Benchmarking Drift Detection Methods Against Simulated Data

A Look Into Drift Detection Methodology

Kristen Hart, Victor Chupka, Daniel Tabach

Introduction

Many companies within the tech sector such as Meta, Amazon, Google, and Microsoft typically launch products that undergo numerous testing criteria to be confident that their product will reach their consumers and encourage more engagement with their services. For instance, Meta's Instagram platform may conduct tests such as UI changes, button behavior changes, and new refinements on top of existing features within the app. Typically, users are placed into a small sample population where tests (AB / ANOVA Tests) are conducted before launching a product.

When a product undergoes launch, key success criteria are tracked and monitored to ensure the product is well-underway to be successful for the overall population. One of the most common success metrics is looking at user behavior and engagement with our products to see if behavior shifts over time. Not all behavior shifts are noticeable within a short period of time; it is especially vital to catch behavior shifts early to alter or refine the product before launching to a larger population where the cost of failure is significantly higher. In this project, we are simulating user behavior data across a fictional online platform, and the launch of a product where user behavior shifts gradually over time. Our goal is to benchmark and test different algorithms against uni-variate behavioral shifts to evaluate the success rate of catching early drift for different models.

"The early detection of item drift is an important issue for frequently administered testing programs because items are reused over time. Unfortunately, operational data tend to be very sparse and do not lend themselves to frequent monitoring analyses, particularly for on-demand testing" [4] as quoted from *Detecting Item Drift in Large-Scale Testing* showcases why this is a topic that modern data scientists should care about.

Problem Statement

How can we compare and contrast different drift detection methodologies on the simulated dataset to gain a deeper understanding of each model's strengths and weaknesses?

Exploratory Data Analysis

Although the data was simulated, we performed exploratory data analysis on the data to get a more intuitive understanding of the underlying data that was generated across the different scenarios. Our objective was to understand trends and patterns with in the data that could be informative for the development of the drift detection models.

From 1 we see that pre-launch variance remained fairly stable which suggests consistent engagement behavior prior to the product launch. This verifies that `Time A` was developed correctly. After launch, we see that variance levels increased across each scenario. The increase in variance can cause challenges to appear in the change detection due to it increasing the baseline noise levels.

Engagement Variance Over Time

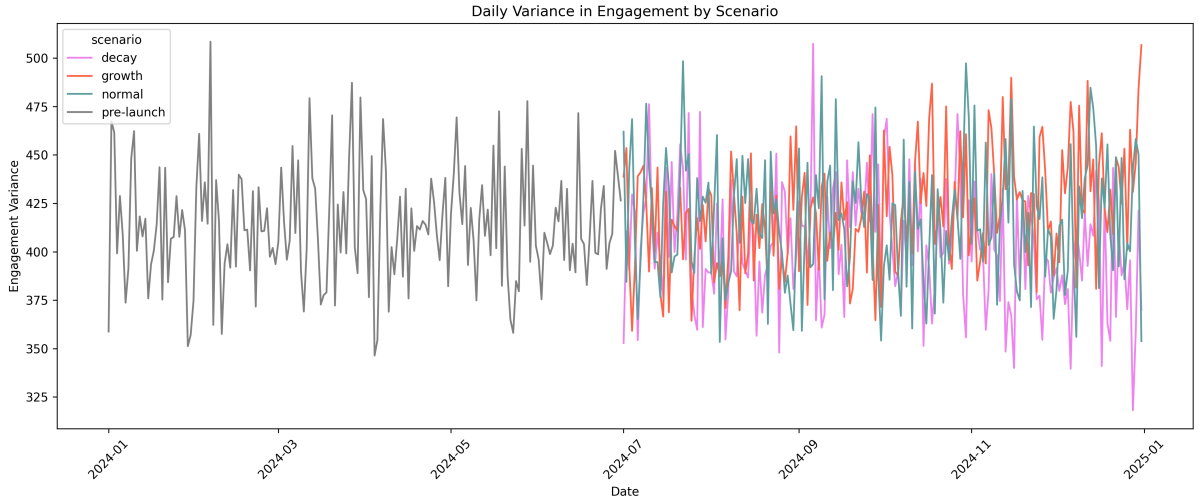


Figure 1: Daily variance in engagement by scenario.

Engagement Trends Over Time

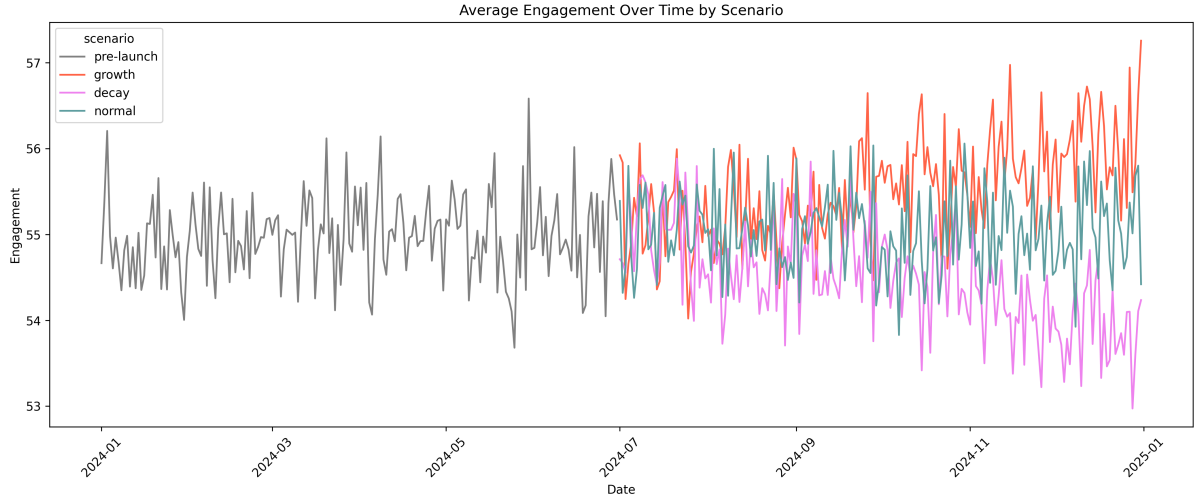


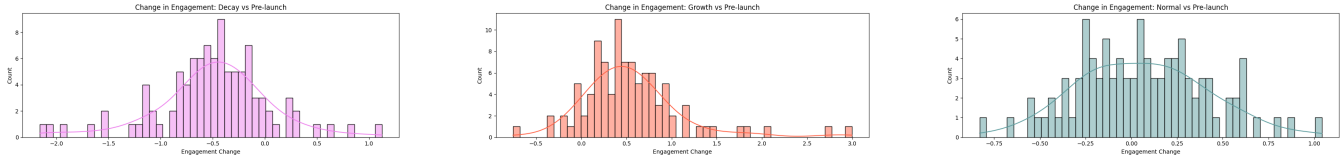
Figure 2: Average engagement over time by scenario.

This figure 2 further verifies that the dataset is drifting or remaining stable in each respective scenario as expected from our data generation.

Distribution of Engagement Changes

The histograms of daily engagement 3 relative to the pre-launch levels showed:

- A left shift in the decay scenario showing declining engagement
- A right shift in the growth scenario showing increasing engagement
- A symmetric and centered distribution in the normal scenario confirming the lack of drift



(a) Decay vs Pre-Launch (b) Growth vs Pre-Launch (c) Normal vs Pre-Launch

Figure 3: Distribution of daily engagement change relative to pre-launch baseline.

Engagement Distribution

The boxplots showed similar median engagement values in all scenarios. The variance increase slightly post launch (especially for growth). No extreme changes were detected in overall engagement levels which suggests that drift was driven by shifting variance rather than large magnitude shifts.

