DOI:10.7544/issn1000-1239.2016.20160140
53(8):1792-1805,2016

HSSM:一种流数据分层次模最大化方法

张奋翔　陈华辉　钱江波　董一鸿
(宁波大学信息科学与工程学院　浙江宁波　315211)
(zhang_fenxiang@163.com)

HSSM: A Hierarchical Method for Streaming Submodular Maximization

Zhang Fenxiang, Chen Huahui, Qian Jiangbo, and Dong Yihong
(*Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo, Zhejiang* 315211)

Abstract How to extract k elements from a large data stream according to some utility functions can be reduced to maximizing a submodular set function. The traditional algorithms had already given some good solutions of summarizing a static data set by submodular method, well-known as standard greedy algorithm. Lastly researches also presented some new algorithms with corresponding distributed solutions to meet the streaming data access and the real-time response limits, but those algorithms are not good enough in maximizing the utility gain. In this paper, we propose a new algorithm called HSSM which runs as a pipelining distributed framework and requires only a single-pass access the data set. Finally, the utility gain of our solution is close to the option standard greedy solution. We also propose using utility vector to compress the set of summarization and filtering out the low gain objects to improve the original HSSM algorithm. Fortunately, this approach can get applications in many related fields such as representative articles selection, k-medoid select problem and so on. Experimental results show that the improved Spark-HSSM+ method can increase the summarization speed in direct proportion to k^2 in contrast to the traditional method. Compared with other distributed algorithms, the result of the Spark-HSSM+ method is the most close to the standard greedy solution.

Key words streaming submodular maximization; hierarchy model; pipelining parallelism; data summarization; Spark distribution platform

摘　要　从大规模数据中“摘要”出最能满足效用函数收益的有限个数据对象,可以被归纳为次模函数最大化问题.并行过滤算法在满足流数据访问次数限制与实时响应的条件下,通过分布式筛选的方式实现次规模最大化,但在提升摘要速率时效用函数收益损失较大.提出一种流数据分层次模最大化算法HSSM,在仅访问一次数据集的条件下,采用流水并行的分布式处理框架得到接近于标准贪心算法的次模函数收益,同时改进HSSM通过累积摘要的压缩存储、分层过滤低增益对象提升摘要速率.该方法在数据摘要问题的相关领域具有广泛的应用性,如文档集中代表性文章的选取、数据集中心点选取等.实验结果显示,分布式算法Spark-HSSM+对比于传统的算法在运行速率上达到与摘要规模k成k^2正比例关系的提升.而相对于其他分布式算法,其实验效用收益与理论最差收益都更接近于贪心算法.

收稿日期:2016-03-11;**修回日期**:2016-05-31
基金项目:国家自然科学基金项目(61572266,61472194)
This work was supported by the National Natural Science Foundation of China (61572266,61472194).

关键词 流次模最大化;分层模型;流水并行;数据摘要;Spark 分布式平台

中图法分类号 TP391

在信息飞速增长的时代,经常需要从海量数据或仅能访问一次的流数据中"摘要"出一个较小的且具有代表性的数据集[1-2],例如,热点文章及热门微博推荐[3-4]、文档中概要语句提取[5]、数据集中心点选取[6]、数据集中噪声的查找[7]和社交网络中最大影响力节点的选取[8]等.摘要方法通常使用某种效用函数或称收益函数(utility function)来评价所选子集的"代表性"是否达到目的.数据对象在摘要过程中往往具有收益递减的所谓次模特性[9].次模最大化即研究如何利用该次模特性摘要出一个小规模对象集合,实现对原始数据集的主要特征概括.

围绕次模最大化问题已有不少的相关研究.最先由 Nemhauser 等人[9]提出使用标准贪心算法在有限次访问数据集的条件下,得到具有最低收益保证的摘要结果;Mirzasoleiman 等人[10]设计出适用于处理静态大规模数据集的主从分布式摘要算法,在运行效率上对标准贪心算法做出改进;Gomes 等人[11]针对无法二次访问的动态流数据,提出了通过有效维护一个指定大小的对象集合完成近似最大化收益摘要,但存在流数据实时响应速率慢的不足;Badanidiyuru 等人[12]使用离散化预估最优收益的方法设计有限数量的过滤器,通过并行筛选的思想提升流数据实时摘要速率.

Badanidiyuru 等人虽然在摘要的实时性方面提出改进,但存在摘要结果收益降低的缺陷.针对该问题,本文提出一种流数据分层次模最大化摘要算法(hierarchical streaming submodular maximization, HSSM),采用分层流水并行和逐层摘要的策略实现分布式运算与次模收益最大化.该方法仅需要访问一遍数据集,即可得到规模为 k 且效用收益至少满足 $1/(2-1/k)$ 倍最优解的摘要结果.对每批数据量为 v 的流数据摘要,仅需要 $O(v+k)$ 的时间开销即可完成更新.该方法摘要的理论与实验收益优于其他并行算法而略低于贪心算法,但在运行效率上较贪心算法有很大的提高.实验证明:本文算法可以有效地解决流数据的次模函数最大化问题.

1 流次模最大化问题

1.1 次模函数最大化

从规模为 n 的数据集合 O 中选取大小为 k 的对象集合 S 作为摘要.效用函数 $f(S)$ 用于表示摘要结果 S 对数据集 O 的效用收益,不同应用场合可有不同的效用函数.如 1.2 节的例子分别使用基于覆盖率与距离损失函数的效用函数,由以下定义得知,该效用函数均为次模函数.

定义 1. 次模函数.有限集合 O 和定义在其幂集 $2^O\to\mathbb{R}$ 上的单调实函数 f,称 f 为次模函数当且仅当 O 的任意 2 个子集 A,B 及元素 e,$A\subseteq B\subseteq O$,$e\in O\backslash B$,满足:

$$f(A\cup\{e\})-f(A)\geqslant f(B\cup\{e\})-f(B). \quad (1)$$

用 $\Delta f(e|A)=f(A\cup\{e\})-f(A)$ 表示元素 e 对集合 A 的效用增益(marginal gain),式(1)说明次模函数的收益递减性质:若其他元素被先于元素 e 添加至集合 A,将导致元素 e 的效用增益降低,即 $\Delta f(e|B)\leqslant\Delta f(e|A)$.

定义 2. 次模函数收益最大化(简称次模最大化).从数据集 O 中萃取出有限规模为 k 的代表性子集 S,通过次模函数的收益递减特性选取近似最优解,使得 S 满足效用收益 $f(S)$ 最大化,该过程称为次模最大化:

$$\max_{S\subseteq O} f(S) \quad \text{s.t.}\ |S|=k. \quad (2)$$

贪心算法是次模最大化的代表,其通过 k 轮迭代添加效用增益最大元素 $e\in O\backslash A$ 至集合 A,使得规模为 k 的集合 A 效用收益尽可能最大化,并通过次模最大化特性证明其在最差情况下的收益保证.

定义 3. 摘要.基于效用评价函数从集合 O 中选取一个规模较小且具有高效用收益的子集 S,则称子集 S 为集合 O 的摘要.

实际应用中,相当多的摘要效用函数为次模函数,而式(2)的求解是 NP-hard 难问题:通过对比集合中所有可能解(从 n 个元素集合中任意组合取出 k 个元素)的效用收益来得到最优解的方法,其开销为 $O(n^k)$.记 $f(S^*)=\max\limits_{S\subseteq O} f(S)$,$S^*$ 是摘要的最优解,则摘要问题可通过次模最大化方法,实现在有限次访问的条件下得到规模为 k 且收益最大化的摘要结果.

1.2 次模最大化的典型问题

本文主要关心持续不断到来的流数据或仅能访问一遍的大规模数据摘要问题,下面是 2 个典型例子.

1.2.1 基于覆盖率的摘要问题

对象 e 的特征向量 $\boldsymbol{G}_e$ 包含若干个信息单元 u(如文章的关键字、图片涉及的人物地点等信息),即 $\boldsymbol{G}_e=\{u_1,u_2,\cdots,u_n\}$. 通过 w_u 表示信息单元 u 的权重,对象 e 的效用收益函数为 $f(\{e\})=\sum_{u\in\boldsymbol{G}_e}w_u$. 在大规模流数据集中选取 k 个最具有代表性的对象集合 S,定义覆盖率函数如下:

$$Cov(S,O)=\frac{f(S)}{f(O)}=\frac{\sum_{u\in\boldsymbol{G}_e,e\in S}w_u}{\sum_{u\in\boldsymbol{G}_e,e\in O}w_u}. \tag{3}$$

摘要集合应尽可能覆盖数据集的总信息收益 $f(O)$,以覆盖率(式(3))表示摘要 S 包含数据集 O 的信息收益比例[1,13]. 在逐个选取摘要对象的过程中,覆盖率函数满足 $\Delta Cov(e|S_{i-1})\geqslant\Delta Cov(e|S_i)$ 的次模特性,故该问题是次模最大化问题.

