文章编号:1674 - 2869(2019)05 - 0499 - 05

逻辑回归中的批量梯度下降算法并行化研究

李姚舜,刘黎志*
智能机器人湖北省重点实验室(武汉工程大学),湖北 武汉 430205

摘 要:逻辑回归中的批量梯度下降算法需要访问全部数据样本,在单节点环境下计算耗时较长。针对大批量数据集的训练问题,提出了一种基于MapReduce框架的并行化算法。首先利用HDFS文件系统存储训练数据集,MapReduce框架会对输入数据集进行分片处理,每一个分片交由一个Map节点进行处理;Map过程的输出结果会传给Combiner节点,进行各个分片内部的数据归并;所有分片的归并结果会通过Shuffle过程,进行各个分片间的数据合并,汇总成一个输出文件;输出文件会传给Reduce节点进行运算,最后将计算结果用于参数更新。实验结果表明,集群环境下的参数训练结果正确,随着数据集的扩大,并行化计算的优势逐渐显现。

关键词:逻辑回归;参数训练;MapReduce;并行化
中图分类号:TP311 **文献标识码**:A **doi**:10. 3969/j. issn. 1674-2869. 2019. 05. 017

Parallel Research on Batch Gradient Descent Algorithm in Logistic Regression

LI Yaoshun, *LIU Lizhi**
Hubei Key Laboratory of Intelligent Robot (Wuhan Institute of Technology), Wuhan 430205, China

Abstract: Because the batch gradient descent algorithm should access all data samples in logistic regression, it will lead to high computational overhead, which is difficult to be handled by a single computer. To resolve the efficiency problem caused by large-scale training datasets, we proposed a parallel algorithm based on the MapReduce distributed computing framework. Firstly, training datasets were stored by using the Hadoop Distributed File System. Then, MapReduce processed the datasets slice by slice. Each slice was handled by a Map node. After that, the outputs of each node were transmitted to a combiner node, which merged data belonging to a same slice. Next, the merging results of all slices were put into one output file through the Shuffle process. Finally, the out put filewas passed to the Reduce node for calculation and updating parameters. Experimental results show that reliable parameters can be also achieved in the cluster environment. Above all, the parallel computing method has the obvious advantage of high efficiency when the training datasets grow huge.
Keywords: logistic regression; parameter training; MapReduce; parallelization

逻辑回归算法是机器学习领域的经典算法,虽然它使用起来比较简单,但是在各个行业中使用效率很高。逻辑回归模型仅在线性回归的基础上,套用了一个逻辑函数,使其成为了一种二分类算法,主要用于寻找危险因素、预测和判别等二分类模型[1-3]。有很多研究人员通过建立多个分类器或者改进逻辑回归的损失函数等方法,让逻辑回归可以解决多分类问题[4-6]。随着大数据时代的到来,数据量已经不仅仅局限于PB的范围,算法的计算过程还要求能够快速处理大批量的数据集。

收稿日期:2019-06-24
基金项目:武汉工程大学第十三期大学生校长基金项目(2018074)
作者简介:李姚舜。E-mail:liyaoshuncn@163.com
***通讯作者**:刘黎志,硕士,副教授。E-mail:llz73@163.com
引文格式:李姚舜,刘黎志. 逻辑回归中的批量梯度下降算法并行化研究[J]. 武汉工程大学学报,2019,41(5):499-503.

逻辑回归算法在计算过程中，不可避免需要对全局训练数据进行遍历以更新参数，这种串行运算随着训练数据集的增加，所消耗的资源也会难以估计，有很多研究人员也从硬件和算法优化上提出了优化的方法[7-8]。目前Apache基金会开发的Hadoop分布式框架，占据了大数据处理的庞大市场，已经成为大数据开发的标准。Hadoop中的HDFS的数据管理能力、MapReduce处理任务时的高效率以及它的开源特性，使它在同类的分布式系统中大放异彩[9-11]。MapReduce是Hadoop中的一个分布式计算框架，它将一项繁杂的计算任务划分为几项相对较轻的计算任务，交由多个计算节点并行处理以加速任务进程[12-15]。本文就如何在MapReduce框架下改进逻辑回归的参数训练过程进行讨论，并与单节点环境下的训练过程进行比较，并通过实验进行验证。

