基于 SparkR 的分类算法并行化研究

刘志强, 顾荣, 袁春风, 黄宜华

（南京大学计算机软件新技术国家重点实验室 南京 210046）

（江苏省软件新技术与产业化协同创新中心 南京 210046）

(MF1433027@smail.nju.edu.cn)

The Parallelization of Classification Algorithms Based on SparkR

Liu Zhiqiang, Gu Rong, Huang Yihua and Yuan Chunfeng

(State Key Laboratory for Novel Software Technology at Nanjing University, Nanjing 210046)

(Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing 210046)

Abstract In recent years, the parallelization of algorithms for big data machine learning and data mining has become an important research issue in the field of big data. A few years ago, research community and industry around the world more concern about the parallel algorithms designed on the Hadoop platform. However, due to the overhead of network and disk read and write, the Hadoop MapReduce parallel computing framework cannot efficiently support machine learning algorithms which usually involve many iterative computations. With the advent and development of Spark, a new generation of big data parallel processing platform launched by UC Berkeley AMPLab, recently the research community and industry pay more attentions on the design and implementation of parallel algorithms of machine learning and data mining on the Spark platform. To support data analysts who are familiar with the R language in the general application areas to conduct the data analysis on the Spark platform, Spark provides data analysts with a programming interface, SparkR, to leverage the Spark parallel programming interfaces and powerful computing ability in the R environment. In this paper, we proposed the design and implementation of several widely-used parallel classification algorithms including Multinomial NaiveBayes, SVM and Logistic Regression based on SparkR. We also presented how we optimize the SVM and LR algorithms to improve the training speed based on conventional parallel strategies. Experiment results show that the efficiency of the classification algorithms based on the SparkR outperforms Hadoop with 8 times of speedup without losing scalability.

Key words SparkR; R; classification; parallelization; local iteration; in-memory computation

摘 要 近几年来，大数据机器学习和数据挖掘的并行化算法研究成为大数据领域一个较为重要的研究热点。早几年国内外研究者和业界比较关注的是在 Hadoop 平台上的并行化算法设计。然而， Hadoop MapReduce 平台由于网络和磁盘读写开销大，难以高效地实现需要大量迭代计算的机器学习并行化算法。随着 UC Berkeley AMPLab 推出的新一代大数据平台 Spark 系统的出现和逐步发展成熟，近年来国内外开始关注在 Spark 平台上如何实现各种机器学习和数据挖掘并行化算法设计。为了方便一般应用领域的数据分析人员使用所熟悉的 R 语言在 Spark 平台上完成数据分析，Spark 提供了一个称为 SparkR 的编程接口，使得一般应用领域的数据分析人员可以在 R 语言的环境里方便地使用 Spark 的并行化编程接口和强大计算能力。本文基于 SparkR 设计并实现了多种常用的并行化的机器学习分类算法，包括多项式贝叶斯分类

收稿日期：2014-11-15

本研究受到江苏省科技支撑计划项目(项目号：BE2014131）资助

通信作者：黄宜华（yhuang@nju.edu.cn）

算法，SVM 算法和 Logistic Regression 算法。进一步地，对于 SVM 和 Logistic Regression 算法，本文在常规的并行化策略的基础之上为了进一步提升训练速度，设计采用了并行化局部优化的迭代计算模式。实验结果表明，本文所设计实现的基于 SparkR 的并行化分类算法与 Hadoop MapReduce 的方案相比，速度上提升了 8 倍左右。

关键词 SparkR；R；分类算法；并行化；局部迭代优化；内存计算

中图法分类号 TP338 TP182

0 引言

近年来，行业应用数据规模的爆炸性增长推动了大数据技术的迅猛发展，机器学习在大数据中的应用也越来越广泛。在大数据环境下，随着数据量的增长，为了提升这些机器学习算法的执行性能，需要在现有的分布式平台上对原本的串行算法进行并行化设计处理[1]。因而，近几年来大数据机器学习和数据挖掘并行化算法研究成为大数据领域一个较为重要的研究热点。

近几年来，在大数据涉及到的各技术层面和系统平台方面都取得了长足的发展，出现了很多典型的大数据处理技术、软件系统和平台。Hadoop[2]是最早出现和广为使用的大数据处理平台，因而早几年国内外研究者和业界比较关注的是在 Hadoop 平台上的并行化算法设计。而随着 UC Berkeley AMPLab 推出的新一代大数据平台 Spark[3]系统的出现和逐步发展成熟，近年来国内外开始关注在 Spark 平台上如何实现各种机器学习和数据挖掘的并行化算法设计。

目前，内存计算成为解决大数据处理性能瓶颈公认的技术手段和发展趋势，而 UC Berkeley AMPLab 推出的 Spark[3]系统是内存计算的一个代表。Spark 使用基于内存计算的并行化计算模型 RDD（Resilient Distributed Datasets），提供了一个强大的分布式内存并行计算引擎，支持快速的迭代计算。然而，Spark 是基于 Scala 语言实现的，而 Scala 语言同时具有面向对象和函数式语言的特点，可编程性不是很好，且缺乏统计分析和常用的作图等功能，不易于让从事统计数据分析的工作人员掌握和使用。R[5]是一种用于统计计算与作图的开源软件，有完整的数据处理和统计分析功能，同时也是一种编程语言。它被广泛应用于企业和学术界的数据分析领域，正在成为最通用的统计分析语言之一。而 SparkR[4]是 R 的一个包，从 R 上提供一个轻量级前端使用 Apache Spark，即提供了 Spark 中弹性分布式数据集（RDD）的 API，用户可以在集群上通过 R shell 交互性的运行 job。

随着大数据智能分析的流行，机器学习分类算法有着广泛的应用场景。考虑到 R 在统计分析领域的广泛适用性以及 Spark 计算性能的高效性，本文基于 SparkR 设计并实现了多种常用的并行化的分类算法。进一步地，本文在常规的并行化策略的基础之上为了进一步提升训练速度，对训练算法设计采用了并行化局部优化的迭代计算模式。

本文的主要工作有以下两个方面：

1）基于 SparkR 设计并实现并行化的 Multinomial NaiveBayes 多分类算法、SVM 以及 Logistic Regression 二分类算法，并评估其与基于 Hadoop MapReduce 实现的并行化算法的性能差别。

2)在常见的并行化 SVM 和 Logistic Regression 的算法的基础之上，进一步对迭代计算进行局部优化，在保障预测精度和召回率的基础下，进一步减少迭代计算的时间。

本文组织如下：第一部分引言介绍了相关背景和研究现状，并说明本文的主要工作内容；第二部分介绍目前并行化算法的相关工作；第三部分简要阐述了 Spark 与 SparkR 的相关知识；第四部分重点阐述实现的三个分类算法的模型以及并行化设计方案，同时介绍了本文针对二分类 SVM 和 Logistic Regression 迭代计算局部优化的设计与实现。第五部分对本文提出的基于 SparkR 的并行化算法的性能进行了评估分析。最后，在第六部分我们对本篇论文的进行了总结，并指出下一步需要展开的工作。

