基于Spark的并行SVM算法研究

刘泽燊 潘志松

(解放军理工大学指挥信息系统学院 南京 210007)

摘 要 随着数据规模的不断增加,支持向量机(SVM)的并行化设计成为数据挖掘领域的一个研究热点。针对SVM算法训练大规模数据时存在寻优速度慢、内存占用大等问题,提出了一种基于Spark平台的并行支持向量机算法(SP-SVM)。该方法通过调整层叠支持向量机(Cascade SVM)的合并策略和训练结构,并利用Spark分布式计算框架实现;其次,进一步分析并行操作算子的性能,优化算法并行化实现方案,有效克服了层叠模型训练效率低的缺点。实验结果表明,新的并行训练方法在损失较小精度的前提下,在一定程度上减少了训练时间,能够很好地提高模型的学习效率。

关键词 并行计算,支持向量机,大规模数据,层叠模型,Spark

中图法分类号 TP18 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.5.044

Research on Parallel SVM Algorithm Based on Spark

LIU Ze-shen PAN Zhi-song

(Institute of Command Information System, PLA University of Science and Technology, Nanjing 210007, China)

Abstract With the constant increasing of data scale, the parallel design of support vector machine(SVM) has become a hot research topic in data mining field. In view of the problems in model training including slow optimization and large memory, we proposed a new parallel SVM algorithm(SP-SVM) based on Spark. First of all, this paper implemented algorithm using Spark parallel computing framework. Secondly, this paper analyzed the performance of the parallel operator and optimized the algorithm in parallel design scheme, solving the problem of low efficiency that cascade training model encounters. Experimental results show that the new parallel training method can save more training time and greatly improve the efficiency in the case of a small precision loss.

Keywords Parallel computing, Support vector machine, Large scale data, Cascade model, Spark

1 引言

支持向量机[1]是一种具有完整理论推导和优异实践性能的机器学习算法,被广泛应用于文本分类、人脸识别和图像检测等领域。近年来,许多SVM的软件模型得到了很好的发展,Libsvm[2]由于实际中应用效果良好而深受学者喜爱。但是当训练样本不断变大时,SVM算法训练的内存和时间消耗急剧增加,所以单机SVM算法不能有效处理大规模数据集。

针对SVM算法处理大数据集的不足,近年来很多数据挖掘相关领域的专家对SVM的并行化设计提出了不同的策略,大体上分为两种:(1)并行化求解SVM算法的优化方法;(2)采用分而治之的策略进行数据分割,同时依靠高性能机器或者集群并行训练SVM算法。Dong等人[3]提出了一种用块对角矩阵近似kernel矩阵的并行优化步骤,将原始问题分解成大量的子问题。Lin等人[4]利用置信域牛顿法并行优化线性SVM算法,极大地提高了求解速度。而张巍等人[5]则借助GPU强大的计算能力并行化SVM算法中的矩阵运算。Grafs等人[6]提出了层叠的支持向量机算法,将数据分割进行级联训练。Sun等人[7]基于MapReduce[8]并行框架实现了层叠支持向量机。张鹏翔等人[9]则利用MapReduce结合层叠和分组两种训练模型实现了层叠分组并行SVM算法。通过这些研究,SVM算法的并行化取得了一定的进展,但仍具有改进空间,比如并行优化后的SMO算法问题处理规模仍然有限;而Cascade SVM算法的后面层级消耗了大量的时间,但对剔除非支持向量的贡献度不是很大;MapReduce计算模型在迭代处理时需要多次访问磁盘,影响了训练速度。Spark[10,11]作为一种基于内存计算的分布式集群框架,具有可扩展、高可靠性和负载均衡等优点,是一种高效处理大数据的并行计算模型。

本文对Cascade SVM算法和Spark计算平台进行了深入分析,以数据划分为基础,实现了一种高性能的并行SVM算法。具体工作主要有:(1)改进Cascade SVM的训练结构和合并策略,并利用Spark并行框架具体实现了算法。(2)调整部分并行操作算子和参数,优化了新算法性能。实验对比

