FEATURE-INTERPRETABLE REAL CONCEPT DRIFT DETECTION

Pranoy Panda, Vineeth N Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad
`{cs20mtech12002, vineethnb}@iith.ac.in`

Gaurav Sinha
Microsoft Research India
`{sinhagaur88}@gmail.com`

ABSTRACT

Classifiers deployed in production degrade in performance due to changes in the posterior distribution, a phenomenon referred to as real concept drift. Knowledge of such distribution shifts is helpful for two main reasons: (i) it helps retain classifier performance across time by telling us when to retrain it; and (ii) understanding the nature of shift in the relationship between input features and output labels, which can be of value for business analytics (e.g., understanding change in demand helps manage inventory) or scientific study (e.g., understanding virus behavior across changing demographics helps distribute drugs better). An interpretable real concept drift detection method is ideal for achieving this knowledge. Existing interpretable methods in this space only track covariate shifts, thus, are insensitive to the optimal decision boundary (true posterior distribution) and vulnerable to benign drifts in streaming data. Our work addresses this issue by proposing an interpretable method that leverages gradients of a classifier in a feature-wise hypothesis-testing framework to detect real concept drift. We also extend our method to a more realistic unsupervised setting where labels are not available to detect drift. Our experiments on various datasets show that the proposed method outperforms existing interpretable methods and performs at par with state-of-the-art supervised drift detection methods w.r.t the average model classification accuracy metric. Qualitatively, our method identifies features that are relevant to the drift in the USENET2 dataset, thus providing interpretability and accurate drift detection.

1 INTRODUCTION

Concept drift is a phenomenon where a given data distribution changes over time. Based on an application setting, this could manifest as either change in distribution of data covariates or change in data-label posterior distribution, or a mixture of both. Gama et al. (2014) termed change in posterior distribution as *real concept drift*, while change in data covariates which does not change the optimal decision boundary as *virtual concept drift*. The relationship between input and output variables is of high value in machine learning tasks, as change in this relationship can degrade performance of a classifier Khamassi et al. (2018). We therefore focus on the real concept drift detection problem in this work. In particular, our objective is to detect real concept drift in data streams and also interpret the drift in terms of input features, i.e., localize features related to the drift.

Interpretable real concept drift detection can be beneficial for knowledge discovery in various applications ranging from business/e-commerce analytics to cybersecurity. Many real-world streaming applications suffer from real concept drift. Examples include monitoring devices or machinery Toubakh & Sayed-Mouchaweh (2015), malware detection Jordaney et al. (2017), credit card fraud detection Salazar et al. (2012); Dal Pozzolo

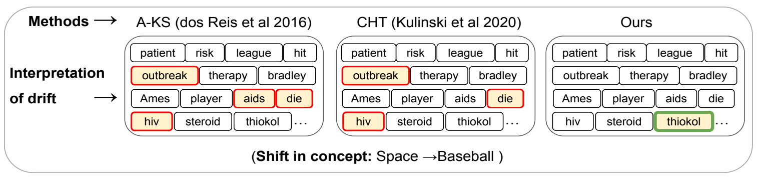
Methods →
A-KS (dos Reis et al 2016)
CHT (Kulinski et al 2020)
Ours
Interpretation of drift →
patient
risk
league
hit
outbreak
therapy
bradley
Ames
player
aids
die
hiv
steroid
thiokol
...
(Shift in concept: Space →Baseball)

Figure 1: Illustrative results of feature-interpretable drift detection on USENET2 dataset. Cause of drift here is shift in user interest from *Space* to *Baseball*. Features (here words) responsible for drift w.r.t different methods are highlighted in yellow. Highlighted boxes are outlined in green if the selected word/feature is semantically related to drift concept, and red otherwise. (*Thiokol* = rocket systems org.)

et al. (2015), predictive maintenance Zenisek et al. (2019). The wide presence of this problem in data streams underscores the value of such methods in practice, especially with the increasing need for interpretability in machine learning.

Real concept drift has been studied in the literature for more than two decades (Pesaranghader et al. (2018); Bifet & Gavalda (2007); Baena-Garcıa et al. (2006); Gama et al. (2004)). Similarly, interpretable drift detection methods such as A-KS (dos Reis et al. (2016); Žliobaite (2010)), CHT (Kulinski et al. (2020)) also exist, but they operate only in the covariate space (virtual concept drift) and hence cannot detect a change in the optimal decision boundary. Existing post-hoc explainability methods are not intended for distribution shifts, and thus give unreliable explanations for concept drifts. For instance, when the relationship between features changes between distributions in drift settings, the explanations produced have lower fidelity Lakkaraju et al. (2020), and hence are not trustworthy. While some recent efforts have been made to improve robustness of explanations to local adversarial perturbations Lakkaraju et al. (2020), these do not capture the setting when the given data distribution changes by itself over time. Thus, *interpreting real-concept drift* is still an open problem. Our work aims to address this problem using classifier gradients in a feature-wise hypothesis testing framework.

To understand the benefits of interpretable real-concept drift detection consider this problem of understanding user preferences: say we have a user-based predictive model that classifies text read by users as engaging (or interesting) or not engaging. Since user interest can change over time, detecting when a predictive model is outdated (or has drifted) is necessary to ensure that the classifier stays accurate. Figure 1 shows this example (on the USENET2 dataset), where input features are words from email messages, and the output label indicates whether an email is interesting to a user or not. Here, drift is caused due to change in user interest from one concept to another. As can be seen from the figure, our method can uncover relevant features for a drift in user interest from space to baseball, as opposed to interpretable virtual drift detectors (A-KS and CHT).

We summarize our key contributions below.

- We propose an *interpretable* real concept drift detection method that tracks changes in posterior distribution by relying only on the classifier gradients and no other auxiliary post-hoc explainer.
- We extend our framework to a more realistic unsupervised setting where we do not require labels immediately to detect drift.
- We analyze our method and show that our feature-wise test statistic's power converges.
- We quantitatively evaluate the drift detection capability of our method on eight datasets, consisting of a mixture of synthetic, semi-real, and real-world datasets.

Below we start by explaining our proposed method. We include the related works in Appendix A.

2 Proposed Methodology

2.1 Notations and Preliminaries

We represent random variables using capital letters and values taken by these variables using small letters. Unless otherwise specified, bold faced letters are used for vectors or sets of variables. For each positive integer n, we denote the set $\{1, \ldots, n\}$ by $[n]$ and the set $\{m, m+1, \ldots, n\}$ by $[m, n]$. The symbols $\mathbb{P}$ and $\mathbb{E}$ will be used to represent probability and expectation. Let X be a random variable and $p(X)$ be a probability distribution over X. By $x \sim p(X)$, we mean that x is obtained by sampling from the distribution $p(X)$. Throughout this work we will represent our covariates by $\boldsymbol{X}$ and assume that $\boldsymbol{X}$ takes values in an open convex set $\mathcal{X} \subset \mathbb{R}^d$. For any differentiable function $f : \mathbb{R}^d \to \mathbb{R}$, we denote its k^{th} ($k \in [d]$) partial derivative $\frac{\partial f}{\partial x_k}$ by $\partial_k f$ and the gradient vector by $\nabla_{\boldsymbol{x}} f = (\partial_1 f, \ldots, \partial_k f)$. We use the term "reference window" to denote a stream of labelled samples $\{(\boldsymbol{x}_t, y_t)\}_{t=1}^{T}$ where $\boldsymbol{x}_t \in \mathbb{R}^d$ represents an input feature vector and y_t is its corresponding class label. A "detection window" is a stream of samples that appears after the reference window (i.e. after $t = T$). Samples in the detection window might or might not contain labels (depending on whether we are in the supervised or unsupervised setting). Next, we define real concept drift and feature driven real concept drift, which is the main kind of drifts we are interested in.

