式计算平台,允许计算节点将中间结果缓存于内存中,从而解决了反复迭代等高密度计算过程中的磁盘频繁 I/O 操作. Spark 主要包括分布式文件系统、弹性分布式数据集、容错机制等几个模块.

1.1 分布式文件系统

Spark 沿用 Hadoop 平台所提供的分布式文件系统(hadoop distributed file system,HDFS),这款文件系统设计之初就是为了便捷的存储及操作大规模数据,此外,HDFS 内嵌了高容错性策略:①数据复写机制使同一份数据在多个节点产生副

投稿日期:2016 - 03 - 04;**修订日期**:2016 - 05 - 27

基金项目:国家自然科学基金资助项目(61572399);陕西省科技新星资助项目(2013KJXX - 29)

作者简介:邓万宇(1979—),男,河南南阳人,西安邮电大学副教授,博士,主要从事数据挖掘、机器学习与知识服务研究,E-mail:58028654@qq.com.

本.当一个节点发生故障,仍可对其他的节点的数据副本进行存取.②HDFS以心跳机制检测节点可用性.HDFS有唯一的主节点(namenode)及多个从节点(datanode),主节点对HDFS文件系统命名空间进行管理,而从节点则用于以块为单位存储数据.从节点周期性发送心包跳给主节点,当主节点无法获取集群上某从节点的心跳包时,则认定该节点已经死亡,并且不再对该节点进行I/O操作.

1.2 弹性分布式数据集(RDD)

RDD主要由位于HDFS之上的分布式文件构造,或是由其他RDD转化而来.在Spark中数据被封装成一个包含多个分片的RDD,分片是指Spark中数据的分布式片段,是Spark缓存管理器加载的基本单元.每个从节点仅需要保存训练集的一部分分片,如果从节点内存充足,分片将被缓存于从节点内存中,否则存储于磁盘内.在同一时间区间内每个从节点仅需要管理本节点所持有的数据分片,而无需对整个数据集负责.该机制可令计算过程实现并行化.设有 s 个从节点以及 p 个分片,当指定 $p > s$ 时,p 个分片可以在 s 个从节点并行计算,从而充分实现并行化.Spark允许用户根据应用的具体需求指定不同的分片个数 p.事实上,Spark根据用户的训练集大小给出一个最小分片个数 $p_{\min}$,当用户未指定 p 的值,或者所指定的 p 值小于 $p_{\min}$ 时,Spark将认为 $p = p_{\min}$.除此之外,Spark支持对RDD的两种并行操作,称为transformation和action,其中transformation用于从已存在的数据创建新的RDD,包括如map和filter等操作.而action则是对RDD进行计算,包括reduce和collect等.

1.3 Spark的容错机制

Spark拥有高容错性的关键在于只读权限的RDD,当数据集中某个分片丢失,Spark将申请对原始RDD进行transformation操作重新计算并创建分片信息.*lineage*机制用于保存并记录RDD之间的transformation操作,并以作为集中式元数据保存于主节点中.*lineage*可以保证对RDD进行有效的重新计算,并且只读权限的RDD可以保证如果需要重新执行*lineage*中已执行过的某个操作时,可以获得相同的计算结果.

2 ELM并行化实现

2.1 ELM算法描述

ELM由Huang于2004年提出[1],单隐藏层前馈神经网络发展而来,但ELM相较于传统的单隐藏层前馈神经网络,仅需要对隐藏层节点的个数进行设置,并随机生成隐藏层偏差和输入权值,采以最小二乘法求得输出权值,仅一次计算而无需迭代,相较于BP等其他神经网络有显著的提高[5],并具有优秀的泛化性能.

设隐藏层节点数为 $\tilde{N}$,激励函数为 $g(x)$,N 个训练样本为 $\{(x_i,t_i)\mid x_i\in R^n, x_i\in R^m\}_{i=1}^{N}$,则单隐藏层前馈神经网络统一模型可表示为:

$$\sum_{i=1}^{\tilde{N}}\beta_i g_i(x_j)=\sum_{i=1}^{\tilde{N}}\beta_i g(\boldsymbol{a}_i\cdot\boldsymbol{x}_j+b_i)=y_j,\quad j=1,2,\cdots,N. \tag{1}$$

其中:

$\boldsymbol{x}_i=[x_{i1},x_{i2},\cdots,x_{in}]^{\mathrm{T}}\in R^n$;

$\boldsymbol{t}_1=[t_{t1},t_{i2},\cdots,t_{in}]^{\mathrm{T}}\in R^m$.

g_i 代表第 i 个隐藏层节点的激励函数 $g(\boldsymbol{a}_i\cdot\boldsymbol{x}_i+b_i)$;$\boldsymbol{a}_i=[\boldsymbol{a}_{i1},\boldsymbol{a}_{i2},\cdots,\boldsymbol{a}_{iN}]^{\mathrm{T}}$ 为连接到第 i 个隐藏层节点的输入权值;$\beta_i=[\beta_{i1},\beta_{i2},\cdots,\beta_{im}]^{\mathrm{T}}$ 为连接到第 i 个隐藏层节点的输出权值;b_i 为第 i 个隐藏层节点的偏差(bias),$\boldsymbol{y}_i$ 为第 j 个样本的输出,训练误差为 $\sum_{j=1}^{N}\|\boldsymbol{y}_j-\boldsymbol{t}_j\|^2$.模型预测与输出完全拟合时,式(1)可表示为:

$$\boldsymbol{H}\beta=\boldsymbol{T}. \tag{2}$$

其中

$$\boldsymbol{H}(\boldsymbol{a}_1,\cdots,\boldsymbol{a}\tilde{N},b_1,\cdots,b\tilde{N},\boldsymbol{x}_1,\cdots,\boldsymbol{x}\tilde{N})=\begin{bmatrix} g(\boldsymbol{a}_1\cdot\boldsymbol{x}_1+b_1)\cdots g(\boldsymbol{a}\tilde{N}\cdot\boldsymbol{x}_1+b\tilde{N}) \\ \vdots \quad \cdots \quad \vdots \\ g(\boldsymbol{a}_1\cdot\boldsymbol{x}_N+b_1)\cdots g(\boldsymbol{a}\tilde{N}\cdot\boldsymbol{x}_N+b\tilde{N}) \end{bmatrix}_{N\times\tilde{N}}$$

$$\beta=\begin{bmatrix}\beta_1^{\mathrm{T}}\\ \vdots \\ \beta_{\tilde{N}}^{\mathrm{T}}\end{bmatrix}_{\tilde{N}\times m},\quad T=\begin{bmatrix}t_1^{\mathrm{T}}\\ \vdots \\ t_{\tilde{N}}^{\mathrm{T}}\end{bmatrix}_{\tilde{N}\times m}.$$

