# Fast Likelihood-Based Change Point Detection

Tatti, Nikolaj

Springer International Publishing AG
2020

Tatti , N 2020 , Fast Likelihood-Based Change Point Detection . in U Brefeld , E Fromont , A Hotho , A Knobbe , M Maathuis & C Robardet (eds) , Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2019 . Lecture Notes in Artificial Intelligence , vol. 11906 , Springer International Publishing AG , pp. 662-677 , European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD) , Wurzburg , Germany , 16/09/2019 . https://doi.org/10.1007/978-3-030-46150-8_39

http://hdl.handle.net/10138/354984
https://doi.org/10.1007/978-3-030-46150-8_39

acceptedVersion

# Fast likelihood-based change point detection

Nikolaj Tatti[0000−0002−2087−5360]

HIIT, University of Helsinki, Helsinki, Finland
nikolaj.tatti@helsinki.fi

**Abstract.** Change point detection plays a fundamental role in many real-world applications, where the goal is to analyze and monitor the behaviour of a data stream. In this paper, we study change detection in binary streams. To this end, we use a likelihood ratio between two models as a measure for indicating change. The first model is a single bernoulli variable while the second model divides the stored data in two segments, and models each segment with its own bernoulli variable. Finding the optimal split can be done in $\mathcal{O}(n)$ time, where $n$ is the number of entries since the last change point. This is too expensive for large $n$. To combat this we propose an approximation scheme that yields $(1-\epsilon)$ approximation in $\mathcal{O}\left(\epsilon^{-1}\log^2 n\right)$ time. The speed-up consists of several steps: First we reduce the number of possible candidates by adopting a known result from segmentation problems. We then show that for fixed bernoulli parameters we can find the optimal change point in logarithmic time. Finally, we show how to construct a candidate list of size $\mathcal{O}\left(\epsilon^{-1}\log n\right)$ for model parameters. We demonstrate empirically the approximation quality and the running time of our algorithm, showing that we can gain a significant speed-up with a minimal average loss in optimality.

## 1 Introduction

Many real-world applications involve in monitoring and analyzing a constant stream of data. A fundamental task in such applications is to monitor whether a change has occurred. For example, the goal may be monitoring the performance of a classifier over time, and triggering retraining if the quality degrades too much. We can also use change point detection techniques to detect anomalous behavior in the data stream. As the data flow may be significant, it is important to develop efficient algorithms.

In this paper we study detecting change in a stream of binary numbers, that is, we are interested in detecting whether the underlying distribution has recently changed significantly. To test the change we will use a standard likelihood ratio statistic. Namely, assume that we have already observed $n$ samples from the last time we have observed change. In our first model, we fit a single bernoulli variable to these samples. In our second model, we split these samples in two halves, say at point $i$, and fit two bernoulli variables to these halves. Once this is done we compare the likelihood ratio of the models. If the ratio is large enough, then we deem that change has occurred.

In our setting, index $i$ is not fixed. Instead we are looking for the index that yields the largest likelihood. This can be done naively in $\mathcal{O}(n)$ time by testing each candidate. This may be too slow, especially if $n$ is large enough and we do not have the resources before a new sample arrives. Our main technical contribution is to show how we can achieve $(1-\epsilon)$ approximate of the optimal $i$ in $\mathcal{O}\bigl(\epsilon^{-1}\log^2 n\bigr)$ time.

To achieve this we will first reduce the number of candidates for the optimal index $i$. We say that index $j$ is a *border* if each interval ending at $j-1$ has a smaller proportion of 1s that any interval that starts at $j$. A known result states that the optimal change point will be among border indices. Using border indices already reduces the search time greatly in practice, with theoretical running time being $\mathcal{O}\bigl(n^{2/3}\bigr)$.

To obtain even smaller bounds we show that we can find the optimal index among the border indices for *fixed* model parameters, that is, the parameters for the two bernoulli variables, in $\mathcal{O}(\log n)$ time. We then construct a list of $\mathcal{O}\bigl(\epsilon^{-1}\log n\bigr)$ candidates for these parameters. Moreover, this list will contain model parameters that are close enough to the optimal parameters, so testing them yields $(1-\epsilon)$ approximation guarantee in $\mathcal{O}\bigl(\epsilon^{-1}\log^2 n\bigr)$ time.

The remaining paper is organized as follows. In Section 2 we introduce preliminary notation and define the problem. In Section 3 we introduce border points. We present our main technical contribution in Sections 4–5: first we show how to find optimal index for fixed model parameters, and then show how to select candidates for these parameters. We present related work in Section 6 and empirical evaluation in Section 7. Finally, we conclude with discussion in Section 8.

## 2 Preliminaries and problem definition

Assume a sequence of $n$ binary numbers $S = s_1, \ldots, s_n$. Here $s_1$ is either the beginning of the stream or the last time we detected a change. Our goal is to determine whether a change has happened in $S$. More specifically, we consider two statistical models: The first model $M_1$ assumes that $S$ is generated with a single bernoulli variable. The second model $M_2$ assumes that there is an index $i$, a change point, such that $s_1, \ldots, s_{i-1}$ is generated by one bernoulli variable and $s_i, \ldots, s_n$ is generated by another bernoulli variable.

Given a sequence $S$ we will fit $M_1$ and $M_2$ and compare the log-likelihoods. Note that the model $M_2$ depends on the change point $i$, so we need to select $i$ that maximizes the likelihood of $M_2$. If the ratio is large enough, then we can determine that change has occurred.