1.2.2 基于距离损失函数的中心点选取问题

求解 k-medoid(k-中心点)问题是典型的数据挖掘应用[14-15],通过最小距离损失选取若干个对象为数据集中心点(如聚类中心等). 其评价标准是最小化中心点与其他对象的距离之和,对于一个数据集 O,使用距离函数(如欧氏距离公式:$d(x,x')=\|x-x'\|$)表示集合 O 中对象间距离:$O\times O\to\mathbb{R}$. 该问题的摘要损失函数 L 被定义如下[11]:

$$L(S)=\frac{1}{|O|}\sum_{e\in O}\min_{h\in S}d(e,h). \tag{4}$$

通过辅助对象 $e_0=0$,即可将 L 转换为具有次模特性的距离损失函数:

$$f(S)=L(\{e_0\})-L(S\cup\{e_0\}). \tag{5}$$

1.3 次模最大化问题的相关算法

1.3.1 标准次模最大化算法

Nemhauser 等人[9]提出的 Standard-Greedy 算法(及与次模相关的标准贪心算法[16-17])在解决次模最大化问题中给出近似最优解方案. 该算法每次迭代访问数据集时,将满足最大化边际收益的对象添加到摘要集合中:

$$S_i=S_{i-1}\cup\{\arg\max\Delta f(e|S_{i-1})\}. \tag{6}$$

文献[9]证明了在最坏情况下,该算法依然可以给出效用收益满足大于$(1-1/e)$倍的最优解收益,并且没有其他方法能在同样访问量级上保证更优的最差解收益.

1.3.2 流次模最大化算法

与静态数据处理不同,流数据具有动态增长的特点(搜索引擎 Bing 每 10 min 就产生超过 1 TB 的数据[18]). 流数据摘要算法的提出是为了解决静态摘要方法在多次访问数据集方面的不足.

流数据的摘要算法是在有限的时间及内存空间开销条件下得到近似最优收益的解决方案,主要包含以下 3 个难点:

1) 摘要算法访问数据集的总次数,即非特定条件下流数据不能被二次访问.

2) 算法完成摘要的运行时间. 流数据处理要求极高的实时性,因此摘要完成的运行时间是评价流算法符合需要的重要指标之一.

3) 算法最差解的摘要收益. 流算法通常由于条件限制而给出近似解,故必须保证算法在最差情况下解收益的可接受性.

Gomes 等人提出的 Stream-Greedy 算法[11]通过对比替换的方式,维持摘要 S 的效用收益最大化:若存在新对象 o_i 替换 $b\in S_k$,使得摘要 S 的效用收益增加,则替换 $S_k=S_k\cup\{o_i\}\backslash\{b\}$. 该方法在解决流次模摘要问题的同时仍存在 2 方面缺陷:1)在数据增益非均匀的情况下,该算法的摘要收益仅为 $1/k$ 最优解;2)该算法对于分批规模为 v 的流数据更新时间复杂度为 $\Omega(kv)$,因此无法在短时间完成对大规模摘要(k 很大)的更新.

1.3.3 分布式次模最大化算法

为了解决流算法更新时间开销大的难题,相应的分布式解决方案被提出. Mirzasoleiman 等人[10]与 Kumar 等人[19]分别提出解决次模函数最大化问题的分布式算法. 其中,文献[19]的 Greedy-Scaling 算法要求已知单对象最大增益值(即摘要至少访问 2 次数据集)且未给出具体实现,故本文未对其进行对比.

由 Mirzasoleiman 等人[10]提出的 GreeDi 算法对标准贪心算法提出改进. 该算法将规模为 n 的数据分割成为 m 块大小为 $\lceil n/m\rceil$ 的数据集,通过分布式贪心选取子集的最大收益集合,再将子集合并进行二次贪心的方式得到摘要结果. 该方法适合处理静态大规模数据,但不适合处理数据规模未知的流数据.

Badanidiyuru 等人[12]提出具有最差收益保证的快速分布式次模摘要方法(Sieve-Stream),实现以当前单个对象的最大收益值 m 及离散系数 ε 预估出最优解集合 $T=\{(1+\varepsilon)^i\,|\,m\leqslant(1+\varepsilon)^i\leqslant k\times m\}$. 对于新数据 e,通过 i 个以 $t_i\in T$ 作为目标收益的过滤器并行筛选:当摘要规模 $|S|<k$ 且对象 e 满足增益 $\Delta f(e|S_i)\geqslant(t_i-f(S_i))/(k-|S_i|)$ 时,将

对象 e 添加至摘要集合 S_i. 该算法通过筛选阈值的方法保证次模收益,使得至少存在一个摘要结果 S_i,其满足大于$(1/2-\varepsilon)$的最优解收益.

文献[12]是具有最低收益保证且满足分布式快速摘要的次模最大化算法,但该算法在流数据筛选过程中缺陷.一方面,若指定摘要规模为 k,由于流数据具有增益不确定性(因此需要最差解保证),使得大多数过滤器通常处于非饱和状态,故摘要并没有达到收益最大化;另一方面,对于因单对象最大收益增大而新产生的过滤器,由于该过滤器无法访问已处理数据(流数据在非特定情况下只能访问一次),将导致许多高收益的对象被忽略处理.实验表明,该算法在实际摘要问题中的确存在效用收益明显低于大部分次模摘要算法的现象.因此,本文旨在提出新的分布式次模最大化算法,同时满足尽可能高的效用收益.

2 流数据分层次模最大化算法

由于传统次模最大化问题的研究无法解决流数据摘要问题;同时,现存的流算法在提高摘要效率的同时存在降低摘要收益或最低收益保证过低的问题.因此,本文提出基于次模最大化的分层流摘要算法旨在解决 3 个难点:1)算法可以适用于解决流数据摘要问题;2)通过次模特性使得摘要获得尽可能高的收益及满足最低收益保证;3)设计分布式处理框架旨在提高摘要速率.

2.1 分层次模最大化

本节设计一种新的并行处理框架用于解决流数据摘要在分布式处理方面的难题.由流水线并行(pipeline)[20]处理技术启发,通过分割次模增益的方法提出分层次模最大化算法 HSSM.

根据 1.1 节给出的定义,为使摘要获得尽可能高的收益,次模最大化方法通过 k 轮迭代添加最大增益对象组成摘要集合 S.由于分批到来的流数据不能够一次性全部访问,因此通过分层保留已访问数据的逐轮次优摘要对象 h_i 来替代全局的最高增益对象.新数据对次优累积摘要的效用增益为:$\Delta f(e_i|S_j)=f(S_j\cup\{e_i\})-f(S_j)$,$S_j=\{h_1,h_2,\cdots,h_j\}$.若数据 e_i 的增益高于某轮最大增益对象 h_j 时,替换 h_j 与 e_i,进而实现流数据的摘要过程满足次模最大化.

由于同一时刻的数据增益筛选过程相互独立(即数据对象 e_i 对于不同的累积摘要 S_j 需被计算 k 次),进而通过如图 1 所示的流水并行模式分层进行数据增益对比.其中,时刻 T_i 的数据在顶层 V(累积摘要为空时)完成增益对比;与此同时,其余层也分别基于不同的累积摘要 S_j 对数据 e_{i-j+1} 进行的增益筛选.

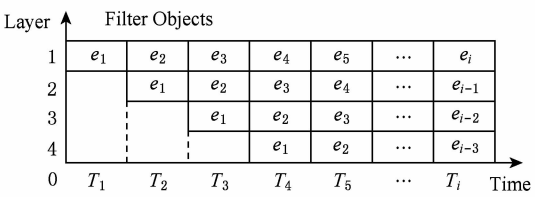


Fig. 1 Data processing in pipeline mode.

图 1 流水并行模式处理数据

总体来说,传统的迭代算法使得规模为 n 的数据集被访问 k 轮后完成摘要筛选工作,其时间复杂度为 $\Omega(nk)$.而流水并行的方法通过在连续的个时间单位比较待处理数据,进而实现基于不同累积摘要的多次增益筛选,使得总处理时间减少为 $\Omega(n+k)$,一般而言 $v\gg k$,故摘要的总时间复杂度约为 $\Omega(n)$.

结合流水并行模式的分层摘要算法 HSSM 如图 2 所示,其对单个数据对象处理的主要步骤如下:

1) 算法从流数据中逐一获取待处理数据 e_i,并将该数据作为输入传入首层.

2) 构建分层(图 1 中 $Layer_l$)数据结构:e_l 表示第 l 层的待处理对象;S_{l-1} 表示第 l 层之上的累积摘要结果:初始化 S_0 对象为空,逐层完成增益筛选后,更新 $S_l=S_{l-1}\cup\{h_l\}$ 并传递给 $l+1$ 层完成增益累积;摘要维持对象 h_l 是自身与待处理对象 e_l 基于累积摘要 S_{l-1} 对比后的高收益者:

$$h'_l=\arg\max(\Delta f(h_l|S_{l-1}),\Delta f(e_l|S_{l-1})). \quad (7)$$

3) 通过自顶(第 1 层)向下的方式,逐层完成数据 e_l 与维持对象 h_l 的筛选.根据增益比较结果输出新摘要维持对象 h'_l、下层待处理数据 e_{l+1}、累积摘要集合 S_l.最终保留摘要维持对象 h_l,并将未处理数据 e_{l+1} 与累积摘要 S_l 传入 $l+1$ 层完成迭代操作,最终所有数据都被分层模型过滤或保留.

4) 算法在处理完所有数据后,通过合并各 $Layer_l$ 中摘要维持对象组成规模为 k 的摘要结果 $S=\{h_1,h_2,\cdots,h_k\}$.

算法 1 完成了分层模型中 $Layer_l$ 的数据筛选工作,行①②为摘要维持对象与待处理数据的增益计算,其中 f 为任意的效用函数;行③~⑨首先将待处理数据与摘要维持对象进行增益对比,对比的结果决定是否更新摘要维持对象,同时返回下层待处理数据与累积摘要.

