2020 年 7 月 第 7 期
Vol. 41 No. 7 2020

MapReduce 和 Spark 两种框架下的大数据极限学习机比较研究

宋丹丹[1]，翟俊海[1,2]，李 艳[1,2]，齐家兴[1]

[1](河北大学 数学与信息科学学院，河北 保定 071002)
[2](河北大学 河北省机器学习与计算智能重点实验室，河北 保定 071002)
E-mail: mczjh@126.com

摘 要： 对 MapReduce 和 Spark 两种框架下的大数据极限学习机进行了比较研究. 具体地，从程序运行时间、任务的同步次数、分类器的泛化性能和需要读写的文件数目 4 个方面进行了比较. 得出了如下结论：1) 在程序运行时间上，基于 Spark 的大数据极限学习机明显优于 MapReduce 的大数据极限学习机，通过理论分析以及对比不同平台的并行指标 speedUp 和 sizeUp 证明了这一结论，而且随着隐含层节点个数的增多，这一优势越发明显；2) 在任务的同步次数上，基于 MapReduce 大数据极限学习机的性能优于基于 Spark 大数据极限学习机；3) 在分类器的泛化性能上，基于 MapReduce 的大数据极限学习机与基于 Spark 大数据极限学习机并无本质的差别；4) 在需要读写的文件数目上，基于 MapReduce 的大数据极限学习机需要读写的文件数目与 Map 任务个数有关，而基于 Spark 的大数据极限学习机需要读写的文件数目与分区数有关. 这些结论对从事相关研究的人员，特别是从事大数据机器学习研究的人员具有较高的参考价值.

关 键 词： 大数据；机器学习；极限学习机；并行计算；任务同步
中图分类号： T181 **文献标识码：** A **文 章 编 号：** 1000-1220(2020)07-1381-08

Comparative Study on Big Data Extreme Learning Machines Under MapReduce and Spark Frameworks

SONG Dan-dan[1], ZHAI Jun-hai[1,2], LI Yan[1,2], QI Jia-xing[1]

[1](College of Mathematics and Information Science, Hebei University, Baoding 071002, China)
[2](Hebei Key Laboratory of Machine Learning and Computational Intelligence, Hebei University, Baoding 071002, China)

Abstract: This paper presents a comparative study on two big data extreme learning machines, the one is based on MapReduce, and the other is based on Spark. Specifically, we compared the two big data extreme learning machines on four aspects: running time of programs, number of synchronization tasks, generalization performance of classifiers, number of files that need to be read or written. The following conclusions are obtained: 1) on running time of programs, Spark based big data extreme learning machine is obviously better than MapReduce based big data extreme learning machine, and this conclusion is proved by theoretical analysis, speedUp and sizeUp. Furthermore, with the increase of the number of hidden layer nodes, this advantage becomes more and more obvious. 2) on number of synchronization tasks, the performance of MapReduce based big data extreme learning machine is better than Spark based big data extreme learning machine. 3) on generalization performance of classifiers, there is no essential difference between MapReduce based big data extreme learning machine and Spark based big data extreme learning machine. 4) on number of files that need to be read or written, the number of files needed to be read and written by the MapReduce based big data extreme learning machine is related to the number of Map tasks. On the other hand, the number of files needed to be read and written by the Spark based big data extreme learning machine is related to the number of partitions. This paper can be very helpful to reseachers in related fields, especifically for the ones engaging in the study of big data machine learning.

Key words: big data; machine learnig; extreme learning machine; parallel computing; task synchronization

1 引 言

极限学习机(Extreme Learning Machine, ELM)是 Huang 等[1]提出的训练单隐含层前馈神经网络的一种算法，它与传统的算法不同，随机生成输入层到隐含层的连接权和隐含层结点的偏置，随后用 Moore-Penrose 伪逆方法求解隐含层到输出层的权值. 研究人员只需手工控制隐含层节点的个数，调整参数的过程无需人工干预，收敛速度快，泛化能力强.

近十年来，ELM 的理论和应用有了很大的发展.

收稿日期：2019-09-10 收修改稿日期：2019-11-06 基金项目：国家自然科学基金项目(71371063)资助；河北省科技计划重点研发项目(19210310D)资助；河北省自然科学基金项目(F2017201026)资助；河北大学研究生创新资助项目(hbu2019ss077)资助；河北省研究生专业学位教学案例库建设项目(KCJSZ2018009)资助. 作者简介：宋丹丹，女，1994 年生，硕士研究生，研究方向为云计算与大数据处理；翟俊海，男，1964 年生，博士，教授，CCF 会员，研究方向为云计算与大数据处理；李 艳，女，1976 年生，博士，教授，研究方向为粒计算与知识获取；齐家兴，男，1995 年生，硕士研究生，研究方向为云计算与大数据处理.

