清 华 大 学

综 合 论 文 训 练

题目：针对罗吉斯回归模型优化算法并行加速的研究

系　　别：计算机科学与技术系

专　　业：计算机科学与技术

姓　　名：彭昊若

指导教师：赵颖　副教授

2013 年 6 月 17 日

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：学校有权保留学位论文的复印件，允许该论文被查阅和借阅；学校可以公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存该论文。

(涉密的学位论文在解密后应遵守此规定)

签　名：_____导师签名：_____日　期：_____

中文摘要

罗吉斯回归模型是一个被广泛应用的有监督学习模型。该模型被广泛应用于机器学习和数据挖掘领域。而在近年来，大规模数据集越来越普遍。而这也给机器学习算法的发展带来了诸多挑战。本文着眼于大数据背景下，使用并行方法解决针对罗吉斯回归模型优化算法的计算效率问题。我们的研究综合考虑了算法与系统，以及两者间的结合问题。

我们研究了三种不同的并行计算系统以及三种不同的算法来有效地提升程序的可扩展性。Hadoop，Mahout 以及 Spark 是当前的三个为人所熟知的系统。它们都能支持大规模机器学习算法在大数据集上的运算。并行梯度下降和随机梯度下降是两个当前非常经典的优化算法，并且可以专门用于解决罗吉斯回归模型的学习问题。另外，我们还开发出一种新的并行次线性算法。我们会比较这些算法在不同并行系统上的运行结果。实验结果显示，在不同的数据集情况以及不同的系统资源下，我们可以选择有针对性的算法更为高效地解决罗吉斯回归模型的学习问题。同时，更进一步的研究表明，对于此种需要较长时间执行的分布式并行计算而言，对节点失败的容错能力既可以体现在系统层面，也可以体现在算法层面，我们可以根据解决问题的需要进行平衡。

关键词：罗吉斯回归模型；大数据；并行计算；次线性方法

ABSTRACT

Logistic regression (LR) is a widely used supervised learning model. It has various applications in the fields of machine learning and data mining. Recently, massive datasets get more and more common. They bring challenges to the development of machine learning algorithms. This paper addresses the issue of computational efficiency for solving LR in big data scenario. We focus on both algorithm level and system level and their integration.

We study three different parallel computation systems with three different types of algorithms to efficiently improve scalability. Hadoop, Mahout and Spark are three existing well-known parallel systems that can be implemented with large scale machine learning algorithms. Parallel gradient Descent and stochastic gradient descent are two existing state-of-art algorithms for parameter optimization in LR. We also present a novel parallel sublinear method. We then make comparisons between these algorithms implemented on different systems. As the outcome shows, we can select different algorithms for the purpose of efficiency in different situations of datasets and machine resources. Moreover, fault tolerance can be provided for those lengthy distributed computations both on algorithm level and system level.

Keywords: Logistic Regression Model; Big Data; Parallel Computing; Sublinear Method

目 录

第1章 引 言

1.1 研究背景

罗吉斯回归模型（Logistic Regression Model）[14]在机器学习和数据挖掘领域的作用和影响越来越大。这个模型主要针对分类问题，并且已经有大量的理论和算法基础来支撑这个模型。罗吉斯回归模型相对于另一种十分流行的支撑向量模型（Support Vector Machines）[25]，同样是很有竞争力的。其优势是，罗吉斯回归模型不仅往往具有很高的拟合精度，而且还有很强的可解释性。这一点体现在罗吉斯回归模型可以直接估计一个类别的条件概率。并且，此优势使得一个可以被罗吉斯回归模型所解决的二分类问题通过简单步骤可以扩展为一个可以被罗吉斯回归模型所解决的多分类问题。在本文中，我们将集中于二分类问题。

在机器学习领域中，解决分类问题的模型有很多。我们之所以选择罗吉斯回归模型，是因为它是一个在数据挖掘领域中常用的且较为快速的分类器。它的统计基础使得在罗吉斯回归模型上的研究往往可以给其他分类问题的模型研究带来启发和指导。

近些年来，大规模数据集越来越普遍，从中演化产生的大规模数据应用也越来越多。这些应用的一个核心特征是，训练数据的样本数量非常大，并且数据维数非常高。例如，在医疗诊断应用中，医生和患者都希望能从上百万条的医疗诊断记录中获得经验，提高诊断正确率，并且一条医疗诊断记录往往有上百维特征可供查询[24]。另一个更明显的例子是现代搜索引擎。这些搜索引擎需要处理十亿级别以上的文本数据以及多媒体数据，并且每条数据的特征空间都在上千维以上[11]。可以想见，如此大规模的数据和高维度特征空间会对机器学习算法在计算能力上提出很高的要求，带来很大的挑战。

而罗吉斯回归模型作为其中的典型代表，由于其在 PageRank[21]以及 Anti-Spam Filtering[1]方面的广泛应用而使其计算效率问题尤为突出。

1.2 论文工作

本文将从算法层面以及系统层面综合解决罗吉斯回归模型优化算法的计算效率问题。我们将主要采用并行方式来进行算法加速。

在开发针对罗吉斯回归模型优化算法的并行模式的过程中，我们无法回避这样一个问题：选择什么并行系统，以及选择怎样的并行计算框架。在经过深入调研之后，我们选择了三个独特而流行的系统来进行测试，它们是 Hadoop[27]，Mahout[20]以及 Spark[29]。

其中 Hadoop 支持 HDFS[3]和 MapReduce[7]。而 Mahout 更像是一个代码库，既可以建立在 Hadoop 上运行，也可以独立运行。Mahout 不能在严格意义上称作为一个系统，但由于其在罗吉斯回归模型上所支持的算法运行框架是独立于 Hadoop 的，我们在此表述为非严格意义上的系统。同时请读者注意，这并不表明我们认为 Mahout 和 Hadoop 完全在一个层次上，因为大部分 Mahout 所支持的机器学习算法还是依赖 Hadoop 的。

Spark 系统最大的特色在于其对含迭代过程算法的良好支持。它采用了更好的内存和缓存控制策略，并且支持 HDFS。我们将会在相关工作部分分别介绍它们的特点。

在针对罗吉斯回归模型的优化算法中，有过很多经典的串行算法。其中的一个代表性例子是随机近似方法。随机近似方法，具体而言又有随机梯度下降（Stochastic Gradient Descent）方法[30]以及随机对偶平均（Stochastic Dual Averaging）方法[28]。它们都可以通过极少量的数据迭代，来趋近于最优优化结果。因而，在训练过程中，当控制预期优化趋近率时，这些算法的运行时间都是正比于数据总规模的，即呈线性。其中最著名的当属随机梯度下降方法。此算法以在线（online）方式运行，我们可以把随机抽取的数据点看作不断引入的数据流。针对此算法进行并行化处理较为困难，但即使是串行，它的运行也十分高效。

如果我们退一步来考察用普通梯度下降算法来解决罗吉斯回归模型的优化问题，尽管它作为串行算法不够高效，但它极易进行并行化处理，从而在大数据背景下大幅提高运行效率，不失为一个潜在的良好算法。我们可以在一次迭代中载入全部数据，以 MapReduce 的形式计算它们的平均梯度下降方向，从而实现并行。更进一步，我们可以使用次线性方法[22]，更进一步的提高运行效率。由于次线性方法可以充分利用在随机近似方法上，所以解决罗吉斯回归模型的优化问题，同样可以利用次线性方法来降低算法复杂度，提高运算效率。次线性方法在每次迭代中只利用训练数据的一维来代替所有维度进行处理，从而实现加速。我们将在后文的算法部分提出基于该算法的一个并行版本。它可以在保证近似率的基础上更为快速的收敛。

1.3 论文结构

第 1 章 引言，介绍了本文的研究背景，论文的主要工作，同时简要概括了一下论文结构。

第 2 章 相关工作，详细介绍了大数据背景下的机器学习算法发展，机器学习算法并行计算框架以及次线性优化方法的发展历程。

第 3 章 带惩罚项的罗吉斯回归模型和次线性方法，详细介绍了罗吉斯回归模型的定义，引入了带惩罚项的罗吉斯回归模型；并对次线性方法做了更深入的介绍。

第 4 章 罗吉斯回归模型优化算法并行框架，介绍了 Hadoop MapReduce 架构、Spark 系统以及 Mahout 软件上所执行的针对罗吉斯回归模型的优化算法。

第 5 章 实验环境，分别介绍了实验数据集信息、集群信息和具体的测试程序。

第 6 章 实验结果，给出了实验在 2D 数据集上的可视化结果，以及所有测试程序在所有数据集上的学习准确度和运行时间，我们分析了上述结果并给出了在不同的数据情况以及不同的系统资源下，选择有针对性算法的建议来更为高效地解决罗吉斯回归模型的学习问题。更进一步的，我们研究了不同计算资源对算法的影响以及算法和平台对节点失败的容错性。

第 7 章 总结，主要回顾了针对解决罗吉斯回归模型的学习问题，在不同数据和不同计算资源下的合理选择，以及次线性方法在此类凸优化问题中的应用。最后提出下一步更深层的研究方向。

第2章 相关工作

2.1 大数据背景下的机器学习算法

在大数据背景下，大规模数据集要求我们开发出的机器学习算法更加高效，并行度更大。在机器学习研究领域，学者们已经针对大规模数据集做了不少卓有成效的工作。

早期的工作，有比方说 PSVM（Parallel Support Vector Machines）[4]的开发。PSVM 利用了近似矩阵分解的方法。这种分解方法是基于行抽取的，它可以在算法运行时大幅降低内存使用。该算法可以通过增加参与并行计算节点的数量来减少算法运行时间。该算法可以运行在拥有上百台机器的集群上。

此后，研究者又提出了 PLDA (Parallel Latent Dirichlet Allocation) [26]算法。该算法可以大幅提高 LDA 算法的运行效率。它采用随机抽样的方法。PLDA 算法所发布的 Hadoop 版本具有很强的鲁棒性，它具有承受节点失败的能力，这其中既利用了算法的随机性，也利用了 Hadoop 的容错优势。

最近，Dean 等学者的工作[18]表明并行化的优势可以在深度学习（deep learning）领域起到重大作用。他们的工作可以把机器学习算法的并行度推进到上亿个计算节点。这是迄今为止机器学习研究领域所达到的最大并行度，同时也促使深度学习算法在图像识别、语音识别和很多其他应用上取得了目前最好的学习效果。

除了 Hadoop 并行系统所代表的框架，机器学习学术界也有很多对其他并行框架的尝试。GraphLab[17]是卡耐基梅隆大学开发的，专门针对机器学习中大规模图算法的一个计算工具。它可以大幅提升此类型算法的运算效率和在集群上的可扩展性。

2.2 并行算法框架

本文的主要工作建立在三个并行系统上，这三个系统在运行机器学习算法时的算法框架都有其各自特点。

2.2.1 Apache Hadoop

Apache Hadoop[27]作为一个软件代码库，实现了对大规模数据集的分布式处理。它被广泛应用与各类机器集群上，并且其使用了简洁的编程模型。它是一个开源并行框架，支持对高密度分布式数据的处理。Hadoop 支持大规模商品级硬件构成的机器集群。Hadoop 是从 Google 的 MapReduce 和 GFS（Google File System）中演化而来的。Hadoop 使用 Java 语言。它的开源特性保证了全球大量开发者的贡献都体现在了这个 Apache 的顶层项目中。Hadoop 平台在一般意义上包含 Hadoop kernel，MapReduce 和 Hadoop Distributed File System (HDFS)，以及一些其他相关的项目，包括 Apache Hive 和 Apache HBase。Hadoop 的设计目标就是要大幅提高机器集群的可扩展性，从几台服务器上升到成千上万个运算节点。在 Hadoop 中每个节点都能提供本地的计算和存储。Hadoop 框架对运行在其上的应用透明地提供数据可靠性保证以及数据通信保证。

Hadoop 所执行的运算模型为 MapReduce[7]。每个应用被分割为大量小片段，每个片段可以在机器集群中的任意一个节点上执行和重复执行。在 MapReduce 运算模型能够应用的问题中，输入会被解析为键-值对的集合。一个 map 函数会把这些键值对经过用户所定义的计算转为中间结果的键-值对。一个 reduce 函数会接着把这些中间结果按照键值索引聚集（其聚集方法是用户所定义的）。事实上，MapReduce 所能应用的问题很广泛，不少问题都可以转化为一个或者一系列 MapReduce 任务。这种运算模型也使得它很易于底层的并行实现。所有的处理都是独立的，所以可以分配给不同的节点分别执行。除此之外，Hadoop 还提供了分布式的文件系统 HDFS。它可以给整个机器集群提供很高的通信带宽。使用 map/reduce 以及 HDFS 的设计保证了当节点失败时，系统可以自动处理错误而不影响整个任务执行。而这种容错性保证，并不依赖于硬件，而是 Hadoop 代码库本身的设计优势。它在应用层考虑了节点失败的检测和处理。所以尽管每个节点都是可能易于出现问题的，但整个机器集群作为一个独立的运算资源来说是可靠的。