To make the above discussion more formal, let us introduce some notation. Given two integers $a$ and $b$, and real number between 0 and 1, we denote the log-likelihood of a bernoulli variable by

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

For a fixed $a$ and $b$, the log-likelihood is at its maximum if $p = a/(a+b)$. In such a case, we will often drop $p$ from the notation and simply write $\ell(a, b)$.

We have the following optimization problem.

*Problem 1 (*CHANGE*).* Given a sequence $S = s_1, \ldots, s_n$, find an index $i$ s.t.

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

is maximized, where

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

Note that CHANGE can be solved in $\mathcal{O}(n)$ time by simply iterating over all possible values for $i$. Such running time may be too slow, especially in a streaming setting when new points arrive constantly, and our goal is to determine whether change has occurred in real time. The main contribution of this paper is to show how to compute $(1-\epsilon)$ estimate of CHANGE in $\mathcal{O}\big(\epsilon^{-1}\log^2 n\big)$ time. This algorithm requires additional data structures that we will review in the next section. As our main application is to search change points in a stream, these structures need to be maintained over a stream. Luckily, there is an amortized constant-time algorithm for maintaining the needed structure, as demonstrated in the next section.

Once we have solved CHANGE, we compare the obtained score against the threshold $\sigma$. Note that $M_2$ will always have a larger likelihood than $M_1$. In this paper, we will use BIC to adjust for the additional model complexity of $M_2$. The model $M_2$ has three parameters while the model $M_1$ has 1 parameter. This leads to a BIC penalty of $(3-1)/2 \log n = \log n$. In practice, we need to be more conservative when selecting $M_2$ due to the multiple hypothesis testing problem. Hence, we will use $\sigma = \tau + \log n$ as the threshold. Here, $\tau$ is a user parameter; we will provide some guidelines in selecting $\tau$ during the experimental evaluation in Section 7.

When change occurs at point $i$ we have two options: we can either discard the current window and start from scratch, or we can drop only the first $i$ elements. In this paper we will use the former approach since the latter approach requires additional maintenance which may impact overall computational complexity.

## 3 Reducing number of candidates

Our first step for a faster change point discovery is to reduce the number of possible change points. To this end, we define a variant of CHANGE, where we require that the second parameter in $M_2$ is larger than the first.

*Problem 2 (*CHANGEINC*).* Given a sequence $S = s_1, \ldots, s_n$, find an index $i$ s.t.

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

is maximized, where

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

with $a_1/(a_1 + b_1) \leq a_2/(a_2 + b_2)$.

From now on, we will focus on solving ChangeInc. This problem is meaningful by itself, for example, if the goal is to detect a deterioration in a classifier, that is, sudden increase in entries being equal to 1. However, we can also use ChangeInc to solve Change. This is done by defining a flipped sequence $S' = s'_1, \ldots, s'_n$, where $s'_i = 1 - s_i$. Then the solution for Change is either the solution of ChangeInc($S$) or the solution of ChangeInc($S'$).

Next we show that we can limit ourselves to *border* indices when solving ChangeInc.

**Definition 1.** *Assume a sequence of binary numbers $S = (s_i)_{i=1}^n$. We say that index $j$ is a* border *index if there are no indices $x, y$ with $x < j < y$ such that*

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

In other words, $j$ is a border index if and only if the average of any interval ending at $j - 1$ is smaller than the average of any interval starting at $j$.

**Proposition 1.** *There is a border index $i$ that solves* ChangeInc.

The proposition follows from a variant of Theorem 1 in [19]. For the sake of completeness we provide a direct proof in Appendix in supplementary material.

We address the issue of maintaining border indices at the end of this section.

The proposition permits us to ignore all indices that are not borders. That is, we can group the sequence entries in blocks, each block starting with a border index. We can then search for $i$ using these blocks instead of using the original sequence.

It is easy to see that these blocks have the following property: the proportion of 1s in the next block is always larger. This key feature will play a crucial role in the next two sections as it allows us to use binary search techniques and reduce the computational complexity. Let us restate the original problem so that we can use this feature. First, let us define what is a block sequence.

**Definition 2.** *Let $B = \langle (u_i, v_i) \rangle_{i=1}^k$ be a sequence of $k$ pairs of non-negative integers with $u_i + v_i > 0$. We say that $B$ is* block sequence *if $\frac{u_{i+1}}{u_{i+1}+v_{i+1}} > \frac{u_i}{u_i+v_i}$.*

We obtain a block sequence $B$ from a binary sequence $S$ by grouping the entries between border points: the counter $u_i$ indicates the number of 1s while the counter $v_i$ indicates the number of 0s.

Our goal is to use block sequences to solve ChangeInc. First, we need some additional notation.

**Definition 3.** *Given a block sequence $B$, we define $B[i;j] = (a,b)$, where $a = \sum_{k=i}^{j} u_k$ and $b = \sum_{k=i}^{j} v_k$. If $i > j$, then $a = b = 0$. Moreover, we will write*

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

*If $B$ is known from the context, we will write $av(i,j)$.*

**Definition 4.** *Given a block sequence $B$, we define the score of a change point $i$ to be*

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

*where $(a_1,b_1) = B[1;i-1]$, $(a_2,b_2) = B[i;k]$, and $a = a_1 + a_2$ and $b = b_1 + b_2$.*

Note that $\ell(a,b)$ is a constant but it is useful to keep since $q(i;B)$ is a log-likelihood ratio between two models, and this formulation allows us to estimate the objective in Section 5.

*Problem 3 (*ChangeBlock*).* Given a block sequence $B$ find a change point $i$ that maximizes $q(i;B)$.