1 相关工作

当前，在并行化机器学习算法的研究方面，常见的并行手段上主要有以下三种：

1）基于多核、众核设备的并行化机器学习算法。文献[13]实现了多核上的一些机器学习的算法，包括 KNN、决策树、贝叶斯网络等；文献[14]采用 Inter Xeon Phi 众核系统实现了对大规模神经网络

训练算法的并行化；文献[18]采用 MPI 和 OpenMP 混合实现了多核上的 Support Vector Machine 算法；文献[19]实现了在多核、众核上的协同过滤算法。多核、众核设备将数据存在 Global Memory 中，且有几十个核的并发处理能力，具有较快的处理速度。但是它有着单节点受限、耦合度过于紧密和可扩展性低的缺点。

2）基于 Hadoop MapReduce[2]并行化机器学习算法，如 Mahout[6]机器学习库包含了很多基于 Hadoop 实现的数据挖掘算法。文献[15]实现了基于 Hadoop 的海量文本挖掘的贝叶斯算法实现，[16]基于 Hadoop 分别实现了基于用户和商品的协同过滤推荐算法。采用 Hadoop MapReduce 并行化方案的优点是节点失效不会影响程序运行，并且还具有较好的可扩展性。但缺点是 Hadoop 在处理迭代式计算时每轮的作业启动开销过大。此外，磁盘 I/O 的开销也使得其执行效率不高，导致在针对机器学习中常用到的迭代计算时其效率低下。

3）基于 Spark 的并行化机器学习算法，如 MLlib[7]。MLLib 是 Spark 提供的机器学习库，包含了分类算法、聚类算法、推荐算法等。该并行化方案基于 Spark 内存计算，具有较好的可扩展性，并且没有了磁盘 I/O 的开销，使得其针对迭代计算也有良好的运行速度。然而，Spark 的编程语言 Scala 结合了面向对象和函数式语言的特点，其使用可编程性较差，且使用范围小众化，不如 R 语言使用范围广泛，并且没有 R 语言其他的众多数据处理分析的功能，如统计作图，而运算出来的结果也不具有良好的可视化性。

本文基于 SparkR 对几种分类算法进行并行化，同时包含了 Spark 和 R 的优势，算法执行速度快、统计计算分析能力强，并且能够将计算出来的结果以图表等可视化的形式进行展现。

2 相关概念与原理

2.1 Spark介绍

Spark 实现了一种支持工作集（working set）重用的集群计算模型，也保持了数据流模型的自动容错、位置感知性调度和可伸缩性三大特点。

Spark 为并行编程提供了两个抽象：RDD（Resilient Distributed Dataset）和并行操作算子，其中 RDD 是 Spark 的核心和基础。RDD 是一种分布式的内存抽象，表示只读的、分区记录的集合，它只能通过在稳定物理存储中的数据集或其它已有的 RDD 上执行一些确定性操作（并行操作中的转换操作）来创建，并且 RDD 仅支持粗粒度转换，即在大量的记录上执行单一的操作。并行操作包括转换（transform）和动作（action）两种类型。转换表示从现有的 RDD 创建一个新的 RDD，动作则表示在 RDD 上执行计算，结果返回一个普通的类型值或将 RDD 中的数据输出到存储系统中。

RDD 之间的依赖关系有两种：窄依赖（narrow dependencies）和宽依赖（wide dependencies）。窄依赖是指父RDD分区至多被一个子RDD的每个分区所依赖；宽依赖是指子 RDD 的多个分区都依赖于父 RDD 的每个分区。区分两种依赖关系对 Spark 作业调度和容错都具有非常重要的意义。

用户对 RDD 的控制还可以通过缓存和分区两个方面来控制。通过将 RDD 在多个并发操作之间缓存起来，进而加速后期对该 RDD 的重用。RDD 也允许用户根据关键字（key）来指定分区顺序，将数据划分到各个分区中。

2.2 SparkR介绍

SparkR[4]是 AMPLab 发布的 R 上的一个包，提供轻量级的前端来使用 Spark。SparkR 提供了 Spark 中弹性分布式数据集（RDD）的 API，用户可以在集群上通过 R shell 交互性的运行 job。除了常见的 RDD 函数式算子 reduce、reduceByKey、groupByKey 和 collect 之外，它也支持利用 lapplyWithPartition 对每个 RDD 的分区进行操作。此外，SparkR 还支持常见的闭包（closure）功能：用户定义的函数中所引用到的变量会自动被发送到集群中其他的机器上。

3 分类算法并行化设计与实现

3.1 Multinomial NaiveBayes多元分类

3.1.1 基本原理

Multinomial NaiveBayes[8]算法处理的是离散型的特征值，可以用于很多多类分类的场景，如文本分类。朴素贝叶斯模型是一种基于概率的、监督性的学习方法。该算法的原理基于贝叶斯定理以及类条件独立的假设，算法的具体表述如下：

设 D 是训练元组和它们相关联的类标号的集合，每个元组用 n 维向量 $\boldsymbol{X}=\{x_1,x_2,\ldots,x_n\}$ 表示，描述的是 n 个特征 A_1，A_2，…，A_n 对元组的测量。

设训练样本共有 m 类，记为

$\boldsymbol{C}=\{C_1, C_2,\ldots, C_m\}$。给定元组 **X**，预测 **X** 属于具有最高后验概率的类 c_{map} (在条件 **X** 下)，即

$$c_{map}=arg\max_{C_i\in C} P(C_i|X)\ 1\le i\le m \quad (1)$$

根据贝叶斯定理，

$$P(C_i|X)=\frac{P(X|C_i)P(C_i)}{P(X)} \quad (2)$$

因为 $P(X)$ 对所有类为常数，所以只需 $P(X|C_i)P(C_i)$ 最大即可。在类条件独立的假设下，

$$P(X\,|\,C_i)=\prod_{k=1}^{n}P(x_k\,|\,C_i) \quad (3)$$

根据式（2）、（3），式（1）可转换为

$$c_{map}=arg\max_{C_i\in C} P(C_i)\prod_{k=1}^{n}P(x_k\,|\,C_i)\ 1\le i\le m \quad (4)$$

其中

$$P(x_k|C_i)=P(A_k\,|\,C_i)^{x_k} \quad (5)$$

即预测元组 **X** 第 k 个特征 A_k 的值 x_k 在 C_i 类的概率等于 C_i 类中特征 A_k 的概率的 x_k 次方。其中 $P(A_k|C_i)$ 表明特征 A_k 在 C_i 中所占的比率。

在式（4）中，由于有很多的条件概率相乘，可能会导致浮点数下溢。而 log 函数是单调递增的，所以在实际计算中 c_{map} 常采用如下公式计算

$$c_{map}=\underset{C_i\in C}{argmax}[logP(C_i)+\sum_{k=1}^{n}logP(x_k\,|\,C_i)]\ 1\le i\le m$$

将（5）式代入得，

$$c_{map}=\underset{C_i\in C}{argmax}[logP(C_i)+\sum_{k=1}^{n}x_k logP(A_k\,|\,C_i)]1\le i\le m \quad (6)$$