式中:$\boldsymbol{H}$ 为 N 个样本相关的隐藏层输出矩阵;β 为模型输出权重,其最小二乘解为

$$\hat{\beta}==\boldsymbol{H}^{\dagger}\boldsymbol{T}. \tag{3}$$

式中:$\boldsymbol{H}^{\dagger}$ 为 $\boldsymbol{H}$ 的摩尔彭若思广义逆(Moore-Penrose generalized inverse).回归预测 $\boldsymbol{Y}$ 可表示为:

$$\boldsymbol{Y}=\boldsymbol{H}\hat{\beta}=\boldsymbol{H}\boldsymbol{H}^{\dagger}\boldsymbol{T}. \tag{4}$$

该多元回归的误差为:

$$\|\boldsymbol{e}\|=\|\boldsymbol{Y}-\boldsymbol{T}\|=\|\boldsymbol{H}\boldsymbol{H}^{\dagger}\boldsymbol{T}-\boldsymbol{T}\|. \tag{5}$$

$\boldsymbol{H}^{\dagger}$ 的求法主要包括正交投影法[6],迭代法以及SVD分解法(single value decomposition)法[17]等.ELM算法描述如下:

算法 1　ELM 算法

给定 N 个独立训练样本集 激励函数 $g(x)$ 及隐藏层节点个数 N.

步骤 1:随机分配输入权值和偏差:

$$(\boldsymbol{a}_i, b_i), \boldsymbol{a}_i \in R^N, b_i \in \boldsymbol{R}, i = 1, 2, \cdots, \tilde{N};$$

步骤 2:计算隐藏层输出矩阵

$$\boldsymbol{H}(\boldsymbol{a}_1, \cdots, \boldsymbol{a}_{\tilde{N}}, b_1, \cdots, b_{\tilde{N}}, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_N) = \begin{bmatrix} g(\boldsymbol{a}_1 \cdot \boldsymbol{x}_1 + b_1), & \cdots, & g(\boldsymbol{a}_{\tilde{N}} \cdot \boldsymbol{x}_1 + b_{\tilde{N}}) \\ \vdots & \cdots & \vdots \\ g(\boldsymbol{a}_1 \cdot \boldsymbol{x}_N + b_1), & \cdots, & g(\boldsymbol{a}_{\tilde{N}} \cdot \boldsymbol{x}_N + b_{\tilde{N}}) \end{bmatrix}_{N \times \tilde{N}};$$

$$\aleph = \{(x_i, t_i) \mid x_i \in R^n, t_i \in R^m, i = 1, 2, \cdots, N\}$$

步骤 3: 计算输出权值:$\hat{\beta} = \boldsymbol{H}^{\dagger}\boldsymbol{T}$.

2.2　ELM 算法并行化策略

ELM 具有无需迭代、一次求解的优点. 但隐藏层输出矩阵 $\boldsymbol{H}$,在训练样本较大时,存在存储、计算开销过大的缺点[8];并且 ELM 需要将数据及中间结果载入内存,这在单节点情况难以实现. 因此,如何减少磁盘 I/O 时间、内存开销,结合 Spark 平台基于内存操作的优势,是 ELM 并行化实现的关键.

算法 1 中步骤(1)需要产生一个$N \times \tilde{N}$维阵. 步骤(2)为矩阵乘法运算,可以采用并行化方式运行. 步骤(3)则主要计算 $\boldsymbol{H}$ 矩阵的摩尔彭若思伪逆矩阵 $\boldsymbol{H}^{\dagger}$,可以采用并行的 EVD 法进行求解. $\boldsymbol{H}$ 是一个 $N \times \tilde{N}$ 维的矩阵,因此 $\boldsymbol{H} \times \boldsymbol{H}^{\mathrm{T}}$ 是一个$N \times N$ 维方阵,在现实应用环境下,训练集往往较为庞大,故而以基于内存的计算方法对 $\boldsymbol{H} \times \boldsymbol{H}^{\mathrm{T}}$ 矩阵进行操作将变得不现实. 而隐藏层节点 $\tilde{N}$ 远小于 N,故转为

$$\boldsymbol{H}^{\dagger} = \mathrm{inv}(\boldsymbol{H}^{\mathrm{T}} \times \boldsymbol{H}) \times \boldsymbol{H}^{\mathrm{T}}.$$

进而对 $\hat{\beta} = \mathrm{inv}(\boldsymbol{H}^{\mathrm{T}} \times \boldsymbol{H}) \times \boldsymbol{H}^{\mathrm{T}} \times \boldsymbol{T}$ 计算. 所以并行化的关键在于对 $\boldsymbol{H}^{\mathrm{T}} \times \boldsymbol{H}, \boldsymbol{H}^{\mathrm{T}} \times \boldsymbol{T}$ 的并行化计算. 令除此之外在获得 $\boldsymbol{H}^{T} \times \boldsymbol{H}$ 之后,令 $\boldsymbol{Y} = \boldsymbol{H} \times \boldsymbol{H}^{\mathrm{T}}$,则 $\boldsymbol{X}$ 的特征值分解为 $\boldsymbol{Y} = \boldsymbol{V} \times \sum \times \boldsymbol{V}^{\mathrm{T}}$,因此可通过 $\boldsymbol{Y}^{-1} = \boldsymbol{V} \times \sum^{-1} \times \boldsymbol{V}^{\mathrm{T}}$ 计算 $\mathrm{inv}(\boldsymbol{H}^{\mathrm{T}} \times \boldsymbol{H})$. 经过上述分析,并行化的关键在于如何将算法设计为基于每个样本的操作. 如上,ELM 计算的瓶颈在于对 $\mathrm{inv}(\boldsymbol{H}^{\mathrm{T}} \times \boldsymbol{H}), \boldsymbol{H}^{\mathrm{T}} \times \boldsymbol{H}$ 的并行化. 因此算法高度并行化的关键在于将数据集的操作转化为针对单个样本向量的操作,并且在减少 transformation 次数及 action 次数的前提下,以向量集合作为输出节点输出[9].

