arXiv:2301.08892v1 [cs.LG] 21 Jan 2023

快速基于似然的变点检测

Nikolaj Tatti[0000−0002−2087−5360]

赫尔辛基信息技术研究所，赫尔辛基大学，芬兰
赫尔辛基 nikolaj.tatti@helsinki.fi

摘要。变点检测在许多现实应用中发挥着基础作用，其目标是分析和监测数据流的行为。本文研究二元数据流中的变化检测。为此，我们使用两个模型之间的似然比作为指示变化的度量。第一个模型是单个伯努利变量，第二个模型则将存储的数据分为两个片段，并分别用各自的伯努利变量对每个片段建模。找到最优分割可在 $\mathcal{O}(n)$ 时间内完成，其中 n 是自上一个变点以来的条目数。对于大型 n，这种方法开销过高。为了解决这个问题，我们提出一种近似方案，可在 $\mathcal{O}\left(\epsilon^{-1}\log^2 n\right)$ 时间内得到 $(1-\epsilon)$ 近似。加速分为几个步骤：首先，我们借鉴分段问题中的一个已知结果，减少可能候选项的数量。然后，我们证明，对于固定的伯努利参数，可以在对数时间内找到最优变点。最后，我们展示如何为模型参数构建大小为 $\mathcal{O}\left(\epsilon^{-1}\log n\right)$的候选列表。我们通过实验证明了该算法的近似质量和运行时间，结果表明，只需付出极小的平均最优性损失，就能获得显著加速。

1 引言

许多现实应用都涉及对持续数据流的监控和分析。这类应用中的一项基本任务是监控是否发生了变化。例如，我们可能需要持续监控分类器的性能，并在其质量下降过多时触发重新训练。我们还可以使用变点检测技术来检测数据流中的异常行为。由于数据流量可能很大，因此开发高效算法至关重要。

本文研究如何检测二进制数数据流中的变化，也就是说，我们关注底层分布近期是否发生了显著变化。为了检验是否发生变化，我们将使用标准的似然比统计量。具体而言，假设自上次观测到变化以来，我们已经观测了 n 个样本。在第一个模型中，我们用一个伯努利变量拟合这些样本。在第二个模型中，我们将这些样本分成两半，比如在位置 i处分开，并分别用一个伯努利变量拟合这两半。完成后，我们比较两个模型的似然比。如果比率足够大，我们就判定发生了变化。

在我们的设定中，索引 i 并非固定不变。相反，我们要寻找能使似然最大的索引。朴素做法是逐一测试每个候选索引，可在 $\mathcal{O}(n)$时间内完成。这可能太慢，尤其是当 n 足够大，而我们又没有足够资源在新样本到达前完成计算时。我们的主要技术贡献是展示如何在 $\mathcal{O}\left(\epsilon^{-1}\log^2 n\right)$ 时间内获得最优i 的$(1-\epsilon)$近似。

为实现这一点，我们首先会减少最优索引 i 的候选项数量。我们称索引 j 为边界索引，若所有以 $j-1$ 结尾的区间中 1 的比例，都小于所有以 j 开始的区间中的 1 的比例。已知结果表明，最优变点必定是边界索引之一。在实践中，使用边界索引已大幅缩短搜索时间，其理论运行时间为 $\mathcal{O}\left(n^{2/3}\right)$。

为获得更小的界，我们证明，对于固定的模型参数（即两个伯努利变量的参数），可以在 $\mathcal{O}(\log n)$ 时间内，从边界索引中找到最优索引。随后，我们为这些参数构造一个包含 $\mathcal{O}\left(\epsilon^{-1}\log n\right)$ 个候选项的列表。此外，该列表还会包含与最优参数足够接近的模型参数，因此对这些参数进行测试可在 $\mathcal{O}\left(\epsilon^{-1}\log^2 n\right)$ 时间内获得 $(1-\epsilon)$ 近似保证。

本文其余部分安排如下。第2节介绍预备符号并定义问题。第3节介绍边界点。我们在第4–5节阐述主要技术贡献：首先展示如何在模型参数固定时找到最优索引，然后展示如何为这些参数选择候选项。第6节介绍相关工作，第7节进行实证评估。最后，我们在第8节总结并讨论。

2 预备知识与问题定义

假设有一个由 $S = s_1$、...、s_n 组成的二进制数序列 n。这里， s_1 要么是数据流的起始位置，要么是我们上一次检测到变点的时刻。我们的目标是确定 S 中是否发生了变化。更具体地说，我们考虑两种统计模型：第一种模型 M_1 假设 S 由一个伯努利变量生成。第二种模型 M_2 假设存在一个索引 i（即变点），使得 s_1、...、s_{i-1} 由一个伯努利变量生成，而 s_i、...、 s_n 由另一个伯努利变量生成。

给定一个序列 S，我们将拟合 M_1 和 M_2，并比较对数似然。注意，模型 M_2 取决于变点 i，因此我们需要选择使 M_2 的似然最大化的 i。如果比率足够大，我们就可以确定已发生 Change。

为了更正式地阐述上述讨论，我们先引入一些记号。给定两个整数 a 和 b, 以及一个介于 0 和 1 之间的实数，我们用以下符号表示伯努利变量的对数似然：

$$\ell(a, b; p) = a \log p + b \log(1-p) \quad .$$

对于固定的 a和 b，当 $p = a/(a+b)$ 时，对数似然达到最大值。在这种情况下，我们通常会在记号中省略 p，直接写作 $\ell(a, b)$。

我们有以下优化问题。

问题 1（CHANGE）。给定序列 $S = s_1, \ldots, s_n$，求索引 i，使得

$$\ell(a_1, b_1) + \ell(a_2, b_2) - \ell(a, b)$$

达到最大值，其中

$$a_1 = \sum_{j=1}^{i-1} s_i,\ b_1 = i - 1 - a_1, \quad a_2 = \sum_{j=i}^{k} s_i,\ b_2 = k - i - a_2,$$
$$a = a_1 + a_2, \text{ and } b = b_1 + b_2 \quad .$$

