云计算平台的海量数据知识提取框架①

邹 裕

(东莞理工学院 计算机学院, 东莞 523808)

摘 要: 针对从海量数据中分析与提取知识计算时间高的问题, 提出一种基于 Hadoop 的知识提取算法. 本文结合 Hadoop 的并行处理能力与分布式存储特点, 设计了一种知识提取框架, 可兼容不同的原型约简方法. 基于 MapReduce 编程方法将约简方法并行化处理, 并且设计了分类准确率高、计算速度快的原型约简组合规则. 最终基于真实 UCI 大数据集进行实验, 本框架将最近邻分类器的分类时间提高两个数量级.

关键词: 海量数据; 知识提取; 原型约简; 云计算; 并行计算; 数据聚类

Massive Data Knowledge Extraction Framework Based on Cloud Computing Platform

ZOU Yu

(College of Computer Science and Technology, Dongguan University of Technology, Dongguan 523808, China)

Abstract: Aimed at problem that analyzing and extracting knowledge form massive data is high computation cost, a Hadoop based knowledge extraction framework is proposed. We designe a knowledge exraction framework which combines with the parallel processing and distributed storage feature, and the framework is compatible different prototype reduction methods. Based on the MapReduce programming method the prototype reduction method is parallelly processed, and a prototype reduction combination rule with high classification accuracy and computational speed is designed. Finally, experiments results based on real UCI big data sets show that the proposed framework improves two orders of magnitude of the classification time of the nearest neighbor classifier.

Key words: massive data; knowledge extraction; prototype reduction; cloud computing; parallel computing; data clustering

已有的高效分类算法对海量数据集处理, 需要极大的时空成本[1], 因此, 目前的研究热点集中于去除大规模数据集的冗余节点, 保留对分类贡献大的代表点, 进而降低数据规模, 提高分类速度[2]. 其中, 原型选择[3]与原型生成是两种重要的原型约简(PR)方法, 可以降低数据规模与噪声敏感度.

文献[4]混合了原型选择与生成两种 PR 方法, 通过优化原型的定位提高了最近邻分类器(NN)的性能. 文献[5]设计了一个两层的并行演化原型生成方法, 该方法进行大量的实验证明其可提高大数据最近邻分类器(NN)的性能. 文献[6]采用粒子群优化方法改善原型生成的结果, 该方法对中小规模的数据集效果较好. 文献[7]提出一种基于局部均值与类全局信息的近邻原型选择方法, 该方法不仅能较好地克服读取序列、异常样本对原型选取的影响, 降低原型集规模, 而且在保持高分类精度的同时, 实现了对数据集的高压缩效果.

已有的原型选择与原型生成算法对中、小规模的数据集效果较好, 但对于海量、高维数据性能依然不佳. 主要问题有: ①时间成本: PR 模型的复杂度高于 $O((n\square D)^2)$, 其中 n 是实例数量, D 是特征数量, 运行时间过长; ②内存消耗: 大多数 PR 方法需要存储许多中间数据或训练集, 若数据过大, 则会耗尽 RAM 内存.

Mapreduce 是一种处理大数据的分布式编程方法, 对编程者具有高度的透明性[8]. 而 Hadoop[9]是一个性能较好的开源分布式文件系统, Hadoop 集群为 master-slave

① 基金项目:广东省自然科学基金(S2013010011858);广东省高校优秀青年创新人才培养计划(2012LYM0125)
收稿时间:2016-02-29;收到修改稿时间:2016-04-08 [doi:10.15888/j.cnki.csa.005409]

架构, 一个 master 节点管理若干的 slave 节点, 并且 Hadoop 具有容错机制, 若节点出现故障, hadoop 在另一个节点自动重启该故障节点的任务. 结合Hadoop的优点, 本文将 NN 作为参考分类器, 基于 Hadoop 设计了一种知识提取框架, 可兼容不同的原型约简方法, 并可提高已有知识提取方法的计算效率与数据压缩率.