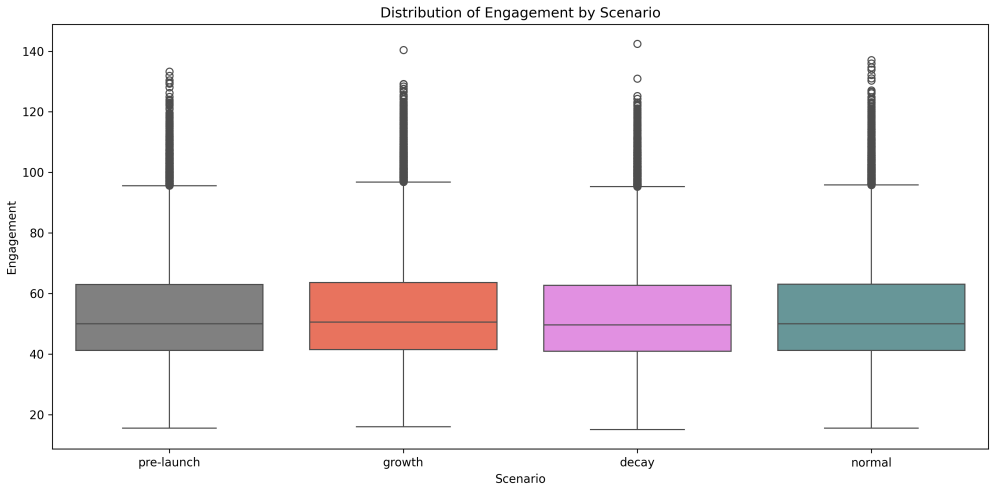


Figure 4: Distribution of engagement across scenarios.

User-Level Engagement Averages

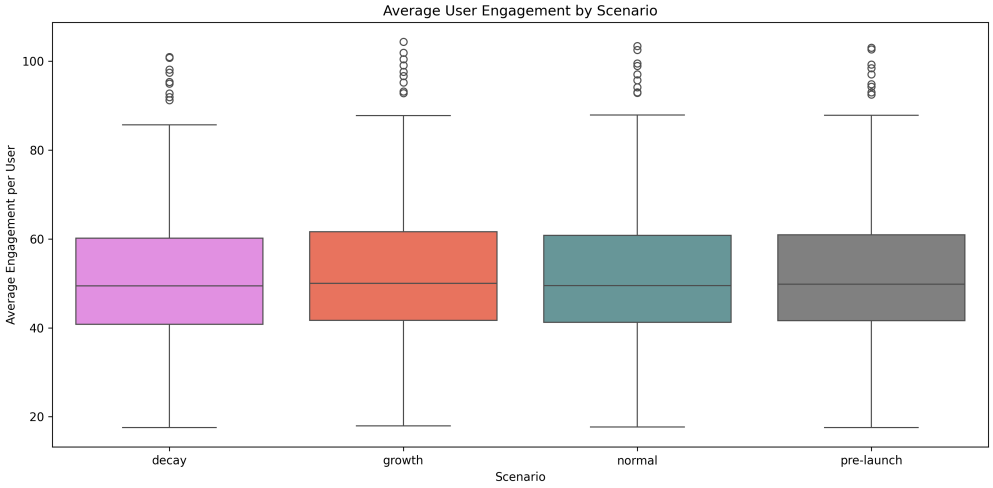
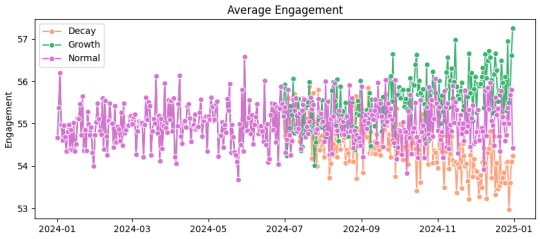


Figure 5: Average user engagement across scenarios.

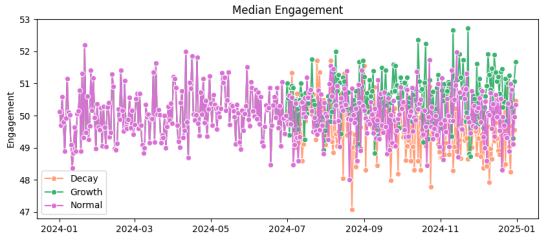
The average user engagement per scenario showed consistency across the groups with a minor increase in the growth scenario. This highlights that drift observed in an aggregated fashion was not isolated to a subset of users but rather the userbase as a whole.

Means, Medians, & Rolling Averages

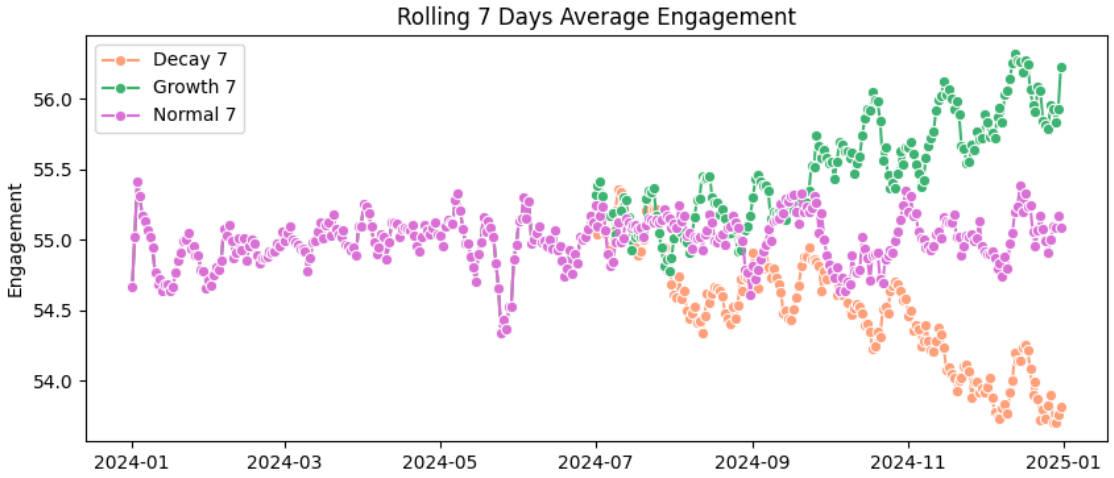
Before running our models, it was important to identify the aggregation of the data that allowed for the model to perform best. For this, we tested daily average, daily medians, and rolling averages on the daily averages. Below 6 is a comparison of the average, median, and rolling average of the average:



(a) Daily Average Engagement



(b) Daily Median Engagement



(c) Daily 7-Day Rolling Average Engagement

Figure 6: Comparison of Daily Engagement Metrics Across Normal, Growth, and Decay Scenarios

From these graphs of the different aggregation of daily data, it is clear that the average and median contain a significant amount of noise that would pose a challenge to any change detection model. Conversely, the rolling average of each day's average presents a smoother graph that shows clear growth and decay appropriately. Due to this, we opted to include the rolling average into each of our change detection models.

Overall

The EDA phase confirmed the presence of engagement shifts that aligned with the intended output of our data simulation. The growth and decay scenarios showed detectable changes in direction, while the normal scenario remained stable proving its utility as a control group. The daily averages and medians of engagement proved to be noisy for this set of data, suggesting that passing in rolling averages into each change detection model may provide better results.

Methodology

Simulating the Data

To evaluate change detection algorithms, we generated synthetic user data to simulate realistic user behavior before and after a product launch. The data is structured into two periods: `Time A` (pre-launch control) and `Time B` (post-launch experimental). The data covers multiple months of daily user engagement, and each day consists of randomly sampled engagement scores for a fixed number of users.

`Time A` data represents a stable engagement pattern, drawn from a base normal distribution with minimal noise to resemble consistent user activity. For `Time B`, three different behavioral trends were simulated:

- **Normal Time B**: Engagement remains statistically unchanged after the launch, serving as a control scenario to test for false positives.
- **Decay Time B**: Engagement declines over time following an exponential decay function: $f(t) = e^{-\lambda t}$, where λ controls the rate of decay.
- **Growth Time B**: Engagement increases post-launch using an exponential growth function: $f(t) = 1 + (e^{\gamma t} - 1)$, where γ sets the growth rate.

The data simulation was seeded for reproducibility using fixed random seeds. The data was then saved into `.csv` files for use in the change detection analyses into `Time_A.csv`, `Time_B_Normal.csv`, `Time_B_Growth.csv`, and `Time_B_Decay.csv`. The controlled set up allows for precise evaluation of detection methods under known conditions of stability, growth, and decay.

