计算机科学
COMPUTER SCIENCE

学习型过滤器综述

李猛, 戴海鹏, 眭永熙, 顾荣, 陈贵海

引用本文

李猛, 戴海鹏, 眭永熙, 顾荣, 陈贵海. 学习型过滤器综述[J]. 计算机科学, 2024, 51(1): 41-49.

LI Meng, DAI Haipeng, SUI Yongxi, GU Rong, CHEN Guihai. Survey of Learning-based Filters[J]. Computer Science, 2024, 51(1): 41-49.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.231000202

学习型过滤器综述

李 猛 戴海鹏 眭永熙 顾 荣 陈贵海
计算机软件新技术国家重点实验室(南京大学) 南京 210023
(meng@nju.edu.cn)

摘 要 作为一种高效的概率性结构,过滤器可以高效地解决近似集合成员查询问题。近年来,随着机器学习技术的发展,一些学习型过滤器表现出色,超越了传统的过滤器。这些学习型过滤器考虑数据分布信息,将集合成员查询问题视为二分类问题,实现了超越传统过滤器的性能。受此启发,学习型过滤器研究领域迅速发展,出现了多个变种。然而,目前还缺乏对近些年相关工作的系统性回顾和比较。为了填补上述空缺,文中全面回顾了近年来的学习型过滤器相关工作,并展望了未来的发展方向。

关键词:近似成员资格查询;机器学习;Bloom过滤器;学习型过滤器;假阳率
中图分类号 TP391

Survey of Learning-based Filters

LI Meng,DAI Haipeng,SUI Yongxi,GU Rong and CHEN Guihai
State Key Laboratory for Novel Software Technology(Nanjing University),Nanjing 210023,China

Abstract As a space-efficient probability structure,filters can efficiently solve approximate set membership queries. In recent years,with the development of machine learning technology,some learning-based filters have exceeded traditional filters in performance. These learning-based filters propose to consider data distribution information and treat set membership queries as a binary classification problem,achieving superior performance compared to traditional filters. Inspired by this,the research field of learning-based filters has progressed rapidly,and several variants have emerged. However,there is still a lack of a systematic review and comparison of recent related work. In order to fill this gap,this paper comprehensively reviews recent related work on learning-based filters,analyzes their structure design and theoretical analysis,and predicts the future development direction.

Keywords Approximate membership query,Machine learning,Bloom filter,Learning-based filter,False positive rate

1 引言

过滤器是一种高效的概率性数据结构,用于解决近似集合成员查询问题,可以用于去除重复或无用的信息。它可以快速地判断一个元素是否已经存在于集合中,从而避免重复添加和删除操作,提高了数据的处理效率。最经典的Bloom过滤器的基本原理是将数据映射到一个二进制向量中,然后将这个向量存储在内存中。当需要判断一个元素是否已经存在于集合中时,只需要检查这个元素对应的二进制向量中的对应位是否为1即可。如果为1,则说明该元素已经在集合中;如果为0,则说明该元素不在集合中。

过滤器的特点是可以快速地判断一个元素是否已经存在于集合中,从而避免重复添加和删除操作,提高了数据的处理效率。布隆过滤器的应用场景非常广泛,例如在网络爬虫[1]、数据挖掘[2-6]、搜索引擎[7]、流量转发[8-10]、大数据系统[11-15]、工业互联网[16]、可信网络[17-18]等领域中,都可以使用布隆过滤器来避免重复爬取和处理数据。例如,在网络爬虫中,Bloom过滤器可以用于避免重复爬取同一个页面。

当爬虫发现一个页面已经被爬取过时,它可以将这个页面的URL添加到布隆过滤器中,然后在下次爬取时检查这个URL是否已经存在于布隆过滤器中。如果存在,则说明这个页面已经被爬取过,不需要再次爬取。这样可以避免重复爬取同一个页面或网站,提高爬虫的爬取效率。

近年来,随着机器学习技术的发展,衍生出了一些基于机器学习技术的过滤器的设计。为了表述方便,我们将这些过滤器统称为学习型过滤器。这些学习型过滤器遵循一个基础的设计思想,也就是将集合元素查询问题视为一个二分类问题,即对于输入y、输出y是否在集合S中。在此基础上,就可以训练一个二分类器用于回答某个给定的元素是否在集合内。基于上述基本想法,衍生出了一系列的学习型过滤器设计方案。然而,目前还缺少文献系统性回顾现有的学习型过滤器的相关工作。因此,本文着重关注近年来的学习型过滤器的设计,回顾它们的结构设计以及理论分析,为未来的研究工作提供全面的参考。表1列出了目前全部的学习型过滤器。

到稿日期:2023-10-27 返修日期:2023-12-20
基金项目:国家自然科学基金(62272223)
This work was supported by the National Natural Science Foundation of China(62272223).
通信作者:戴海鹏(haipengdai@nju.edu.cn)

表 1 学习型过滤器列表

Table 1 Learning-based filters

过滤器	是否支持更新	是否有 FNR	查询延迟	构造延迟
LBF[19]	否	无	高	高
Sandwiched LBF[20]	否	无	高	高
Ada-BF[21]	否	无	高	高
Partitioned LBF[22]	否	无	高	高
Stable LBF[23]	是	是	高	高
HABF[24]	否	无	低	低
Stacked Filter[25]	是	低	低	低

本文第 2 章介绍了学习型过滤器的研究背景；然后根据不同学习过滤器的基本原理将其分为基于分类器的学习型过滤器和基于结构设计的学习型过滤器两类；第 3 章介绍了最新的基于分类器的学习型过滤器；第 4 章介绍了最新的基于结构设计的学习型过滤器；最后总结全文并展望未来。

2 学习型过滤器研究背景概述

布隆(Bloom)过滤器和布谷鸟(Cuckoo)过滤器都是一种基于概率的数据结构，用于解决集合成员查询问题，即判定某个元素是否在给定的某个集合内。经典的过滤器包括 Bloom[26]过滤器及其变种[7,11,27-33]。

Bloom 过滤器是一种基于概率的数据结构，用于解决集合成员查询问题。它可以将大量的元素映射到一个较小的二进制向量中，从而实现快速的集合成员查询。Bloom 过滤器的核心思想是将元素映射到一个二进制向量中的多个位置，并将这些位置的值设置为 1。当查询一个元素是否在集合中时，Bloom 过滤器会检查该元素对应的多个位置的值是否都为 1，从而实现近似结果。当查询一个元素是否在集合中时，Bloom 过滤器会检查该元素对应的多个位置的值是否都为 1，如果都为 1，则该元素可能在集合中；如果有一个位置的值为 0，则该元素一定不在集合中。

然而，这些过滤器的基本原理是利用随机性均匀分散元素，从而快速检测元素的存在性，但是忽略了元素的分布信息。为了更好地利用数据分布信息，学习型过滤器被提出。

3 基于分类器的学习型过滤器

3.1 学习型 Bloom 过滤器

Learned Bloom Filter(LBF)首先由 Kraska 等[19]于 2018 年提出，他们观察到所有的存在性索引结构都能被机器学习模型取代。LBF 从历史查询中抽取非键值数据集，与键值数据集 K 合成标注数据集 $D=\{(x_i,y_i=1)\mid x_i\in K\}\cup\{(x_i,y_i=0)\mid x_i\in N\}$，在数据集 D 上训练一个二分类器模型，并使用 sigmoid 函数激活。该模型可以看作一个函数 $f:U\rightarrow(0,1)$，其中 U 是所有元素的全集。当查询一个元素 x 时，$f(x)$ 的值代表元素 $x\in K$ 的概率。通过最小化损失函数 $L=\sum_{(x,y)\in D} y\log f(x)+(1-y)\log(1-f(x))$[19]，该模型可以将键值和非键值区分开。