2.2.2 Apache Mahout

Apache Mahout[20]是一个专门针对大规模机器学习算法所开发的代码库。其核心设计目标就是要在大规模数据上，提高机器学习算法的可扩展性。它的核心算法包括聚类算法、分类算法和协同过滤。这些算法的很大一部分是建立在 Apache Hadoop 的基础之上运用 MapReduce 框架开发的。同时，Mahout 的核心代码库也

含有一部分非并行计算程序（针对罗吉斯回归模型的算法就属于这一部分）。它们在代码优化上做了大量的工作，从而达到了很好的运算效果。Mahout 甚至广泛应用与商业领域，它良好的可扩展性，以及对很多流行机器学习算法的完整支持都促进了它的推广。

目前，Mahout 主要支持以下四个应用场景：

1. 推荐系统：把用户行为作为输入，并根据用户行为进行个性化推荐。
2. 聚类：例如把文档按照主题进行分类。
3. 分类：例如从已经分好类的文档中学习模型，并将一个新的未被分类的文档分入正确的分类中。
4. 频繁模式：将一系列事项集合作为输入（例如查询词、销售图表等等），从中识别出哪些事项模式出现得较为频繁。

Mahout 当前支持很多机器学习算法，包括协同过滤算法、基于用户和基于产品的推荐算法、K-Means 算法、模糊 K-means 聚类算法、平均偏移聚类算法、狄利克雷过程聚类算法、LDA 算法、奇异值分解算法、并行频繁模式算法、互补朴素贝叶斯分类器和随机决策树分类器。

2.2.3 Spark

Spark[29]系统是由加州大学伯克利分校的 AMPLab 开发的。它是一个开源的分布式计算系统。Spark 的设计目标是使得数据分析更加快速：一方面是运行时间的加速，另一方面是读写的加速。为了使程序运行得更快，Spark 提供了基于内存和缓存控制的集群计算基本操作：一个任务可以将数据最充分地载入内存中，从而当反复不断地访问数据时，内存访问会比硬盘读写更快。而硬盘读写多这一点，正是 Hadoop 系统的最大劣势。为了使得在 Spark 系统上的编程开发更为快速，Spark 支持 Scala 和 Python 语言，并提供了简洁的 API。用户甚至可以以交互式的方式，通过 Scala 和 Python 语言来快速处理大数据集。Spark 系统在设计之初，主要是面向两类应用问题：迭代程序（这是在机器学习算法中广泛使用的）和交互式数据挖掘。如果能够把需要重复利用的数据始终保存在内存中，这两类应用问题的算法执行效率都必然能够获得极大提升。而随后的测试也表明，相比于 Hadoop 系统，Spark 确实在这两个应用中取得了 100 倍以上的加速效果。

Spark 系统较为年轻，但也已经在工业界获得了不少应用。Spark 系统是 Shark 系统的后端引擎，而 Shark 系统是一个与 Apache Hive 系统相兼容的数据仓储系统。同样的，Shark 系统也比 Hive 系统有 100 倍以上的加速。尽管 Spark 系统是

新开发的一个并行框架，它支持访问 HDFS 上的数据，从这个意义上讲，它也与 Hadoop 兼容。这个特性使得很多基于 Hadoop 的程序不用再重新开发，从而给 Spark 也带来了更广阔的应用前景。

Spark 是建立在 Mesos 上的系统。Mesos 是一种运行于集群上的运算系统。Mesos 可以支持多个并行程序良好地共享一个集群。它也同时提供了 API 来支持其上的应用在集群上部署并行任务。Spark 通过利用 Mesos 系统可以在集群上与其他系统良好共存，比如说同样建立在 Mesos 系统上的 Hadoop 和 MPI。另外，利用 Mesos 系统，使得 Spark 的开发过程节约了原本所需的大量编程工作。

Spark 系统得以成功的关键在于运用了弹性分布式数据集（Resilient Distributed Dataset，简称 RDD）这个概念。每个弹性分布式数据集代表一块分布在集群中一部分节点上的只读对象。它具有容错特性，即当一个弹性分布式数据集的分布方式信息丢失时，系统可以自动重构这个弹性分布式数据集，从而保证数据的可靠性。用户可以清晰地在应用程序代码中创建 RDD，并根据需要把它载入整个集群的缓存中，并在像如前所述的 MapReduce 框架下的 map 和 reduce 函数中不断重复访问这些数据。弹性分布式数据集通过仅支持线性操作（lineage）来保证容错性。如果一个弹性分布式数据集丢失，系统可以通过它所对应的线性操作方式（也可以是一串线性操作）来对重新生成这个弹性分布式数据集。

2.3 次线性方法

近些年来，次线性方法逐渐为人所熟知。Clarkson 等学者充分运用随机算法提出了这一新的方法[5]。他们在机器学习领域首先提出了这种次线性时间的近似优化算法。他们把这个方法应用在线性分类器和最小闭包球这两个基础问题上。次线性方法结合了一种新型抽样方法和一种新的可乘式更新算法。他们同时证明了算法下界。该算法下界表明算法理论运行时间对于 RAM 模型来说，已接近最优。

Hazan 等学者把这种次线性方法用在了带二阶惩罚项的支持向量机模型上[16]。此后，Cotter 更进一步，把次线性方法运用在了带核函数的支持向量机模型中[6]。随后，Hazan 继续把此方法做进一步推广，分别发展到了带一阶惩罚项和带二阶惩罚项的线性回归模型中[15]。Garber 和 Hazan 合作，又把此方法用在了半正定规划问题（Semidenfinite Programming）当中[10]。在 2012 年，彭昊若等研究人员，

在次线性方法上做了进一步研究，把它用在了解决带一阶或二阶惩罚项的罗吉斯回归模型中，并提出了对应的串行算法[22]。

第3章 带惩罚项的罗吉斯回归模型和次线性方法

3.1 罗吉斯回归模型

罗吉斯回归模型在机器学习领域中的分类问题上获得了广泛应用。在本文中，特别的，我们将讨论二分类问题。具体的模型定义如下：

3.1.1 数据集定义

假设有一个训练数据集为

$$\mathcal{X} = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \ldots, n\}$$

其中 $\mathbf{x}_i \in \mathbb{R}^d$ 代表输入的训练样本，而 $y_i \in \{-1, 1\}$ 是相对应的分类标签。

这里 n 是样本个数，即样本空间大小；而 d 是样本维度，即特征空间大小。

为了简化表达起见，我们可以将训练数据重组为两部分。一部分为训练数据矩阵，其中每行代表一个训练样本，每列代表一维特征，即

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T$$

另一部分为分类标签向量，其中的每个元素与训练数据矩阵中的每一行对应，即

$$\mathbf{y} = (y_1, y_2, \ldots, y_n)^T$$

3.1.2 模型定义

不带惩罚项的简单罗吉斯回归模型可以表述为下面的形式

$$P(y_i|\mathbf{x}_i) = \frac{1}{1 + \exp(-y_i(\mathbf{x}_i^T\mathbf{w} + b))} \triangleq g_i(y_i),$$

即定义了每个数据样本所对应的分类标签的条件概率。

其中$\mathbf{w} = (w_1, \ldots, w_d)^T \in \mathbb{R}^d$ 是回归向量，而 $b \in \mathbb{R}$ 是偏移量。

这两个变量正是学习算法需要近似优化求解的目标。

3.1.3 优化问题定义

为了优化求解上述变量，对于这种显示表达的模型，一种常用的经典方法是考虑最大似然。更进一步，为了后面的计算方便，这里我们考虑取过对数后的最大似然。由于对数函数本身是单调增函数，所以在优化问题中，优化取对数后的最大似然与优化原始最大似然完全等价。因而，在整个训练数据集上的对数最大似然可以表达为

$$F(\mathbf{w}, b|\mathcal{X}) = \sum_{i=1}^{n} \log g_i(y_i).$$

3.2 带惩罚项的罗吉斯回归模型

在带惩罚项的模型框架中，我们往往需要假设回归向量$\mathbf{w}$服从某一特定分布$p(\mathbf{w})$。这一过程也可以称作先验假设。它使得我们可以将最大似然估计转换为考虑最大后验估计（maximum a posteriori，简称 MAP）。所以，在整个训练数据集上的最大后验估计可以表达为

$$\max_{\mathbf{w},b} \left\{ \log p(\mathbf{w}, b|\mathcal{X}) \propto F(\mathbf{w}, b|\mathcal{X}) + \log p(\mathbf{w}) \right\}. \tag{1}$$

我们下面将会分别推导出一阶和二阶带惩罚项的罗吉斯回归模型优化求解目标的表达式。对于二阶模型，我们会引入拉普拉斯先验假设；对于一阶模型，我们会引入高斯先验。

3.2.1 二阶惩罚项

如果我们假设回归向量$\mathbf{w}$服从一个高斯分布。假设该高斯分布的均值为$\mathbf{0}$，协方差矩阵为$\lambda\mathbf{I}_d$。其中$\mathbf{I}_d$表示一个$d \times d$的单位矩阵。数学上表达为，

$$\mathbf{w} \sim N(\mathbf{0}, \lambda\mathbf{I}_d)$$

那么，由于

$$\log p(\mathbf{w}) = \frac{d}{2} \log \frac{\lambda}{2\pi} - \frac{\lambda}{2}\|\mathbf{w}\|_2^2,$$

在这种情况下，在(1)式中带入上述表达式，得到

$$\max_{\mathbf{w},b} \left\{ F(\mathbf{w}, b|\mathcal{X}) - \frac{\lambda}{2}\|\mathbf{w}\|_2^2 \right\}. \tag{2}$$

从而，我们就把罗吉斯回归模型的参数优化问题发展为一个带二阶惩罚项的优化目标。带二阶惩罚项进行优化的优势在于可以得到更为稳定的优化近似解，在算法迭代过程中促进更为快速的求解收敛。

3.2.2 一阶惩罚项

同样的过程，如果我们假设回归向量$\mathbf{w}$服从一个参数为γ的拉普拉斯分布。我们可以得到，

$$\log p(\mathbf{w}) = d \log \frac{\gamma}{2} - \gamma \|\mathbf{w}\|_1.$$

那么，在这种情况下，在(1)式中带入上述表达式，得到

$$\max_{\mathbf{w},b} \left\{ F(\mathbf{w}, b|\mathcal{X}) - \gamma \|\mathbf{w}\|_1 \right\}. \tag{3}$$

从而，我们就把罗吉斯回归模型的参数优化问题发展为一个带一阶惩罚项的优化目标。

带一阶惩罚项进行优化相比于带二阶惩罚项进行优化往往更有优势。其独特性在于，一阶惩罚项易于进行稀疏模型构建[16]。因而，带一阶惩罚项的罗吉斯回归问题求解得到的结果往往既有良好的分类效果，也可以同时用来进行对数据的特征选择。

3.3 次线性方法

我们所使用的次线性方法的设计框架是一个分别同时处理的硬边际量和软边际量的混合方法。这种方法可以同时在硬边际量和软边际量的优化求解过程中进行快速收敛。

3.3.1 次线性方法算法框架

在次线性方法中，每个迭代过程需要经过两步。第一步是随机原始更新（Stochastic Primal Update）。它又包含以下两个步骤：

1 从样本数据中随机抽取出一个$i \in \{1 \dots n\}$。其中，每个样本数据的被抽取的概率组成长度为n的概率分布向量$\mathbf{p}$。（初始为每个样本等概率，随后概率分布向量$\mathbf{p}$会被不断更新。）

2 通过计算上一步骤被抽取的样本数据$\mathbf{x}_i$的梯度方向，来更新回归向量$\mathbf{w}$以及偏移量b。注意，这是考虑带惩罚项的在线(online)更新过程。

第二步是随机对偶更新（Stochastic Dual Update）。它也包含两个步骤：

1 对训练数据集中的每个数据样本都使用$O(1)$的时间计算一个在该数据样本上的硬边际量加软边际量的近似估计值。

2 使用针对在单纯形（Simplex）上做在线优化的可乘式更新（Multiplicative Updates，简称 MW）算法，来更新第一个步骤中的概率分布向量$\mathbf{p}$。

3.3.2 使用次线性方法的罗吉斯回归模型串行优化算法

我们借助以下符号来定义使用次线性方法的罗吉斯回归模型串行优化算法。我们在引文[22]的基础上做了必要的简化。辅助定义的符号也会在随后涉及的其他算法中使用。

1 我们定义一个投射函数$clip(\cdot)$，如下

$$clip(a,b) \triangleq \max(\min(a,b),-b) \quad a,b \in \mathbb{R}.$$

2 我们用符号$\mathrm{sgn}(\cdot)$代表示性函数，定义为

$$\mathrm{sgn}(x)=\begin{cases}+1 & \text{if } x>0\\ 0 & \text{if } x=0\\ -1 & \text{if } x<0.\end{cases}$$