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

Huang 等[2] 证明了 ELM 具有很好的 SLFNs 通用逼近能力,利用常见的激活函数,就能够学习到所有参数,能够得到传统 FNN 的最优泛化边界. Wang 等[3] 利用初始局部误差模型(LGEM)研究了 ELM 的泛化能力. 为应对不同的实际应用,研究人员提出了多种 ELM 的变体. 例如,在 ELM 分类器优化方面,Bai 等[4] 提出了一种稀疏 ELM (S-ELM),用不等式约束代替了传统 ELM 模型中的等式约束,大大减少了存储空间和测试时间. 与支持向量机相似,带有不等式约束的 S-ELM 也会导致二次规划问题. 然而,由于没有涉及偏置项,S-ELM 在训练中比 svm 更有效. 翟等[5] 提出了一种改进灵敏度分析的在线自适应极限学习机算法,该算法引入了新型的计算灵敏度的方法,降低了算法复杂度和训练时间并提高了预测精度. 在集成方面,Lan 等[6] 提出了一种在线序列 ELM (EOS-ELM) 集成,该集成以多个独立训练的 OS-ELM 的平均预测作为最终预测. EOSELM 进一步提高了 OS-ELM 的预测精度,实现了 ELM 算法在线序列数据方面的扩展. 翟等[7] 提出了一种用模糊积分集成重复训练极限学习机的数据分类方法,与其他集成 ELM 不同,该方法能很好地刻画 ELM 基本分类器之间的交互作用,而且具有更好的泛化性能. 处理噪声或缺失数据方面,Horata 等[8] 提出了三种改进的 ELM 算法: 1) 基于迭代重加权最小二乘(IRWLS-ELM) 的 ELM 算法; 2) 基于多元最小边缘平方(mlt -ELM) 的 ELM; 3) 基于一步重加权 mlt (rmlt -ELM) 的 ELM. 在 IRWLS 中,通过 M-estimate 函数对训练数据进行迭代加权,通过降低异常值的权重,可以逐渐减小异常值的影响. 在 MLTS-ELM 和 RMLTSELM 中,引入了基于最小协方差行列式(MCD) 估计的多元最小二乘估计(MLTS),提高了 ELM 对异常值的鲁棒性. Man 等[9] 提出了一种基于有限脉冲响应的 ELM 模型,模型的输入权用有限脉冲响应滤波器方法确定,提高了 ELM 对噪声数据鲁棒性. 在无监督学习方面,Huang 等[10] 将 ELM 推广到半监督和无监督学习的场景,提出了半监督 ELM 和无监督 ELM,极大地扩展了 ELM 的适用性. Heeswijk 等[11] 提出了一种基于 GPU 加速的并行 ELM 集成方法,并用于解决大规模回归问题,取得了良好的效果. 在 ELM 算法的表示学习方面,Kasun 等[12] 提出了一种基于 ELM 的表示学习的自编码器,该自编码器使用 ELM 学习的自编码器进行分层表示学习,形成了多层前馈网络. 实验结果表明,该方法比深度信念网络和深度玻尔兹曼机要快几个数量级,所获得的精度与深度学习算法相比具有很强的竞争力. 在应用方面,An 等[13] 提出了一种基于 ELM 的高效图像超分辨率方法,其目标是对输入的低分辨率图像进行处理,以生成高分辨率图像. 张和李[14] 提出了深度极限学习机的立体图像质量评价方法,该方法将立体图像数据经过稀疏自编码器预训练抽取图像特征,再输入 ELM 分类器进行训练,分类器的输出即为质量评分,提升了该模型的准确性和稳定性. 徐等[15] 提出了一种融合加权 ELM 和 Adaboost 的交通标志识别算法,该算法迭代更新训练样本权重,将 ELM 分类器作为 AdaBoost 的弱分类器,最终构造出最优的强分类器. 该方法的优点是能够实现对训练样本和弱分类器的双重加权,提高了交通标志的识别精度. 黄和王[16] 提出了结合超限学习机和融合卷积网络的 3D 物体识别方法,该方法采用多层融合卷积网络提取特征,用半随机的 ELM 网络进行分类. 与现有的卷积神经网络相比,有更好的效果和更短的训练时长.

针对大数据,传统的 ELM 算法已不能适应大数据环境的需要,针对这一问题,He 等[17] 提出了基于 MapReduce 的 ELM. 针对大规模图数据分类问题,Sun 等[18] 提出了一种基于分布式 ELM 的分类方法,提出的方法具有好的扩展性,而且易于实现. Yao 和 Ge[19] 提出了一种分布式并行 ELM 和一种层次 ELM,解决了用大数据进行多模式质量预测问题,取得了良好的效果. 为了解决 ELM 在大数据环境中的可扩展性问题,Ming 等[20] 提出了两种 ELM 并行化变体. 一种是基于局部数据的模型并行化 ELM,另一种是基于全局数据的模型并行化 ELM. 基于开源大数据平台 Flink,Chen 等[21] 提出了一种基于 GPU 加速的并行化层次 ELM,并采用几种优化方法提高模型的并行性和可伸缩性. Duan 等[22] 提出了基于 Spark 的 ELM. 本文对基于 MapReduce 的 ELM 和 Spark 的 ELM 进行全面的比较研究,得出了一些有价值的结论.

2 基础知识

2.1 MapReduce

MapReduce 是由谷歌最先提出,用于完成并行式计算的一种工具,其主要思想是“分而治之”,将要处理的数据随机发送至集群上的各个节点上,这些数据存放于集群的分布式文件系统 HDFS(Hadoop Distributed File System) 上. 各个节点并行地处理在该节点上的数据,然后对各个节点的中间结果加以处理,输出处理结果. 用户主要是通过 MapReduce 中的两个基本函数 Map 和 Reduce 来完成具体的大数据处理逻辑. 在 Map 阶段将数据根据具体要求逐条处理,并以键值对的形式传送给 Reduce 阶段. Reduce 阶段将键相同的值合并以文件的形式输出结果.

2.2 Spark

Spark 是一种用于内存计算的大数据处理框架. 它的设计理念中最重要的一点是改善大数据并行运算时网络数据流量承载过重和磁盘 I/O 开销过大的问题. 在 Spark 中,弹性分布式数据集 RDD(Resilient Distributed Dataset) 是最重要的组成成分之一. RDD 是一个抽象的数据结构,待处理的数据均被高度抽象至 RDD 中. 从物理层面上分析,RDD 将待处理的数据散布在集群的各个节点上,存储于内存或外存上,通过唯一的标识符对其进行操作; 从逻辑层面上分析,待处理的数据块根据实际操作划分分区数. RDD 通过算子操作分区,完成转换或行动操作,形成新的 RDD. 在 Spark 平台上的一切操作都是对 RDD 的操作. 这些操作统分成两大类,被称为 Transformation(转换) 操作和 Action(行动) 操作. RDD 中所有的转换操作都不会直接计算结果,都是延迟加载的操作. 它们只是记住这些应用到基础的数据集(例如一个文件) 上的转换操作. 只有当发生一个要求返回给 Driver 的动作时,这些转换操

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

作才会真正运行,这种设计使 Spark 更加有效率. MapReduce 和 Spark 的差异性如表 1 所示.

表 1　MapReduce 和 Spark 平台特性对比

Table 1　Comparison of features of MapReduce and Spark platform

平台	磁盘 IO 操作	处理方式	处理单元
MapReduce	生成中间文件,IO 操作较频繁	逐条处理	一条数据
Spark	基于内存计算,IO 操作较少	分区处理	RDD

3　ELM 的并行化及其比较研究

本节重点介绍如何用 MapReduce 和 Spark 两种平台实现 ELM 的并行,并对这两种实现进行对比分析.