但由于 Bloom 过滤器保证了没有假阴性，因此，简单地用二分类器替换 Bloom 过滤器是不行的，但可以使用辅助结构来消除假阴性。为了消除假阴性，LBF 提出了两种数据结构，分别将模型用于前置过滤器和哈希函数。

3.1.1 模型用于前置过滤器

选择一个阈值 τ，模型将 K 划分成两部分：$K_\tau^-=\{f(x)<\tau\mid x\in K\}$ 和 $K_\tau^+=K-K_\tau^-$，其中 K_τ^- 是模型 f 的假阴性集合，LBF 使用 K_τ^- 构造一个后置 Bloom 过滤器，用于消除这些假阴性。如图 1 所示，当查询元素 x 时，如果 $f(x)\geqslant\tau$，则 LBF 输出阳性；否则继续查询后置过滤器。

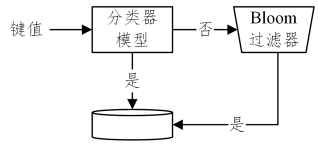

图 1 模型用于前置过滤器

Fig. 1 Model as pre-filter

令 $FPR_\tau=\sum_{x\in\tilde{\mu}}\mathbb{I}(f(x)>\tau)/|\tilde{\mu}|$ 表示模型 f 的假阳性率，其中 $\tilde{\mu}$ 是测试集中的非键值集合。令 FPR_B 表示后置 Bloom 过滤器的假阳性率，则 LBF 的假阳性率可以表示为[19]：

$$FPR_O=FPR_\tau+(1-FPR_\tau)FPR_B \tag{1}$$

由于后置 Bloom 过滤器需要表示的集合 K_τ^- 非常小，因此在相同的目标假阳性率下，该结构可以有效地减少 Bloom 过滤器的大小。具体地，设 $|K|=m$，$FNR=|K_\tau^-|/m$，$|f|=\zeta$，其中 FNR 表示模型的假阴性率。后置 Bloom 过滤器只需要表示 $mFNR$ 个元素，设其内存预算为 bm 比特，则 $FPR_B=\alpha^{\frac{b}{FNR}}$，其中 α 是一个常数(对于最优的 Bloom 过滤器，$\alpha\approx0.6185$)。对于相同大小的 Bloom 过滤器，只需要使 $FPR_\tau+(1-FPR_\tau)\alpha^{\frac{b}{FNR}}\leqslant\alpha^{\frac{\zeta}{m}}$，即[20]：

$$\zeta/m\leqslant\log_\alpha(FPR_\tau+(1-FPR_\tau)\alpha^{\frac{b}{FNR}})-b \tag{2}$$

通过调整参数 τ 和 b，可以得到满足给定假阳性率和不等式约束的 LBF。

3.1.2 模型用于哈希函数

LBF 希望学习一个哈希函数 $d:U\rightarrow\{1,\cdots,s\}$，满足键值与键值、非键值与非键值之间的碰撞最大，且键值和非键值之间的碰撞最小。由于模型 f 可以将键值集中地映射到(0,1)区间的右端，将非键值集中地映射到(0,1)区间的左端，因此令 $d=\lfloor f(x)\times s\rfloor$ 就可以学习出这样的哈希函数。

如图 2 所示，LBF 由一个大小为 $s=cm$ 的位图 M 和一个 Bloom 过滤器 B 组成，当插入元素 $x\in K$ 时，令 $M[d(x)]=1$，并将 x 插入 B 中。查询时，只有当 M 和 B 都返回阳性，结果才为阳性，否则为阴性。

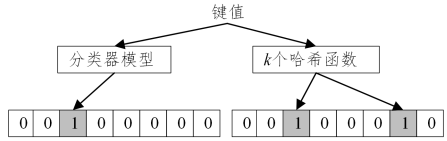

图 2 模型作为哈希函数

Fig. 2 Model as has function

令 M 的假阳性率 $FPR_M=\sum_{x\in\tilde{\mu}}M[d(x)]/|\tilde{\mu}|$，B 的假阳性率为 FPR_B，则 LBF 的假阳性率为：

$$FPR_O=FPR_M\times FPR_B \tag{3}$$

类似地，为了在相同大小下使 LBF 比 Bloom 过滤器有

更小的假阳性率，需要满足 $FPR_M \leqslant \alpha^c$。通过调整参数 c 和 b，可以得到给定假阳性率。

3.2 Sandwiched LBF

Mitzenmacher[20]在 LBF 的基础上提出了一种新的结构，在 LBF 的上层再加一个 Bloom 过滤器，如图 3 所示。当插入 $x \in K$ 时，首先将 x 插入初始过滤器中，如果 $f(x)<\tau$，再将 x 插入到后置过滤器中。当查询 $x \in U$ 时，如果初始过滤器返回阴性，则结果为阴性；否则和 LBF 一样，如果 $f(x) \geqslant \tau$，则结果为阳性，否则查询后置过滤器。

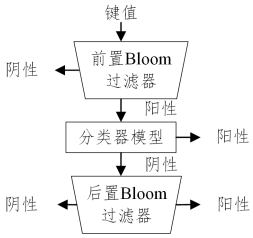

图 3　Sandwiched LBF

Fig. 3　Sandwiched LBF

由于初始过滤器已经筛选掉大部分的阴性，因此能达到后置过滤器的阴性非常少，从而允许后置过滤器用更大的假阳性率换取更小的空间。Sandwiched LBF 有非常好的理论保障，通过最小化假阳性率可以得到最优的前后过滤器大小。

与 LBF 类似，设 Bloom 过滤器总的空间预算为 bm 比特，其中前置过滤器和后置过滤器分别占 $b_1 m$ 和 $b_2 m$ 比特。令模型 f 的假阳性率为 FPR_f，假阴性率为 FNR，则后置过滤器的假阳性率为 $\alpha^{\frac{b_2}{FNR}}$，前置过滤器的假阳性率为 α^{b_1}，因此 Sandwiched LBF 的假阳性率为[20]：

$$\begin{aligned} FPR_O &= \alpha^{b_1}(FPR_f + (1-FPR_f)\alpha^{\frac{b_2}{FNR}}) \\ &= FPR_f \alpha^{b_1} + (1-FPR_f)\alpha^{\frac{b}{FNR}} \times \alpha^{b_1(1-\frac{1}{FNR})} \end{aligned} \tag{4}$$

对 b_1 求偏导，并令导数为 0，可得[20]：

$$b_2^* = FNR \times \log_\alpha \frac{FPR_f}{(1-FPR_f)(1/FNR-1)} \tag{5}$$

其中，b_2^* 是 FPR_O 取最小值时的 b_2 的取值，有趣的是，该值与总的空间预算 b 无关，当模型 f 和阈值 τ 给定时，b_2^* 是一个常数。当空间预算变大时，只需要相应地增大前置过滤器的大小，即 $b_1^* = b - b_2^*$。

将 b_1^*，b_2^* 代入式(4)，并令其不大于相同大小 Bloom 过滤器的假阳性率 $\alpha^{b+\frac{\zeta}{m}}$，可得约束[20]：

$$\zeta/m \leqslant \log_\alpha \frac{FPR_f}{1-FNR} - FNR \log_\alpha \frac{FPR_f}{(1-FPR_f)(1/FNR-1)} \tag{6}$$

不等式约束(6)与 b 具体的取值无关，只与模型 f 有关，因此可以通过调节参数 τ 得到满足约束的模型 f，再通过调节参数 b 使 Sandwiched LBF 的假阳性率满足要求。