1 研究背景

逻辑回归的原理是将样本的特征与样本发生的概率联系起来，计算结果是通过样本的特征来拟合计算出一个事件发生的概率，它实际上是一种分类模型，主要用于解决二分类问题。

逻辑回归的线性决策边界形式如下[16]：

$$\theta_0+\theta_1x_1+\theta_2x_2+,\dots,+\theta_nx_n=\sum_{i=1}^{n}\theta_ix_i=\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x} \quad (1)$$

其中 θ_n 表示第 n 个特征参数，x_n 表示一行样本的第 n 个特征值。

其回归预测利用了Sigmoid函数，形式如下：

$$g(z)=\frac{1}{1+\mathrm{e}^{-z}} \quad (2)$$

构造预测函数为：

$$h_\theta(\boldsymbol{x})=g\left(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}\right)=\frac{1}{1+\mathrm{e}^{-\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}}} \quad (3)$$

$h_\theta(\boldsymbol{x})$ 的值表示结果取1的概率。设训练数据集中的样本数量为 m，基于最大似然估计推导，得到的损失函数如下：

$$J(\boldsymbol{\theta})=-\frac{1}{m}\sum_{i=1}^{m}[y_i\log h_\theta(\boldsymbol{x}_i)+(1-y_i)\log(1-h_\theta(\boldsymbol{x}_i)] \quad (4)$$

与线性回归相比，逻辑回归的损失函数并不能推导出一个正规方程解，即 $J(\boldsymbol{\theta})$ 并没有数学的解析解，所以只能使用梯度法进行求解。此处 $J(\boldsymbol{\theta})$ 中乘了一个负的系数 $-\frac{1}{m}$，因此采用梯度下降算法求 $J(\boldsymbol{\theta})$ 取最小值时的 $\boldsymbol{\theta}$ 为所需要的最佳系数，$\boldsymbol{\theta}$ 参数更新过程为：

$$\theta_j:=\theta_j-\alpha\frac{\delta}{\delta_{\theta_j}}J(\boldsymbol{\theta}) \quad (5)$$

其中 α 为学习率。对 $J(\boldsymbol{\theta})$ 求偏导数的结果为：

$$\frac{\partial}{\partial\theta_j}J(\boldsymbol{\theta})=-\frac{1}{m}\sum_{i=1}^{m}\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)} \quad (6)$$

从参数更新过程中可以看出，式(6)(即计算梯度向量)是最基本的步骤，而式(6)中的 $\sum_{i=1}^{m}\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}$ 在计算过程中只需要进行向量间的点乘、相加，此处可以将求和过程拆分成相互独立的计算步骤，即对每行数据均单独计算 $\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}$，最后再归并计算求和结果，并代入式(6)中得到目标函数的梯度向量。

通过分析，可以考虑对式(6)中的求和计算进行改进以达到并行化的目的。若将训练数据集拆分为 {Split(1), Split(2), … , Split(s)} 等 s 个分片，每个分片中有 p 条数据(p 不一定相同)，将每个分片交由一个节点独立计算，最后将结果进行汇总，则式(6)求偏导数过程可转化为：

$$\frac{\delta}{\delta_{\theta_j}}J(\boldsymbol{\theta})=-\frac{1}{m}\sum_{t=1}^{s}\mathrm{Split}(t)\sum_{i=1}^{p}\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)} \quad (7)$$

其中 Split(t) 表示第 t 个分片中的数据。综上所述，$\boldsymbol{\theta}$ 的更新过程可以表示为：

$$\theta_j:=\theta_j-\alpha\frac{1}{m}\sum_{t=1}^{s}\mathrm{Split}(t)\sum_{i=1}^{p}\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)} \quad (8)$$

2 批量梯度下降算法并行化

2.1 批量梯度下降算法

梯度法是一种基于搜索的最优化方法，它是人工智能领域的一个非常重要的方法，但是它的作用是用于优化一个目标函数，如果要最小化一个损失函数，使用的就是梯度下降法，如果要最大化一个效用函数，使用的是梯度上升法。

式(5)中采用梯度下降算法求解损失函数最小值，有两种不同的形式：批量梯度下降(batch gradient descent，BGD)、随机梯度下降(stochastic gradient descent，SGD)。由于SGD在每次迭代过程只用到一个训练数据来更新参数，使得SGD并不是每次迭代都向着整体最优化方向，其在解空间的搜索过程看起来很盲目。与SGD相比，BGD每次学习都使用整个训练集，而逻辑回归的 $J(\boldsymbol{\theta})$ 是一个凸函数，没有局部最优解，只有唯一的全局最优解，因此选用BGD求解 $J(\boldsymbol{\theta})$ 最小值。BGD可以保证每次更新都会朝着正确的方向进行，最后能够使训练过程收敛于全局极值点，而且BGD对每行样本都进行了计算处理，因此也更能够改进

