文章编号:1674-2869(2019)03-0283-07

基于Spark并行SVM参数寻优算法的研究

何经纬[1,2],刘黎志[*1,2],彭 贝[1,2],付星堡[1,2]
1. 智能机器人湖北省重点实验室(武汉工程大学),湖北 武汉 430205;
2. 武汉工程大学计算机科学与工程学院,湖北 武汉 430205

摘 要:针对传统支持向量机(SVM)参数寻优算法在处理大样本数据集时存在的寻优时间过长,内存消耗过大等问题,提出了一种基于Spark通用计算引擎的并行可调SVM参数寻优算法。该算法首先使用Spark集群将训练集以广播变量的形式广播给各个Executor,然后并行化SVM的参数寻优过程,并在在寻优过程中控制Task并行度,使各个Executor负载均衡,从而加快寻优速度。实验结果表明,本文提出的参数寻优算法,通过设置合理的Task并行度,可以在充分使用集群资源的同时提高最优参数的寻找速度,减少寻优时间。

关键词:支持向量机;参数寻优;Spark;并行度;负载均衡
中图分类号:TP311 **文献标识码**:A **doi**:10.3969/j.issn.1674-2869.2019.03.015

Spark Parallel SVM Parameter Optimization Algorithm

HE Jingwei[1,2], *LIU Lizhi*[*1,2], *PENG Bei*[1,2], *FU Xingbao*[1,2]
1. Hubei Key Laboratory of Intelligent Robot (Wuhan Institute of Technology), Wuhan 430205, China;
2. School of Computer Science & Engineering, Wuhan Institute of Technology, Wuhan 430205, China

Abstract: To solve the problems of the traditional support vector machine parameter optimization algorithm in dealing with large sample data sets, such as long time-consuming and excessive memory consumption, we proposed a parallel adjustable Support Vector Machine (SVM) parameter optimization algorithm based on Spark universal computing engine. Firstly, this algorithm uses Spark cluster to distribute the training set to each executor in the form of broadcast variables, and then makes the parameter optimization process of SVM parallel. In the parameter optimization process, each executor is load-balanced by controlling the parallelisms of the tasks, thereby speeding up the parameter optimization. At last the experimental results show that the proposed algorithm in this paper can improve the search speed and reduce the optimization time by setting the reasonable tasks parallelisms with making full use of the cluster resources.

Keywords: support vector machine; parameter optimization; spark; parallelism; load balancing

随着互联网的发展,越来越来的智能设备被接入到网络中来,数以万计的设备每天都在产生大量的数据,如何从海量的数据中获取有价值的信息成为当前研究的热点。支持向量机[1-5](support vector machine,SVM)算法在参数设置合理的情况下,处理小样本、高维度数据集时表现出很好的性能和准确率,而不合理的参数设置将会导致糟糕的性能和极低的准确率,所以参数的选取是SVM算法中至关重要的一环。传统的SVM参数寻优算法在处理大规模数据集时往往会遇到计算机性能的瓶颈,计算机的处理器资源、内存资源全部被占用,在耗费相当长的时间后才能得到处理

收稿日期:2019-02-24
基金项目:武汉工程大学第十届研究生教育创新基金(CX2018215)
作者简介:何经纬,硕士研究生。 E-mail:1415997594@qq.com
***通讯作者**:刘黎志,硕士,副教授。E-mail:llz73@163.com
引文格式:何经纬,刘黎志,彭贝,等.基于Spark并行SVM参数寻优算法的研究[J].武汉工程大学学报,2019,41(3):282-289.

结果。

集群环境下的并行计算方式为大数据的处理提供了新的思路，目前主流的大数据处理技术基本都用到了集群环境[6-13]。集群环境并行计算是提高大规模数据集SVM参数寻优速度的一种有效途径，多计算机并行的SVM参数寻优算法可以有效解决计算机单机计算能力不足、宕机等问题。目前主流的集群计算平台有Hadoop和Spark，基于内存计算的Spark目前应用非常广泛，如雅虎、Uber等公司都在使用Spark平台处理自己的业务，所以使用Spark实现并行化的SVM参数寻优算法是可行的方案。