Evaluation Methods

We evaluated each model using three criteria:

1) **How quickly was a change detected after the product launch date?** We wanted to identify the time delta between when the product was launched and how many days after launch a change was detected in the growth and decay Time B. In the business application, a model that detected changes quicker would be more beneficial to make critical roll out or scale back decisions in a timely fashion. **Smaller/Shorter amount of time is better.**

2) **How many false positives were detected?** A false positive would be any time the change detection model detected changes prior to launch or in the normal Time B model. In the business application, a false positive could make decision makers change course prematurely. With any model, we wanted to find a good balance (if possible) between the how quickly to detect changes and false positives that may lead to bad decision making). **Smaller amount false positives is better.**

3) **How easy was each change detection model to use?** Some models required a significant amount of pre-processing to work effectively and/or advanced knowledge about the nature of the change to work effective (sudden or gradual changes may work better). More flexible models that could detect changes with minimal preprocessing or fine tuning could be desired in a business setting, especially when there is lots of uncertainty if/how the change may occur after launch. **Models that required less advance knowledge of the online data to perform well are better. Additionally, models that required minimal preprocessing or hyperparameter tuning are desired**

Pruned Linear Time with Radial Basis Cost Function

We chose the Pruned Exact Linear Time (PELT) algorithm with a Radial Basis Function (RBF) cost model as one of the models to detect behavioral drift in user engagement post-product launch. As mentioned, the

data used was `Time A`: the user engagement before the launch, and `Time B`, the data after the launch, with 3 different scenarios for `Time B`: normal, decay, and growth. The analysis focuses solely on the `Engagement` feature to analyze how user engagement has shifted since the product launch. Please refer to the appendix A for a deeper explanation behind this algorithm.

The PELT algorithm is implemented using the `ruptures` library [3]. It works by minimizing a cost function that is penalized by the number of change points to identify which shifts are statistically significant. Time series from both `Time A` and `Time B` are concatenated at the product launch date, and then the change points are detected only within `Time B`. The detected change points are evaluated against the known behavioral change to calculate evaluation metrics such as precision, recall, and F1 score.

Both daily averages and 7-day rolling averages were analyzed to assess the model across different levels of noise and volatility. Detected change points were evaluated against the ground truth of known behavioral shifts and performance was measured through True Positive, True Negative, False Positive, and False Negative counts. Additionally, metrics such as precision, recall, and F1 score were calculated.

Kolmogorov Smirnov

Another methodology that we used was the Kolmogorov Smirnov test. This test detects changes in user behavior across different time series scenarios. The K-S test is a non-parametric method used to determine if two distributions differ significantly. This approach compares the empirical distribution of engagement in `Time A` to sliding windows in the post-launch period of `Time B`. It then flags windows with statistically significant shifts.

For each point in `Time B`, we performed a K-S test between `Time A`, and a moving window of future data. If the p-value is below the threshold defined (which was defined at 0.01) then the point is marked as a change point. The points then are filtered and evaluated to distinguish true post-launch drift from noise. Please refer to the appendix D for a deeper explanation.

We tested daily averages and rolling 7-day averages for the Kolmogorov Smirnov model. Detected change points were evaluated using the true positive, true negative, false positive, and false negative rates as well as the precision, recall, accuracy, and F1 score.

ADWIN

Another model used was the ADWIN (Adaptive Windowing) method for online drift detection.

This online drift detection detects changes by having an window of changing length that is split into two sub-windows. If the most recent sub-window's mean is significantly different from the older sub-window (with significance user defined by a variable δ), then the algorithm flags a change. When a change happens, the most recent sub-window is defined from the change point moving forward, ensuring that only changes from the current values are included. In our implementation, the algorithm is run after each data point is added to the data stream.

For this method, we wanted to aggregate daily data in different fashions to try to ensure the algorithm picked up on change the fastest and minimized false positives. To do this, we tested both means of daily engagement and rolling averages with different window lengths.

Additionally, we wanted to tune the ADWIN parameters, such as the significance value (δ) and the minimum length allowed for a sub window. It was important to identify appropriate parameters for the ADWIN model, such as the significance value (δ) and the minimum length allowed for a sub-window. We needed to identify which length of rolling average produced the best results. We ran several models with different rolling average windows, different significance levels, and different minimum sub-window lengths. To

identify the best value, we balanced false positives (number of flags in the normal-time B group) and the delay before first capturing a change detected on the growth- and decay-time B groups.

Bayesian Changepoint Detection

We also implemented a Bayesian Online Change-point Detection (BOCPD) algorithm, as introduced by Adams and MacKay (2007), as one of the models to detect behavioral drift in user engagement following the product launch. As in the other methods, the data was structured into Time A (user engagement before launch) and Time B (engagement after launch), with three separate scenarios for Time B: normal, decay, and growth. The analysis focuses exclusively on the 'Engagement' feature to detect shifts in user behavior.

The BOCPD algorithm is implemented based on the original Bayesian framework (with some adjustments), modeling the Time A engagement data's mean and variance as a baseline for what may occur in Time B (see appendix). A changepoint is flagged when the new data looks unlikely enough that the model decides a new set of data has probably started. See Appendix for more details on the algorithm. [1] [8]

CUSUM

Another model that we chose was the Cumulative Sum (CUSUM) algorithm to detect shifts in engagement metrics across our three scenarios. CUSUM works by monitoring the cumulative deviation from a reference mean which allows it to identify persistent changes in behavior that, although they may be subtle, accumulate over time. CUSUM is often used in quality control and change monitoring were it is essential to detect change rapidly. Please refer to the appendix B for a deeper explanation behind this algorithm.

`Time A` was utilized as the control period and `Time B` was the experimental period. Engagement was measured daily and compared against the pre-launch baseline using one-sided cumulative sums. One-sided CUSUM tests were then applied to the `Time B` data and analyzed on its performance for both the daily average and the 7-day rolling averages for the engagement metric.

Detection thresholds were tuned to increase the sensitivity of the model due to its over-frequent detection of change points.

We tested daily averages and rolling 7-day averages for the Kolmogorov Smirnov model. Detected change points were evaluated using the true positive, true negative, false positive, and false negative rates as well as the precision, recall, accuracy, and F1 score.

Windowed CUSUM with Log-Likelihoods

We developed a windowed change-point detection model inspired by cumulative sum (CUSUM) methods and likelihood-based scoring. The objective was to detect distributional shifts in user engagement following the product launch and use log-likelihood to gently detect drift.

The baseline behavior (Time A) was modeled - as new observations arrived during the Time B period, a sliding window of fixed size k days was used to compute the cumulative log-likelihood of the data under the baseline model

At each step, the model compares the current window's log-likelihood $\log \mathcal{L}(w_t)$ to a rolling average of the previous log-likelihoods. A changepoint is flagged if the drop exceeds a user-defined threshold δ. The model accumulates evidence of gradual deviations from prior means in the data rather than reacting to individual outliers. Our goal was to create a reliable CUSUM model that would be able to adjust to the underlying data using log-likelihoods.

Both daily engagement and 7-day rolling averages were analyzed to study the model's performance across different levels of noise and volatility. Detected changepoints were compared to known behavioral drift

points to assess detection quality using precision, recall, and F1 score metrics.

Grid searches were conducted over window sizes (k) and detection thresholds (δ) to optimize sensitivity to true drift while minimizing false positives.

Findings

Pruned Linear Time with Radial Basis Cost Function

Across the three different scenarios for `Time B`, the PELT with RBF model performed strongly.

Normal

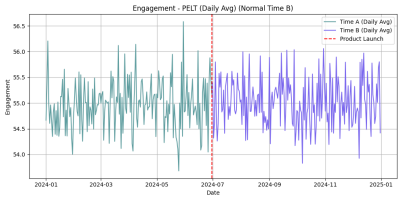
- No change points detected across both daily average and rolling average series
- Specificity remained perfect with 0 false positives. This highlights the model's stability in the absence of drift

Decay

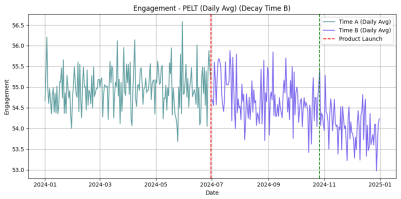
- For daily average, a change point was detected 119 days after launch
- For rolling average, two change points were detected at 29 days and 139 days post launch
- Precision and recall were perfect for both daily and rolling, which indicates a high sensitivity yet also high accuracy in detecting decay

Growth

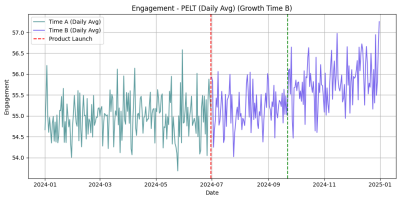
- For daily average, one change point was detected at 84 days after launch
- For rolling average, 3 change points were detected at -1, 84, and 134 days after launch
- Detection was strong, however there was slight overfitting in the rolling average series which reduced the precision to 0.5