3.1　ELM 算法

在 ELM 中,隐含层节点的权重和偏置值被随机初始化,不存在迭代调优的过程. ELM 算法中的目标是计算隐含层到输出层之间的权重矩阵 β,计算公式为 $\beta=\left(\frac{I}{\gamma}+H^TH\right)^{-1}H^TY$,$U=H^TH$,$V=H^TY$(其中 H 是隐含层节点输出矩阵,Y 是类别矩阵) . 从上述计算公式可以看出,H,U,V 三个矩阵的计算是主要计算任务,而这三个矩阵的计算能够并行化. 下面分别介绍基于 MapReduce 和 Spark 的并行化 ELM.

3.2　基于 MapReduce 的并行化 ELM

用 MapReduce 实现 ELM 的并行化,关键是 Map 函数和 Reduce 函数的设计. 算法 1 和算法 2 是计算 U,V 两个矩阵的 Map 函数和 Reduce 函数.

算法 1. Map 函数

输入: $X=\{(x_i,y_i)\mid x_i\in R^n,y_i\in R^m,i=1,2,\cdots,N\}$,$w_j\in R^n$,$b_j\in R$,$j=1,2,\cdots,L$.

输出: 输出中间子矩阵 U 和 V.

```
1. for(i =1; i <=L; i =i +1) do
2.    H [i] = g(x × w_i + b_i)
3. end
4. for(i =1; i <=L; i =i +1) do
5.    for(j =1; j <=L; j =j +1) do
6.       u [i,j] = H_i^T × H_j
7.       context. write(triple("U",i,j),u [i,j])
8.    end
9.    for(j =1; j <=m; j =j +1) do
10.      v [i,j] = H_i^T × Y_j
11.         context. write((triple("V",i,j),v [i,j])
12.   end
13. end
```

算法 2. reduce 函数

输入: 键值对 <triple1,U 的子矩阵 u > , <triple2,V 的子矩阵 v > .

输出: 矩阵 U 和矩阵 V.

1. 键值相同的子矩阵 u 求和,得到最终结果矩阵 U;
2. 键值相同的子矩阵 v 求和,得到最终结果矩阵 V;
3. 输出 U 和 V.

以 MapReduce 处理两条数据为例,详细的数据处理流程图 1 所示.

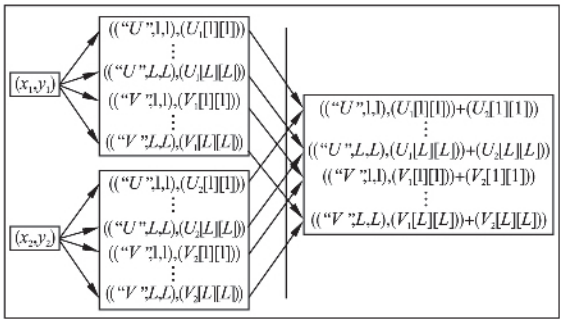

图 1　基于 MapReduce 的 ELM 数据处理流程示意图

Fig. 1　Schematic diagram of data processing process by MapReduce based ELM

其中,

$$H_1=[w_1x_{11}+b_1,\cdots,w_Lx_{1L}+b_L] \tag{1}$$

$$H_2=[w_1x_{21}+b_1,\cdots,w_Lx_{2L}+b_L] \tag{2}$$

$$U_1=\begin{bmatrix}H_1[1]H_1[1] & \cdots & H_1[1]H_1[L]\\ \vdots & \ddots & \vdots\\ H_1[L]H_1[1] & \cdots & H_1[L]H_1[L]\end{bmatrix} \tag{3}$$

$$U_2=\begin{bmatrix}H_2[1]H_2[1] & \cdots & H_2[1]H_2[L]\\ \vdots & \ddots & \cdots\\ H_2[L]H_2[1] & \cdots & H_2[L]H_2[L]\end{bmatrix} \tag{4}$$

$$V_1=\begin{bmatrix}H_1[1]y_1[1] & \cdots & H_1[1]y_1[m]\\ \vdots & \ddots & \vdots\\ H_1[L]y_1[1] & \cdots & H_1[L]y_1[m]\end{bmatrix} \tag{5}$$

$$V_2=\begin{bmatrix}H_2[1]y_2[1] & \cdots & H_2[1]y_2[m]\\ \vdots & \ddots & \vdots\\ H_2[L]y_2[1] & \cdots & H_2[L]y_2[m]\end{bmatrix} \tag{6}$$

基于 MapReduce 的 ELM 算法执行流程如下:

1) 将隐含层节点参数放置在集群缓存中;

2) 在 Map 节点上执行的操作:

将数据 x 逐条输入至 Map 任务中,与缓存中的数据计算得出隐含层节点输出向量 H;

计算 U 的子矩阵;

计算 V 的子矩阵;

将 U 和 V 的子矩阵按照键值对的形式输出至 Reduce 节点;

3) 在 Reduce 节点上执行的操作:

将接收到的所有的 U 的子矩阵 u 按照矩阵加法得出新矩阵 U;

将接收到的所有的 V 的子矩阵 v 按照矩阵加法得出新矩阵 V;

输出 U 和 V 矩阵.

3.3　基于 Spark 的并行化 ELM

用 Spark 实现 ELM 的并行化,关键是 RDD 的设计,用各种 RDD 处理数据的流程如图 2 所示.

根据图 2,可以看出基于 Spark 的 ELM 执行流程:

1) 第一次遍历: 生成 RDD 并记录依赖关系

读取外部数据生成第一个 RDD(TextFileRDD) ; 然后执

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

行 Map 转换操作对数据进行初步处理，将数据的类型转换成数组，得到 MappedRDD，再执行 Filter 转换操作，剔除格式错误的数据(例如缺省值的情况)，生成 FilteredRDD，再将数据格式转换生成 PairRDD，读成分布式行矩阵的形式，执行其内部自定义函数 computeGramianMatrix 和 multipy，具体算子如下：先由 mapPartition 操作计算分区中得到 mapPartitionRDD，再由转换操作 ReduceByKey 得到 shuffledRDD，然后再次执行 mapPartition 操作得到新的 mapPartitionRDD. 所有的转换操作完毕，即在该次作业中所有 RDD 生成完毕，遍历结束.

2) 第二次遍历：划分阶段

根据第一次遍历中 RDD 记录的依赖关系，从最后一次执行转换操作 MapPartitionRDD 向前执行，遇到窄依赖就将其并入阶段内部，遇到宽依赖就形成一个新的阶段继续遍历，直至向前遍历至 TextFileRDD 为止. 如图 2 所示，划分成了两个阶段.