注意，只需遍历 i 的所有可能取值，即可在 $\mathcal{O}(n)$ 时间内求解 CHANGE 。这样的运行时间可能太慢，尤其是在新数据点不断到来的流式场景中，而我们的目标是实时判断是否发生了变化。本文的主要贡献是展示如何在 $\mathcal{O}\left(\epsilon^{-1}\log^2 n\right)$ 时间内计算 CHANGE 的 $(1-\epsilon)$ 估计。该算法需要额外的数据结构，我们将在下一节回顾这些结构。由于我们的主要应用是在数据流中搜索变点，因此需要在数据流上维护这些结构。幸运的是，正如下一节所示，存在一种摊销常数时间算法来维护所需结构。

求解 CHANGE 后，我们将得到的得分与阈值 σ 进行比较。注意， M_2 的似然总是大于 M_1。本文将使用贝叶斯信息准则（BIC）来调整 M_2 带来的额外模型复杂度。模型 M_2 有三个参数，而模型 M_1 有 1 个参数。因此，BIC 惩罚项为 $(3-1)/2\log\, n = \log\, n$。实践中，由于多重假设检验问题，在选择 M_2 时需要更加谨慎。因此，我们将使用 $\sigma = \tau + \log\, n$ 作为阈值。这里， τ 是用户参数；我们将在第7节的实验评估中提供有关如何选择 τ 的一些指导。

当变点出现在位置 i 时，我们有两种选择：要么丢弃当前窗口并从头开始，要么只删除最前面的 i 个元素。本文采用前一种方法，因为后一种方法需要额外维护，可能会影响整体计算复杂度。

3 减少候选项数量

加快变点发现的第一步是减少可能的变点数量。为此，我们定义 CHANGE的一种变体，其中要求 M_2 中的第二个参数大于第一个参数。

问题 2（CHANGEINC）。给定一个序列 $S = s_1, \ldots, s_n$，求一个索引 i，使得

$$\ell(a_1, b_1) + \ell(a_2, b_2) - \ell(a, b)$$

达到最大值，其中

$$a_1 = \sum_{j=1}^{i-1} s_i,\ b_1 = i - 1 - a_1, \quad a_2 = \sum_{j=i}^{k} s_i,\ b_2 = k - i - a_2,$$
$$a = a_1 + a_2, \text{ and } b = b_1 + b_2$$

使用 $a_1/(a_1 + b_1) \leq a_2/(a_2 + b_2)$。

从现在起，我们将专注于解决 ChangeInc。这个问题本身就有意义，例如，若目标是检测分类器性能的恶化，也就是值为 1 的条目突然增加。不过，我们也可以使用 ChangeInc 来解决 Change。具体做法是定义一个翻转序列$S' = s'_1, .., s'_n$，其中 $s'_i = 1 - s_i$。那么， Change 的解要么是 ChangeInc (S) 的解，要么是 ChangeInc(S') 的解。

接下来我们将证明，求解ChangeInc时只需考虑边界索引。

定义 1. 假设有一个由二进制数组成的序列 $S = (s_i)_{i=1}^n$。如果不存在满足 $x < j < y$ 的索引 x、y，使得.....，则称索引 j 为边界索引。

$$\frac{1}{j-x}\sum_{i=x}^{j-1} s_i \geq \frac{1}{y-j}\sum_{i=j}^{y-1} s_i \quad .$$

换句话说， j 是边界索引，当且仅当以 $j-1$ 结尾的任意区间的平均值都小于以 j 开始的任意区间的平均值。

命题1. 存在一个边界索引 i，它可以求解 ChangeInc。

该命题可由 [19] 中定理1的一个变体推出。为求完整起见，我们在补充材料的附录中给出一个直接证明。

本节末尾将讨论如何维护边界索引。

根据该命题，我们可以忽略所有非边界索引。也就是说，可以将序列中的元素分组成块，每个块都以一个边界索引开头。然后，我们可以使用这些块而不是原始序列来搜索 i。

不难看出，这些块具有以下性质：下一个块中 1 的比例总是更大。这一关键特性将在接下来的两节中发挥重要作用，因为它使我们能够使用二分查找技术并降低计算复杂度。为了利用这一特性，我们重新表述原问题。首先，我们来定义什么是块序列。

定义 2. 设 $B = \langle (u_i, v_i) \rangle_{i=1}^k$ 是由 k 个非负整数对组成的序列，且满足 $u_i + v_i > 0$。若 $\frac{u_{i+1}}{u_{i+1}+v_{i+1}} > \frac{u_i}{u_i+v_i}$，则称 B 为块序列。

我们通过对边界点之间的条目进行分组，从二进制序列 S 得到块序列 B：计数器 u_i 表示 1 的个数，而计数器 v_i 表示 0 的个数。

我们的目标是使用块序列来解决 ChangeInc。首先，我们需要引入一些额外的记号。

定义 3. 给定一个块序列 B，我们定义 $B[i;\ j] = (a, b)$，其中 $a = \sum_{k=i}^{j} u_k$ 且 $b = \sum_{k=i}^{j} v_k$。如果 $i > j$，那么 $a = b = 0$。此外，我们将记作

$$av(i, j; B) = \frac{a}{a + b} \quad .$$

如果上下文中已知 B，我们将记作 $\mathsf{av}(i, j)$。

定义 4。给定块序列 B，我们将变点 i 的得分定义为

$$q(i; B) = \ell(a_1, b_1) + \ell(a_2, b_2) - \ell(a, b)\,, \tag{1}$$

其中 $(a_1, b_1) = B[1;\ i-1]$, $(a_2, b_2) = B[i;\ k]$，且 $a = a_1 + a_2$ 和 $b = b_1 + b_2$。注意，$\ell(a, b)$ 是一个常数，但保留它很有用，因为 $\mathsf{q}(i;\ B)$ 是两个模型之间的对数似然比，而且这种表述使我们能够估计第 5 节中的目标函数。

问题 3（ChangeBlock）。给定块序列 B，求使 $\mathsf{q}(i; B)$ 最大化的变点 i。

我们可以通过维护由边界点诱导的块序列来求解 ChangeInc，并求解 ChangeBlock。最直接的方法是对 $\mathcal{O}$ 中的每个索引计算 $\mathsf{q}(i; B)$，耗时 $|B|$。如果分布是静态的，那么在实践中 $|B|$ 会很小。然而，如果存在概念漂移，也就是说，序列越靠后 1 越多，那么 $|B|$ 可能会显著增加。Calders 等人 [8] 指出，在处理长度为 n 的二元序列时，块的数量 $|B| \in \mathcal{O}\left(n^{2/3}\right)$。接下来的两节将介绍如何更快地求解 ChangeBlock。