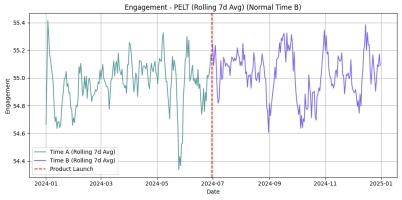
(a) Normal (Daily Avg)



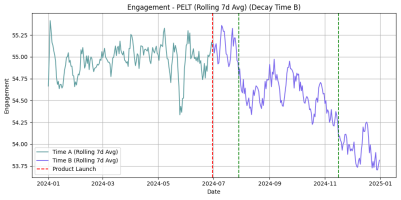
(b) Decay (Daily Avg)



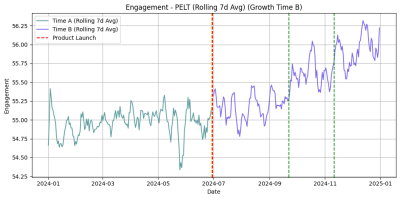
(c) Growth (Daily Avg)



(d) Normal (Rolling 7-Day Avg)



(e) Decay (Rolling 7-Day Avg)



(f) Growth (Rolling 7-Day Avg)

Figure 7: Engagement over time across scenarios. Red dashed lines represent the product launch; green dashed lines indicate detected change points.

Table 1: Summary of PELT Change Point Detection Metrics Across Scenarios

Scenario	Days After Launch	Count	TP	FP	FN	TN	Precision	Recall	F1 Score
Normal (Daily Avg)	N/A	0	0	0	0	1	N/A	N/A	N/A
Normal (Rolling Avg)	N/A	0	0	0	0	1	N/A	N/A	N/A
Decay (Daily Avg)	119	1	1	0	0	0	1.0	1.0	1.0
Decay (Rolling Avg)	29, 139	2	1	0	0	0	1.0	1.0	1.0
Growth (Daily Avg)	84	1	1	0	0	0	1.0	1.0	1.0
Growth (Rolling Avg)	-1, 84, 134	3	1	1	0	0	0.5	1.0	0.67

Kolmogorov Smirnov

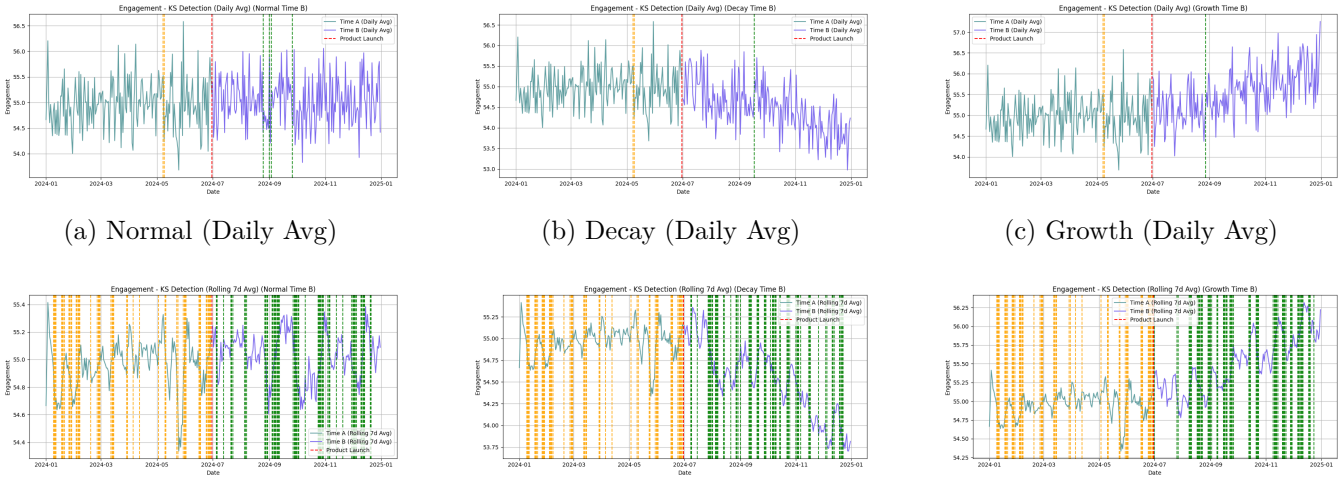
The Kolmogorov Smirnov test showed high sensitivity to minor fluctuations in engagement. It, however, lacked sufficient specificity to truly and reliably distinguish drift from noise.

Normal

- K-S detected four false positives in the daily and rolling series
- Precision, recall, and F1 score were all 0.00 which indicates over-sensitivity to noise

Growth & Decay

- Able to detect decay after launch
- Many false positives leading to low precision, but a perfect recall. This indicates that this model can produce a lot of false alarms.



(a) Normal (Daily Avg) (b) Decay (Daily Avg) (c) Growth (Daily Avg)

(d) Normal (Rolling 7-Day Avg) (e) Decay (Rolling 7-Day Avg) (f) Growth (Rolling 7-Day Avg)

Figure 8: K-S Test detection across Time B scenarios. Red dashed lines indicate product launch; green dashed lines indicate detected true change points; orange dashed lines indicate false positives.

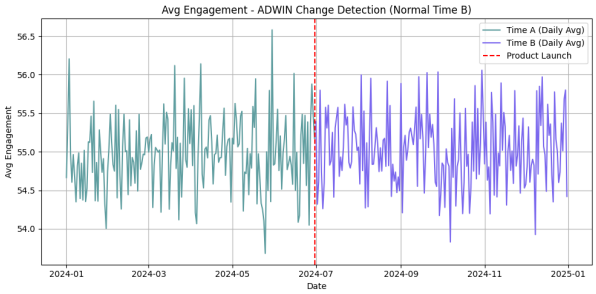
Table 2: Summary of Kolmogorov–Smirnov Change Point Detection Metrics Across Scenarios

Scenario	Days After Launch	Count	TP	FP	FN	TN	Precision	Recall	F1 Score
Normal (Daily Avg)	Multiple	6	0	6	0	0	0.00	0.00	0.00
Normal (Rolling Avg)	Multiple	103	0	103	0	0	0.00	0.00	0.00
Decay (Daily Avg)	Multiple	3	1	2	0	0	0.33	1.00	0.50
Decay (Rolling Avg)	Multiple	109	1	48	0	0	0.02	1.00	0.04
Growth (Daily Avg)	Multiple	3	1	2	0	0	0.33	1.00	0.50
Growth (Rolling Avg)	Multiple	112	1	48	0	0	0.02	1.00	0.04

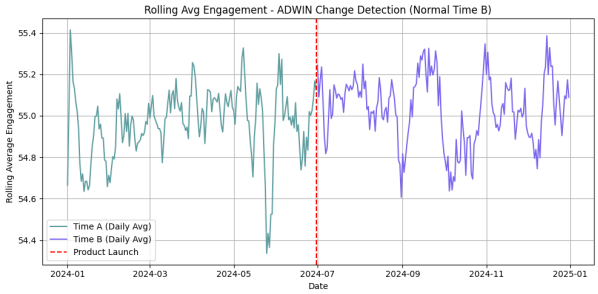
ADWIN

We ran the ADWIN model with varying values of δ ranging from 0.01-0.5, minimum window lengths of 1 to 50, and different rolling window days from 1 to 14. Additionally, we wanted to be fairly certain a change had occurred before flagging it, but not being so conservative there was a substantial delay in detection. Thus adhering to standard .05 and below significance values seemed to balance these two concerns appropriately.