其中，类先验概率用 $P(C_i)=\frac{N_{C_i}}{\mathrm{N}}$ 估计，N_{C_i} 表示属于类 C_i 的训练元组数，N 表示总的训练元组数。而条件概率的计算公式如下，

$$P(A_k|C_i)=\frac{T_{i,k}}{\sum_{j=1}^{n}T_{i,\mathrm{j}}} \quad (7)$$

$T_{i,k}$ 表示类 C_i 中特征 A_k 的所有测量值之和，也就是在 C_i 类中的权重。而 $\sum_{j=1}^{n}T_{\mathrm{i,j}}$ 则表示类 C_i 中总的权重。

为了解决 0 概率的问题，通常会引入 Laplace smoothing 的方法，设平滑值为 λ，（7）式变为

$$P(A_k|C_i)=\frac{T_{i,k}+\lambda}{\sum_{j=1}^{n}T_{\mathrm{i,j}}+\lambda\cdot \mathrm{n}} \quad (8)$$

3.1.2 并行化

由（6）式知，该算法在训练过程中，需要统计计算出每个类的概率 $P(C_i)$，以及每个类中每个特征的概率 $P(A_k\,|\,C_i)$，即需要知道每个类的频数以及每个类中每个特征值的总的频数。

考虑到 Spark 对文本的处理的默认方式是按行处理，所以我们将训练数据组织成每行是类标签加特征值的格式，值中间用空格作为分隔符。

训练流程如下：

1）首先读取文件，形成 RDD；

2）然后，对 RDD 进行 map 操作，将每一行映射成(label,(1,features))的形式(即(key,value))，其中 label 表示类标号，1 表示计数，features 是特征值。

3）再根据 label 这个 key 做 reduceByKey(或 combineByKey)操作，相同的 key 对 value 进行相加，得到（label,（N，featuresSum)），N 表示 label 这个类的频数，featuresSum 是在 label 这个类对相同的特征的值进行相加所得到。最终 label 的个数就是类的个数。

这样我们就统计出了每个类出现的频数、类的个数、每个类的每个特征的频数，就可以训练出模型了。

训练算法的伪代码如下：

算法 1： NaiveBayes 并行化的训练算法

输入：集群环境 *master*，训练文本 *dataFile*，平滑值 *lambda*

输出：类概率 *pi*，每个类每个特征概率 *theta*

① 定义函数 *parseVector*，将行字符串 line 映射成 list(label, list(1, features))；

② 根据 *master* 初始化 spark 环境；

③ 读取文本 *dataFile*，形成 RDD；

④ 对 RDD 进行 map，每个元素应用函数 *parseVector*，得到新的 RDD *lines*；

⑤ 将 *lines* 相同 key(类标号)的元组的 value 值相加，即执行 reduceByKey 操作，得到新 RDD *aggre*；

⑥ 将 RDD *aggre* 本地化，即执行 collect 操作，记为 *cltaggr*，变量包含每个类的个数和每个类特征向量的权重；

⑦ 根据 *cltaggr* 求出类的个数 C，根据每个类的元组数计算训练的元组总数 *N*；

⑧ 对 *cltaggr*，根据公式计算 *pi* 和 *theta*。

其对应的训练时 RDD 转换流程如图 1 所示：

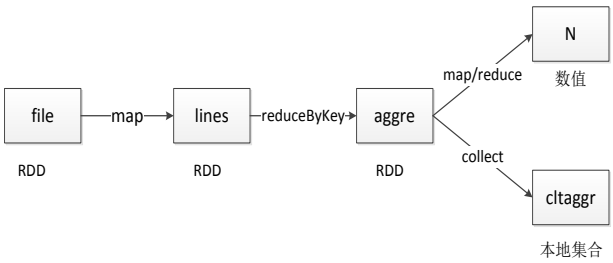


Fig. 1 The RDD transformation in the NaiveBayes training

图 1 NaiveBayes 训练的 RDD 转换流程图

并行化预测的流程相对比较简单，只需要一个 map 函数就可以并发地预测各个待测样本所对应的类标号。首先，读取待预测文件形成 RDD，然后通过 map 函数根据训练出的模型对 RDD 里面的每个元素，按照（6）式，计算该元素（待测样本）属于每个类的概率，把取值最大的那个类作为类标号。因为 SparkR 目前尚未提供 savsAsTextFile 方法，所以只能将这映射后的 RDD collect 为本地集合，然后用 R 的函数保存到文件中。

预测算法的伪代码算法如下：

算法 2：NaiveBayes 并行化的预测算法

输入：类概率 *pi*, 每个类每个特征概率 *theta*, 预测文件 *dataFile*

输出：预测的类标号文件 labelFile

① 定义函数 *predict*，根据 *pi* 和 *theta*，预测元组 d 属于的类标号；

② 读取预测文件，形成 RDD *file*(*file* 的每个元素是字符串，表示一个预测的元组)；

③ 对 RDD *file* 的每个元素按空格划分，转换成 double，形成 RDD *predictData*；

④ 对 *predictData* 执行 *map* 操作，每个元素应用 *predict* 函数，得到预测出的类标号 RDD *predictLabel*；

⑤ 将 *predictLabel* 进行 collect 操作，然后保存到 *labelFile* 文件中。

3.2 二类分类

3.2.1 基本原理

很多二分类算法可归结为一个凸优化的问题，即找到一个最小化凸函数 f 的值，函数f依赖于一个向量变量 **w**（称之为权重），**w** 有 d 个分量。形式上，我们可以将这个写成优化问题，其中目标 $\min_{w\in R^d} f(w)$ 函数的形式如下：

$$f(w)=\lambda R(w)+\frac{1}{n}\sum_{i=1}^{n}L(w;x_i;y_i) \qquad (9)$$

其中，向量 $x_i\in R^d$ 是训练的元组数据，$1\le i\le n$，而 $y_i\in R$ 是训练元组所对应的类标号。

目标函数 f 由两部分组成：1）损失函数。它测量的是模型在训练数据上的误差，损失函数 $L(w;.)$ 必须是 w 上的凸函数。2）正则化函数。它通过惩罚模型 w 的复杂度来得到简单的模型，从而可以避免过拟合等问题。

解决形如 $\min_{w\in R^d} f(w)$ 的最优化问题的一个简单和常用的方法就是梯度下降（gradient descent），它是一阶优化算法，非常适合大规模分布式计算。

梯度下降方法找到一个函数的局部最小值，它通过迭代的方式，沿着函数在当前点的导数（即梯度）的负方向（即陡峭下降的方向）搜索解。算法开始的时候，权重 **w** 被初始为 1 或随机值，梯度下降策略采用贪心的爬山法；每次迭代，算法都向看上去是最优解的方向移动而不回溯，每次迭代都更新权重；最终，**w** 收敛于一个局部最优解。