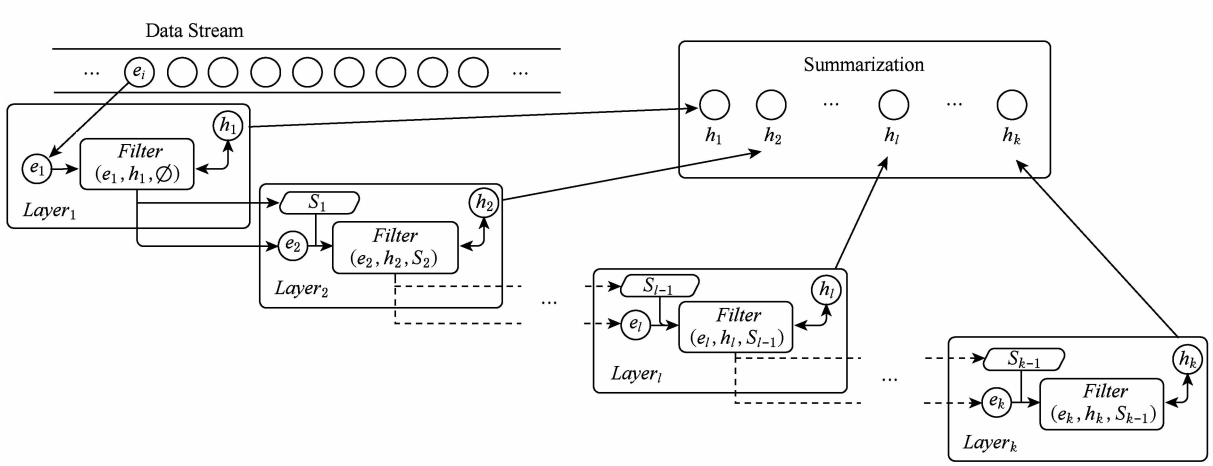


Fig. 2 The processing of per object in HSSM.

图 2 流数据对象在 HSSM 算法中的顺序处理流程

算法 1. 增益筛选算法 $Filter(e_i, h_l, S_{l-1})$.

输入:待处理数据 e_i、摘要维持对象 h_l、累积摘要 S_{l-1};

输出:下层待处理对象 e_{i+1}、本层摘要维持对象 h_l、下层累积摘要 S_l.

① 计算维持对象 h_l 增益:$g\_h=\Delta f(h_l|S_{l-1})$;

② 计算待处理对象 e_i 增益:$g\_e=\Delta f(e_i|S_{l-1})$;

③ if $g\_e>g\_h$

④ /* e_i 替换 h_l,返回原 h_l 与新累积摘要 */

⑤ $tmp=h_{ll}, h_l=e_i$;

⑥ return $[tmp, h_l, S_{l-1}\cup\{h_l\}]$;

⑦ else

⑧ /* 返回 e_i 与原累积摘要 */

⑨ return $[e_i, h_l, S_{l-1}\cup\{h_l\}]$.

综上所述,HSSM 算法通过流水线并行的方式分层解决流数据次模最大化摘要问题,同时该算法通过指定分层数量 k 确定摘要结果规模.

2.2 HSSM 算法的收益分析

HSSM 算法与其他次模最大化方法都无法保证摘要结果在所有情况下都达到最优解收益.本节旨在分析该算法在最差情况下的摘要效用收益保证.

定理 1. 在摘要维持对象$(h_1, h_2, \cdots, h_k)$未改变的情况下(即被过滤对象 $e_k=e_i=e_1$),过滤对象 e_k 满足 $\Delta f(e_k|S_{k-1})\leqslant f(S_k)/k$.

证明. 在 HSSM 算法中,设 h_i 是第 i 层的摘要维持对象,$i=1,2,\cdots,k$,满足收益递减特性:

$$\Delta f(h_1|S_0)\geqslant\Delta f(h_2|S_1)\geqslant\cdots\geqslant\Delta f(h_k|S_{k-1}). \quad (8)$$

被第 i 层过滤的数据对象 e_i,其增益小等于该层摘要维持对象:$\Delta f(e_i|S_{i-1})\leqslant\Delta f(h_i|S_{i-1})$.特别地,当该对象被所有层过滤时,其增益小于式(8)中最小增益维持对象:$\Delta f(e_i|S_{k-1})\leqslant\Delta f(h_k|S_{k-1})$.

由式(8)同时得出,第 i 层摘要维持对象 h_i 的收益小于等于本层累积摘要的平均收益:$\Delta f(h_i|S_{i-1})\leqslant f(S_i)/i$,且平均累积收益同时满足逐层递减特性:$f(S_1)/1\geqslant f(S_2)/2\geqslant\cdots\geqslant f(S_k)/k$.故推出被过滤对象 e_k 在所有层被过滤时,满足增益小等于累积摘要集合的最小平均收益:

$$\Delta f(e_k|S_{k-1})\leqslant\Delta f(h_k|S_{k-1})\leqslant f(S_k)/k.$$ 证毕.

定理 2. 摘要过程中,若摘要维持对象 S 被替换 $S'=(h'_1, h'_2, \cdots, h'_k)$,则集合中任意对象 h'_i 对于已过滤对象 e_i 满足 $\Delta f(e_i|S'_{i-1})\leqslant\Delta f(h'_i|S'_{i-1})$.

证明. 由式(7)替换得到新摘要维持对象 h'_i,满足增益增加原则:$\Delta f(h'_i|S'_{i-1})>\Delta f(h_i|S_{i-1})$,即 $\Delta f(h'_i|S'_{i-1})=\Delta f(h_i|S_{i-1})-\alpha+\beta$,其中 β 表示增长收益,α 表示舍弃收益,且 $\beta>\alpha$.此时,被舍弃对象 e_i 的增益为:$\Delta f(e_i|S_{k-1})=\Delta f(e_i|S_{k-1})+\delta$,而贪心替换原则使得 $\beta-\alpha>\delta$,故 h'_i 对已过滤数据 e_i 满足 $\Delta f(e_i|S'_{i-1})\leqslant\forall\Delta f(h'_i|S'_{i-1}), h'_i\in S'_i$. 证毕.

定理 3. HSSM 算法的摘要收益 $f(S_k)$对比最优解 S^* 的收益 $f(S^*)$,满足 $f(S_k)\geqslant 1/(2-1/k)\times f(S^*)$.

证明.

$$\begin{aligned} f(S_k) &\geqslant f(S^*)-f(S^*\backslash S_k)\\ &\geqslant f(S^*)-\sum_{i=2}^{k}\Delta f(e_i\mid S_{k-1})\\ &\geqslant f(S^*)-(k-1)\frac{f(S_k)}{k}\\ &\geqslant 1/(2-1/k)f(S^*), \end{aligned}$$

易得知:最优解 S^* 与摘要 S_k 的收益交集不大于 S_k 的效用收益;同时摘要 S_k 中所有不被包含的效用收益至多为定理 1 中被 HSSM 算法过滤的增益;进而通过定理 2 中第 k 层过滤对象的增益损失上界,推导出摘要 S_k 的最低效用收益. 证毕.

定理 1 表明被过滤的对象其增益不大于所选摘要 S 的平均增益;定理 2 表明,即使摘要 S 改变,已被过滤的对象增益也不会超过新摘要 S' 的平均增益;定理 3 得知,基于分层思想的次模最大化方法,在最坏情况下仍可以得到大于 $1/(2-1/k)$ 最优解的收益.

2.3 HSSM 算法的改进

进一步分析 HSSM 算法,存在 3 点可改进之处:

1) HSSM 算法中的分层模型存在收益递减特性,使得底层出现大量满足 $\Delta f(e_i|S_{i-1})<\sigma$ 的低增益对象,存在重复计算效用收益的问题;

2) HSSM 算法通过式(7)进行数据增益对比的过程中,当累积摘要集合规模(即 $|S_{i-1}|\leqslant k$)较大时,将导致效用收益计算 $f(S_{i-1})$ 的时间开销过大;

3) 摘要规模(k)较大时可能导致分层数量过多.

本节对这 3 方面不足提出进一步改进的做法.

2.3.1 通过阈值过滤低增益对象

针对效用函数收益计算的时间开销问题,HSSM 算法提出基于次模函数最大化收益递减特性的最低增益阈值过滤方法,使得各层不再将增益小于 σ 的待处理对象 e_i 传输至下一层,进而减少了低增益对象的增益计算开销.