3 我们用符号$g(\cdot)$来表示罗吉斯函数，即

$$g(x)=\frac{1}{1+e^{-x}}$$

在算法 1 中，我们给出了使用次线性方法的罗吉斯回归模型串行优化算法的具体细节。

在下面所示的算法 1 中，从第 4 行到第 11 行是原始更新部分。其中$coef$是对梯度方向的估计。从第 12 行到第 16 行是对偶更新部分。在这一部分中，σ是对硬边际量和软边际量加和的估计值。而与此同时，σ也是$\mathbf{p}(i)$的梯度值。尽管第 15 行和第 16 行的计算使得$\hat{\sigma}$成为一个对σ的带偏移近似值，但这对保障整个算法的稳定性至关重要。而由此所产生的近似误差对最终优化结果的影响很小，可以忽略不计。在这个算法描述中，我们统一了带一阶惩罚项和二阶惩罚项的次线性方法罗吉斯回归模型串行优化算法。两者不同的处理体现在第 8 行和第 9 行。如果需要更进一步研究，请参考引文[22]中分开定义的两种算法。

算法 1：串行-次线性-罗吉斯回归模型-优化算法

Algorithm 1 SLLR

Input parameters: ε, ν *or* γ, X, Y

Initialize parameters: $T, \eta, \mathbf{u}_0, \mathbf{w}_1, \mathbf{q}_1, b_1$

Iterations: $t = 1 \sim T$

$\mathbf{p}_t \leftarrow \mathbf{q}_t / \|\mathbf{q}_t\|_1$

Choose $i_t \leftarrow i$ with probability $\mathbf{p}(i)$

$coef \leftarrow y_{i_t} g(-y_{i_t}(\mathbf{w}_t{}^T \mathbf{x}_{i_t} + b_t))$

$\mathbf{u}_t \leftarrow \mathbf{u}_{t-1} + \frac{coef}{\sqrt{2T}} \mathbf{x}_{i_t}$

Update soft margin if input ν for ℓ_2-penalty

Update $\mathbf{u}_t$ by soft-threshold operations if input γ for ℓ_1-penalty

$\mathbf{w}_t \leftarrow \mathbf{u}_t / \max\{1, \|\mathbf{u}_t\|_2\}$

$b_t \leftarrow \mathrm{sgn}(\mathbf{p}_t{}^T \mathbf{y})$

Choose $j_t \leftarrow j$ with probability $\mathbf{w}_t(j)^2 / \|\mathbf{w}_t\|_2{}^2$

Iterations: $i = 1 \sim n$

$\sigma \leftarrow \mathbf{x}_i(j_t) \|\mathbf{w}_t\|_2{}^2 / \mathbf{w}_t(j_t) + y_i b_t$

$\hat{\sigma} \leftarrow clip(\sigma, 1/\eta)$

$\mathbf{q}_{t+1}(i) \leftarrow \mathbf{p}_t(i)) 1 - \eta\hat{\sigma} + \eta^2\hat{\sigma}^2)$

Output: $\bar{\mathbf{w}} = \frac{1}{T}\sum_t \mathbf{w}_t, \bar{b} = \frac{1}{T}\sum_t b_t$

第4章 罗吉斯回归模型优化算法并行框架

在此章，我们首先将给出在 Hadoop MapReduce 架构下设计的运用次线性方法的并行优化算法。紧接着，我们将会给出一个在 Spark 系统上运行的运用次线性方法的并行优化算法。该版本算法与 Hadoop MapReduce 架构下设计的算法略有不同。然后我们将介绍在实验测试中作为基准所使用的传统算法。它们包括在 Spark 系统下运行的并行梯度下降方法，以及在 Mahout 中集成的在线随机梯度下降法。

4.1 Hadoop MapReduce 架构下使用次线性方法

我们通过充分利用 MapReduce 的编程模式，在 Hadoop 系统上设计了一个针对罗吉斯回归模型的并行优化算法。具体的算法请看下面的算法 2、原始 Map 过程、原始 Reduce 过程、原始更新过程、对偶 Map 过程和对偶更新过程。

4.1.1 算法框图

算法 2：并行−次线性−罗吉斯回归模型−MapReduce 架构算法

Algorithm 2 PSUBPLR-MR

1: Input parameters: ε, ν *or* γ, X, Y, n, d
2: Initialize parameters: $T, \eta, \mathbf{u}_0, \mathbf{w}_1, \mathbf{q}_1, b_1$
3: Iterations: $t = 1 \sim T$
4: $\mathbf{w}_t$ →storeInHdfsFile("hdfs://paraw").addToDistributedCache()
5: $\mathbf{p}_t$ →storeInHdfsFile("hdfs://parap").addToDistributedCache()
6: conf_primal←new Configuration()
7: job_primal←new MapReduce-Job(conf_primal)
8: conf_primal.passParameters(T, n, d, b_t)
9: job_primal.setInputPath("...")
10: job_primal.setOutputPath("tmp/primal$t$")
11: job_primal.run()

12: $(\mathbf{w}_{t+1}, b_{t+1})$←PrimalUpdate($\mathbf{w}_t, b_t$)

13: Choose $j_t \leftarrow j$ with probability $\mathbf{w}_{t+1}(j)^2 / \|\mathbf{w}_{t+1}\|_2{}^2$

14: $\mathbf{w}_{t+1}$ →storeInHdfsFile("hdfs://paraw").addToDistributedCache()

15: conf_dual ← new Configuration()

16: job_dual ← new MapReduce-Job(conf_dual)

17: conf_dual.passParameters(d, j_t, b_{t+1}, η)

18: job_primal.setInputPath("...")

19: job_dual.setOutputPath("tmp/dual$t$")

20: job_dual.run()

21: $\mathbf{p}_{t+1}$ ←DualUpdate($\mathbf{p}_t$)

22: Output: $\bar{\mathbf{w}} = \frac{1}{T}\sum_t \mathbf{w}_t, \bar{b} = \frac{1}{T}\sum_t b_t$

过程 原始 Map

Procedure Primal-Map(inputfile)

1: Configuration.getParameters(T, n, d, b_t)

2: $\mathbf{w}_t$ ←readCachedHdfsFile("paraw")

3: $\mathbf{p}_t$ ←readCachedHdfsFile("parap")

4: i_t ←parseRowIndx(inputfile)

5: $\mathbf{x}_{i_t}$ ←parseRowVector(inputfile)

6: y_{i_t} ←parseRowLabel(inputfile)

7: r ←*random(seed)*

8: *if* $\mathbf{p}_t(i_t) > \frac{r}{n}$

9: $tmp\_coef = \mathbf{p}_t(i_t) y_{i_t} g(-y_{i_t}(\mathbf{w}_t{}^T \mathbf{x}_{i_t} + b_t))$

10: *else*

11: $tmp\_coef = 0$

12: Iterations: $j = 1 \sim d$

13: Set $key \leftarrow j$

14: Set $value \leftarrow \frac{tmp\_coef}{\sqrt{2T}} \mathbf{x}_{i_t}(j)$

15: *Output* (key, value)

过程：原始 Reduce 过程

Procedure Primal-Reduce(key_in, value_in)

1: key_out←key_in

2: $value\_out \leftarrow \sum_{for\ same\ key\_in} value\_in$

3: *Output* (key_out, value_out)

过程：原始更新过程

Procedure PrimalUpdate($\mathbf{w}_t, b_t$)

1: $\Delta\mathbf{w}_t$ ←readFromHdfsFile("tmp/primal$t$")

2: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \Delta\mathbf{w}_t$

3: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_{t+1} / \max\{1, \|\mathbf{w}_{t+1}\|_2\}$

4: $b_{t+1} \leftarrow \text{sgn}({\mathbf{p}_t}^T \mathbf{y})$

过程：对偶 Map 过程

Procedure Dual-Map(inputfile)

1: Configuration.getParameters(d, j_t, b_{t+1}, η)

2: $\mathbf{w}_{t+1}$ ←readCachedHdfsFile("paraw")

3: i_t ←parseRowIndx(inputfile)

4: $\mathbf{x}_{i_t}$ ←parseRowVector(inputfile)

5: y_{i_t} ←parseRowLabel(inputfile)

6: $\sigma \leftarrow \mathbf{x}_{i_t}(j_t)\|\mathbf{w}_{t+1}\|_2{}^2 / \mathbf{w}_{t+1}(j_t) + y_{i_t} b_{t+1}$

7: $\hat{\sigma} \leftarrow clip(\sigma, 1/\eta)$

8: $res \leftarrow 1 - \eta\hat{\sigma} + \eta^2\hat{\sigma}^2$

9: $key \leftarrow i_t$

10: $value \leftarrow res$

11: *Output* (key, value)

过程：对偶更新过程

Procedure DualUpdate ($\mathbf{p}_t$)
1: $\mathbf{var} \leftarrow$readFromHdfsFile("tmp/dual$t$")
2: Iterations: $j = 1 \sim n$
3: $\mathbf{p}_{t+1}(j) \leftarrow \mathbf{p}_t(j) * \mathbf{var}(j)$

4.1.2 算法描述

算法 2 的并行设计基本遵循了使用次线性方法的罗吉斯回归模型串行优化算法的算法框架。我们在并行算法 2 中依然在每次迭代计算中保留了两个主要部分：从第 4 行到第 12 行是原始更新过程，而从第 13 开始，一直到第 21 行是对偶更新过程。在原始更新过程中，有并行执行的部分，它们是从第 4 行到第 11 行部分；但同时，也又无法避免的串行部分，即由第 12 行的调用 PrimalUpdate 函数体现。而在对偶更新的过程中，情况是类似的，从第 14 行到第 21 行是并行执行的部分，但第 13 行和 21 行是串行执行的。

在算法 2 的并行执行部分，我们充分考虑了 MapReduce 本身的设计特点。在算法 2 的 job_primal 中，我们将训练数据矩阵$\mathbf{X}$全部载入并进行解析和处理，这符合 MapReduce 以统一方式同时处理大量数据的设计思想。在原本的次线性串行算法中，我们在原始更新的过程中只随机抽取一个样本数据用来计算梯度；而在新设计的并行框架中，我们计算了“一部分”样本数据上的梯度，并根据概率向量$\mathbf{p}$进行了加权叠加，并以 Reduce 过程的最终结果作为更新的梯度。实现的细节具体参考原始 Map 过程和原始 Reduce 过程。在此，我们使用随机方法来确定计算哪些样本数据上的梯度。从原始 Map 过程的第 7 行和第 8 行可以看出，如果取 $r = 0$，那么所有的样本节点都将参与梯度计算。又由于概率向量$\mathbf{p}$中$\mathbf{p}_t(i_t)$的期望值是$\frac{1}{n}$，所以在算法执行中，我们一般会取$0 \leq r \leq 1$。

在算法 2 的 job_dual 中，我们设计的是原始串行算法的简单并行处理。并行算法对每个样本数据根据可乘式更新算法计算出一个值。这个过程可以直接分散到各个计算节点上单独完成，甚至可以省略 Reduce 的过程。

4.1.3 进一步增大并行度

更进一步考察，我们事实上可以在实际程序执行中进一步增大并行度，其方法是在同一次迭代计算过程中，同时启动原始 MapReduce 任务和对偶 MapReduce 任务。由于这两个 MapReduce 任务中所访问和修改的参数通过延后更新的准则考

虑可以互相隔离，所以这样同时执行的方式和两个任务串行执行的方式是等价的。从而，我们可以在 Hadoop MapReduce 的框架下更充分地增大并行度，是算法执行更为高效。这样的并行设计在图 4.1 中得到了更加清晰的说明。

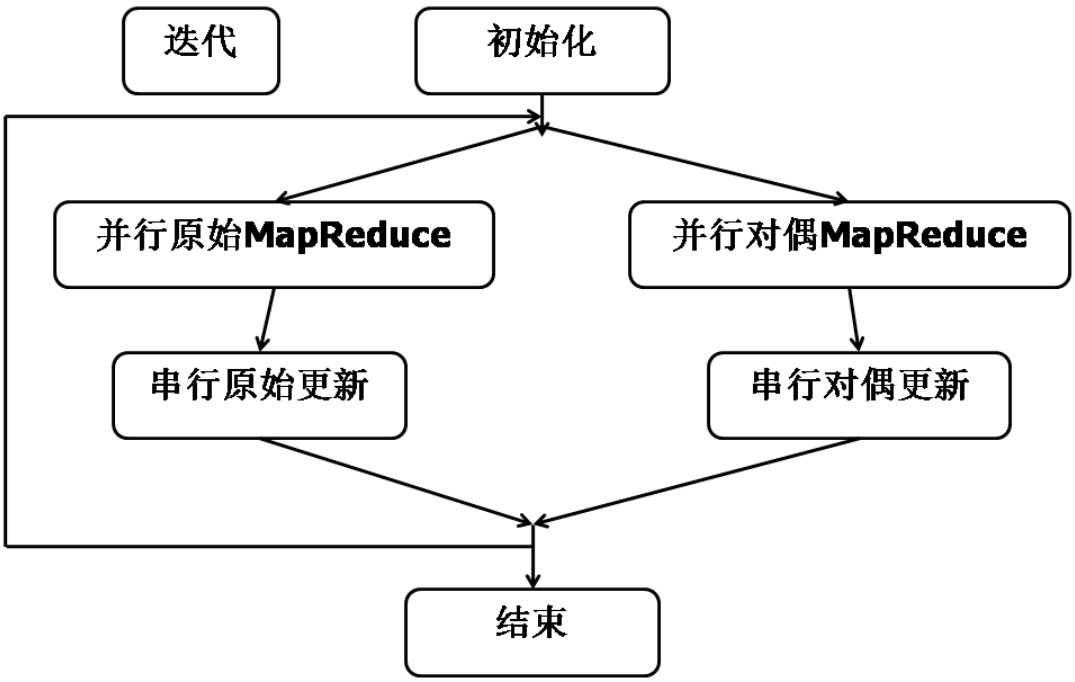

图 4.1 算法 2 并行框架设计图

4.1.4 其他设计问题

1 有关参数传递的问题

在 Hadoop 框架下，合适地选择一种高效传递参数的方式对性能是尤为关键的。在算法 2 中，参数传递的过程既有不同迭代步之间的传递，也有不同 MapReduce 任务之间的传递。同时，另一个挑战是，需要针对 Hadoop 框架下的 HDFS 有针对性的进行处理。