3) 第三次遍历：按阶段顺序执行任务

由第二次遍历可知，划分出了两个阶段，顺序执行即可.

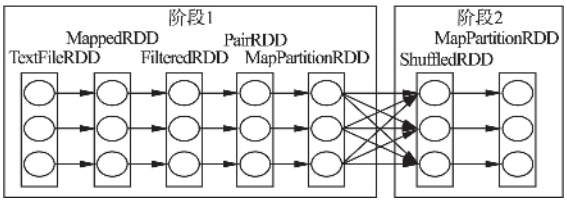

图 2 基于 Spark 的 ELM 数据处理流程示意图

Fig. 2 Schematic diagram of data processing process by Spark based ELM

与基于 MapReduce 的并行 ELM 类似，在基于 Spark 的 ELM 中，关键也是计算矩阵 H、U 和 V. 计算 H 矩阵的流程如下：将样例集合 X 经过预处理读成 RowMatrix 的形式，将随机初始的矩阵 w 读成 Matrix 的形式，执行 RowMatrix 中的函数操作：def multipy(B: Matrix): RowMatrix 即可求得 $X[i] \times w[j]$，该函数的具体实现过程如算法 3 所示. 计算 U 矩阵的流程如下：在缓存中读取 H 矩阵，经过数据预处理，读成 RowMatrix 的形式，执行 RowMatrix 中的函数操作：def computeGramianMatrix(): Matrix 得到 $H_i^T \times H_j$，具体算法实现过程如算法 4 所示. 计算 V 矩阵的 RDD 转换流程如下：在缓存中读取 H 矩阵，经过预处理读成 RowMatrix 的形式，将标签矩阵 Y 读成 Matrix 的形式，执行 RowMatrix 中的函数操作：def multipy(B: Matrix): RowMatrix 得到 $H_i^T \times H_j$，具体算法实现过程如算法 5 所示.

算法 3. 计算 H 矩阵

输入： 训练集 $X = \{(x_i, t_i) \mid x_i \in R^n, t_i \in R^m, i = 1, 2, \cdots, N\}$，隐含层结点的权值和偏置 $w_j \in R^n, b_j \in R, j = 1, 2, \cdots, L$.

输出： 隐含层节点输出矩阵 H.

```
1. 将训练集 X 按行划分成 L 个分区，隐含层权重矩阵 W 按列划分成 L 个分区;
2. //矩阵初始化;
3. value = 0;
4. for(i = 1; i < = L; i = i + 1) do
5.    for(j = 1; j < = L; j = j + 1)
6.    //计算 x[i] × w[j];
7.    value = ∑_{z=1}^{n}(x[i][z] × w[z][j] + b[j]);
8.    end
9. end
```

隐含层节点权重矩阵 w 按列分成 L 个分区，每一个完整的隐含层节点参数独立地放在一个分区中. 也就是说，w 的每个分区缓存相对应的隐含层节点的所有参数，为了有更多的本地计算，将待训练的数据集 X 按行分为 L 个分区，对于 X 来说，这将保证在每个分区中有 1 个或多个完整的训练数据；对于 w 来说，可以保证每个分区有 1 个完整的隐含层节点参数. 计算矩阵 H 中元素的值时，只涉及到公式中训练样例所在的分区和对应隐含层节点参数所在的分区数，减少了网络的传输. 计算结果是隐含层节点输出矩阵 H，将其缓存在 L 个分区的内存中.

算法 4. 计算 $\overline{U}$ 矩阵

输入： 矩阵 H，主对角线均为 γ 的对角矩阵 $\gamma_{L \times L}$.

输出： 矩阵 $\overline{U}$.

```
1. 生成矩阵 γ_{L×L};
2. //矩阵初始化;
3. outValue = 0;
4. //遍历 H 的每一个分区;
5. for(i = 1; i < = L; i = i + 1) do
6.    //遍历 H 的每一个分区;
7.    for(j = 1; j < = L; j = j + 1) do
8.       //计算 H_i^T × H_j;
9.       outValue = ∑_{z=1}^{N} value[z][i] × value[z][j];
10.   end
11. end
12. //矩阵求和;
13. outValue = outValue + γ_{L×L};
14. 输出 Ū.
```

矩阵 H 分布在 L 个分区中，由 $U = H^T H$ 可知，矩阵 U 的计算可以解释为：U_{ij} 的求解相当于是矩阵 H 第 i 个分区与矩阵 H 第 j 个分区存放的数据做向量内积运算，其中 $i = 1, 2, \cdots, L; j = 1, 2, \cdots, L$，即可得出得到矩阵 U，此时 U 是一个 $L \times L$ 的小规模矩阵，与 $\gamma_{L \times L}$ 做矩阵加法即可得出 $\gamma_{L \times L}$ 的值.

算法 5. 计算矩阵 V

输入： 矩阵 H，训练集类别矩阵 $Y_{N \times m}$.

输出： 矩阵 V.

```
1. 将 Y_{N×m} 按列划分成 m 个分区;
2. //矩阵初始化;
3. outValue = 0
4. //遍历 H 的每一个分区;
5. for(i = 1; i < = L; i = i + 1) do
6.    for(j = 1; j < = m; j = j + 1) do
7.    //计算 H_i^T × Y_j;
8.       outValue = ∑_{z=1}^{N} value[z][i] × Y[z][j];
9.    end
10. end
11. 输出 V.
```

矩阵 H 分布在 L 个分区中，由 $V = H^T Y$ 可知，矩阵 V 的计算可以解释为：V_{ij} 的求解相当于是矩阵 H 第 i 个分区与矩阵 Y 第 j 个分区存放的数据做向量内积运算，其中 $i = 1, 2, \cdots, L; j = 1, 2, \cdots, m$，即可得出得到矩阵 V 的值.

3.4 基于 MapReduce 和 Spark 的并行 ELM 的比较

3.4.1 两种算法实现的相同点

ELM 算法最重要的就是计算出隐含层与输出层之间的参数矩阵 β，通过对 ELM 算法的研究分析，矩阵 β 的计算可

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

分解为 $\beta=\left(\frac{I}{\gamma}+H^TH\right)^{-1}H^TY, U=H^TH, V=H^TY$. 其中,隐含层输出矩阵 H、中间矩阵 U 和 V 均可以在 MapReduce 和 Spark 框架下实现并行计算.