不过，新条目到达时，我们还需要维护块序列。幸运的是，有一种高效的更新算法，例如参见 [8]。假设我们已经观察到 n 个条目，并且有一个由边界点诱导、包含 k 个块的块序列 B。假设有一个新条目 s_{n+1}。我们将第 $(k+1)$ 个块 (u_{k+1}, v_{k+1}) 加入 B，其中 $u_{k+1} = [s_{n+1} = 1]$ 且 $v_{k+1} = [s_{n+1} = 0]$。然后，我们检查 $\mathsf{av}(k+1, k+1) \leq \mathsf{av}(k, k)$，也就是说，检查最后一个块的平均值是否小于或等于倒数第二个块的平均值。如果是，则合并这两个块并重复检查。该算法能够正确维护边界点，且摊销运行时间为 $\mathcal{O}(1)$。

值得一提的是，边界索引也与保序回归有关（例如参见 [16], ）。也就是说，如果对序列 S 进行保序回归拟合，那么边界点就是拟合曲线发生变化的点。事实上，更新算法对应于相邻违背者合并算法（PAVA），这是一种用于求解保序回归的方法 [16]。

4 固定参数下寻找最优变点

本节将展示，如果模型参数已知且固定，我们就能在对数时间内找到最优变点。

首先，让我们扩展 $\mathsf{q}(\cdot)$ 的定义，使其能够处理固定参数。

定义5。给定块序列 B、索引 i 和两个参数 p_1 andp_2，我们定义

$$q(i; p_1, p_2, B) = \ell(a_1, b_1; p_1) + \ell(a_2, b_2; p_2) - \ell(a, b)\,,$$

其中 $(a_1, b_1) = B[1;\ i-1]$，$(a_2, b_2) = B[i;\ k]$，且 $a = a_1 + a_2$ 和 $b = b_1 + b_2$。

现在我们可以定义固定参数下的优化问题。

问题 4。给定块序列 B、两个参数 $0 \leq p_1 < p_2 \leq 1$，求使 $q(i;\ p_1, p_2, B)$ 最大化的 i。

设 i^* 为问题 4 的解。事实证明，我们可以构造一个数列，下面将其称为 d_j，使得 $d_j > 0$ 当且仅当 $j < i^*$。这使我们能够使用二分查找来找到 i^*。

命题 2。假设块序列 $B = \langle (u_j, v_j) \rangle$ 和两个参数$0 \leq p_1 < p_2 \leq 1$。定义

$$d_j = \ell(u_j, v_j, p_1) - \ell(u_j, v_j, p_2) \quad .$$

那么存在一个指标 i，使得 $d_j > 0$ 当且仅当 $j < i$。此外，指标 i 是问题 4 的解。

证明。我们先证明 i 的存在性。设 $t_j = u_j + v_j$，并记 $X = \log p_1 - \log p_2$ 和 $Y = \log(1 - p_1) - \log(1 - p_2)$。那么

$$\frac{d_j}{t_j} = \frac{u_j}{t_j} X + \frac{v_j}{t_j} Y = \frac{u_j}{t_j} X + Y - \frac{u_j}{t_j} Y = \frac{u_j}{t_j}(X - Y) + Y \quad .$$

由于 B 是一个块序列，分数 u_j/t_j 是递增的。由于 $X < 0$ 且 $Y > 0$，我们有 $X - Y < 0$，因此 d_j/t_j 是递减的。由于 d_j 和 d_j/t_j 同号，存在一个满足命题条件的指标 i。

为证明 i 的最优性，首先注意到

$$d_j = q(j+1; p_1, p_2, B) - q(j; p_1, p_2, B) \quad .$$

设 i^* 是问题 4 的一个解。若 $i < i^*$。那么

$$q(i^*; p_1, p_2, B) - q(i; p_1, p_2, B) = \sum_{j=i}^{i^*-1} d_j \leq 0,$$

这证明了 i 的最优性。对于 $i > i^*$ 的情形，证明类似。 □

命题 2 表明，我们可以使用二分查找在$\mathcal{O}(\log |B|) \in \mathcal{O}(\log n)$ 时间内求解问题 4。我们将此算法称为 FindSegment(p_1, p_2, B)。

5 选择模型参数

我们已经证明，如果已知最优的 p_1 和 p_2，那么可以使用上一节所述的二分查找来找到变点。我们的主要思路是为 p_1 和 p_2 测试若干候选项，使其中一个候选项接近最优参数，从而获得近似保证。

假设给定一个块序列 B，并选取一个变点 i。令 $(a_1, b_1) = B[1;\ i-1]$，$(a_2, b_2) = B[i;\ k]$，$a = a_1 + a_2$，$b = b_1 + b_2$为计数。我们可以将式 1 中给出的目标函数改写为

$$\begin{aligned} q(i; B) &= \ell(a_1, b_1) + \ell(a_2, b_2) - \ell(a, b) \\ &= \left(\ell(a_1, b_1, p_1) - \ell(a_1, b_1, q)\right) + \left(\ell(a_2, b_2, p_2) - \ell(a_2, b_2, q)\right), \end{aligned} \tag{2}$$

其中模型参数为 $p_1 = a_1/(a_1 + b_1)$、$p_2 = a_2/(a_2 + b_2)$ 和 $q = a/(a + b)$。

式 2 中的得分分为两部分，第一部分取决于 p_1，第二部分取决于 p_2。我们首先只关注第二部分的估计。首先，我们来说明在保持对数似然比良好的同时，p_2 可以变化多大。

命题 3。假设 a、$b > 0$，且令 $p = a/(a + b)$。假设 $0 < q \leq p$。还假设 $\epsilon > 0$。定义 $h(x) = \ell(a, b;\ x) - \ell(a, b;\ q)$。假设存在 r 使得

$$\log q + (1 - \epsilon)(\log p - \log q) \leq \log r \leq \log p \quad . \tag{3}$$

那么 $h(r) \geq (1 - \epsilon)h(p)$。

证明。定义 $f(u) = h(\exp u)$。我们声称 f是凹的。为证明这一点，注意到 f 的导数等于

$$f'(u) = a - b\frac{\exp u}{1 - \exp u} \quad .$$

因此，当 $u < 0$时，f'递减，这就证明了 f的凹性。

定义 $c = \frac{\log r - \log q}{\log p - \log q}$。由式 3 可知 $1 - \epsilon \leq c$。$f(u)$的凹性以及 $h(q) = 0$ 这一事实意味着

$$h(r) = f(\log r) \geq f(\log q) + c\left[f(\log p) - f(\log q)\right] = ch(p) \geq (1 - \epsilon)h(p),$$