到稿日期:2015-10-19 返修日期:2016-01-03 本文受国家自然科学基金项目(61473149)资助。

刘泽燊(1991—),男,硕士生,主要研究方向为模式识别、并行计算;潘志松(1973—),男,教授,博士生导师,主要研究方向为模式识别、网络安全。

表明，该算法能够加快模型训练的收敛速度，提高算法分类效率，适合处理大规模数据集的应用。

2 Apache Spark 技术

Spark 是 UC Berkeley 研发的通用大数据计算平台，采用 Scala 函数式编程语言实现，同时提供 Java、Python 等语言的 API 用来开发应用程序。Spark 完全兼容 Hadoop 原有的生态系统，利用 HDFS 作为分布式存储系统，同时克服了 MapReduce在迭代运算和交互式数据分析中的不足。

弹性分布式数据集(RDD)[12]是 Spark 的数据存储核心，在迭代计算时可以高效地共享数据，目标是为基于工作集的应用提供一种抽象。RDD 本质上是一个元数据结构，提供了一种高度受限的内存模型，记录着只读的、分区记录的集合，存储着数据分区及其逻辑结构映射关系；在 Spark 编程模型中 RDD 被表示为对象，相应的 API 为 RDD 提供转换(Transformations)和动作(Actions)两种算子，其中 Transformations 算子是懒操作，执行后创建新的 RDD，从而 RDD 之间形成相互依赖关系，同时 RDD 的依赖分为窄依赖和宽依赖，其中窄依赖是指子 RDD 的分区只依赖父 RDD 的某个分区，而宽依赖则指子 RDD 的每个分区依赖父 RDD 的多个分区；Action 算子则真正触发程序的执行，结果是向 driver 程序返回普通值类型或者向外部存储系统导出数据。再者，RDD 实现了两种容错方式：记录转换信息和数据检查点，由于数据检查点操作代价很高，需要大量的网络带宽，因此对于一般的 RDD 操作默认采用记录转换信息进行容错，以便恢复丢失的分区，当遇到宽依赖或者转换太多时可以采用检查点操作进行容错。

3 相关概念和原理介绍

3.1 支持向量机

SVM 算法在线性可分的情况下的学习策略是使边缘超平面的距离最大化，寻找具有最大边缘距离(Maximum Marginal)的分类超平面，在一定程度上克服了传统机器学习算法中过拟合、维数灾难和局部最小值等不足，具有很强的学习和泛化能力等。对于给定的训练集，SVM 算法训练的本质是求解一个优化问题，如式(1)所示：

$$\min \frac{1}{2}\|w\|^2 \quad \text{s.t. } y_i(w^{\mathrm{T}}x_i+b)\geqslant 1, i=1,\cdots,n \tag{1}$$

这是一个凸二次规划问题，可以通过已有的 Quadratic Programming 优化包进行求解；同时也可以引入拉格朗日乘子并且对参数求偏导，进而求出与原问题对应的对偶问题，具体公式如下：

$$\max_{a}\sum_{i=1}^{n}a_i-\frac{1}{2}\sum_{i,j=1}^{n}a_ia_jy_iy_jx_ix_j \quad \text{s.t. } a_i\geqslant 0, i=1,\cdots,n; \sum_{i=1}^{n}a_iy_i=0 \tag{2}$$

这样转换有两个目的：1)对偶问题比原问题更容易求解；2)为线性不可分的情况下引入核函数做好铺垫。

最后通过 SMO 等优方法求解对偶问题，从而得到原问题的解，当需要确定测试样本类别时，可通过式(3)所示的判别函数进行分类。由于非边界样本对应的参数 a_i 都是 0，因此只有支持向量样本对分类有贡献，进行测试时只需计算测试样本与支持向量的内积。

$$f(x)=w^{\mathrm{T}}x+b=\sum a_iy_i\langle x,x_i\rangle+b \tag{3}$$

对于线性不可分的情形，可以加入惩罚函数，在包容噪声点的同时也避免了过拟合的情况；而对于非线性问题，由于对偶函数和决策函数都有样本点的内积计算，因此引进核函数将样本从低维空间映射到高维空间，核函数很巧妙地避免了映射后的维数灾难问题，用核函数计算代替映射后高维空间的内积运算。其优点是直接在原始的低维空间进行内积计算，而实质的分类效果却表现在高维空间上，从而达到线性可分的目的。常用的核函数主要有多项式核、高斯核和线性核，一般的核函数表达式为：