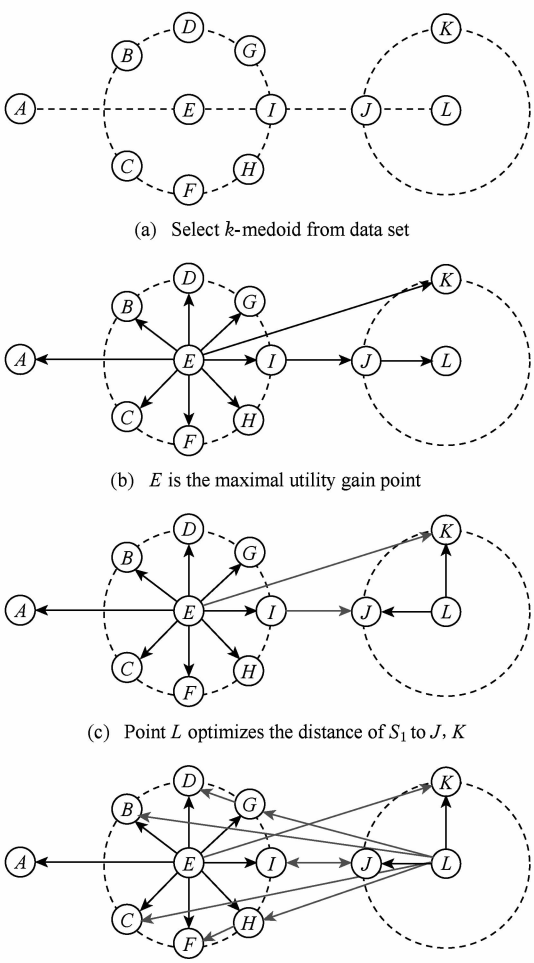
由式(1)得知,对于第 i 层的待处理对象 e_i 增益为 $\Delta f(e_i|S_{i-1})$,其在第 i 至 k 层的增益将不大于 $\Delta f(e_i|S_{i-1})$. 因此,已知底层(第 k 层)摘要维持对象 h_k 的增益 $\Delta f(h_k|S_{k-1})$ 时,若待处理数据 e_i 的增益满足:$\Delta f(e_i|S_{i-1})<\Delta f(h_k|S_{k-1})$,则该对象一定不会成为摘要维持对象. 设置过滤阈值 $\sigma=\Delta f(h_k|S_{k-1})$ 可以有效解决摘要维持对象不变时低增益数据过滤问题. 然而,基于流水并行的 HSSM 算法,其底层增益 $\Delta f(h_k|S_{k-1})$ 可能因为上层摘要维持对象的改变而变化(增大或减少),由于该变化具有 k 个单位的延迟,使得其他数据被过滤时可能导致摘要收益损失,即当 $\Delta f(h'_k|S'_{k-1})<\Delta f(h_k|S_{k-1})$ 时,e_i 增益满足 $\Delta f(e_i|S'_{i-1})\geqslant\Delta f(h'_k|S'_{k-1})$,此时过滤 e_i 将导致摘要收益降低. 因此,本文同时设置一个较小的最低增益阈值上限 μ(如 $\mu\approx 0.001f(S_k)$),该参数用于减少最低增益阈值延迟改变而导致摘要 S 效用收益损失的问题. 综上所述,最低增益阈值 σ 为

$$\sigma=\min(\mu,\Delta f(h_k|S_{k-1})).$$

2.3.2 效用向量替代累积摘要

围绕次模最大化方法中存在效用函数增益计算问题,大多数算法通过遍历累积摘要集合 S_i 与新增数据 e 的方法得到效用函数收益差值:$\Delta f(e|S_i)=f(S_i\cup\{e\})-f(S_i)$. 其中,计算效用收益 $f(S_i)$ 的集合遍历开销与摘要规模 $|S_i|$ 成正比例关系,为了进一步减少该计算开销,本文提出一种使用效用向量替换累积摘要 S_i 的信息压缩方法.

以图 3 的中心点选取问题为例,式(4)为该问题的次模效用函数. 从图 3(a)的 12 个数据点中选取 3 个中心点,使得各点到中心点的距离之和最小;初始化摘要集合为空;图 3(b)首先选取到各点距离损失最小的点 E 添加至摘要 S;图 3(c)选取基于累积摘要 S_1 的第 2 轮最大收益对象点 L;第 3 轮中,累积摘要 S_2 的效用收益为图 3(d)中点 E 与点 L 到其余各点的最近距离(同样的,对规模为 i 的累积摘要 S_i,也需要遍历 i 个数据获取最优值作为效用收益).



(a) Select k-medoid from data set

(b) E is the maximal utility gain point

(c) Point L optimizes the distance of S_1 to J, K

(d) Summarization on S_2 (utility vector) equals the minimum distance lose of point E, L

Fig. 3 k-medoid select problem with utility vector.

图 3 效用向量在中心点选取问题中的应用

次模最大化问题通常满足累积摘要在某一维度的收益单调增特性(如图3(c)实际是优化了摘要到点 J,K 的距离),因此,该类问题中摘要添加最高增益对象 h_i 与直接优化摘要的某一维度的最优效用值是等价的.本文提出对具有以下特性的效用函数进行累积摘要集合 S_i 的信息压缩,引入效用向量(utility vector)概念.

对于效用函数 $f(S)$,若存在函数族 $G=\{g_i|i=1,2,\cdots,k\}$,满足 $f(S)=\sum_{i=1}^{k}\max_{\{e\}\subseteq S}g_i(\{e\})$,且函数 g_i 的时间复杂度 $\Omega(g_i(\{e\}))=1/k\times\Omega(f(\{e\}))$,则可以通过效用向量 $\boldsymbol{U}_k$ 保存最优效用收益:

$$U_{ki}=\max\{U_{ki},g_i(\{e\})\},\tag{9}$$

其中,U_{ki} 表示第 k 层向量的第 i 维度效用值.效用向量 $\boldsymbol{U}_k$ 仅在新增摘要维持对象时完成更新,因此时间复杂度满足 $\Omega(f(\{\boldsymbol{U}_k\}))=1/k\times\Omega(f(S_k))$,而效用向量的收益 $f(\{\boldsymbol{U}_k\})$ 则满足 $f(\{\boldsymbol{U}_k\})=f(S_k)$.

同时,本文指出该类问题的函数族 G 一般可以通过归纳法得出,如1.2节中基于覆盖率的摘要问题以信息单元为维度,即 $g_i=\max_{u_j\in e}(w_u)$,而基于距离损失函数的中心点选取问题则以各点间距离为维度,即可设置 $g_i=\min(d_i(x,x'))$.

综上,对于未压缩的累积摘要增益计算 $\Delta f(S)$,其开销为 $\Omega(|S|\times|f(\{h_i\})|\times2)$.而通过效用向量的增益计算与更新开销为 $\Omega(|f(\{\boldsymbol{U}_i\})|+|f(\{\boldsymbol{U}_i\})|\times2)$,其中 $|f(\{\boldsymbol{U}_i\})|=|f(\{h_i\})|$,$|S|=k$.因此相对于原算法,效用向量在时间复杂度上到达 $2/3\times k$ 倍的提升.

2.3.3 可变的分层摘要维持对象数量

HSSM算法中每层的增益筛选过程仅保留单个摘要维持对象 h_i,故具有最低收益保证的优势.但考虑到某些场景中可能遇到需求摘要规模 k 过大,导致流水并行模式延迟过大(规模为 v 的数据摘要其时间开销为 $\Omega(v+k)$)及层数过多导致存储空间增大等问题.本文提出分层模型中每层摘要维持数量是可变的(即 $Layer_l$ 可以保留 l_i 个摘要维持对象 $\{h_{i1},h_{i2},\cdots,h_{il}\}$).该方法将摘要 S 不规则地分配在 i 层中,使得层数是可控的,但会导致并行度降低,从而减慢摘要时间.

2.3.4 HSSM+算法

改进的HSSM算法的实现如算法2所示.

算法2. HSSM+:改进的流次模最大化分层算法.

输入:数据集 O、摘要规模 k、过滤阈值 σ;

输出:摘要结果 S.

① 初始化:$\boldsymbol{U}:=\boldsymbol{0}$; /* 初始化效用向量 */
  h 是摘要维持对象的集合;
② for $i=1$ to n do {
③ $e=O_i$; /* 载入新数据 */
④ for $l=1$ to k do { /* 在每一层中筛选该数据 */
⑤ if ($\Delta f(e|\boldsymbol{U}_{l-1})<\sigma$)
⑥ break; /* 对象增益过低被过滤 */
⑦ else /* 筛选数据并将$[e,\boldsymbol{U}_l]$输入下层 */
⑧ $[e,h_l,\boldsymbol{U}_l]=Filter(e,h_l,\boldsymbol{U}_{l-1})$;}}
⑨ $S:=\varnothing$; /* 初始化摘要结果 */
⑩ for $l=1$ to k do
⑪ $S=S\cup\{h_l\}$;
⑫ return S.

其中,行①表示对分层信息的初始化操作;行②~⑧为所有数据的进行分层的增益对比,其中,行⑤是对上层增益结果是否满足阈值 σ 进行判断;行⑧是根据2.3.1节中改进Filter算法使用效用向量 $\boldsymbol{U}$ 对比待处理数据 e_i 与原摘要维持对象 h_i 增益得到新的摘要维持对象;行⑨~⑫为分层遍历汇总摘要集合.

3 分布式分层次模最大化算法及实现

根据第2.1节分析可知,HSSM+算法在不同层中数据增益对比具有可并行的特点,通过分布式处理框架实现不同分层的数据同时进行筛选,可达到提高摘要速率的目标.本节主要介绍HSSM+算法的分布式实现.

3.1 分布式HSSM+算法

HSSM+算法构建流水并行模式(图1)使得某一时刻各层可同时进行数据增益对比 Δf 操作,进一步提出Spark-HSSM+算法.相对于单对象处理模式,分布式流数据框架需解决3个问题:

1) 对于实时到来的流数据,以单一对象作为分布式任务(Job)存在资源分配与任务调度(调度中心将任务分配给空闲的Worker并等待其返回运行结果)的时间开销问题.