为并行化算法。

2.2　基于MapReduce的批量梯度下降

MapReduce是一种可用于数据处理的编程框架，采用“分而治之”的思想，把对大规模数据集的操作，分发给各个子节点共同完成，然后通过整合各个节点的中间结果，得到最终结果。MapReduce把处理过程高度抽象为map和reduce两个函数，map负责把任务分解成多个子任务，reduce负责把分解后多个子任务处理的结果汇总起来。

在逻辑回归计算过程中，可以将式(6)中的$\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}$放入Map过程中进行计算，将求和的过程放入Reduce中。MapReduce的工作流程如图1所示。

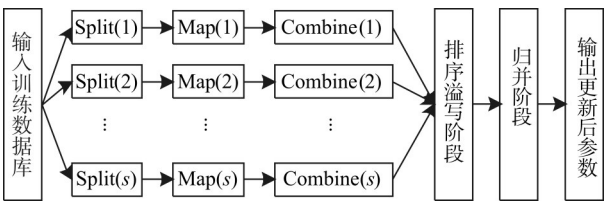

图1　MapReduce工作流程

Fig. 1　MapReduce workflow

在MapReduce框架下，采用JDK编程环境，矩阵运算在编程过程中均可转化为循环操作，实验过程中将矩阵向量均作为一维数组处理。

逻辑回归中的BGD算法并行化步骤如下：

2.2.1　Split分片　将数据集放入HDFS文件系统下，当Job开始时，MapReduce会读取数据文件并按照HDFS数据块大小进行分片{Split(1), Split(2), …, Split(*s*)}，每一个分片交由一个Map节点进行处理。

2.2.2　解析键值对　MapReduce处理数据的最小单位是键值对，当分片Split(*t*)中的数据输入map函数之前，会将其按照默认规则转化为“<文本起始位置，文本内容>”的键值对形式，分片中的每一行样本$x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}, y^{(i)}$便会转化为一个输入键值对$<\text{key},\{x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}, y^{(i)}\}>$。

其中$x_n^{(i)}$表示第*i*行的第*n*个特征值，$y^{(i)}$表示第*i*行的标签值。

2.2.3　Map过程　每一个分片由一个Map节点处理，每解析出分片中的一条记录$<\text{key},\{x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}, y^{(i)}\}>$，便会调用一个map函数。只需要将value值$x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}, y^{(i)}$（此时为文本形式）通过Java分割函数分割为字符串数组$\{x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}, y^{(i)}\}$。然后将其中的特征值存储到数组$\boldsymbol{x}[\,]=\{x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}, y^{(i)}\}$中，在存储过程中，须对每一个数组手动添加一个参数偏移量$x_0^{(i)}$（此处取常量1.0）；将标签值存储到标签变量$y=y^{(i)}$中，从而达到获取特征值与标签值的效果。

将客户端中存储的回归系数向量$\boldsymbol{\theta}[\,]=\{\theta_0, \theta_1, \theta_2, \ldots, \theta_n\}$与特征值向量$\boldsymbol{x}[\,]$作矩阵乘法得到式(1)中的线性决策边界$\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}$，代入式(3)得到该行样本的预测值$h_\theta\left(\boldsymbol{x}^{(i)}\right)$，将$h_\theta\left(\boldsymbol{x}^{(i)}\right)$、标签变量$y$、特征值向量$\boldsymbol{x}[\,]$代入式(7)（此时只有1行数据）求得第*i*行样本的偏导数向量$\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}$。

以“<参数序号$j,\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}$>”的形式输出到中间结果。

输出完成后map函数结束，当分片Split(*t*)中的所有样本均被map函数处理后，该分片的Map过程结束。当所有分片的Map过程均完成处理后，整个Job的Map过程结束，所有的输出结果会溢出到HDFS本地磁盘中保存为一个文件。

2.2.4　Combine过程　此过程是对Reduce过程的优化，如果将所有的键值对均传输给Reduce过程进行处理，势必要耗费大量的网络传输资源。因此Map在输出结果到磁盘之前，会先将数据在Combine过程中提前处理，将键值对中key相同的value值进行求和，得到当前分片中的$\sum_{i=1}^{p}\left[h_\theta\left(\boldsymbol{x}^{(p)}\right)-y^{(p)}\right]x_j^{(p)}$向量，将向量按照“<参数序号$j,\sum_{i=1}^{p}\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}$>”的键值对形式整理后再进行输出。