1 背景知识与相关问题

1.1 原型约简**(PR)**与大数据

PR 问题可如下描述: 设 TR 与 TS 分别为训练集与测试集, 其样本数量均为定值, 分别为 n 与 t. 假设其样本 $\mathbf{x}_p$ 是一个元组形式 $(\mathbf{x}_{p1},\mathbf{x}_{p2},...,\mathbf{x}_{pD},\omega)$, 其中 $\mathbf{x}_{pf}$ 表示第 p 个样本的第 f 个特征, 该样本属于类 ω, 数据空间维度为 D. 对于 TR 集, ω 为已知, 而 TS 集未知.

PR 的目标是获得约简的 RS 集合, 假设包括 $\mathbf{rs}$ ($\mathbf{rs}<\mathbf{n}$)个原型. 应该将 RS 充分地压缩, 以解决 NN(最近邻)分类器存储与计算成本过高的问题.

1.2 **Mapreduce**

Mapreduce 分为两个重要阶段: map 阶段与 reduce 阶段. 两个阶段的输入输出均为键值对($<k,v>$)形式, 并且两个阶段均由编程者设计与定义. (1)Map 阶段输入 $<k,v>$ 对, 生成 $<k,v>$ 对的中间集合, 然后, 合并所有键值相同的中间值, 并将其组织成一个列表. (2)Reduce 阶段将该列表作为输入, 生成最终的值. 在 Mapreduce 编程中, 所有的 map 与 reduce 操作均并行地运行: 首先, 所有的 map 函数独立地运行, 同时, 每个 reduce 函数等待对应的 map 函数结束; 然后, reduce 函数处理不同的键值. 同一个 MapReduce 所有任务的输入与输出存储于同一个分布式文件系统之中.

2 HDFR: 基于MapReduce的知识提取

2.1 **Map** 阶段

假设一个大小确定的训练集 TR, 以一个文件的形式存储于 HDFS(Hadoop 分布式文件系统)中. HDFR 的第一步是将 TR 分割为一定数量的不相交子集, 而从 Hadoop 的角度看, TR 文件由 h 个 HDFS 块组成. 假设 m 是 map 任务(task)的数量, 每个 map 任务表示为 $Map_1,Map_2,...,Map_m$, 其中 $1\le j\le m$. 因为分类程序按顺序运行, 因此, Map_j 对应 HDFS 子数据集(共 h/m 个)的第 j 个数据块, 每个 map 任务处理合适数量的数据.

如果直接对 TR 使用分类算法, 则每个子集 TR_j 的类分布会接近数据实例的原有分布(文件中的分布情况). 如引言部分所述, 如果 TR 的大小大于主内存容量, 则无法运行分类算法. 为了使用分类算法, 首先需要将整个大文件随机化分区处理, 虽然, 无法确保每个分区可合适地代表 TR 中实例的数量, 然而, 根据原有 TR 中属于类 ω 的概率, 总体概率上每个数据块应该包含类 ω 中的合适数量样本.

所有 map 生成其对应的 TR_j 之后, 运行一个 PR 程序将 TR_j 作为训练数据输入, 该步骤最终生成一个约简的集合 RS_j. 算法 1 所示为 map 函数的伪代码, 该函数对每个训练集分区运行 PR 算法.

算法 1 map 阶段

输入: 分割 j 的数量
输出: RS_j

1.使用分割的 j 个实例构成 TR_j
2. $RS_j=\text{PrototypeReduction}(TR_j)$ //原型约简

2.2 **Reduce** 阶段

Reduce 阶段将所有 RS_j 迭代地聚集为一个 RS. 算法 2 所示为 reduce 函数的伪代码, 初始化 $RS=\varnothing$. 本文设计的 reduce 方法包括:

(1) Join(连接): 连接是一种关系运算, 用于合并关系. Mapper 结束之后, 基于分层处理, 将所有的 RS_j 集合连接为一个最终的约简集 RS. 连接程序无法保证最终 RS 不包含冗余或噪声样本, 但可将其作为基线 RS 版本.