另一方面，因为目标函数 f 是写成和的形式，并且损失的定义为每个数据点单独的损失的平均值，所以比较适合采用随机子梯度下降（Stochastic subGradient Descent，SGD）方法来解决，这样可以不用一次计算全部数据集的梯度，从而减小了计算开销，进而降低了执行时间。一个随机的子梯度是一个向量的随机化选择，它的期望值是原始目标函数的一个真实的子梯度。均匀随机选择一个数据点 $i\in[1..n]$，我们得到（9）式一个随机子梯度，其相对于 **w** 关系如下：

$$f'_{w,i}=f'_{w,i}+\lambda R'_w \qquad (10)$$

其中，$L'_{w,i}\in R^d$ 是由第 i 个数据点决定的损失函数部分的子梯度，即 $L'_{w,i}\in\frac{\partial}{\partial w}L(w;x_i;y_i)$。而 R'_w 是正则化项 R(w)的子梯度，即 $R'_w\in\frac{\partial}{\partial w}R(w)$。并且 R'_w 是不依赖于随机点的选择的。在随机选择 i 的期望值下，$f'_{w,i}$ 是原始目标函数 f 的一个子梯度，意味着 $\mathrm{E}[f'_{w,i}]\in\frac{\partial}{\partial w}f(w)$。执行 SGD 变成了简单的沿着随机子梯度 $f'_{w,i}$ 的负方向下降，即

$$w_{t+1}=w_t-\gamma f'_{w,i} \qquad (11)$$

参数γ表示步长（step-size），默认实现的是 $\gamma=\frac{s}{\sqrt{t}}$，$t$ 表示迭代的轮数，s 是输入参数 stepSize。

本文实现的是二分类 SVM（Support Vector Machine）线性分类和 Logistic Regression 分类。其中，SVM 和 Logistic Regression 采用的损失函数分别是 hinge-loss 和 logistic loss，优化函数都为 L2 正则优化。

本文后续实现中采用的数据集中训练样本的类标号采用的是 0 和 1，而不是-1 与 1，所以需要对y_i进行转化，公式为 $y_i = 2y_i - 1$，这样得到的 $y_i \in \{-1, 1\}$。并且除了权重 **w** 之外，还需要计算出截距，可以把其当成 w_0，同时还需要把每个元组x_i最前面加常数 1。

SVM 采用的 Hinge Loss 函数形式为：

$$L(w; x_i; y_i) = max\{0, 1 - y_i w^T x_i\}$$

随机子梯度为：

$$L'_{w,i} = \begin{cases} -y_i x_i & 若 y_i w^T x_i < 0 \\ 0 & 否则 \end{cases} \quad (12)$$

LR 实现采用的 Loss 函数形式为：

$$L(w; x_i; y_i) = \log\left(1 + \exp\left(-y_i w^T x_i\right)\right)$$

随机子梯度为：

$$L'_{w,i} = -y_i x_i \left(1 - \frac{1}{1 + exp(-y_i w^T x_i)}\right) \quad (13)$$

L2 正则优化函数为：

$$R(w) = \frac{1}{2} w^2$$

随机子梯度为：

$$R'_w = w \quad (14)$$

式（11）变为

$$w_{t+1} = w_t - \gamma f'_{w,i} = w_t - \gamma\left(L'_{w,i} + \lambda w_t\right) \quad (15)$$

该算法在训练中每轮迭代时，首先随机选取数据集数据形成一个子集 S，然后根据（12）或（13）式计算每个数据点的子梯度，求出均值 $L'_{w,S}$，然后根据（14），（15）式求出新的权重 **w**，继续迭代计算。训练模型流程图如 2 所示。

SVM 预测时，根据训练出的权重 **w** 以及预测的数据向量 **x**，得到

$$y = \begin{cases} 0, w \cdot x < 0 \\ 1, w \cdot x \geq 0 \end{cases} \quad (16)$$

LR 预测时，根据训练出的权重 **w** 以及预测的数据向量 **x**，得到

$$y = \begin{cases} 0, 1/\left(1 + \exp\left(-w \cdot x\right)\right) < 0.5 \\ 1, 1/\left(1 + \exp\left(-w \cdot x\right)\right) \geq 0.5 \end{cases} \quad (17)$$

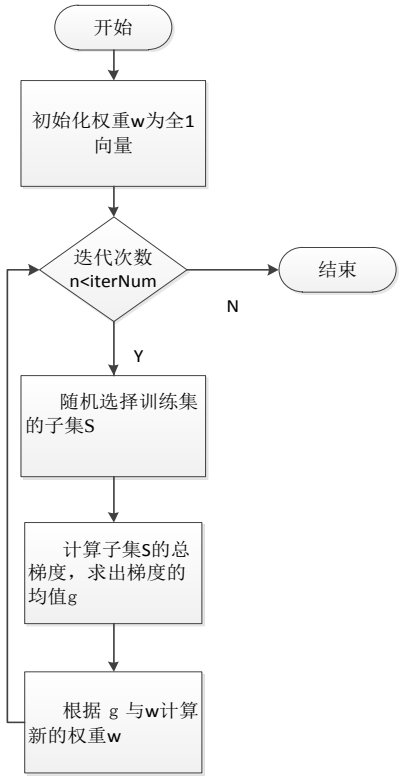


Fig. 2 The workflow of SVM and LR training

图 2 SVM 和 LR 训练流程图

3.2.2 二分类 SVM 和 Logistic Regression 并行化

在上述的流程图中耗时多的部分在计算子集 S 的总梯度上。另一方面，因为训练的元组数目非常多，而每个元组计算梯度之间没有关联性，所以很适合并行计算。

首先，训练文件每一行都是一个类标号 y 和特征向量 **x**。因为 **w** 有 d+1 个元素（w_0 是截距），$\boldsymbol{x} \in R^d$，所以在 map 并行处理的时候，我们需要在 **x** 之前加个常数 1 变成 d+1 维的向量。

基于 SparkR 的并行化训练算法的伪代码如下：

算法 3： SVM 和 LR 并行化的训练算法

输入： 集群环境 *master*，训练文件 *dataFile*，优化参数 *regParam*，迭代次数 *numIter*，步长 *stepSize*，所取训练集合的比例 *miniBF*

输出： 权重 *weights*（第一个值为 *intercept*）

① 定义函数 *parseLine*，将参数字符串 line 映射成（label，features）形式；

② 定义函数 *calGradient*，根据公式计算元组 data 的梯度；

③ 定义正则化函数 *squareL2Updater*，根据公示（15）优化权重；

④ 根据 *master* 初始化 spark 环境；

⑤ 读取训练文本 *dataFile*，形成 RDD *lines*；

⑥ 对 RDD*lines* 进行 map 操作，每个元素应用 *parseLine* 函数，再进行 map 操作，每个元素前加常数 1，形成 RDD *input*；

⑦ 初始化权重 *weights*，值全为 1；

⑧ 将 RDD input 执行 cache 操作，保存到内存中；

⑨ for (*i* in 1:*numIter*)

⑩ 对 RDD *input* 执行 sample 取子集，得到的子集再 map 每个元素应用 *calGradient* 函数计算每个元组梯度，最后对梯度执行 reduce 得总梯度 *value*；

⑪ 根据 weights，总梯度 *value*，正则化参数 *regParam*，步长 *stepSize*，迭代次数 *i*，执行 *squareL2Updater* 函数，得到新权重 *weights*；