3.3 Ada-BF

LBF 和 Sandwiched LBF 都训练模型 f 区分键值和非键值，然而它们都只是将 $f(x)$ 和单个阈值 τ 进行比较，而没有利用 f 的值域 $R(f)$ 带有的语义：假设存在 $\tau \gg f(x_1) \gg f(x_2)$，那么 x_1 和 x_2 都会被认为是阴性，即使 x_1 相比 x_2 有更大的可能性是阳性。此外，对于键值而言，f 的分布密度函数随 $f(x)$ 呈递增趋势，对于非键值呈现递减趋势。基于以上两点考虑，Dai 等[21]提出了 Ada-BF，他们通过选取 $g+1$ 个阈值将 $R(f)$ 分成了 g 个区域。Ada-BF 设计了两种结构，如图 4 所示。图 4(a)给出了第一种结构，对于映射在不同区域的元素，Ada-BF 使用不同个数的哈希函数；图 4(b)给出了第二种结构，对于映射在不同区域的元素，Ada-BF 分配不同大小的 Bloom 过滤器。

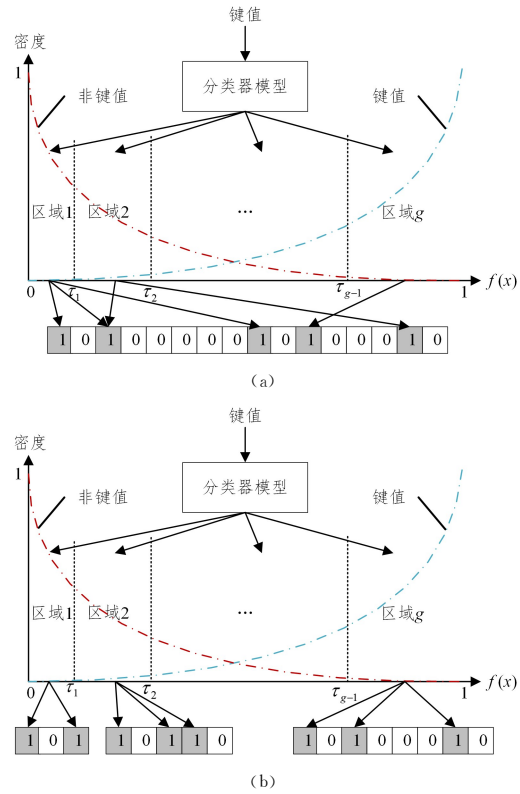

图 4　Ada-BF

Fig. 4　Ada-BF

3.3.1 使用不同数量的哈希函数

令选取的 $g+1$ 个阈值为 $0=\tau_0<\tau_1<\cdots<\tau_{g-1}<\tau_g=1$，如果元素 x 属于区域 j，则 x 满足 $f(x) \in [\tau_{j-1}, \tau_j)$，$j \in \{1, 2, \cdots, g\}$。当插入属于区域 j 的元素 $x \in K$ 时，使用 k_j 个独立哈希函数将 x 插入到 Bloom 过滤器中。查询时，先计算 $f(x)$，根据 x 所在的区域 j，使用 k_j 个哈希函数查询。因此，区域 j 的假阳性率可以表示为[21]：

$$FPR_j = \left(1-\left(1-\frac{1}{R}\right)^{\sum_{t=1}^{g} m_t k_t}\right)^{k_j} = \mu^{k_j} \tag{7}$$

其中，R 是 Bloom 过滤器的比特数，是 $m_t = \sum_{t=1}^{g} \mathbb{I}(\tau_{t-1} \leqslant f(x_i \mid x_i \in K) < \tau_t)$ 映射在区域 t 中的键值数量。Ada-BF 总的假阳性率可以表示为 $FPR_O = \sum_{j=1}^{g} p_j \mu^{k_j}$，其中 p_j 是非键值映射到区域 j 的概率，n_j 是映射在区域 j 上的非键值个数。由于 $\sum_{j=1}^{g} n_j \mu^{k_j} = O(\max(n_j \mu^{k_j}))$，因此当所有的 $n_j \mu^{k_j}$ 差不多大时，FPR_O 取得最小值。

因为 μ^{k_j} 随着 k_j 的减小呈指数级地增大，为了使所有 $n_j \mu^{k_j}$ 差不多，也需要指数级地减小 n_j。又因为 n_j 呈递减趋势，因此

随着 j 的增长，需要线性地减小 k_j 并指数级地减小 n_j。令 $k_j - k_{j+1}=1$，$n_j/n_{j+1}=c$，$k_g=0$，Ada-BF 将原来 $2g-1$ 个参数简化为了 c 和 k_1 两个参数。Dai 等证明，当 $p_j/p_{j+1}\geqslant c>1$，$m_{j+1}>m_j$，且存在 λ 使得 $c\mu\geqslant 1+\lambda$ 成立，当 g 足够大且不大于 $\lfloor 2k \rfloor$ 时，Ada-BF 有比 LBF 更小的假阳性率[21]。

因此给定假阳性率要求，可以通过调整 c 和 k_1 的值达到要求的假阳性率，并且根据 $R(f)$ 的分布确定 $g-1$ 个阈值。

3.3.2 分配不同大小的 Bloom 过滤器

与第一种结构不同的是，对于映射到区域 j 的键值 x，Ada-BF 将 x 插入到大小为 R_j 比特的 Bloom 过滤器中，且 $R=\sum_{j=1}^{g}R_j$ 是总的空间预算。当查询 x 时，根据 $f(x)$ 确定 x 所在的 Bloom 过滤器，并在该过滤器上查询，因此 $FPR_j=\alpha^{R_j/m_j}$（当 $k_j=R_j/m_j\log 2$ 时，$\alpha\approx 0.618$）。与之前的分析类似，令 $n_j/n_{j+1}=c$，则为了使 FPR_O 取得最小值，R_j 需要满足[21]：

$$n_j\alpha^{\frac{R_j}{m_j}}=n_1\alpha^{\frac{R_1}{m_1}}\Leftrightarrow\frac{R_j}{m_j}-\frac{R_1}{m_1}=(j-1)\log c/\log\mu \tag{8}$$

Dai 等证明，当 $p_j/p_{j+1}=c>1$，$m_{j+1}>m_j$，且 g 较大时，为了达到相同的假阳性率，Ada-BF 需比 LBF 使用更小的空间[21]。

3.4 Partitioned LBF

尽管 Ada-BF 已经考虑到模型 f 的值域 $R(f)$ 所具有的语义，并对 $R(f)$ 分组，但是 Ada-BF 的分组不是最优的。Vaidya 等[22]通过将分组描述为一个优化问题，使用动态规划求解出最优的划分，提出了 PLBF（Partitioned Learned Bloom Filter）。

PLBF 使用与 2.3.2 节中类似的结构，即将 $R(f)$ 划分成几组，对于映射到不同分组上的键值，PLBF 使用不同比特数的 Bloom 过滤器来表示。对于非键值分布密度低的区域，PLBF 倾向于分配更小的空间，而对于非键值分布密度高的区域，PLBF 倾向于分配更大的空间。