2) 在建立分布式任务时,解决待处理数据与分层信息结合的问题.

3) 每轮摘要任务完成后,如何实现动态更新分层信息.

针对分布式系统中以单个对象(或少量数据)为数据处理单元(如Spark中的RDD)将导致频繁的

启动任务,整个摘要过程的调度总开销甚至远大于摘要过程中效用收益的计算与对比的问题.分布式HSSM+算法将待处理数据按时间间隔 t 或数据量(最小数据规模阈值 θ)进行分批处理,该方法确保每一次任务处理的数据量尽可能满足负载均衡与流数据处理的任务实时性.

为保证所有待处理数据 O 逐一被所有层过滤或保留.本文通过递增的方式给每批新数据添加批次号(i),同时建立一个有限大小的循环队列用于存储批数据(图4步骤①):首先,将新数据存储于队列的指定单元($i\%k$);其次,Spark-HSSM+算法以时间片为单元,完成待处理批数据 v_i 的摘要任务(即 v_i 在对应层 $Layer_l$ 的增益筛选操作);最后,在下一时间片开始阶段,读入新数据 v_{i+1} 并将 $(i+1)\%k$ 位置的原数据 v_{i+1-k} 替代为该数据.该方法实现批数据 v_{i-k} 在 k 个时间单位内被有规律的存储,并通过算法3实现批次号 i 与各批次数据 v_i 及分层信息的映射.

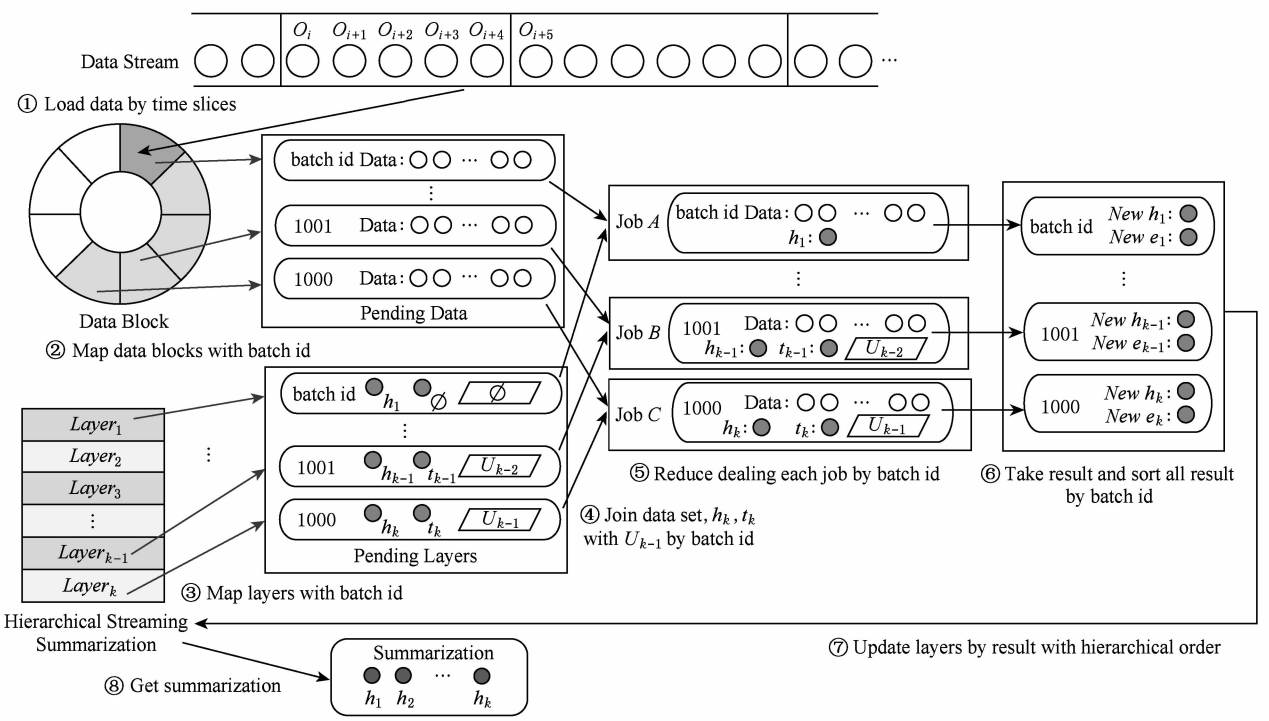


Fig. 4 Spark-HSSM+ algorithm in Spark.

图4 Spark框架实现Spark-HSSM+算法

算法3. $BatchMap$:基于批次号 i 获取循环队列数据与分层信息的映射关系.

输入:当前批次号 i、数据总批数 b、摘要大小 k;

输出:起始循环数据块索引 f_start、起始层次号 l_start、对应使用的数据块及层次数目 $length$.

① 初始化:$f\_start=0$,$l\_start=0$,$length=0$;

② if $i>(k-2)$ && $i<b$ /*循环队列满时*/

③ $f\_start=i\%k$,$l\_start=0$,$length=k$;

④ else if $i<(k-1)$ /*循环队列未满*/

⑤ $f\_start=i$,$l\_start=0$,$length=i+1$;

⑥ else /*待处理数据具有 k 个单位的摘要延迟*/

⑦ $f\_start=(b-1)\%k$,$l\_start=i-b+1$,$length=k-i+b-1$;

⑧ return $[f\_start, l\_start, length]$.

为达到分层次模最大化算法的最差解收益保证,要求每轮摘要任务结束时,各层的累积摘要(或效用向量 $\boldsymbol{U}$)更新为上层累积摘要与本层筛选的最高增益对象 h_i.通过分布式任务可完成批数据 v_i 的最优增益对象 h_i 的选取,然而该方法无法实现自顶向下的累积摘要(或效用向量)更新,本文通过对各分布式任务的结果进行集中汇总更新的方式解决该难题.由于分布式摘要结果是无序的,故更新前还需对摘要结果进行排序,以保证累积摘要是自顶向下添加实现分层次模最大化摘要.

3.2 Spark平台实现Spark-HSSM+算法

Spark[21]是基于Hadoop[22]文件存储系统HDFS与MapReduce处理模式的分布式处理平台,使用共享内存的弹性分布式数据集(RDD)技术与DAG(有向无环图)任务结构改进了Hadoop框架在

迭代任务开始与结束阶段因数据频繁进行磁盘读写操作产生开销的缺陷.

Spark-HSSM+算法通过建立如图 4 所示的分布式处理流程,利用 Spark 的 Streaming 流数据处理框架的 *fileStream*()方法将流数据通过时间间隔(例如,1 s)分批读入;并以循环队列的形式实现 3.1 节中替换存储数据至分布式内存(RDD)中.

步骤②向分布式内存申请 k 个单元存储待处理数据块($DataBlock_i$),步骤③读取图 4 中 Hierarchy Streaming Summarization 的分层摘信息,通过 *map*()方法完成摘要维持对象 h_i,上层待处理数据 t_i 及效用向量 $\boldsymbol{U}_{i-1}$的数据映射操作,最终形成具有关联批次号(batch id)的分布式待处理数据与分层信息.

图 4 中步骤④通过按批次号连接(Spark 中 *Join*()方法)各分布式任务数据(包括 $DataBlock_i$, h_i, t_i 及 $\boldsymbol{U}_{i-1}$)组成分布式任务(Job).

由于分层增益对比满足数据间的独立性,将所有任务(Job $A/B/\cdots$)通过 Spark 的 *reduceByKey*()方法分布在若干个节点上并行完成,使得所有数据分别执行增益计算,并对 batch id 相同的数据(在同一节点上)进行两两增益对比,最终仅返回最高增益对象为该层新摘要维持对象 h_i'(如图 4 步骤⑤),若发生摘要维持对象的替换时,需返回原摘要维持对象 e_{i+1},使其在下层摘要筛选过程中被访问.

效用函数通过式(7)满足次模收益最大化,因此, Spark-HSSM+算法需自顶向下的添加最大增益对象 h_i,同时完成效用向量 $\boldsymbol{U}_i$ 的更新. 步骤⑥通过 RDD 的 *take*()方法获取所有分布式摘要结果并完成排序. 步骤⑦完成新摘要维持对象 h_1 对顶层的效用向量 $\boldsymbol{U}_1'$ 更新,随后,其余层次是基于上层的新效用向量 $\boldsymbol{U}_{i-1}'$与新摘要维持对象 h_i' 完成本层累积摘要 $\boldsymbol{U}_i'$ 的更新.

步骤⑧通过实时访问分层模型合并各层摘要维持对象($h_1, h_2, \cdots, h_k$),最终得到规模为 k 的摘要结果 S_k.

算法 4. Spark-HSSM+:Spark 实现 HSSM+算法.

输入:分批数据集 O、数据总批数 b、摘要规模 k;

输出:摘要结果 S.