Definition 2.1 (Real Concept Drift)**.** *Consider a stream of samples $\{(\boldsymbol{x}_t, y_t) : t = 1, 2, \ldots\}$. We say that a real concept drift occurs at time $t = T$, if there exist two distributions $p(\boldsymbol{X}, Y), q(\boldsymbol{X}, Y)$ on the joint variable $(\boldsymbol{X}, Y)$, such that*

1. *$(\boldsymbol{x}_t, y_t) \sim p(\boldsymbol{X}, Y)$ for $t \leq T$,*
2. *$(\boldsymbol{x}_t, y_t) \sim q(\boldsymbol{X}, Y)$ for $t > T$, and*
3. *$p(y|\boldsymbol{x}) \neq q(y|\boldsymbol{x})$, for some $\boldsymbol{x} \in \mathcal{X}, y \in \mathcal{Y}$.*

Definition 2.2 (Feature Driven Real Concept Drift). *Consider a stream of samples $\{(\boldsymbol{x}_t, y_t) : t = 1, 2, \ldots\}$. We say that a feature driven real concept drift occurs at time $t = T$, if there exist two distributions $p(\boldsymbol{X}, Y), q(\boldsymbol{X}, Y)$ (assuming $q(Y|\boldsymbol{X}) > 0$) on joint variable $(\boldsymbol{X}, Y)$, such that,*

1. *for each $y \in \mathcal{Y}$, $p(y|\boldsymbol{x}), q(y|\boldsymbol{x})$ are differentiable w.r.t. $\boldsymbol{x}$.*
2. *$(\boldsymbol{x}_t, y_t) \sim p(\boldsymbol{X}, Y)$ for $t \leq T$,*
3. *$(\boldsymbol{x}_t, y_t) \sim q(\boldsymbol{X}, Y)$ for $t > T$, and*
4. *$\nabla_{\boldsymbol{x}} \left(\frac{p(y|\boldsymbol{x})}{q(y|\boldsymbol{x})} \right) \neq 0$, for some $\boldsymbol{x} \in \mathcal{X}, y \in \mathcal{Y}$.*

In order to understand the difference between the above definitions, note that, when $q(Y|\boldsymbol{X}) > 0$, $p(y|\boldsymbol{x}) \neq q(y|\boldsymbol{x})$, implies that either $\frac{p(y|\boldsymbol{x})}{q(y|\boldsymbol{x})}$ depends non trivially on $\boldsymbol{x}$ (equivalent to $\nabla_{\boldsymbol{x}} \left(\frac{p(y|\boldsymbol{x})}{q(y|\boldsymbol{x})} \right) \neq 0$[1]) or equals $h(y)$ for some non-trivial function $h : \mathcal{Y} \to \mathbb{R}$. Therefore, the only drifts we do not consider in Definition 2.2 are the ones where the conditional probability changes by a multiple that depends only on y and not on $\boldsymbol{x}$. It's easy to see that we can replace condition (4) in Definition 2.2 with $\nabla_{\boldsymbol{x}} \left(\log \frac{p(y|\boldsymbol{x})}{q(y|\boldsymbol{x})} \right) \neq 0$, for some $\boldsymbol{x} \in \mathcal{X}, y \in \mathcal{Y}$. This is true since $\nabla_{\boldsymbol{x}} \left(\frac{p(y|\boldsymbol{x})}{q(y|\boldsymbol{x})} \right) = 0, \forall \boldsymbol{x} \in \mathcal{X}$[2] $\Leftrightarrow$ (using mean value theorem) $\frac{p(y|\boldsymbol{x})}{q(y|\boldsymbol{x})}$ is a function of $y \Leftrightarrow \nabla_{\boldsymbol{x}} \left(\log \frac{p(y|\boldsymbol{x})}{q(y|\boldsymbol{x})} \right) = 0, \forall \boldsymbol{x} \in \mathcal{X}$.

2.2 Feature-wise Hypothesis Test

Existing interpretable drift detection methods for streaming data can only cater to changes in the covariate distribution $p(\boldsymbol{X})$, i.e., technically, they are virtual drift detection methods. On the other hand, the goal of interpretable real concept drift detection (Defintion 2.1) is to detect a change in the conditional distribution $p(Y|\boldsymbol{X})$, and at the same time reveal the features that are responsible for this change. Unfortunately, as discussed in Section 2.1, when there exists a non-trivial function $h : \mathcal{Y} \to \mathbb{R}$ such that $p(y|\boldsymbol{x}) = h(y)q(y|\boldsymbol{x}), \forall y \in \mathcal{Y}, \boldsymbol{x} \in \mathcal{X}$, we will observe a Real Concept Drift which is captured by the function $h(y)$ and not by the conditional probabilities. This motivates us to use the Feature Driven Real Concept Drift (Definition 2.2) instead. In this section we design hypothesis tests which help us detect feature driven real concept drifts.

Hypothesis Tests: Since we wish to obtain an interpretable result, we use a feature-wise hypothesis testing framework i.e., for each feature in the covariates $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$, we perform a hypothesis test to check if the feature is able to detect a change in the conditional distribution. Formally, for the k^{th} feature ($k \in [d]$), our test essentially captures the change in the partial derivative $\partial_k \log p(y|\boldsymbol{x})$. If even one feature detects a significant change, we signal a drift. The set of features signaling the drift comprise our interpretation of the drift. Let $p(\boldsymbol{X}, Y), q(\boldsymbol{X}, Y)$ be the joint distribution of the streaming data pre and post the time stamp T under consideration. The hypothesis test corresponding to the k^{th} feature has the following null hypothesis $\mathbf{H}_0$ and alternate hypotheses $\mathbf{H}_a$.

$\mathbf{H}_0$: For all $\boldsymbol{x} \in \mathcal{X}, y \in \mathcal{Y}$, $\quad \partial_k \log(p(y|\boldsymbol{x})) = \partial_k \log(q(y|\boldsymbol{x}))$

$\mathbf{H}_a$: $\exists A \subset \mathcal{X}$ with lebesgue measure > 0 and $y \in \mathcal{Y}$, such that,

$$\partial_k \log(p(y|\boldsymbol{x})) \neq \partial_k \log(q(y|\boldsymbol{x})) \;\; \forall \boldsymbol{x} \in A$$

Recall that we signal a drift if and only if $\mathbf{H}_0$ is rejected for some feature $k \in [d]$. Under the assumptions mentioned in Definition 2.2, we make the following observations.

1. If $\mathbf{H}_0$ is true $\forall k \in [d]$, then there is no feature driven real concept drift and vice versa.
2. If $\mathbf{H}_a$ is true for some $k \in [d]$, then there is a feature driven real concept drift.
3. If $\mathbf{H}_a$ is false $\forall k \in [d]$, then for each $y \in \mathcal{Y}$, $\nabla_{\boldsymbol{x}} \left(\frac{p(y|\boldsymbol{x})}{q(y|\boldsymbol{x})} \right) = 0$, almost surely (w.r.t. lebesgue measure) on $\mathcal{X} \subset \mathbb{R}^d$.

Test statistic: Given a stream of samples $\mathcal{S} = \{(\boldsymbol{x}_i, y_i) \sim p(X, Y) : i \in [n]\}$, along with the two distributions $p(Y|X), q(Y|X)$, we define our test statistic for k^{th} feature:

$$c_n^k(p, q) = \sum_{i=1}^{n} \left(\partial_k \log p(y_i|\boldsymbol{x}_i) - \partial_k \log q(y_i|\boldsymbol{x}_i) \right)^2 \tag{1}$$

In practice we will use the plug-in estimator $c_n^k(\widehat{p}, \widehat{q})$ of $c_n^k(p, q)$, where $\widehat{p}, \widehat{q}$ are estimates (learned classifiers) of the two conditional distributions $p(Y|X), q(Y|X)$ respectively.

[1]under differentiability assumption

[2]we assume $\mathcal{X} \subset \mathbb{R}^d$ is an open convex set

Remarks: We want to mention that given $\widehat{p}, \widehat{q}$ and $\mathcal{S}$, the above statistic can be simultaneously computed for all $k \in [d]$ by computing the gradients of $\widehat{p}, \widehat{q}$ (say using back-propagation) and then plugging in the respective co-ordinates into each of the tests. This simultaneous calculation helps us significantly reduce the dependency of the time complexity on the covariate dimension and therefore makes our statistic scale easily to high-dimensional datasets.
Next, we analyze the power of our test statistic. In particular, we show that under the alternate hypothesis, for any fixed t, the probability that our statistic is greater than t tends to 1 as $n \to \infty$. To prove this, we assume that for all $y \in \mathcal{Y}$, $p(y) > 0$, and $p(\boldsymbol{X}|y)$ is strictly positive almost surely (w.r.t. lebesgue measure), i.e. lebesgue measure of $B_y = \{\boldsymbol{x} : p(\boldsymbol{x}|y) = 0\}$ is 0. We state the proposition below and include the proof in Appendix B.