令 $G(t)$ 表示键值集合 K 中，使 $f(x)<t$ 的 x 的比例，$g(t)$ 是 $G(t)$ 的密度函数；相应地，$H(t)$ 表示按分布 D 抽样的非键值集合 N 中，使 $f(x)<t$ 的 x 的比例，$h(t)$ 是 $H(t)$ 的密度函数。给定预期的假阳性率 F 和分组数 k，PLBF 的目标是最小化总空间大小[22]：

$$\min_{t_i,f_i}\left(\sum_{i=1}^{k}|K|\times(G(t_i)-G(t_{i-1}))\times c\log_2\frac{1}{f_i}\right)+\zeta \tag{9}$$

$$\text{s.t.}\ \sum_{i=1}^{k}(H(t_i)-H(t_{i-1}))\times f_i\leqslant F \tag{10}$$

$$f_i\leqslant 1,i=1,\cdots,k \tag{11}$$

$$(t_i-t_{i-1})\geqslant 0,i=1,\cdots,k;t_0=0;t_k=1 \tag{12}$$

其中，$0=t_0\leqslant t_1\leqslant\cdots\leqslant t_{k-1}\leqslant t_k=1$ 是分组的阈值，f_i 是分组 i 对应的 Bloom 过滤器的假阳性率，常数 c 取决于 Bloom 过滤器的实现。

为了求解该优化问题，首先需要求解放松后的问题。去掉约束(11)，对放松后的问题使用 KKT 条件求解，得到该放松问题取最小值时 $f_i=F(G(t_i)-G(t_{i-1}))/(H(t_i)-H(t_{i-1}))$，并代入式(9)，可以得到[22]：

$$\text{Min. Term}=\sum_{i=1}^{k}|K|(G(t_i)-G(t_{i-1}))c\log_2\frac{1}{F}-$$

$$c|K|D_{\mathrm{KL}}(\hat{g}(t),\hat{h}(t)) \tag{13}$$

其中，$\hat{g}(t)$ 表示 $G(t_i)-G(t_{i-1})$，$i=1,\cdots,k$，$\hat{h}(t)$ 表示 $H(t_i)-H(t_{i-1})$，$i=1,\cdots,k$，D_{KL} 表示标准 KL 距离。为了使(13)式最小，就需要最大化 D_{KL}，PLBF 使用动态规划来求解使 D_{KL} 最大的阈值。先将区间 $[0,1]$ 等分为 N 份，令 $DP_{\mathrm{KL}}(n,j)$ 表示将前 n 个微分划分为 j 个区域可以得到的最大的 KL 距离，则 DP 的递推式为[22]：

$$DP_{\mathrm{KL}}(n,j)=\max_i\left(DP_{\mathrm{KL}}(n-i,j-1)+\left(\sum_{r=i}^{n}g(r)\times\log_2\left(\frac{\sum_{r=i}^{n}g(r)}{\sum_{r=i}^{n}h(r)}\right)\right)\right) \tag{14}$$

$DP_{\mathrm{KL}}(N,k)$ 就是所求的最大 KL 距离，算法的时间复杂度为 $O(N^2k)$。在该最优划分的基础上，可以求到原问题的最优解。

由于放松问题去掉了 $f_i\leqslant 1$ 的约束，因此可能存在 $f_i>1$ 的区域，令这些 $f_i=1$，然后等比例地放大其他区域的 f_i，使得约束(10)成立。不断重复该过程，使得所有 $f_i\leqslant 1$。该算法求出的结果不一定是最优的，但如果数据满足一定的分布条件，则可以使用贪心算法求解最优值。

如果 $(G(t_{i-1})-G(t_i))/(H(t_{i-1})-H(t_i))$ 关于 i 单调递增（即 f_i 单调递增），那么所有 f_i 取值大于 1 的区域都在最右边。遍历最右边区域的左端点 t_{k-1} 的所有可能取值，对于 $[0,t_{k-1}]$ 区间，用 DP 算法划分为 $k-1$ 份，这样就得到了一个划分。通过上文所述的算法使得该划分满足约束后，计算该划分下的 PLBF 大小，遍历结束后取使 PLBF 的大小最小的划分。相比 Ada-BF，PLBF 对数据分布的假设更弱，且能够求得最优的划分，但是 PLBF 的训练耗时相比 Ada-BF 更长。

3.5 Stable LBF

上述学习型 Bloom 过滤器都是针对静态的键值集合设计的，键值的个数是固定的，当不断地有新键值插入时，过滤器的假阳性率将迅速上升。这种性能退化对于学习型 Bloom 过滤器而言尤为严重，因为学习型 Bloom 过滤器的后置过滤器通常很小，导致假阳性率上升更快。针对这个问题，Liu 等[23]提出了 Stable LBF，以产生小部分假阴性为牺牲，保证了学习型过滤器在无限的数据插入中，假阳性率可以收敛到上界。

Stable LBF（SBF）的结构如图 5 所示，其与 Ada-BF 类似，只是将所有的 Bloom 过滤器替换成了稳定的 Bloom 过滤器（Stable Bloom Filter，SBF），这是一种用来解决密集插入型工作流下 Bloom 过滤器性能下降问题的过滤器。SBF 将 Bloom 过滤器的比特位替换成一个 d 位的计数器，计数器的取值为 $[0,Max]$（$Max=2^d-1$）。当插入一个元素时，将随机地选择 P 个计数器，如果计数器的值不为 0 则令其值减 1。然后将元素哈希到的 k 个计数器值置为 Max。查询时，如果 k 个计数器都不为 0 则返回阳性，否则返回阴性。这种设计使得 SBF 可以随机地遗忘掉那些存在时间超过 Max 次插入的元素（即不常被插入的元素），但这也会导致假阴性的产生。但是 SBF 可以保证当插入元素趋于无穷大时，其假阳性率有一个上界，并且和 SBF 的计数器数量 m 无关，而 m 的大小

影响了 SBF 收敛到这个上界的速度。

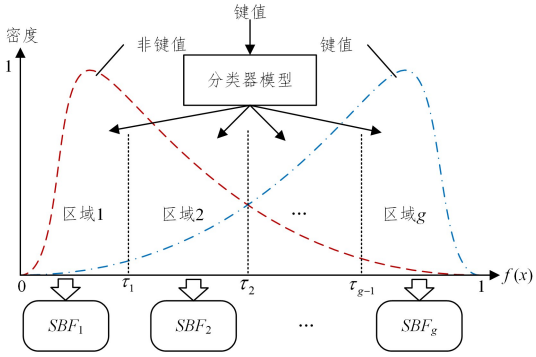

图 5　Stable LBF

Fig. 5　Stable LBF

经过无限次插入后 SBF 的假阳性率可表示为:

$$\lim_{n\to\infty} FPR_{SBF} = \left(1-\left(\frac{1}{1+k/P}\right)^{Max}\right)^{k} \tag{15}$$

其中,k 表示哈希函数的个数,FPR_{SBF} 的收敛速度可以表示为 $(k/m)(1-k/m)^n$。

与 Ada-BF 一样,SLBF 也将 f 的值域分为 g 个区域,并令 p_j,q_j 分别为非键值和键值落在区域 j 的概率。SLBF 假设键值和非键值分别服从分布 D_P 和 D_N,且对于 g 个区域,p_j 递减,q_j 递增。

SLBF 的期望假阳性率为 $FPR_O=\sum_{j=1}^{g} p_j \cdot \alpha_j$,其中 α_j 表示第 j 个 SBF 的假阳性率,由于 p_j 递减,根据排序不等式,当 α_j 递增时 FPR_O 最小[23]。

为了在不超过空间预算,同时达到预期的假阳性率的条件下,使假阴性率最小且整体收敛速度最快,SLBF 将[0,1]区间等分为 g 个区域,并计算 p_j 和 q_j,设置 $\alpha_j^{\text{obj}}=(\varepsilon/p_j)/(\sum_{l=1}^{g}1/p_l)$,其中 ε 是 SLBF 的预期假阳性率。接着,遍历 (k_j, Max_j) 的取值寻找使得第 j 个 SBF 的假阴性率最小的取值,并根据 (α_j, k_j, Max_j) 的值计算出 P_j。最后,为了使所有区域的 SBF 收敛速度相近,从而最小化整个 SLBF 的收敛速度,设置 $m_j=B\cdot(k_j/q_j)/(\sum_{l=1}^{g}k_l/q_l\cdot\log_2 Max_l+1)$,其中 B 是所有 SBF 总的空间预算[23]。