(2) Filtering(过滤): 过滤将 map 端输入数据中不需要的部分丢弃. 此类方法主要基于简单的启发式方法, 过滤噪声点及其邻居点, 该操作可为 NN 分类器提供一个平滑的决策边界. 本文在 map 阶段将每个分区约简为一个代表实例, 为了将 map 阶段的输出聚集为一个 RS 集合, 删除其中的噪声样本.

(3) Fusion(融合): 融合冗余的原型. 该类型方法主要指基于中心的原型生成方法[10].

HDFR 仅使用一个单独的 reducer, 根据不同的应用场景选择具体的 reducer 类型, 以此降低 Mapreduce 的成本[11]. 图 1 所示为 HDFR 的流程框图.

算法 2 reduce 阶段

输入: RS_j ,reducer 类型
输出: RS

1. $RS=RS\cup RS_j$

2.**IF** reducer 类型为 Filtering　//过滤
3.　$RS = Filtering(RS)$
4.**ENDIF**
5.**IF** reducer 类型为 Fusion　//融合
6.　$RS = Fusion(RS)$
7.**ENDIF**

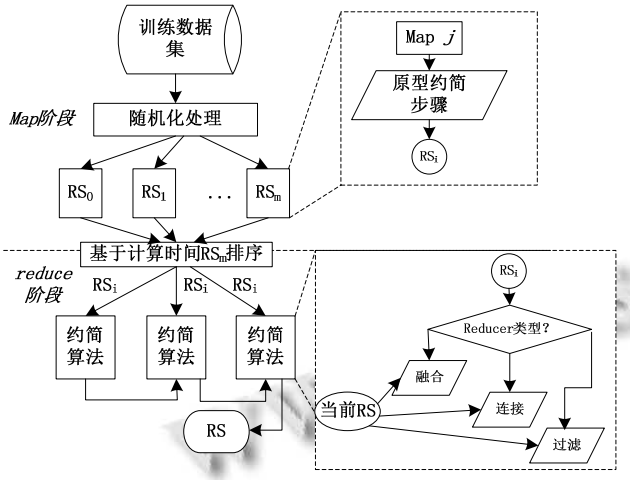

图 1　HDFR 的流程框图.

图 1 本文基于 Mapreduce 的知识提取框架框图

2.3 PR 函数族的选择

文献[12]总结了 6 个主要的 PR 方法族: edition、condensation、混合类算法、定位调整、基于质心的方法、空间分割类方法. 每个方法族内的方法思想接近.

(1) Edition: 此类方法通过删除噪声数据删除冗余数据. 此类方法的特点是速度快、准确率高, 但其压缩比例较低. 此类方法适合放入本文框架的快速约简 reduce 操作.

(2) Condensation、hybrid 与空间分割类三种方法总体而言, 可较好地权衡压缩率、分类准确率与运行时间之间的关系, 其压缩比例约在 60%~80%.

(3) 定位调节技术: 此类方法可获得较高的压缩率, 总体而言, 可在合理的时间内获得较为准确的结果. 为了实现此类技术, 推荐使用 ICPL 作为 fusion 的方法, 该方法速度较快、压缩率搞.

(4) 基于质心的算法: 此类方法准确率高, 但时间消耗较高, 用于 reduce 阶段较为合适.

在 map 与 reduce 阶段结合不同的 PR 技术, 本文推荐了如下两条规则: ①压缩率高的 map 阶段与准确率高 reducer 配合; ②压缩率低的 map 阶段与速度快的 reducer 配合.

为了证明本方法的通用性, 采用几个经典的 PR 方法进行试验. 遵从上文推荐的规则, 本实验将基于 edition 的方法作为过滤方法, 将基于质心的方法作为融合方法. 本实验的 edition 技术采用经典的 ENN 方法[13], 该方法思想为: 如果某个样本与其 k 个近邻中的大多数样本均不一致, 则删除该样本, 该方法实现简单, 效果较好. 本文融合方法采用 ICLP2 算法[14], ICPL2 模型通过识别边界并合并边界以外的样本, 以此整合多个原型. 将 GMCA 方法[15]作为快速的 reducer 方法, GMCA 方法基于多层的聚类合并原型, 该方法较好地权衡了准确率与运行时间, 性能较好.