Proposition 2.1 (Convergence of Power of our Test Statistic)**.** *If the alternate hypothesis $\boldsymbol{H}_a$ is true for some feature $k \in [d]$, then, for any $t > 0$,*

$$\lim_{n\to\infty} \mathbb{P}[c_n^k(p, q) > t] = 1$$

Supervised Drift Detection: Here, we consider the supervised case, i.e., when samples in the detection window contain both covariates and labels. We explain the procedure in detail in Algorithm 1 and give a summary of the main steps here. Our algorithm uses a moving window framework on the input stream $\mathcal{S} = \{\boldsymbol{s}_t = (\boldsymbol{x}_t, y_t) : t = 1, 2 \ldots\}$ and iteratively applies the hypothesis testing methodology from Section 2.2 to continuously detect drifts. To perform the test we need to train classifiers and compute the statistic, and both of these should be done on disjoint samples. Let r be the ratio of the amount of samples used for training the model to amount of samples used for calculating the statistic. In the first iteration, we train classifiers $\widehat{p}$ and $\widehat{q}$ using the first $\lfloor nr \rfloor$ samples in reference window $\mathcal{S}_R = \{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n\}$ and detection window $\mathcal{S}_D = \{\boldsymbol{s}_{n+1}, \ldots, \boldsymbol{s}_{2n}\}$ respectively. Then, we compute the scores $c_k^n(\widehat{p}, \widehat{q})$ using $\mathcal{S}_R$ (last $n - \lfloor nr \rfloor$ samples are used), $\widehat{p}, \widehat{q}$ for all $k \in [d]$ as defined in Equation 1. Using a significance value α and number of bootstraps K as inputs to our algorithm, we compute a threshold T_α^k for each $k \in [d]$ by using a bootstrapping procedure explained in Algorithm 3 in Appendix D. Once we have these thresholds, we identify the features $k \in [d]$, where $c_n^k(\widehat{p}, \widehat{q}) > T_\alpha^k$. We store all the features for which this happens in a set called "Explanation", and report a drift with Explanation being the set of features that explains this drift. Then we move our reference window to $\mathcal{S}_R = \{\boldsymbol{s}_{n+1}, \ldots, \boldsymbol{s}_{2n}\}$ and detection window to $\mathcal{S}_D = \{\boldsymbol{s}_{2n+1}, \ldots, \boldsymbol{s}_{3n}\}$. Since, we already learned $\widehat{q}$ on this new reference window, we update $\widehat{p} = \widehat{q}$. If for all $k \in [d]$, $c_n^k \leq T_\alpha^k$, we do not report a drift and shift our reference and detection windows by δ (input to our algorithm). Since no drift was found $\widehat{p}$ remains unchanged. The algorithm then continues with these new reference and detection windows and classifier $\widehat{p}$. We refer the reader to Appendix D for complete details on the bootstrapping technique.

Algorithm 1 Supervised Interpretable Drift Detection

```
Require: $n, \alpha, K, \delta, \mathcal{S} = \{\boldsymbol{s}_t = (\boldsymbol{x}_t, y_t) : t = 1, 2, \ldots\}, r$
  Initialize $i \leftarrow 0$
  $\widehat{p} \leftarrow$ GETCLASSIFIER($\{\boldsymbol{s}_1, ..., \boldsymbol{s}_n\}$, $\lfloor nr \rfloor$)        ▷ use first $\lfloor nr \rfloor$ pts
  Create empty list Explanation $\leftarrow \phi$.
  while $True$ do
      Flag $\leftarrow False$                                         ▷ Drift flag
      $\mathcal{S}_R = \{\boldsymbol{s}_t : t \in [i+1, i+n]\}$.
      $\mathcal{S}_D = \{\boldsymbol{s}_t : t \in [i+n+1, i+2n]\}$.
      $\widehat{q} \leftarrow$ GETCLASSIFIER($\mathcal{S}_D$, $\lfloor nr \rfloor$)        ▷ train on first $\lfloor nr \rfloor$ pts
      Compute $c_n^k(\widehat{p}, \widehat{q}), \forall k \in [d]$, using the last $n - \lfloor nr \rfloor$ samples in
  $\mathcal{S}_R, \widehat{p}, \widehat{q}$ in Equation 1.
      Get thresholds $(T_\alpha^1, \ldots, T_\alpha^d) \leftarrow$ BOOTSTRAP1($\widehat{p}, \widehat{q}, \alpha, K, \mathcal{S}_R, r$)
      if for any $k \in [d]$, $c_n^k(\widehat{p}, \widehat{q}) > T_\alpha^k$ then
          Flag $\leftarrow True$, Add all such $k$s to Explanation.
      end if
      if Flag $= True$ then                                       ▷ Drift detected
          $i \leftarrow i + n$, and $\widehat{p} \leftarrow \widehat{q}$                      ▷ Shift windows by $n$
          Print the list Explanation and reset it to $\phi$.
      else
          $i \leftarrow i + \delta$             ▷ Shift windows by $\delta$ if no drift detected
      end if
  end while
```

Unsupervised Drift Detection:

In this section we adapt our statistic to the unsupervised setting where the samples in the detection window are unlabelled i.e., only the covariates are available (Algorithm 2 in Appendix C). Since labels in the detection window are not available we cannot learn the classifier $\widehat{q}$, and therefore Algorithm 1 will not work directly in this setting. In order to tackle this, we create a new "uncertainity" based approximation $\widehat{q}$ and use it in our statistic. Let $\mathcal{S}_R$ be the reference window and $\mathcal{S}_D$ be the detection windows. We define mean entropy of $p(Y|\boldsymbol{X})$[3] on reference and detection windows as

[3]conditional distribution of samples in reference window

follows:

$$\overline{H}(p, \mathcal{S}_R) = -\frac{1}{|\mathcal{S}_R|} \sum_{(\boldsymbol{x}_t, y_t) \in \mathcal{S}_R} \sum_{y \in \mathcal{Y}} p(y|\boldsymbol{x}_t) \log p(y|\boldsymbol{x}_t)$$

$$\overline{H}(p, \mathcal{S}_D) = -\frac{1}{|\mathcal{S}_D|} \sum_{\boldsymbol{x}_t \in \mathcal{S}_D} \sum_{y \in \mathcal{Y}} p(y|\boldsymbol{x}_t) \log p(y|\boldsymbol{x}_t)$$

Using the mean entropy defined above, we define an uncertainty measure $\tau(p, \mathcal{S}_R, \mathcal{S}_D)$ as follows:

$$\tau(p, \mathcal{S}_R, \mathcal{S}_D) = \exp(\overline{H}(p, \mathcal{S}_R) - \overline{H}(p, \mathcal{S}_D))$$

Note that $\tau(p, \mathcal{S}_R, \mathcal{S}_D)$ is equal to 1 when $p(Y|\boldsymbol{X}) = q(Y|\boldsymbol{X})$[4]. As p becomes more uncertain on samples in $\mathcal{S}_D$ compared to $\mathcal{S}_R$, $\tau(p, \mathcal{S}_R, \mathcal{S}_D)$ increases. Since we have access to labelled samples $\mathcal{S}_R$ with distribution $p(Y|\boldsymbol{X})$, we learn a classifier $\widehat{p}$ that estimates $p(Y|\boldsymbol{X})$ and compute the plug in estimator $\tau(\widehat{p}, \mathcal{S}_R, \mathcal{S}_D)$. In order to create an estimate of $q(Y|X)$, we further assume that the classifier $\widehat{p}$ is obtained post application of softmax on a vector of functions $f(\boldsymbol{x}) = (f_y(\boldsymbol{x}))_{y \in \mathcal{Y}}$ learned using $\mathcal{S}_R$. We define an approximation $\widehat{q}(Y|\boldsymbol{X})$ of $q(Y|\boldsymbol{X})$ as follows:

$$\widehat{q}(y|\boldsymbol{x}) = \frac{\exp\left(\frac{f_y(\boldsymbol{x})}{\tau}\right)}{\sum\limits_{y' \in \mathcal{Y}} \exp\left(\frac{f_{y'}(\boldsymbol{x})}{\tau}\right)} \tag{2}$$

Apart from the above definition of $\widehat{q}$, the algorithm in this setting is similar to the one in Algorithm 1. We expand on this further in Appendix C.