$$K(x,z)=\langle\sigma(x)\cdot\sigma(z)\rangle \tag{4}$$

3.2 层叠支持向量机

Cascade SVM 是一种分组级联的训练模型，基本思想是通过切分数据和剔除非支持向量达到加速训练的目的。层叠训练过程如图 1 所示，在第一层开始训练时将大数据集分割成独立的训练子集，然后每个小数据集单独在处理器上进行 SVM 训练以剔除大量的非边界样本，剩下局部支持向量两两组合形成层叠的效果作为下一层的输入，直到最后合并为一个数据集训练出全局的支持向量，此时判断全局支持向量是否满足训练的精度要求，如果满足则训练结束，否则将最后一层的全局支持向量反馈到第一层与原来的数据集一起分割进行迭代训练，直至最后收敛输出训练模型。

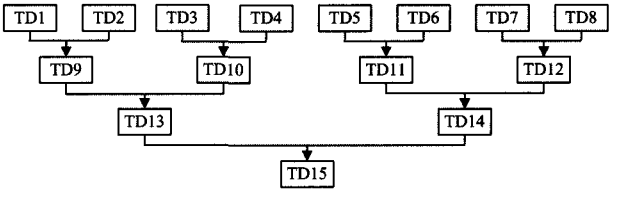

图 1 层叠支持向量机的训练流程

Cascade SVM 通过这种特定的二分级联训练结构，在训练大规模数据集时能够节省大量的存储空间和计算时间，是一种有效的 SVM 并行学习大数据集方法。但是由文献[13]关于层叠 SVM 的性能分析可知，在分布式集群环境下层叠训练结构有两个不足：(1)训练过程中集群资源利用率不高，出现大量空闲节点；(2)后面层级消耗大量的时间，但过滤的非支持向量较少。

4 基于 Spark 的并行 SVM 算法的设计与实现

4.1 改进 Cascade SVM 算法的思路

本文以 Cascade SVM 算法为基础，从数据分块的思想出发，针对 SVM 算法训练大规模数据存在的问题，提出了一种改进的 Cascade SVM 训练算法。针对上述层叠训练模型后面层级资源利用率低以及耗时长等缺点，改进 Cascade SVM 算法在训练层次结构和合并策略上做出了相应的调整，即训练结构采取固定的 3 层串联训练结构，层与层间的合并方式则由两两合并优化变为随机合并。新型 SVM 算法的并行化思想继承了 Cascade SVM 算法的主要架构，把大规模数据的优化问题变成了小的、独立的优化问题，进而通过局部寻优达到全局最优。第一次训练采用层叠过滤方法，把大量的非边界样本剔除掉，此时得到的结果基本上是支持向量样本。如果此时把所有子集进行全局训练，则训练样本数量依然过多，

所以需要整合所有局部支持向量，再随机均分为 4 份，进行 SVM 训练，得到 4 个支持向量数据集。最后把这 4 个局部模型整合成一个数据集进行全局训练。算法的详细流程如图 2 所示，模型分为 3 个层次，TD 表示训练的数据块，SV 则为每个数据集训练后的局部支持向量。

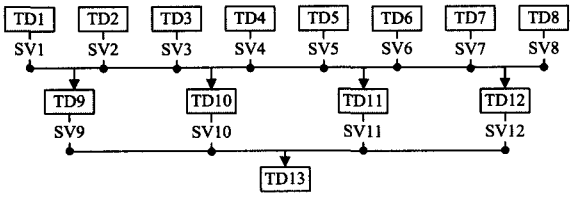

图 2　改进的层叠支持向量机的训练流程

第一层为分组训练层。基本上与标准 Cascade SVM 算法类似，采取剔除非支持向量的策略。当训练数据规模较大时，将数据集随机划分为 8 个数据块（TD1－TD8），切分处理后的数据分布与原数据集相似，从而避免了最后得到的全局支持向量与单机 SVM 算法训练结果差别较大的问题。然后单独对每个数据子集进行 SVM 训练，第一次迭代剔除许多非边界样本后得到的是局部支持向量（SV1－SV8）。

第二层称为混合训练层。考虑到数据子集两两合并后存在过滤非边界样本不足的缺点，将上层得到的局部支持向量先整合到一起，然后再随机分成 4 个新的训练集（TD9－TD12），并做相应的并行 SVM 训练。