We can solve ChangeInc by maintaining a block sequence induced by the border points, and solving ChangeBlock. Naively, we can simply compute $q(i;B)$ for each index in $\mathcal{O}(|B|)$ time. If the distribution is static, then $|B|$ will be small in practice. However, if there is a concept drift, that is, there are more 1s in the sequence towards the end of sequence, then $|B|$ may increase significantly. Calders et al. [8] argued that when dealing with binary sequences of length $n$, the number of blocks $|B| \in \mathcal{O}\big(n^{2/3}\big)$. In the following two sections we will show how to solve ChangeBlock faster.

However, we also need to maintain the block sequence as new entries arrive. Luckily, there is an efficient update algorithm, see [8] for example. Assume that we have already observed $n$ entries, and we have a block sequence of $k$ blocks $B$ induced by the border points. Assume a new entry $s_{n+1}$. We add $(k+1)$th block $(u_{k+1}, v_{k+1})$ to $B$, where $u_{k+1} = [s_{n+1} = 1]$ and $v_{k+1} = [s_{n+1} = 0]$. We then check whether $av(k+1,k+1) \leq av(k,k)$, that is, whether the average of the last block is smaller than or equal to the average of the second last block. If it is, then we merge the blocks and repeat the test. This algorithm maintains the border points correctly and runs in amortized $\mathcal{O}(1)$ time.

It is worth mentioning that the border indices are also connected to isotonic regression (see [16], for example). Namely, if one would fit isotonic regression to the sequence $S$, then the border points are the points where the fitted curve changes its value. In fact, the update algorithm corresponds to the pool adjacent violators (PAVA) algorithm, a method used to solve isotonic regression [16].

## 4 Finding optimal change point for fixed parameters

In this section we show that if the model parameters are known and fixed, then we can find the optimal change point in logarithmic time.

First, let us extend the definition of $q(\cdot)$ to handle fixed parameters.

**Definition 5.** *Given a block sequence $B$, an index $i$, and two parameters $p_1$ and $p_2$, we define*

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

*where $(a_1, b_1) = B[1; i-1]$, $(a_2, b_2) = B[i; k]$, and $a = a_1 + a_2$ and $b = b_1 + b_2$.*

We can now define the optimization problem for fixed parameters.

*Problem 4.* Given a block sequence $B$, two parameters $0 \leq p_1 < p_2 \leq 1$, find $i$ maximizing $q(i; p_1, p_2, B)$.

Let $i^*$ be the solution for Problem 4. It turns out that we can construct a sequence of numbers, referred as $d_j$ below, such that $d_j > 0$ if and only if $j < i^*$. This allows us to use binary search to find $i^*$.

**Proposition 2.** *Assume a block sequence $B = \langle (u_j, v_j) \rangle$ and two parameters $0 \leq p_1 < p_2 \leq 1$. Define*

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

*Then there is an index $i$ such that $d_j > 0$ if and only if $j < i$. Moreover, index $i$ solves Problem 4.*

*Proof.* Let us first show the existence of $i$. Let $t_j = u_j + v_j$, and write $X = \log p_1 - \log p_2$ and $Y = \log(1 - p_1) - \log(1 - p_2)$. Then

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

Since $B$ is a block sequence, the fraction $u_j / t_j$ is increasing. Since $X < 0$ and $Y > 0$, we have $X - Y < 0$, so $d_j / t_j$ is decreasing. Since $d_j$ and $d_j / t_j$ have the same sign, there is an index $i$ satisfying the condition of the statement.

To prove the optimality of $i$, first note that

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

Let $i^*$ be a solution for Problem 4. If $i < i^*$. Then

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

proving the optimality of $i$. The case for $i > i^*$ is similar. □

Proposition 2 implies that we can use binary search to solve Problem 4 in $\mathcal{O}(\log |B|) \in \mathcal{O}(\log n)$ time. We refer to this algorithm as FindSegment$(p_1, p_2, B)$.

## 5 Selecting model parameters

We have shown that if we know the optimal $p_1$ and $p_2$, then we can use binary search as described in the previous section to find the change point. Our main idea is to test several candidates for $p_1$ and $p_2$ such that one of the candidates will be close to the optimal parameters yielding an approximation guarantee.

Assume that we are given a block sequence $B$ and select a change point $i$. Let $(a_1, b_1) = B[1; i-1]$, $(a_2, b_2) = B[i; k]$, $a = a_1 + a_2$, $b = b_1 + b_2$ be the counts. We can rewrite objective given in Eq. 1 as

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

where the model parameters are $p_1 = a_1/(a_1 + b_1)$, $p_2 = a_2/(a_2 + b_2)$, and $q = a/(a + b)$.

The score as written in Eq. 2 is split in two parts, the first part depends on $p_1$ and the second part depends on $p_2$. We will first focus solely on estimating the second part. First, let us show how much we can vary $p_2$ while still maintaining a good log-likelihood ratio.

**Proposition 3.** *Assume $a, b > 0$, and let $p = a/(a + b)$. Assume $0 < q \leq p$. Assume also $\epsilon > 0$. Define $h(x) = \ell(a, b; x) - \ell(a, b; q)$. Assume $r$ such that*

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

*Then $h(r) \geq (1 - \epsilon)h(p)$.*

*Proof.* Define $f(u) = h(\exp u)$. We claim that $f$ is concave. To prove the claim, note that the derivative of $f$ is equal to

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

Hence, $f'$ is decreasing for $u < 0$, which proves the concavity of $f$.