3 Experiments and Results

We comprehensively study our contributions in this section on eight different datasets widely used in the drift detection community Lu et al. (2018); Pesaranghader et al. (2018). Our datasets are divided into three categories. The first type is purely synthetic – Hyperplane Bifet et al. (2010) with gradual drifts, Sine & Mixed Gama et al. (2004) with abrupt drifts. The second type of datasets contain real-world data with synthetically induced drifts – USENET1, USENET2 Katakis et al. (2008); therefore, we know the location of drifts for these datasets. The third type of datasets are real-world data streams – Electricity, Poker & Cover Type from the popular MOA repository Details of each dataset and the drift therein are included in Appendix E. For qualitative evaluation of interpretability, we use USENET2 dataset, where ground truth for interpretation of concept drift can be gleaned directly from the dataset design. Below, we describe datasets, evaluation metrics, & implementation details before presenting our results.

Evaluation Metrics Real-world streaming datasets do not have information on the true drift location. Thus, the average classification accuracy of the underlying classifier (classifier trained on sampling distribution of reference window) is a commonly used metric Tahmasbi et al. (2021) for assessing drift detection performance. Here, accuracy refers to the underlying classifier's accuracy w.r.t. the set of class labels (which does not change) of the data stream. This accuracy is computed on the detection window at every step and averaged across the data stream. We qualitatively evaluate interpretability on USENET2 dataset. On datasets with true drift location information (USENET1, USENET2, Sine and Mixed), we study the correctness of detected drifts in Section 4. For synthetic datasets with abrupt drifts, average model accuracy is not indicative of true performance since a higher average accuracy can be achieved with higher false positives. For such datasets, we study the precision and recall of detected drifts.

Baselines We evaluate our drift detection method against 5 well-known baseline methods: two state-of-the-art interpretable (virtual) drift detectors: A-KS dos Reis et al. (2016), CHT Kulinski et al. (2020)); and three popular supervised black-box drift detectors: DDM Gama et al. (2004), EDDM Baena-Garcıa et al. (2006), and the state-of-the-art method MDDM Pesaranghader et al. (2018). Since we are the first interpretable real drift detection method, our objective is to perform at par with state-of-the-art methods (which are black-box) on the above mentioned detection metrics, while providing interpretability on the outcomes. For all baseline methods, we use default parameters specified in respective papers.

Implementation Details *Sliding Window Procedure:* Drift detection methods, especially interpretable ones (including ours), require two windows to operate: *reference window* (containing samples from training/past distribution) and *detection window* (containing current samples which may or may not be from the same distribution as the reference window). Following the baselines, we use

[4]conditional distribution of samples in detection window

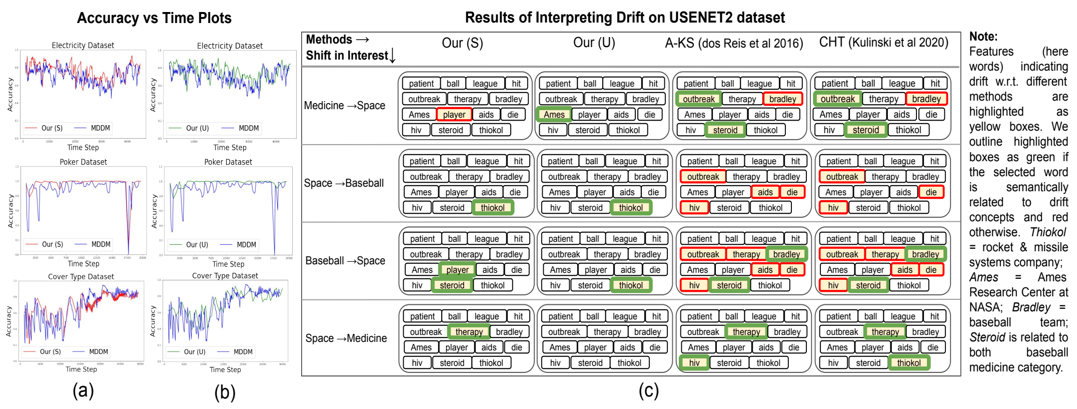

Figure 2: (a) & (b) Comparison of classifier accuracy over time of our supervised (Our (S)) & unsupervised version (Our (U)) method w.r.t. MDDM (c) Interpretability result: A-KS & CHT often select words from irrelevant concepts, whereas our method selects words relevant to shift.

the same length n for both these windows, and consider them to be adjacent in time. As the data stream progresses, both reference and detection windows are shifted forward by δ (we use $\delta = 50$ for window sizes $\geq$ 500 and 10 otherwise). δ is generally much smaller than n to avoid missing drifts and thereby localization. If there is a drift, both windows are shifted by the window size n, as the old detection window is now the new reference window. In the unsupervised setting, we cannot shift the reference window at every time step as we require the labels of the reference window, which may not be available for a test data window. Hence, when there is no drift in this setting, only the detection window is shifted forward by δ. When there is a drift, the reference window is updated to the recently drift-detected window with corresponding true labels. For our main results reported in Table 1, the window size for all data streams (except USENET1,USENET2) is set to 500 (much smaller than data stream lengths). For USENET 1 & 2, window size is set to 150 as drifts occur after every 300 samples. We further study the impact of window size on drift detection in Appendix F.

Modeling the Posterior Distribution: Since our drift detection framework relies on classifier gradients, we use neural networks (NNs) to model the posterior distributions (other classifiers like logistic regression can also be used; we use NNs considering their wide use). For all the main results (Tables 1 & 2), we use a 2-layer NN with a softmax activation function at the output layer, which is interpreted as probabilities for class labels. Batch sizes are generally small in a streaming setting, due to the nature of the setting itself; a small NN is hence more well-suited. Appendix E contains more details on the architecture and hyperparameters used to train the NN. We also study the effect of different NN architectures on drift detection in Appendix F.

Hypothesis Testing Protocol: Give a small batch of samples from a new window of a data stream, this batch is required to train the classifier, as well as to estimate the statistic to detect the drift. We hence divide this batch into two disjoint parts – one to train the classifier, and the other to estimate the statistic. As stated in Section 2.2, the ratio r denotes the fraction of this batch used for training the classifier. For all our experiments herein, we set $r = 0.8$ i.e. 80% of the reference window is used to train the model and the rest to compute the statistic. We study the impact of this choice of r in Appendix F. To obtain the threshold to reject the null hypothesis, we use a sampling-based bootstrap test (similar to Kulinski et al. (2020)). α is set to 0.05, as typically done in literature. We also use Bonferroni correction to correct for multiple comparisons. For more details refer to App D.

Results Table 1 reports our results on drift detection via the average model accuracy metric computed on the data stream. Figure 2 (a) shows model acc across the data stream on real-world datasets. From these, we can observe Our (supervised and unsupervised) outperforms the interpretable baselines consistently and is on par with existing black-box methods. Please note that interpretable baselines A-KS and CHT latch on to benign drifts in real-world datasets, thus always signaling drift (a similar observation was made by Kulinski et al. (2020)). However, our method focuses on the posterior and thus is robust to such benign shifts and hence more useful in real-world.

Method → **Dataset ↓**	Ours(S)	Ours(U)	A-KS	CHT	MDDM	DDM	EDDM
Hyperplane	**88.0**	**88.0**	<u>87.6</u>	**88.0**	86.7	85.9	85.7
USENET1	**57.2**	52.2	<u>55.0</u>	51.0	58.4	56.5	57.2
USENET2	**68.6**	<u>68.5</u>	67.5	68.4	67.1	67.1	67.1
Cover type	**75.6**	<u>71.0</u>	*AD*	*AD*	68.1	50.1	55.1
Electricity	**77.9**	<u>75.6</u>	*AD*	*AD*	74.6	73.4	73.4
Poker	<u>96.1</u>	**97.0**	*AD*	*AD*	90.1	80.0	78.5

Table 1: Comparison with interpretable (in gray) & black-box baselines w.r.t. average model accuracy across the stream. (S - supervised, U - unsupervised, AD - always detect drift)

Interpretability Results Localizing which features are related to the drift is an important aspect of our work. To evaluate this aspect, we use the USENET2 dataset, which was constructed in Katakis et al. (2008) by asking a labeler to read email messages and categorize them as *interesting* or *not interesting* (output labels). The dataset's input features correspond to words in these messages. It contains 5 time segments of 300 examples each, and the emails belong to *medicine*, *baseball* or *space* categories. At the end of each segment, there is change in the labeler's interest, indicating a concept drift. Figure 2(c) shows features that were identified to be responsible for drift according to different methods at different drift locations in the dataset. As can be seen, our method (supervised and unsupervised) is able to assign importance to features that reflect the change in interest from one category to another in three out of four drifts. Our unsupervised method (Our (U)) does not detect the final drift and thus no highlighted boxes. (We used a 4-layer NN for this study across all the methods to better model the task.)