⑫ end for

上述训练算法的 RDD 转换的流程如图 3 如下：

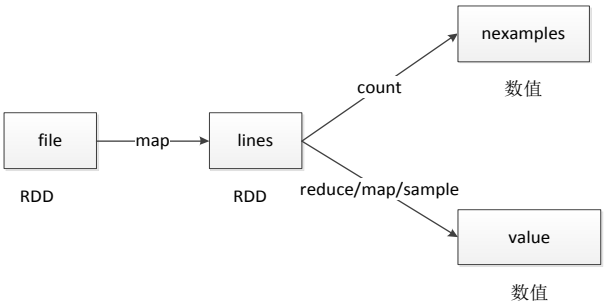


Fig. 3 The RDD transformation in the SVM training

图 3 SVM 训练的 RDD 转换流程图

其中，从 *lines* 的 RDD 到梯度均值 *value* 的计算需要进行 *numIter* 次迭代计算。

与朴素贝叶斯算法类似，在并行化预测时，只需要一个 map 函数就可以并发地对预测各个待测样本所对应的类标号。首先，读取预测文件形成 RDD，根据训练出的权重 **w**，按照（16）或（17）式，计算元组 d 的类标号，这样每个元组都映射成为一个类标号，最后将这映射后的 RDD collect 成本地集合，保存到文件中。预测算法如下：

算法 4： SVM 和 LR 并行化的预测算法

输入：权重 *weights*, 预测文件 *dataFile*

输出：预测的类标号文件 *labelFile*

① 定义函数 *predict*，根据 *weights*，预测元组 d 属于的类标号；

② 读取预测文件，形成 RDD *file*(*file* 的每个元素是字符串，表示一个预测的元组)；

③ 对 RDD *file* 的每个元素按空格划分，转换成 double 型向量形式，形成 RDD *predictData*；

④ 对 *predictData* 执行 *map* 操作，每个元素应用 *predict* 函数，得到预测出的类标号 RDD *predictLabel*；

⑤ 将 *predictLabel* 进行 collect 操作，然后保存到 *labelFile* 文件中。

3.2.3 局部迭代计算优化

在上述实现的并行化训练算法中，每次迭代的时候都需要执行一次 map/reduce 的操作，而 reduce 操作比较耗时。为了进一步降低训练时间，我们设计了一种局部迭代的思路以减少 reduce 操作的次数。本文提出的局部迭代优化中，数据集被分在 N 个分区中，每个分区 P_i 首先基于自己分区内的数据，迭代计算梯度并直接更新本地权重 w_i；等待本地经过一定轮数的训练之后，系统再对 N 个分区的权重求均值得出权重 **w**，并将 **w** 传递给各个分区；收到新的权重之后，每个分区再进行更新，继续根据自己分区的数据迭代更新权重，照此执行满足迭代次数为止，流程图见 4 所示。

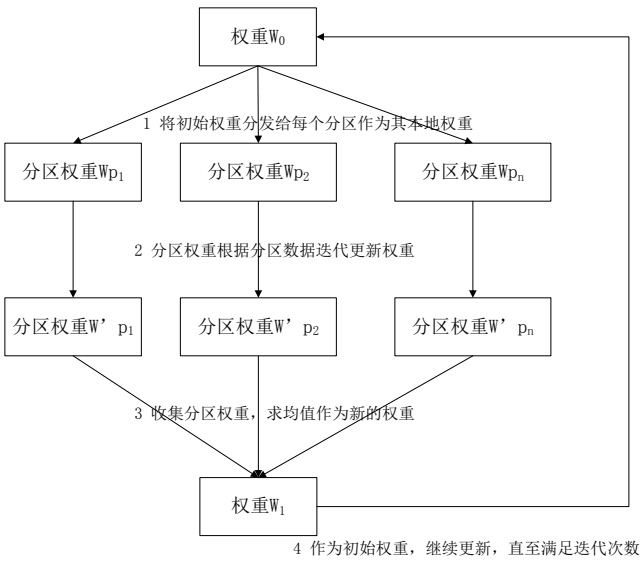


Fig. 4 The workflow of local iterative optimized computation

图 4 局部迭代优化计算流程图

对于上述的优化算法，在训练的元组分布均匀的情况下，各个分区计算的权重差别不大，最后得出的权重损失精度也很少，但是却较多地减少了执行 reduce 的次数，从而减少了训练时间。

下面是针对算法 3 中 for 循环的局部迭代优化后的算法：

算法 5： SVM 和 LR 局部迭代优化算法

输入： 分区内部循环数 *innerIter*

输出： 权重 *weights*

① 定义函数 *calPartitionWeights*，参数为 *part*，具体定义如下：

初始化局部权重 *partWeights* 等于全局权重 *weights*；

for (*iter* in 1:*innerIter*) {

对 *part* 的每个元组应用 *calGradient* 函数得到梯度，再求出平均梯度 *gradient*；

根据 *partWeights*，*gradient*，*stepSize*，*iter*，*regParam* 执行函数 *squareL2Updater* 更新权重 *partWeights*；

end for

② for (*i* in 1:*numIter*) {

③ 对 RDD *input* 执行 *lapplyPartitionsWithIndex* 函数，对 RDD 的每个分区 *part* 应用函数 *calPartitionWeights*，得到表示每个分区权重的 RDD *weightsRdd*；

④ 对 *weightsRdd* 执行 collect 操作，对结果再求均值得到平均的权重 *weights*

⑤ end for

4 实验

4.1 实验环境与设置

我们采用的实验集群由 1 个主控制节点（JobTracker）和 16 个计算节点（TaskTracker）组成，集群的节点配置参见表 1。

Table 1 The configuration of computing nodes

表 1 计算节点配置信息

Item	Information or Setting
CPU	4 Core Intel Xeon 2.4GHz × 2
Memory	24GB
Disk	2 TB SAS × 2
Network Bandwidth	1Gbps
OS	RedHat Enterprise Linux Server 6.0
JVM Version	Java 1.6.0
Hadoop Version	Hadoop 1.0.3

4.2 Multinomial NaiveBayes测试

分区数是对程序并发粒度的一种设定，本项实验的目标是为了评估分区数对训练时间、数据可扩展性、系统可扩展性、性能对比以及预测精度和召回率的影响。所使用的数据集都是保存在 HDFS 上的，前四项实验所采用的数据集来自于 UCI 机器学习库中 pokerhand 数据集[9]，因为前四项实验是评估对大规模数据集的训练时间性能，故采用这个有 100 万样本的数据集。为了便于参照对比，我们还同时在运行测试了基于 Hadoop MapReduce 的这些机器学习算法的性能。

4.2.1 训练时间与并行化分区数关系

为了评估不同数据规模下、不同并行分区数下 NaiveBayes 实际训练时间的变化情况，并且尽量避免因采用复杂性不同的数据集对结果造成的影响，我们选用复杂性相同但大小不同的四组数据进行测试，每组数据分别做 5 趟测试，最后取平均时间，最终形成结果如图 5 所示。