3.4.2　两种算法实现的不同点

基于 MapReduce 机制计算矩阵 $U_{L\times L}$ 时,是对于每一个样例 x 经过与隐含层节点矩阵 H 计算后得到一个 $L\times L$ 的子矩阵 U_i,将所有样例(即 N 个样例)经过计算后得到的 N 个 $L\times L$ 子矩阵 $U_1, U_2, \cdots, U_N$ 进行矩阵加法运算,结果即为矩阵 U. 此时,存在中间结果的输出,即存在内存→硬盘→内存的数据流动,网络开销不可避免,同时存在子矩阵的叠加过程,计算 V 矩阵同上; MapReduce 并没有生成完整的 H 矩阵,而是逐条处理数据,这是由自身机制造成的; 一次 MapReduce 操作就算出了 U 和 V 矩阵,有较强的并行力度.

基于 Spark 机制计算矩阵 $U_{L\times L}$ 时,这个矩阵中所有元素均是对应位置分区里存放元素的向量内积运算,在此过程中只涉及两个分区内部元素的相互作用,网络开销较少,且求得即为最终结果,不存在子矩阵加和的情况,计算矩阵 V 同上; 隐含层节点输出矩阵 H 在 Spark 平台上是并行完成的,从逻辑上讲,生成了完整的矩阵 H,从物理层面上讲,矩阵 H 中的元素散落在各个分区中; 矩阵 U 和 V 模块内部是并行执行的,模块间串行操作,并行度相对较弱.

4　实验结果及分析

4.1　实验结果对比分析

4.1.1　并行 ELM 算法在 MapReduce 和 spark 平台上执行时间对比

在 2 个 UCI 数据集和 3 个人工数据集上,对基于 MapReduce 的 ELM 和基于 Spark 的 ELM 进行了对比分析研究. 其中三个人工数据集均由高斯分布产生,第一个人工数据集是二类数据集,服从的概率分布是:

$$p(x|w_1)\sim N\left(\begin{bmatrix}1.0\\1.0\end{bmatrix},\begin{bmatrix}0.6 & -0.2\\-0.2 & 0.6\end{bmatrix}\right)$$

$$p(x|w_2)\sim N\left(\begin{bmatrix}2.5\\2.5\end{bmatrix},\begin{bmatrix}0.2 & -0.1\\-0.1 & 0.2\end{bmatrix}\right)$$

第二个人工数据集是三类数据集,服从的概率分布是:

$$p(x|w_1)\sim N\left(\begin{bmatrix}0\\0\end{bmatrix},\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right)$$

$$p(\boldsymbol{x}|w_2)\sim N\left(\begin{bmatrix}1\\1\end{bmatrix},\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right)$$

$$p(\boldsymbol{x}|w_3)\sim \frac{1}{2}N\left(\begin{bmatrix}0.5\\0.5\end{bmatrix},\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right)+\frac{1}{2}N\left(\begin{bmatrix}-0.5\\0.5\end{bmatrix},\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}\right)$$

第三个人工数据集是四类数据集,服从的概率分布是:

$$p(\boldsymbol{x}|w_1)\sim N\left(\begin{bmatrix}0\\0\\0\end{bmatrix},\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}\right)$$

$$p(\boldsymbol{x}|w_2)\sim N\left(\begin{bmatrix}0\\1\\0\end{bmatrix},\begin{bmatrix}1 & 0 & 1\\0 & 2 & 2\\1 & 2 & 5\end{bmatrix}\right)$$

$$p(\boldsymbol{x}|w_3)\sim N\left(\begin{bmatrix}-1\\0\\1\end{bmatrix},\begin{bmatrix}2 & 0 & 0\\0 & 6 & 0\\0 & 0 & 1\end{bmatrix}\right)$$

$$p(\boldsymbol{x}|w_4)\sim N\left(\begin{bmatrix}0\\0.5\\1\end{bmatrix},\begin{bmatrix}2 & 0 & 0\\0 & 1 & 0\\0 & 0 & 3\end{bmatrix}\right)$$

该实验是在拥有 1 个主节点和 7 个从节点的云计算平台上进行的,在云计算平台上搭建了 Hadoop 和 Spark 两个运行环境,云计算平台节点的基本配置信息和功能规划如表 2 和表 3 所示,主机的基本配置如表 4 所示,实验所需数据集的基本信息如表 5 所示. 实验将从训练数据维度、训练数据规模和隐层节点个数三个方面作为实验参数对含有 10、20、30、50、100 个隐含层节点的 ELM 网络模型,比较了基于 MapReduce 和 Spark 的并行 ELM 算法的执行时间,实验结果列于表 6 ~ 表 10 中. 对实验结果进行分析,得出了结论: 基于 Spark 的

表 2　云计算平台节点的基本配置信息

Table 2　Basic configuration of the nodes of the cloud compution platform

配置项目	配置情况
处理(CPU)	Inter Xeon E5-4603 2.0GHz (双核)
内存	8GB RDIMM
硬盘	1TB
网卡	Broadcom 5720 QP 1Gb 网络子卡
网络设备	华为 S3700 系列以太网交换机
操作系统	Ubuntu kylin 13.04
云计算平台	Hadoop-2.7.3,Spark-2.1.0
JDK 版本	Jdk-7u71-linux-i586
Scala 版本	Scala-2.11.0

表 3　云平台的节点功能规划

Table 3　Role assignment of the nodes of the cloud computing platform

主机名	IP 地址	节点类型
master	10.187.86.242	NameNode,Master,ResourceManager
node1	10.187.86.243	DataNode,Worker,NodeManager
node2	10.187.86.244	DataNode,Worker,NodeManager
node3	10.187.86.245	DataNode,Worker,NodeManager
node4	10.187.86.246	DataNode,Worker,NodeManager
node5	10.187.86.247	DataNode,Worker,NodeManager
node6	10.187.86.248	DataNode,Worker,NodeManager
node7	10.187.86.249	DataNode,Worker,NodeManager

表 4　主机基本配置信息

Table 4　Basic configuration of the host

配置项目	配置情况
处理器(CPU)	Intel Xeon CPU E5-1650 3.5GHz
内存	16GB
硬盘	500G
操作系统	Win10
JDK 版本	jdk-8u144-windows-x64
MapReduce 程序环境	Eclipse-Phtotn-Releae-4.8.
Scala 版本	Scala-2.11.0
Spark 程序环境	Eclipse-4.7.0

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

ELM 算法的效率远远高于基于 MapReduce 的效率,随着隐含层节点个数逐渐增多,Spark 平台的优势愈加明显. 在下一节中,我们将从理论层面分析产生该现象的原因.