4　基于结构设计的学习型过滤器

第 3 章介绍的学习型过滤器都是根据数据分布的特点,学习出可以区分键值和非键值的分类器,再使用较小的 Bloom 过滤器消除假阴性。然而这些过滤器严重依赖于分类器模型的性能,当分类器模型的准确率较低时就会导致学习型过滤器的假阳性率较高。而提升分类器模型的准确率又可能导致模型的参数变多(例如增加 RNN 的层数),从而大大增加学习型过滤器的大小和训练、查询的时间。

因此,出现了一类学习型过滤器,通过利用学习到的数据分布特征,改造自身的结构,从而在相同大小下达到更好的假阳性率。本章介绍的学习型过滤器都是属于此类。

4.1　Stacked Filter

Deeds 等[25]提出了一种多层的过滤器结构 Stacked Filter,它增加一层过滤器来记录最经常被查询的假阳性,并再加一层过滤器来消除由此产生的假阴性,通过这样成对地增加过滤器可以实现较低的假阳性率,其结构如图 6 所示。

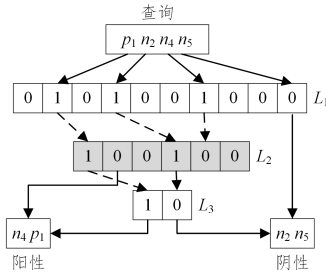

图 6　Stacked Filter

Fig. 6　Stacked Filter

Stacked Filter 的总层数一定为奇数,令 T_L 表示总层数,L_i 表示第 i 层的过滤器。令 S_p 表示键值集合,S_{N_f} 表示非键值集合,初始时 $S_p=K$,S_{N_f} 为工作负载中最常被查询的非键值元素集合 N_f。

构建时,对于奇数层,先将 S_p 中的元素插入到 L_i 中,然后在 L_i 上查询 S_{N_f} 中的元素,删去其中结果为阴性的元素,此时 S_{N_f} 表示 L_i 产生的假阳性;对于偶数层,先将 S_{N_f} 中的元素插入到 L_i 中,然后在 L_i 上查询 S_p 中的元素,此时 S_p 表示 L_i 产生的假阴性。经过这样的构造过程,L_1 记录了所有的键值,L_{2i} 记录了 L_{2i-1} 中所有经常被查询的假阳性。如果元素查询 L_{2i} 的结果为阳性,则说明该元素大概率是非键值,但是也会有小概率是键值,因此使用 L_{2i+1} 来记录这些假阴性。

Stacked Filter 使用交替式的查询方式:当查询元素 x 时,对于奇数层,如果查询结果为阴性,则返回阴性,否则继续查询偶数层;如果偶数层的查询结果为阴性,则返回阳性,否则继续查询下一层;如果对于所有的 T_L 层,查询 x 的结果都是阳性,则返回阳性。因此,对于 N_f 中的元素来说,只有穿过所有的 T_L 层,即对每一层的查询结果都是阳性,才会产生假阳性,这就过滤了大部分工作负载中最常被查询的假阳性,因此对于整个工作负载来说,极大降低了假阳性率。

Stacked Filter 的假阳性率分为两部分,分别是查询 N_f 中的元素的假阳性率和查询 $N-N_f$ 中的元素的假阳性率。设 $\psi=P(x\in N_f \mid x\in N)$,$L_i$ 的假阳性率都为 α,则 Stacked Filter 的假阳性率可以表示为[25]:

$$FPR_O=\psi\alpha^{\frac{T_L+1}{2}}+\frac{(1-\psi)(\alpha+\alpha^{(T_L+1)})}{1+\alpha} \tag{16}$$

由于每层需要表示的元素数量都呈几何级减少,每层的大小也呈几何级减少,因此 Stacked Filter 的总大小和 L_1 的大小相近。设 $s'(\alpha)$ 表示 Stacked Filter 存储每个键值所需的比特数,$s(\alpha)$ 表示假阳性率为 α 时过滤器存储每个键值所需的比特数,则 $s'(\alpha)$ 可以表示为[25]:

$$s'(\alpha)=s(\alpha)\cdot\left(\frac{1}{1-\alpha}+\frac{|N_f|}{|K|}\frac{\alpha}{1-\alpha}\right) \tag{17}$$

由于不需要使用分类器模型,因此 Stacked Filter 的构建和查询速度相比基于分类器的学习型 Bloom 过滤器更快。设过滤器的插入和查询操作所需的耗时为 c_i 和 c_q,则 Stacked Filter 的构建时间可以表示为[25]:

$$\frac{|K|(c_i+c_q)}{1-\alpha}+|N_f|c_q+\frac{|N_f|(c_i+c_q)\alpha}{1-\alpha} \tag{18}$$

当 α 很小时，Stacked Filter 的构建时间约为 $(|K|+|N_f|)c_q+|K|c_i$，即构建 L_1 的时间。

Stacked Filter 的查询时间可以表示为[25]：

$$c'_{q,K}\leqslant\frac{2}{1-\alpha}c_q,c'_{q,N_f}\leqslant\frac{1+2\alpha}{1-\alpha}c_q,c'_{q,N-N_f}\leqslant\frac{1}{(1-\alpha)}c_q \tag{19}$$

当 α 很小时，查询键值的时间接近于 2 倍的过滤器查询时间。

Stacked Filter 有 3 个需要优化的参数，即 N_f，T_L 和 α，优化过程分为内外两个循环。外循环贪婪地遍历 N_f，每次增大 $|N_f|$ 都从采样的查询集合选取前 $|N_f|$ 个查询最频繁的非键值组成 $|N_f|$，并计算出 ψ。对于每一个 N_f，内循环首先假设 $T_L\to\infty$，则 $FPR_O=(1-\psi)\cdot\alpha/(1-\alpha)$ 对于 α 单调递增，因此可以对 $s'(\alpha)$ 使用梯度下降，求得使 $s'(\alpha)$ 最小的 α。接着从 1 开始增大 T_L，直到在当前 T_L 下，Stacked Filter 的假阳性率和所用空间的大小与 $T_L\to\infty$ 时差距不超过 ε。由于 Stacked Filter 的收敛速度很快，因此 T_L 的取值通常在 5 到 7 之间[25]。

Stacked Filter 使用的过滤器并不只局限于 Bloom 过滤器，可以简单地替换成 Quotient 过滤器和 Cuckoo 过滤器，并达到相同的性能保障。此外，Stacked Filter 还具有动态构建的能力，针对变化的工作负载，Stacked Filter 可以实时采样查询集合并更新 N_f，再根据需要逐层地扩展。

4.2 PHBF

Bhattacharya 等[27]提出了一种学习型过滤器 Projection Hash Bloom Filter(PHBF)，PHBF 通过学习数据分布，构造一类能够聚类键值和非键值的哈希函数，并替换 Bloom 过滤器中的哈希函数，其结构如图 7 所示。

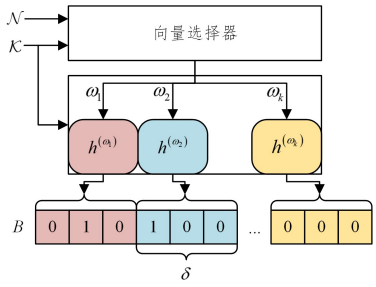

图 7 PHBF

Fig. 7 PHBF

PHBF 将 Bloom 过滤器等分为 k 个长度为 δ 的区域，每个区域 i 对应一个哈希函数 $h^{(\omega_i)}:U\to\{1,\cdots,\delta\}$，将元素映射到该区域内。插入元素 x 时，将根据这 k 个哈希函数插入到 Bloom 过滤器 B 中，即 $B[(i-1)\delta+h^{(\omega_i)}(x)]=1$。查询和 Bloom 过滤器一样，如果哈希到的 k 个比特位都是 1，则为阳性，否则为阴性。