2.2.1　并行算法设计

训练样本结合个数为 N,隐藏层输出矩阵 $\boldsymbol{H}$ 表示为 $\boldsymbol{H} = \begin{bmatrix} H_{[1]\cdot} \\ \vdots \\ H_{[N]\cdot} \end{bmatrix}$.

其中:

$H_{[i]\cdot} = [g(\boldsymbol{a}_1 \cdot \boldsymbol{x}_i + b_1), \cdots, g(\boldsymbol{a}_{\tilde{N}} \cdot \boldsymbol{x}_i + b_{\tilde{N}})]$为 $\boldsymbol{H}$ 的第 i 个行向量,即为第 i 个训练样本 x_i 的隐藏层节点输出向量.

令 $\boldsymbol{Y} = \boldsymbol{H}^{\mathrm{T}} \times \boldsymbol{H}, \boldsymbol{Z} = \boldsymbol{H}^{\mathrm{T}} \times \boldsymbol{T}$,则 $\boldsymbol{Y}$、$\boldsymbol{Z}$ 可以表示如下

$$Y_{[i][j]} = \sum_{k=1}^{N} H_{[k][j]} \times H_{[k][j]}, \tag{6}$$

$$Z_{[i][j]} = \sum_{k=1}^{N} H_{[k][j]} \times H_{[k][j]}. \tag{7}$$

观察可知,不妨令 $\boldsymbol{Y}$ 的第 i 行向量

$$\begin{aligned} Y_{[i]\cdot} &= [Y_{[i][1]}, \cdots, Y_{[i][\tilde{N}]} \\ &= [\sum_{k=1}^{N} H_{[k][i]} \times H_{[k][1]}, \cdots, \sum_{k=1}^{N} H_{[k][i]} \times H_{[k][\tilde{N}]}] \\ &= \sum_{k=1}^{N} [H_{[k][i]} \times H_{[k][1]}, \cdots, H_{[k][i]} \times H_{[k][\tilde{N}]}] \\ &= \sum_{k=1}^{N} H_{[k][i]} \times [H_{[k][1]}, \cdots, H_{[k][\tilde{N}]}] \\ &= \sum_{k=1}^{N} H_{[k][i]} \times H_{[k]\cdot} \end{aligned} \tag{8}$$

由式(8)可得

$$\boldsymbol{Y} = \begin{bmatrix} Y_{[1]\cdot} \\ \vdots \\ Y_{[\tilde{N}]\cdot} \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{N} H_{[k][1]} \times H_{[k][\cdot]} \\ \vdots \\ \sum_{k=1}^{N} H_{[k][\tilde{N}]} \times H_{[k][\cdot]} \end{bmatrix}_{\tilde{N} \times \tilde{N}} =$$

$$\sum_{k=1}^{N} \begin{bmatrix} H_{[k][1]} \\ \vdots \\ H_{[k][N]} \end{bmatrix} \times H_{[k][\cdot]} = \sum_{k=1}^{N} (H_{[k][\cdot]})^{\mathrm{T}} \times H_{[k][\cdot]}. \tag{9}$$

同理 $\boldsymbol{Z}$ 可表示为

$$Z=\begin{bmatrix} Z_{[1]}\cdot \\ \vdots \\ Z_{[\tilde{N}]}\cdot \end{bmatrix}=\begin{bmatrix} \sum_{k=1}^{N} H_{[k][1]}\times T_{[k][\cdot]} \\ \vdots \\ \sum_{k=1}^{N} H_{[k][\tilde{N}]}\times T_{[k][\cdot]} \end{bmatrix}_{\tilde{N}\times 1}=$$

$$\sum_{k=1}^{N}\begin{bmatrix} H_{[k][1]} \\ \vdots \\ H_{[k][N]} \end{bmatrix}\times T_{[k][\cdot]}=\sum_{k=1}^{N}(H_{[k][\cdot]})^{\mathrm{T}}\times T_{[k][\cdot]}. \tag{10}$$

经上述一系列推导,$\boldsymbol{Y}$、$\boldsymbol{Z}$ 的计算可以转化为对 N 训练样本独立并行化运算,并最终求和的方式获得,令第 k 个样本输出为 $\tilde{N}\times\tilde{N}$ 维矩阵 $\tilde{\boldsymbol{Y}}_{[k]}$ 和 $\tilde{N}\times 1$ 维向量 $\tilde{\boldsymbol{Z}}_{[k]}$.

$$\tilde{\boldsymbol{Y}}_{[k]}=(H_{[k][\cdot]})^{\mathrm{T}}\times H_{[k][\cdot]}; \tag{11}$$

$$\tilde{\boldsymbol{Z}}_{[k]}=(H_{[k][\cdot]})^{\mathrm{T}}\times T_{[k][\cdot]}. \tag{12}$$

即 $\boldsymbol{Y}=\sum_{k=1}^{N}\tilde{\boldsymbol{Y}}_{[k]}$;$\boldsymbol{Z}=\sum_{k=1}^{N}\tilde{\boldsymbol{Z}}_{[k]}$. $\tilde{\boldsymbol{Y}}_{[k]}$ 和 $\tilde{\boldsymbol{Z}}_{[k]}$ 可以在 N 个分片上分布式存储,并实现并行 map 方法计算. 在 map 计算结束后,需要将所分片上的 $\tilde{\boldsymbol{Y}}_{[k]}$ 和以 reduce 方法进行求和,并最终在主节点获取 $\tilde{\boldsymbol{Z}}_{[k]}$、$\boldsymbol{Y}$ 和 $\boldsymbol{Z}$ 的值. 整个算法主节点和从节点需要两次通信,第一次通信是主节点生成输入神经元权值向量 $\boldsymbol{a}$ 和隐层阈值 b 等参数,并将各参数传递给从节点[10]. 第二次通信则是将各从节点数据通过 reduce 操作汇集到主节点内.

算法 2、算法 3 将详细描述计算过程.

算法 2 $\hat{\beta}=\boldsymbol{H}^{+}\boldsymbol{T}$ 的分布式求解

步骤 1:主节点构造输入权值和偏差向量$(\boldsymbol{a}_i,b_i)$,$\boldsymbol{a}_i\in R^N$;$b_i\in R$;$i=1,2,\cdots,\tilde{N}$,激励函数 $g(x)$,并将$(\boldsymbol{a},b)$,$g(x)$,$\tilde{N}$ 以广播形式交给各从节点.