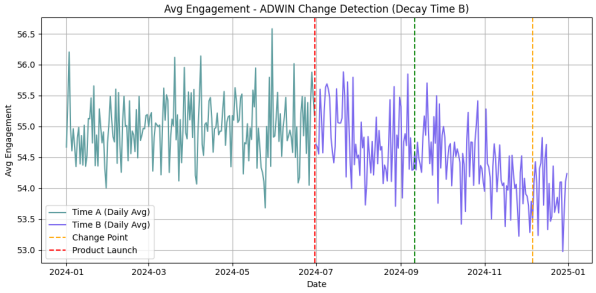
Across varying values of delta, minimum window length, and rolling days average, a trend of not flagging any changes on the normal group, but taking between 45–60 days in both the growth and decay groups to catch a change was present. As such, the below groups will highlight the model with a .05 significance value, a rolling average window of 7 days, and a minimum window length of 1:



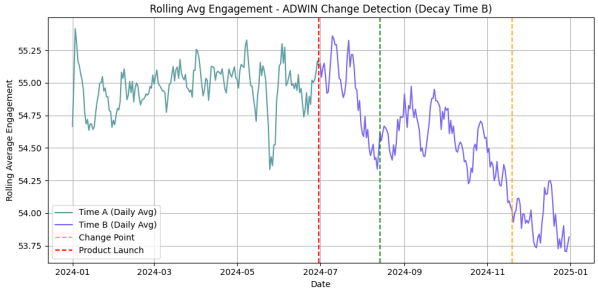
(a) Normal Time B (Daily Avg)



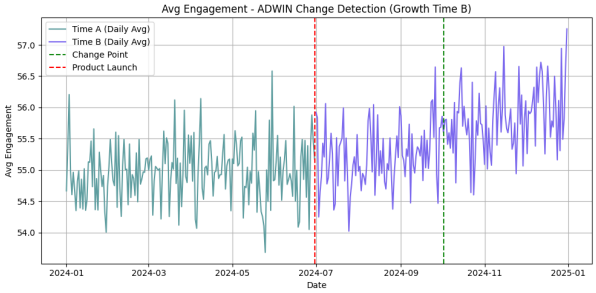
(b) Normal Time B (Rolling 7-Day Avg)



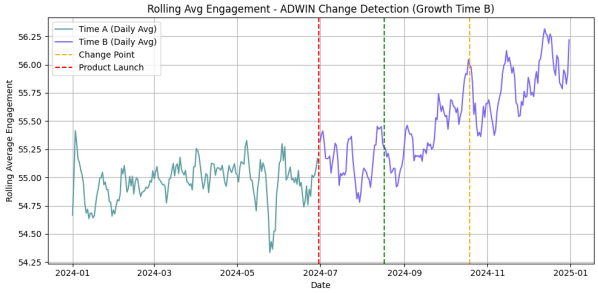
(c) Decay Time B (Daily Avg)



(d) Decay Time B (Rolling 7-Day Avg)



(e) Growth Time B (Daily Avg)



(f) Growth Time B (Rolling 7-Day Avg)

Figure 9: Engagement over time using ADWIN (Adaptive Windowing) Change Detection. Red dashed lines mark the product launch; orange and green lines indicate detected change points.

Table 3: Summary of ADWIN Change Point Detection Metrics Across Scenarios (Rolling Averages)

Scenario	Days After Launch	Count	TP	FP	FN	TN	Precision	Recall	F1 Score
Normal (Daily Avg)	N/A	0	0	0	0	1	0.0	N/A	N/A
Normal (Rolling Avg)	N/A	0	0	0	0	1	0.0	N/A	N/A
Decay (Daily Avg)	[73, 159]	2	2	0	0	0	1.0	1.0	1.0
Decay (Rolling Avg)	[45, 142]	2	2	0	0	0	1.0	1.0	1.0
Growth (Daily Avg)	[94]	1	1	0	0	0	1.0	1.0	1.0
Growth (Rolling Avg)	[48, 111]	2	2	0	0	0	1.0	1.0	1.0

Through data preparation, specifically by creating rolling averages, and carefully tuning hyperparameters, the model detected changes within two months of the product launch, without any false positives in gradually changing data.

The model performed well out-of-the-box on streaming data, requiring minimal prior knowledge about the nature of the changes. Its ease of use was further enhanced by having few hyperparameters and automatically adjusting window lengths for optimal performance.

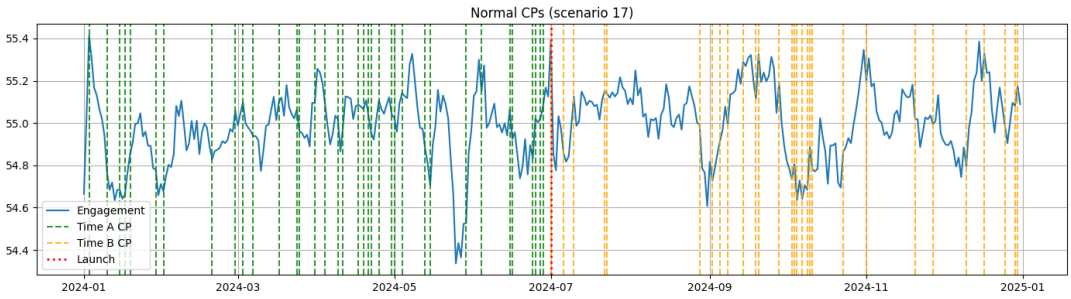
Bayesian Changepoint Detection

Bayesian Changepoint Detection offers a principled framework for detecting structural shifts in time series data, but its performance is sensitive to both the underlying data structure and the choice of hyperparameters. In our implementation, we conducted a grid search over some parameters mentioned in the paper, including the hazard rate, baseline prior strength (κ_0), variance prior dispersion (α_0, β_0), and the posterior reset threshold.

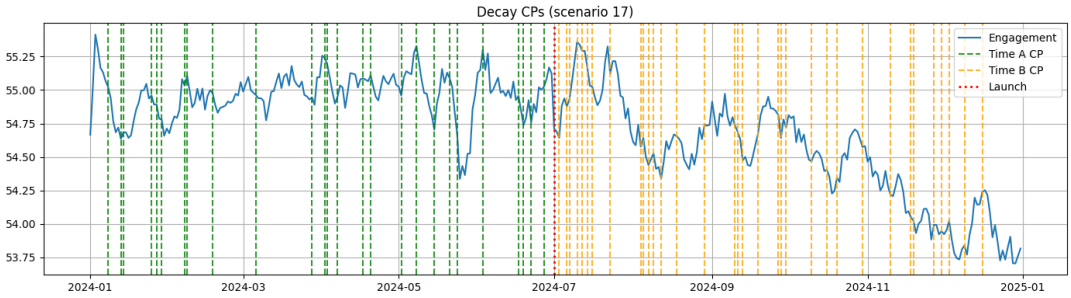
The criteria for success was based on how well changepoints were detected in terms of timing. However, the underlying data, which simulates gradual & noisy behavior, proved very challenging for this model. Even after smoothing the engagement feature with a seven-day rolling average, realistic hyperparameter settings (moderate hazard rates, stronger trust in priors) failed to detect drift in any cohesive manner. The results shown above represent one of the *more reasonable* iterations from the grid search that yielded interpretable outcomes.

However, this semblance of detection was only achieved under extreme settings in the grid search (see Appendix F): low hazard values (hazard = 5), near-zero baseline trust ($\kappa_0 = 1$, meaning the model did not trust the underlying averages), loose variance priors ($\alpha_0 = 0.5$, $\beta_0 = 1$), and a low reset threshold (0.2). These aggressive configurations caused the model to overfit noise, leading to widespread, unstable changepoint detections throughout Time A and Time B. This is not a good result for the model setup and cannot be scaled for implementation.

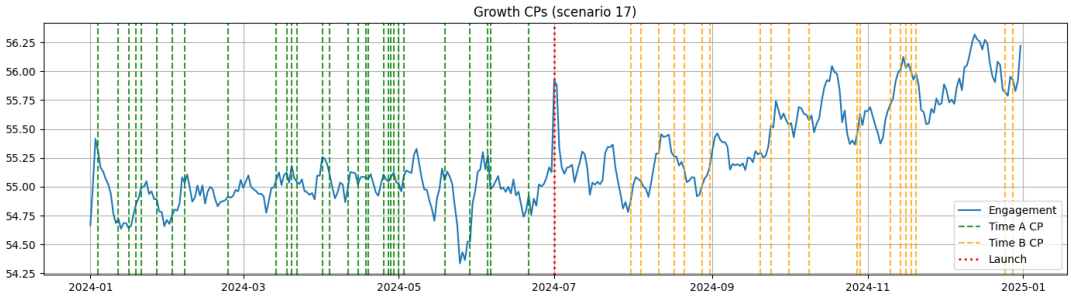
This result is actually expected - with observations from Turner and Van de Meent [8]: in high-noise, low-signal settings, Bayesian drift detectors may require impractical parameters to "fit" to the underlying data. In a more ideal setting with sharper drifts or smoothed data points with more intense changes, higher hazard values (250–500), stronger baseline priors ($\kappa_0 = 50$–100), stronger variance assumptions, and strict thresholds (≥ 0.5) would potentially improve detection. However, under the conditions in this single-variate simulation, such configurations failed to capture the subtle drift altogether.