刘泽燊等[14]使用Spark实现了并行的SVM算法，李坤等[15]使用Spark集群建立了SVM参数并行寻优模型，但是他们都忽略了集群Task分配、负载均衡等方面对参数寻优效率的影响。为了更加合理地利用集群资源，同时使集群中的Executor达到负载均衡，本文对SVM算法最优参数网格搜索的过程以及Spark并行计算引擎的特点进行了分析，调整优化网格搜索算法的结构，使用Spark平台实现具体的并行算法，并通过调节Task的并行度对Spark的Task分配进行优化，使集群中各个Executor达到负载均衡，从而大幅度地减少寻优时间。

1 概 述

1.1 SVM算法

SVM算法是一种基于结构风险最小化，建立在统计学理论上的有监督机器学习算法，具有很好的泛化能力，在分类与回归分析中有着广泛的应用，如人脸识别、文本分类、手写字体识别等方面。

SVM算法的目的是求解最优超平面，本质上是一个凸二次规划问题，假设训练样本集为 $D=\{(x_i,y_i)|x_i\in R^n,y_i\in\{-1,1\}\}_{i=1}^m$，设超平面系数为 $\boldsymbol{w}=(w_0,w_1,\cdots,w_n)$，截距为 b，求解最优超平面原问题描述如下：

$$\min_{\boldsymbol{w},b}\frac{1}{2}\|\boldsymbol{w}\|^2$$
$$\text{subject to } y_i(\boldsymbol{w}\cdot x_i+b)-1\geqslant 0 \quad i=1,2,\cdots,m \quad (1)$$

式(1)表示在满足条件 $y_i(\boldsymbol{w}\cdot x_i+b)-1\geqslant 0$ 的约束下，超平面系数向量 $\boldsymbol{w}$ 的模最小，从而使得超平面距离支持向量的物理间距最大。原问题不容易求解，可以通过原问题的对偶问题求解，引入拉格朗日算子并且对参数求偏导，进而求出与原问题对应的对偶问题，具体对偶问题如下所示：

$$\max_a\sum_{i=1}^m\alpha_i-\frac{1}{2}\sum_{i=1}^m\sum_{j=1}^m\alpha_i\alpha_j y_i y_j K(x_i,x_j)$$
$$\text{subject to } 0\leqslant a_i\leqslant c \qquad i=1,2,\cdots,m$$
$$\sum_{i=1}^m\alpha_i y_i=0 \quad (2)$$

式(2)中 m 为支持向量的个数，α_i 为支持向量对应的拉格朗日算子，c 为惩罚参数，表示对分类错误样本点的惩罚代价。

由式(2)可以看出，非边界样本点对应的参数 α_i 都是0，因此只有支持向量样本点对问题的求解有用，惩罚参数 c 可以剔除样本集中的一些噪声点。

对于样本集线性可分的情况，最优超平面可以很容易求解出来；若样本集线性不可分，此时需要引进核函数，将低维空间线性不可分问题映射成高维空间线性可分的问题。SVM核函数主要有四种，分别为线性核函数(linear Kernel)、多项式核函数(polynomial kernel)、径向基核函数(RBF kernel)、Sigmoid核函数(sigmoid kernel)。径向基核函数也称高斯核函数，是比较常用的一种核函数，公式为：$H(x,x')=\exp(1-g\|x-x'\|^2)$。其中本文参数寻优涉及的2个参数 c、g，c 代表式(2)中的惩罚参数，g 代表径向基核函数中的参数 g。

1.2 Spark

Apache spark是一种基于内存计算的通用计算引擎，常用来处理大规模数据集。与Hadoop相同的是，Spark可以执行Map、Reduce等操作，但Spark还包含了很多Hadoop不具备的算子，在数据处理方面要比Hadoop灵活很多。Spark的各种操作主要集中在内存，但Hadoop在数据处理过程中需要频繁读写HDFS，造成大量的磁盘I/O和通信开销，所以在计算速度上，Spark要比Hadoop快很多。同时Spark与Hadoop完全兼容，Spark可以使用Hadoop集群上的HDFS做为分布式文件存储系统。