第三层则是全局训练层。将第二层的局部边界样本（SV9－SV12）整合成一个数据块（TD13）进行全局支持向量的训练，得到最终的支持向量模型，并判断是否满足停机条件。如果不满足，则将结果送回反馈，重复训练，直至满足收敛条件。

4.2　并行 SVM 基于 Spark 框架的实现

根据上述对改进 Cascade SVM 训练结构的描述，同时结合 Spark 平台的并行编程模型，设计了基于 Spark 的 SVM 算法并行化实现方案。算法的详细实现步骤如图 3 所示。在模型训练开始之前，需将数据集上传到 HDFS 分布式文件存储系统。Spark 集群的任务调度系统则会为分割后的每个数据子集在 Executor 里创建新任务，并利用资源调度器给相应的任务分配计算资源。然后算法在 Spark 集群上进行分层的并行 SVM 训练并将局部支持向量进行合并操作，直至最后训练完成，输出全局最优模型到 HDFS 文件系统。

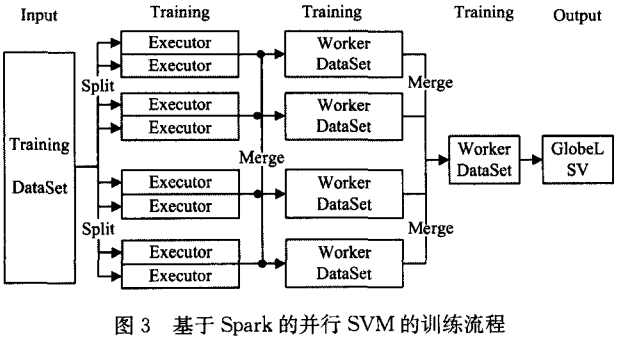

图 3　基于 Spark 的并行 SVM 的训练流程

改进算法在 Spark 分布式框架上的并行实现首先利用 RDD 编程模型提供的 textFile 函数从 HDFS 文件系统读取训练集并自动转换成 RDD 数据模型，同时根据指定的 partition 参数将大规模训练数据随机切分成大小适中的独立数据分区。接着 mapPartitions 函数将每个数据块里的训练数据组织成 SVM 算法的输入格式，随后再并行训练 SVM 算法；把大量非边界样本点剔除掉后保留了部分支持向量 SV_i，从而减少了后面层级训练的数据样例，加快了算法的整体训练速度。在第一次训练完成后，repartition 函数通过洗牌操作（shuffle）整合训练结果，并随机划分为小数据块 TD_i 作为本次训练的输入。数据块划分的具体数量可以根据集群规模而定，大致与集群拥有的节点数相等，这是因为每个集群节点都有一个合并后的大数据块，可以降低下一层合并的通信消耗并且提高算法的训练效率；同样，mapPartitions 函数在每个数据块上单独训练 SVM 算法，过滤部分非支持向量后得到训练结果 SV_i。最后连接上层输出的局部边界结点，汇总得到一个大的数据集 GTD_i，在 GTD_i 上训练 SVM 算法，得到全局支持向量模型 GModel。基于 Spark 的并行 SVM 算法的详细描述如算法 1 所示。

算法 1　基于 Spark 的并行 SVM 算法

input：TrainingDatasets，FilePath，Partition

output：SV_i，GModel

1. val sc=newSparkcontext(conf)
2. val DataRDD←sc. textFile(FilePath，Partition)
3. Do i=1，…，L
4. val SV_i←DataRDD. mapPartitions(Train_SVM)
5. val TD_i←SV_i. repartition()
6. val SV_i←TD_i. mapPartitions(Train_SVM)
7. val GTD_i←SV_i. repartition()
8. val GModel←GTD_i. mapPartitions(Train_SVM)
9. Until 满足停机条件 Return GModel

4.3　算法调优

coalesce 和 repartition 是 RDD 上两个重新划分数据分区的并行操作算子。两者的区别是 repartition 算子在重新划分数据时使用了 shuffle 操作，可以减少或者增加数据分区的数量；而 coalesce 则由于没有 shuffle 步骤，只能减少原 RDD 的数据分片。改进算法在把所有局部支持向量合并为一个数据块时，coalesce 和 repartition 都能够得到算法要求的结果，但由于合并为一个数据块是一个非常极端的合并操作，如果使用 coalesce 算子，则可能导致只在一个节点上进行所有的合并计算，将会严重影响合并的效率。为了避免这种情况的发生，在具体实现时选择使用 repartition 算子，这样虽然在合并时多了一个 shuffle 步骤，但是所有的数据将会并行执行合并操作，极大地缩短了合并的时间，提高了算法整体的训练效率。