从图示可以看出，对每组相同数据大小的数据而言，随着分区数增加，Multinomial NaiveBayes 的训练时间先减少后增加，比如 100 万样本的数据集的最佳分区数在 20~30 之间。这是因为开始随着分区的变大表明并行度的提高，另一方面也带来了从分区收集数据等所需要的网络通信开销，开始时分区增大会减少每个分区并行计算的时间，但分区到一定数目之后，每个分区计算时间趋于定值，而网络通信的开销逐渐变大，因此训练时间会呈先减少后增加的趋势。

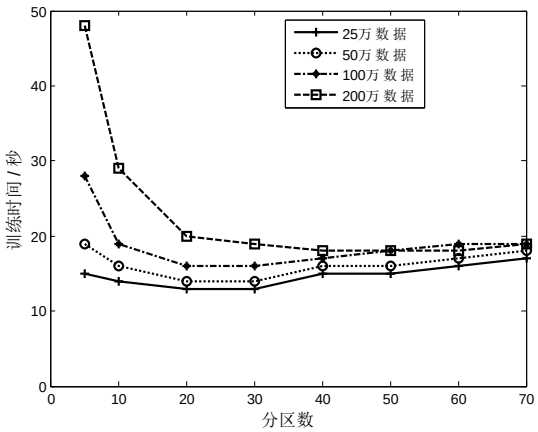


Fig. 5 The training time of different partitions and data size

图 5 分区数、数据量对训练时间的影响图示

此外，数据规模不同，其最优训练时间下的最佳分区数也不一样，从图 5 中可以看出，随着数据规模的扩大（从 25 万到 200 万），最佳分区数也在不断变大（从 20-30 到 40 左右），这是因为数据规

模变大后，增大分区数可以减少每个分区中的数据量，从而降低了每个分区的计算时间。

4.2.2 SparkR 与 Hadoop 实现训练时间对比

本实验实现了 SparkR 与 Hadoop 的 Multinomial NaiveBayes 算法，并在不同数据规模下进行训练时间的对比，数据的复杂性相同。

Table 2 The training time of different data size in the SparkR and Hadoop

表 2 SparkR 与 Hadoop 在不同规模数据集下训练时间

训练集（万）	Hadoop	SparkR	性能提升倍数
25	35 s	13 s	2.69
50	40 s	14 s	2.85
100	49 s	16 s	3.06
200	66 s	18 s	3.67

从表格中可以看出，随着数据的增大，SparkR 的实现与 Hadoop 的实现的时间比在增大，平均要快 2-4 倍。这表明了基于内存计算的 SparkR 比基于磁盘读写的 Hadoop 在算法执行的速度上有一定的优势。

4.2.3 数据可扩展性测试

为了评估数据规模增长时算法实际运行时间的变化情况，从表格 2 中可以得到在四种不同数据规模下（25 万、50 万、100 万和 200 万），Multinomial NaiveBayes 实际训练时间的变化情况，如图 6 所示：

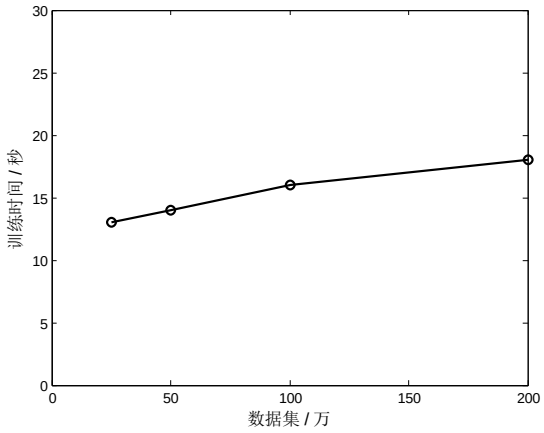


Fig. 6 The relation between training time and data size

图 6 训练时间与数据集数量之间关系图示

从图示可以看出，随着数据集的数量增长，Multinomial NaiveBayes 的训练时间呈线性增长的趋势，即具备了较好的数据可扩展性。

4.2.4 系统可扩展性测试

为了评估当集群规模增大时，Multinomial NaiveBayes 的性能变化情况，我们还进行了系统可扩展性实验。实验测试的数据集大小为 100 万训练元组。

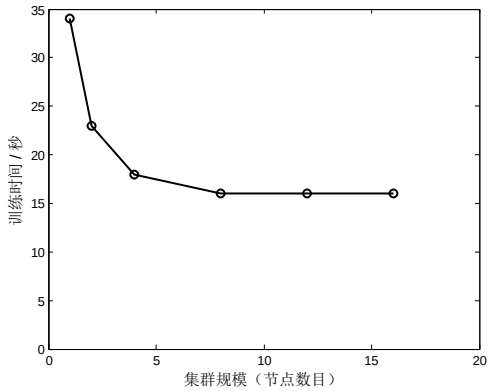


Fig. 7 The relation between training time and the number of nodes

图 7 训练时间与节点个数之间关系图示

从图 7 可以看出，随着节点个数的增加，Multinomial NaiveBayes 的训练时间会显著下降，并最终维持在一个固定水平，这个极限是由单个数据块在单机环境下所需的计算时间加上整个算法并行化的额外开销决定的，由训练时间下降的趋势可见系统还是展示出了较好的可扩展性。

4.2.5 预测精度和召回率验证

为了验证我们并行化之后的算法的有效性，我们对并行化训练的模型的预测精度以及召回率进行了统计，并与算法串行版本的结果进行了对比，结果如表 3 所示。

Table 3 The results of stand-alone and parallel algorithms

表 3 单机和并行化算法实验结果

数据集	正确率 (单机/并行化)	平均召回率 (单机/并行化)
pen-digits	88.9% / 88.9%	77.3% / 77.3%
skin segment	90.9% / 90.9%	95.9% / 95.9%

采用的第一个数据集是手写数字的识别[10]，该集合共有 10 个类别（即 0-9），每个类有 16 个属性。该集合训练数据共 7494 条，预测数据共 3498 条。从表 3 中可以看出，串行执行的准确率和召回率和并行化的执行结果相等，表明算法并行化没有任何损失。

第二个预测的是 Skin Segmentation 数据集[11]，它是根据 RGB 三种颜色判断是否是皮肤的颜色，属于二分类，每个类有三个属性。训练和预测采用同样的数据集，数据集共 245057 组数据。从表 3 中可以看出串行执行的准确率和召回率和并行

化的执行结果相等。

从以上两个预测结果可以看出，Multinomial NaiveBayes 的精度和召回率虽然会随着数据集不同而产生变化，但与串行化的实验结果是相同的，说明了在模型并行化的过程中没有损失正确性。

4.3 二类分类测试

因为 SVM 和 LR 只是损失函数不同，在实验结果上没有很大区别，故在此只列出 SVM 的实验结果。

本节共做 6 项实验，分别为了评估缓存数据集和并行化切分的分区数对训练时间的影响、与 Hadoop 性能对比、准确率和召回率验证、局部优化性能对比、数据可扩展性和系统可扩展性。实验所采用的数据集一个来自于 AVU06a 的 cod-rna 数据[12]，它是 Andrew V Uzilov, Joshua M Keegan 和 David H Mathews 在生物信息学领域中有关检测非编码 RNA 序列的相关研究中所使用的数据集，共 8 个属性。另一个数据集是 Skin Segment 数据集[11]，它是根据 RGB 三种颜色判断是否是皮肤的颜色，共三个属性。