Spark的核心部分是弹性分布式数据集(resilient distributed datasets, RDD)，RDD是一个基于内存具有容错性的分区只读记录集合，通过RDD分区(partition)来决定集群中Worker的任务分配。RDD包含转换(transformation)和动作(action)两种算子，转换，如map()、flatmap()、filter()等，它是将一种格式的RDD转换为另外一种格式的RDD；而动作，如collect()、count()、take()等，它的功能则是得到具体的结果。其中转换操作不会被立即执行，只有遇到动作时，动作之前的转换操作和动作才会被执行。

Spark的运行模式有Local、Standalone和Yarn等模式，本文中采用的是Standalone模式，在Standalone模式下，Driver程序可以在Master节点运行也可以在本地的Client端运行，本文使用Eclipse向集群提交Application，所以Driver程序运行在Client端。

1.3　支持向量机软件包

支持向量机软件包（library for support vector machines，LIBSVM）是台湾大学林智仁教授等开发的一个用于SVM快速建模程序包，它提供了大量的API给开发者进行调用，各个方法的参数设置非常灵活，目前很多SVM算法相关的研究都是基于LIBSVM的二次开发。

在SVM分类模型建立过程中，惩罚参数c和核函数参数g的选取直接影响模型分类的准确率。由于不能确定使模型分类准确率最高的参数，为了获得最优的(c, g)参数，通常使用LIBSVM自带的网格搜索（grid search）算法进行参数寻优，网格搜索即通过穷举将所有的参数组合进行交叉验证（cross-validation），找出分类准确率最高的参数组合，是一个非常耗时的过程。

2　参数寻优算法并行与优化

2.1　算法并行化的思路

网格搜索过程中，因为每组(c, g)参数组合的交叉验证过程相互独立，所以可以通过Spark并行计算引擎将搜索过程并行化。利用RDD的MapReduce原理，将所有的参数组合存入RDD中，RDD触发动作后被分解为很多逻辑相同的Task，这些Task会被分配到相同或者不同的Executor上并行执行。算法将交叉验证的过程放在RDD的Map阶段，使交叉验证在各个Task上并行执行，等待所有Executor中的Task完成交叉验证后，利用Reduce动作汇总所有结果并计算出最高准确率和参数组合。算法中使用LIBSVM包提供的交叉验证方法对参数进行交叉验证，由于原生LIBSVM交叉验证算法的输入输出不能够满足实验需求，所以实际算法对svm_train.java的训练集读取方式以及交叉验证结果的输出形式进行了改写，使其能适应并行网格搜索的输入和输出。交叉验证的基本流程为：

1）将原始训练集均匀划分成k份的数据集；

2）选取其中1份数据集（未被作为测试集的数据集）作为测试集，其他的$k-1$份作为训练集；

3）用训练集训练出模型，再用测试集去测试模型的准确率；

4）重复上述第二步和第三步，直到原始训练集中所有的数据集都被作为测试集进行测试为止；

5）求出所有测试所得准确率的均值作为最终准确率。

上述步骤即为k折交叉验证（k-fold cross-validation），本文所提到的交叉验证都为k折交叉验证，k折交叉验证的过程中对训练集中所有的数据都进行了测试，可以有效地避免过拟合和欠拟合问题。

2.2　广播变量的使用

并行网格搜索前，将Driver端读取的训练集以广播变量的形式广播给各个Executor，每个Executor保存一份训练集副本；如果Driver端读取的训练集以List形式保存共享，Executor的每个Task都会保存一份训练集副本。

假设在1个Application中分配m个Executor，每个Executor中有n个Task在执行，当训练集使用广播变量的形式进行广播时，整个Application中总共保存m份训练集副本；但当训练集使用List形式在Driver端保存共享时，整个Application中总共保存$m\cdot n$份训练集副本，所以采用List形式保存共享训练集会比广播变量形式多产生$m\cdot n-m=(n-1)\cdot m$份训练集副本。当训练集较大、Task的数量较多时，重复保存的$(n-1)\cdot m$份训练集副本会占用大量的内存，甚至会导致内存溢出。

2.3　Task并行度与Executor负载均衡

在Spark集群中，根据Action的不同Application被划分为不同的Job，Job中的每处宽依赖被划分一个Stage，每个Stage中包含多个Task，Task是运行在Executor处理器内核中，执行Job的最小逻辑单元。并行网格搜索计算量最大，最耗时的交叉验证阶段是由Executor中的Task来完成的，为了让Application中分配的所有Executor能够发挥最大效能，本文通过在Map阶段控制Task的并行度，让每个Executor分配到的Task数目一致或者接近一致，尽可能的使Executor之间达到负载均衡，从而加快搜索的速度。Executor的Task分配情况如图1的Map阶段所示，图1描述的为理想情况，所有Executor分配的Task数目一致，此时的寻优速度较快。