(a) Normal Time B



(b) Decay Time B



(c) Growth Time B

Figure 10: Bayesian Changepoint Detection on Engagement feature across Normal, Decay, and Growth scenarios. Red dashed lines mark product launch; orange and green lines indicate detected change points. Green lines indicate change points detected in Time A, while orange are change points detected in Time B

Ultimately, we are not recommending using this framework for this sort of problem due to the noisy and gradual drift existing within the data.

CUSUM

CUSUM proved to be highly sensitive to changes, detecting many change points in all `Time B` scenarios. Its sensitivity did however lead to a high rate of false positives, in particular for the normal scenario. For the normal scenario, four change points were detected, resulting in a precision, recall, and F1 score of 0.00.

Normal

- Despite the absence of real drift, CUSUM flagged multiple false change points on both the daily and rolling series even after model tuning.
- Precision, recall, and F1 score were all 0. This implies that CUSUM struggles to maintain a stable baseline even in the absence of true drift.

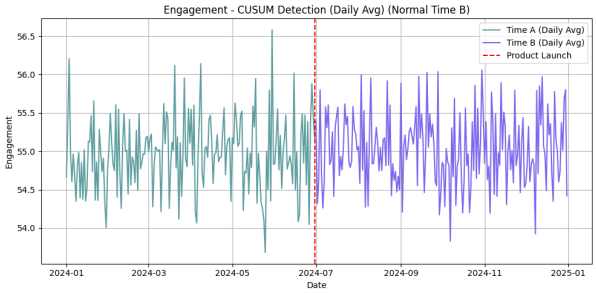
Decay

- CUSUM was able to detect the decay
- 8 false positives were recorded which lowered the precision to 0.11, despite it's perfect recall of 1.0.

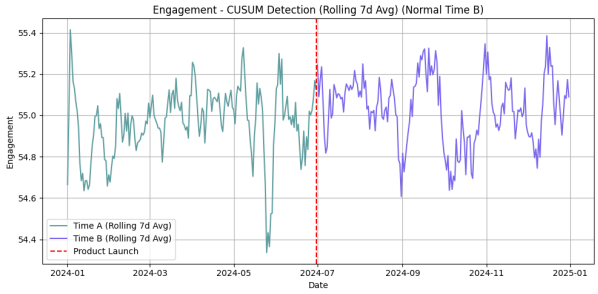
Growth

- Detected growth successfully
- Misidentified 9 false positives
- Low precision at 0.10

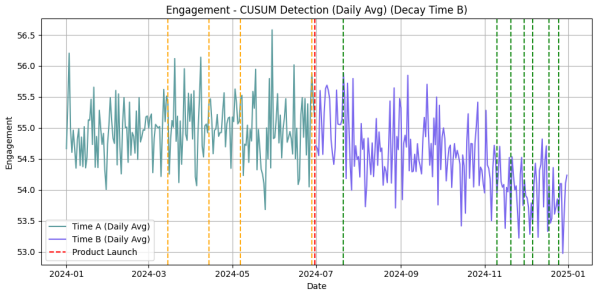
Ultimately, although CUSUM was able to detect the drift for the growth and decay scenarios, due to it's poor performance in the normal scenario, CUSUM is not recommended for the given data.



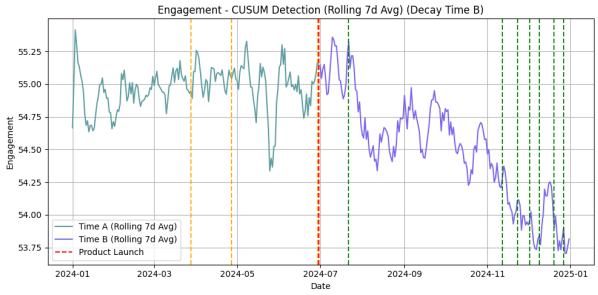
(a) Normal Time B (Daily Avg)



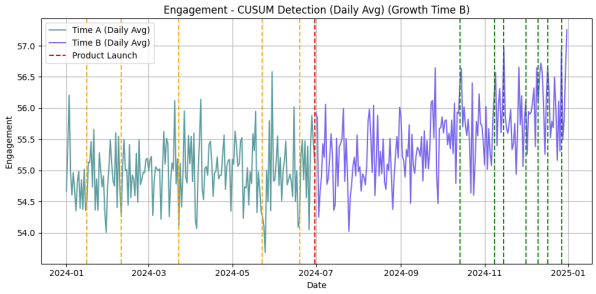
(b) Normal Time B (Rolling 7-Day Avg)



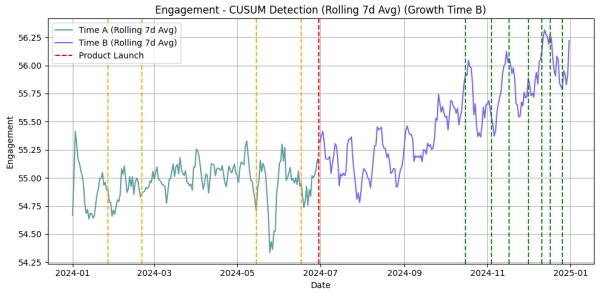
(c) Decay Time B (Daily Avg)



(d) Decay Time B (Rolling 7-Day Avg)



(e) Growth Time B (Daily Avg)



(f) Growth Time B (Rolling 7-Day Avg)

Figure 11: CUSUM Detection across Time B scenarios. Red dashed lines denote product launch; green dashed lines indicate post-launch detected change points; orange dashed lines show false detections pre-launch.

Table 4: Summary of CUSUM Change Point Detection Metrics Across Scenarios

Scenario	Days After Launch	Count	TP	FP	FN	TN	Precision	Recall	F1 Score
Normal (Daily Avg)	N/A	4	0	4	0	1	0.00	0.00	0.00
Normal (Rolling Avg)	N/A	4	0	4	0	1	0.00	0.00	0.00
Decay (Daily Avg)	Multiple	19	1	8	0	0	0.11	1.00	0.20
Decay (Rolling Avg)	Multiple	10	1	3	0	0	0.25	1.00	0.40
Growth (Daily Avg)	Multiple	20	1	9	0	0	0.10	1.00	0.18
Growth (Rolling Avg)	Multiple	11	1	4	0	0	0.20	1.00	0.33

Windowed CUSUM with Log-Likelihoods

Windowed CUSUM proved to be more stable under smoothed data, particularly in the normal scenario. When applied to rolling averages, it did not flag any false positives for the normal group. However, the detector was highly sensitive to noise when using daily averages, resulting in unstable performance. Detection quality depended strongly on the degree of smoothing applied to the data and the chosen window parameters.

Normal

- When applied to the rolling average series, Windowed CUSUM did not detect any false positives, maintaining a stable baseline throughout Time B.
- On the raw daily series, multiple false positives were observed, this is because of the model's sensitivity to noise without smoothing.

Decay

- Successfully detected the decay trend, typically identifying a changepoint approximately one month after the launch.
- No false positives were recorded when using the rolling average series, resulting in high precision.
- Performance deteriorated on the raw daily series again due to noise. Since the result for the normal group was similar to the decay group - we cannot assume that detection in decay was a success.

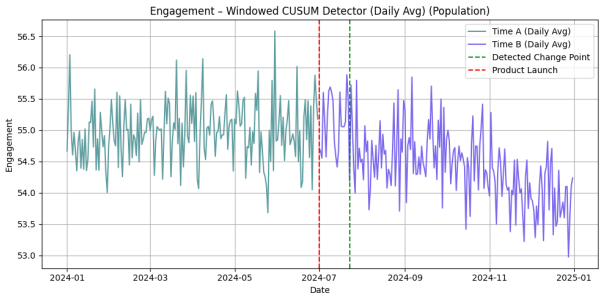
Growth

- Detected growth consistently when applied to rolling averages.
- Performance was less stable on daily averages due to noise.

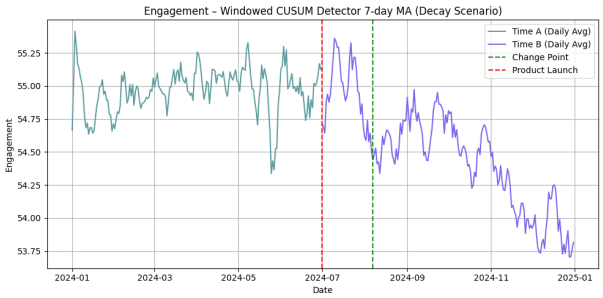
Overall, Windowed CUSUM was effective at detecting major drift events when applied to smoothed engagement data. However, it is highly sensitive to noise and relies on user-defined parameters such as window size and comparison look back period. It's strengths come from it not being sensitive to outliers in smoothed data, and also being able to derive a window and measure error in a simple way.