Note that in the second drift i.e. when user interest changes from space to baseball, our method selects the feature "*thiokol*" (company related to rocket and missile systems) which corresponds to space category. A-KS and CHT methods instead pick words such as "*hiv*","*aids*", which belong to medicine category. We believe that this happens as A-KS and CHT focus only on covariate distribution, and are hence not sensitive to change in relationship between input features and output label. Our method's ability to use posterior distribution helps it detect and understand this change.

4 Discussion and Analysis

Here we analyze our method using more studies (in both supervised & unsupervised settings).

Correctness of Detected Drifts. For datasets that have the true drift location available, we use precision and recall of detected drifts to study correctness. We define true positive (TP), false positive (FP) and false negative (FN) in our context as follows: TP is a detection that is made within a small fixed time range of the true drift location; FN refers to missing a detection within the fixed time range; FP is a detection outside the fixed time range (around the true drift location) or an extra detection in the time range of true drift location. Table 2 reports precision and recall for time range equal to the length of detection window (for all datasets and baselines). It shows that our method (supervised and unsupervised versions) achieves best precision values across the datasets at high recall values. Our unsupervised version attains a lower recall than the supervised version across all datasets. We believe this is due to the error in approximation of the posterior distribution $\widehat{q}$ at test time and the noise in pseudolabels used in the bootstrap test.

Method → **Dataset** ↓	Ours(S) (P/R)	Ours(U) (P/R)	A-KS (P/R)	CHT (P/R)
Sine	**1.0**/**1.0**	**1.0**/0.3	0/0	0/0
Mixed	**1.0**/**1.0**	0.6/**1.0**	0.5/**1.0**	0.5/**1.0**
USENET1	**0.7**/**1.0**	**0.7**/**1.0**	**0.7/1.0**	**0.7**/**1.0**
USENET2	0.7/**1.0**	**0.8**/0.8	0.7/0.7	0.6/**1.0**

Table 2: Precision(P) and Recall(R) of detected drifts for all interpretable methods on datasets that have true drift loc. (S - supervised, U - unsup.)

Other Studies. We include other studies, including the impact of window size and model complexity, in Appendix F. As we operate in the streaming data regime, we analyze our method's time complexity and compare it with existing interpretable baselines in Appendix F. Our method incurs a time complexity of $O(nK(C + d))$ (given we have access to the classifiers) i.e. linear in n and d, and thus scales to high-dim datasets (C - complexity of model that is used to approximate the posterior distribution).

5 Conclusions

We presented an interpretable method to detect and understand real concept drifts. We used classifier gradients to understand the contribution of different features toward the difference in posterior distributions at train and test times. We proposed two versions of our method for supervised and unsupervised settings, making our solution flexible and practically relevant. We conducted experiments to analyze our method at four levels: (i) Precision-Recall of detected drifts on datasets with known drift locations, (ii) Average model classification accuracy on data streams, (iii) Plots of the accuracy of classifier across time for real-world data streams, and (iv) Qualitative results on USENET2 to study interpretability aspect. From these wide ranges of experiments, we infer that our drift detector is accurate and interpretable at the same time.

Acknowledgments

We would like to thank Kancheti Sai Srinivas (Indian Institute of Technology, Hyderabad) for his useful feedback and ideas.

REFERENCES

Manuel Baena-Garcıa, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, R Gavalda, and Rafael Morales-Bueno. Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams*, volume 6, pp. 77–86, 2006.

Albert Bifet and Ricard Gavalda. Learning from time-changing data with adaptive windowing. In *International Conference on Data Mining*, pp. 443–448. SIAM, 2007.

Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, and Thomas Seidl. Moa: Massive online analysis, a framework for stream classification and clustering. In *First Workshop on Applications of Pattern Analysis*, pp. 44–50. PMLR, 2010.

Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3):131–151, 1999.

Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. Credit card fraud detection and concept-drift adaptation with delayed supervised information. In *International Joint Conference on Neural networks (IJCNN)*, pp. 1–8. IEEE, 2015.

Denis Moreira dos Reis, Peter Flach, Stan Matwin, and Gustavo Batista. Fast unsupervised online drift detection using incremental kolmogorov-smirnov test. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1545–1554, 2016.

Dheeru Dua, Casey Graff, et al. Uci machine learning repository. 2017.

Joao Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. Learning with drift detection. In *Brazilian symposium on artificial intelligence*, pp. 286–295. Springer, 2004.

João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):1–37, 2014.

Michael Harries and New South Wales. Splice-2 comparative evaluation: Electricity pricing. 1999.

Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. Transcend: Detecting concept drift in malware classification models. In *USENIX Security Symposium (USENIX Security 17)*, pp. 625–642, 2017.

Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. An ensemble of classifiers for coping with recurring contexts in data streams. In *ECAI 2008*, pp. 763–764. IOS Press, 2008.

Imen Khamassi, Moamar Sayed-Mouchaweh, Moez Hammami, and Khaled Ghédira. Discussion and review on evolving data streams and concept drift adapting. *Evolving systems*, 9(1):1–23, 2018.

Sean Kulinski, Saurabh Bagchi, and David I Inouye. Feature shift detection: Localizing which features have shifted via conditional distribution tests. In *Neural Information Processing Systems*, 2020.

Himabindu Lakkaraju, Nino Arsov, and Osbert Bastani. Robust and stable black box explanations. In *International Conference on Machine Learning*, pp. 5628–5638. PMLR, 2020.

Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2018.

Ali Pesaranghader, Herna L Viktor, and Eric Paquet. Mcdiarmid drift detection methods for evolving data streams. In *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–9. IEEE, 2018.

Kevin B Pratt and Gleb Tschapek. Visualizing concept drift. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 735–740, 2003.

Addisson Salazar, Gonzalo Safont, Antonio Soriano, and Luis Vergara. Automatic credit card fraud detection based on non-linear signal processing. In *International Carnahan Conference on Security Technology (ICCST)*, pp. 207–212. IEEE, 2012.

Tegjyot Singh Sethi and Mehmed Kantardzic. On the reliable detection of concept drift from streaming unlabeled data. *Expert Systems with Applications*, 82:77–99, 2017.

Ashraf Tahmasbi, Ellango Jothimurugesan, Srikanta Tirthapura, and Phillip B Gibbons. Driftsurf: Stable-state/reactive-state learning under concept drift. In *International Conference on Machine Learning*, pp. 10054–10064. PMLR, 2021.

Houari Toubakh and Moamar Sayed-Mouchaweh. Hybrid dynamic data-driven approach for drift-like fault detection in wind turbines. *Evolving Systems*, 6(2):115–129, 2015.

Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(2):58, 2004.

Geoffrey I Webb, Loong Kuan Lee, Bart Goethals, and François Petitjean. Analyzing concept drift and shift from sample data. *Data Mining and Knowledge Discovery*, 32(5):1179–1199, 2018.

Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. {CADE}: Detecting and explaining concept drift samples for security applications. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2327–2344, 2021.

Jan Zenisek, Florian Holzinger, and Michael Affenzeller. Machine learning based concept drift detection for predictive maintenance. *Computers & Industrial Engineering*, 137:106031, 2019.

Indre Žliobaite. Change with delayed labeling: When is it detectable? In *International Conference on Data Mining Workshops*, pp. 843–850. IEEE, 2010.