2.2.5　排序溢写过程　每一个分片处理完成后均会输出2.2.4中描述的键值对，所有分片的输出结果会传给排序溢写(Shuffle)过程进行排序、归并处理，再将结果传给Reduce阶段进行处理。整理结果会按照“$<j, <\text{Split}(1)\sum_{i=1}^{p}\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}$, Split(2) $\sum_{i=1}^{p}\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}, \ldots, \text{Split}(s)\sum_{i=1}^{p}\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}$ >>”的键值对形式进行输出。

2.2.6　归并过程　归并(Reduce)输出时为保证将更新后的所有参数均同时输出，在Job过程中设置Reduce的个数为1，即将所有的汇总结果均传给同

一个Reduce节点进行处理。将同一个key中value值进行求和运算，得到代入式(7)求和得到 $\sum_{t=1}^{s}\mathrm{Split}(t)\sum_{i=1}^{p}\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}$，代入式(8)更新参数，将结果以“<参数序号$j$, θ_j>”的键值对形式输出。输出完成后Reduce过程结束，一次参数更新过程完成，循环上述过程，直至参数收敛，便完成训练。

2.2.7 参数收敛过程判断

1)判断 $\boldsymbol{\theta}$ 向量更新后的变化距离的平方和，若值 d 小于指定阈值，则判断系数已经收敛：

2)训练数据集按设的次数结束迭代后，计算最后一次更新的 $\boldsymbol{\theta}$ 与上一次 $\boldsymbol{\theta}$ 的变化距离的平方和，即

$$d=\sum_{i=1}^{n}(\theta_i-\theta_i')^2$$

3)若 d 小于指定阈值，则判断系数已经收敛。

2.3 算法的具体实现

2.3.1 算法1 Map过程接收训练数据集，计算 $\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}$

Function map(regex, theta[], data)

输入：regex←训练数据分割字符串，theta[]←回归系数向量，data←训练数据集文件夹路径

输出：将键值对<参数序号j, $\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}$>写入中间输出文件

Begin

1: words[]←line.split(regex)

2: for i←0 to theta.length - 1

3: do x[i + 1]← words[i]

4: y ← words[words.length-1]

5: for i←0 to theta.length

6: do yHat← yHat + x[i] * theta[i]

7: pHat←1.0 / (1.0 + Math.exp(- yHat))

8: for i←0 to theta.length

9: do sum[i]←(y - pHat) * x[i]

End map

2.3.2 算法2 Combiner过程归并各分片输出结果，计算 $\sum_{i=1}^{p}\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}$

Function combiner(<key, values>)

输入：key← 参数序号 j, values← $\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}$

输出：键值对<参数序号j, $\sum_{i=1}^{p}\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}$>

Begin

1: for value←values [0] to values [values.length-1]

2: do sum←sum+ value

End combiner

2.3.3 算法3 Reduce过程合并所有分片输出结果，计算 $\sum_{t=1}^{s}\mathrm{Split}(t)\sum_{i=1}^{p}\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_n^{(i)}$，更新参数

Function reduce(<key, values>,alpha)

输入：key← 参数序号j，values← $\sum_{i=1}^{p}\left[h_\theta\left(\boldsymbol{x}^{(i)}\right)-y^{(i)}\right]x_j^{(i)}$，alpha←学习率

输出：键值对<参数序号j, θ_j>

Begin

1: for value←values [0] to values [values.length-1]

2: do sum←sum+value

3: theta[key]←theta[key]+value * alpha

End reduce

2.3.4 算法4 开始一个训练过程(Job)，调用Map、Combiner、Reduce过程，输出更新后参数

Function trainData(data, result, theta[], lastTheta[], alpha, regex, diff=100)

输入：result←输出文件夹路径，lastTheta[]←上一次训练的回归系数向量，diff←误差初始值

输出：更新后参数

Begin

1: while diff > E-4

2: do for i←0 to theta.length

3: do lastTheta[i]←theta[i]