Define $c = \frac{\log r - \log q}{\log p - \log q}$. Eq. 3 implies that $1 - \epsilon \leq c$. The concavity of $f(u)$ and the fact that $h(q) = 0$ imply that

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

which proves the proposition. □

We can use the proposition in the following manner. Assume a block sequence $B$ with $k$ entries. Let $i^*$ be the optimal change point and $p_1^*$ and $p_2^*$ be the corresponding optimal parameters. First, let

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

be the set of candidate model parameters. We know that the optimal model parameter $p_2^* \in P$. Instead of testing every $p \in P$, we will construct an index

set $C$, and define $R = \{av(i,k) \mid i \in C\}$, such that for each $p \in P$ there is $r \in R$ such that Eq. 3 holds. Proposition 3 states that testing the parameters in $R$ yields a $(1-\epsilon)$ approximation of the second part of the right-hand side in Eq. 2.

We wish to keep the set $C$ small, so to generate $C$, we will start with $i = 1$ and set $C = \{i\}$. We then look how many values of $P$ we can estimate with $av(i,k)$, that is, we look for the smallest index for which Eq. 3 does not hold. We set this index to $i$, add it to $C$, and repeat the process. We will refer to this procedure as FindCands$(B, \epsilon)$. The detailed pseudo-code for FindCands is given in Algorithm 1.

**Algorithm 1:** FindCands$(B, \epsilon)$, given a block sequence $B$ of $k$ entries and an estimation requirement $\epsilon > 0$, constructs a candidate index set $C$ that is used to estimate the model parameter $p_2$.

```
C ← {1}; i ← 1; q ← av(1, k);
while i < k do
    ρ ← (log av(i, k) − log q)/(1 − ε);
    i ← smallest index j s.t. log av(j, k) − log q > ρ, or k if j does not exist;
    add i to C;
return C;
```

**Proposition 4.** *Assume a block sequence $B$ with $k$ entries, and let $\epsilon > 0$. Set $P = \{av(i,k) \mid i = 1, \ldots, k\}$. Let $C =$ FindCands$(B, \epsilon)$, and let $R = \{av(i,k) \mid i \in C\}$. Then for each $p \in P$ there is $r \in R$ such that Eq. 3 holds.*

*Proof.* Let $p \in P \setminus R$. This is only possible if there is a smaller value $r \in R$ such that $(1-\epsilon)(\log p - \log q) < \log r - \log q$ holds. □

Finding the next index $i$ in FindCands can be done with a binary search in $\mathcal{O}(\log |B|)$ time. Thus, FindCands runs in $\mathcal{O}(|C| \log n)$ time. Next result shows that $|C| \in \mathcal{O}\big(\epsilon^{-1} \log n\big)$, which brings the computational complexity of FindCands to $\mathcal{O}\big(\epsilon^{-1} \log^2 n\big)$.

**Proposition 5.** *Assume a block sequence $B$ with $k$ entries generated from a binary sequence $S$ with $n$ entries, and let $\epsilon > 0$. Let $P = \{av(i,k) \mid i = 1, \ldots, k\}$. Assume an increasing sequence $R = (r_i) \subseteq P$. Let $q = av(1,k)$. If*

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

*then $|R| \in \mathcal{O}\Big(\frac{\log n}{\epsilon}\Big)$.*

*Proof.* We can rewrite Eq. 4 as $(1-\epsilon)(\log r_i - \log q) > \log r_{i-1} - \log q$ which automatically implies that

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

To lower-bound the right-hand side, let us write $r_2 = x/y$ and $q = u/v$, where $x$, $y$, $u$, and $v$ are integers with $y, v \leq n$. Note that $r_2 > q$, otherwise we violate Eq. 4 when $i = 2$. Hence, we have $xv \geq uy + 1$. Then

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

We can also upper-bound the left-hand side with

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

Combining the three previous inequalities leads to

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

Solving for $i$,

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

completes the proof. □

We can now approximate $p_2^*$. Our next step is to show how to find similar value for $p_1^*$. Note that we cannot use the previous results immediately because we assumed that $p \geq q$ in Proposition 3. However, we can fix this by simply switching the labels in $S$.

**Proposition 6.** *Assume $a, b > 0$, and let $p = a/(a+b)$. Assume $q$ with $0 < p \leq q$. Assume also $\epsilon > 0$. Define $h(x) = \ell(a, b; x) - \ell(a, b; q)$. Assume $r$ such that*

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

*Then $h(r) \geq (1-\epsilon)h(p)$.*

*Proof.* Set $a' = b$, $b' = a$, $q' = 1 - q$, and $r' = 1 - r$. The proposition follows immediately from Proposition 3 when applied to these variables. □

Proposition 6 leads to an algorithm, similar to FindCands, for generating candidates for $p_1^*$. We refer to this algorithm as FindCands′, see Algorithm 2.

Assume that we have computed two sets of candidate indices $C_1$ and $C_2$; the first set is meant to be used to estimate $p_1^*$, while the second set is meant to be used to estimate $p_2^*$. The final step is to determine what combinations of parameters should we check. A naive approach would be to test every possible combination. This leads to $\mathcal{O}(|C_1||C_2|)$ tests.