Appendix

In this appendix, we provide the following additional information that could not be included in the main paper owing to space constraints:

- Related Work
- Proof of proposition
- Unsupervised drift detection
- Bootstrap testing details
- Dataset and hyperparameter details
- Ablation studies and time complexity analysis

A Related Work

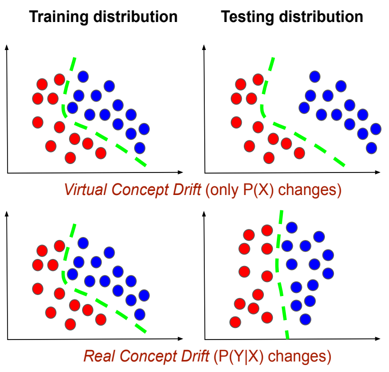

Figure 3: **Different types of drifts:** Real and virtual concept drift. Real concept drift focuses on optimal decision boundary (shown in green) whereas virtual drift tracks change in covariate distribution (illustration inspired by Fig 1 in (A Gama et al., 2014))

Concept Drift Detection. Concept drift in a machine learning context refers to change in joint likelihood i.e. $p(\mathbf{X}, Y) \neq q(\mathbf{X}, Y)$ Gama et al. (2014), where $\mathbf{X}$ corresponds to data, Y corresponds to labels, and p, q refer to train and test distributions respectively. Concept drift detection has been studied for over two decades now Tsymbal (2004), and the general framework has remained as follows Lu et al. (2018): (i) Retrieve temporal chunks of data from the data stream. The chunk of data from the old distribution is often referred to as *reference window*, and a new chunk of data is called as *detection window* (length of these windows can be fixed or adaptive); (ii) Extract a *key statistic* from both windows; (iii) Calculate a metric between reference and detection windows using the *key statistic*; and (iv) Use a statistical test to check if the distance is significant for declaring a drift.
Based on the statistic used for drift detection, existing methods can be categorized into *virtual concept drift* (change in data covariates without a change in the decision boundary) or *real concept drift* methods (change in posterior distribution). Please refer to Figure 3 for a visual description of the differences between the two types of drifts. Virtual concept drift methods include A-KS dos Reis et al. (2016), CHT Kulinski et al. (2020), and MD3 Sethi & Kantardzic (2017); while real concept drift methods include MDDM Pesaranghader et al. (2018), DDM Gama et al. (2004), EDDM Baena-Garcıa et al. (2006), ADWIN Bifet & Gavalda (2007) . These methods can also be categorized as *black-box drift detectors* (only drift localization) or *interpretable drift detectors* (drift localization with additional insights about drift). Methods such as MDDM, DDM, FW-DDM, HDDM, ECDD and EDDM fall in the category of black-box methods as they are not interpretable, while methods such as A-KS and CHT along with detecting drift also provide some additional insights about the drift. As stated earlier, our method is an interpretable drift detector, i.e. we aim to maintain an accurate classifier across drifts and also provide feature-level insights on the detected drift. To the best of our knowledge, this is the first interpretable real concept drift detection method, as detailed below.

Explaining Concept Drift. Beyond standard drift detection, there have been very few efforts in literature that aim to provide insights about detected drifts. These efforts can be viewed as visualization-based methods or feature-interpretable methods. *Visualization-based methods* use plots and maps to inform a user about the distribution shift. For e.g., Webb et al. (2018) studied drift using quantitative descriptions of drift in the marginal distributions, and then used marginal drift magnitudes between time periods to plot heat maps that gives insights of the drift. Pratt & Tschapek (2003) developed a visualization tool that used parallel histograms to study concept drift, that a user could visually inspect. On the other hand, *feature-interpretable methods* attempt to detect drift and simultaneously notify which features might be responsible for it. CADE Yang et al. (2021) addresses a special kind of concept drift detection and understanding problem where drifting samples come from a class that is not seen earlier by the classifier. In this work, we consider a more common scenario where classes encountered in the system remain the same over the drift. A-KS dos Reis et al. (2016) performed an attribute-wise KS test, while CHT Kulinski et al. (2020) performed a hypothesis test to check for change in distribution of each feature conditioned on rest of input features. Each of these methods, focuses on virtual concept drift, while ours focuses on real concept drift detection.

B PROOF OF PROPOSITION

Proposition 2.1 (Convergence of Power of our Test Statistic)**.** *If the alternate hypothesis $\boldsymbol{H}_a$ is true for some feature $k \in [d]$, then, for any $t > 0$,*

$$\lim_{n\to\infty} \mathbb{P}[c_n^k(p,q) > t] = 1$$

Proof. From the definition of the statistic in Equation 1 it's easy to see that the expected value of our statistic is:

$$\mu_n^k = \mathbb{E}_{\mathcal{S}}[c_n^k(p,q)] = n\mathbb{E}_{(\boldsymbol{x},y)\sim p(\boldsymbol{X},Y)}\left[\left(\partial_k \log \frac{p(y|\boldsymbol{x})}{q(y|\boldsymbol{x})}\right)^2\right]$$

Let $\lambda(.)$ denote the lebesgue measure on $\mathbb{R}^d$. Since $\mathbf{H}_a$ is true for feature k, we know that there exists a set $A \subset \mathcal{X}$ ($\lambda(A) > 0$) and $y \in \mathcal{Y}$, such that $\partial_k \log(p(y|\boldsymbol{x})) \neq \partial_k \log(q(y|\boldsymbol{x}))\ \ \forall \boldsymbol{x} \in A$. By sub-additivity, $\lambda(A^c \cup B_y) \leq \lambda(A^c) + \lambda(B_y) < 1$, implying that $\lambda(A \cap B_y^c) > 0$.

Recall, the expected value of our statistic is $\mu_n^k = n\theta^k$ where:

$$\theta^k = \sum_{y'\in\mathcal{Y}} \int_{\boldsymbol{x}\in\mathcal{X}} \left(\partial_k \log \frac{p(y'|\boldsymbol{x})}{q(y'|\boldsymbol{x})}\right)^2 p(\boldsymbol{x}, y')\, d\boldsymbol{x}$$

Clearly $\theta^k \geq 0$. Also, the above expression contains the term $\int_{\boldsymbol{x}\in\mathcal{X}} \left(\partial_k \log \frac{p(y|\boldsymbol{x})}{q(y|\boldsymbol{x})}\right)^2 p(\boldsymbol{x}, y)\, d\boldsymbol{x}$, which is further greater than or equal to

$$\int_{\boldsymbol{x}\in A\cap B_y^c} \left(\partial_k \log \frac{p(y|\boldsymbol{x})}{q(y|\boldsymbol{x})}\right)^2 p(\boldsymbol{x}, y)\, d\boldsymbol{x}$$

Since, $\left(\partial_k \log \frac{p(y|\boldsymbol{x})}{q(y|\boldsymbol{x})}\right)^2$ is strictly positive on A, it is also positive on $A \cap B_y^c$. Probability $p(\boldsymbol{x}|y)$ is strictly positive on B_y^c, therefore it is strictly positive on $A \cap B_y^c$. Also, by assumption $p(y)$ is strictly positive. Therefore, the integrand above is strictly positive and being integrated on a set of positive lebesgue measure implying that $\theta^k > 0$. Thus, for any $t > 0$, there exists integer N such that $\mu_n^k - t = n\theta^k - t > 0$ for all $n \geq N$. Now, for $n \geq N$, we use Hoeffding inequality to obtain:

$$\mathbb{P}[c_n^k > t] \geq \mathbb{P}[|c_n^k - \mu_n^k| < \mu_n^k - t] \geq 1 - exp(-\frac{2(\mu_n^k - t)^2}{n(M-m)^2})$$

where $m, M > 0$ are such that

$$m \leq \left(\partial_k \log p(y_i|\boldsymbol{x}_i) - \partial_k \log q(y_i|\boldsymbol{x}_i)\right)^2 \leq M$$

Substituting $\mu_n^k = n\theta^k$, we get:

$$\mathbb{P}[c_n^k > t] \geq 1 - exp(-\frac{2n(\theta^k - \frac{t}{n})^2}{(M-m)^2}) \to 1 \text{ as } n \to \infty.$$