很显然，最为快速的通信方式是通过$Configuration()$函数进行参数传递。但是，这样的传参方法在 Hadoop 环境下总是会遇到缓冲区大小限制的问题。一旦当需要传递的参数很大时（高维数据正符合这一点），如果仅使用此方法，将严重危害算法的可扩展性。

另一种方法是把需要传递的参数在信道传递时进行信息压缩。这需要在发送时进行压缩，在接收时进行解压。其弊端是会造成额外的计算开销。并且这种方

法也不能从本质上解决参数传递时缓冲区大小限制的问题。

当我们重新回顾 Hadoop 的设计思想时，可以发现，最自然而然的选择是通过文件传递参数。因为 Hadoop 本身在自动连接 Map 任务和 Reduce 任务之间就使用了通过文件传递键-值对的方法。当然，由于频繁的读写 HDFS 系统，这种方法并不高效，但这是为了支持大数据环境而采取的合理选择。

2 有关一阶和二阶惩罚项的问题

算法 2 在数学上严格来说是针对不带惩罚项的罗吉斯回归模型的。如果要是模型带有一阶或二阶惩罚项，我们必须对算法 2 进行一些小的改动。这些改动只会在原始更新的过程中出现。而改动方法和算法 1 中第 8 和第 9 行所示的方法是完全一致的。这些改动仅发正在串行执行部分，对本身的算法框架和执行没有本质影响。

3 训练数据集的数据稀疏问题

可以被罗吉斯模型所描述的训练数据集，如果具有很高的维度，往往具有数据稀疏的特点。这就要求我们在真正实现算法时，充分考虑这一点，在代码中进行有针对性的优化。但这一点在算法 2 自身的描述中难以体现，在此特别说明。

我们放弃了针对稠密数据的简单存储和运算模式。针对数据稀疏性，我们会分别存储每个样本数据对应的稀疏向量中每个数据点的维度坐标和具体数值。采用这样的方式，自然也会使得所有涉及稀疏向量的运算都要对代码进行相应更改优化，比如说最简单的稀疏向量求点积过程。考虑数据稀疏性的代码优化是极为有效的，也对算法性能带来了显著提升。我们在后文所展示的实验结果均为使用针对数据稀疏性优化后的代码进行测试所得。

4.2 Spark 系统上使用次线性方法

我们设计了在 Spark 系统上运行的运用次线性方法的并行优化算法。具体的算法请看下面的算法 3。而其中所使用的原始更新过程和对偶更新过程是与前面介绍的算法 2 完全相同的。

该版本算法与 Hadoop MapReduce 架构下设计的算法略有不同。然后我们将介绍在我们测试中作为基准所使用的传统算法。它们包括在 Spark 系统下运行的并行梯度下降方法，以及在 Mahout 中集成的在线随机梯度下降法。

4.2.1 算法框图

算法 3 并行-次线性-罗吉斯回归模型-Spark 架构算法

Algorithm 3 PSUBPLR-SPARK

1: Input parameters: $\varepsilon, \nu \ or \ \gamma, X, Y, n, d$

2: Initialize parameters: $T, \eta, \mathbf{u}_0, \mathbf{w}_1, \mathbf{q}_1, b_1$

3: ps←spark.textFile(inputfile).map(parsePoint()).cache()

4: Iterations: $t = 1 \sim T$

5: g←ps.map($(1/(1+e^{-y(\mathbf{w}^T\mathbf{x}+b)})-1)*y*\mathbf{p}[index]$).reduce(_+_)

6: $(\mathbf{w}_{t+1}, b_{t+1})$←PrimalUpdate($\mathbf{w}_t, b_t$)

7: Choose $j_t \leftarrow j$ with probability $\mathbf{w}_{t+1}(j)^2/\|\mathbf{w}_{t+1}\|_2{}^2$

8: pAdjust←points.map(MW-Update()).reduce(copy())

9: $\mathbf{p}_{t+1}$ ←DualUpdate($\mathbf{p}_t$)

10: Output: $(\mathbf{w}, b)$

4.2.2 算法描述

算法 3 与算法 2 在基本并行框架上的设计非常相似。而它们最主要的区别在于算法 3 的第 3 行执行了$cache()$操作。为了使得算法能够适合于 Spark 环境，我们遵循 Spark 的规则生成 RDD 来更合理高效的利用内存和缓存。

同样，考虑到数据稀疏性问题，我们设计的 RDD 也采用了存储稀疏向量每个数据点的维度坐标和具体数值的方式。同样的，数据点的维度坐标将和数据点具体数值一样参与针对稀疏向量专门设计的运算。

而针对带惩罚项的罗吉斯回归问题，算法所需要的改动由于体现在原始更新的过程中。由于这一部分算法 3 与算法 2 一致，其改动方法也是完全相同的。

4.2.3 时间复杂度理论分析

现在，我们可以着手分析罗吉斯回归模型优化问题并行算法的时间复杂度。在理想的并行状态下，原始更新的串行部分需要更新回归向量$\mathbf{w}_t$，这需要$O(n)$的时间来完成；理想条件下，并行部分可以看作在$O(1)$时间内完成。而在对偶更新过程中，串行部分含有一个随机抽样过程来抽取j_t，需要$O(d)$的时间；并行部分更新概率向量$\mathbf{p}$，可以看作在$O(1)$时间内完成。所以，总的来说，每个迭代需要$O(n+d)$的时间，这正体现出了对训练数据集的次线性。

将此结果和算法 1 所代表的次线性串行算法相比较，可以发现两者相同，即上述的并行算法设计并没有降低算法复杂度。但是考虑常量的话，并行算法可以把常数由 2 降为 1，即在完全不考虑额外计算和通信开销的情况下，理论上速度能快一倍。更进一步，如果同前所述，将每次迭代过程中的两个 MapReduce 任务同时启动，那么理论上的算法复杂度可以降为$O(\max\{n,d\})$。

4.3 Spark 系统上使用并行梯度下降法

我们设计了在 Spark 系统上运行的并行梯度下降法。具体的算法请看下面的算法 4。此算法为经典算法，但为了与前文统一描述风格，我们给出如下的算法框图。

4.3.1 算法框图

算法 4 并行-梯度下降-罗吉斯回归模型-Spark 架构算法

Algorithm 4 PGDPLR-SPARK

1: Input parameters: $\varepsilon, \nu \ or \ \gamma, X, Y, n, d$

2: Initialize parameters: $T, \eta, \mathbf{u}_0, \mathbf{w}_1, \mathbf{q}_1, b_1$

3: ps←spark.textFile(inputfile).map(parsePoint()).cache()

4: Iterations: $t = 1 \sim T$

5: gradient←ps.map($(1/(1+e^{-y(\mathbf{w}^T\mathbf{x}+b)})-1)*y$).reduce(_+_)

6: $\mathbf{w}_{t+1} = \mathbf{w}_t - gradient * \mathbf{x}$

7: $b = b - gradient$

8: Output: $(\mathbf{w}, b)$

4.3.2 算法描述

算法 4 与其对应的串行算法而言，是经过了最为简单直接的并行化处理。算法 4 每次载入所有数据，并以类似于 MapReduce 任务的方式来计算出平均梯度。而至于算法 4 的$cache()$函数运用以及针对数据稀疏性而特别设计的 RDD 都与算法 3 一致，在此不做赘述。

4.4 Mahout 系统上使用在线随机梯度下降法

尽管随机梯度下降方法本质上是一个串行算法，但它运行非常高效。并且又由于是在线算法，运行时内存占用低。所以此算法并不影响 Mahout 处理百万数量级以上的训练样本集。由于该算法采用自顶向下的抽样方法，这样的可扩展性其实是和其他算法直接处理十亿级别以上的数据量等价。在线算法其实也是一个增量式训练模式，并且我们还可以在训练的过程中就进行性能测试。这样从而可以使我们在训练模型达到预期训练效果时就终止训练过程。

在 Mahout 中使用的随机梯度下降模块包含使用交叉验证（Cross Validation）的在线评估方法（CrossFoldLearner）和一个革命性的可以在运算过程中继续进行超参数优化的系统(AdaptiveLogisticRegression)。这个系统大量使用多线程来弥补并行上的不足，从而提高机器节点利用效率。Mahout 会控制一系列的 CrossFoldLearners 运行在不同的线程上。其中的每个学习器在进行学习时都设定了不同的学习参数。当更好的学习设定被找到时，这些新的学习设定将会被系统传播到其他学习器上。

由于随机梯度下降的算法需要固定长度的特征向量，又因为在学习过程之前就建立完全索引的开销太大，大部分的随机梯度算法使用哈希后的特征向量空间系统。这个系统来自于 RandomAccessSparseVector。用户可以根据需求，使用多种特征编码方式来大幅提升这个向量的特征维数。而由于要满足哈希条件，所以向量需要足够大来防止特征冲突。Mahout 中本身集成了很多专门为各类数据类型提供的特征编码器。一般而言，用于可以将数据编码为字符串形式，或者为了避免字符串转换操作而把数据按字节编码。

在我们的实际实验执行中，我们直接使用 RandomAccessSparseVector 来应对数据稀疏性，并用 OnlineLogisticRegression 来进行模型训练。同时也进行交叉验证。为了使得该算法的程序能够更好地与其他算法程序兼容，我们自己写出代码进行交叉验证。

第5章 实验环境

在本章中，我们会介绍实验数据集的细节信息，实验集群信息和所参与实验比较的测试程序。

5.1 实验数据集信息

我们选择了 5 个数据集来进行测试：

1. 仿真 2D 数据集：由电脑依据用户给出的限制信息，自动随机生成。数据集含有 200 个数据样本，每个数据样本有 2 维特征。可以在平面上直观显示。用于初步测试和调试和检验测试程序正确性。
2. 20NewsGroup 数据集：来自于美国时代周刊文本分类信息。是著名的运用于罗吉斯回归模型的数据集。我们选择了其中 rec.sport.baseball 和 rec.sport.hockey 两个主题的文本信息，从而进行二分类实验。数据集中正例和反例的数量非常接。经过处理后，数据集含有个 1988 数据样本，每个数据样本有 16428 维特征。我们取其中 1800 个数据样本作为训练集，其他 188 个数据样本作为测试集。
3. Gisette 数据集[13]：来自 UCI 机器学习公共数据集。相对于 20NewsGroup 数据集，该数据集数据量更大，数据也更为稀疏。数据集含有 7000 个数据样本，每个数据样本有 5000 维特征。我们取其中 6000 个数据样本作为训练集，其他 1000 个数据样本作为测试集。
4. ECUESpam 数据集[8]：描述垃圾邮件与非垃圾邮件分类的文本数据。其特点是正例和反例分配并不均衡。尽管该数据集的特征维数比 Gisette 数据集要高，但由于其数据更为稀疏，该数据集有更少的非零元素参与运算。经过处理后，数据集含有个 10678 数据样本，每个数据样本有 100249 维特征。我们取其中 9000 个数据样本作为训练集，其他 1678 个数据样本作为测试集。
5. URL-Reputation 数据集[19]：描述恶意 URL 与正常 URL 分类的文本数据。数据规模和特征维数都十分巨大。原始数据是以 svmlight 格式存

储，超过 2GB。该数据集基本达到了大规模数据的标准，并且在本文的实验环境中超出了 Liblinear 可能承受的计算范围。数据集含有个 2376130 数据样本，每个数据样本有 3231961 维特征。我们取其中 2356130 个数据样本作为训练集，其他 20000 个数据样本作为测试集。

除了仿真 2D 数据集意外，其他四个数据集的数据都是稀疏的。我们对数据集上训练集和测试集的分割是随机的，并且会重复 20 次，所有以下实验数据均是这 20 次交叉验证的结果。数据集的详细情况也可以参见表 5.1。

表 5.1 数据集信息表

名称	特征维数	数据个数	稀疏性	非零元素个数	正例个数与反例个数比
2D	2	200	1.0	400	1.000
20NewsGroup	16248	1988	7.384×10^{-3}	238511	1.006
Gisette	5000	7000	0.12998	4549319	1.000
ECUESpam	100249	10678	2.563×10^{-3}	2746159	5.882
URL-Reputation	3231961	2376130	3.608×10^{-5}	277058644	0.500