4.3.1 缓存数据和并行化分区数对训练时间的影响

因为 Spark 是基于内存计算的，数据保存在内存中可以直接计算，速度会很快，所以对于迭代计算而言，是否将数据缓存到内存中对训练时间有着重要的影响。本次实验将 cod-rna 数据集所有数据都用于实验，共有 488565 个训练数据，进行 20 轮迭代计算，得到结果如图 8 所示。

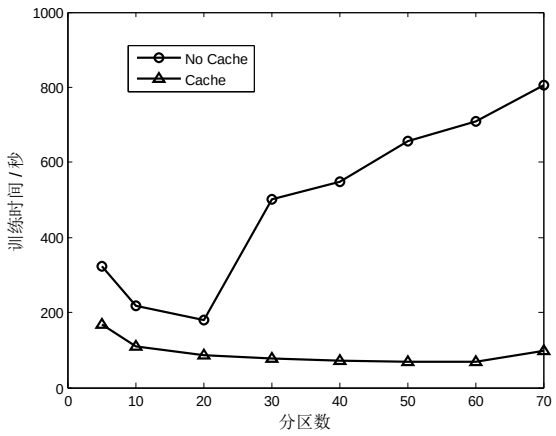


Fig. 8 The training time of different partitions and whether adopting cache or not

图 8 不同分区数、缓存与否的训练时间

从图示中可以看出，当采用缓存策略时，分区数为 50~60 时训练时间最少，为 68s，而不采用缓存时，分区数为 20 左右时，训练时间最少为 180s。表明对于 SVM 算法，训练时间不仅受分区数的影响，更受是否缓存数据影响，从数据中可以看出采用缓存会大大减少训练的时间。

4.3.2 SparkR 与 Hadoop 实现的 SVM 时间对比

本次实验分别实现了 SparkR 和 Hadoop 的二分类 SVM 算法，并针对不同的迭代轮数测试了两种实现的训练时间，采用 cod-rna 数据集[12]共 488565 个训练数据，实验结果见表 4。

表格中的时间比表示的是 SparkR 的缓存实现与 Hadoop 实现的训练时间的比值。从表格中可以看出，缓存后的 SparkR 实现的训练时间平均比 Hadoop 版本的快 10 倍，相对于不缓存的实现则快了 2 倍左右。这表明了基于内存进行迭代计算所具有的速度上的优越性。

Table 4 The training time of different iterations in the SparkR and Hadoop

表 4 SparkR 与 Hadoop 实现不同迭代轮数的训练时间

迭代轮数	Hadoop	SparkR (不缓存)	SparkR (缓存)	性能对比倍数
10	374 s	103 s	43 s	8.7
20	720 s	183 s	68 s	10.6
30	1065s	274 s	94 s	11.3

4.3.3 预测精度和召回率验证

为了验证我们并行化之后的算法的有效性，我们对并行化训练的模型的预测精度以及召回率进行了统计并与算法串行版本的结果进行了对比。

实验采用的训练数据集是 cod-rna 数据集[12]，共 488565 组样本。测试数据是数据集提供的共 271617 组数据。而 Skin Segmen[11]的训练和预测是使用同样的数据集，共 245057 组数据。

SVM 的预测结果的精度是受输入的参数迭代步长（stepSize）以及正则化参数λ 影响的，本实验 cod-rna 数据集设定步长为 1.0，λ 为 0，总迭代次数共 60 次，分区数为 50；skin segment 数据集设定步长为 0.01，λ 为 1.0，总迭代次数共 40 次，分区数为 50。实验结果如表 5 所示：

Table 5 The results of stand-alone and parallel algorithms

表 5 单机和并行化算法实验结果

数据集	正确率 (单机/并行化)	平均召回率 (单机/并行化)	并行化执行时间
cod-rna	88.8% / 88.8%	86.8% / 86.8%	171 s
skin segment	91.8% / 91.8%	90.6% / 90.6%	100 s

从表 5 中可以看出，串行化 SVM 和未采用局部迭代优化的 SVM 执行结果的准确率和召回率是相等的，表明算法并行化过程中没有损失正确性。

4.3.4 SVM 局部迭代优化对比

本实验评估的是文中 3.2.3 节提出的局部迭代优化所带来的时间上与精度、召回率的性能差异。本实验的数据集与 4.3.3 实验相同。因为原始训练数据集是按照类 0 和 1 依次分布的，这里需要将该数据集进行 shuffle 使得数据分布均匀。本实验 SVM 参数设定同 4.3.3 实验，cod-rna 数据集设定步长为 1.0，λ 为 0，总迭代次数共 60 次，分区数为 50；skin segment 数据集设定步长为 0.01，λ 为 1.0，总迭代次数共 40 次，分区数为 50。实验结果如表 6 所示：

Table 6 The performance results of using local iterative optimization in the SVM

表 6 SVM 局部迭代优化实验结果

数据集	分区内部迭代次数	SparkR 局部迭代优化训练时间	正确率	平均召回率
cod-rna	5	43 s	88.9%	86.7%
	10	34 s	88.8%	85.6%
	20	29 s	88.9%	85.1%
skin segment	5	34 s	90.9%	88.2%
	10	26 s	91.0%	88.4%
	20	21 s	91.6%	90.0%

从表 5 中可以看出， 对 cod-rna 数据集，SVM 未采用局部迭代优化的训练时间为 171s，在参数一致的情况下，正确率为 88.8%，平均召回率为 86.8%；对于 skin segment 数据集，SVM 未采用局部迭代优化的训练时间为 100s，在参数一致的情况下，正确率为 91.8%，平均召回率为 90.6%。将上述表 5 结果与表 6 中结果对比，可以看出在数据分布均匀的情形下，局部迭代优化的方法大大减少了时间，同时又不会牺牲很大的正确率和召回率。

4.3.5 数据可扩展性测试

本实验是为了评估 SVM 与局部优化的 SVM 在不同数据规模下的性能情况。数据集的规模分别为 10 万、25 万、50 万、100 万，进行迭代的次数为 40 轮，分区数为 50，优化的分区内部循环数设为 10。实验结果见图 9：

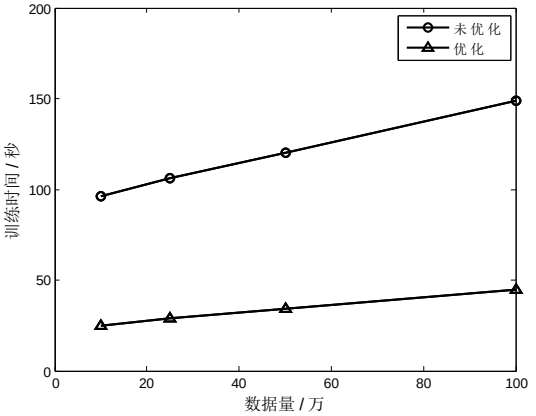


Fig. 9 The training time of different data size in the SVM

图 9 SVM 训练时间随数据规模变化图示

从图示可以看出，随着数据集的数量增长，SVM 无论优化与否，它们的训练时间都是呈线性增长的趋势，即都具备了理想的数据的可扩展性。

4.3.6 系统可扩展性测试

当集群规模增大时，为了评估 SVM 的性能变化情况，我们还进行了系统可扩展性实验，针对不同的节点数测试 40 轮迭代的训练时间。实验测试的 cod-rna 数据集大小为 488565。优化的内部循环数设为 10。实验结果如图 10 所示：

