Concept Drift Detection via Species Estimation Metrics

Igor Dovzhenko

University of Vienna, Faculty of Computer Science, Austria
a12244821@unet.univie.ac.at

Abstract. In modern business processes, automatic detection of conceptual drift is crucial for timely response to failures and process optimization. Most existing solutions rely on complex algorithmic schemes or machine learning methods, complicating deployment and result interpretation. In this work, we investigate the effectiveness of simple threshold-based approaches inspired by biodiversity estimation methods. We implemented a processing pipeline for XES logs, including segmentation into N equal time windows; computation of "species estimation" metrics (Chao1, Hill numbers for $q = 0, 1, 2$, completeness, and coverage); construction of additional features (delta changes, z-scores, relative deltas); and threshold-based drift detection (median + 2·MAD). Experiments with splits into 15, 20, 30, 50, and 100 windows showed that the thresholding strategy based on delta-Hill number for $q = 1$ achieves the highest average F1 score of up to 0.59 (for 15 windows), whereas other biological metrics reach maximum F1 scores of no more than 0.29. These results demonstrate that even simple biodiversity-inspired threshold-based approaches can provide moderately high drift detection accuracy and can be easily integrated into process mining tools.

Keywords: Concept drift detection · Process mining · Species estimation · Thresholding · F1-score

1 Introduction

In modern business and manufacturing environments, event logs capture detailed traces of process execution, but process behavior often evolves over time. Such *concept drift* can lead to failures, quality degradation, or missed opportunities for improvement. Automatic and rapid detection of drift is therefore essential for maintaining process reliability and ensuring timely intervention.

Most existing drift-detection techniques rely on complex process-mining algorithms or resource-intensive machine learning models, which complicates deployment, interpretation, and real-time monitoring.

In this work, we propose a simple, transparent pipeline that leverages two key inspirations:

– Fixed-window segmentation of the XES log, following the procedure of Han *et al.* [1];

– Biodiversity-inspired metrics computed via the **SpeciAL-core** library (Chao1, Hill numbers for $q = 0, 1, 2$, completeness and coverage) from Kabierski *et al.* [2].

The primary advantage of this method is its ability to perform a *rapid screening* of logs for drift without black-box models or extensive parameter tuning, making it highly suitable for real-time process monitoring.

The rest of this report is organized as follows. Section 2 provides a formal problem definition; Section 3 presents our pipeline overview; Section 4 details implementation, code availability, and environment requirements; Section 5 covers the experimental setup and results; and Section 6 concludes.

2 Problem Illustration and Definition

Event logs play a central role in monitoring and analyzing business processes: they record sequences of events that describe the execution of operations in various cases. Each trace (case) in the log contains three mandatory attributes:

– **case ID**: a unique identifier of a process instance;
– **activity**: the name of the performed operation;
– **timestamp**: the time at which the event started or completed.

In such logs, two fundamental types of concept drift may occur:

Sudden drift an abrupt change in the control-flow structure over a short segment of the log;
Gradual drift a smooth evolution of the process over an interval of time.

In this work, we focus exclusively on *sudden drift*, as detecting abrupt changes is a necessary "first step"—gradual drifts require more complex modeling and are left for future research.

To detect drift points, we partition the log into N equal windows (segments), each containing a comparable number of events or cases. Our method then returns a set of drift points

$$D = \{\, p_1, p_2, \ldots, p_k \},$$

where each p_i is defined as the center of the window in which an abrupt change in process behavior is detected.

Thus, our goal is to develop a lightweight, modular, and efficient screening method for event logs that identifies *sudden drift* by outputting clear, interpretable drift points without relying on complex machine-learning models or extensive parameter tuning.

3 Approach

In this section, we present the overall methodology of our sudden-drift detection pipeline. Our goal is to balance simplicity, interpretability, and efficiency by decomposing the task into modular stages that require no heavy machine-learning models or extensive parameter tuning. We first describe how logs are segmented into fixed windows, then detail the computation of biodiversity-inspired metrics, feature engineering, threshold-based detection, and finally evaluation.