步骤 2:从节点调用算法 3 计算 $\tilde{\boldsymbol{Y}}_{[k]}$ 和 $\tilde{\boldsymbol{Z}}_{[k]}$,并汇总各从节点结果,将 $\boldsymbol{Y}=\sum_{k=1}^{N}\tilde{\boldsymbol{Y}}_{[k]}$;$\boldsymbol{Z}=\sum_{k=1}^{N}\tilde{\boldsymbol{Z}}_{[k]}$ 交还主节点.

步骤 3:主节点以 EVD 分解,得到 $\hat{\beta}=\mathrm{inv}(\boldsymbol{Y})\times\boldsymbol{Z}$.

算法 3:$\tilde{\boldsymbol{Y}}_{[k]}$ 和 $\tilde{\boldsymbol{Z}}_{[k]}$ 的分布式求解

1:从节点得到输入权值和偏差向量$(\boldsymbol{a},b)$,$g(x)$,$\tilde{N}$,当前训练样本的输入样本数组 *inarr*,输出样本为 *out*.
2:定义空数组 h 作为样本的隐藏层输出向量;

3:for $i=1$ to $\tilde{N}$;
4: $x=0$;
5: for $j=1$ to *InArr. length*;
6: $x+=inputarr_{[j]}\times\boldsymbol{a}_{[i][j]}$
($\boldsymbol{a}_{[i][j]}$ 为第 j 个输入神经元到第个 i 隐藏层节点的权值);
7: endfor.
8: h. append$(g(x+b_{[i]}))$
(将 $g(x+b_{[i]})$ 添加到 h 中,$b_{[i]}$ 为第 i 个隐藏层节点阈值),
9:endfor.

10:定义长度为 $\tilde{N}\times\tilde{N}$ 的向量 $\boldsymbol{y}$,和长度为 N 的向量 $\boldsymbol{z}$,

11:for $i=1$ to $\tilde{N}$,
12: $x=0$,

13: for $j=1$ to $\tilde{N}$,
14: y. append$(h_{[i]}\times h_{[j]})$
(将 $h_{[i]}\times h_{[j]}$ 添加到 $\boldsymbol{y}$ 中),
15: endfor.
16: z. append$(h_{[i]}\times out)$
($b_{[i]}$ 为第 i 个隐藏层节点阈值),
17:endfor.

18:当前从节点得到单个样本的输出 $\tilde{\boldsymbol{Y}}_{[k]}$ 和 $\tilde{\boldsymbol{Z}}_{[k]}$.

3　基于 Spark 的具体实现

3.1　数据的存储

当样本规模 N 和隐层节点个数 $\tilde{N}$ 较大时,优化向量 $\tilde{\boldsymbol{Y}}_{[k]}$、$\tilde{\boldsymbol{Z}}_{[k]}$ 的数据存储方式,将有效提升算法性能.

算法 3 中,每个样本 k 都将产生向量 $\tilde{\boldsymbol{Y}}_{[k]}$ 和 $\tilde{\boldsymbol{Z}}_{[k]}$,该向量通常较为稀疏. 例如当前为第 j 个样本,目标数组 $\tilde{\boldsymbol{Y}}_{[j]}$ 用于表示矩阵 $H^{\mathrm{T}}_{[j]} \cdot \times H^{\mathrm{T}}_{[j]} \cdot$,由于该矩阵为 $\tilde{N}^2$ 维的方阵,因此 $\tilde{\boldsymbol{Y}}_{[j]}$ 可表示为 $(0.92,0,0,0,0.88,0.35,0,\cdots,0)_{\tilde{N}^2}$,设 $\tilde{\boldsymbol{Y}}_{[l]}$ 只有三组非零的 index-value 对需要存储. 在内存中 index-value 的存储形式为“1:0.92”,“5:0.88”和“7: 0.35”,其封装形式主要有两种:①类表示(class approach CA);②数组表示(array approach AA)[11]. 如图 1、图 2 所示.

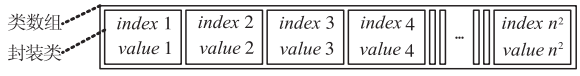

图 1　类表示

Fig. 1　Class approach

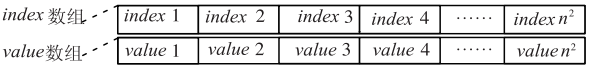

图 2　数组表示

Fig. 2　Array approach

类表示是将 index-value 对封装在类中,并以数组形式存储每一样本的一簇类对象:数组表示直接将 *index* 和 *value* 分别用两个数组进行存储.

类表示的优势在于源码的高可读性,但由于每个类在内存还需存储该类相应的头文件,因此类表示相较于数组表示需要消耗更多的内存. 除此之外,类表示法需要通过指针才能对类元素进行存取,数组表示则可以直接操作索引和值,因此算法中选择以数组表示法对向量 $\tilde{\boldsymbol{Y}}_{[k]}$、$\tilde{\boldsymbol{Z}}_{[k]}$ 进行存储[12].

3.2　任务均衡 MapPartition 和 Coalesce 函数

在算法 2($\hat{\beta} = \boldsymbol{H}^{\dagger}\boldsymbol{T}$ 的分布式求解)中,每个样本 $x[k]$($k \in N$)在 map 阶段都会产生向量组 $\tilde{\boldsymbol{Y}}_{[k]}$ 和 $\tilde{\boldsymbol{Z}}_{[k]}$,在 reduce 阶段主节点将会对所有 N 组向量进行求和:$\boldsymbol{Y} = \sum_{k=1}^{N} \tilde{\boldsymbol{Y}}_{[k]}$;$\boldsymbol{Z} = \sum_{k=1}^{N} \tilde{\boldsymbol{Z}}_{[k]}$. 该过程需要从节点把全部 N 组向量 $\tilde{\boldsymbol{Y}}_{[k]}$ 和 $\tilde{\boldsymbol{Z}}_{[k]}$ 传输并缓存于主节点,该默认方式将使计算任务严重倾斜并趋于串行,无法充分发挥集群的并行计算优势[13].

为减少主节点计算量,均衡计算任务,从节点在提交结果之前进行两级运算:第一级,选择 Spark 中的 mapPartitions 代替 map 函数,对每个分片内所有向量进行求和,一个分片仅产生唯一一组中间变量;第二级,调用 Coalesce 函数对当前节点内所有分片结果进行求和,最终每个节点仅产生一组结果,如图 3.