PHBF 的关键在于 k 个哈希函数 $h^{(\omega_i)}$ 的选择：在构建过滤器之前，PHBF 会根据键值集合 K 和抽样的非键值集合 N 选择出 k 个最佳的哈希函数，这些哈希函数使 K 和 N 中的元素有最少的碰撞。

PHBF 将 U 中的元素看作 d 维的向量(对于字符串，可以通过切片的方式构造)，首先从 d 维单位向量空间 $\mathbf{1}_d$ 中抽样 sk 个单位向量 $v_1,\cdots,v_{sk}$，则 $h^{(v_i)}(x)$ 表示将 x 投影到 v_i 上并缩放到区间 $[0,m]$ 上，即 $h^{(v_i)}(x)=m\times|v_i\cdot x|/x$，其中 $m=\delta k$ 是 Bloom 过滤器的大小。接着，计算在投影向量 v_j 下 K 和 N 的碰撞次数 c_j，$c_j=\sum_{i=1}^{m}\mathbb{I}(h^{(v_j)}(x)=h^{(v_j)}(y)=i)$，$x\in K$，$y\in N$。最后选择前 k 小的 c_j 对应的 v_j 组成 ω，并令 $h^{(\omega_i)}(x)=\delta\times|\omega_i\cdot x|/x$[27]。

PHBF 的查询和训练速度都非常快，计算单个哈希函数的时间为 $\Theta(d)$，因此查询时间为 $\Theta(kd)$，整个 PHBF 的构建时间为 $\Theta((s+1)nkd)$[27]。

令 K 和 N 中元素的平均值分别为 μ_K 和 μ_N，方差为 σ_K^2 和 σ_N^2。对于 PHBF 中的任意区域 i，可以将其看作是 $k=1,\delta=1$ 的 PHBF，令 $\hat{x}_m=\max_{x\in K}\langle x,\omega_i\rangle$，$\hat{y}=\langle y,\omega_i\rangle$，$y\in N$，则区域 i 的假阳性 FPR_i 的计算式如下[27]：

$$FPR_i\leqslant\int_{-\infty}^{\infty}Pr\{\hat{y}\leqslant r\}\mathrm{d}r \tag{20}$$

设 ω_i 与 $l=\mu_K-\mu_N$ 形成夹角 θ，使用切比雪夫不等式，可以得到[27]：

$$FPR_i\leqslant 4n(\sigma_K^2+\sigma_N^2)/l^2\cos^2\theta \tag{21}$$

Bhattacharya 等证明，以至少 $1-f(d,l)^s$ 的概率，夹角 θ 满足 $\max_i\{\cos\theta_i\}\geqslant\sqrt{d/l}$，其中 $f(d,l)=1-1/d\,(1-1/dl)^{\frac{2}{d}}$，因此 $FPR_i\leqslant 4nd(\sigma_K^2+\sigma_N^2)/l$。整个 PHBF 的假阳性率为 k 个区域的假阳性率相乘，取 $s=2d(\ln k+\ln 2/\varepsilon)$，则以至少 $(1-f(d,l)^s)^k\geqslant 1-\varepsilon/2$ 的概率，$FPR_O\leqslant(4nd(\sigma_K^2+\sigma_N^2)/l)^k=\varepsilon/2$；且以最多 $\varepsilon/2$ 的概率，$FPR_O\leqslant 1$。因此，$FPR_O\leqslant(1-\varepsilon/2)(\varepsilon/2)+\varepsilon/2=\varepsilon$，即 PHBF 的假阳性率上界为 ε。

4.3 哈希自适应 Bloom 过滤器

Bloom 过滤器产生假阳性的原因是非键值元素和键值元素之间的哈希碰撞，但由于传统的哈希函数考虑的是降低所有元素的碰撞概率，因此这些哈希函数并不会区分不同的元素。此外，由于不同的假阳性导致的误判代价各不相同(例如频繁被查询的假阳性)，如果能针对其中代价较高的假阳性，定制与其冲突的键值所使用的哈希函数集合，消除它们的碰撞从而消除假阳性，则可以大大降低假阳性率。基于这种想法，Li 等[24]提出了一种可以针对不同元素调整哈希集合的学习型 Bloom 过滤器 HABF，其结构如图 8 所示。设全局哈希函数集合 $H=\{h_1,\cdots,h_{|H|}\}$，键值 e 使用的哈希函数集合用 $\phi(e)\subset H(|\phi(e)|=k)$ 表示。初始时，所有键值使用初始哈希函数集合 H_0，对于定制了哈希函数集合的键值，HABF 使用一种叫 HashExpressor 的轻量级数据结构来记录它们。

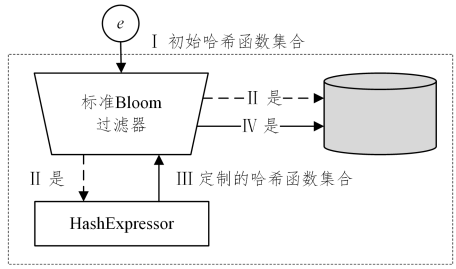

图 8 HABF

Fig. 8 HABF

在构建时，HABF 为不同键值定制哈希函数集合并存储

在 HashExpressor 中。在查询时,首先使用 H_0 查询 Bloom 过滤器 B,如果成功,则返回阳性,否则将查询 HashExpressor,通过其返回的哈希函数集合重新查询 B。

HashExpressor 是一张大小为 ω 的哈希表,其中每个位置 $C[i]=\langle endbit,hashindex\rangle$ 是一个二元组,$hashindex$ 是哈希函数在 H 中的索引,$endbit$ 用于判断查询到的哈希函数集合是否有效,当 $C[i]$ 为空时,$C[i]=\langle 0,0\rangle$。向 HashExpressor 中插入键值 x 的定制哈希函数集合 $\phi(x)$ 时,首先设置 $\varphi(x)$ 中所有的哈希函数为无效,接着使用哈希函数 $f:U\to\{1,\cdots,\omega\}$ 将 x 映射到 $C[f(x)]$ 上。如果 $C[f(x)]$ 为空,则随机选择 $\phi(x)$ 中的哈希函数 h 使其有效并设置 $C[f(x)]=\langle 0,i_h\rangle$($i_h$ 是 h 的索引)。若 $C[f(x)].hashindex$ 是一个在 $\phi(x)$ 中无效的哈希函数,则令 $\phi(x)$ 中的 $h=C[f(x)].hashindex$ 有效。如果不是以上两种情况,则插入失败。如果 $C[f(x)]$ 插入成功,则重复插入 $C[f(x)]$,直到 $\phi(x)$ 中所有哈希函数均有效,并令第 k 个单元的 $endbit=1$。

为了获取定制后的哈希函数集合 $\phi(x)$,首先查看 $C[f(x)]$,如果为空,则 $\phi(x)=H_0$;否则将 $C[f(x)]$ 中的哈希函数 h_{c1} 存入 $\phi(x)$,并映射到下一个单元 $C[h_{c1}(x)]$,再将其中的 h_{c2} 存入 $\phi(x)$,并重复 k 次。如果最后一个单元的 $endbit=1$,说明这是一个有效的哈希函数集合,查询返回 $\phi(x)$,否则返回 H_0。显然,HashExpressor 存在少量的假阳性,但不存在假阴性。