这就证明了该命题。 □

我们可以按以下方式使用该命题。假设有一个包含 k 个条目的块序列B。令 i^* 为最优变点，p_1^* 和 p_2^* 为相应的最优参数。首先，令

$$P = \{av(i, k) \mid i = 1, \ldots, k\}$$

是候选模型参数的集合。我们知道，最优模型参数 $p_2^* \in P$。我们不会逐一测试每个 $p \in P$，而是构建一个索引

集合 C，并定义 $R = \{\textsf{av}(i,k) \mid i \in C\}$，使得对于每个 $p \in P$，都存在 $r \in R$，使式 3 成立。命题 3 表明，测试 R 中的参数，可以得到式 2 右侧第二部分的 $(1-\epsilon)$ 近似。

我们希望保持集合 C 较小，因此，为了生成 C，我们从 $i = 1$开始，并设置 $C = \{i\}$。接着，我们查看使用 $\textsf{av}(i,k)$ 可以估计 P 的多少个值，也就是说，寻找使式 3 不成立的最小索引。我们将该索引设为 i，将其加入 C，然后重复此过程。我们将这一过程称为 FindCands(B,ϵ)。FindCands的详细伪代码见算法 1。

算法 1： FindCands(B,ϵ)，给定一个包含 k 个条目的块序列 B 以及估计要求 $\epsilon > 0$，构造一个候选索引集 C，用于估计模型参数 p_2。

```
C ← {1}; i ← 1; q ← av(1, k);
当 i < k 时执行
    ρ ← (对数 av(i, k) − 对数 q)/(1 − ε);
    i ← 满足 log av(j, k) − log q > ρ 的最小索引 j，若 j 不存在则为 k;
    将 i 添加到 C;
返回 C;
```

命题 4. 假设块序列 B 有 k 个条目，且令 $\epsilon > 0$。设 $P = \{\textsf{av}(i,k) \mid i = 1,\ldots,k\}$。令 $C = \textsc{FindCands}(B,\epsilon)$，并令 $R = \{\textsf{av}(i,k) \mid i \in C\}$。那么对于每个 $p \in P$，都存在 $r \in R$，使得等式 3 成立。

证明。令 $p \in P \setminus R$。这只有在存在一个较小的值 $r \in R$，使得 $(1-\epsilon)(\log p - \log q) < \log r - \log q$成立时才可能。□

在 FindCands 中查找下一个索引 i，可以通过二分查找在 $\mathcal{O}(\log |B|)$ 时间内完成。因此，FindCands 的运行时间为 $\mathcal{O}(|C| \log n)$。下一个结果表明 $|C| \in \mathcal{O}\left(\epsilon^{-1}\log n\right)$，这使得FindCands 的计算复杂度降至 $\mathcal{O}\left(\epsilon^{-1}\log^2 n\right)$。

命题 5。假设块序列 B 有 k 个条目，这些条目由具有 n 个条目的二进制序列 S 生成，并令 $\epsilon > 0$。令 $P = \{\textsf{av}(i,k) \mid i = 1,\ldots,k\}$。假设递增序列 $R = (r_i) \subseteq P$。令 $q = \textsf{av}(1,k)$。如果

$$\log q + (1-\epsilon)(\log r_i - \log q) > \log r_{i-1}, \tag{4}$$

然后 $|R| \in \mathcal{O}\left(\frac{\log n}{\epsilon}\right)$。

证明。我们可以将公式 4 重写为 $(1-\epsilon)(\log r_i - \log q) > \log r_{i-1} - \log q$，这便自动意味着

$$(1-\epsilon)^i(\log r_{i+2} - \log q) > \log r_2 - \log q \quad .$$

为了给右侧构造下界，令 $r_2 = x/y$ 和 $q = u/v$，其中x、 y、 u 和 v 都是整数，且满足 y、 $v \leq n$。注意到 $r_2 > q$，否则当 $i = 2$ 时就会违反公式 4。因此，我们有 $xv \geq uy + 1$。接着

$$\log r_2 - \log q = \log xv - \log uy \geq \log(uy+1) - \log uy = \log(1 + \frac{1}{uy})$$
$$\geq \log(1 + \frac{1}{n^2}) \geq \frac{n^{-2}}{1+n^{-2}} = \frac{1}{1+n^2} \quad .$$

我们还可以用以下表达式对左端进行上界估计

$$\log r_{i+2} - \log q \leq \log 1 - \log u/v = \log v/u \leq \log n \quad .$$

结合前面的三个不等式可得

$$\log n \geq \log r_{i+2} - \log q > \frac{\log r_2 - \log q}{(1-\epsilon)^i} \geq \frac{1}{(1-\epsilon)^i}\frac{1}{1+n^2} \quad .$$

解出 i,

$$i \leq \frac{\log(1+n^2) + \log\log n}{\log \frac{1}{1-\epsilon}} \leq \frac{\log(1+n^2) + \log\log n}{\epsilon} \in \mathcal{O}\left(\frac{\log n}{\epsilon}\right),$$

证明完成。 □

现在我们可以近似求出 p_2^*。接下来，我们将说明如何为 p_1^* 找到类似的值。请注意，我们不能直接使用之前的结果，因为我们在命题 3 中假设了 $p \geq q$ 。不过，只需在 S 中交换标签即可解决这个问题。

命题 6. 假设 a、 $b > 0$，且 $p = a/(a+b)$。假设 q满足 $0 < p \leq q$。还假设 $\epsilon > 0$。定义 $h(x) = \ell(a, b; x) - \ell(a, b; q)$。假设存在 r，使得

$$\log(1-q) + (1-\epsilon)(\log(1-p) - \log(1-q)) \leq \log(1-r) \leq \log(1-p) \quad . \quad (5)$$

那么 $h(r) \geq (1-\epsilon)h(p)$。

证明。令 $a' = b$、 $b' = a$、 $q' = 1-q$ 和 $r' = 1-r$。将命题 3 应用于这些变量即可立即得到该命题。□

命题 6 导出了一个类似于 FindCands 的算法，用于生成 p_1^* 的候选项。我们将此算法称为 FindCands′，见算法 2。

假设我们已经计算出两组候选索引 C_1 和 C_2；第一组用于估计 p_1^*，第二组用于估计 p_2^*。最后一步是确定应该检查哪些参数组合。一个简单的方法是测试所有可能的组合。这会产生 $\mathcal{O}(|C_1||C_2|)$ 次测试。