However, since $p_1^*$ and $p_2^*$ are induced by the same change point $i^*$, we can design a more efficient approach that leads to only $\mathcal{O}(|C_1| + |C_2|)$ tests. In order to do so, first we combine both candidate sets, $C = C_1 \cup C_2$. For each

**Algorithm 2:** FindCands$'(B, \epsilon)$, given a block sequence $B$ of $k$ entries and an estimation requirement $\epsilon > 0$, constructs a candidate index set $C$ that is used to estimate the model parameter $p_1$.

```
C ← {k}; i ← k; q ← av(1, k);
while i > 1 do
    ρ ← (log(1 − av(1, i − 1)) − log(1 − q))/(1 − ε);
    i ← largest index j s.t. log(1 − av(1, j − 1)) − log(1 − q) > ρ, or 1 if j does
      not exist;
    add i to C;
return C;
```

index $c_i \in C$, we compute the score $q(c_i; B)$. Also, if there are blocks between $c_{i-1}$ and $c_i$ that are not included in $C$, that is, $c_{i-1} + 1 < c_i$, we set $p_1 = av(1, c_i - 1)$ and $p_2 = av(c_{i-1}, k)$, compute the optimal change point $j = \textsc{FindSegment}(p_1, p_2, B)$, and test $q(j, B)$. When all tests are done, we return the index that yielded the best score. We refer to this algorithm as $\textsc{FindChange}(B, \epsilon)$, and present the pseudo-code in Algorithm 3.

**Proposition 7.** $\textsc{FindChange}(B, \epsilon)$ *yields* $(1 - \epsilon)$ *approximation guarantee.*

*Proof.* Let $i^*$ be the optimal value with the corresponding parameters $p_1^*$ and $p_2^*$. Let $C_1$, $C_2$ and $C$ be the sets as defined in Algorithm 3. If $i^* \in C$, then we are done. Assume that $i^* \notin C$. Then there are $c_{j-1} < i^* < c_j$, since $1, k \in C$. Let $r_2 = av(c_{j-1}, k)$. Then $r_2$ and $p_2^*$ satisfy Eq. 3 by definition of $C_2$. Let $r_1 = av(1, c_j - 1)$. Then $r_1$ and $p_1^*$ satisfy Eq. 5 by definition of $C_1$. Let $i$ be the optimal change point for $r_1$ and $r_2$, that is, $i = \textsc{FindSegment}(r_1, r_2, B)$.

Propositions 3 and 6 together with Eq. 2 imply that

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

This completes the proof. □

We complete this section with computational complexity analysis. The two calls of FindCands require $\mathcal{O}\big(\epsilon^{-1} \log^2 n\big)$ time. The list $C$ has $\mathcal{O}\big(\epsilon^{-1} \log n\big)$ entries, and a single call of FindSegment for each $c \in C$ requires $\mathcal{O}(\log n)$ time. Consequently, the running time for FindChange is $\mathcal{O}\big(\epsilon^{-1} \log^2 n\big)$.

## 6 Related work

Many techniques have been proposed for change detection in a stream setting. We will highlight some of these techniques. For a fuller picture, we refer the reader to a survey by Aminikhanghahi and Cook [2], and a book by Basseville and Nikiforov [5].

A standard approach for change point detection is to split the stored data in two segments, and compare the two segments; if the segments are different,

**Algorithm 3:** FindChange$(B, \epsilon)$, yields $(1-\epsilon)$ approximation guarantee for ChangeBlock.

```
C_2 ← FindCands(B, ε);
C_1 ← FindCands'(B, ε);
C ← C_1 ∪ C_2;
foreach c_j ∈ C do
    test q(c_j; B);
    if c_{j-1} + 1 < c_j then
        r_1 ← av(1, c_j − 1);
        r_2 ← av(c_{j-1}, k);
        i ← FindSegment(r_1, r_2, B);
        test q(i; B);
    return index i* having the best score q(i; B) among the tested indices;
```

then a change has happened. Bifet and Gavalda [7] proposed an adaptive sliding window approach: if the current window contains a split such that the averages of the two portions are different enough, then the older portion is dropped from the window. Nishida and Yamauchi [17] compared the accuracy of recent samples against the overall accuracy using a statistical test. Kifer et al. [15] proposed a family of distances between distributions and analyzed them in the context of change point detection. Instead of modeling segments explicitly, Kawahara and Sugiyama [14] proposed estimating density ratio directly. Dries and Rückert [9] studied transformations a multivariate stream into a univariate stream to aid change point detection. Harel et al. [13] detected change by comparing the loss in a test segment against a similar loss in a permuted sequence.

Instead of explicitly modeling the change point, Ross et al. [18] used exponential decay to compare the performance of recent samples against the overall performance. Baena-Garcia et al. [4], Gama et al. [10] proposed a detecting change by comparing current average and standard deviation against the smallest observed average and standard deviation. Also avoiding an explicit split, a Bayesian approach for modeling the time since last change point was proposed by Adams and MacKay [1].

An offline version of change point detection is called *segmentation*. Here we are given a sequence of entries and a budget $k$. The goal is divide a sequence into $k$ minimizing some cost function. If the global objective is a sum of individual segment costs, then the problem can be solved with a classic dynamic program approach [6] in $\mathcal{O}(n^2k)$ time. As this may be too slow speed-up techniques yielding approximation guarantees have been proposed [11, 20, 21]. If the cost function is based on one-parameter log-linear models, it is possible to speed-up the segmentation problem significantly in practice [19], even though the worst-case running time remains $\mathcal{O}(n^2k)$. Guha and Shim [12] showed that if the objective is the maximum of the individual segment costs, then we can compute the exact solution using only $\mathcal{O}(k^2 \log^2 n)$ evaluations of the individual segment costs.

## 7 Experimental evaluation

For our experiments, we focus on analyzing the effect of the approximation guarantee $\epsilon$, as well as the parameter $\tau$.[1] [2] Here we will use synthetic sequences. In addition, we present a small case study using network traffic data.