The overall pipeline is shown in Figure 1.

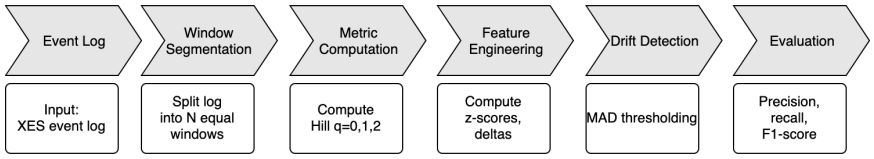

Fig. 1. Overview of the drift detection pipeline: (1) Event Log, (2) Window Segmentation, (3) Metric Computation, (4) Feature Engineering, (5) Drift Detection, (6) Evaluation.

3.1 Log Segmentation into Windows

For our experiments we use the CDLG test event logs[1]. Each log already contains 2,000–6,000 time-ordered traces (cases) and requires no further preprocessing.

To pinpoint sudden drift, we partition each log into N consecutive windows of equal size (by trace count). In this study, we evaluate five segmentation schemes:

$$N \in \{15,\ 20,\ 30,\ 50,\ 100\}.$$

Equal-sized windows ensure comparable data density across segments and eliminate bias due to varying event counts.

Drift detection is then performed at the window level: if any biodiversity metric or derived feature in window W_i exhibits an abrupt change, we label W_i as a drift window. We define the drift point p_i as the center of W_i.

Using fixed windows provides a formal bound on localization error. Selecting the window midpoint yields a maximum relative error of

$$\frac{1}{2N} \times 100\%.$$

[1] `https://huggingface.co/pm-science/cv4cdd_project/tree/main/datasets/cdlg/zipped_test`

Thus, with $N = 100$ the error does not exceed approximately 0.5%, while for $N = 15$ it is approximately 3.33%. Increasing N reduces localization error but increases sensitivity to noise.

3.2 Species Extraction via Bigrams

We begin by extracting the set of "species" from each window W_i as directly-follows pairs (bigrams) of activities, using SpeciAL-core.[2] Each trace σ in W_i is a sequence of activities, e.g.

$$\sigma = [A \to B \to C \to D],$$

from which we form the bigrams

$$(A \to B),\ (B \to C),\ (C \to D).$$

Collecting all such bigrams across traces yields a multiset of species, and we let p_j denote the relative frequency of species j in W_i:

$$p_j = \frac{\text{count of species } j}{\sum_k \text{count of species } k}.$$

3.3 Biodiversity Metric Computation

For each window W_i, we assess the diversity of species (directly-follows pairs) using the SpeciAL-core library. We compute the following metrics:

- **Hill number for** $q = 0$ (^0D) — observed species richness S_{obs}.
- **Hill number for** $q = 1$ (^1D) — exponential of Shannon entropy:

$$^1D = \exp\left(-\sum_j p_j \ln p_j\right).$$

- **Hill number for** $q = 2$ (^2D) — inverse Simpson index:

$$^2D = \left(\sum_j p_j^2\right)^{-1}.$$

- **Completeness** — proportion of singleton species relative to ^0D:

$$\text{completeness} = 1 - \frac{F_1}{^0D},$$

  where F_1 is the number of species observed exactly once.
- **Coverage** — ratio of observed species to estimated richness:

$$\text{coverage} = \frac{S_{\text{obs}}}{^0D}.$$

[2] `https://github.com/MartinKabierski/SpeciAL-core/tree/main/special4pm`

3.4 Feature Engineering

Absolute metric values m_i in each window W_i are influenced by background variability and metric scale. To robustly highlight abrupt changes, we derive:

1. **Δ–features (absolute differences):**

$$\Delta_i^m = |m_i - m_{i-1}| .$$

2. **z–scores (standardized deviations):**

$$z_i^m = \frac{m_i - \mu_m}{\sigma_m},$$

   where μ_m and σ_m are the mean and standard deviation of metric m across all windows.
3. **Relative deltas:**

$$r_i^m = \frac{\Delta_i^m}{m_{i-1} + \varepsilon}.$$

For each metric m, we obtain three feature series: Δ_i^m, z_i^m, and r_i^m, yielding a feature matrix of size $N \times 3P$ with P base metrics.