然而，由于 p_1^* 和 p_2^* 是由同一个变点 i^*诱导的，我们可以设计一种更高效的方法，只需进行 $\mathcal{O}(|C_1| + |C_2|)$ 次测试。为此，我们首先合并两个候选集 $C = C_1 \cup C_2$。对于每个

算法 2：FindCands′(B, ϵ)，给定一个由 k 个条目组成的块序列 B 以及估计要求 $\epsilon > 0$，构造候选索引集 C，用于估计模型参数 p_1。

```
C ← {k}; i ← k; q ← av(1, k);
当 i > 1 时
    ρ ← (log(1 − av(1, i − 1)) − log(1 − q))/(1 − ϵ);
    i ← 满足 log(1 − av(1, j − 1)) − log(1 − q) > ρ 的最大索引 j，若 j 不存在则取 1;
    将 i 添加到 C;
返回 C;
```

索引 $c_i \in C$，我们计算得分 q(c_i; B)。此外，如果 c_{i-1} 和 c_i 之间存在未包含在 C中的块，也就是说，$c_{i-1} + 1 < c_i$，我们就设置p_1 =av(1, $c_i - 1$) 和 p_2 =av (c_{i-1}, k)，计算最优变点j = FindSegment(p_1, p_2, B)，并测试 q(j, B)。完成所有测试后，我们返回得分最高的索引。我们将此算法称为FindChange(B, ϵ)，并在算法 3 中给出其伪代码。

命题 7. FindChange(B, ϵ) 可提供 $(1 - \epsilon)$ 近似保证。

证明。令 i^* 为对应参数 p_1^* 和 p_2^*下的最优值。令 C_1、C_2 和 C 为算法 3 中定义的集合。若 $i^* \in C$，则证毕。假设 $i^* \notin C$。那么存在 $c_{j-1} < i^* < c_j$，因为 1，$k \in C$。令 r_2 =av(c_{j-1}, k)。根据 C_2的定义，r_2 和 p_2^* 满足等式 3。令 r_1 =av(1, $c_j - 1$)。根据 C_1的定义，r_1 和 p_1^* 满足等式 5。令 i 为 r_1 和 r_2的最优变点，即 i = FindSegment(r_1, r_2, B)。

命题3和命题6结合等式 2 可推出

$$q(i; B) \geq q(i; r_1, r_2, B) \geq q(i^*; r_1, r_2, B) \geq (1 - \epsilon) q(i^*; B) \quad .$$

证明至此完成。 □

本节最后分析计算复杂度。两次调用 FindCands 需要 $\mathcal{O}(\epsilon^{-1} \log^2 n)$ 的时间。列表 C 包含 $\mathcal{O}(\epsilon^{-1} \log n)$ 个条目，对每个 $c \in C$ 单独调用一次 FindSegment 需要 $\mathcal{O}(\log n)$ 的时间。因此，FindChange 的运行时间为 $\mathcal{O}(\epsilon^{-1} \log^2 n)$。

6 相关工作

针对流式场景中的变化检测，人们提出了许多技术。我们将重点介绍其中的一些。若要了解更全面的情况，读者可参阅 Aminikhanghahi 和 Cook 的综述 [2], ，以及 Basseville 和Nikiforov 的著作 [5]。

变点检测的一种标准方法是将存储数据拆分为两个片段，并对这两个片段进行比较；如果两个片段不同，

算法 3： FindChange(B, ϵ)，可得 $(1-\epsilon)$ 近似保证——证 ChangeBlock。

```
C_2 ← FindCands(B, ε);
C_1 ← FindCands′(B, ε);
C ← C_1 ∪ C_2;
对每个 c_j ∈ C 执行
    测试 q(c_j; B);
    如果 c_{j-1} + 1 < c_j，则
        r_1 ← av(1, c_j − 1);
        r_2 ← av(c_{j-1}, k);
        i ← FindSegment(r_1, r_2, B);
        测试 q(i; B);
    返回得分 q 最佳的索引 i* (i; B)，该索引从经过测试的索引中选出;
```

那么就发生了变化。Bifet 和 Gavalda [7] 提出了一种自适应滑动窗口方法：如果当前窗口中存在一种拆分，使得两部分的平均值差异足够大，那么就从窗口中移除较早的部分。Nishida 和 Yamauchi [17] 使用统计检验，将近期样本的准确率与总体准确率进行比较。Kifer 等人 [15] 提出了一系列分布间距离，并在变点检测的背景下对其进行了分析。Kawahara 和 Sugiyama [14] 没有显式地对片段进行建模，而是提出直接估计密度比。Dries 和 R¨uckert [9]研究了将多变量流转换为单变量流，以辅助变点检测。Harel 等人 [13] 通过比较测试段中的损失与置换序列中的相似损失来检测变化。

Ross 等人 [18] 使用指数衰减，将近期样本的表现与整体表现进行比较，而不是显式地对变点建模。Baena-Garcia 等人 [4], 、Gama 等人 [10] 提出了一种检测方法：将当前平均值和标准差与观测到的最小平均值和标准差进行比较。同样无需显式划分，Adams 和 MacKay [1]提出了一种贝叶斯方法，用于对距上次变点的时间进行建模。

变点检测的离线版本称为分段。给定一个由条目组成的序列和预算 k，目标是将序列划分为k ，使某个代价函数最小化。如果全局目标函数是各个片段代价之和，那么可以采用经典的动态规划方法在 $\mathcal{O}(n^2k)$ 时间内求解该问题 [6] 。由于这可能过于缓慢，人们提出了能够提供近似保证的加速技术 [11, 20, 21]。如果代价函数基于单参数对数线性模型，那么在实践中可以显著加速分段问题 [19], ，尽管最坏情况下的运行时间仍为 $\mathcal{O}(n^2k)$。Guha 和 Shim [12] 证明，如果目标函数是各个片段代价的最大值，那么只需对各个片段代价进行 $\mathcal{O}(k^2\log^2 n)$ 次求值，即可计算出精确解。