Table 5: Summary of Windowed CUSUM Change Point Detection Metrics Across Scenarios

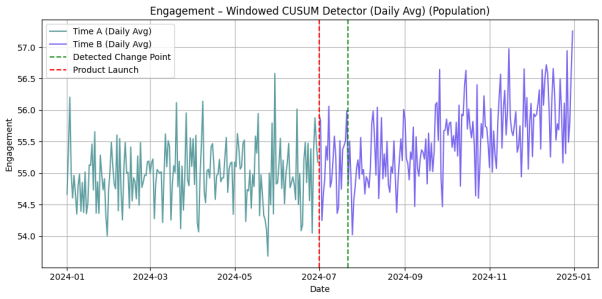
Scenario	Days After Launch	TP	FP	FN	TN
Normal (Daily Avg)	34	0	1	0	0
Normal (Rolling Avg)	N/A	1	0	0	1
Decay (Daily Avg)	22	1	0	0	0
Decay (Rolling Avg)	37	1	0	0	0
Growth (Daily Avg)	21	1	0	0	0
Growth (Rolling Avg)	44	1	0	0	0



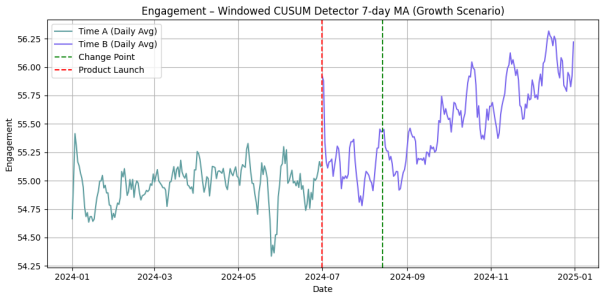
(a) Decay Time B (Daily Avg)



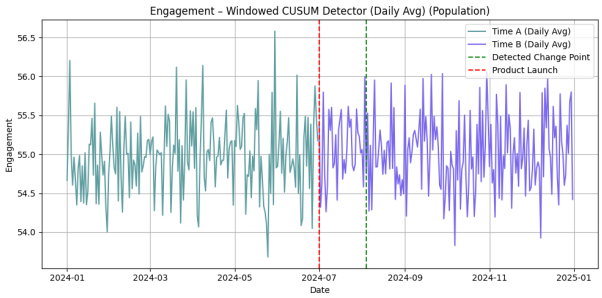
(b) Decay Time B (Rolling 7-Day Avg)



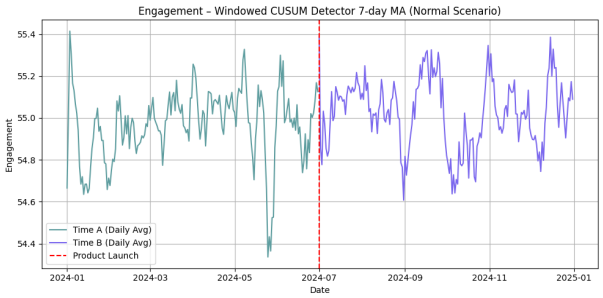
(c) Growth Time B (Daily Avg)



(d) Growth Time B (Rolling 7-Day Avg)



(e) Normal Time B (Daily Avg)



(f) Normal Time B (Rolling 7-Day Avg)

Figure 12: Windowed CUSUM detection results across decay, growth, and normal scenarios. Red dashed lines represent the product launch; green dashed lines indicate detected change points.

Conclusion

Model Comparisons

1. **Windowed CUSUM with Log-Likelihoods**
   (a) *Strengths*
      i. Works well with smoothed data
      ii. Conceptually sound - uses log-likelihood to measure error and compares windowed-views

(b) *Weaknesses*

i. Some manual implementation is required

ii. Parameters are heuristic-based - some parameter setups offer similar performances

2. **ADWIN**

(a) *Strengths*

i. Automatically adjusts window lengths to perform optimally, decreasing a hyperparameter for the user to define

ii. Performed well even with slightly noisy data, although does perform better with smoothed rolling averages. No false positives and fast detection time

(b) *Weaknesses*

i. Parameters are heuristic-based and many parameter setups offer similar performance. It can be difficult to fine tune the model for very sensitive cases

3. **PELT with RBF**

(a) *Strengths*

i. Easy to set up and use

ii. Works well with non-linear data

iii. No false positives on daily average data

(b) *Weaknesses*

i. Requires a lot of computational resources

ii. Does not perform as well on rolling average data (1 FP in growth scenario)

iii. Prone to overfitting to noise

iv. Requires a penalty term

4. **Kolmogorov-Smirnov**

(a) *Strengths*

i. Makes no assumptions regarding the distribution of the data

ii. Simple to set up

(b) *Weaknesses*

i. Sensitive to noisy data with outliers

ii. Extremely weak performance on rolling data

iii. Many false positives in daily and rolling data across each scenario

5. **Bayesian Changepoint**

(a) *Strengths*

i. Trains on the entire dataset - is able to handle *online/live* data

ii. Works well with sharp changes within the data

(b) *Weaknesses*

i. Weak at detecting gradual shifts

ii. Difficult to implement - (no plug and play package)

iii. Extremely susceptible to noise

iv. Parameters are difficult to grid-search and evaluate

6. **CUSUM**

(a) *Strengths*

i. Quickly detects small shifts in the mean

(b) *Weaknesses*

i. Requires extremely precise tuning for a usable model

Overall, each model has their own strengthens and weaknesses, and scenarios that they would excel in over the others. For the particular dataset that was generated for the purpose of this project, the Windowed CUSUM with Log-Likelihood performed the best. ADWIN and PELT with RBF also performed very strongly with the simulated data. CUSUM, KS, and Bayesian Change Point Detection performed very poorly due to the lack of alignment between the simulated data and each model's respective strength.

Table 6: Comparison of Change Point Detection Methods

Model	**Use Cases**	**Ease of Use**
Windowed CUSUM	Gradual shifts in the mean	Medium
ADWIN	Automatic window length	Easy
PELT with RBF	Non-linear time series	Easy
KS	Sharp drifts with minimal noise	Medium
Bayesian Changepoint	Sharp drifts with minimal noise	Hard
CUSUM	Small consistent shifts	Hard

Future Work & Final Thoughts

The first step towards future work for such a project would include further fine tuning of each model and the generation of different types of data. For example, create data with much less noise, or sharper shifts, or smoother shifts, etc. We expect that model performance would differ and we could get a better understanding of scenarios to use each model when being particular sensitive to either gradual or sudden changes. This can support the appropriate deployment of a particular model to allow quick decision-making when a change is detected.

Furthermore, it would be interesting to combine some of the models developed (and possibly combine with models that were not tested within the scope of this project) to create a a multi-layered approach to change detection. This would ensure that the data is being filtered through several types of data to minimize noise and capture real drift.

Finally, many of our models were tested on a single variable. Multivariate shift could also be important to analyze when viewing if there is concept drift (or in this case use behavior changes). These may require testing different anomaly detection (such as One-Class SVM) or modifications of some of the methods tested.

Task Breakdown

Table 7: Task Assignments

Task	**Member**
Project Idea Selection	Daniel Tabach
Data Generation	Daniel Tabach
EDA	Kristen Hart, Victor Chupka
Kolmogorov Smirnov Development	Kristen Hart
CUSUM Development	Kristen Hart
PELT with RBF Development	Kristen Hart
ADWIN Development	Victor Chupka
Bayesian Change Point Detection Development	Daniel Tabach
Windowed CUSUM	Daniel Tabach
Code Review	Everyone
Report Write Up	Everyone

Note: Please note that all team members provided an equal amount of effort. We met weekly to discuss issues and ideas for each of our tasks. The main owner of the task is listed, but we all contributed to each step.

Appendix

A Pruned Exact Linear Time with Radial Basis Cost Function

The PELT algorithm identifies change points by minimizing the cost function:

$$\text{minimize} \quad \sum_{i=0}^{m} \left[C(y_{\tau_i+1:\tau_{i+1}}) \right] + \beta m$$

where:

- $C(\cdot)$ is a segment cost function.
- β is a penalty term controlling the number of change points.
- $\{\tau_1, \tau_2, \ldots, \tau_m\}$ are the change points.