5.2 集群信息

本文实验所使用的集群设置如表 5.2 所示。这一配置，在各类研究性实验室也越来越普遍。因而，我们下面所得到的实验结果预计会给学术界带来一定的影响。

表 5.2 集群信息表

CPU 型号	Intel Xeon E5-1410
CPU 主频	2.80GHz
节点数	6
每个节点上 CPU 核数	4 核 8 线程
每个节点内存大小	16G
每个节点硬盘大小	4T HDD
节点间连接方式	Gigabyte Ethernet

5.3 测试程序

在实验中一共有 6 个测试程序参与实验。

1. 在 Mahout 上运行的在线随机梯度下降算法。该程序采用单核多线程方式运行。
2. 直接运行在 Linux 系统上的 Liblinear[9]程序。该程序作为基准。该程序单核串行执行。其对罗吉斯回归问题的优化求解结果在单核的情况下效果极佳。
3. 算法 1 所对应的 SLLR 程序，在单核上执行串行次线性优化算法。
4. 算法 2 所对应的 PSUBPLR-MR 程序，在 Hadoop 环境下的集群上执行并行次线性优化算法。
5. 算法 4 所对应的 PGDPLR-SPARK 程序，在 Spark 环境下的集群上执行并行梯度下降优化算法。
6. 算法 3 所对应的 PSUBPLR-SPARK 程序，在 Spark 环境下的集群上执行并行次线性优化算法。

第6章 实验结果

在本章，我们将会介绍所有测试程序在各个数据集上的测试结果以及相关对比。

6.1 仿真 2D 数据集上的结果

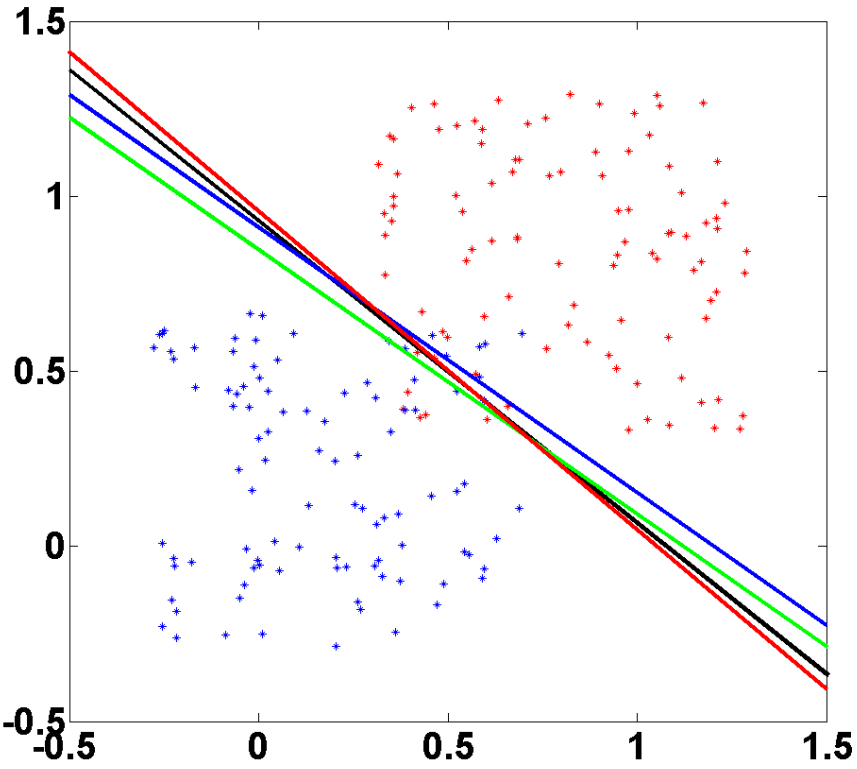

图 6.1 仿真 2D 数据集上的学习结果图

图 6.1 显示了 6 个测试程序在仿真 2D 数据集上的可视化结果。在这里，我们使用所有的数据来进行训练和测试，因而在下面针对仿真 2D 数据集的测试误差（Test Error）也可以被理解为训练误差（Training Error）。其中蓝线代表 Mahout 上在线随机梯度下降算法的学习结果。黑线代表 Liblinear 的学习结果。而绿线则代表所有的次线性方法（包括 SLLR 算法，PSUBPLR-MR 算法和 PSUBPLR-SPARK 算法）的学习结果。在这里，尽管次线性方法带有一定随机性，但在这样的较小规模的数据集上，迭代次数较少，学习结果是几乎一致的。这也

显示了次线性方法尽管具有随机性，但其稳定性仍然是良好的。整体上，这四条由不同测试程序得到的分割线之间也差别不大，相互验证了测试程序的正确性。

6.2 精度结果

表 6.1 学习精度数据表

	2D	20NewsGroup	Gisette	ECUESpam	URL-Reputation
Mahout	93.5%	71.3%	91.5%	85.2%	91.5%
Liblinear	93.0%	92.0%	97.4%	97.1%	96.2%*
SLLR	93.5%	91.5%	94.8%	92.3%	94.2%
PSUBPLR-MR	93.5%	90.5%	94.6%	91.7%	93.8%
PGDPLR-SPARK	93.5%	92.0%	97.0%	93.7%	96.0%
PSUBPLR-SPARK	93.5%	90.5%	95.8%	91.7%	94.0%

6 个测试程序在 5 个数据集上分别取得的精度结果可以参见表 3。这些都是经过交叉验证后得到的平均值。而从图 6.2 到图 6.6，我们测试了 6 个测试程序分别在各个数据集上不同迭代次数下的测试误差。所以，针对每个数据集，我们分别得到了一张有 6 条折线的图。每张图的横坐标为测试程序迭代次数，纵坐标为测试误差。

请注意，在表 3 中有标“*”号的数据，即 Liblinear 在 URL-Reputation 数据集上的测试精度。这里，“*”号表示该结果不是在 URL-Reputation 的全部数据集上获得的。在测试时，由于节点的内存限制，Liblinear 无法执行次规模的数据集在罗吉斯回归模型下的训练。所以，我们只能随机从 URL-Reputation 的训练数据集中随机抽取$\frac{1}{3}$的数据进行训练（该比例也是在本文实验环境下的极限值）。但由于该数据集本身比较规整，使用较小的数据量并没有影响学习出非常准确的模型参数。