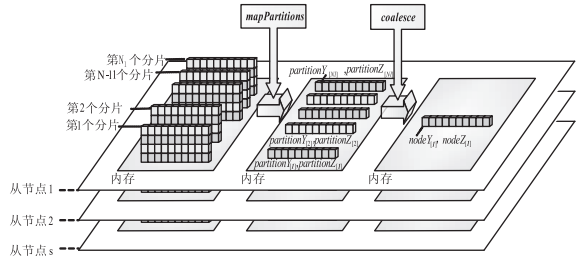

图 3　任务均衡

Fig. 3　Task equilibrium

如图 3 所示,当前第 i 个节点内共有 N_i 个分片,通过 mapPartitions 操作对分片内的向量进行粗粒度求和,令一个分片产生一组结果,向主节点提交的数据量下降至每个节点仅包含 N_i 个向量 partition$Y_{[j]}$ 和 partition$Z_{[j]}$ ($j \in N_i$). mapPartitions 使原本的计算结果 $\tilde{\boldsymbol{Y}}_{[k]}$ 和 $\tilde{\boldsymbol{Z}}_{[k]}$ ($k \in N$) 下降为每个节点仅包含 N_i 个向量 partition$\boldsymbol{Y}_{[j]}$ 和 partition$\boldsymbol{Z}_{[j]}$ ($j \in N_i$),共 p 组($\sum_{i=1}^{s} N_i = p$,s 为节点总数).

第二步,每个节点仅包含 N_i 组向量,为达到充分利用从节点计算资源的目的,在 mapPartitions 之后调用 coalesce 函数,同一从节点 i 的所有分片最终仅产生一组输出向量 Node$\boldsymbol{Y}_{[i]}$,Node$\boldsymbol{Z}_{[i]}$ ($i \in s$):

$$\begin{cases} \mathrm{Node}Y_{[i]} = \sum\limits_{j \in Ni} \mathrm{partition}Y_{[j]} \\ \mathrm{Node}Z_{[i]} = \sum\limits_{j \in Ni} \mathrm{partition}Z_{[j]} \end{cases} \quad i \in s.$$

通过 mapPartitions 与 coalesce 相互使用,使得所有本地分片在从节点得以充分计算,令从节点提交到主节点的向量数量由 N 个下降至 s,从而使计算任务得以均衡分配,算法并行度显著提高.

此外,稀疏向量的求和操作通过构造稠密向量进行,由于稠密向量包含所有列元素,因此可以与稀疏矩阵直接进行对位求和,从而避免反复构

造索引数组与值数组.

3.3 broadcast(广播变量)

广播变量机制,是指将主节点的参数以只读形式缓存于从节点中,每个从节点仅保存一份副本,该节点内所有分片都可共享使用.

算法2($\hat{\beta} = \boldsymbol{H}^{+}\boldsymbol{T}$ 的分布式求解)中,$\tilde{\boldsymbol{Y}}_{[k]}$ 和 $\tilde{\boldsymbol{Z}}_{[k]}$ 的计算需要同一组神经元权值向量 a、隐层阈值 b、隐层个数 $\tilde{N}$ 等参数信息,由于节点个数 s 小于分片个数 p,因此算法实现阶段,以从节点为单位进行广播变量拷贝,既可保证所有分片获得参数信息,又能减少参数副本的数量,从而节省内存空间及通信消耗.

p 个分片以只读方式共享 a、b、$\tilde{N}$ 等参数信息,因此将第一传递过程中的参数 a、b、$\tilde{N}$ 定义为广播变量. 由3.2节mapPartitions阶段操作可知,位于同一节点的所有分片可共享广播变量.

4 性能评估

为验证算法的性能,我们以不同规模的数据集对算法进行测试,并将SparkELM与SVM、BP、ELM等算法进行对比,主要以时间加速比(speed-up ratio)和测试精度(testing accuracy)进行评估. 所使用数据集在LIBSVM数据集网站均可以下载[14]. Spark实验平台为Debian操作系统机器,共7台机器,都为16 GB内存,四核Xeon E5440(2.83 G),集群版本Hadoop 2.5.0、Java 1.7.0_74、Spark1.3.1.

4.1 数据集

选定卫星图像(satimage)、航空记录(shuttle)等4种真实的分类数据集以及汽车价格(autoPrice)、股票价格(stocks)等9种回归数据集对SparkELM算法的性能进行验证. 回归数据集信息如表1,分类数据信息如表2所示.

表1 回归数据集信息

Tab.1 Regression datasets

数据集	样本数目		属性数目		数据集	样本数目		属性数目	
	训练样本	测试样本	连续属性	类别		训练样本	测试样本	连续属性	类别
diabetes	576	192	8	2	shuttle	43 500	14 500	7	2
segment	1 500	810	36	7					
iJCNN	464 810	116 202	22	2	satimage	4 435	2 000	36	6

表2 分类数据集信息

Tab.2 Classification datasets

数据集	样本数目		属性数目		数据集	样本数目		属性数目	
	训练样本	测试样本	连续属性	类别		训练样本	测试样本	连续属性	类别
delta	3 000	4 129	6	1	bank domains	4 500	3 692	8	0
elevators	4 000	5 517	6	0	stocks domain	450	500	10	0
auto price	80	79	14	1	machine cPU	100	109	6	0
Triaziones	100	86	60	0	califomia housing	8 000	1 246	8	0
servo	80	87	0	4					

Satimage通过已知的3×3像素的图像,对中心像素的类别进行预测. 陆地卫星一次扫描结果(Landsat MSS imagery)包括4幅不同频谱图像,每幅图像包含2 340×3 380个像素. 为了实验需要,每幅图像截取82×100像素构造Satimage数据集. 构造方法是:将82×100像素划分为多个3×3像素的正方形,因每组图像共有4幅,因此可切出4个正方形,这4个正方形构造一条数据记录,显然数据维度是4×9=36. 图像是矩阵形式,为便于计算,采用自上到下、自左向右的顺序将其矢量化(如图4所示). 所求目标特征应该是第

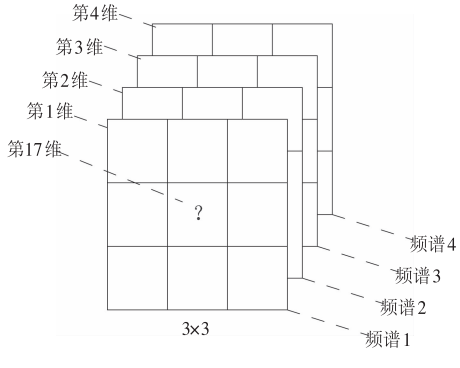

图4 Satimage由36维像素预测中心像素类型

Fig.4 Satimage prediction center pixel type by 36 dims

17、18、19、20 维. 若认为目标特征的类别仅与自己有关,那么可以直接采用第 17、18、19、20 维进行预测. 若目标特征还和周围像素有关,则选择全部像素. 笔者取用全部 36 维像素,该方法较普遍. 数据共有 6 种类型:red soil、cotton crop、grey soil、damp grey soil、soil with vegetation stubble、very damp grey soil,分类目标是根据 36 维像素预测中心像素类型.