5　实验过程与结果分析

5.1　实验环境与数据集

本文采用的数据挖掘工具是 LibSVM3. 17。利用 5 台 Dell 服务器构建 Spark 集群，包括 1 台 Master 节点和 4 台 Slave 节点，集群的任务调度模式为 standalone。硬件配置：服务器拥有 8GB 内存以及四核处理器。软件配置：集群搭建使用 spark-1. 2. 0-bin-hadoop1 和 Hadoop-1. 2. 1 稳定版，Java 选用 JDK1. 7. 0 _ 71（Linux 版），操作系统选择 ubuntu12. 04 Server 版。

训练集采用 a9a、ijcnn1、covtype. binary 这 3 个标准数据集[14]。其中 a9a 数据集用来预测一个成人的年收入，拥有

123 个属性、32562 个训练样本和 16281 个测试样例；ijcnn1 则包括 49990 个 22 维的训练样本和 91701 个测试样本；covtype. binary 数据集原始为森林覆盖率数据，有 581012 个样本，每个样本具有 54 维特征。

5.2 实验结果及性能分析

5.2.1 coalesce 和 repartition 性能对比

首先对 covtype. binary 数据集进行抽样，选择出 7 个不同大小的训练数据，然后在相同条件下使用 coalesce 和 repartition 算子实现的算法训练不同规模数据集，相关结果对比展示在图 4 中。从中可以看出，repartition 算子编写的算法对每个数据集的训练时间都明显少于 coalesce 算子，且随着样本数量的增加，两者的运行时间都在增大，但是差距也越来越大，这是因为 repartition 算子中由于 shuffle 操作能够对剧烈合并产生并行计算的效果，算法整体效率更高，极大地减少了合并时间。由于 repartition 算子性能更优，因此最后选择采用 repartition 算子把所有的局部支持向量合并到一个数据块。

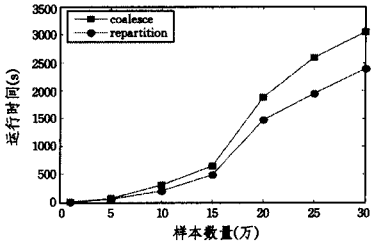

图 4 算子调优过程

5.2.2 训练结果与数据分区的关系

为了评估算法在不同分区下训练时间和准确率的变化情况，选用了 a9a 数据集进行实验，每个分区在数据集上的训练时间取 3 次训练的平均时间，准确率则是由训练出的模型对相应测试集进行预测。最终结果如图 5 所示。

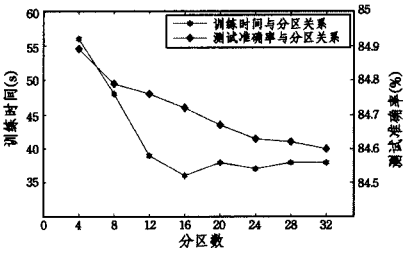

图 5 分区数对测试精度和训练时间的影响

从图 5 中可看到，随着分区数目的增加，算法的训练时间先是急剧减少而后缓慢增加。这是由于分区数的变大导致算法的并行程度提高，虽然降低了分区的计算时间，但同时也增加了分区数据传输的通信开销。开始分区数量的增加会显著降低每个分区的计算时间，从而减少程序整体的运行时间；但是分区数达到一定的量后，单个分区的计算时间趋于平稳，而网络开销则随着分区的增大而迅速增加，从而训练时间将会呈现出小幅度增加的趋势。此外，算法的准确率则随着分区数的增加一直下降，这是由于数据块划分过多，每个数据块与原数据集的分布差别较大，从而训练出来的全局支持向量与单机 SVM 训练的结果差别较大，因此测试准确率一直下降，所以对于分区数的设定，需要考虑集群的计算能力和数据集的大小，追求计算速度的同时须兼顾算法的准确率。