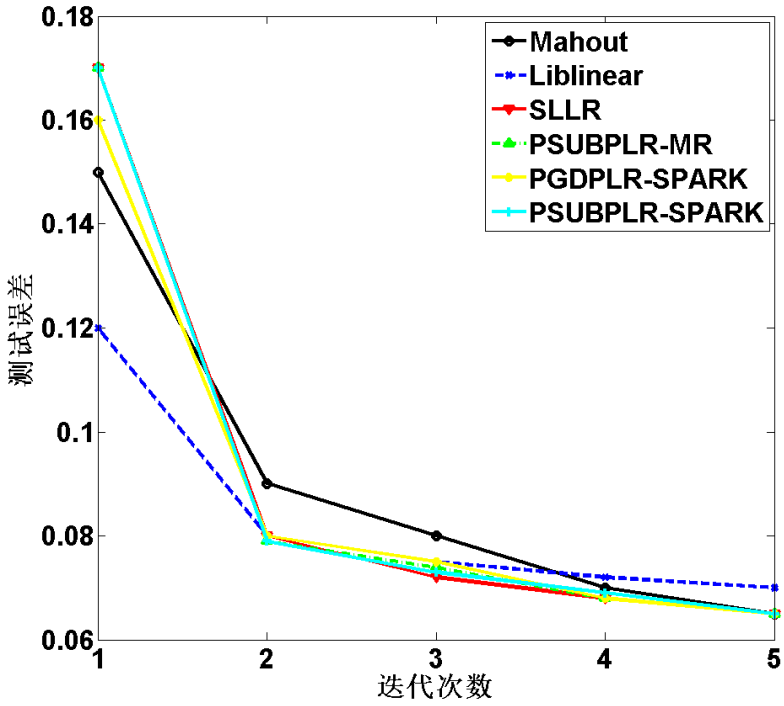

图 6.2 仿真 2D 数据集上的学习过程图

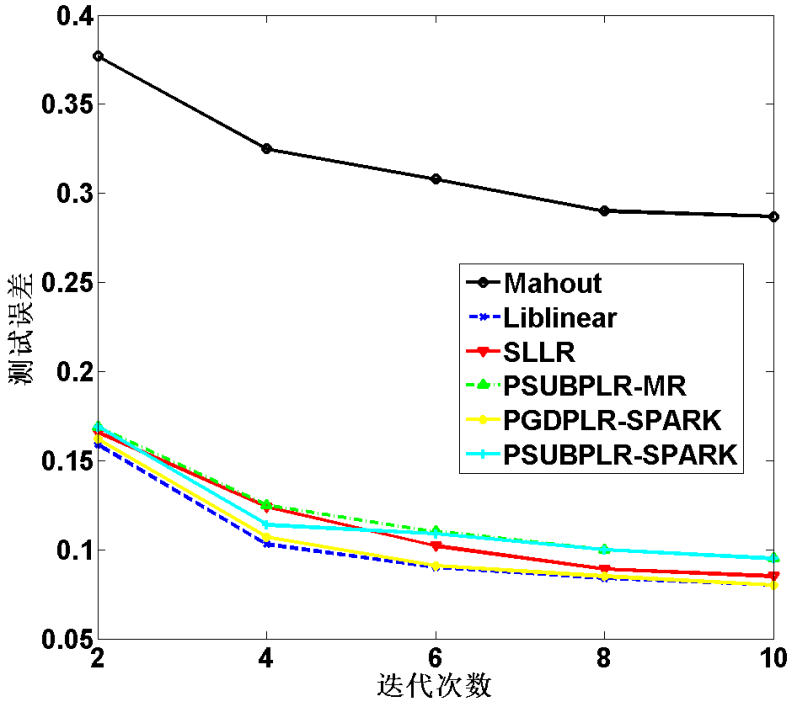

图 6.3 20NewsGroup 数据集上的学习过程图

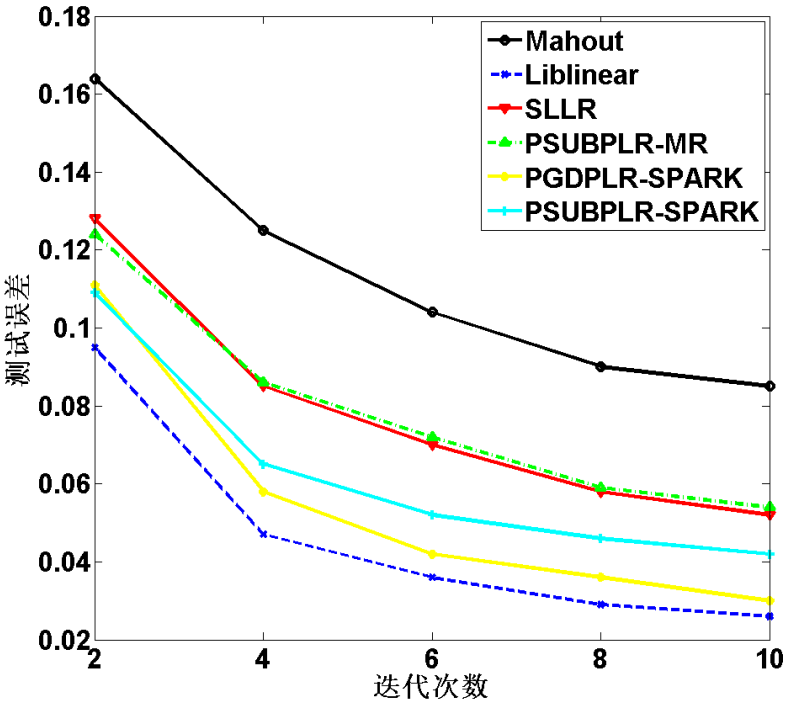

图 6.4 Gisette 数据集上的学习过程图

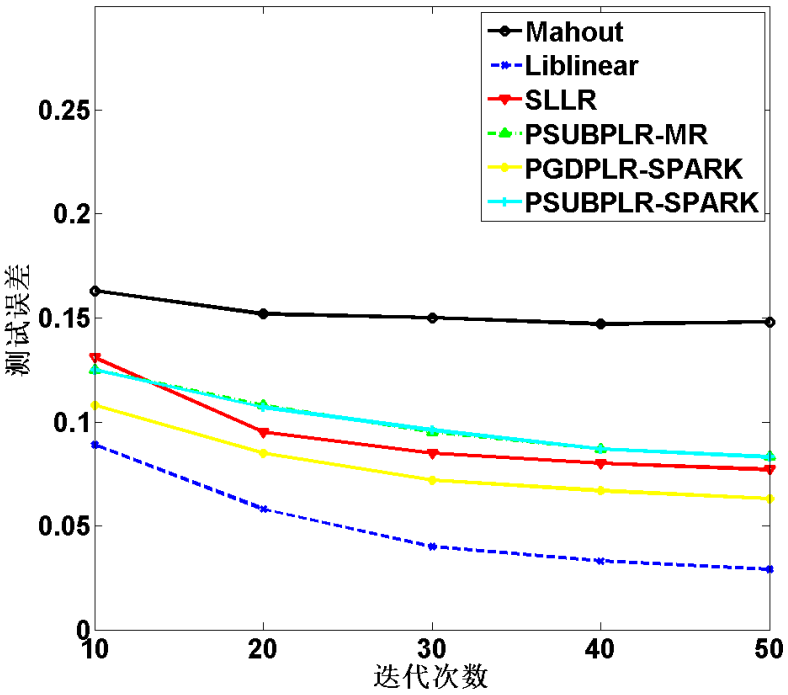

图 6.5 ECUESpam 数据集上的学习过程图

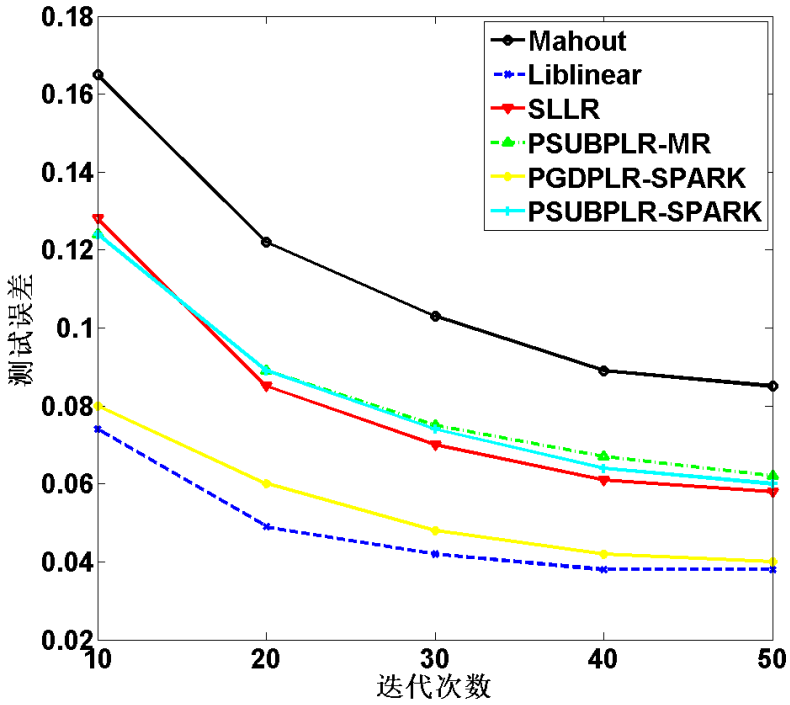

图 6.6 URL-Reputation 数据集上的学习过程图

6.3 训练时间结果

表 6.2 运行时间数据表

	2D	20NewsGroup	Gisette	ECUESpam	URL-Reputation
Mahout	0.595s	9.827s	131.807s	96.611s	10100.209s
Liblinear	0.078s	0.793s	2.364s	13.161s	519.115s*
SLLR	1.761s	20.046s	130.451s	1028.185s	3248.473s
PSUBPLR-MR	120.186s	1360.854s	3687.941s	11478.706s	16098.260s
PGDPLR-SPARK	0.681s	10.517s	99.156s	924.020s	3615.780s
PSUBPLR-SPARK	1.325s	8.571s	89.094s	796.802s	2918.470s

6 个测试程序在 5 个数据集上分别取得的训练时间结果可以参见表 6.2。这些结果也都是进行交叉验证时记录的时间进行平均值计算后得到的结果。因而这些结果与表 6.1 中的精度结果是有良好对应关系的。为了方便对比，我们设计了柱状图 6.7 来更为直观的表示以上 30 个数据。我们以每个数据集为一组，6 个测试程在该数据集上所使用的训练时间依次排列。所以，这张图的横坐标为数据集，纵坐标为训练时间。

请注意，在表 6.2 中有标“*”号的数据，即 Liblinear 在 URL-Reputation 数据集上的训练时间。同前面 6.2 节所述，这里“*”号表示该结果仅是在 URL-Reputation 的$\frac{1}{3}$训练数据进行上执行而得出的结果。从理论上讲，假如节点内存足够大，由于 Liblinear 的训练时间大致与数据集大小成正比，可以预计 Liblinear 在 URL-Reputation 全数据集上的训练时间约为 1500s 左右，仍然会是所有测试程序中最快速的。

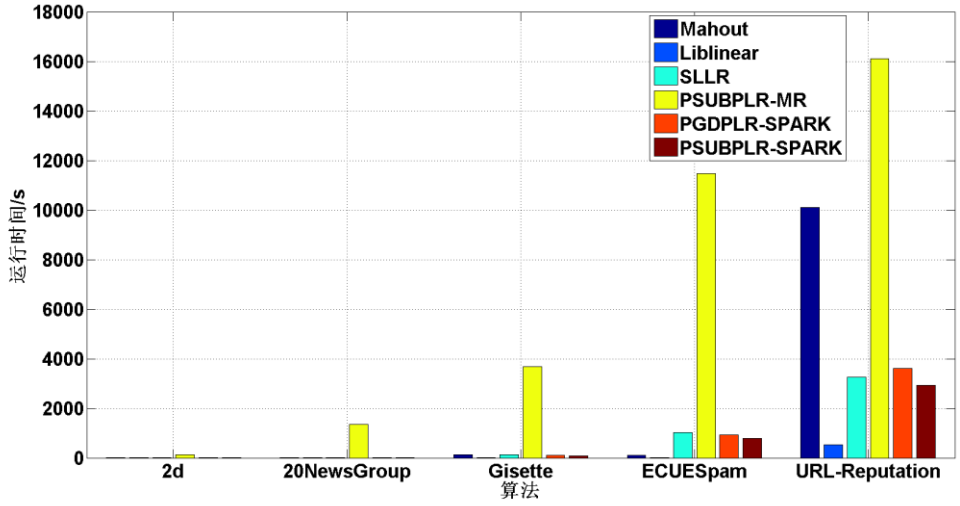

图 6.7 所有程序在所有数据集上的运行时间图

6.4 精度与训练时间综合分析

从上述显示所有测试程序在所有数据集上的训练时间表 3 和所有测试程序在所有数据集上的测试精度表 4。我们可以对比发现以下结论：

1. Liblinear 在所有的测试程序中总是学习效果最好的。这既体现在高测试精度，也体现在短训练时间上。这归功于 Liblinear 充分利用了节点内存，优异的底层代码优化，并且由于单机运行，不需要额外的通信开销。但是 Liblinear 的可扩展性是严重受到节点内存限制的。这使得 Liblinear 在事实上不适合对大规模数据集进行学习。

   考虑到 Liblinear 优异的性能，在数据集能够被完全在实验中被载入内存并进行计算的情况下，选择 Liblinear 依然是明智的。在此种情况下，高效算法的原则是，单机串行执行并充分利用内存。

2. Mahout 的测试精度相对来说是不高的。特别是针对正例和反例分布并不均匀的数据集，其学习效果较差，这是其劣势。然而，这是一个具有较强扩展性的串行算法的代表，并且其训练时间也在可以接受的范围内。在实际实验中，我们监测它的内存利用情况，可以看出其内存利用率一直保持在较低水平。这一点保证了在节点内存较低的情况下，Mahout 测试程序仍然能够执行大规模数据集的训练过程。而这也正是在线算法所带来的一个独特优势。

   因而，Mahout 在罗吉斯回归模型优化问题上的应用场景是，单机、内存有限，但数据集较大。
3. 在所有测试中，使用次线性方法的测试程序的最终学习精度都是非常接近最优值的，在实际应用背景下都可以接受。而其中使用次线性方法的并行测试程序相对于串行程序来说在精度上也只有很微小的下降。
4. Hadoop 系统对于解决针对罗吉斯回归模型优化算法的执行来说有巨大的缺陷。Hadoop 框架下的编程模式主要是针对非环形数据流的。这样设计的原因是可以更为高效的在程序运行过程中决定任务以及计算载荷的分配，并从可能的节点失败中恢复。但这样的非环形数据流正是与迭代算法相矛盾的。然而，测试程序是针对罗吉斯回归模型的近似优化算法，不可避免的含有迭代计算过程。这使得 PSUBPLR-MR 算法的训练时间甚至长于 Mahout 中的串行算法。

   我们进一步分析 Hadoop 系统上的程序运行细节，我们可以发现在当前实验环境下的每次 MapReduce 任务的启动时间就需要长达 20s。其中包括了系统配置，任务调度和参数传递时间。由于需要多次迭代，这样的额外开销严重影响了运行效率，这也是在 Hadoop 系统上运行的 PSUBPLR-MR 测试程序在小规模数据集上训练时间尤为长的原因。另一点是 PSUBPLR-MR 测试程序运行中，原始 Map 过程的运行时间占到了一次完整迭代时间的 66% 以上，成为主要瓶颈。这一点在 PSUBPLR-SPARK 测试程序的运行中的情况一致。
5. Spark 系统一定程度上是 Hadoop 的优化系统。它充分利用了所有节点上的内存和缓存，合理设计了 RDD，以及它所对应的内存和缓存分配机制。实验中，测试程序 PGDPLR-SPARK 和 PSUBPLR-SPARK 没有使用 HDFS，而是使用了本地文件系统，这要求在每个节点上都有一

份数据集的完整拷贝，但这对于节点硬盘存储不成问题。实验结果显示了 Spark 系统相对于 Hadoop 系统的显著优越性，并且显示 Spark 系统的并行效率可以超过 Mahout 的单机版本。

因而在大规模数据集上，并且可以利用多机并行加速的情况下，针对罗吉斯回归模型优化算法，我们推荐使用 Spark 系统。如果追求训练精度，可以采用 PGDPLR-SPARK 算法，如果更关心训练时间，可以采用 PSUBPLR-SPARK 算法牺牲一点学习精度换取更高的运行效率。由于当前 Spark 系统仍未完全成熟（比方说 Reduce 过程只有一个 Reducer），同时在本文实验中，所用集群资源也有拓展空间，我们期待 Spark 系统上的算法会取得更好的运算效果。

6.5 不同集群资源下的结果

在本文的实验环境下，集群满载是 6 个节点进行并行计算。前文所有实验结果均为埋在情况下取得。现在，我们关闭一部分节点，调整了整个集群资源，考察不同算法的运行时间变化。图 6.8 到图 6.12 显示了在不同参与计算的节点数下，测试程序每次迭代的运行时间（结果进行了平均）。因而，针对每个数据集，我们分别得到了一张柱状图。每张图的横坐标为参与计算的节点数，纵坐标为测试程序每次迭代时间。在每张图上，同一个测试程序的结果组成一组，利于对比。