Shuttle 根据飞行器所处的环境信息来判断飞机应选择人工降落或是自动降落(如图 5 所示). 数据有 9 个属性,2 个类别,58 000 条样本. 7 个属性分别是 time、stability、error、sign、wind、magnitude、visibilitya. 7 个类别分别是 Auto(自动降落)和 NoAuto(人工降落). 80% 的数据将选择人工降落(NoAuto),其余为自动降落,因此默认的分类精度是 80%,目标的精度是 99% ~99.9%. 数据集以随机切割产生训练集(43 500 个样本)和测试集(14 500 个样本).

数据集 IJCNN. 给定一个长度为 $T = 49\ 990$ 的时间序列,序列中的每个样本都包含 4 个输入:$x_1(k),\cdots,x_4(k)$ 和一个输出 $y(k)$. 另外长度为 91 701的时间序列用来测试. 属性 $x_1(k)$ 表示与系统自然周期相关的二进制同步脉冲,为 10 位二进制形式,并以 9 个 0,一个 1 的规模出现. 属性 $x_2(k),x_3(k),x_4(k)$ 是[-1.5,1.5] 之间的实数,其中 $x_4(k)$ 和 $y(k)$ 最为相关,因此后面构造特征时不仅考虑了 $x_4(k)$,还考虑了它的前后时刻的信息. 我们采用文中的方法对时间序列行进行特征构造:$x_1(k)$ 扩展为 $x_1(k-5),\cdots,x_1(k+4)$ 等 10 个特征;$x_4(k)$ 扩展为 $x_4(k-5),\cdots,x_4(k+4)$ 等 10 个特征,$x_2(k),x_3(k)$ 直接使用;最终形成 22 个特征. 训练数据中 90% 的训练集为第 1 类,因此分类精度默认为 90%,分类任务为构造模型已到达更高分类精度.

图 5　Shuttle 判断飞行器的降落方式

Fig. 5　Shuttle determine the landing mode of an aircraft

表 3 和 4 为不同算法的比较. 对比表 3、表 4 中 4 种算法在 13 个真实数据集的表现知,SparkELM 算法在大部分数据集上的"Sigmoid"函数:$g(x) = 1/(1+e^{\lambda x})$ 隐藏层初始节点都选取 5 个,每次递增 5,并基于 5 - 折交叉验证的方式选取最优个数,继而进行 50 次试验选取最优结果并取其平均值进行采集. 对于 SVM 的核函数则选取径向基函数,并采用 Hsu 和 Lin 提出的排列组合方法选择最优的参数 C 和高斯核 γ 进行设置:C 和 γ 的取值范围是 $C = [2^{12},2^{11},\cdots,2^{-2}]$,$\gamma = [2^{4},2^{3},\cdots,2^{-10}]$,共 225 种组合,对于每组 (C,γ) 进行 50 次随机试验,并对最优结果的平均值进行采集. 实验数据被归一化到[0,1]之间,输出归一化到[-1,1].

4.2　训练时间与误差

BP、ELM 和 SparkELM 的激励函数都选择分类及回归 RMSE 精度比 BP、SVM 有更好的泛化性能,并相较于 ELM 也有优秀表现.

表 3　4 种不同算法的均方差(RMSE)

Tab. 3　RMSE of 4 algorithms

数据集	BP		SVM		ELM		SparkELM	
	训练样本	测试样本	训练样本	测试样本	训练样本	测试样本	训练样本	测试样本
delta ailerons	0.040 9	0.048 1	0.041 8	0.042 9	0.042 3	0.043 1	0.038 2	0.041 5
delta elevators	0.054 4	0.059 2	0.053 4	0.054 0	0.055 0	0.056 8	0.054 0	0.053 1
auto price	0.065 2	0.093 7	0.065 2	0.093 7	0.075 4	0.099 4	0.075 4	0.104 3
triaziones	0.143 2	0.182 9	0.143 2	0.182 9	0.189 7	0.200 2	0.161 4	0.203 5
servo	0.084 0	0.117 7	0.084 0	0.117 7	0.070 7	0.119 6	0.096 1	0.118 3
bank domains	0.045 4	0.046 7	0.045 4	0.046 7	0.040 6	0.036 0	0.043 7	0.043 1
stocks domain	0.050 3	0.051 8	0.050 3	0.051 8	0.025 1	0.034 8	0.024 1	0.298 7
machine cpu	0.035 2	0.082 6	0.057 4	0.081 1	0.033 2	0.053 9	0.025 9	0.056 4
califomia housing	0.108 9	0.118 0	0.108 9	0.118 0	0.121 7	0.126 7	0.117 4	0.153 6

表4 4种算法在分类问题上的精度比较

Tab.4 Accuracy comparison of 4 algorithms in classification problem

数据集	BP		SVM		ELM		SparkELM	
	训练样本	测试样本	训练样本	测试样本	训练样本	测试样本	训练样本	测试样本
Diabetes	0.866 3	0.744 3	0.787 6	0.773 1	0.786 8	0.775 7	0.784 7	0.781 2
Segment	0.969 2	0.862 7	—	—	0.973 5	0.949 5	0.969 3	0.954 3
shuttle	0.999 7	0.994 0	—	—	0.996 5	0.994 0	0.995 2	0.993 1
Satimage	0.952 6	0.823 4	—	—	0.935 2	0.890 4	0.926 2	0.892 0

4.3 不同参数下的性能变化分析

在Spark1.3.1中,reduce阶段,数据集的分片方式将直接影响算法性能,同时算法的最优分片数与群节点有很大联系,因此我们在数据集拥有不同节点数的情况下,调整数据分片数量,来进行并行性能测试.我们选择使用shuttle、ijcnn两个数据集进行试验,为保证结果的准确性,将所使用的shuttle数据增加至原始数据的100倍.集群的节点数由1个增加至6个,分片由1个增加至10个.为在不同节点情况下,不同分片数对算法产生的影响,首先控制每组实验的节点数不变,令分片个数逐渐增多,算法的执行时间如图6(a)~(f)所示.