① 初始化: $S:=\varnothing$, $DataBlock$ 是规模为 k 的弹性分布式数据集(RDD), $layers$ 是规模为 k 的层次集合(含 h_i, t_i 及 $\boldsymbol{U}_{i-1}$);

② for $i=1$ to $(b+k-1)$ do {/* 处理所有数据获取该批次号映射的层次信息 */

③ $[f\_start, l\_start, length]=BatchMap(i, b, k)$;

④ if $i<b$ /* 读入新数据 */

⑤ 将 o_i 存入 $DataBlock_{f\_start}$ 数据块;

⑥ $j=f\_start$, $l=l\_start$; /* 待处理层次信息 */

⑦ for $c=0$ to $length$ do {/* 所有数据合并同层待处理数据与维持对象 */

⑧ $tmpBlcok_j=DataBlock_j \cup \{h_l\} \cup \{t_l\}$; /* 连接同层数据与该层效用向量 */

⑨ 基于映射关系连接 $DataBlock_j$ 与 $\boldsymbol{U}_{l-1}$;

⑩ $j=(j-1+k)\%k$, $l+=1$; /* 环形处理下层 */}

⑪ $AllData=$empty RDD; /* Reduce 任务数据合并所有待处理数据 */

⑫ for $j=f\_start$ to $(j-length+1+k)\%k$ do

⑬ $AllData=AllData \cup DataBlock_j$;

⑭ 对 $AllData$ 进行增益映射(Map)与计算;

⑮ $Result$ 为 $AllData$ 基于批次号进行 $reduceByKey$ 分布式运算本层摘要结果; /* 基于层次结构逐层更新 $layers$ */

⑯ $Update(layers, Result)$;}

⑰ for $i=1$ to k do /* 获取摘要结果 */

⑱ $S=S \cup \{h_i\}$;

⑲ return S.

完整 Spark-HSSM+算法在 Spark 分布式平台的实现如算法 4 所示. 其中,行①为存储单元与分层信息的初始操作;行②~⑯为每时刻数据的分层批数据摘要操作,其中行⑦~⑩为批数据与分层信息的连接(Join)操作,行⑪~⑮为分布式对比各层待处理数据增益;行⑰~⑲完成获取 Spark-HSSM+算法的摘要结果.

4 次模最大化相关算法的性能对比

将本文算法与常见的次模最大化算法进行分析,分别通过访问数据集次数、最低(最差解)收益保证、时间复杂度及流数据更新的时间与空间开销方面作出对比,各算法主要性能如表 1 所示:

Table 1 Comparison Between the Existing Submodular Maximization Methods

表 1 常见次模最大化算法性能对比

Algorithm	# Passes	Approximate Guarantee	Time Complexity	Update Time	Memory
Standard-Greedy[9]	k	$1-1/e$	$k^2\times n\times\Delta f$		
GreeDi[10]	1	$\leqslant 1/2$	$[k^2\times n+k^2\times(m\times k)]\times\Delta f$	$\Omega(k\times(v/m+m))$	$\Omega(k+v)$
Stream-Greedy[11]	multiple	$1/2-\varepsilon$	$k^2\times n\times\Delta f$	$\Omega(k\times v)$	$\Omega(k+v)$
Sieve-Streaming[12]	1	$1/2-\varepsilon$	$t\times n\times\log(k)/\varepsilon\times\Delta f$	$\Omega(v)$	$\Omega(\log(k)/\varepsilon\times v)$
Spark-HSSM+	1	$1/(2-1/k)$	$k\times n\times\Delta f$	$\Omega(v)$	$\Omega(k\times v)$

首先对比各算法的数据集访问次数.除标准贪心算法(Standard-Greedy)需进行 k 轮迭代访问外,其余算法均可满足流数据仅一次访问数据集的限制.

在最差解收益保证方面,Standard-Greedy 拥有最高的收益保证;Stream-Greedy 算法需多次访问数据集以维持其最差解下界,即仅访问一次数据集时,其最差解收益的下界为 $1/k\times f(S^*)$,而在多次访问数据集的条件下,其上界也不大于 $1/2*$ 最优解收益;Sieve-Streaming 算法的最差解保证则与其离散度参数 ε 相关;本文提出的 HSSM+算法在 2.2 小节中证明摘要至少达到 $1/(2-1/k)$的最优解收益.

对比各算法处理规模为 n 的数据集,其时间复杂度量级都不低于 $\Omega(n\times k^2\times\Delta f)$.但本文通过 2.3.2 节中提出效用向量压缩方法使得计算每个数据对象增益的时间开销由 $\Omega(k^2\times\Delta f)$降为 $3/2\times k\times\Delta f$.

基于文献[12]得到一个规模为 $t\leqslant k$ 的摘要集合,若存在 $t\ll k$,将大幅提升 Sieve-Streaming 算法摘要速率,故对同等收益情况下摘要 S_t^{SS} 的最小摘要规模进行分析.

定理 4. 已知规模为 $t(t\leqslant k)$的 Sieve-Streaming 算法摘要结果 S_t^{SS} 及规模为 k 的 HSSM 算法摘要结果 S_k^{HSSM},在摘要收益 $f(S_y^{SS})\geqslant f(S_k^{HSSM})$的条件下,摘要规模 t 满足:$t\geqslant(2k-1)/(3-2/k)$.

证明.

$$f(S_t^{SS})=f(S_t^*)\geqslant f(S_k^{HSSM}),$$

$$f(S_t^*)\geqslant f(S_t^{HSSM})+(k-t)\times\frac{f(S_t^*)}{t},$$

$$f(S_t^*)\geqslant\frac{1}{(2-1/k)}f(S_t^*)+(k-t)\times\frac{f(S_t^*)}{t},$$

$$t\geqslant(2k-1)/(3-2/k),$$

其中,不妨设 $f(S_t^{SS})$即为规模 t 的摘要最优解 $f(S_t^*)$,且该收益满足不低于规模 k 的 HSSM 算法收益 $f(S_k^{HSSM})$条件;同时,$f(S_k^{HSSM})$的收益大于规模为 t 的 HSSM 算法效用收益 $f(S_t^{HSSM})$与$(k-t)$个满足最低收益 $f(S_t^*)/t$ 的被舍弃数据对象收益(定理 1);进而通过定理 3,得到 $f(S_t^{HSSM})$满足至少高于 $1/(2-1/t)$的最优解收益 $f(S_t^*)$的推导结果. 证毕.

文献[12]得到最优解收益,即 $f(S_t^{SS})=f(S_t^*)$,且 HSSM 获取规模为 k 的最差摘要结果$f(S_k^{HSSM})=1/(2-1/k)f(S_k^*)$,定理 4 证明了在这种最差情况下 Sieve-Streaming 算法的规模仍满足最低规模阈值 $t\geqslant(2k-1)/(3-2/k)$.同时,第 5 节的对比实验证明,Sieve-Streaming 算法在实际应用中很难得到该理论存在的高收益且小规模的摘要集合.

对于流数据(仅通过访问增量数据 v 更新摘要 S)的摘要问题,GreeDi 算法在处理已知数据规模的情况下(不符合一般流数据使用条件),可基于不同的数据规模划分提高摘要速率;分布式 Sieve-Stream 与 Spark-HSSM+算法适用解决所有流数据摘要问题,且较其余非分布式算法具有约 k 倍的提高.其中,Spark-HSSM+算法因流水并行结构对实时的摘要结果产生 k 个时间单位的延迟,但摘要过程的处理时间仍为 $\Omega(v)$.

在空间开销方面,GreeDi 算法与 Stream-Greedy 算法仅保存规模为 k 的摘要结果与当前处理的批数据 v 即可; Sieve-Stream 算法根据其离散参数 ε 的不同,共保存 $\log k/\varepsilon$ 个过滤器,并分别为这些过滤器保存待处理数据 v;Spark-HSSM+算法将批数据 v 需保存 k 个时间单位故产生 k 倍于批数据大小的额外内存开销.

从上述理论分析中得出,本文提出的流数据分层次模最大化摘要算法 Spark-HSSM+,应用流水并行结构完成其在 Spark 分布式平台的实现,通过效用向量 $\boldsymbol{U}$ 使得次模效用函数的计算开销减少为原本的 $3/(2k)$,并以分布式任务的方式使得摘要速率比串行算法上具有 k 倍的提升,最终实现较贪心算法具有 $2/3\times k^2$ 倍的速率优化.

5 实验结果与分析

本节分别在单机环境及分布式集群上完成基于覆盖率的摘要问题与基于距离损失函数的中心点选取问题的次模最大化算法对比实验：

随机采样(Random)：对数据集 O 随机选取 k 个对象作为摘要.

Standard-Greedy(Greedy)：通过标准贪心算法[9]遍历 k 次数据集选取 k 个对象作为摘要结果.

Stream-Greedy(SG)：基于 1.3.2 节中描述的流贪心算法[11]，维护规模为 k 个对象的结果作为摘要.

GreeDi：通过 1.3.3 节中描述的算法[10]，将数据集用分布式标准贪心算法进行 m 次分割进行摘要.

Sieve-Streaming(SS)：通过流数据过滤算法[12]选取指定的离散参数 ε 实现分布式过滤数据集并选取最优解作为输出.

Spark-HSSM+算法通过 Scala 2.11.6 开发完成. 所有单机对比实验运行于具有 8 GB 内存、双核 2.4 GHz 处理器的 PC 机上，Java 环境为 JRE1.8. 集群为基于 Hadoop yarn 2.2.0 资源调度管理的 Spark1.2.0 分布式框架，包括 15 个拥有 8 GB 内存及 4 核 2.4 GHz 处理器的普通 PC 节点. 而所有分布式算法的集群运行参数统一设置：driver-memory 为 6 GB，executor-memory 为 6 GB，num-executors 为 10 个，executor-cores 为 4 个. 在本文的实验中，为保证所有摘要结果都能够达到 $1/(2-1/k)$ 的最优解收益下限，因此每层的摘要维持对象数目均为 1.