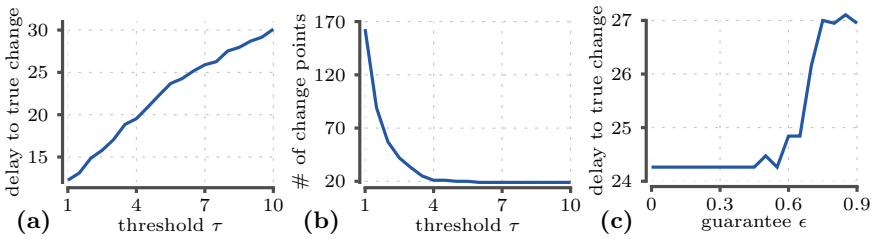


**Fig. 1.** Change point detection statistics as a function of threshold parameter $\tau$ and approximation guarantee $\epsilon$ *Step* data: (a) average delay for discovering a true change point ($\epsilon = 0$), (b) number of discovered change points ($\epsilon = 0$), and (c) average delay for discovering a true change point ($\tau = 6$). Note that in *Step* there are 19 true change points. For $\tau = 0.5$, the algorithm had average delay of 1.42 to a true change point but reported 46 366 change points (these values are omitted due to scaling issues).

**Synthetic sequences:** We generated 3 synthetic sequences, each of length 200 000. For simplicity we will write $Bern(p)$ to mean a bernoulli random variable with probability of 1 being $p$. The first sequence, named *Ind*, consists of 200 000 samples from $Bern(1/2)$, that is, fair coin flips. The second sequence, named *Step*, consists of 10 000 samples from $Bern(1/4)$ followed by 10 000 samples from $Bern(3/4)$, repeated 10 times. The third sequence, named *Slope*, includes 10 segments, each segment consists of 10 000 samples from $Bern(p)$, where $p$ increases linearly from 1/4 to 3/4, followed by 10 000 samples from $Bern(p)$, where $p$ decreases linearly from 3/4 to 1/4. In addition, we generated 10 sequences, collectively named *Hill*. The length of the sequences varies from 100 000 to 1 000 000 with increments of 100 000. Each sequence consists of samples from $Bern(p)$, where $p$ increases linearly from 1/4 to 3/4.

**Results:** We start by studying the effect of the threshold parameter $\tau$. Here, we used *Step* sequence; this sequence has 19 true change points. In Figure 1a, we show the average delay of discovering the true change point, that is, how many entries are needed, on average, before a change is discovered after each true change. In Figure 1b, we also show how many change points we discovered: ideally we should find only 19 points. In both experiments we set $\epsilon = 0$. We see from the results that the delay grows linearly with $\tau$, whereas the number of false change points is significant for small values of $\tau$ but drop quickly as $\tau$

[1] Recall that we say that change occurs if it is larger than $\sigma = \tau + \log n$.
[2] The implementation is available at `https://version.helsinki.fi/dacs/`.

grows. For $\tau = 6$ we detected the ideal 19 change points. We will use this value for the rest of the experiments.

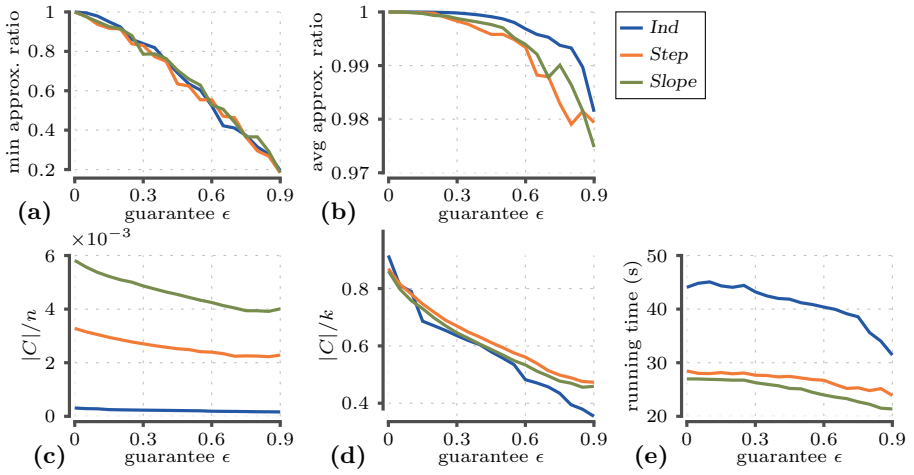


**Fig. 2.** Performance metrics as a function of approximation guarantee $\epsilon$ on synthetic data. Y-axes are as follows: (a) minimum of ratio FindChange$(B, \epsilon)/OPT$, (b) average of ratio FindChange$(B, \epsilon)/OPT$, (c) number of candidates tested / window size (note that $y$-axis is scaled), (d) number of candidates tested / number of blocks, and (e) running time in seconds.

Our next step is to study the quality of the results as a function of $\epsilon$ on synthetic data. Here we measure the ratio of the scores $g =$ FindChange$(B, \epsilon)$ and $OPT =$ FindChange$(B, 1)$, that is, the score of the solution to Change. Note we include all tests, not just the ones that resulted in declaring a change. Figure 2a shows the smallest ratio that we encountered as a function of $\epsilon$, and Figure 2b shows the average ratio as a function of $\epsilon$. We see in Figure 2a that the worst case behaves linearly as a function of $\epsilon$. As guaranteed by Proposition 7, the worst case ratio stays above $(1-\epsilon)$. While the worst-case is relatively close to its theoretical boundary, the average case, shown in Figure 2b, performs significantly better with average ratio being above 0.97 even for $\epsilon = 0.9$. The effect of $\epsilon$ on the actual change point detection is demonstrated in Figure 1c. Since, we may miss the optimal value, the detector becomes more conservative, which increases the delay for discovering true change. However, the increase is moderate (only about 10%) even for $\epsilon = 0.9$.