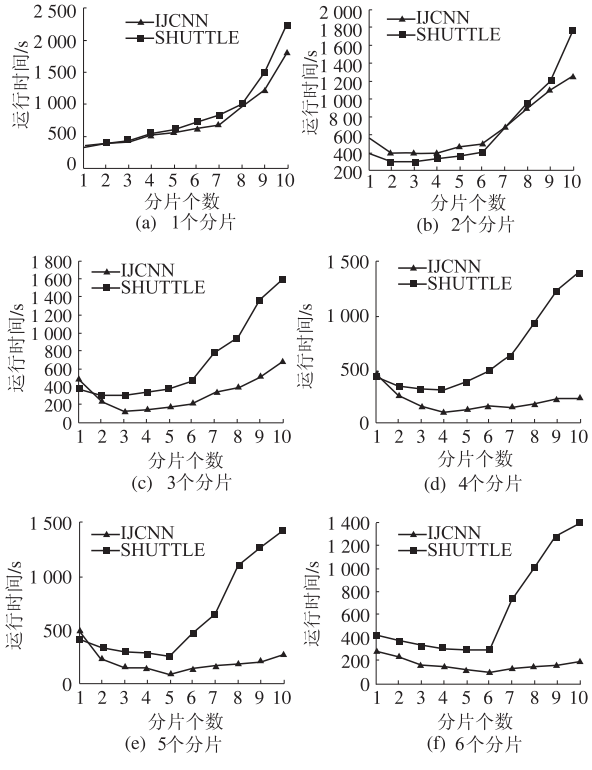

图6 运行时间随分片变化的情况

Fig.6 Time cost changing with splits

由图6(a)~(f)所示,在每组实验中保持集群节点个数不变,将分片由1个增加至10个,ELM算法训练时间都呈现先下降再逐渐攀升的趋势.训练时间首先随着分片的增多而逐渐减少,当分片数趋近于节点数的某个邻域范围时,训练时间达到最小值,而后随着分片数量的增多,算法的时间逐渐增加.呈现上述结果的主要原因是由于,ELM程序计算时间主要由mapPartitions与reduce两阶段的计算组成,当mapPartitions阶段完成后,需要将各从节点计算结果汇总至主节点进行reduce计算,因此算法除从节点和主节点计算用时外,还包括信息传递的通讯时间;在当前节点数不变得情况下,而随着分片数量的增加,当分片数趋近于节点数时,主节点得以最佳分片方式将数据集分发至各从节点进行运算,此时可保证在不需要额外通信开销的情况下,最大程度利用集群所有计算资源,并且由于分片个数趋近于节点个数,因此仅需要一个批次的数据传输即可完成分片结果至主节点的数据汇总,从而使通信开销降至最低;集群中所有从节点在分片个数等于节点个数时已被充分利用,因此在分片个数大于节点个数并逐渐呈现增长趋势时,某些集群将出现各节点数据分片分配不平均的情况,获取较多计算任务的从节点在进行额外任务的计算时,其他节点将出现停等现象,除此之外,数据汇总时由于该节点将产生额外的传输开销,因此ELM算法整体通信代价将逐渐增加,导致算法时间进一步提升,故而其所用时间也将呈增长趋势.

当数据集分片方式固定,集群节点个数对算法性能的影响可通过观察图7(a)~(f)可知:当提升集群从节点数量,集群节点逐渐趋近于分片个数时,主节点将计算任务重新分配,减少了各节点的平均任务数量及计算时间,提升了算法的整体并行性能,新添加节点提交任务所增加通信时

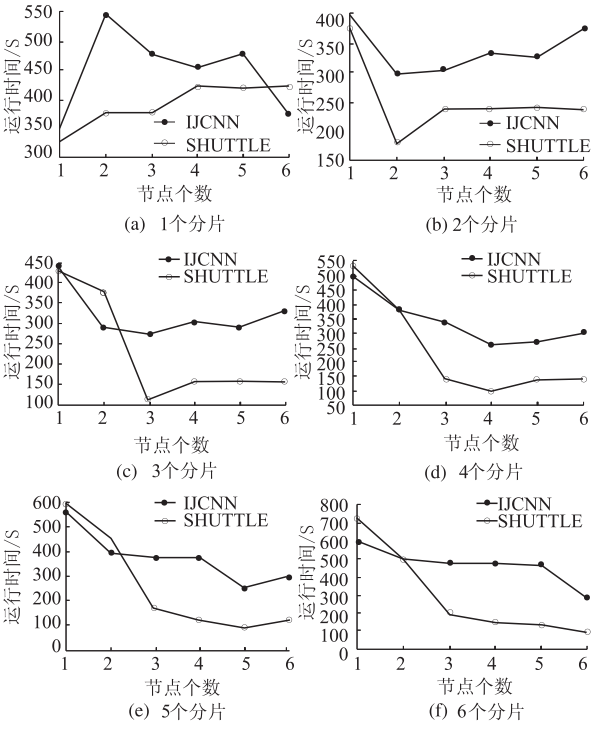

图 7　分片固定时运行时间随节点个数变化情况

Fig. 7　Time cost changing with nodes(fixed partition number)

间,相较于算法提升时间而言要小得多,该阶段 ELM 算法的整体性能随着节点增加将有所提升;当节点数大于分片个数并继续增加时,由于集群所有从节点已获得较为均匀的最小分片计算任务,其后所添加的节点并未使 mapPartitions-reduce 阶段的计算时间有所减少,反而增加了额外通信开销,整体而言,并未使 ELM 算法性能有所提升.

因此,节点个数接近于分片数时,ELM 算法整体时间代价最小,性能最优. 通常而言,集群在没有任务丢失的前提下,可以使 ELM 算法性能最为高效的分片个数 $P_{\text{最佳}} \approx$ 节点个数 s.

4.4　加速比

为评价 SparkELM 算法的加速性能,我们引入加速比 $S(n)$ 对算法性能进行衡量,加速比的计算公式定义如下:

$$S(n) = \frac{\text{单节点下的计算时间}}{n \text{ 个节点下的计算时间}}.$$

选取数据集 IJCNN 为例,并增加节点个数,将内核个数从 4 枚增加至 24 枚,为保证在节点个数不同时算法具有最优的运算性能,算法基于上述 3.2 节中论述的最优分片原则,将按照在当前节点最优分片区间对数据集进行划分,以保证 ELM 算法在当前集群的节点情况下具有最佳的并行性能. 此外,为增加测试的数据规模,提升 mapPartitions-reduce 阶段运算时间的比重,减少通信消耗对计算的影响,实验中将原数据集 IJCNN 分别复制 100 倍、200 倍、400 倍和 800 倍作为训练集对 ELM 算法进行测试. 4 种情况表现如图 8 所示.