5.2.3 算法运行时间对比

同样从 covtype. binary 数据集中抽样出 7 个训练集，并利用单机 SVM 算法、Cascade SVM 算法和 SP-SVM 算法训练选出来的数据集，图 6 记录对比了 3 种算法在不同规模训练数据集上的运行时间。当数据集较小时，单机 SVM 要比在 Spark 集群环境下运行 Cascade SVM 和 SP-SVM 算法训练时间少，这是由于集群计算需要初始化相关任务，包括新建作业、资源的分配和调度，而且不同任务之间也需要通信，这都需要消耗部分时间。随着样本规模增大，并行算法在集群上运行的优势也就越明显，产生这种结果是因为在大规模数据下训练支持向量的任务初始化和通信时间占整个作业运行时间的比重较小，同时并行 SVM 算法会将样本分割为固定大小的数据块在不同处理器上并行计算，从而节省了大量的训练时间。Spark 集群上不同的并行 SVM 算法在大样本集上的运行时间差别也很大，SP-SVM 算法在效率要明显高于标准 Cascade SVM 算法，这是由于 SP-SVM 算法在训练结构和合并策略上要优于标准的 Cascade SVM 算法，能够减少训练时间并提高分类效率。

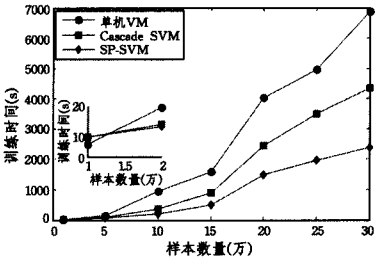

图 6 算法的运行时间对比

5.2.4 算法准确率对比

为了验证改进后的 SP-SVM 算法的有效性，本次实验采用 a9a、ijcnn1 和 covtype. binary 数据集为训练数据，然后分别运行不同类型的 SVM 算法训练支持向量，并对预测精度进行分析对比。图 7 评价了单机 SVM、Cascade SVM 和 SP-SVM 算法在不同数据集上的预测准确率，从中可看出 3 种算法的精度会随着数据集的不同而产生变化。SP-SVM 算法的准确率在 3 个数据集上明显高于 Cascade SVM 的准确率，而与单机 SVM 算法相比，在 a9a 和 covtype. binary 数据集的精度虽略有下降，但在 ijcnn1 数据集上还有所提高，同时两个算法的误差不超过 0.01。在所有数据集上 Cascade SVM 算法的准确率都是最低的，而单机 SVM 的大部分测试精度比使用并行策略的 SVM 算法都要好。由此可以得出，SP-SVM 算法的精度比 Cascade SVM 算法的准确率高，同时相比单机 SVM 的准确率在部分训练集上有微小的损失，但是相差不大。总体而言，SP-SVM 算法精度高，分类效果好，能够有效地学习大规模数据。

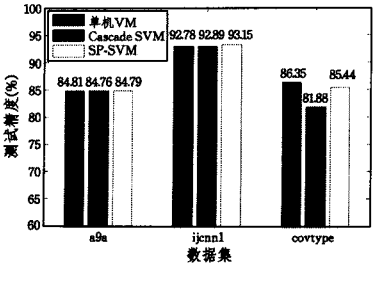

图 7 算法的测试精度对比

结束语 支持向量机是计算密集型学习算法，不论 SVM 还是 Cascade SVM 算法，它们对大数据集的模型训练代价都较高，时间长。通过对 Cascade SVM 的并行策略进行深入的分析，结合 Spark 计算平台，实现了一种基于 Spark 的并行 SVM 训练算法，这个模型以数据划分和级联训练为基础，对 Cascade SVM 的训练结构进行了相应的改进，进而解决了层叠模型训练效率低的缺点。通过大量实验分析表明，该算法在准确率没有明显降低的前提下，能够提高分类速度，是 SVM 算法求解大规模数据集的一种有效解决方案。但是该算法还存在很多问题需要研究，例如当数据集的支持向量太多时算法训练效率不高，数据分块过多时准确率会有所下降，并行设计的合理性没有严谨的数学证明和理论推导。

在接下来的学习中，将对算法的理论支持和训练结构的优化做更加深入的研究，同时也会着重探讨 Spark 平台的参数调优，以期获得更好的训练效果，满足广大学者对采用支持向量机训练大规模数据集的需求。