表 5 实验数据集

Table 5 Basic information of data sets

数据集	类别数	属性数	节点类型
Gauss1	2	2	1000000
Gauss2	3	2	1200000
Gauss3	4	3	1000000
Covtype	7	54	581012
Skin	2	3	245057

表 6 两种算法在 Gauss1 数据集上实验结果的对比(单位: s)

Table 6 Comparison of experimental results of two algorithms on data set Gauss1 (s)

隐结点个数	Spark ELM	MapReduce ELM
10	184	423
20	346	845
30	501	1923
50	969	6076
100	1856	73696

表 7 两种算法在 Gauss2 数据集上实验结果的对比(单位: s)

Table 7 Comparison of experimental results of two algorithms on data set Gauss2(s)

隐结点个数	Spark ELM	MapReduce ELM
10	226	230
20	419	1993
30	607	4794
50	1065	6025
100	2235	71458

表 8 两种算法在 Gauss3 数据集上实验结果的对比(单位: s)

Table 8 Comparison of experimental results of two algorithms on data set Gauss3(s)

隐含层个数	Spark ELM	MapReduce ELM
10	195	205
20	353	690
30	518	1478
50	884	4205
100	1900	32442

4.1.2 并行 ELM 算法在 MapReduce 和 Spark 平台上 speedUp 指标和 sizeUp 指标对比

以 gauss3 数据集的 0.1 倍作为本实验的原始数据集,我们将对基于 MapReduce 和 Spark 平台的 ELM 算法进行并行性能的评估. 实验中采取的训练数据集分别是原始数据集的 5 倍、8 倍和 10 倍,同时在并行度为 7、14、28 的平台上运行隐层节点个数为 10、20、30 的 ELM 并行算法,根据 speedUp 指标和 sizeUp 指标对两种大数据平台进行比较说明,实验结果如图 3 和图 4 所示.

speedUp(m) = 程序运行在 1 台设备上的执行时间/程序运行在 m 台设备上的执行时间

sizeUp(m) = 程序处理 m 倍的数据集执行时间/程序处理原始数据集执行时间

表 9 两种算法在 Skin 数据集上实验结果的对比(单位: s)

Table 9 Comparison of experimental results of two algorithms on data set Skin(s)

隐结点个数	Spark ELM	MapReduce ELM
10	56	280
20	89	1167
30	133	2797
50	215	8638
100	435	43432

表 10 两种算法在 covertype 数据集上实验结果的对比(单位: s)

Table 10 Comparison of experimental results of two algorithms on data set covertype(s)

隐结点个数	Spark ELM	MapReduce ELM
10	76	85
20	231	248
30	320	544
50	508	1671
100	1064	9704

图 3 是隐层节点分别是 10、20、30,并行度分别是 7、14、28 的 ELM 算法在大数据平台 MapReduce 和 Spark 平台上的

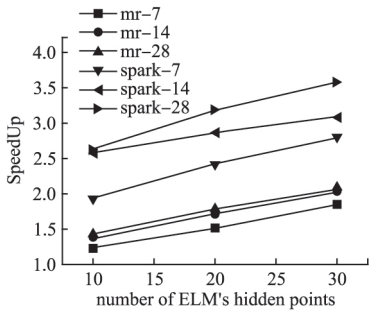

图 3 大数据平台的 SpeedUp 性能对比

Fig. 3 SpeedUp performance comparison

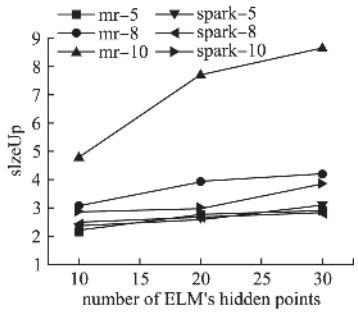

图 4 大数据平台的 sizeUp 性能对比

Fig. 4 SizeUp performance comparison

对比情况,结果显示: 基于 Spark 平台的 ELM 算法的 speedUp 值总体高于基于 MapReduce 平台的 ELM 算法的 speedUp 值,随着隐层节点数的增多,这种差异更加突出. speedUp 折线图

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

近似于一条直线,且斜率有减小的趋势这是因为并行计算将不可避免的增加在不同机器上的数据交换,造成时间的浪费. 图 4 是隐层节点分别是 10、20、30,训练数据集分别是原始数据集的 5 倍、8 倍、10 倍. 由图 4 可知,基于 Spark 平台的 ELM 算法的 sizeUp 值总体低于基于 MapReduce 平台的 ELM 算法 sizeUp 值,也就是说,训练的数据集规模越大时,Spark 平台上数据处理效率越突出. 这是因为 Spark 是一种基于内存计算的大数据平台,中间结果可以缓存至内存中,且只有 RDD 的行动算子才会触发计算,相较于 MapReduce 平台,不存在中间数据流动至 HDFS 的情况,数据 IO 操作较少,但在 MapReduce 平台上随着隐层节点增多时,在算法执行过程中会存在更大规模的矩阵运算,这将导致更大规模的数据 IO 操作,造成严重耗时.

4.2　理论分析

在本节中,针对 MapReduce 和 Spark 这两种大数据框架,受文献 [23] 的启发,我们从算法执行时间、同步次数、读写文件数目和分类器的泛化性能 4 个方面进行比较和分析.

4.2.1　算法执行时间对比分析

基于大数据平台的 ELM 算法执行时间 T 为: $T = T_r + T_s + T_t + T_w$(其中 T_r 表示文件读取时间,T_s 表示中间数据排序时间,T_t 表示中间数据传输时间,T_w 表示文件写出时间) . 其中,T_r = 文件读速度 × 输入文件大小; T_w = 文件写速度 × 输出文件大小.