Spark 框架通过 DAG 的任务结构，使得所有数据在未执行前，不进行实际的数据分配，进而使得所有被访问的数据尽可能的存储在同一台机器上，减少不同机器间的任务开销.

摘要集合的效用收益随着摘要规模 k 的扩大而提高，本文通过增量 k 值的方法对数据集进行摘要收益对比，当效用函数值增长率低于 γ(本文选取 $\gamma=10^{-5}$)时停止扩展摘要规模. 算法的性能对比主要基于 2 个指标：1)指定规模的条件下，各算法的摘要效用收益；2)指定规模的条件下，各算法完成摘要的总时间. 相较于其他流摘要算法，随机选取策略仅访问一遍数据集且无需计算效用收益，故其时间开销是忽略不计的. 对于无法在流数据或大规模数据集上完成摘要的 Standard-Greedy 算法，本文通过一个较小的数据集实现所有算法与该算法的对比.

5.1 非分布式环境下的性能对比

本文首先对包含 50 000 个涉及 1 000 维特征(即每个对象包含一个或多个特征)的合成数据集进行基于最大覆盖率(式(3))的摘要实验. 该数据集规模较小，能够进行包括静态算法(Standard-Greedy)及非分布式算法在覆盖率与运行时间的对比. 实验中 Sieve-Streaming 算法选取 $\varepsilon=0.1$ 对最优解进行离散. 图 5、图 6 分别为各算法对该数据集在不同摘要规模 k 的效用收益(覆盖率)及运行时间对比.

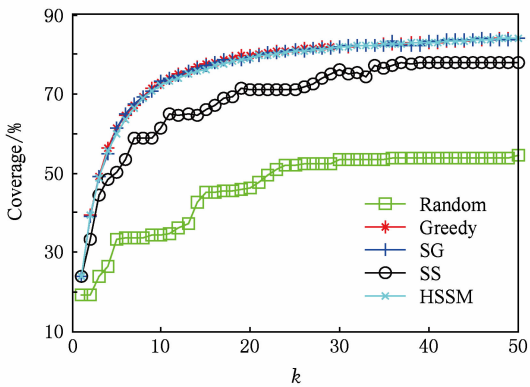


Fig. 5 The coverage of algorithms in different k.

图 5 不同规模下各算法覆盖率对比

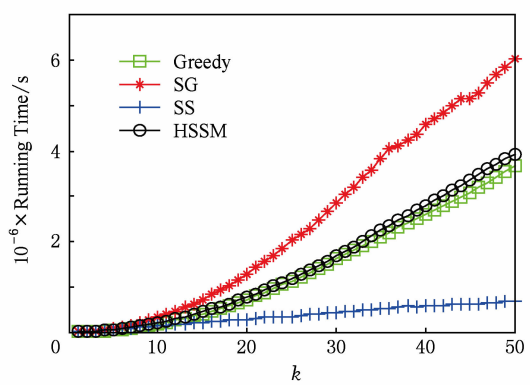


Fig. 6 The running time of algorithms in different k.

图 6 不同规模下各算法完整摘要时间对比

从图 5 中可以看出，在规模不断增加的过程中，Stream-Greedy(SG)，Standard-Greedy(Greedy)，Sieve-Stream(SS)及 HSSM 算法的收益都有次模特性，摘要的收益基本保持单调增长特性，但 Sieve-Stream 由于规模改变导致每次选取的摘要可能不同，有可能出现规模增大摘要收益反而降低的现象. 总体来说，Stream-Greedy 算法与 HSSM 算法在覆盖率收益方面都更接近于标准贪心算法 Standard-Greedy 的收益，而 Sieve-Stream 算法虽优于随机采样方法(Random)，但相比于其他算法仍有 10 个百分比左右的差距.

图 6 表明,对于同一数据集,摘要时间随规模 k 呈指数型增长. 而 Stream-Greedy(SG)算法在仅访问一遍数据集的情况下需依次计算新数据 e 的增益 $\Delta f(e|S_k\backslash\{h_i\})$与原摘要维持对象 h_i 的增益 $\Delta f(h_i|S_k\backslash\{h_i\})$,即二次访问累积摘要集合,故比标准贪心算法多近一倍的开销时间. 同时,未优化的 HSSM 算法与标准贪心算法时间开销接近;而 Sieve-Stream 运行时间则主要受限于过滤器数目及过滤器中摘要数量,本实验中 $k=50$ 时过滤器数目为 27 个,其时间开销约为标准贪心算法的 1/6.

通过 2.3 节中引入最优效用向量及最低增益过滤($\sigma=5$)改进后的 HSSM+算法在覆盖率收益与时间开销如表 2、表 3 所示. 通过对比结果发现,仅当 k 较大($k=45$ 与 $k=50$)的情况下,HSSM+算法的收益较原算法略有降低,但仍与 Stream-Greedy 算法收益接近. 而在算法的运行时间开销方面,对比 $k=50$ 的摘要规模,HSSM+比未改进的 HSSM 算法提升了约 39 倍,比 Stream-Greedy 算法要高约 60 倍,对比不同的摘要规模,HSSM+算法也能较原算法达到接近于理论(2/3×k 倍)的速率提升.

Table 2 Comparison of HSSM, HSSM+ & Stream-Greedy(SG) in Coverage

表 2 改进的 HSSM 算法(HSSM+)与朴素算法(HSSM)及流贪心算法(SG)的覆盖率对比 %

Algorithm	k									
	5	10	15	20	25	30	35	40	45	50
SG	61.20	73.37	77.07	70.98	81.09	81.74	82.39	83.37	83.80	84.13
HSSM	59.89	72.28	76.30	79.13	80.65	81.85	82.50	83.04	83.59	84.13
HSSM+	59.89	72.28	76.30	79.13	80.65	81.85	82.50	83.04	83.37	83.59

Table 3 Comparison of HSSM, HSSM+ & Stream-Greedy(SG) in Running Time

表 3 改进的 HSSM 算法(HSSM+)与朴素算法(HSSM)及流贪心算法(SG)的运行时间对比 s

Algorithm	k									
	5	10	15	20	25	30	35	40	45	50
SG	72 574	324 560	718 510	1 291 662	2 026 154	2 842 168	3 829 363	4 592 976	5 148 913	6 049 288
HSSM	59 608	210 537	448 187	768 497	1 175 349	1 671 399	2 241 066	2 793 001	3 359 083	3 934 525
HSSM+	9 873	22 465	32 743	45 205	54 398	65 603	78 839	87 211	98 161	109 233

5.2 基于距离损失函数的分布式中心点选取

通过对 CensS1990 数据集[23]采用基于距离损失式(5)的效用函数进行中心点选取实验,该数据集生成包含 2 458 285 个 68 维特征的$|O|\times|O|$距离矩阵数据,非流算法对于该数据集的多次访问需数天时间才能得到选取结果,已不符合一般问题的解决时效条件. 文献[10]中证明该问题的摘要方法是增量可分的(additively decomposable),故实验使用从原数据集中采样$|W|=1/100|O|$进行流算法的对比实验. 由于文件大小的限制,本实验将该数据集以 500 个对象为一批次进行分割,而该数据量过小使得 Spark-HSSM+算法不能够达到分布式最高运行效率(分布式任务调度开销过大),但我们仍进行对比以证明优化后的 Spark-HSSM+算法在分布式环境下运行效率得到提升. 本实验设置过滤阈值 $\mu=20$,Sieve-Stream 算法选取 $\varepsilon=0.1$ 离散化最优解,分布式中共包含 72 个过滤器. GreeDi 不适合处理流数据,故仅用其作增量式(每批数据保存贪心摘要结果用于下一批次比较)贪心算法对比.

图 7 显示在规模 $k=50$ 的情况下,GreeDi, Stream-Greedy 与 Spark-HSSM+算法的效用收益相差不大,而 Sieve-Stream 算法则遇到过滤器最优解估计不适的情况,其效用收益值甚至低于随机采样.

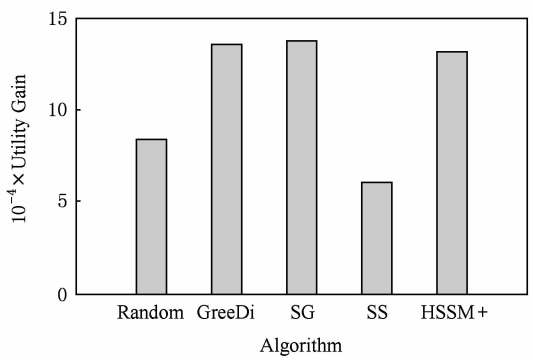


Fig. 7 The utility of different algorithms in $k=50$.

图 7 $k=50$ 时效用函数值对比

图 8 表明 Sieve-Stream 算法与 Spark-HSSM+算法的运行速率要明显优于 Stream-Greedy 与 GreeDi 算法,因此更适合处理需要更高效响应的流数据,其中,Spark-HSSM+算法较 Stream-Greedy 算法仅提高了 29 倍,未达到理论值提升的主要原因是数据规模过小不适合分布式任务且包含任务调度与资源分配同时产生额外开销,故该算法在处理小批量数据时,优化效果不明显.