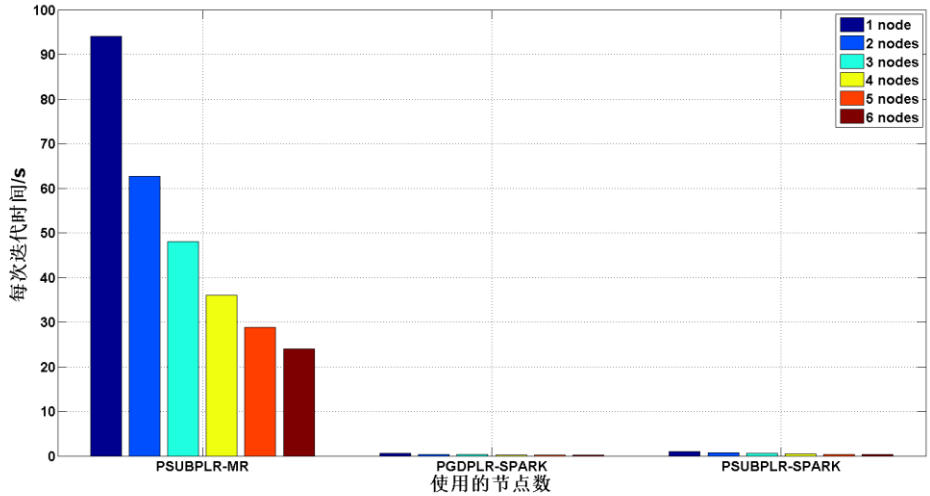

图 6.8 所有程序在仿真 2D 数据集上的迭代时间图

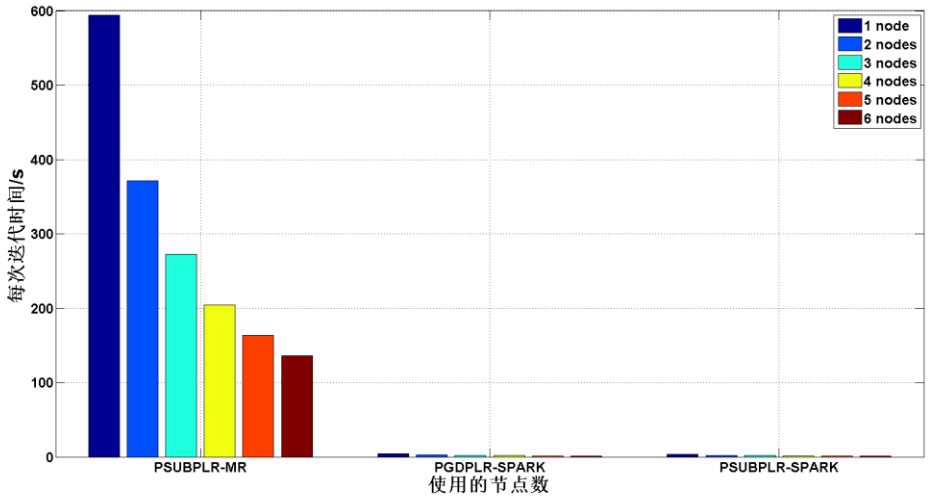

图 6.9 所有程序在 20NewsGroup 数据集上的迭代时间图

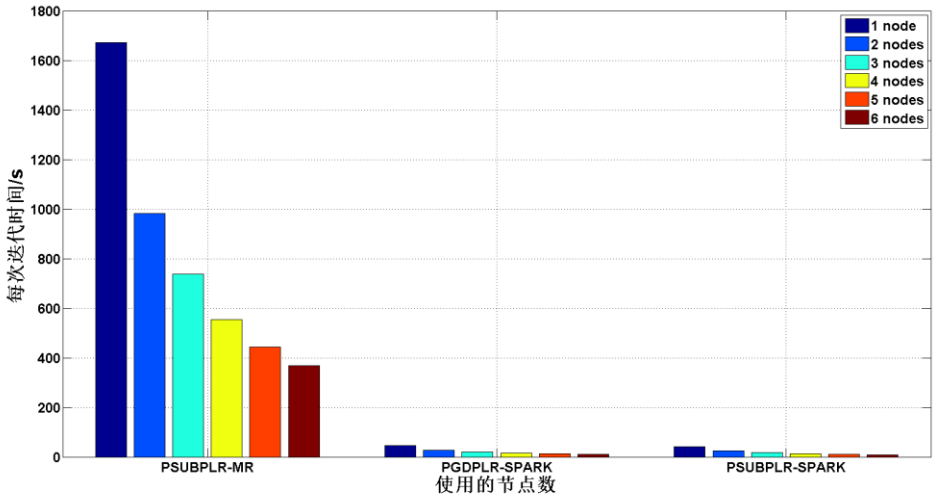

图 6.10 所有程序在 Gisette 数据集上的迭代时间图

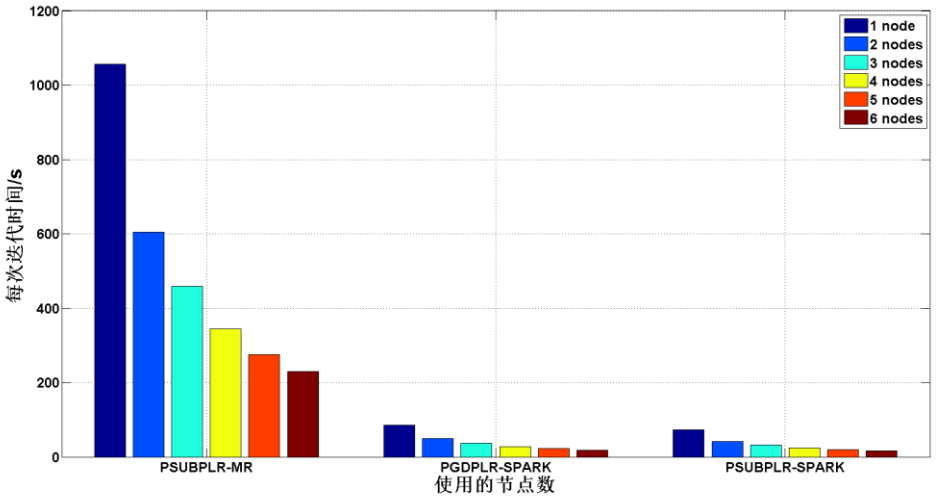

图 6.11 所有程序在 ECUESpam 数据集上的迭代时间图

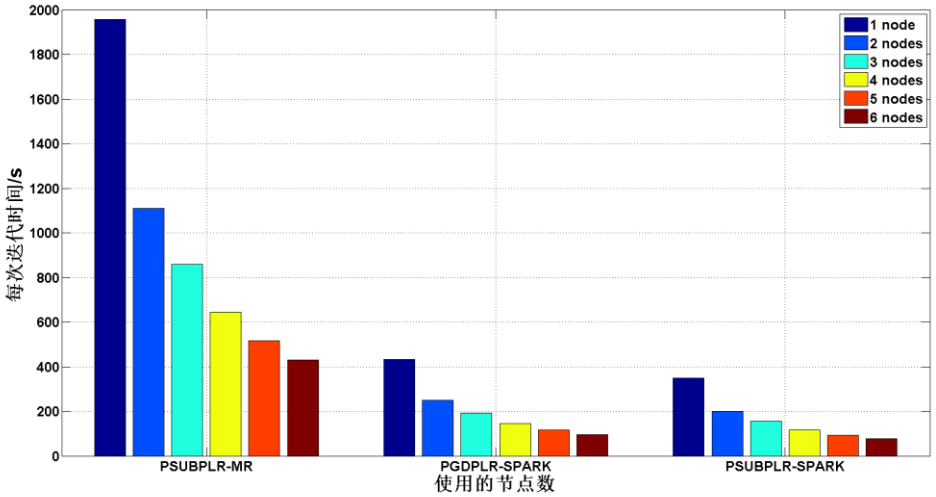

图 6.12 所有程序在 URL-Reputation 数据集上的迭代时间图

以上结果清晰地显示出当计算节点数增大时，并行程序的执行时间将会减少。仔细观察具体数值可以发现，这个单调趋势并非恒定，即程序每次迭代的执行时间并不与节点数成反比。特别的，理论上来说，当计算节点数增大到一定程度时，由于通信开销的增大，以及串行部分计算占迭代总时间比例的增大，总的单次迭代时间将不会有明显降低。这一节点极限值，也正是代表了各个不同并行算法的真正可扩展性。由于实验资源限制，这一点留待后续工作继续检验。

关于数据集，我们也可以从实验结果中发现一个有趣的事实。如果我们比较 ECUESpam 数据集和 GISETTE 数据集，从表 5.1 中可以看出前者有较高的特征维度，但数据更为稀疏，以至于非零元素更少。我们可以从图 6.10 和图 6.11 中比较发现，在 ECUESpam 数据集上，各个测试程序都具有更短的单次迭代运行时间。这是由于测试程序代码针对稀疏数据都做了优化，因而计算量是与数据集中非零元素个数直接相关的。但同时由于 ECUESpam 数据集的高纬度，从图 6.4 和图 6.5 中可以看出其在达到收敛前需要的迭代次数也更多，造成了总的训练时间比 GISETTE 数据集要多（这一点从表 6.2 中可以看出）。

以上分析说明，在对比不同稀疏数据集上相同算法的运行时间的时候，不能单独用数据集的大小，训练样本个数，特征维度来衡量。我们还需要对数据集的稀疏程度有所认识，并综合考虑以上所有因素。

6.6 对节点失败的鲁棒性

在分布式的并行计算中，对节点失败的鲁棒性既可以体现在并行框架层面，也可以体现在算法层面。我们在表 5 列出了各个测试程序在对节点失败的鲁棒性上的表现。

表 6.3 对节点失败鲁棒性比较表

	System Level	Algorithm Level
PSUBPLR-MR	√	√
PGDPLR-SPARK	√	×
PSUBPLR-SPARK	√	√

表 6.3 中的符号 √ 表示能提供相关鲁棒性，而 × 表示无法提供相关鲁棒性。测试程序 PSUBPLR-MR 使用的是 Hadoop 系统，主要通过数据拷贝机制保证鲁棒性。而测试程序 PGDPLR-SPARK 和 PSUBPLR-SPARK 使用 SPARK 系统，主要通过 SPARK 中 RDD 的线性操作来保证鲁棒性。而在算法层面上，由于测试程序 PSUBPLR-MR 和 PSUBPLR-SPARK 都采用次线性的优化方法，带有随机性，所

以一定数量的 MAP 失效，不会严重影响最终训练结果。而普通梯度下降法不具有这一特性，所以这一点也是次线性方法的额外优势。

我们继续深入定量研究节点失败对测试程序的影响。这里，我们把节点失效模拟为 MAP 失效，在实验中我们在一次迭代中随机设置一定比例的 MAP 失效。实验结果如图 6.13 所示。

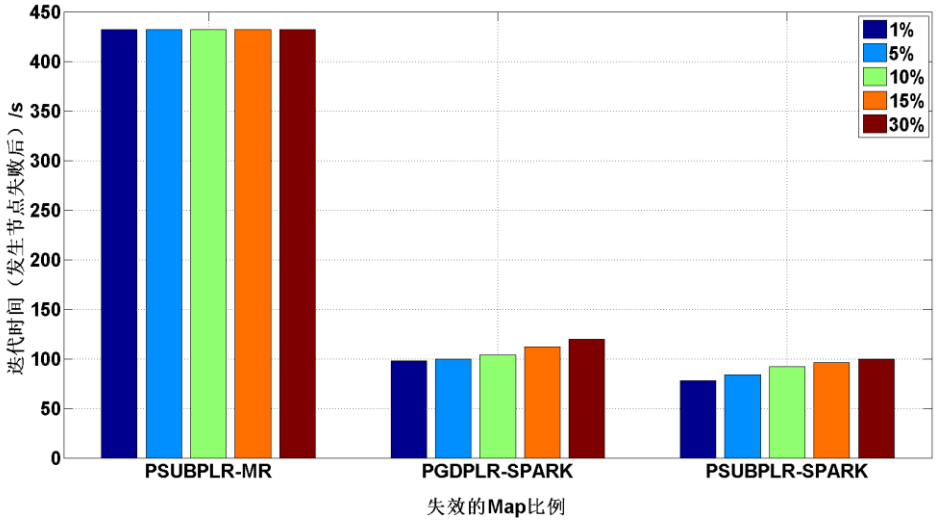

图 6.13 对节点失败鲁棒性测试结果图

在这一实验中，我们选择 URL-Reputation 数据集上的结果作为代表。图中横坐标为失效 MAP 的比例（我们设置了 1%，5%，10%，15%，30%这五档形成柱状图），纵坐标为发生 MAP 失效后的下一次迭代运行时间。正常迭代时间在图 6.13 上可以近似为发生 1%MAP 失效的情况。我们把 3 个并行算法的结果各自组成一组，易于比较。

从结果可以看出，测试程序 PSUBPLR-MR 可以完全不受 MAP 失效的影响，迭代时间没有增长。当然，这会带来迭代次数的增加或者是精度损失（在不增加迭代次数的情况下）。但实验中仅在一次迭代中出现 MAP 失效，这对最终结果的影响事实上是非常微小的。对于测试程序 PGDPLR-SPARK 和 PSUBPLR-SPARK 来说，MAP 失效后的那次迭代需要比平常花费更多的时间，因为其中增加了重建 RDD 所需的时间。但是这个增长并不明显，并且也不随着 MAP 失效比例的增大而有显著增长。并且，在这次迭代之后，RDD 又将恢复，此后的每次迭代时间将恢复到出现 MAP 失效以前的状态，因而 MAP 失效对程序总运行时间的影响微乎

其微，证明了对节点失败的较强鲁棒性。同时，从 Hadoop 和 Spark 上测试程序对 MAP 失效的不同表现，也可以体现出 Hadoop 与 Spark 系统提供对节点失败鲁棒性机制上的不同。

第7章 总结

7.1 论文主要工作的总结

7.1.1 工作成果

在本文中，我们分析了针对罗吉斯回归模型的优化求解问题，和它在大规模数据集上的并行加速框架。我们选择的 5 个数据集规模范围从 KB 递增到 GB。我们给出了普通的并行梯度下降方法、经典的在线随机梯度下降方法、提出了新的并行次线性优化方法。与此同时，我们也比较了 Mahout、Hadoop 和 Spark 三种各有特色的并行框架系统。从而引出了次线性方法的普通串行版本，以及在 Hadoop 系统和 Spark 系统下的两个并行版本。再加上以 Liblinear 作为测试基准程序，我们最终组成了 6 个测试程序。我们通过以上 6 个测试程序在 5 个数据集上进行的充分测试，验证了算法正确性，测试了学习精度，比较了训练时间。同时，本文也更近一步的分析了并行测试程序对在不同数量并行计算节点下的表现，以及对节点失败鲁棒性的表现。

7.1.2 工作贡献

本文研究了针对罗吉斯回归模型优化问题求解的机器学习算法在实际大数据应用背景下的表现。

充分考虑了以下维度上的各个特点：

1 数据集：包括大小、数据样本个数、特征维度、稀疏性

2 并行系统：包括设计思想、特色机制

3 算法：包括算法思想、框图和具体实现

4 综合测试：包括正确性、学习精度、运行效率、可扩展性、对节点失败的鲁棒性

由于在目前的学术界，机器学习算法影响巨大，但重点关注相关算法性能比较和测试的工作相对匮乏，并往往欠缺客观性。因而，本文的最大贡献在于，站在客观和综合的角度上，研究、测试、评价和比较了各个针对罗吉斯回归模型优化问题求解的机器学习算法的性能特点。同时，针对不同特点的数据集和不同的运行资源，本文给出了算法选择的建议。

本文也提出了针对罗吉斯回归模型的新的并行次线性算法。其在 SPARK 系统上的运行取得了优良的效果。

同时，本文将机器学习算法与系统综合考虑的研究思路也是本文的一大特点。

7.2 进一步的研究工作

首先，本文在综合测试部分的测试与分析还可以进一步增加，包括对算法稳定性、算法收敛性的分析。同时，也需要继续增加实验节点数，在更大规模的数据集和更大规模的集群上进行测试。

其次，本文提出的并行次线性方法也有继续研究优化的空间。

最后，本文较为客观而全面地针对罗吉斯回归模型优化问题求解进行了研究。可以在今后继续研究罗吉斯回归模型中的多分类问题。而在机器学习领域，还有很多其他模型有待研究。

插图索引

表格索引

参考文献

[1] Ion Androutsopoulos, John Koutsias, Konstantinos V Chandrinos, George Paliouras, and Constantine D Spyropoulos. An evaluation of naive bayesian anti-spam filtering. arXiv preprint cs/0006013, 2000.