3 实验结果与分析

3.1 性能参数选择与实验环境

本文使用并行 PR 系统提高 NN 分类器性能. 本文测试以下 5 个性能参数: ①分类精度; ②原型压缩率: 以存储空间的减少比例来衡量: 约简率 $=1-size(RS)/size(TR)$; ③运行时间: HDFR 生成 RS 消耗的总时间; ④分类时间: 分类消耗的总时间; ⑤速度提高: 提速=$\frac{\text{并行时间}}{\text{最小数量mapper的并行时间}}$.

本文的实验环境为: 1 个 master 节点与 10 个 slave 节点. 每个节点配置为: 处理器: Intel Xeon CPU E5-2620, 核心数量: 6, 主频: 2.0GHz, 网络: 100Mbps, 硬盘大小: 1TB, 内存: 8GB. 软件参数为: hadoop-2.0.0-cdh4.2.0; 最大 map 任务数量: 128; 最大 reducer 数量: 1; 机器学习库: Apache Mahout 0.8, 操作系统: Ubuntu 12.04.

3.2 数据集与实验方法

本文采用 UCI 数据集[16]的高维数据集, 表 1 所示为所选的两个数据集的简介, 对这些数据集使用 5 层验证方法(5-fcv)分区: 数据集分为 5 层, 每层包含 20% 样本, 选择一层作为测试库, 其余 4 层为训练集. 然后, 使用 NN 规则测试测试库的 RS, 因为部分操作具有随机性, 因此每个分区独立运行 3 次取均值.

如上文所述, 本文采用混合 SSMA-SFLSDE 算法[17]测试 HDFR 模型, 该方法是一种原型选择与生成的混合算法.

将 NN 分类器作为性能结果的基准参考, 实验的具体参数设置为: mapper 数量: 64/128/256/512/1024, reducer 数量: 1, reduce 类型: 连接/过滤/融合. 其他文

献的参数均根据文献作者的推荐设置. 本文研究的目标并非提高 PR 方法的分类准确率, 而是提高运行时间, 因此, 本文研究了 mappers 数量与 reduce 类型对分类准确率、运行时间的影响.

表 1 KddCup 与 Susy 数据集简介

数据集	样本数	维度	ω
KddCup 1999	4856151	41	2
Susy	5000000	18	2

3.3 实验结果与分析

将 SSMA-SFLSDE 方法融合本文 HDFR 框架, 测其性能, 从两个角度统计结果: (1)三个 reducer 的分类准确率与压缩率, 将结果与 NN 规则进行比较; (2)本方法的运行时间.

表 2~3 总结了 SSMA-SFLSDE 算法与本文框架获得的训练集与测试集准确率、运行时间与压缩率. 本文 HDFR 框架中, 将不同的 mapper 数量与 reduce 类型组合, 统计每个组合的平均值与标准偏差. 平均分类时间定义为: 将所有 TS 样本分类与 HDFR 生成 RS 所需的时间之和.