基于不同大数据框架下的并行 ELM 运行机制,导致程序执行时间的差异. 不同的只是大数据处理平台,而程序的输入数据和输出数据、网络传输速度以及文件读写速度在实验中作为常量存在,其差异造成的影响忽略不计. 此时 T_r 和 T_w 在这两种大数据平台下的 ELM 算法中的值近似相同,其差异并不是本文中要研究的重点,在此不进行详细分析. 主要对 T_s 和 T_t 在两个大数据平台下的差异进行精要阐述说明.

a) 基于大数据平台的 ELM 算法 Ts 中间数据排序时间的比较

对于 Spark 平台来说,不存在中间数据排序这一过程,故而:

$$T_{s\text{-}spark} = 0$$

而对于 MapReduce 平台来说,每次执行 shuffle 操作时,对中间数据的排序过程是必不可少的,目的是通过对中间数据的排序,实现初步的归并操作这一步骤,确保一个 Map 任务最终只有一个有序的中间数据文件输出,缓解网络传输压力,具体过程如下: 在 Map 阶段,对每一分区的数据分别进行排序,若共有 P 个 Map 任务,每个 Map 任务负责处理 p 条数据,采用快速排序的机制进行排序,则:

$$T_{s\text{-}mr} = \mathrm{Plog_p}$$

此时:

$$T_{s\text{-}mr} > T_{s\text{-}spark}$$

b) 基于大数据平台的 ELM 算法 T_t 中间数据传输时间的比较

在中间数据传输时间的比较中,我们主要从数据传输数量和数据存放位置两方面进行比较说明. 基于 MapReduce 的 shuffle 机制,在 Map 任务结束后传至 Reduce 端之前,会存在文件合并的情况,这是 MapReduce 独有的特色,因此当 MapReduce 和 Spark 平台处理相同的数据时,MapReduce 平台的中间数据传输量会小于 Spark 的中间数据传输量. 当传输速率相同时,则存在:

$$T_{t\text{-}mr\text{-}Files} < T_{t\text{-}spark\text{-}Files}$$

Spark 是一种基于内存计算的大数据平台,即所有的操作都是基于 RDD 的操作,中间数据也是以 RDD 的形式在内存中运算并存储,这是由 Spark 自身的机制决定的. MapReduce 从硬盘上读取数据,经过 Map 计算后产生的中间数据以文件的形式存储于分布式文件系统 HDFS 上,然后再从 HDFS 上读取并进行 Reduce 计算. 显而易见,这比 Spark 的中间数据传输时间要长得多. 所以:

$$T_{\text{t-mr-io}} > T_{\text{t-spark-memory}}$$

HDFS 读取操作的时长远远大于内存操作,因此:

$$T_{t\text{-}mr\text{-}Files} + T_{t\text{-}mr\text{-}io} > T_{t\text{-}spark\text{-}Files} + T_{t\text{-}spark\text{-}memory}$$

综上所述,可得:

$$T_{t\text{-}mr} > T_{t\text{-}spark}$$

根据以上两方面的分析,可得出结论 ELM 算法在 MapReduce 平台上的运行时间大于在 Spark 平台上的运行时间.

4.2.2　同步次数对比分析

同步操作要求当前阶段中的所有节点都执行完毕才开始下一阶段. 在 MapReduce 框架下,所有的 Map 节点都完成之后,Reduce 阶段才开始执行,这是典型的同步操作. 一次 MapReduce 操作,记一次同步. 根据前面的分析可知,在基于 MapReduce 框架下的 ELM 算法,并行操作主要是进行矩阵 H、矩阵 U 和矩阵 V 的运算,它主要是通过一次 MapReduce 操作实现了这三个矩阵的计算,因此,同步次数是 1.

由前面基于 Spark 框架下的 ELM 算法流程可知,首先并行计算出矩阵 H,矩阵 H 计算完毕后,执行矩阵 U 和矩阵 V 的计算,这二者的计算可同时执行,因此算法实现过程中的同步次数是 2.

4.2.3　读写文件数目对比分析

在 MapReduce 框架下,由于存在分区内部排序操作,分区间的数据用偏移量标记,可以将每一个 Map 中不同分区间的数据汇总成一个大的中间数据文件,即一个 Map 对应一个中间文件. 所以在 MapReduce 中,读写文件数目是 Map 任务的数量. 但在 Spark 平台上,RDD 是以分区为单位并行执行的,所以在 Spark 中,读写文件数目取决于 RDD 的分区数. 尽管并行 ELM 算法在这两个大数据平台上的具体实现有所差异,但这并不是导致读写文件数目不同的原因,这是由大数据平台自身的机制造成的. 在 MapReduce 中,待处理的数据集逐条输入至 Map 任务中,在每个 Map 任务中,根据实际需求生成一个大的数据文件,写入 HDFS 中; Reduce 节点再从 HDFS 中读取这些文件,根据不同的需求执行不同的操作. 在基于 MapReduce 框架下的 ELM 算法中,并行操作主要是进行矩阵 H、U 和 V 的运算,它通过一次 MapReduce 操作实现了这三个矩阵的计算,读写文件数目等于 Map 任务数目,读写文件数目只与该算法执行过程中 Map 任务数目有关; 在 Spark 平台上的一切操作都是对 RDD 的操作,以图 2 为例,数据划分为 3 个分区进行并行处理,每个分区对应一份数据文件,数据经过 TextFileRDD 后,将数据分成 3 份,读入 3 个文件中进行并行操作,当数据经过 mapPartitionRDD 后,同一个

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

文件中的数据会进入 shuffleRDD 中不同的分区,这就会导致 mapPartitionRDD 每个分区中的一个数据文件拆分 3 个子数据文件,分别对应 shuffleRDD 中不同的分区,实现文件的 reduceByKey 操作,此时读写文件数目为 $3+3+3+3+3\times3+3=24$. 其中,分区数 3 是可手动控制的超参数,如当分区数重置为 4 时,文件读写数目 $=4+4+4+4+4\times4+4=36$. Spark 平台读写文件数目只与 RDD 中的分区数有关.

4.2.4 分类器的泛化性能对比分析

基于 MapReduce 和 Spark 平台的并行 ELM,使用了相同的方法计算从隐含层到输出层之间的参数矩阵 β,即在这两个大数据平台下,计算两个中间结果矩阵 U(即 H^TH)和矩阵 V(即 H^TY),当所有的相关参数均相等时,用该算法对相同的训练数据训练分类器,显而易见,训练出的分类器也是相同的,它们的泛化性能一致. 因此在这两个平台下,相同的测试数据会得到相同的精度.