7 实验评估

在实验中，我们重点分析近似保证 ϵ以及参数 τ的影响。[1] [2] 我们将使用合成序列。此外，我们还将通过网络交通数据进行一个小型案例研究。

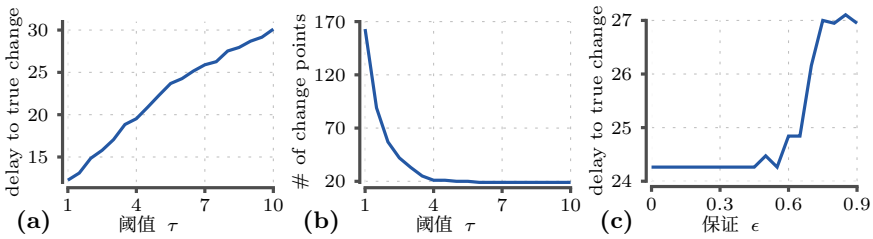


图 1. 变点检测统计量随阈值参数 τ 和近似保证 ϵ *Step* 变化：(a) 发现真实变点的平均延迟（$\epsilon = 0$），(b) 发现的变点数量（$\epsilon = 0$），以及 (c) 发现真实变点的平均延迟（$\tau = 6$）。请注意，在 *Step* 中有 19 个真实变点。对于 $\tau = 0.5$，算法发现真实变点的平均延迟为 1.42，但报告了 46366 个变点（由于缩放问题，省略了这些数值）。

合成序列：我们生成了 3 个合成序列，每个序列长度为 200 000。为简便起见，我们用 Bern(p) 表示成功概率为 p 的伯努利随机变量。第一个序列名为 Ind，由 200 000 个来自 Bern($1/2$) 的样本组成，也就是公平硬币抛掷。第二个序列名为 Step，由 10 000 个来自 Bern($1/4$) 的样本，后接 10 000 个来自 Bern($3/4$) 的样本组成，并重复 10 次。第三个序列名为 Slope，包含 10 个片段，每个片段由 10 000 个来自 Bern(p) 的样本组成，其中 p 从 $1/4$ 线性递增至 $3/4$；随后是 10 000 个来自 Bern(p) 的样本，其中 p 从 $3/4$ 线性递减至 $1/4$。此外，我们还生成了 10 个序列，统称为 Hill。这些序列的长度从 100 000 到 1 000 000 不等，增量为 100 000。每个序列由来自 Bern(p) 的样本组成，其中 p 从 $1/4$ 线性递增至 $3/4$。

结果：我们首先研究阈值参数 τ 的影响。这里，我们使用了 Step 序列；该序列有 19 个真实变点。在图1a中，我们展示了发现真实变点的平均延迟，即每次真实变化发生后，平均需要多少个数据项才能检测到变化。在图1b中，我们还展示了检测到的变点数量：理想情况下，我们应该只检测到 19 个点。在这两项实验中，我们都设置了 $\epsilon = 0$。结果表明，延迟随 τ 线性增长，而当 τ 较小时，虚假变点的数量相当多；不过，随着 τ

[1] 回想一下，如果变化幅度大于 $\sigma = \tau + \log n$，我们就认为发生了 Change。
[2] 实现可在 https://version.helsinki.fi/dacs/ 获取。

增大，虚假变点数量会迅速减少。当 $\tau = 6$ 时，我们检测到了理想的 19 个变点。其余实验中我们将使用这个值。

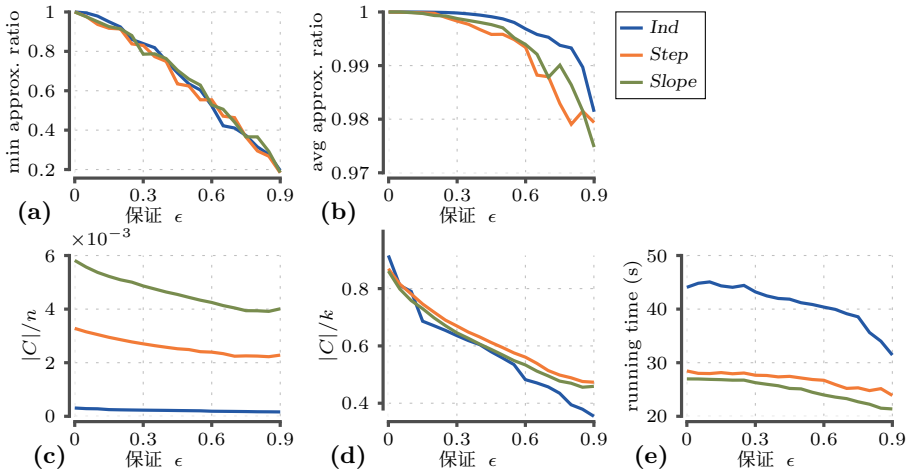


图 2. 合成数据上作为近似保证 ϵ 函数的性能指标。纵轴如下：(a) 比率 FindChange(B, ϵ)/OPT的最小值，(b) 比率 FindChange(B, ϵ)/OPT的平均值，(c) 测试的候选项数量 / 窗口大小（注意， y 轴经过缩放），(d) 测试的候选项数量 / 块数量，以及 (e) 以秒为单位的运行时间。

下一步，我们将研究合成数据上结果质量作为 ϵ 函数的变化。这里我们测量得分 $g = \text{FindChange}(B, \epsilon)$ 与 OPT $= \text{FindChange}(B, 1)$ 的比率，即解相对于 Change 的得分。请注意，我们纳入了所有测试，而不仅仅是那些最终宣布发生变点的测试。图 2a 展示了我们遇到的最小比率随 ϵ 的变化，图 2b 展示了平均比率随 ϵ 的变化。我们从图 2a 中可以看到，最坏情况随 ϵ 呈线性变化。如命题 7 所保证的，最坏情况比率保持在 $(1-\epsilon)$ 以上。虽然最坏情况相对接近其理论界限，但图 2b 所示的平均情况表现显著更好，即使在 $\epsilon = 0.9$ 时，平均比率也高于 0.97。 ϵ 对实际变点检测的影响如图 1c 所示。由于我们可能错过最优值，检测器会变得更加保守，从而增加发现真实变点的延迟。然而，即使在 $\epsilon = 0.9$ 时，这种增幅也很小（仅约 10%）。