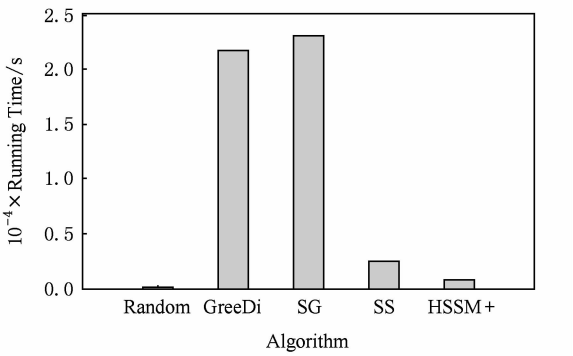


Fig. 8 The running time of different algorithms in $k=50$.

图 8 $k=50$ 时各算法完整摘要时间对比

5.3 分布式环境下基于覆盖率的数据摘要

由于 5.2 节中实验未体现大规模流数据摘要的特点(每批次数据量过少),因此,本节选取 Yahoo! 提供的 28 041 015 条用户首页访问日志数据集[24]以 280 000 条数据进行分批,同样以基于最大覆盖率的效用函数进行实验.

该数据集包含 126 维用于描述用户访问时的行为特征,根据文献[1]中提出的关联规则对其中最频繁的 50 维特征的频繁二项集进行最大覆盖率摘要实验.实验选取 $k=30$,Spark-HSSM+算法选取过滤阈值上界 $\mu=20$,Sieve-Stream 算法设置离散系数 $\varepsilon=0.01$(该离散值使其效用收益更加接近于

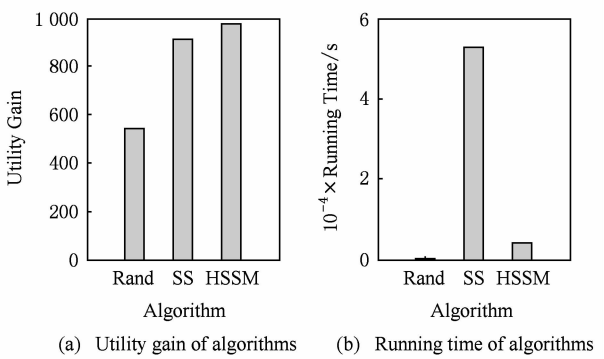


Fig. 9 Comparison of utility & running time on Yahoo! dataset.

图 9 Yahoo! 数据集的效用收益与运行时间对比

Spark-HSSM+算法). 图 9 为随机采样、SS 算法及 Spark-HSSM+算法的摘要时间及其效用收益对比.

图 9 表明 Spark-HSSM+算法在可接受响应时间内,较 Sieve-Stream 算法与随机选取得到更优的效用收益.同时其在运行速率方面也较 Sieve-Stream 算法有 10 倍左右的提升.除上述 3 个算法外,其余算法均无法在 24 h 内给出响应结果,对于流摘要的代表算法 Stream-Greedy,本文通过其完成 1%的数据处理量耗时预估出其时间开销约为本文算法的 500 倍(理论值为 $2/3\times k^2=600$ 倍,但应除去分布式任务的启动与资源调度的时间开销).故大规模流数据摘要算法 Spark-HSSM+,在具有高效用收益的同时,其速率提升也是显著的.

6 结 论

本文提出了基于次模最大化的流数据分层摘要算法.该算法在分布式环境下仅一次访问数据集即可完成选取大小为 k 的摘要集合,其摘要的效用函数收益至少能达到 $1/(2-1/k)$的最优解.实验表明 Spark-HSSM+算法在处理大规模数据集时能够在达到近似于标准贪心算法收益(次模最大化问题中最好的解决方案)的同时,相比于其他算法在实时性上具有优势,其速率在理论较标准贪心算法提升约为 $2/3\times k^2$ 倍.本文的贡献旨在对大规模数据及流数据集在次模最大化收益问题及运行效率方面得到改进.

参 考 文 献

[1] Xu J, Kalashnikov D V, Mehrotra S. Efficient summarization framework for multi-attribute uncertain data [C] //Proc of the 33rd ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2014: 421-432

[2] Wang Zhenhua, Shou Lidan, Chen Ke, et al. On summarization and timeline generation for evolutionary tweet streams [J]. IEEE Trans on Knowledge and Data Engineering, 2014, 27(5): 1301-1315

[3] Sipos R, Swaminathan A, Shivaswamy P, et al. Temporal corpus summarization using submodular word coverage [C] // Proc of the 21st ACM Int Conf on Information and Knowledge Management. New York: ACM, 2012: 754-763

[4] Shou L, Wang Z, Chen K, et al. Sumblr: Continuous summarization of evolving tweet streams [C] //Proc of the 36th Int ACM SIGIR Conf on Research and Development in Information Retrieval. New York: ACM, 2013: 533-542

[5] Barzilay R, Mckeown K R. Sentence fusion for multidocument news summarization [J]. Computational Linguistics, 2005, 31(3): 297-328

[6] Shi L, Ola S. Approximating *k*-median via pseudo-approximation [C] //Proc of the 45th Annual ACM Symp on Theory of Computing. New York: ACM, 2013: 901-910

[7] El-Arini K, Veda G, Shahaf D, et al. Turning down the noise in the blogosphere [C] //Proc of the 15th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2009: 289-298

[8] Chen Hao, Wang Yitong. Threshold-based heuristic algorithm for influence maximization [J]. Journal of Computer Research and Development, 2012, 49(10): 2181-2188 (in Chinese)
(陈浩,王轶彤. 基于阈值的社交网络影响力最大化算法[J]. 计算机研究与发展, 2012, 49(10): 2181-2188)

[9] Nemhauser G L Wolsey L A, Fisher M L. An analysis of approximations for maximizing submodular set functions—I [J]. Mathematical Programming, 1978, 14(1): 265-294

[10] Mirzasoleiman B, Karbasi A, Sarkar R, et al. Distributed submodular maximization: Identifying representative elements in massive data [J]. Advances In Neural Information Processing Systems, 2013, 87(1): 382-384

[11] Gomes R, Krause A. Budgeted nonparametric learning from data streams [C] //Proc of the Int Conf on Machine Learning. Haifa: ICML, 2010: 391-398

[12] Badanidiyuru A, Mirzasoleiman B, Karbasi A, et al. Streaming submodular maximization: Massive data summarization on the fly [C] //Proc of the 20th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2014: 671-680

[13] Vee E, Srivastava U, Shanmugasundaram J, et al. Efficient computation of diverse query results [C]//Proc of the 24th IEEE Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2008: 228-236

[14] Mishra D, Hiranwal S. Analysis & implementation of item based collaboration filtering using K-Medoid [C] //Proc of 2014 Int Conf on Advances in Engineering and Technology Research (ICAETR). Piscataway, NJ: IEEE, 2014: 1-5

[15] Carvalho F, Melo F, Lechevallier Y. A multi-view relational fuzzy c-medoid vectors clustering algorithm [J]. Neurocomputing, 2015, 163(c): 115-123

[16] Minoux M. Accelerated greedy algorithms for maximizing submodular set functions [J]. Lecture Notes in Control and Information Sciences, 1978, (7): 234-243

[17] Liu Yong, Gao Hong, Li Jianzhong. Top-*K* graph patterns mining based on some joint significance measure [J]. // Chinese Journal of Computers, 2010, 33(2): 215-230 (in Chinese)
(刘勇,高宏,李建中. 基于联合意义度量的 Top-*K* 图模式挖掘[J]. 计算机学报, 2010, 33(2): 215-230)

[18] Yan Y, Zhang J, Huang B, et al. Distributed outlier detection using compressive sensing [C] //Proc of the 2015 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2015: 3-16

[19] Kumar R, Moseley B, Vassilvitskii S, et al. Fast greedy algorithms in MapReduce and streaming [J]. ACM Trans on Parallel Computing, 2015, 2(3): 14:1-14:22

[20] Beltrame G, Brandolese C, Fornaciari W, et al. An assembly-level execution-time model for pipelined architectures [C] //Proc of the 2001 IEEE/ACM Int Conf on Computer-Aided Design. Piscataway, NJ: IEEE, 2001: 195-200

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

[22] Apache. Hadoop [OL]. [2009-03-06]. http://hadoop.apache.org

[23] Christopher M, Bo T, David H. The learning-curve sampling method applied to model-based clustering [J]. The Journal of Machine Learning Research, 2002, 2(3): 397-418

[24] Li Lihong, Chu Wei, Langford J, et al. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms [C] //Proc of the 4th ACM Int Conf on Web Search and Data Mining (WSDM'11). New York: ACM, 2011: 297-306

Zhang Fenxiang, born in 1989. Master candidate in the Faculty of Electrical Engineering and Computer Science, Ningbo University. His main research interests include stream data processing and cloud computing.

Chen Huahui, born in 1964. PhD, professor. Member of China Computer Federation. His main research interests are in the areas of data streams and massive data processing (chenhuahui@nbu.edu.cn).

Qian Jiangbo, born in 1974. PhD, professor. Member of China Computer Federation. His main research interests include database management, streaming data processing, multidimensional indexing, and query optimization (qianjb@163.com).

Dong Yihong, born in 1969. PhD, professor and master instructor in the Faculty of Electrical Engineering and Computer Science, Ningbo University. His main research interests include big data, data mining and artificial intelligence (dongyihong@nbu.edu.cn).