5 结束语

本文通过对基于 MapReduce 和 Spark 平台的两种并行 ELM 进行比较研究,得出如下结论: 1) 从算法实现层面上看,这两个大数据平台均并行计算了矩阵 H、U 和 V,计算过程大致相同; 2) 基于 Spark 的 ELM 的执行效率远远高于基于 MapReduce 的 ELM 的执行效率,而且从大数据平台理论层面和并行指标 speedUp,sizeUp 证明了这一结果的必然性; 3) 基于 MapReduce 的 ELM 在同步次数方面优于 Spark 平台的 ELM; 4) 从读写文件数目来看,基于 MapReduce 的 ELM 需要读写文件数目与 Map 任务个数有关,而基于 Spark 的 ELM 读写文件数目与分区数有关; 5) 从分类器泛化能力来看,两种并行 ELM 并无差别.

References:

[1] Huang G B,Zhu Q Y,Siew C K. Extreme learning machine: a new learning scheme of feedforward neural networks [C] //Proceedings of International Joint Conference on Neural Networks (IJCNN2004), Budapest,Hungary,25-29 July,2004,2: 985-990.

[2] Huang G B,Zhou H,Ding X,et al. Extreme learning machine for regression and multiclass classification [J]. IEEE Transactions on Systems,Man,and Cybernetics,Part B: Cybernetics,2012,42(2): 513-529.

[3] Wang X Z,Shao Q Y,Miao Q,et al. Architecture selection for networks trained with extreme learning machine using localized generalization error model [J]. Neurocomputing,2013,102(2): 3-9.

[4] Bai Z,Huang G B,Wang D. Sparse extreme learning machine for regression [J]. IEEE Transations on Cybernetics,2014,44(10): 1858-1870.

[5] Zhai Hua-wei,Cui Li-cheng,Zhang Wei-shi. Novel online adaptive algorithm of extreme learning machine based on improved sensitivity analysis [J]. Journal of Chinese Computer Systems,2019,40(7): 1386-1390.

[6] Lan Y,Soh Y C,Huang G B. Ensemble of online sequential extreme learning machine [J]. Neurocomputing,2009,72(13-15): 3391-3395.

[7] Zhai Jun-hai,Zhang Su-fang,Zhou Zhao-yi. Ensemble retrained extreme learning machine by fuzzy integral for data classification [J]. Journal of Chinese Computer Systems,2018,39(6): 1223-1227.

[8] Horata P,Chiewchanwattana S,Sunat K. Robust extreme learning machine [J]. Neurocomputing,2013,102: 31-44.

[9] Man Z,Lee K,Wang D,et al. A new robust training algorithm for a class of single-hidden layer feedforward neuralnetworks [J]. Neurocomputing,2011,74(16): 2491-2501.

[10] Huang G,Song S,Gupta J N D,et al. Semi-supervised and unsupervised extreme learning machines [J]. IEEE Transactions on Cybernetics,2014,44(12): 2405-2417.

[11] Heeswijk M V,Miche Y,Oja E,et al. GPU-accelerated and parallelized ELM ensembles for large-scale regression [J]. Neurocomputing,2011,74(16): 2430-2437.

[12] Kasun L L C,Zhou H M,Huang G B,et al. Representational learning with ELMs for big data [J]. IEEE Intelligent Systems,2013,28(5): 31-34.

[13] An L,Bhanu B. Image super-resolution by extreme learning machine [C] //IEEE International Conference on Image Processing, Orlando,FL,USA,2012: 2209-2212.

[14] Zhang Bo-yang,Li Su-mei. Stereoscopic image quality assessment method using deep extreme learning machine [J]. Journal of Chinese Computer Systems,2017,38(11): 2586-2590.

[15] Xu Yan,Wang Quan-wei,Wei Zhen-yu. Traffic sign recognition algorithm combining weighted ELM and AdaBoost [J]. Journal of Chinese Computer Systems,2017,38(9): 2028-2032.

[16] Huang Qiang,Wang Yong-xiong. 3D object recognition method combining extreme learning machine and coalesce convolutional network [J]. Journal of Chinese Computer Systems,2019,40(9): 1909-1914.

[17] He Q,Shang T,Zhuang F,et al. Parallel extreme learning machine for regression based on MapReduce [J]. Neurocomputing,2013,102(2): 52-58.

[18] Sun Y J,Li B Y,Yuan Y,et al. Big graph classification frameworks based on extreme learning machine [J]. Neurocomputing,2019,330: 317-327.

[19] Yao L,Ge Z Q. Distributed parallel deep learning of hierarchical extreme learning machine for multimode quality prediction with big process data [J]. Engineering Applications of Artificial Intelligence,2019,81: 450-465.

[20] Ming Y,Zhu E,Wang M,et al. DP-ELMs: data and model parallel extreme learning machins for large-scale learning tasks [J]. Neurocomputing,2018,320: 85-97.

[21] Chen C,Li K L,Ouyang A J,et al. GPU-accelerated parallel hierarchical extreme learning machine on flink for big data [J]. IEEE Transactions on Systems,Man,and Cybernetis: Systems,2017,47(10): 2740-2753.

[22] Duan M,Li K,Liao X,et al. A parallel multiclassification algorithm for big data using an extreme learning machine [J]. IEEE Transactions on Neural Networks and Learning Systems,2018,29(6): 2337-2351.

[23] Wu Xin-dong,Ji Sheng-wei. Comparative study on mapreduce and Spark for big data analytics [J]. Journal of Software,2008,29(6): 1770-1791.

附中文参考文献:

[5] 翟华伟,崔立成,张维石. 一种改进灵敏度分析的在线自适应极限学习机算法 [J]. 小型微型计算机系统,2019,40(7): 1386-1390.

[7] 翟俊海,张素芳,周昭一. 用模糊积分集成重复训练极限学习机的数据分类方法 [J]. 小型微型计算机系统,2018,39(6): 1223-1227.

[14] 张博洋,李素梅. 应用深度极限学习机的立体图像质量评价方法 [J]. 小型微型计算机统,2017,38(11): 2586-2590.

[15] 徐 岩,王权威,韦镇余. 一种融合加权 ELM 和 AdaBoost 的交通标志识别算法 [J]. 小型微型计算机系统,2017,38(9): 2028-2032.

[16] 黄 强,王永雄. 结合超限学习机和融合卷积网络 3D 物体识别方法 [J]. 小型微型计算机系统,2019,40(9): 1909-1914.

[23] 吴信东,嵇圣硙. MapReduce 与 Spark 用于大数据分析之比较 [J]. 软件学报,2018,29(6): 1770-1791.

(C)1994-2022 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net