接下来，我们将研究运行时间的加速效果。图 2c 展示了所执行的测试次数与 n（即最后一个变点之后的条目数）之间的关系，并以 ϵ 为函数进行呈现。从结果中可以看出，与朴素的 $\mathcal{O}(n)$ 方法相比，加速效果显著；所需的测试次数减少了 2–3 个数量级。这一减少的主要原因在于

边界点。使用 FindCands 带来的减少情况如图 2d 所示。可以看到，候选项的数量随 ϵ 线性减少；当 ϵ 取较大值时，候选项数量大约减少一半。运行时间（秒）见图 2e。正如预期，运行时间随 ϵ 增大而减少。

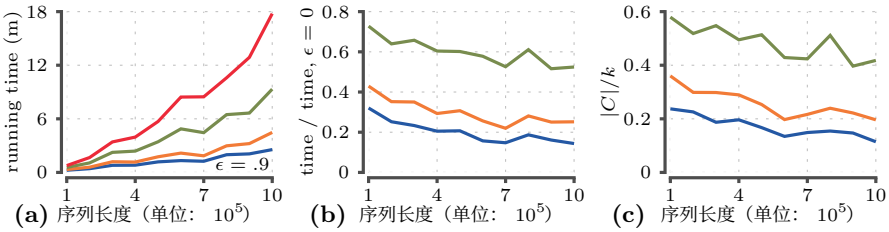


图 3. *Hill* 个序列的计算指标随序列长度的变化：(a) 运行时间（分钟），(b) 运行时间 / $\epsilon = 0$的运行时间，以及 (c) 测试的候选项数量 / 块数。请注意，$\epsilon = 0$ 等价于测试每个边界索引。

虽然加速的主要原因是使用边界索引，但在某些情况下，使用 FindCands 也会变得至关重要。当边界索引的数量增加时，就会出现这种情况。我们通过图 3 所示的 Hill 序列来说明这一效应。这里，为了便于说明，我们提高了变点检测的阈值 τ，使得整个过程中都不会检测到变点。大量概率缓慢递增的条目会产生许多边界点，这表现为 $\epsilon = 0$的运行时间快速增加。此外，FindCands 测试的候选项与块数之比，以及运行时间，都会随着序列规模增大而下降。

交通数据应用案例：我们将变化检测算法应用于 Amit 等人 [3] 收集的交通数据 Network2。该数据包含数周内许多主机之间观测到的连接，并按 10 分钟的时段分组。我们仅使用了 24.12–29.12 期间收集的数据，因为相邻时段存在明显的每小时周期性伪影。随后，我们将收集到的数据转换为二进制序列：如果连接与 SSL 相关，则设为 1，否则设为 0。该序列包含 282754 个条目，分为 743 个 10 分钟时段。我们的算法（$\epsilon = 0$，$\tau = 6$）检测到 12 个变点，如图 4 所示。这些模式显示了非 SSL 连接的短暂突发。一个例外是索引 300 之后的变化，此处原先较高的 SSL 活动恢复为正常状态。

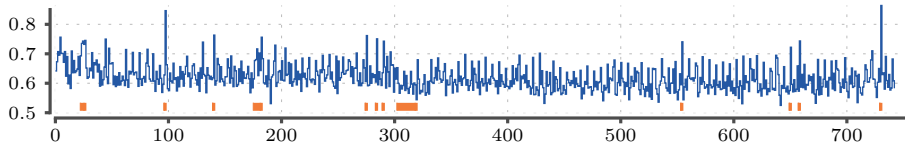


图 4. *Network2* 交通数据中非 SSL 连接占比随时间的变化情况，按 10 分钟为一个时间段。柱条表示变点：柱条末端表示检测到变化的时间，起始端表示最优分割点。

8 结论

本文提出了一种针对二元数据流的变点检测方法，通过在当前窗口中寻找使似然比最优的分割点来进行检测。寻找最优分割点需要 $\mathcal{O}(n)$ 时间，因此，为了使该方法具有实用性，我们引入了一种近似方案，可在 $\mathcal{O}\left(\epsilon^{-1}\log^2 n\right)$ 时间内实现（$1-\epsilon$）近似。该方案利用边界点实现，这一思路借鉴自对数线性模型的分段；随后，通过忽略与相似块相邻的索引，进一步减少候选项。

大多数情况下，边界数量都很少，只有边界数量开始增加时才需要进行额外剪枝。这说明采用混合方法是合理的：边界数量较少时，我们遍历边界；只有当边界数量增加时，才切换到近似技术。

需要指出的是，尽管运行时间是多对数级的，空间需求最坏情况下为 $\mathcal{O}\left(n^{2/3}\right)$。只需删除较早的边界点即可解决这一问题，但这样做可能导致答案不是最优的。未来一个有趣的研究方向是，在不牺牲近似保证的前提下，探究如何降低空间复杂度。

本文仅关注二元数据流。同一概念也有望适用于其他类型的数据，例如整数或实数值。这里的瓶颈在于命题 5，因为它依赖于底层数据流为二元这一事实。将这些结果应用于其他数据类型将留作未来工作。

参考文献

[1] Adams, R.P., MacKay, D.J.：贝叶斯在线变点检测。技术报告，剑桥大学，英国剑桥（2007）[2] Aminikhanghahi, S., Cook, D.J.：时间序列变点检测方法综述。《Knowledge and Information Systems》51(2)，339–367（2017年5月）[3] Amit, I., Matherly, J., Hewlett, W., Xu, Z., Meshi, Y., Weinberger, Y.：网络安全中的机器学习——问题、挑战与数据集。载于：《AAAI-19 可信赖与安全机器学习系统工程研讨会》（2019）