参 考 文 献

[1] Vapnik V N. The Nature of Statistical Learning Theory[M]. Springer New York, 1995: 988-999

[2] Chang C C, Lin C J. LIBSVM: a Library for Support Vector Machines[J]. ACM Transactions on Intelligent Systems & Technology, 2006, 2(3): 389-396

[3] Dong J X, Krzyzak A, Suen C Y. Fast SVM training algorithm with decomposition on very large data sets[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2005, 27(4): 603-618

[4] Lin C Y, Tsai C H, Lee C P, et al. Large-scale logistic regression and linear support vector machines using spark[C]// 2014 IEEE International Conference on Big Data. IEEE, 2014: 519-528

[5] Zhang Wei, Zhang Gong-xuan, Wang Yong-li, et al. Research on parallel SVM algorithm based on CUDA[J]. Computer Science, 2013, 40(4): 69-72(in Chinese)
张巍，张功萱，王永利，等. 基于 CUDA 的 SVM 算法并行化研究[J]. 计算机科学，2013，40(4)：69-72

[6] Graf H P, Cosatto E, Bottou L, et al. Parallel Support Vector Machines: The Cascade SVM[C]// Advances in Neural Information Processing Systems(NIPS). 2004: 521-528

[7] Sun Zhan-quan, Fox G. Study on Parallel SVM Based on MapReduce[C]// The 2012 International Conference on Parallel and Distributed Processing Techniques and Applications. Las Vegas NV USA, 2012

[8] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters[J]. Proceedings of Operating Systems Design and Implementation(OSDI), 2004, 51(1): 107-113

[9] Zhang Peng-xiang, Liu Li-min, Ma Zhi-qiang. Research of parallel SVM algorithm based on MapReduce[J]. Computer Applications and Software, 2015, 32(3): 172-176(in Chinese)
张鹏翔，刘利民，马志强. 基于 MapReduce 的层叠分组并行 SVM 算法研究[J]. 计算机应用与软件，2015，32(3)：172-176

[10] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets[C]// Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing USENIX Association. 2010: 10

[11] http://spark.apache.org

[12] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]// Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. 2012: 141-146

[13] Guo Xin-xin. SVM optimization algorithm based on Distributed Computing[D]. Xi'an: Xi'an Electronic and Science University, 2014(in Chinese)
郭欣欣. 基于分布式计算的 SVM 算法优化[D]. 西安：西安电子科技大学，2014

[14] http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets

(上接第 233 页)

[6] Guo S M, Chen L C, Tsai J S H. A boundary method for outlier detection based on support v-ector domain description[J]. Pattern Recognition, 2009, 42(1): 77-83

[7] Zhang Y, Chi Z, Li K. Fuzzy multi-class classifier based on support vector data description and improved PCM[J]. Expert Systems with Applications, 2009, 36(5): 8714-8718

[8] Hu Chen-long, Zhou Bo, Hu Jing-lu. Fast support vector data description training using edge detection on large datasets[C]// 2014 International Joint Conference on Neural Nerworks(IJCNN). IEEE, 2014: 2076-2182

[9] Wu Ding-hai, Zhang Pei-lin, Wang Huai-guang, et al. One-class classification method based on multi-kernel support vector data description[J]. Computer Engineering, 2013(5): 165-168(in Chinese)
吴定海，张培林，王怀光，等. 基于多核支持向量数据描述的单类分类方法[J]. 计算机工程，2013(5)：165-168

[10] Gao Zhi-hua, Ben Ke-rong. Study on acoustic sources identification based on multi-svdd[J]. Computer Science, 2012, 39(11): 233-236(in Chinese)
高志华，贲可荣. 基于多分类支持向量数据描述的噪声源识别研究[J]. 计算机科学，2012，39(11)：233-236

[11] Tax D M J, Duin R P W. Support Vector Data Description[J]. Machine Learning, 2004, 54(1): 45-66

[12] Chai J, Liu H, Bao Z. Generalized re-weighting local sampling mean discriminant analysis[J]. Pattern Recognition, 2010, 43(10): 3422-3432

[13] Tsang I W, Kocsor A, Kwok J T. Efficient kernel feature extraction for massive data sets[C]// Kdd Proceedings of ACM Sigkdd International Conference on Knowledge Discovery & Data. 2006