3.5 Threshold-Based Drift Detection

We flag windows as drift candidates using a robust thresholding scheme.

Threshold calculation. For each feature x, compute:

$$\tau_x = \text{median}(x) + 2 \cdot \text{MAD}(x), \quad \text{MAD}(x) = \text{median}\left(|x_i - \text{median}(x)|\right).$$

Window flagging. A window W_i is marked as drift if any feature $x_i > \tau_x$.

Drift-point determination. The drift point p_i is the temporal midpoint of flagged window W_i.

This approach is efficient, interpretable, and avoids reliance on black-box models or heavy parameter tuning.

4 Implementation and Repository

Code and Repository

All of scripts, the pipeline configuration, and sample logs are available on GitHub:

`https://github.com/iDovj/concept-drift-detection-via-species.git`

Environment and Dependencies

- **SpeciAL-core** (Chao1, Hill numbers, completeness, coverage)
- **special4pm** (PM4Py-adapter for SpeciAL)
- **PM4Py** (XES parsing and bigram extraction)
- **NumPy**, **Pandas** (data handling)
- **Matplotlib** (plots)
- **SciPy** (MAD, z-score)

5 Evaluation

In this section, we empirically evaluate the performance of our sudden-drift detection pipeline. We begin by detailing the test dataset and its characteristics, then present the experimental setup and metrics used, followed by quantitative results for key feature variants under different windowing schemes.

5.1 Data

For evaluation, we randomly selected 300 event logs—each containing at least one sudden drift—from the CDLG test dataset.[3] All logs are generated by the Concept Drift Log Generator and exhibit the following characteristics:

- **Trace count per log:** 2,000–6,000 traces
- **Trace length:** 1–65 events
- **Process model size:** 6–20 activities, with sequential, choice, parallel, and loop constructs
- **Drift type:** exclusively sudden (abrupt), with 1–3 drifts per log

Each log is segmented into $N \in \{15, 20, 30, 50, 100\}$ equal-sized windows (by trace count) before applying our detection pipeline. This setup ensures a representative testbed for assessing sensitivity and robustness across different process complexities and drift densities.

5.2 Experiment 1: Illustrative Examples of Metric Behavior

To illustrate how our biodiversity-inspired metrics respond to sudden drift, we present three representative cases using different windowing schemes. Each plot displays five metrics—${}^0D$, ${}^1D$, ${}^2D$, completeness, and coverage—with the true drift points (window centers) marked by dashed red lines.

[3] `https://huggingface.co/pm-science/cv4cdd_project/tree/main/datasets/cdlg/zipped_test`

Case 1: Clear drift (Log 1704, $N = 50$). An abrupt change occurs in window 28. All three Hill numbers exhibit a pronounced spike, while completeness and coverage remain stable.

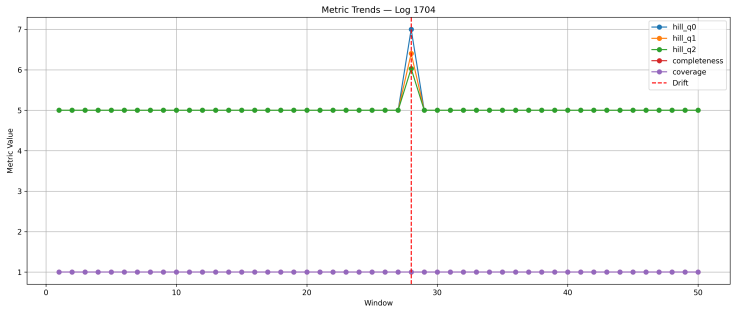

Fig. 2. Metric trends for Log 1704 with $N = 50$ windows. A clear spike in Hill numbers at window 28 marks the sudden drift.