为将Task并行度变为自主可控参数，本文把Spark集群配置文件中的spark.default.parallelism参数提取出来并重写覆盖，将其定义为一个变量（Parallelism），算法中的通过控制Parallelism来控

制并行Task的数量。并行可调的网格搜索算法主要流程如图1所示,实现步骤如下:

1)输入Application的Task并行度,c、g参数数目,交叉验证折数。

2)根据c、g的数量和步长自动生成参数组合,并将其存入RDD中。

3)读取训练样本,并将其转换为广播变量。

4)根据输入的Task并行度以及存储参数组合的RDD为每个Executor分配Task。

5)对存有c、g参数组合的RDD执行mapToPair()转换,并在转换过程中对广播变量中的训练样本进行交叉验证,将参数组合和准确率以键值对的形式返回。

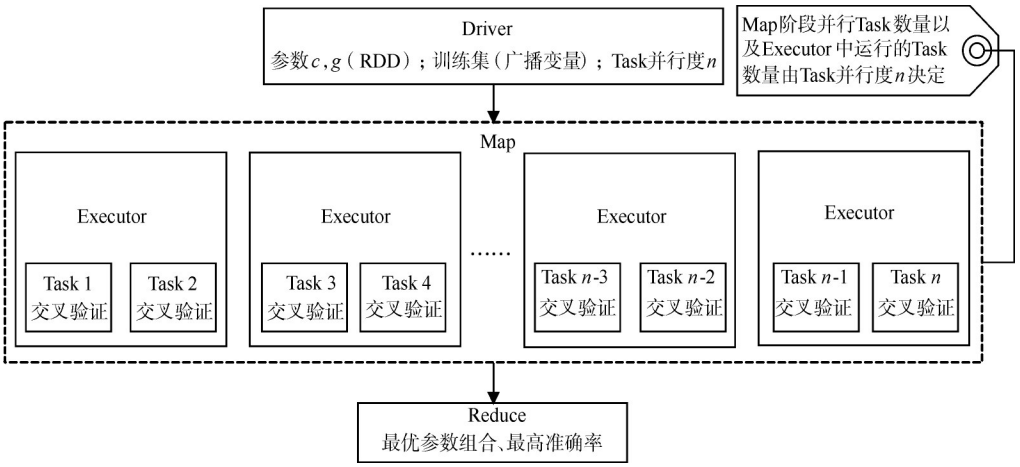

图1 并行网格搜索过程

Fig. 1 Searching process of parallel grid

6)通过Reduce()动作计算出最优参数组合以及准确率,Driver计算出寻优总时间。

并行可调网格搜索算法的核心算法:

Input: TrainDatasetPath, Parallelism, CNum, GNum, K-Fold

Output: C, G, Accuracy, TotalTime

1. Application初始化,根据Parallelism设置Task并行度;

2. JavaSparkContext jsc = new JavaSparkContext(spark.sparkContext());

3. List<String> cgList = new ArrayList<String>();

4. for (int i = 0; i < CNum; i++) { for (int j = GNum; j > 0; j--) { //生成c,g参数组合

5. String sparam = String.valueOf(初始值 + i * 步长) + "-" + String.valueOf(初始值 +

j * 步长);cgList.add(sparam);}}

6. JavaRDD<String> lines = jsc.parallelize(cgList);//RDD形式的c、g参数

7. 调用ReadTrainFromHDFS算法

8. 调用mapToPair算法

9. 调用reduce算法

10. Driver计算出寻优总时间TotalTime;

并行网格搜索前需要将HDFS中的训练集读取出来,并转换为广播变量,ReadTrainFromHDFS算法如下:

Input: fs - FileSystem对象, pt-训练集HDFS路径, jsc - JavaSparkContext对象

Output: broadcastssvRecords

1. Vector<String> svRecords = new Vector<String>();

2. if (fs != null) {BufferedReader br = new BufferedReader(new InputStreamReader(fs.open(pt)));