4: 开始Map、Combiner、Reduce过程

5: diff←0

6: for i←0 to theta.length

7: do diff←diff + (th $eta[i]-lastTheta[i]$)^2

End trainData

3 实 验

实验用服务器为机械革命s1笔记本，其配置为1个物理CPU(Intel i7-8550u 1.8 GHZ，CPU含4核心8线程)，16 GB内存，1T硬盘，1个物理网卡。服务器安装win10专业版操作系统，使用VMware Workstation Pro15软件新建3个虚拟机，每个虚拟机的配置为1内核CPU，2 GB内存，20 GB硬盘，1

个物理网卡。每个虚拟机安装CentOS7操作系统，Hadoop 2.7.3分布式计算平台，组成含1个主节点，2个从节点的集群。使用eclipse4.11、Java SE 1.8作为开发环境。

实验内容分为两部分，第一部分验证并行MapReduce并行化结果的正确性，第二部分将分布式集群运行结果与单节点运算结果进行比较，说明并行化集群的优势所在。

实验中使用Java语言构建了一个针对大规模抵押贷款数据的逻辑回归分类模型，用于预测客户是否会如期归还贷款，数据集中共包含5个样本特征，数据集描述如表1所示。

表1　抵押贷款数据集描述

Tab.1　Description of mortgage loan data set

属性	标识符
抵押贷款人信用评级	θ_1
抵押房屋的使用年限	θ_2
抵押贷款人的工作年限	θ_3
抵押贷款人信用卡债务数额	θ_4
抵押贷款人申请贷款的年份	θ_5
判定值（0表示按期归还贷款，1表示逾期归还）	y

取2001至2005年的数据样本进行分析测试，每一年的数据文件中包含100万条数据样本，以9∶1的比例划分训练集和测试集，即在每一年的数据文件中取90万条训练数据集、10万条测试数据集，最终试验时共包含450万条训练数据和50万条测试数据。

3.1　验证并行化实验结果正确

取50万条训练数据分别在集群环境和单节点环境下进行测试，单机训练的过程此处不再进行赘述，实验所得两种环境下训练结果相同，可以验证并行化改进算法思路正确。参数训练结果如表2所示。

表2　小数据集参数训练结果

Tab.2　Training results of small data set's parameters

属性	值
θ_1	−0.008 2
θ_2	0.029 6
θ_3	−0.316
θ_4	0.0015
2001 year	1.248
2002 year	0.305
2003 year	−0.184
2004 year	−0.830
2005 year	0.227

3.2　集群与单机环境下训练结果比较

在上一步测试的环境下，每次添加80万条数据，在单机和集群中分别迭代运算1000次，统计两者的运行时间，以训练数据总量Data为横坐标，集群与单机的运行时间比 λ（倍）为纵坐标，画出对应的散点图如图2所示。

理想情况下，集群中有3个节点参与运算，时间比 λ 应该在3倍左右。实际实验中，在初期训练数据集较小的情况下，运行效率没有达到预测值，但是随着训练数据集的扩大，集群运行效率达到理想值，甚至超过理想值。原因在于MapReduce过程中分割文件、传输数据均包含大量的文件输入输出操作，在数据集较小时，消耗资源不能忽略不计。

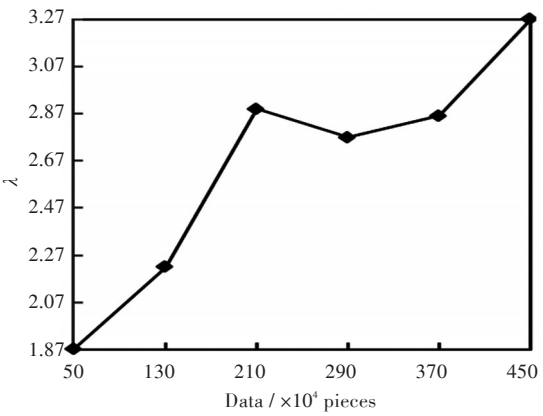

图2　集群与单机训练时间比

Fig. 2　Time ratio between cluster and single machine training

4　结　语

本文基于MapReduce分布式计算框架，提出了逻辑回归中BGD算法训练参数时的并行处理办法，实验结果表明，这一思路可行，集群环境下的运算效率显著提高。在实际应用过程中，随着节点数量的增加以及集群计算性能的提升，单位时间内能够处理的数据量也会越来越多。在实验过程中没有从根本上改进BGD算法，只是将求和过程进行分解，验证了并行化结果的正确以及高效性。在后续实验和研究过程中，将试图结合算法优化以及集群平台升级，从算法、软件和硬件平台三方面使逻辑回归等机器学习算法能适应更多生产环境。

（下转第510页）