Case 2: Hidden drift (Log 1722, $N = 50$). The true change point produces only a minor perturbation, insufficient to cross thresholds.

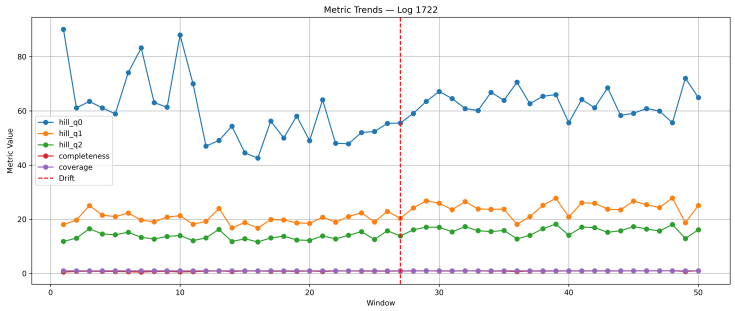

Fig. 3. Metric trends for Log 1722 with $N = 50$ windows. The sudden drift at window 28 is barely visible and not detected.

Case 3: Window granularity dependence (Log 1739). Using $N = 20$, moderate drift is detected; with $N = 50$, peaks are smoothed.

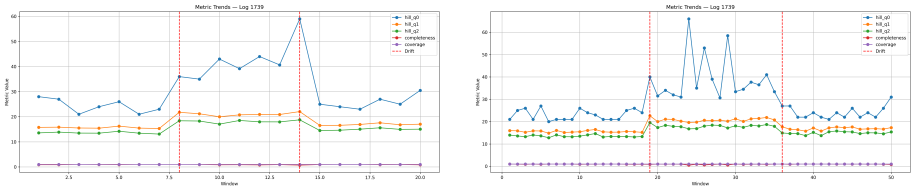

Fig. 4. Log 1739: metric trends for $N = 20$ (left) show clear drifts at windows 8 and 15; with $N = 50$ (right), one drift is missed.

5.3 Experiment 2: Window-Level Detection Accuracy

This experiment evaluates how accurately features identify windows containing drift. Each window is treated as a binary prediction: drift or no drift.

Protocol.

1. Segment each log into N windows.
2. Convert ground-truth drift points to binary window labels.
3. Apply MAD-thresholding for each feature.
4. Compute:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad \text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Noise tolerance. Any window containing a true drift counts as correct. Localization error is bounded by $\frac{1}{2N}$.

Table 1. Window-level performance (mean scores over 300 logs, $N = 20$)

Feature	Precision	Recall	F1-score
delta_chao1	0.231	0.369	0.258
delta_completeness	0.056	0.111	0.070
delta_coverage	0.081	0.144	0.097
delta_hill_q1	**0.535**	**0.655**	**0.554**
delta_hill_q2	0.500	0.613	0.512
reldelta_chao1	0.228	0.202	0.190
reldelta_hill_q1	0.494	0.594	0.505
reldelta_hill_q2	0.465	0.563	0.476
zscore_chao1	0.400	0.425	0.378
zscore_hill_q1	0.540	0.586	0.516
zscore_hill_q2	0.471	0.519	0.446

Table 2. Window-level performance (mean scores over 300 logs, $N = 100$)

Feature	Precision	Recall	F1-score
delta_chao1	0.134	0.350	0.154
delta_completeness	0.017	0.130	0.029
delta_coverage	0.022	0.166	0.037
delta_hill_q1	0.114	0.455	0.162
delta_hill_q2	0.108	0.434	0.155
reldelta_chao1	0.131	0.225	0.126
reldelta_hill_q1	0.100	0.446	0.148
reldelta_hill_q2	0.096	0.418	0.141
zscore_chao1	**0.208**	0.343	**0.212**
zscore_hill_q1	0.203	0.403	0.226
zscore_hill_q2	0.187	0.387	0.207

5.4 Experiment 3: Latency-Aware Evaluation

We assess how precisely predicted drift points align with true drift timestamps.