□

Algorithm 2 Unsupervised Interpretable Drift Detection

```
Require: $n, \alpha, K, \delta, \mathcal{S}_R = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}, \mathcal{S} = \{\boldsymbol{x}_{n+1}, \boldsymbol{x}_{n+2}, \ldots\}$
  i $\leftarrow n$, and classifier $\widehat{p} \leftarrow$ GETCLASSIFIER($\mathcal{S}_R$)
  Create empty list Explanation $\leftarrow \phi$.
  while $True$ do
      Flag $\leftarrow False$
      $\mathcal{S}_D = \{\boldsymbol{x}_t : t \in [i+1, i+n]\}$
      Use definition of $\widehat{q}$ in Equation 3.2 and compute $\tilde{c}_n^k(\widehat{p}, \widehat{q})$.
      Get thresholds $(T_\alpha^1, \ldots, T_\alpha^d) \leftarrow$ BOOTSTRAP2($\widehat{p}, \alpha, K, \mathcal{S}_R, \mathcal{S}_D$)
      if for any $k \in [d]$, $c_n^k(\widehat{p}, \widehat{q}) > T_\alpha^k$ then
          Flag $\leftarrow True$, Add all such $k$s to Explanation.
      end if
      if Flag $= True$ then                                        ▷ Drift detected
          $i \leftarrow i + n$                                      ▷ Shift windows by $n$
          Get labels $y_t$ for $\boldsymbol{x}_t$, $t \in [i+1, i+n]$.
          Update $\mathcal{S}_R \leftarrow \{(\boldsymbol{x}_t, y_t) : t \in [i+1, i+n]\}$.
          $\widehat{p} \leftarrow$ GETCLASSIFIER($\mathcal{S}_R$).                         ▷ relearn $\widehat{p}$
          Print the list Explanation and reset it to $\phi$.
      else
          $i \leftarrow i + \delta$                    ▷ Shift windows by $\delta$ if no drift detected
      end if
  end while
```

C Unsupervised Drift Detection

Continuing our discussion in Section 2.2, here we present our algorithm for the unsupervised setting where the samples in the detection window are unlabelled i.e., only the covariates are available (Algorithm 2). One minor difference, that was not discussed in the main paper, between supervised and unsupervised algorithm is that whenever we update the reference window $\mathcal{S}_R$, we need to obtain labels for all samples in the new window. Another difference is in the bootstrapping technique where we use pseudo labels (labels predicted by classifier $\widehat{q}$) instead of true labels. Since we don't have access to labels in $\mathcal{S}_D$, we use pseudo labels instead i.e. the labels predicted by classifier $\widehat{q}$. This is also shown as a separate Algorithm 4 for the unsupervised version in the next section (Appendix D).

D Bootstrap Testing Details

In Sections 2, we discussed the algorithm of our method (both supervised and unsupervised versions) where we used the bootstrap test to calculate thresholds for checking which features rejected the null hypothesis. As stated in Section 3 (under '*Hypothesis Testing Protocol*'), we follow Kulinski et al. (2020) for this test, which we describe herein. Algorithm 3 (BOOTSTRAPTEST1) is used for the supervised setting bootstrap test, and Algorithm 4 (BOOTSTRAPTEST2) for the unsupervised setting. To obtain the threshold to reject the null hypothesis, we perform bootstrap sampling $K = 50$ number of times. Specifically, we merge and shuffle samples from p and q distributions (as defined in Section 3), and then pick K two-samples from this mixture. This simulates the null hypothesis. Now, we calculate the test statistic for these K two-samples and return the (1-α)-th quantile of the test statistic values as our threshold. We use $\alpha = 0.05$ as the significance level of our test as is standard in hypothesis testing literature. Similar to Kulinski et al. (2020), while computing the test statistic during the bootstrap test, we refit our models for each of the K two-samples. Specifically, we train models on the original given reference and detection windows, and update these models (1-2 epochs of finetuning) for each bootstrap episode.

E Dataset and Hyperparameter Details

This section describes all datasets used in our experiments and hyperparameters used in our experiments.

E.1 Datasets.

Table 3 provides a summary of relevant statistics about the datasets used. *Window Size* in Table 3 refers to detection and reference window size used for different datasets.

Algorithm 3 BOOTSTRAPTEST1

```
Require: $\widehat{p}, \widehat{q}, \alpha, K, \mathcal{S}_R, \mathcal{S}_D$
  $n, d \leftarrow$ DIM($S_R$)                                  ▷ $n$: num of samples, $d$ : num of feats
  ScoreMatrix $\leftarrow$ ZEROS($shape = (K, d)$)
  $M \leftarrow$ CONCATENATE($S_R, S_D$)
  for $i$ in $[1, 2, \dots, K]$ do                              ▷ calc. for $K$ bootstrap samples
      $\widetilde{\mathcal{S}}_R \leftarrow$ SAMPLE($M$,$n$)                   ▷ sample $n$ pts with replacement
      $\widetilde{\mathcal{S}}_D \leftarrow$ SAMPLE($M$,$n$)                   ▷ sample $n$ pts with replacement
      FINETUNE($\widehat{p}, \widetilde{\mathcal{S}}_R$)            ▷ finetune for 2 epochs (A Kulinski et al., 2020)
      FINETUNE($\widehat{q}, \widetilde{\mathcal{S}}_D$)            ▷ finetune for 2 epochs (A Kulinski et al., 2020)
      Compute $c_n^k(\widehat{p}, \widehat{q})$, $\forall k \in [d]$, using $\widetilde{\mathcal{S}}_R, \widehat{p}, \widehat{q}$ in Equation 3.1.
      [ScoreMatrix]$_i = [c_n^1, c_n^2, \dots, c_n^d]$
  end for
  return $(1 - \frac{\alpha}{d})$ quantile of $c_n^k$ values stored in ScoreMatrix $\forall k$
```

Algorithm 4 BOOTSTRAPTEST2

```
Require: $\widehat{p}, \alpha, K, \mathcal{S}_R, \mathcal{S}_D$
  $n, d \leftarrow$ DIM($S_R$)                                  ▷ $n$: num of samples, $d$ : num of feats
  ScoreMatrix $\leftarrow$ ZEROS($shape = (K, d)$)
  $\widehat{y}_D \leftarrow$ ARGMAX($\widehat{p}(S_D)$)                     ▷ Pseudo labels for detection window
  $\widehat{S}_D \leftarrow (\widehat{y}_D, S_D)$                            ▷ combine label and covariates
  $M \leftarrow$ CONCATENATE($S_R, \widehat{S}_D$)
  for $i$ in $[1, 2, \dots, K]$ do                              ▷ calc. for $K$ bootstrap samples
      $\widetilde{S}_R \leftarrow$ SAMPLE($M$,$n$)                   ▷ sample $n$ pts with replacement
      $\widetilde{S}_D \leftarrow$ SAMPLE($M$,$n$)                   ▷ sample $n$ pts with replacement
      Use definition of $\widehat{q}$ in Equation 3.2 and compute $\tilde{c}_n^k(\widehat{p}, \widehat{q})$ $\forall k \in [d]$ using samples $\widetilde{S_R}, \widetilde{S_D}$.
      [ScoreMatrix]$_i = [\tilde{c}_n^1, \tilde{c}_n^2, \dots, \tilde{c}_n^d]$
  end for
  return $(1 - \frac{\alpha}{d})$ quantile of $\tilde{c}_n^k$ values stored in ScoreMatrix $\forall k$
```

Hyperplane (A Bifet et al., 2010): A hyperplane is used here to simulate time-varying concepts. As the position and orientation of the hyperplane is changed smoothly, the old classifier (trained to separate datapoints lying on different halfspaces of the hyperplane) gets outdated. The magnitude of change is 0.001 to simulate a gradual drift. The drift locations are not known in this dataset.

Mixed (A Gama et al., 2004): This dataset has four input features (x_1, x_2, x_3, x_4), where x_1 and x_2 are Boolean and x_3 , x_4 are uniformly distributed in $[0, 1]$. Label of each data is determined to be positive if two of x_1, x_2, and $x_4 < 0.5 + 0.3\sin(3\pi x_3)$ conditions hold. A drift is caused when the distribution of x_3 and x_4 are perturbed i.e. changed from a uniform to a non-uniform distribution. We use a slightly larger window size for this dataset as drift locations are known to be far apart (greater than the window sizes used).

Sine (A Gama et al., 2004): This dataset contains two attributes (x_1, x_2), uniformly distributed in $[0, 1]$. The output label is determined by using the curve $x_2 = \sin(x_1)$, and data points lying below it are classified as positive. The first drift is created by reversing the labels. For the next drift, the curve (or labeling function) is changed. The subsequent drift is again simulated by reversing the labels. We use a slightly larger window size for this dataset as drift locations are known to be far apart (greater than the window sizes used).