[2] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta algorithm and applications. Manuscript, 2005. Preliminary draft of paper available online at http://www.cs.princeton.edu/ arora/pubs/MWsurvey.pdf, 2005.

[3] Dhruba Borthakur. Hdfs architecture guide. Hadoop Apache Project. http://hadoop.apache. org/common/docs/current/hdfs design.pdf, 2008.

[4] Edward Y Chang. Psvm: Parallelizing support vector machines on distributed computers. In Foundations of Large-Scale Multimedia Information Management and Retrieval, pages 213-230. Springer, 2011.

[5] K.L. Clarkson, E. Hazan, and D.P. Woodruff. Sublinear optimization for machine learning. In Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, pages 449-457. IEEE Computer Society, 2010.

[6] A. Cotter, S. Shalev-Shwartz, and N. Srebro. The kernelized stochastic batch perceptron. Arxiv preprint arXiv:1204.0566, 2012.

[7] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. Communications of the ACM, 51(1):107-113, 2008.

[8] S. J. Delany, P. Cunningham, A. Tsymbal, and L. Coyle. A case-based technique for tracking concept drift in spam filtering. Knowledge-Based Systems, 18(4-5):187-195, 2005.

[9] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-RuiWang, and Chih-Jen Lin. Liblinear: A library for large linear classification. The Journal of Machine Learning Research, 9:1871-1874, 2008.

[10] D. Garber and E. Hazan. Approximating semidefinite programs in sublinear time. In Advances in Neural Information Processing Systems, 2011.

[11] A. Genkin, D.D. Lewis, and D. Madigan. Large-scale bayesian logistic regression for text categorization. Technometrics, 49(3):291-304, 2007.

[12] William Gropp, Ewing L Lusk, and Anthony Skjellum. Using MPI-: Portable Parallel Programming with the Message Passing Interface, volume 1. MIT press, 1999.

[13] I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror. Result analysis of the nips 2003 feature selection challenge. Advances in Neural Information Processing Systems, 17:545-552, 2004.

[14] T. Hastie, R. Tishirani, and J. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer-Verlag, New York, 2001.

[15] E. Hazan and T. Koren. Optimal algorithms for ridge and lasso regression with partially observed attributes. Arxiv preprint arXiv:1108.4559, 2011.

[16] E. Hazan, T. Koren, and N. Srebro. Beating sgd: Learning svms in sublinear time. In Advances in Neural Information Processing Systems, 2011.

[17] Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. Graphchi: Large-scale graph computation on just a pc. In Proceedings of the 10th conference on Symposium on Operating Systems Design & Implementation, 2012.

[18] Quoc V Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S Corrado, Jeff Dean, and Andrew Y Ng. Building high-level features using large scale unsupervised learning. arXiv preprint arXiv:1112.6209, 2011.

[19] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Identifying suspicious urls: an application of large-scale online learning. In Proceedings of the 26th Annual International Conference on Machine Learning, pages 681-688. ACM, 2009.

[20] Apache Mahout. Scalable machine-learning and data-mining library. available at mahout. apache. org.

[21] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[22] Haoruo Peng, Zhengyu Wang, Edward Y Chang, Shuchang Zhou, and Zhihua Zhang. Sublinear algorithms for penalized logistic regression in massive datasets. In Machine Learning and Knowledge Discovery in Databases, pages 553-568. Springer, 2012.

[23] R. Tibshirani. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological), pages 267-288, 1996.

[24] S. Tsumoto. Mining diagnostic rules from clinical databases using rough sets and medical diagnostic model. Information sciences, 162(2):65-80, 2004.

[25] V. Vapnik. Statistical Learning Theory. John Wiley and Sons, New York, 1998.

[26] Yi Wang, Hongjie Bai, Matt Stanton, Wen-Yen Chen, and Edward Y Chang. Plda: Parallel latent dirichlet allocation for large-scale applications. In Algorithmic Aspects in Information and Management, pages 301-314. Springer, 2009.

[27] Tom White. Hadoop: The definitive guide. O'Reilly Media, Inc., 2012.

[28] L. Xiao. Dual averaging methods for regularized stochastic learning and online optimization. The Journal of Machine Learning Research, 11:2543-2596, 2010.

[29] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, pages 10-10, 2010.

[30] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In Proceedings of the twenty-first international conference on Machine learning, page 116. ACM, 2004.

致 谢

从毕设开题到结题，很多老师和同学无私的给予了我很多意见和帮助。

在此，我要衷心感谢赵颖副教授一直以来给予我的帮助和建议。

在此，也衷心感谢宾行言同学和胡东文同学给予我的帮助。我在与你们的讨论中获得了不少想法，受益匪浅。

声 明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____　　日　期：_____

附录 A 外文资料的调研阅读报告

Exploration of Parallel Systems and Algorithms in the Field of Machine Learning

In big data scenario, large datasets require us to develop machine learning algorithms towards a more efficient and more parallelized end. Researchers have already done much work to cater for the needs of massive datasets. Early work like PSVM (Parallel Support Vector Machines) [1] employed an approximation matrix decomposition method based on row selections to reduce the memory when running the algorithm. It can then increase the number of participated parallel computation nodes to several hundreds. Later, the work of PLDA (Parallel Latent Dirichlet Allocation) [2] further improve the computational efficiency of the LDA model through the use of sampling methods. The proposed parallel algorithm in Hadoop is robust, for it has fault tolerance of machine failures, taking the advantage of Hadoop. Recently, the work by Dean et al. [3] showed that the advantage of parallelization is fully taken in deep learning algorithms. It pushed the limits of parallel computational nodes to a hundred-million level, and in the meantime, achieved a best learning performance ever. Besides Hadoop, there are many other trials on parallelization in machine learning. GraphLab [4], a recent developed tool in CMU (Carnegie Mellon University) for large-scale machine learning, tried to tackle with efficiency and scalability problems of machine leaning algorithms when applied to graphs.

Three computing platforms we are working on each feature for their own advantage in implementing machine learning algorithms.

The Apache Hadoop [5], which uses simple programming models, is a framework that manages the distributed processing of massive datasets across multiple machines. It is an open-source software library. One of its advantages is that it supports data-intensive distributed applications. Another advantage is that it also supports the implementation of applications on machines of commodity hardware. Hadoop was developed based on MapReduce and Google File System (GFS). Hadoop kernel, MapReduce and Hadoop Distributed File System (HDFS), along with other related projects, including Apache Hive and Apache HBase build up the entire Hadoop platform. Hadoop is designed to scale up from single machine to hundreds of machines,

each providing local computation and local storage. The Hadoop framework offers reliability to applications in a transparent way.

Hadoop implements a computational paradigm called MapReduce [6]. In its concept, the application is separated into many small fractures of work. Each of them can be executed on any node in the cluster. The MapReduce framework design specifically applies to computational problems where the input is a set of key-value pairs. Particularly, a *map* function in MapReduce turns the input key-value pairs into some intermediate key-value pairs after user-defined operations; while a *reduce* function merges values for each intermediate key to produce output according to user's definition. Actually, a lot of problems can be formulated as MapReduce problems. This kind of design also makes itself quite easy to parallelization.

In addition, the distributed file system used in Hadoop offers very high aggregate bandwidth across the whole set of machines that are connected. Both map functions, reduce functions and the distributed file system are designed to handle node failures automatically. Instead of relying on hardware to deliver high-availability, the Hadoop system code is designed to identify and process failures at the application layer.

It is the goal of Apache Mahout [7] machine learning library to make scalable computations for machine learning algorithms. The outcome of this project shows that the scalability of existing classical machine learning algorithms is improved. The central parts in Mahout include algorithms for clustering, classification and collaborative filtering. It is also widely used in business. Mahout is also involved in the Apache Project.

Currently four best-known applications in Mahout include the following: Recommendation Engines, which take in users' behavior as input and predict items that users might have interests in as output. Clustering algorithms, which for example take in text documents as input and group them into topically related clusters as output. Classification algorithms, which learn from categorized documents and predict an unlabeled document of a correct category label. Frequent itemset mining algorithms, which learn from a set of item groups, and identify what individual items may appear together. Mahout currently supports many other machine learning algorithms including K-Means, fuzzy K-Means clustering, mean shift clustering, dirichlet process clustering,

Latent Dirichlet Allocation, singular value decomposition, parallel frequent pattern mining, random forest decision tree and etc.

Spark [8] is developed in the UC Berkeley AMPLab. It is an open source computing system built on cluster. The goal of Spark is to scale up data analytics. To make programs run faster, Spark offers primitives for in-memory cluster computing. A driver (application in Spark) can load data into memory or cache. Then the same driver can repeatedly query it. This in-memory strategy makes Spark much more efficient than Hadoop MapReduce, which is a disk-based system. Spark also supports programming in Scala, Java and Python. It offers clean APIs to all these programming languages.

At the beginning, Spark was developed for two applications iterative algorithms and interactive data mining. In both applications, it is critical to keep data in memory. As the outcome in these two applications shows, Spark can run a hundred times faster than Hadoop. As iterative algorithms are very common in the field of machine learning, the Spark system has great impact on the research community of machine learning. Spark is also widely used in business, for example, a fully Apache Hive-compatible data warehousing system called Shark. Taking full advantage of Spark, Shark can run a hundred times faster than Hive. Spark can also access data from HDFS.

Spark is built based on another cluster operating system called Mesos. The advantage of Mesos is that it can manage multiple parallel jobs to share a single cluster. It also makes it easy for users to launch tasks on a cluster through its API. Taking such advantage, Spark has the ability to run alongside existing cluster computing systems. It can also share data with them at the same time. Moreover, Mesos helps to reduce the programming workload in the development process of Spark.

The critical concept used in Spark is named resilient distributed dataset (RDD). It is a collection of objects divided across the cluster. An RDD can be reconstructed if a fault comes up in a machine in the cluster. Those RDDs are read-only and an application can load the RDD in memory through cache operations. The application can then reuse the RDD via a number of parallel operations. RDDs help Spark system to achieve robust fault tolerance via lineage: if a fault comes up in a machine in the cluster, the RDD still has enough critical messages so that it can be automatically reconstructed based on these explicit messages.

References

[1] Edward Y Chang. Psvm: Parallelizing support vector machines on distributed computers. In Foundations of Large-Scale Multimedia Information Management and Retrieval, pages 213-230. Springer, 2011.

[2] Yi Wang, Hongjie Bai, Matt Stanton, Wen-Yen Chen, and Edward Y Chang. Plda: Parallel latent dirichlet allocation for large-scale applications. In Algorithmic Aspects in Information and Management, pages 301-314. Springer, 2009.

[3] Quoc V Le, Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S Corrado, Jeff Dean, and Andrew Y Ng. Building high-level features using large scale unsupervised learning. arXiv preprint arXiv:1112.6209, 2011.

[4] Aapo Kyrola, Guy Blelloch, and Carlos Guestrin. Graphchi: Large-scale graph computation on just a pc. In Proceedings of the 10th conference on Symposium on Operating Systems Design & Implementation, 2012.

[5] Tom White. Hadoop: The definitive guide. O'Reilly Media, Inc., 2012.

[6] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. Communications of the ACM, 51(1):107-113, 2008.

[7] Apache Mahout. Scalable machine-learning and data-mining library. available at mahout. apache. org.

[8] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, pages 10-10, 2010.

综合论文训练记录表

<table>
<tr><td>学生姓名</td><td></td><td>学号</td><td></td><td>班级</td><td></td></tr>
<tr><td>论文题目</td><td colspan="5"></td></tr>
<tr><td>主要内容以及进度安排</td><td colspan="5">指导教师签字：_____
考核组组长签字：_____
年　　月　　日</td></tr>
<tr><td>中期考核意见</td><td colspan="5">考核组组长签字：_____
年　　月　　日</td></tr>
</table>

指导教师评语	指导教师签字：_____ 年　　月　　日
评阅教师评语	评阅教师签字：_____ 年　　月　　日
答辩小组评语	答辩小组组长签字：_____ 年　　月　　日

总成绩：_____

教学负责人签字：_____

年　　月　　日