Matching protocol. A predicted point p_d matches a true drift p_g if:

$$|p_d - p_g| \leq \theta \cdot |\Sigma_L|,$$

with latency margin $\theta \in \{0.01, 0.025, 0.05\}$, and $|\Sigma_L|$ = number of traces. Greedy matching is used.

Table 3. Latency-aware evaluation at different thresholds ($N = 30$)

	1%			**2.5%**			**5%**		
Feature	Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
delta_hill_q1	0.250	0.39	0.30	0.335	0.72	0.43	0.410	0.81	0.55
delta_hill_q2	0.220	0.42	0.29	0.277	0.81	0.39	0.365	0.85	0.51
reldelta_hill_q1	0.280	0.42	0.33	0.349	0.62	0.42	0.410	0.79	0.54
reldelta_hill_q2	0.310	0.45	0.37	0.398	0.63	0.44	0.455	0.81	0.58
zscore_hill_q1	0.390	0.46	0.42	0.425	0.49	0.43	0.478	0.55	0.51

Summary of Evaluation Results Metrics derived from Hill numbers ($q = 1, 2$) achieve the highest F1 scores — up to 0.55 (window-based) and 0.58 (latency-aware). Simpler metrics such as completeness and coverage yield significantly lower performance.

Window-based segmentation introduces localization error $\leq \frac{1}{2N}$. No comparisons to learning-based methods are included, but our method is fast, interpretable, and requires no training.

6 Conclusion and Future Work

In this work, we proposed a simple and interpretable approach for detecting sudden concept drift in business processes, based on biodiversity-inspired metrics. We implemented a modular pipeline consisting of event log segmentation, extraction of species as activity bigrams, computation of diversity indicators (Chao1, Hill numbers, completeness, coverage), construction of derived features, and threshold-based drift detection.

Experiments on 300 synthetic event logs with known drift points showed that metrics such as Δ^{hill_q1}, z-scores, and relative changes in Hill numbers can achieve F1 scores of up to 0.55 in window-based evaluation and up to 0.58 in latency-aware settings. Although performance varies across logs depending on drift magnitude and segmentation, the method demonstrates consistently strong results without requiring any training.

Due to its simplicity, minimal parameter tuning, and computational efficiency, our approach is a promising candidate for practical process monitoring applications, especially in low-resource settings where interpretability and fast deployment are critical.

As future work, we plan to:

- compare our method against learning-based approaches such as CV4CDD;
- extend the framework to detect gradual drifts;
- explore alternative segmentation strategies (e.g., sliding or adaptive windows);
- evaluate performance on real-world event logs from industrial settings;
- and train a binary classifier using the same feature set to assess whether learning-based drift detection improves over static thresholding.

Acknowledgements This study was conducted as part of the Informatics Practicum (P1) at the University of Vienna under the supervision of Professor Han van der Aa.

References

1. van der Aa, H., Francescomarino, T.D., de Leoni, M., et al.: Looking for change: A computer vision approach for concept drift detection in process mining. In: Proceedings of the 2024 International Conference on Advanced Information Systems Engineering (CAiSE) (2024), https://www.researchgate.net/publication/383621926_Looking_for_Change_A_Computer_Vision_Approach_for_Concept_Drift_Detection_in_Process_Mining
2. Kabierski, M.: Special-core: Species estimation library for process mining. https://github.com/MartinKabierski/SpeciAL-core/tree/main/special4pm (2023)