USENET1 and USENET2 (A Katakis et al., 2008): Both these datasets are created by labeling a stream of email messages as interesting or not interesting depending on the labeler's interest. They contain data for 5 time periods of 300 samples each. A drift is present after each time period because of change in user (or labeler) interest. Since these datasets contain words as input features, we follow standard language processing pre-processing steps and remove stop words such as pronouns, dates, numbers and articles from this dataset. Examples of such words from the USENET2 dataset are: *'taken', 'eng', 'wrench', '1993apr20', '22', 'young', 'bob', 'mda'*.

Stats → Dataset ↓	Dimensions	Type	Size	Window Size	Classes
Hyperplane	10	Synthetic	20000	500	2
Mixed	3	Synthetic	20000	1500	2
Sine	3	Synthetic	20000	1500	2
USENET1	42	Semi-Real	1500	150	2
USENET2	25	Semi-Real	1500	150	2
Electricity	5	Real	45312	500	2
Poker	10	Real	20000	500	10
Cover Type	54	Real	30000	500	7

Table 3: Details of datasets used for experiments

Electricity (A Harries & Wales, 1999): This dataset contains information about the New South Wales Electricity Market in Australia. Input features correspond to price, demand and amount of energy transferred between two states in Australia. The class label indicates whether the transfer price (cost to transfer electricity from one state to another) is increased or decreased relative to a moving average of the last 24 hours. Drift is expected due to changes in consumption habits, change in electricity board and other unexpected deviations.

Poker (A Dua et al., 2017): Each data point in this dataset represents a hand of 5 cards drawn from a deck of 52 cards. Each card is described by two features (suit and rank). The classification task is to predict the poker hand i.e. one out of the ten classes. An adjustment of the poker hand (change of order) constitutes a concept drift. We use the first 30000 instances of this dataset for experimentation.

Cover Type (A Blackard & Dean, 1999): This dataset contains 54 attributes representing 7 forest cover types (i.e. 7 classes) for 30×30 meter cells obtained from US Forest Service (USFS) information system, for 4 wilderness areas located in the Roosevelt National Forest of Northern Colorado. Concept drift appears due to change in weather conditions. We use the first 20000 instances of this dataset for experimentation.

E.2 Hyperparameters.

We herein describe various hyperparameters used in our experiments to facilitate reproducibility. As stated earlier, we used neural networks as the model class of choice across all our experiments. In particular, we used 3 neural network architectures across our experiments (inclusive of ablation studies in Appendix F):

1. 2-layer Neural Network: First layer has 1024 neurons and second has 512 neurons. This was used in all our experiments reported in Section 4.
2. 4-layer Neural Network: We used 1024-512-256-128 neuron layers from first to fourth layer respectively. This was used in all our ablation studies reported in Section F.
3. 6-layer Neural Network: We used 1024-512-256-128-128-64 neuronlayers from first to sixth layer respectively. This was used in all our ablation studies reported in Section F.

We used Dropout of 10% after every layer in each neural network. An $L2$ weight decay regularizer with co-efficient 0.001 in the loss term was used consistently across all models during training. ReLU activation function is used in all layers across all our networks. We use a learning rate of 1e-04, batch size of 16 and Adam Optimizer for all our experiments.

For both supervised and unsupervised versions of our method (Algorithms 1 and 2), we use number of bootstraps $K = 100$ (following Kulinski et al. (2020)), $\delta = 10$ for window sizes less than 500 and $\delta = 50$ for window sizes ≥ 500.

F Ablation Studies and Time Complexity Analysis

Continuing from our discussion in Section 5, we report the results of our ablation studies where we analyze the impact of different hyperparameters used in our drift detection algorithm (Algorithm 1, 2). We also analyze the time complexities of different interpretable baselines and report the accuracy across time plots for the unsupervised version of our method.

F.1 Impact of r

- The ratio that defines the proportion of samples used for training the model to computing the test statistic in the reference window is referred to as r. In Sections 4, 5 all the results were reported with the r value being equal to 0.8. Here we vary the value of r and study its impact on the drift detection performance of our method as measured using the metric of average model classification accuracy.

r values → Dataset ↓	0.6	0.7	0.8
Hyperplane	86.6	87.7	88.0
Electricity	77.2	78.3	77.9

Table 4: Average model accuracy for different r values in supervised setting. Win size = 500; Model = 2 layer NN

r values → Dataset ↓	0.6	0.7	0.8
Hyperplane	88.5	87.8	88.0
Electricity	76.4	77.1	75.6

Table 5: Average model accuracy for different r values in unsupervised setting. Win size = 500; Model = 2 layer NN

It can be observed from Tables 4 & 5 that the final classifier accuracy stays approximately the same (1-2% for both the datasets) when r is varied indicating robustness of our method (both versions) to r value.

F.2 Impact of Window Size.

Results reported in Table 1 & 2 of the main paper used detection (and reference) window size of 500, as online settings often have small batch sizes. We now vary the window size in Tables 6 & 7 to study the impact of window size on drift detection performance of our supervised and unsupervised versions.

Window Size → Dataset ↓	500	750	1000
Hyperplane	88.0	88.9	87.9
Electricity	77.9	75.7	76.7

Table 6: Average model accuracy for different window sizes for our method in supervised setting

The results – across both supervised and unsupervised versions of our method – indicate that for different window sizes, average model accuracy stays approximately the same (within 0.5-1% for hyperplane dataset and within 2-3% for the electricity dataset). Thus our method is relatively robust to the window size. As a design principle, however, a large window size could increase the delay in detecting the drift, so one should choose the window size based on the application. These results also indicate that our method (both supervised and unsupervised versions) achieves good average model accuracy even for small window sizes, which can be beneficial in real-world online settings.

F.3 Impact of Model Complexity.

We conducted a study where the neural network model depth was varied while capturing the posterior distribution, to study the effect of model complexity on our method's performance. Reference and detection window sizes were set to 1500 in this study as we were using complex models which require more samples to train.

It can be observed from Table 8 & 9 that for both versions of our method the average model accuracy stays practically the same (within 0.5-1% for the hyperplane dataset and within 3-4% for the real-world electricity dataset). Given that these data streams are of significant length and are diverse (one

Window Size $\rightarrow$ Dataset $\downarrow$	500	750	1000
Hyperplane	88.0	87.6	87.4
Electricity	75.6	73.0	76.9

Table 7: Average model accuracy for different window sizes for our method in unsupervised setting

Num of Layers $\rightarrow$ Dataset $\downarrow$	2 layer	4 layer	6 layer
Hyperplane	88.0	88.6	88.7
Electricity	76.4	75.0	75.2

Table 8: Average model accuracy for different model complexities of the posterior network in supervised setting. Window size = 1500

Num of Layers $\rightarrow$ Dataset $\downarrow$	2 layer	4 layer	6 layer
Hyperplane	86.8	88.6	88.6
Electricity	77.8	73.4	74.3

Table 9: Average model accuracy for different model complexities of the posterior network in unsupervised setting. Window size = 1500

is synthetic and the other is real-world), it can be inferred that our method (both versions) is able to achieve good classifier performance across different neural network architectures.

F.4 Time Complexity Analysis.

For a given reference and detection window, we compare the time complexity of different interpretable methods from the lens of two key variables - sample size n, and input dimension d. Our method incurs the time complexity of $O(nK(C + d))$ (given we have we have access to the classifiers) where C is a variable that represents complexity of the model that estimates the posterior distribution and K is the number of bootstrap samples. Thus, time complexity of our method is linear in n and d, and thus scales to high-dimensional datasets.

CHT also has a cost of $O(nK(C + d))$ where C is a variable that represents complexity of the model that estimates the data distribution. A-KS has a cost of $O(dn \log n)$ for its non-incremental version. While A-KS has an incremental version which uses a randomized tree and gets better time complexity, this method was intended for virtual drift detection. Our method, on the other hand, performs real concept drift detection with about the same time complexity as these methods. An incremental version of our method that has a lower time complexity is an interesting future direction.

Real-world datasets such as the Electricity dataset (A Harries & Wales, 1999) have sampling rates in the order of hours. The supervised version of our method takes roughly 100 secs and the unsupervised version takes roughly 60 secs on the electricity dataset to perform our test on a window of size 500 (both reference and detection window) when we use a 2-layer NN to model the posterior distribution. With respect to the sampling rate of the dataset, our proposed drift detectors (supervised and unsupervised versions) are near real-time, thus making them practically useful for real-world applications.