3. try {String line; while ((line = br.readLine()) != null && line.length() > 1) {

4. svRecords.addElement(line);}} finally {br.close();}}//将训练集读取为Vector<String>格式

5. Broadcast<List<String>> broadcastssvRecords = jsc //将训练集转换为广播变量

.broadcast(Arrays.asList(svRecords.toArray(new String[svRecords.size()])));

6. return broadcastssvRecords;

Map阶段的并行Task的数量即并行交叉验证的数量由Parallelism决定,mapToPair算法如下:

Input: s - String类型格式为"c-g"的参数组合, K-Fold

Output: Tuple2(s, acc)

1. String[] svr = (String[]) broadcastssvRecords.value().toArray();// 使用广播变量

2. Double c=Double.valueOf(s.split("-")[0]);

3. Double g=Double.valueOf(s.split("-")[1]);

4. MSSvmTrainer svmTrainer = new MSSvmTrainer(svr, c,g, K-Fold);

5. String acc = svmTrainer.do_cross_validation(); //交叉验证

6. return new Tuple2(s, acc);//返回<参数组合，准确率>的键值对

Reduce阶段主要处理Map阶段产生的键值对，通过比较准确率大小，得出最优参数组合，reduce算法如下：

Input: x, y

Output: 返回x, y键值对中准确率高的键值对

1. if (Double.parseDouble(x._2().replace("% ", "")) > Double.parseDouble(y._2().replace("%", ""))) {

2. return x;} else {return y;}

3　实验部分

3.1　实验环境与数据

Spark集群的主要硬件环境为一台戴尔R720服务器，服务器配置为两颗E5-2620V2 6核12线程处理器，主频2.1 GHz，最大睿频2.6 GHz，32 GB内存，8 TB硬盘，服务器被虚拟化为4个节点，一个Master节点4个Worker节点（Master节点也是Worker节点），每个节点有3个内核，8 GB内存，2 TB硬盘；集群使用的主要软件有Spark2.1.1、Hadoop2.7.3、JDK1.8等，操作系统为Ubuntu-16.04.1-Server-amd64。

实验采用的是LIBSVM官网提供的a8a二分类数据集，该数据集大小为1.6 MB，包含22 696个样本，每个样本有123维特征。

3.2　实验结果及性能分析

实验选择64组(c, g)参数组合作为测试对象，交叉验证折数k为4，参数c、g各8组。参数c的初始值为0.5，递增步长为0.25，搜索范围为0.5~2.25；参数g的初始值为0.05，递增步长为0.012 5，搜索范围为0.05~0.137 5。实验过程中为Application分配4个Executor，每个Executor 3个内核，2 GB内存。

在不设置并行Task数量与通常采用的最大并行Task数量的情况进行实验，实验结果如表1所示。

表1　两种Task并行度实验结果

Tab. 1　Experiment results of two parllelisms of tasks

并行Task数量/个	寻优总时间/s	最优参数 c,g	准确率/%
2	6 731	1.5,0.1	84.6
64	2 544	1.75,0.062 5	84.7

从表1可以看出，不设置并行度的情况下只有2个Task并行执行，设置最大并行度后，64个Task并行执行，寻优总时间极大地减少。从Spark Web UI上的Executors上查询出，不设置并行度时集群中只启动了2个Executor来执行Task，显然Application没有使用本次分配的全部集群资源，有一部分资源处于闲置状态；而设置最大并行度后，集群使用了分配的全部资源，启动了4个Executor来执行Task。

为了比较集群并行Task的数量对寻优效率和速度的影响，在实验中设置核心算法中的Parallelism参数分别为4、8、12、16、20、24进行参数寻优的测试，实验数据如表2所示。

表2　不同Task并行度实验结果

Tab. 2　Experimental results of different parallelisms of tasks

并行Task数量/个	寻优总时间/s	最优参数 c,g	准确率/%
4	4 809	1.25,0.125	84.6
8	2 618	1.25,0.125	84.7
12	1 974	1.75,0.75	84.7
16	2 446	2.25,0.087 5	84.7
20	2 084	2.0,0.087 5	84.6
24	1 961	2.25,0.025	84.6