表 2 Kddcup 数据集的实验结果统计

Reduce 类型	Mapper 数量	训练集		测试集		运行时间		约简率		分类时间
		均值	标准偏差	均值	标准偏差	均值	标准偏差	均值	标准偏差	分类时间(秒)
Join	256	0.9991	0.0003	0.9993	0.0003	8536.4206	153.7057	99.9208	0.0007	1630.8426
Filtering	256	0.9991	0.0003	0.9991	0.0003	8655.6950	148.6363	99.9249	0.0009	1308.1294
Fusion	256	0.9994	0.0000	0.9994	0.0000	8655.6950	148.6363	99.9279	0.0008	1110.4478
Join	512	0.9991	0.0001	0.9992	0.0001	4614.9390	336.0808	99.8645	0.0010	5569.8084
Filtering	512	0.9989	0.0001	0.9989	0.0001	4941.7682	44.8844	99.8708	0.0013	5430.4020
Fusion	512	0.9992	0.0001	0.9993	0.0001	5018.0266	62.0603	99.8660	0.0006	2278.2806
Join	1024	0.9990	0.0002	0.9991	0.0002	2620.5402	186.5208	99.7490	0.0010	5724.4108
Filtering	1024	0.9989	0.0000	0.9989	0.0001	3103.3776	15.4037	99.7606	0.0011	4036.5422
Fusion	1024	0.9991	0.0002	0.9991	0.0002	3191.2468	75.9777	99.7492	0.0010	4247.8348
NN	0	0.9994	0.0001	0.9993	0.0001					2354279.8650

表 3 Susy 数据集的实验结果统计

Reduce 类型	Mapper 数量	训练集		测试集		运行时间		约简率		分类时间
		均值	标准偏差	均值	标准偏差	均值	标准偏差	均值	标准偏差	分类时间(秒)
Join	256	0.6953	0.0005	0.7234	0.0004	69153.3210	4568.5774	97.4192	0.0604	30347.0420
Filtering	256	0.6941	0.0001	0.7282	0.0003	66370.7020	4352.1144	97.7690	0.0046	24686.3550
Fusion	256	0.6870	0.0002	0.7240	0.0002	69796.7260	4103.9986	98.9068	0.0040	11421.6820
Join	512	0.6896	0.0012	0.7217	0.0003	26011.2780	486.6898	97.2050	0.0052	35067.5140
Filtering	512	0.6898	0.0002	0.7241	0.0003	28508.2390	484.5556	97.5609	0.0036	24867.5478
Fusion	512	0.6810	0.0002	0.7230	0.0002	30344.2770	489.8877	98.8337	0.0302	12169.2180
Join	1024	0.6939	0.0198	0.7188	0.0417	13524.5692	1941.2683	97.1541	0.5367	45387.6154
Filtering	1024	0.6826	0.0005	0.7226	0.0006	14510.9125	431.5152	97.3203	0.0111	32568.3810
Fusion	1024	0.6757	0.0004	0.7208	0.0008	15562.1193	327.8043	98.7049	0.0044	12135.8233
NN	0	0.6899	0.0001	0.7157	0.0001					1167200.3250

3.3.1 准确率与约简结果与分析

从表 2、3 中不同数量 mapper 获得的准确率测试结果可看出, reduce 类型对准确率有影响, 同时可获得以下结论:

① 因为 HDFR 框架中 PR 算法并未采用完整的信息, 所以随着训练集中样本数量降低分类准确率下降. 在 Susy 数据集中, 随着 mapper 数量的增加, 准确率稍有下降; 而对于 Kddcup, 随着 mapper 数量的增加其性能略微下降.

② 尽管 PR 算法的精确率随着 mapper 数量增加而逐渐下降, 并且与 NN 规则的差距较小, 原因主要在于: PR 方法从 TR 集中删除了噪声样本, 从而影响了 NN 分类器的分类性能, 此外, PR 模型会使得类簇之间的决策边界平滑.

③ 随着 mapper 数量的增加, 压缩率应该降低, 然而, 从结果看出 PR 的约减比例极高.

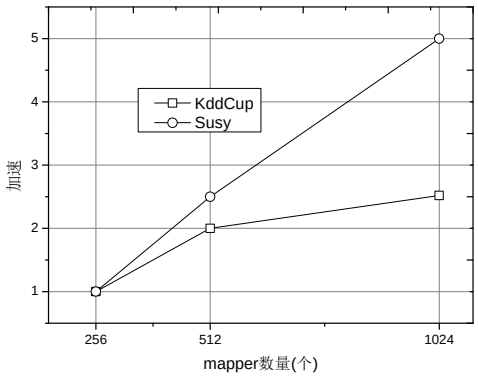

图 2　HDFR 融合 reducer 的加速结果

④ 总体而言, 融合方法优于其它类型的 reducer. 融合方法可获得较好的训练与测试准确率.