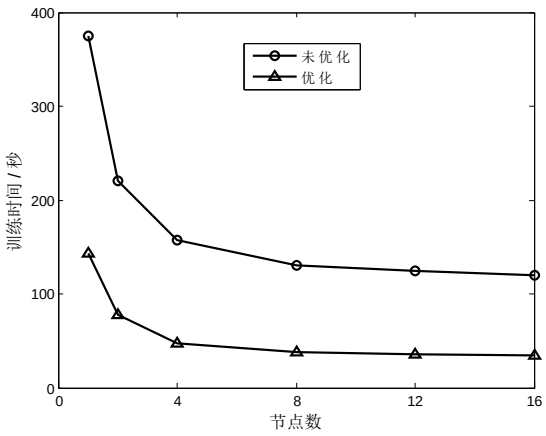


Fig. 10 The training time of different nodes in the SVM

图 10 SVM 训练时间随节点数变化图示

从图中可以看出，随着节点个数的增加，无论是否采用局部迭代优化，SVM 的训练时间开始时（节点数从 1 到 8）会显著下降，节点数大于 8 之后训练时间缓慢下降并最终维持在一个固定水平，这个极限是由单个数据块在单机环境下所需的计算时间加上整个算法并行化的额外开销决定的，由训练时间下降的趋势可见并行化的 SVM 算法还是展示出了较好的系统可扩展性。

5 结束语

针对大数据机器学习的训练速度慢的问题，本文基于 SparkR 实现了多个并行化的分类算法，包括多项式贝叶斯、二分类 SVM、以及 Logistic Regression。通过与 Hadoop 的实现进行对比，验证了其速度上的优越性，并且有良好的扩展性。此外，本文通过对迭代计算进行局部迭代优化进一步降低了训练时间。实验表明该优化策略能有效地提升训练的速度，同时对预测的正确率和召回率基本不造成影响。

下一步工作方面，我们准备在提升算法的训练效率和模型的精度方面进行更加深入的研究。通常，数据中不是所有的特征都对训练起很大作用的，而且有时候数据的特征数量过多也会增加训练的时间。这时候就需要采用特征选择。此外，对于文中提出的 SVM 和 LR 的局部迭代优化，我们还拟定研究其他的计算权重 **w** 的方式以期获得更好的结果。

参 考 文 献

[1] Liu Huayuan, Yuan Qinqin, Wang Baobao. Survey of Parallel Algorithms for Data Mining[J]. Electronic Science and Technology. 2006,1:65-73.
(刘华元,袁琴琴,王保保.并行数据挖掘算法综述.电子科技,2006,1:65-73.)

[2] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters [J]. Communications of the ACM, 2008, 51(1): 107-113..

[3] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing//Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012.

[4] SparkR(R frontend for Spark)[EB/OL], http://amplab-extras.github.io/SparkR-pkg/.

[5] The R Project for Statistical Computing[EB/OL], http://www.r-project.org/.

[6] Apache Mahout: Scalable machine learning and data mining [EB/OL], http://mahout.apache.org/.

[7] Machine Learning Library (MLlib)[EB/OL], http://www.cs.berkeley.edu/~marmbrus/sparkdocs/_site/mllib-guide.html.

[8] Naive Bayes text classification[EB/OL], http://nlp.stanford.edu/IR-book/html/htmledition/naive-bayes-text-classification-1.html, 2009

[9] Poker Hand Data Set[EB/OL], http://archive.ics.uci.edu/ml/datasets/Poker+Hand

[10] Pen-Based Recognition of Handwritten Digits Data Set[EB/OL], http://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits.

[11] Skin Segmentation Data Set[EB/OL], http://archive.ics.uci.edu/ml/datasets/Skin+Segmentation

[12] cod-rna [EB/OL], http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#cod-rna.

[13] Liu Chuang.Research on Classfication Algorithmsbased on Multicore Computing[D]. Nanjing: Nanjing University of Aeronautics and Astronautics, 2011.
(刘闯，基于多核计算的分类数据挖掘算法研究[D]. 南京：南京航空航天大学, 2011.)

[14] Lei Jin, Zhaokang Wang, RongGu, Chunfeng Yuan and Yihua Huang. Training Large Scale Deep Neural Networks on the Intel Xeon Phi Many-core Coprocessor[C]. Proc. of the 2014 IEEE 28th International Parallel & Distributed Processing Symposium Workshops (ParLearning), pp. 1622 - 1630, Phoenix, AZ, USA, May. 19-25, 2014.

[15] Wei Jie, Shi Hongbo, Ji Suqin. Distributed Naive Bayes Text Classification Using Hadoop[J]. Computer Systems and Applications. 2012
（卫洁，石洪波，冀素琴. 基于Hadoop的分布式朴素贝叶斯文本分类[J]. 计算机系统应用, 2012.）

[16] Jiang J, Lu J, Zhang G, et al. Scaling-Up Item-Based Collaborative Filtering Recommendation Algorithm Based on Hadoop[C]. //Services (SERVICES), 2011 IEEE World Congress on. IEEE, 2011:490 - 497.

[17] ZhiDan Zhao，Mingsheng Shang. User-Based Collaborative-Filtering Recommendation Algorithms on Hadoop[C]. 2010 Third International Conference on Knowledge Discovery and Data Mining，2010-01-09

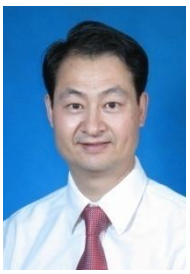
[18] Woodsend K, Gondzio J. Hybrid MPI/OpenMP Parallel Linear Support Vector Machine Training [J]. Journal of Machine Learning Research. 2009,10:1937-1953

[19] Narang A, Gupta R, Joshi A, et al. Highly scalable parallel collaborative filtering algorithm[C]. High Performance Computing (HiPC), 2010 International Conference on. IEEE, 2010:1 - 10.

Liu Zhiqiang, born in 1992, M.S. candidate. His research interests include distributed machine learning.

GuRong, born in 1988. PhD candidate. Student member of China Computer Federation. His research interests include Big Data processing, large scale machine learning and distributed systems.

Yuan Chunfeng, born in 1963. Professor in Nanjing University. Her main research interests include compute system architecture, Big Data processing and Web Information Extraction and Integration.



Huang Yihua, born in 1962. Professor in Department of Computer Science and Technology at Nanjing University. Member of China Computer Federation. His research interests include big data parallel processing and cloud computing, computer architecture and parallel computing.

校对联系人

刘志强 E-mail：MF1433027@smail.nju.edu.cn 手机：15996265823