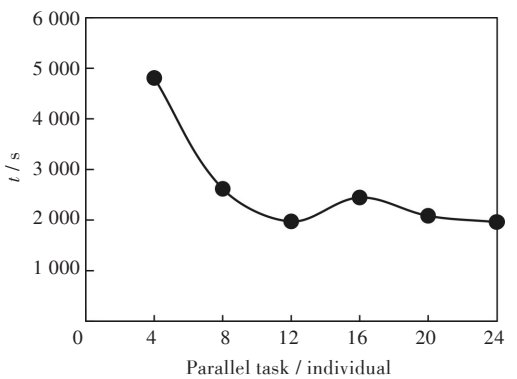

图2　不同并行度寻优总时间趋势图

Fig. 2　Trend diagram of total optimization time for different parallelisms

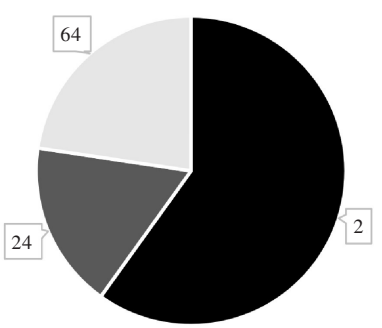

图3　三种并行度寻优总时间对比图

Fig. 3　Comparison diagram of total optimization time for three parallelisms

从表2和图2可以看出，Task的并行度并非设置越大越好，当并行的Task数量小于12时，训练总时间随着并行的Task数量的增加而降低；但当并行的Task数量超过12时，总训练时间开始上升，在并行的Task数量为24时，总训练时间接近Task并行数量为12时。在设置合理的并行Task数量后，参数寻优的准确率基本不变（上下波动不超过0.1%），时间性能提升了（4 890−1 961）/1 961≈149%。从图3可以看出，在并行的Task数量为24的时候，寻优的时间性能相对在不设置并行度、并行度最大的情况下都有一定的提升，相对在不设并行度的情况下提升了（6 731−1 961）/1 961≈243%，相对在最大并行度的情况下提升了（2 544−1 961）/1 961≈30%。

为了进一步测试并行Task数量为12整数倍对寻优总时间的影响，设置Parallelism参数为36再次进行测试，结果如表3所示，并行Task数量为12或12的整数倍的时候，总寻优时间比较接近。

表3 并行度为12整数倍的实验结果

Tab. 3 Experimental results of parallelisms in integer multiples of 12

并行Task数量/个	寻优总时间/s
12	1 974
24	1 961
36	2 013

从表1和表3以及图2相关数据可以看出，Task并行度的设置对寻优总时间有很大的影响，进一步分析Task并行度对Executor负载均衡的影响，在程序中设置标签来统计每个Executor完成的Task数量，将相关数据在Logs输出，统计数据如表4所示。

表4 不同并行度各Executor分配Task数量

Tab. 4 Numbers of tasks assigned to each executor under different parallelisms 个

并行Task数量	Executor 0 Task数量	Executor 1 Task数量	Executor 2 Task数量	Executor 3 Task数量
4	48	0	0	16
8	24	0	16	24
12	16	16	16	16
16	12	20	16	16
20	19	14	16	15
24	16	16	16	16

通过表4和图2以及图4可以看出，当Task数量是12或者12的整数倍的时候，各个Executor分配的Task数量相同，达到负载均衡，此时的寻优总时间也是最短的；当Task数量不是12或者12的整数倍的时候，各个Executor分配的Task数量不一致，分配Task数量较多的Executor的交叉验证的总时间会相对较长，分配Task数量较少的Executor在完成交叉验证Task后会等待分配Task较多的Executor，直到所有Executor完成交叉验证Task，网格搜索结束，所以网格搜索的总时间是由交叉验证总用时最长的那个Executor决定的。默认情况下，Executor的一个内核在同一时间只处理一个Task，所以设置并行Task的数量为Application的Executor内核总数或总数的整数倍可以使各个Executor分配到的Task数目相等，达到负载均衡，从而使并行网格搜索的速度达到最快。