[4] Baena-Garcia, M., Campo-Avila, J.D., Fidalgo, R., Bifet, A., Gavalda, R., Morales-Bueno, R.：早期漂移检测方法。载于：第4届数据流知识发现国际研讨会（2006）[5] Basseville, M., Nikiforov, I.V.：《突变检测——理论与应用》。Prentice-Hall（1993）[6] Bellman, R.：利用动态规划通过线段近似曲线。《ACM通讯》4(6)，284–284（1961）[7] Bifet, A., Gavalda, R.：通过自适应窗口从时变数据中学习。载于：SIAM国际数据挖掘会议，第443–448页（2007）[8] Calders, T., Dexters, N., Goethals, B.：使用灵活窗口挖掘数据流中的频繁项。《Intell. Data Anal.》12(3)，293–304（2008）[9] Dries, A., R¨uckert, U.：自适应概念漂移检测。《Stat. Anal. Data Min.》2(5–6)，311–327（2009）[10] Gama, J., Medas, P., Castillo, G., Rodrigues, P.：带漂移检测的学习。载于：巴西人工智能研讨会，第286–295页（2004）[11] Guha, S., Koudas, N., Shim, K.：直方图构造问题的近似与流式算法。《ACM数据库系统汇刊》31(1)，396–438（2006）[12] Guha, S., Shim, K.：关于最大误差直方图线性时间算法的简注。《IEEE知识与数据工程汇刊》19(7)，993–997（2007）[13] Harel, M., Mannor, S., El-Yaniv, R., Crammer, K.：通过重采样检测概念漂移。载于：第31届机器学习国际会议论文集，第1009–1017页。ICML（2014）[14] Kawahara, Y., Sugiyama, M.：基于直接密度比估计的序贯变点检测。《统计分析与数据挖掘》5，114–127（2012）[15] Kifer, D., Ben-David, S., Gehrke, J.：检测数据流中的变化。载于：第13届超大型数据库国际会议论文集，第180–191页。VLDB（2004）[16] de Leeuw, J., Hornik, K., Mair, P.：R语言中的保序优化：相邻池违背者算法（PAVA）与活动集方法。《统计软件杂志》，论文集32(5)，1–24（2009）[17] Nishida, K., Yamauchi, K.：使用统计检验检测概念漂移。载于：第10届发现科学国际会议论文集，第264–269页（2007）[18]Ross, G.J., Adams, N.M., Tasoulis, D.K., Hand, D.J.：用于检测概念漂移的指数加权移动平均控制图。《Pattern Recognition Letters》33(2)，191–198（2012）[19] Tatti, N.：使用对数线性模型进行快速序列分割。《数据挖掘与知识发现》27(3)，421–441（2013）[20]Tatti, N.：用于分段的强多项式时间高效近似方案。《Inf. Process. Lett.》142，1–8（2019），https://doi.org/10.1016/j.ipl.2018.09.007[21] Terzi, E., Tsaparas, P.：序列分割的高效算法。载于：第6届SIAM国际数据挖掘会议（SDM）论文集，第316–327页（2006）

命题1的证明

在证明该命题之前，我们需要一些额外的记号。记作

$$\Delta(i; p_1, p_2) = \ell(a_1, b_1; p_1) + \ell(a_2, b_2; p_2) \quad .$$

若 $p_1 = a_1/(a_1 + b_1)$ 和 $p_2 = a_2/(a_2 + b_2)$，我们将在记号中省略它们，改写为 $\Delta(i)$。注意，对于任意 r_1 和 r_2，$\Delta(i) \geq \Delta(i; r_1, r_2)$。

证明。令 i 为 ChangeInc 的一个解。若存在并列情况，则令 i 为产生最优解的最小索引。

令 $p_1 \leq p_2$ 为两个伯努利变量对应的参数，$\Delta(i) = \Delta(i; p_1, p_2)$。

假设 $p_1 = 0$。若 $s_i = 0$，则易于证明

$$\Delta(i) \leq \Delta(i + 1; p_1, p_2) \leq \Delta(i + 1) \quad .$$

重复这一论证，我们可以证明存在 i'，使得 $\Delta(i') \geq \Delta(i)$。此外，对于任意 $j < i$，$s_{i'} = 1$ 且 $s_j = 0$（可以放心地假设 S 的取值非零）。这使得 i' 成为一个边界索引。因此，我们可以放心地假设 $p_1 > 0$，类似地也有 $p_2 < 1$。

若 i 是边界索引，则结论成立。假设 i 不是边界索引，也就是说，存在两个整数 $x < i < y$，使得

$$\frac{1}{i - x} \sum_{j=x}^{j-1} s_j \geq \frac{1}{y - i} \sum_{j=i}^{y-1} s_j \quad .$$

我们将不等式 d_1 的左侧记为 d_2，右侧记为

定义

$$z_1 = \log(1 - p_1),\ z_2 = \log(1 - p_2),\ \alpha_1 = \log \frac{p_1}{1 - p_1},\ \text{and}\ \alpha_2 = \log \frac{p_2}{1 - p_2} \quad .$$

现在可以将 $\Delta(k; p_1, p_2)$ 写成

$$\Delta(k; p_1, p_2) = (k - 1)z_1 + \alpha_1 \sum_{j=1}^{k-1} s_j + (n - k + 1)z_2 + \alpha_2 \sum_{j=k}^{n} s_j,$$

对于任意索引 k。现在我们可以将两个得分之差写为

$$\Delta(y; p_1, p_2) - \Delta(i; p_1, p_2) = (z_1 - z_2)(y - i) + (\alpha_1 - \alpha_2) \sum_{j=i}^{y-1} s_j \quad .$$

将这个差值用 $y - i$ 归一化后，得到

$$\frac{1}{y - i} \left(\Delta(y; p_1, p_2) - \Delta(i; p_1, p_2) \right) = (z_1 - z_2) + (\alpha_1 - \alpha_2) d_2 \quad .$$

同样,

$$\frac{1}{i-x}\left(\Delta(i;p_1,p_2)-\Delta(x;p_1,p_2)\right)=(z_1-z_2)+(\alpha_1-\alpha_2)d_1 \quad .$$

由于 $\Delta(i)$ 是最优的，因此 $\Delta(i)\geq\Delta(y)$。于是

$$\begin{aligned}\frac{1}{i-x}\left(\Delta(i;p_1,p_2)-\Delta(x;p_1,p_2)\right)&=(z_1-z_2)+(\alpha_1-\alpha_2)d_1\\&\leq(z_1-z_2)+(\alpha_1-\alpha_2)d_2\\&=\frac{1}{y-i}\left(\Delta(y;p_1,p_2)-\Delta(i;p_1,p_2)\right)\\&\leq\frac{1}{y-i}\left(\Delta(y)-\Delta(i)\right)\leq 0 \quad .\end{aligned}$$

这里我们用到了这样一个事实： $\alpha_1-\alpha_2\leq 0$ 因为 $p_1\leq p_2$，并且根据定义， $d_2\leq d_1$。换句话说， $\Delta(x)\geq\Delta(x;p_1,p_2)\geq\Delta(i;p_1,p_2)=\Delta(i)$。这违背了 i的极小性，因此 i 必须是一个边界索引。证明至此完成。□