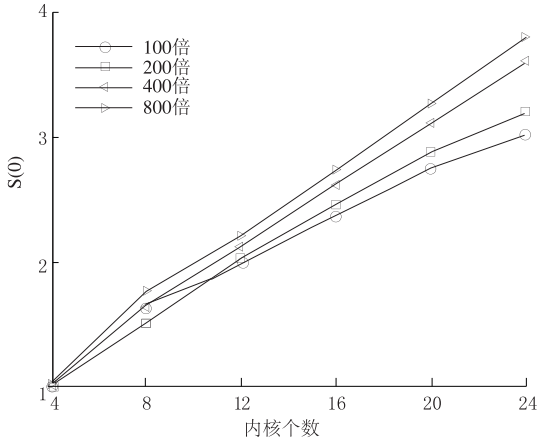

图 8　最优分片下集群加速比

Fig. 8　The acceleration ratio under optimal slicing number

最优的并行化系统的加速效果应呈现线性增长特性,集群节点个数以 n 倍增长时,加速效果也保持 n 倍提升. 但事实上,随着节点个数的增长,节点间通信代价也随着增大,加速效果难以呈现线性增长. 正如图 8 中所示,随着数据规模增大,尤其当数据量极为庞大时,通信代价将被弱化,各节点的计算时间在总时间中将占较大比重,算法的加速效果也将趋于线性增长. 结合具体情况,当数据规模较大时,以最优分片对数据集合进行划分,同时大规模提升集群的节点数量,可以使 ELM 算法的并行度呈线性提升.

5　结论与展望

ELM 算法能解决回归及分类问题,并有训练速度快、泛化性能良好等特点,但在面对日益膨胀的大规模数据时仍亟待并行化扩展. 本论文通过分析 ELM 的运算机制,基于 Spark 并行编程框架,设计并实现了并行 SparkELM 算法,实验表明,SparkELM 算法不仅满足大数据量的计算需求,而且有良好的加速表现. 在未来的工作中,我们将进一步对并行算法进行改进,以充分利用计算机资源. 由于 Spark 仍然在持续升级更新之中,因此使用 API 之前必须明确底层实现机制及版本差异,优化算法设计及实验. 此外,不同于传统的

同时支持规约操作和主从结构的交互式接口(如MPI),Spark仅支持主从结构模式,需要两种交互模式相结合:主节点首先应分配任务,并将重要参数信息传递给从节点;从节点应将计算结果返回给主节点.因此应在设计上谨慎操作以减少算法在此种交互模式下开销,保证算法拥有最优性能,在后续开发中,我们将结合更高性能的Spark版本,对算法进行更新调整.

参考文献:

[1] HUANG G B,ZHU Q Y,SIEW C K. Extreme learning machin:Theory and applications[J]. Neu rocomputing,2006,70(1/3):489 – 501.

[2] HE Q,ZHUANG F Z,LIN J C,et al. Parallel implementation of classification algorithms based on mapreduce[C].//In Proceedings of Rough Sets and Knowledge Technology,2010 ,655 – 622.

[3] VERMA A,LORA X,GOLDBERG D E. ,et al. Scaling genetic algorithms using mapreduce[J],In Proceedings of International Conference on Intelligent Systems Design and Applications,2009,13 – 18.

[4] HUANG G B,ZHOU H,DING X,et al. Extreme learning machine for regression and multiclass classification [J]. Syst Man Cybern Part B Cybern IEEE Trans,2012,42(42):513 – 529.

[5] CAO J, LIN Z, HUANG G B, et al . Voting based extreme learning machine[J]. Inf Sci ,2012,185(1):66 – 77.

[6] LIANG N Y,HUANG G B,SARATHANDRAN P,et al. A fast and accurate on-line sequential learning algorithm for feedforward networks[J]. IEEE Transactions on Neural Network,2010,17(6):1411 – 1423.

[7] SUN Z,LIN T,RISHE N,Large-scale matrix factorization using mapReduce[C]. In Proceedings of IEEE International Conference on Data Mining Workshops,2010,242 – 1248.

[8] ZAHARIA M,CHOWDHURY M,RANKLIN M J,et al. Spark:cluster computing with working sets[C] in Proceedings of the 2nd USENIX conference on Hot topics in cloud computing,2010. 15(1):1765 – 1773.

[9] ODERSKY M,SPOON L,VENNERS B. Programming in Scala[M]. Artima,2008.

[10] ODERSKY M,ALTHERR P,CREMET V,et al. The Scala language specification[M],2008.

[11] PANDA B,HERBACH J S,BASU S,et al. Planet:massively parallel learning of tree ensembles with mapreduce[C].//In:Proceedings of the 35th international conference on very large data bases,2009,2(2):1426 – 1437.

[12] CHU C T,SANG K K,LIN Y A,et al. Map-Reduce for machine learning on multicore[J]. In Proceedings of Advances in Neural Information Processing Systems,2006(19):281 – 288.

[13] FAN R E,CHANG K W,HSIEH C J,et al. LIBLINEAR:A library for large linear classification[J] Journal of Machine Learning Research,2010,9(12):1871 – 1874.

[14] HE Q, SHANG T, ZHUANG F, et al, Parallel extreme learning machine for regression based on mapreduce[J] . Neurocomputing 102(2):52 – 58.

Sparked-based Parallel Extreme Learning Machine

DENG Wanyu, LI Li, NIU Huijuan

(School of Computing Science, Xi'an University of Posts and Telecommunications, Xi'an 710061, China)

Abstract: With the rapid expansion of the scale of the data, stand-alone serial neural networks are faceing normous computational challenges. It is difficult to meet the expansion of real-world applications. In this paper, we proposed a parallel extreme learning machine (parallel ELM) algorithm based on Spark. By leveraging Spark parallel platform with efficient management mechanism and efficient matrix computations of large-scale data, the acceleration for solving ELM is achieved. The learning process of parallel ELM takes only one set of Map and Reduce operation to complete. The experimental results on a large number of real data sets show Spark-based parallel ELM algorithm can achieved a significant performance improvment compared with the serial ELM.

Key words: extreme learning machine; neural network; parallelization ELM algorithm; Spark