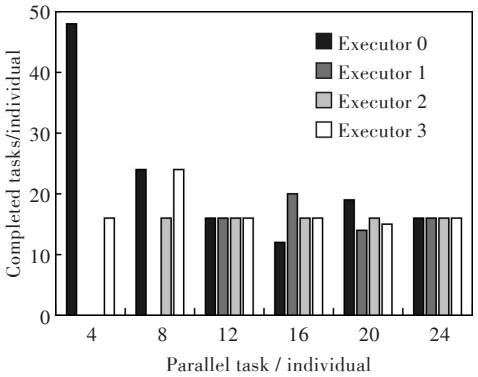

图4 不同并行度各Executor分配Task数量对比图

Fig. 4 Comparison diagram of tasks assigned to each executor with different parallelisms

4 结 语

SVM大数据集参数寻优的计算量相当大，用传统的单机参数寻优算法来处理大数据集显然不现实。本文提出了一种基于Spark通用计算引擎的并行可调SVM参数寻优算法，通过分析算法在Task不同并行度下的寻优时间，发现并非Task并行度设置的越大寻优速度越快，需要根据Application分配的集群资源，调整Task的并行度（设Application的Executor内核数量为m，Executor数量为n，则Task最优并行度为$m\cdot n$或$m\cdot n$的整数倍），使各个Executor达到负载均衡，从而显著提高寻优速度。从集群的角度来看，在Application中每个Task耗时相差不大的情况下，Task分配的越均匀，Application的总耗时越少，当Task完全均匀分配时，即负载均衡的时候，Application总耗时最少。

参数寻优过程中集群内存资源的消耗优化是今后研究的重点之一，通过动态评估内存消耗，给Executor设置合理的内存，在不降低寻优速度的前提下，消耗尽可能少的内存资源完成SVM参数寻

优算法。

参考文献

[1] 吴云蔚,宁芊. 基于Hadoop平台的分布式SVM参数寻优[J]. 计算机工程与科学,2017,39(6):1042-1047.

[2] 张鹏翔,刘利民,马志强. 基于 MapReduce 的层叠分组并行 SVM 算法研究[J]. 计算机应用与软件,2015,32(3):172-176.

[3] 王越. Hadoop平台参数寻优的分布式SVM算法研究[D]. 西安:西安理工大学,2016.

[4] 张小琴,胡景,肖炜. 基于Hadoop云平台的分布式支持向量机[J]. 山西师范大学学报(自然科学版),2015,29(4):19-23.

[5] 秦军,戴新华,童毅,等. 基于MapReduce的SVM分类算法研究[J]. 计算机技术与发展,2015(6):87-91.

[6] 米允龙,米春桥,刘文奇. 海量数据挖掘过程相关技术研究进展[J]. 计算机科学与探索,2015,9(6):641-659.

[7] 宋泊东,张立臣,江其洲. 基于Spark的分布式大数据分析算法研究[J]. 计算机应用与软件,2019,36(1):39-44.

[8] 张红,王晓明,曹洁,等. Hadoop云平台MapReduce模型优化研究[J]. 计算机工程与应用,2016,52(22):22-25.

[9] ALHAM N K,LI M,YANG L,et al. A MapReduce-based distributed SVM algorithm for automatic image annotation [J]. Computers & Mathematics with Applications,2011,62(7):2801-2811.

[10] KE X,JIN H,XIE X,et al. A distributed SVM method based on the iterative MapReduce [C]// IEEE International Conference on Semantic Computing. Piscataway:IEEE,2015:116-119.

[11] GUO W,ALHAM N K,LIU Y,et al. A resource aware mapreduce based parallel SVM for large scale image classifications[J]. Neural Processing Letters,2016,44(1):161-184.

[12] MEYER O,BISCHL B,WEIHS C. Support vector machines on large data sets:simple parallel approaches [M]. Berlin:Springer International Publishing,2014.

[13] YAN B,YANG Z,REN Y,et al. Microblog sentiment classification using parallel SVM in apache spark[C]// IEEE International Congress on Big Data (BigData Congress). Piscataway:IEEE,2017:282-288.

[14] 刘泽燊,潘志松. 基于Spark的并行SVM算法研究[J]. 计算机科学,2016,43(5):238-242.

[15] 李坤,刘鹏,吕雅洁,等. 基于Spark的LIBSVM参数优选并行化算法[J]. 南京大学学报(自然科学版),2016,52(2):343-352.

本文编辑:陈小平