HABF 采用一种两阶段的联合优化算法定制键值的哈希函数集合,其流程如图 9 所示,其中阶段一为更换哈希函数,阶段二为插入 HashExpressor。首先使用 H_0 查询非键值集合,使用优先级队列 CQ 存储抽假阳性元素,并以误判代价作为优先级排序。接着构造辅助数据结构 V 和 Γ,V 用于记录 B 中只被映射一次的比特位以及对应键值的指针,Γ 用于记录映射到某位的非键值元素集合。

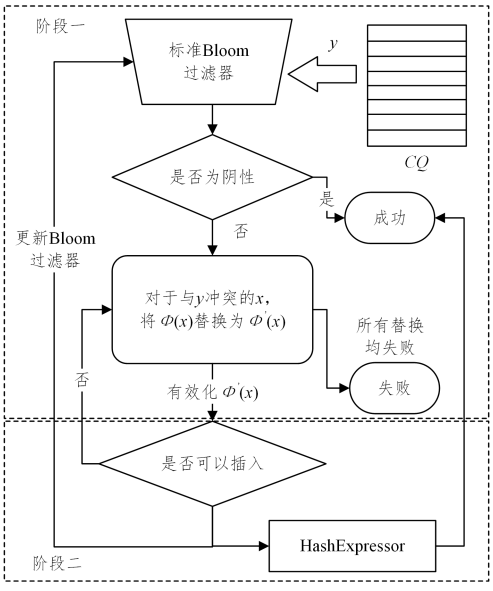

图 9　两阶段联合优化算法

Fig. 9　Two-phase joint optimization

在阶段一,每次从 CQ 中取出元素 y,用 H_0 将 y 映射到 k 位,并查询 V 获取那些只被映射过一次的比特位的集合 ξ。对于任意的 $u\in\xi$,u 对应的键值为 x,h_u 为对应的哈希函数且 $h_u(x)=u$。HABF 先将 u 设置为 0 消除哈希碰撞,再将 h_u 替换为 $H_c=H-\phi(x)$ 中的哈希函数,以调整 $\phi(x)$,从而消除由此产生的假阴性。

如果存在 $h_c\in H_c$,并且 $B[h_c(x)]=1$,则替换不会产生假阴性,因此可以直接将 h_u 替换为 h_c。否则需要将比特位设置为 1,然而这可能导致新的哈希碰撞,因此先将 x 映射到 $|H_c|$ 个比特位,然后再检查设置这些位会不会引起哈希碰撞。对于其中任意比特位 v,先将其设置为 1,查询 Γ 并令 ξ_v 表示新产生的假阳性,即 $\forall y\in\xi_v,h\in\phi(y):\sum_k\mathbb{I}(B[h(y)])=k-1$。如果存在 v,使得 $\xi_v=\varnothing$,则可以直接用该位对应的哈希函数 h_v 替换 h_u;否则选择能使所有假阳性的误判代价之和最小的位,并用映射到该位的哈希函数替换。

在阶段二,如果能将在阶段一调整了 $\phi(x)$ 的键值元素 x 插入到 HashExpressor 中,那么就把 y 插入到 Γ 中并更新 V,然后把新产生的假阳性插入到 CQ 中。

HashExpressor 不存在假阴性的性质也保证了 HABF 不存在假阴性:如果元素 x 没有被插入过 HashExpressor,即 $\phi(x)=H_0$,则其对应的位没有被修改为 0,因此不会产生假阴性;如果元素 x 被插入过 HashExpressor,由于 HashExpressor 不存在假阴性,因此使用 HashExpressor 返回的 $\phi(x)$ 也不会产生假阴性。使用如图 8 所示的两阶段查询可以计算 HABF 的假阳性率:

$$\begin{aligned}FPR_O&=1-(1-FPR_{bf}^*)(1-FPR_h+FPR_h\cdot\\&\qquad(1-FPR_{bf}^*))\\&=FPR_{bf}^*(1+FPR_h-FPR_hFPR_{bf}^*)\end{aligned}\tag{22}$$

其中,FPR_{bf}^* 表示优化后的 Bloom 过滤器的假阳性率,FPR_h 表示 HashExpressor 的假阳性率。假设有 t 个元素插入到了 HashExpressor 中,则 $FPR_h\leqslant t/\omega$,因此 $FPR_O\leqslant((\omega+t)/\omega)\cdot FPR_{bf}^*$,当 $t\ll\omega$ 时,$FPR_O\approx FPR_{bf}^*$。由于优化后的 FPR_{bf}^* 一定小于传统 Bloom 过滤器的假阳性率,因此 HABF 有更小的假阳率。

5　总结与展望

相比传统的 Bloom 过滤器,学习型 Bloom 过滤器有效地利用了数据分布特征的信息,具有突破传统过滤器尺寸的理论下界的潜力,因此在近年来成为了热门研究方向。

学习型 Bloom 过滤器大致可以分为两类:基于分类器的学习型 Bloom 过滤器和基于结构化改造的学习型 Bloom 过滤器。前者使用机器学习方法,从数据特征中学习一个分类器,用于区分键值与非键值,并使用辅助的过滤器数据结构消除假阴性。后者可以根据学习到的数据分布信息,改变自身的结构,并使用最优结构以适应当前的数据分布。

基于分类器的过滤器可以更好地利用新型硬件(例如 GPU 和 TPU)的加速,基于结构化改造的过滤器则具有更小的计算时间复杂度。但这两种学习型 Bloom 过滤器都被证明,在满足一定数据分布条件下比传统 Bloom 过滤器有着更小的理论下界。因此,学习型 Bloom 过滤器具有广泛的应用前景。

然而,目前对于学习型 Bloom 过滤器的研究依然有许多

不足，未来的工作可以着力于以下方面。

1)设计适合过滤器场景的损失函数。目前基于分类器的学习型 Bloom 过滤器使用的损失函数是对数损失函数，其作用是提高分类器的分类准确率。但是在过滤器场景下，优化目标是最小化整体的假阳性率，因此相关研究可以从假阳性率的表达式出发，探索最适合的损失函数。

2)支持动态插入和删除。当前的学习型过滤器绝大多数都面向静态数据集，唯一考虑了插入的 Stable LBF 也是以牺牲零假阴性的性能保障为前提，因此限制了学习型 Bloom 过滤器的应用场景。针对数据集动态更新的场景，如何实时感知数据分布的变化，提升过滤器的鲁棒性和安全性，也是一个重要的研究方向。

3)降低训练耗时。学习型过滤器相比传统的过滤器最大的缺点在于其计算成本太高，这是由分类器模型和复杂的结构造成的。尽管基于结构化改造的学习型过滤器已经考虑到降低计算的时间复杂度，但是其训练阶段依然有较高的开销。是否可以设计更快的训练方法，使得学习型过滤器的构建时间接近传统过滤器，也值得研究。

4)提高并行性。目前 SIMD、MIMD 体系结构和 NUMA 架构等并行计算技术已经得到广泛的应用，而学习型过滤器中还存在许多串行操作，例如需要等到分类器的分类结果、需要等待上层过滤器的查询结果等，并没有针对并行计算做适配。相关研究可以考虑设计并行化的数据结构和算法，使得学习型过滤器具有良好的并行性。

5)适配新型存储设备。学习型 Bloom 过滤器降低假阳性率的目的是减少昂贵的外存访问，而由于非易失性内存(Non-volatile Memory，NVM)的出现，外存和内存之间的速度差距大大减小，在这种背景下，就不能简单地考虑以牺牲计算速度为代价降低假阳性率或者空间使用，需要针对新型存储设备做相关的适配。

结束语 学习型过滤器是一种基于机器学习技术的概率性结构，用于解决集合成员查询问题。这些过滤器提出了考虑数据分布信息，将集合成员查询问题视为二分类问题，从而实现了超越传统过滤器的性能。本文重点回顾了近些年最新的学习型过滤器的相关工作，并对未来的发展方向做了展望。未来，随着学习型过滤器的技术发展，其应用也会越来越广泛。

参 考 文 献

[1] FAN B, ANDERSEN D G, KAMINSKY M, et al. Cuckoo Filter: Practically Better Than Bloom[C]// Proceedings of the International Conference on emerging Networking Experiments and Technologies. ACM, 2014: 75-88.