The cost function $C(\cdot)$ is calculated by using the Radial Basis Function (RBF) kernel which enables the detection of nonlinear changes in the time series data.

For this project, the β term was set to 10. The RBF kernel that was applied was

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right)$$

The γ value was set to its default setting to allow for automatic scaling based on feature dimensions.

B CUSUM

The Cumulative Sum (CUSUM) method monitors cumulative deviation from the mean μ_0. The positive and negative CUSUM statistics at time t are given by:

$$S_t^+ = \max(0, S_{t-1}^+ + x_t - \mu_0 - k)$$

$$S_t^- = \max(0, S_{t-1}^- + \mu_0 - x_t - k)$$

such that k is the drift allowance parameter. The drift is flagged if either S_t^+ or S_t^- exceeds a specific threshold h. For this project, h was set to a threshold of 12, and the drift allowance k was set to 0.01.

C Windowed CUSUM with Log-Likelihoods

The Windowed CUSUM Detector models pre-launch engagement (Time A) as a Gaussian distribution with mean μ_A and variance σ_A^2. Post-launch (Time B), it computes the cumulative log-likelihood over sliding windows of size k:

$$\log \mathcal{L}(w_t) = \sum_{i=0}^{k-1} \log \left(\frac{1}{\sqrt{2\pi\sigma_A^2}} \exp\left(-\frac{(x_{t+i} - \mu_A)^2}{2\sigma_A^2} \right) \right)$$

At each step, it compares $\log \mathcal{L}(w_t)$ to the average log-likelihood of the past m windows ($m = 5$) in this case. A drift is flagged if:

$$\log \bar{\mathcal{L}}_{t-5:t-1} - \log \mathcal{L}_t > \delta$$

The method accumulates evidence over time, flagging gradual shifts rather than reacting to single outliers.

D Kolmogorov Smirnov

The Kolmogorov Smirnov (KS) test compares two cumulative distribution functions. Given a pre-launch sample $F_n(x)$ and a post-launch sample $G_m(x)$ the KS statistic is defined by:

$$D_{n,m} = \sup_x |F_n(x) - G_m(x)|$$

such that sup is the supremum of the absolute differences between the pre and post-launch samples. A drift is then detected if the p value of $D_{n,m}$ falls below the chosen significance threshold that was tuned to 0.05.

E ADWIN

ADWIN (Adaptive Windowing) is a sliding window algorithm that detects changes in a data stream. It holds a window of recent data and then automatically shrinks the window when a significant enough change is detected. A change is then flagged when the means of two sub-windows differ by more than a specific threshold. Let W be the current window. It is then split into two sub-windows W_0 and W_1 and drift is recorded when:

$$\mu_{W_0} - \mu_{W_1} > \epsilon_{\text{cut}}$$

where ϵ_{cut} is a confidence bound that is based on Hoeffding's inequality [5] [2]. At every step, the probability that ADWIN detects a change given that the new mean is the same is at most δ. For this project, we tested many different values for δ, but used $\delta = 0.05$ for the final model.

F Bayesian Changepoint Detection

[1] [8]

1. **Initialize**

   $$P(r_0 = 0) = 1$$

   Start with the assumption that no changepoint has occurred yet; the run length is zero.

2. **Observe new data point** x_t
   Receive the next observation in the time series.

3. **Evaluate predictive probability for each possible previous run length** r_{t-1}

   $$\pi_t^{(r_{t-1})} = P(x_t \mid \alpha_{t-1}^{(r_{t-1})}, \beta_{t-1}^{(r_{t-1})})$$

   Estimate how likely the new data point is based on the model of past stable behavior associated with each potential run length. a "run" just means a sequence of data points where the underlying mean and variance are assumed to remain the same. When this changes: we assume a "changepoint" occurred.

4. **Calculate growth probability for each run length (no changepoint)**

$$C \cdot P(r_t = r_{t-1} + 1 \mid x_{1:t}) = (1 - H(r_{t-1} + 1))\, \pi_t^{(r_{t-1})}\, P(r_{t-1} \mid x_{1:t-1})$$

Update the probability that the same "regime" continues, assuming no changepoint has occurred and the data remains consistent with the previous observations in the data.

5. **Calculate changepoint probability**

$$C \cdot P(r_t = 0 \mid x_{1:t}) = \sum_{r_{t-1}} H(r_{t-1} + 1)\, \pi_t^{(r_{t-1})}\, P(r_{t-1} \mid x_{1:t-1})$$

Calculate the probability that a changepoint has occurred at the current time step, based on how surprising the new observation is relative to past data.

6. **Normalize to find full run-length distribution**

$$P(r_t \mid x_{1:t}) = \frac{C \cdot P(r_t \mid x_{1:t})}{\sum_{r_t} C \cdot P(r_t \mid x_{1:t})}$$

Adjust all the calculated probabilities so that they form a valid probability distribution that sums to one.

7. **Return to Step 2 for the next data point**
Proceed to the next observation and repeat the detection process continuously over time.

G Parameter Definitions

- **Hazard Rate** (h): Defines the expected frequency of changepoints. A constant hazard function assumes that a changepoint occurs independently at each time step with probability $\frac{1}{h}$.

$$P(r_t = 0 \mid r_{t-1}) = \frac{1}{h} \quad \text{and} \quad P(r_t = r_{t-1} + 1 \mid r_{t-1}) = 1 - \frac{1}{h}$$

Interpretation: A lower h value increases the model's expectation of frequent changes, making it more sensitive to noise.

- **Baseline Mean Prior Strength** (κ_0): Controls the model's confidence in the initial mean of the data distribution. The less confident it is, the more likely it will determine that a change is occurring.

$$\text{Posterior mean update:} \quad \mu_n = \frac{\kappa_0 \mu_0 + n\bar{x}}{\kappa_0 + n}$$

Interpretation: A larger κ_0 gives more weight to the initial baseline estimate of the mean, - it's intended to stabilize potential noise. A smaller κ_0 causes the model to rapidly adapt to incoming data.

- **Baseline Variance Prior Strength** (α_0, β_0): Controls the model's confidence in the initial variance of the data distribution.

$$\text{Posterior variance estimate:} \quad \sigma_n^2 = \frac{\beta_n}{\alpha_n}$$

where α_n and β_n are updated based on observed data.
Interpretation: - Lower values of α_0 and β_0 produce a wider, more uncertain prior, causing the model to underestimate natural noise and react more readily to fluctuations. - Higher values encourage a more conservative estimate of variability.

- **Threshold** (δ): Defines the minimum posterior probability required to declare a changepoint.

$$\text{Flag changepoint if} \quad P(r_t = 0 \mid x_{1:t}) > \delta$$

Interpretation: - A lower threshold (e.g., $\delta = 0.2$) allows weak evidence to trigger resets, making the model highly sensitive but prone to false positives. - A higher threshold (e.g., $\delta = 0.7$) demands stronger evidence before accepting a changepoint.

References

[1] Ryan P. Adams and David J. C. MacKay. Bayesian online changepoint detection. In *Proceedings of the 2007 Neural Information Processing Systems (NeurIPS)*, 2007. Original BOCPD paper.

[2] Albert Bifet and Ricard Gavaldà. Learning from time-changing data with adaptive windowing. *Proceedings of the 2007 SIAM International Conference on Data Mining (SDM)*, pages 443–448, 2007.

[3] Centre Borelli Developers. ruptures documentation. `https://centre-borelli.github.io/ruptures-docs/`, 2024. Library for Change Point Detection.

[4] Hongwen Guo, Frederic Robin, and Neil Dorans. Detecting item drift in large-scale testing. *Journal of Educational Measurement*, 54(3):265–284, 2017.

[5] O. Krafft and N. Schmitz. A note on hoeffding's inequality. *Journal of the American Statistical Association*, 64(327):907–912, 1969.

[6] River ML Developers. Adwin — drift detection. `https://riverml.xyz/dev/api/drift/ADWIN/`, 2024. River Machine Learning Library Documentation.

[7] SciPy Developers. scipy.stats.ks_2samp — scipy v1.11.3 manual. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ks_2samp.html`, 2024. SciPy Documentation.

[8] Richard E. Turner and Jan-Willem van de Meent. Bayesian time series models and the standard model of bocpd. `https://talks.cam.ac.uk/talk/index/44372`, 2013. Presented at the University of Cambridge Machine Learning Group Seminar.