Our next step is to study speed-up in running time. Figure 2c shows the number of tests performed compared to $n$, the number of entries from the last change point as a function of $\epsilon$. We see from the results that there is significant speed-up when compared to the naive $\mathcal{O}(n)$ approach; the number of needed tests is reduced by 2–3 orders of magnitude. The main reason for this reduction is due

to the border points. Reduction due to using FindCands is shown in Figure 2d. Here we see that the number of candidates reduces linearly as a function of $\epsilon$, reducing the number of candidates roughly by 1/2 for the larger values of $\epsilon$. The running times (in seconds) are given in Figure 2e. As expected, the running times are decreasing as a function of $\epsilon$.

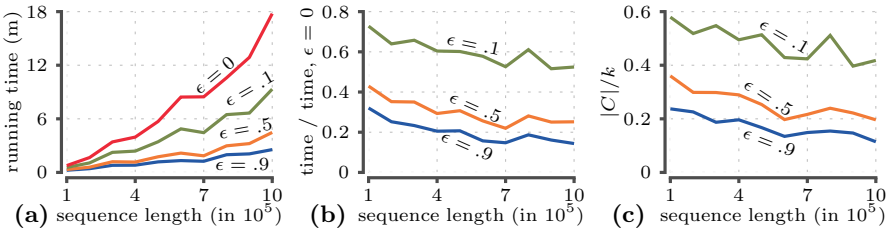


**Fig. 3.** Computational metrics as a function of sequence length for *Hill* sequences: (a) running time in minutes, (b) running time / running time for $\epsilon = 0$, and (c) number of candidates tested / number of blocks. Note that $\epsilon = 0$ is equivalent of testing every border index.

While the main reason for speed-up comes from using border indices, there are scenarios where using FindCands becomes significant. This happens when the number of border indices increases. We illustrate this effect with *Hill* sequences, shown in Figure 3. Here, for the sake of illustration, we increased the threshold $\tau$ for change point detection so that at no point we detect change. Having many entries with slowly increasing probability of 1 yields many border points, which is seen as a fast increase in running time for $\epsilon = 0$. Moreover, the ratio of candidates tested by FindCands against the number of blocks, as well as the running time, decreases as the sequence increases in size.

**Use case with traffic data:** We applied our change detection algorithm on traffic data, *network2*, collected by Amit et al. [3]. This data contains observed connections between many hosts over several weeks, grouped in 10 minute periods. We only used data collected during 24.12–29.12 as the surrounding time periods contain a strong hourly artifact. We then transformed the collected data into a binary sequence by setting 1 if the connection was related to SSL, and 0 otherwise. The sequence contains 282 754 entries grouped in 743 periods of 10 minutes. Our algorithm ($\epsilon = 0$, $\tau = 6$) found 12 change points, shown in Figure 4. These patterns show short bursts of non-SSL connections. One exception is the change after the index 300, where the previously high SSL activity is resolved to a normal behavior.

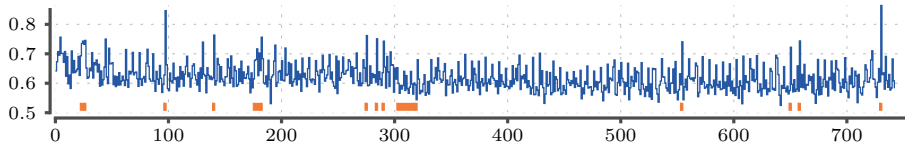


**Fig. 4.** Proportion of non-SSL connections in *Network2* traffic data over time, in 10 minute periods. The bars indicate the change points: the end of the bar indicates when change was discovered and the beginning of the bar indicate the optimal split.

## 8 Conclusions

In this paper we presented a change point detection approach for binary streams based on finding a split in a current window optimizing a likelihood ratio. Finding the optimal split needs $\mathcal{O}(n)$ time, so in order for this approach to be practical, we introduced an approximation scheme that yields $(1-\epsilon)$ approximation in $\mathcal{O}\big(\epsilon^{-1}\log^2 n\big)$. The scheme is implemented by using border points, an idea adopted from segmentation of log-linear models, and then further reducing the candidates by ignoring indices that border similar blocks.

Most of the time the number of borders will be small, and the additional pruning is only required when the number of borders start to increase. This suggests that a hybrid approach is sensible: we will iterate over borders if there are only few of them, and switch to approximation technique only when the number of borders increase.

We should point that even though the running time is poly-logarithmic, the space requirement is at worst $\mathcal{O}\big(n^{2/3}\big)$. This can be rectified by simply removing older border points but such removal may lead to a suboptimal answer. An interesting direction for a future work is to study how to reduce the space complexity without sacrificing the approximation guarantee.

In this paper, we focused only on binary streams. Same concept has the potential to work also on other type of data types, such as integers or real-values. The bottleneck here is Proposition 5 as it relies on the fact that the underlying stream is binary. We will leave adopting these results to other data types as a future work.