[2] DUTTA S, NARANG A, BERA S K. Streaming quotient filter: a near optimal approximate duplicate detection approach for data streams[C]// Proceedings of the VLDB Endowment. VLDB, 2013: 589-600.

[3] LI M, CHEN D, DAI H, et. al. Seesaw Counting Filter: An Efficient Guardian for Vulnerable Negative Keys During Dynamic Filtering[C]// Proceedings of the Web Conference 2022. ACM, 2022: 2759-2767.

[4] DAI H, SHAHZAD M, LIU A X, et al. Identifying and Estimating Persistent Items in Data Streams[J]. IEEE/ACM Transactions on Networking, 2018, 26(6): 2429-2442.

[5] WANG H, DAI H, LI M, et al. Bamboo Filters: Make Resizing Smooth[C]// Proceedings of the International Conference on Data Engineering. IEEE, 2022: 979-991.

[6] LI M, DAI H, WANG X, et al. Thresholded Monitoring in Distributed Data Streams[J]. IEEE/ACM Transactions on Networking, 2020, 28(3): 1033-1046.

[7] ABDENNEBI A, KAYA K. A Bloom Filter Survey: Variants for Different Domain Applications[J]. arXiv: 2106. 12189, 2021.

[8] DAI H, YU J, LI M, et al. Bloom Filter With Noisy Coding Framework for Multi-Set Membership Testing[J]. IEEE Transactions on Knowledge and Data Engineering, 2023, 35(7): 6710-6724.

[9] DAI H, ZHONG Y, LIU A X, et al. Noisy Bloom Filters for Multi-Set Membership Testing[C]// Proceedings of the International Conference on Measurement and Modeling of Computer Science. ACM, 2016: 139-151.

[10] CHEN Z, HUANG J, WANG Q, et al. MEB: an Efficient and Accurate Multicast using Bloom Filter with Customized Hash Function[C]// Proceedings of the 7th Asia-Pacific Workshop on Networking. ACM, 2023: 157-163.

[11] EVEN T, EVEN G, MORRISON A. Prefix filter: practically and theoretically better than bloom[C]// Proceedings of the VLDB Endowment. VLDB, 2022: 1311-1323.

[12] DAYAN N, TWITTO M. Chucky: A Succinct Cuckoo Filter for LSM-Tree[C]// Proceedings of the International Conference on Management of Data. ACM, 2021: 365-378.

[13] GUBNER T, TOMÉ D, LANG H, et al. Fluid Co-processing: GPU Bloom-filters for CPU Joins[C]// Proceedings of the 15th International Workshop on Data Management on New Hardware. ACM, 2019: 1-10.

[14] LI Y, WANG Z, YANG R, et al. Learned Bloom Filter for Multi-key Membership Testing[C]// Proceedings of the Database Systems for Advanced Applications. Cham: Springer Nature Switzerland, 2023: 62-79.

[15] FU P, LUO L, LI S, et al. The Vertical Cuckoo Filters: A Family of Insertion-friendly Sketches for Online Applications[C]// Proceedings of the International Conference on Distributed Computing Systems. IEEE, 2021: 57-67.

[16] DAI M, XU S, SHAO S, et al. Blockchain-Based Reliable Fog-Cloud Service Solution for IIoT [J]. Chinese Journal of Electronics, 2021, 30(2): 359-366.

[17] LI P, YU X, XU H, et al. Secure Localization Technology Based on Dynamic Trust Management in Wireless Sensor Networks [J]. Chinese Journal of Electronics, 2021, 30(4): 759-768.

[18] LI L, WU J, HU H, et al. Secure Cloud Architecture for 5G Core Network [J]. Chinese Journal of Electronics, 2021, 30(3): 516-522.

[19] KRASKA T, BEUTEL A, CHI E H, et al. The Case for Learned Index Structures[C]// Proceedings of the International Conference on Management of Data. ACM, 2018: 489-504.

[20] MITZENMACHER M. A Model for Learned Bloom Filters and

Optimizing by Sandwiching[C]// Proceedings of the Advances in Neural Information Processing Systems. Curran Associates, Inc.,2018:462-471.

[21] DAI Z,SHRIVASTAVA A. Adaptive Learned Bloom Filter (Ada-BF): Efficient Utilization of the Classifier with Application to Real-Time Information Filtering on the Web[C]// Proceedings of the Advances in Neural Information Processing Systems. ACM,2020:11700-11710.

[22] VAIDYA K,KNORR E,KRASKA T,et al. Partitioned Learned Bloom Filter[J]. arXiv:2006.03176,2020.

[23] LIU Q,ZHENG L,SHEN Y,et al. Stable learned bloom filters for data streams[C]// Proceedings of the VLDB Endowment. VLDB. 2020:2355-2367.

[24] LI M,XIE R,CHEN D,et al. A Pareto optimal Bloom filter family with hash adaptivity[J]. The Springer International Journal on Very Large Data Bases,2022,32(3):525-548.

[25] DEEDS K,HENTSCHEL B,IDREOS S. Stacked filters: learning to filter by structure[C]// Proceedings of the VLDB Endowment. VLDB,2020:600-612.

[26] BLOOM B H. Space/time trade-offs in hash coding with allowable errors[J]. Communications of the ACM,1970,13(7):422-426.

[27] BHATTACHARYA A,GUDESA C,BAGCHI A,et al. New wine in an old bottle: data-aware hash functions for bloom filters [C]// Proceedings of the VLDB Endowment. VLDB, 2022: 1924-1936.

[28] DAYAN N,ATHANASSOULIS M,IDREOS S. Optimal Bloom Filters and Adaptive Merging for LSM-Trees[J]. ACM Transactions on Database Systems,2018,43(4):1-48.

[29] CHEN H,LIAO L,JIN H,et al. The dynamic cuckoo filter [C]// Proceedings of the International Conference on Network Protocols. IEEE,2017:1-10.

[30] BRESLOW A D,JAYASENA N S. Morton filters: faster, space-efficient cuckoo filters via biasing, compression, and decoupled logical sparsity[C]// Proceedings of the VLDB Endowment. VLDB,2018:1041-1055.

[31] GRAF T M,LEMIRE D. Binary Fuse Filters: Fast and Smaller Than Xor Filters[J]. ACM Journal of Experimental Algorithmics,2022,27:1-15.

[32] FICARA D,DI PIETRO A,GIORDANO S,et al. Enhancing counting bloom filters through Huffman-coded multilayer structures[J]. IEEE/ACM Transactions on Networking, 2010, 18(6):1977-1987.

[33] LAUFER R P,VELLOSO P B,DUARTE O C M B. A Generalized Bloom Filter to Secure Distributed Network Applications [J]. Computer Networks,2011,55(8):1804-1819.

LI Meng, born in 1993, Ph.D, is a member of CCF(No. B5508M). His main research interests include data-intensive IOT systems and so on.

DAI Haipeng, born in 1985, Ph.D, associate professor, Ph. D supervisor, is a distinguished member of CCF (No. 19625S). His main research interests include data mining and mobile computing.

(责任编辑:喻藜)