⑤ 本文 HDFR 模型的准确率与 NN 分类器极为接近, 但可以极大地降低了分类时间.

3.3.2 本文框架的加速效果

本框架利用并行处理框架旨在提高传统分类器的运行速度. 图 2 所示是融合 reducer 下 HDFR 的加速结果. 因为本文的加速度量采用最小数量的 mapper(minMaps)运行时间作为参考时间, 所以加速结果具有一定的一般性. 将每个数据集的 minMaps 运行时间设为 1. 从图 2 中可看出, 不同的数据集结果不同, 原因在于不同数据集样本的特征数量不同, 由此决定了 PR 技术的复杂度, 所以 susy 数据集的加速性能优于 KddCup.

4　结语

本文结合 Hadoop 的并行处理能力与分布式存储特点, 设计了一种知识提取框架, 可兼容不同的原型约简方法, 在保持与分类器原有分类精确率接近的情况下, 大幅度提高海量数据分类的计算效率. 最终基于真实 UCI 大数据集进行实验, 本方法将最近邻分类器的分类时间提高两个数量级.

参考文献

1 李娟,王宇平.考虑局部均值和类全局信息的快速近邻原型选择算法.自动化学报,2014,40(6):1116–1125.

2 李娟,王宇平.自适应边界逼近的原型选择算法.模式识别与人工智能,2015,28(6):568–576.

3 García-Pedrajas N, Pérez-Rodríguez J. Multi-selection of instances: A straightforward way to improve evolutionary instance selection. Applied Soft Computing, 2012, 12(11): 3590–3602.

4 Triguero I, García S, Herrera F. Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification. Pattern Recognition, 2011, 44(4): 901–916.

5 Triguero I, Peralta D, Bacardit J, et al. A combined MapReduce-windowing two-level parallel scheme for evolutionary prototype generation. 2014 IEEE Congress on Evolutionary Computation (CEC). IEEE. 2014. 3036–3043.

6 Chen JL, Tseng SP, Yang CS. Using particle swarm method to optimize the proportion of class label for prototype generation in nearest neighbor classification. Advanced Technologies, Embedded and Multimedia for Human-Centric Computing. Springer Netherlands, 2014: 239–245.

7 李娟,王宇平.考虑局部均值和类全局信息的快速近邻原型选择算法.自动化学报,2014,40(6):1116–1125.

8 李建江,崔健,王聃,等.MapReduce 并行编程模型研究综述. 电子学报,2011,11(11):2635–2642.

9 崔杰,李陶深,兰红星.基于 Hadoop 的海量数据存储平台设计与开发.计算机研究与发展,2012,49:12–18.

10 Triguero I, Derrac J, Garcia S, et al. A taxonomy and experimental study on prototype generation for nearest neighbor classification. IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 2012, 42(1): 86–100.

11 Chu C, Kim S K, Lin YA, et al. Map-reduce for machine learning on multicore. Advances in Neural Information Processing Systems, 2007, 19: 281.

12 Garcia S, Derrac J, Cano JR, et al. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2012, 34(3): 417–435.

13 Wilson DL. Asymptotic properties of nearest neighbor rules using edited data. IEEE Trans. on Systems, Man and Cybernetics, 1972, (3): 408–421.

14 Lam W, Keung CK, Liu D. Discovering useful concept prototypes for classification based on filtering and abstraction. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2002, 24(8): 1075–1090.

15 Mollineda R, Ferri FJ, Vidal E. A merge-based condensing strategy for multiple prototype classifiers. IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics, 2002, 32(5): 662–668.

16 Asuncion A, Newman D. UCI machine learning repository. University of California Irvine School of Information, 2008, 14/8.

17 Triguero I, García S, Herrera F. Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification. Pattern Recognition, 2011, 44(4): 901–916.