## References

[1] Adams, R.P., MacKay, D.J.: Bayesian online changepoint detection. Technical report, University of Cambridge, Cambridge, UK (2007)
[2] Aminikhanghahi, S., Cook, D.J.: A survey of methods for time series change point detection. Knowledge and Information Systems 51(2), 339–367 (May 2017)
[3] Amit, I., Matherly, J., Hewlett, W., Xu, Z., Meshi, Y., Weinberger, Y.: Machine learning in cyber-security — problems, challenges and data sets. In: The AAAI-19 Workshop on Engineering Dependable and Secure Machine Learning Systems (2019)

[4] Baena-Garcia, M., Campo-Avila, J.D., Fidalgo, R., Bifet, A., Gavalda, R., Morales-Bueno, R.: Early drift detection method. In: In 4th Int. Workshop on Knowledge Discovery from Data Streams (2006)
[5] Basseville, M., Nikiforov, I.V.: Detection of Abrupt Changes – Theory and Application. Prentice-Hall (1993)
[6] Bellman, R.: On the approximation of curves by line segments using dynamic programming. Communications of the ACM 4(6), 284–284 (1961)
[7] Bifet, A., Gavalda, R.: Learning from time-changing data with adaptive windowing. In: In SIAM Int. Conf. on Data Mining. pp. 443–448 (2007)
[8] Calders, T., Dexters, N., Goethals, B.: Mining frequent items in a stream using flexible windows. Intell. Data Anal. 12(3), 293–304 (2008)
[9] Dries, A., Rückert, U.: Adaptive concept drift detection. Stat. Anal. Data Min. 2(5–6), 311–327 (2009)
[10] Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. In: In SBIA Brazilian Symposium on Artificial Intelligence. pp. 286–295 (2004)
[11] Guha, S., Koudas, N., Shim, K.: Approximation and streaming algorithms for histogram construction problems. ACM Transactions of Database Systems 31(1), 396–438 (2006)
[12] Guha, S., Shim, K.: A note on linear time algorithms for maximum error histograms. IEEE Transactions on Knowledge and Data Engineering 19(7), 993–997 (2007)
[13] Harel, M., Mannor, S., El-Yaniv, R., Crammer, K.: Concept drift detection through resampling. In: Proc. of the 31st Int. Conf. on Machine Learning. pp. 1009–1017. ICML (2014)
[14] Kawahara, Y., Sugiyama, M.: Sequential change-point detection based on direct density-ratio estimation. Statistical Analysis and Data Mining 5, 114–127 (2012)
[15] Kifer, D., Ben-David, S., Gehrke, J.: Detecting change in data streams. In: Proc. of the 13th Int. Conf. on Very Large Data Bases. pp. 180–191. VLDB (2004)
[16] de Leeuw, J., Hornik, K., Mair, P.: Isotone optimization in r: Pool-adjacent-violators algorithm (pava) and active set methods. Journal of Statistical Software, Articles 32(5), 1–24 (2009)
[17] Nishida, K., Yamauchi, K.: Detecting concept drift using statistical testing. In: Proc. of the 10th Int. Conf. on Discovery Science. pp. 264–269 (2007)
[18] Ross, G.J., Adams, N.M., Tasoulis, D.K., Hand, D.J.: Exponentially weighted moving average charts for detecting concept drift. Pattern Recognition Letters 33(2), 191–198 (2012)
[19] Tatti, N.: Fast sequence segmentation using log-linear models. Data mining and knowledge discovery 27(3), 421–441 (2013)
[20] Tatti, N.: Strongly polynomial efficient approximation scheme for segmentation. Inf. Process. Lett. 142, 1–8 (2019), https://doi.org/10.1016/j.ipl.2018.09.007
[21] Terzi, E., Tsaparas, P.: Efficient algorithms for sequence segmentation. In: Proceedings of the 6th SIAM International Conference on Data Mining (SDM). pp. 316–327 (2006)

## A Proof of Proposition 1

Before proving the proposition, we need some additional notation. Let us write

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

If $p_1 = a_1/(a_1 + b_1)$ and $p_2 = a_2/(a_2 + b_2)$, we will drop them from notation and write instead $\Delta(i)$. Note that $\Delta(i) \geq \Delta(i; r_1, r_2)$ for any $r_1$ and $r_2$.

*Proof.* Let $i$ be a solution for ChangeInc. In case of ties, let $i$ be the smallest index producing the optimal solution.

Let $p_1 \leq p_2$ be the corresponding parameters for the two bernoulli variables, $\Delta(i) = \Delta(i; p_1, p_2)$.

Assume that $p_1 = 0$. If $s_i = 0$, then it is easy to show that

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

By repeating this argument, we can show that there is $i'$ such that $\Delta(i') \geq \Delta(i)$. Moreover, $s_{i'} = 1$ and $s_j = 0$ for any $j < i$ (it is safe to assume that $S$ has non-zero values). This makes $i'$ a border index. Thus we can safely assume that $p_1 > 0$, and similarly $p_2 < 1$.

If $i$ is a border index, then we are done. Assume that $i$ not a border index, that is, there are two integers $x < i < y$ such that

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

We will denote the left hand-side of the inequality $d_1$ and the right hand-side with $d_2$.

Let us define

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

We can now write $\Delta(k; p_1, p_2)$ as

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

for any index $k$. We can now write the difference between the two scores as

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

Normalizing this difference with $y - i$ leads to

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

Similarly,

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

Since $\Delta(i)$ is optimal, $\Delta(i)\geq\Delta(y)$. Then

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

Here we used the fact that $\alpha_1-\alpha_2\leq 0$ since $p_1\leq p_2$ and, by definition, $d_2\leq d_1$. In other words, $\Delta(x)\geq\Delta(x;p_1,p_2)\geq\Delta(i;p_1,p_2)=\Delta(i)$. This violates the minimality of $i$, hence $i$ must be a border index. This completes the proof. □