# Task-Sensitive Concept Drift Detector with Constraint Embedding

## Andrea Castellani, Sebastian Schmitt, Barbara Hammer

## 2021

# Task-Sensitive Concept Drift Detector with Constraint Embedding

Andrea Castellani
*CITEC, Bielefeld University*
Bielefeld, Germany
acastellani@techfak.uni-bielefeld.de

Sebastian Schmitt
*Honda Research Institute Europe GmbH*
Offenbach, Germany
sebastian.schmitt@honda-ri.de

Barbara Hammer
*CITEC, Bielefeld University*
Bielefeld, Germany
bhammer@techfak.uni-bielefeld.de

***Abstract*—Detecting drifts in data is essential for machine learning applications, as changes in the statistics of processed data typically has a profound influence on the performance of trained models. Most of the available literature on drift detection employs either supervised methods, that requires true labels during inference time, or unsupervised, that aim for any changes in the data distribution. We propose a novel task-sensitive semi-supervised drift detection framework, which uses label information to train a model, but detects drifts during inference time only when they affect the model performance, without the ground truth label information. It utilizes a constrained low-dimensional embedding representation of the input data. This way, the dimensionality of the input data is reduced, and the learned representation is best suited for the classification task. We propose two change detectors, which are customized for our framework, but any method to detect a change in the statistic of a data stream can be chosen freely. Experimental evaluation on nine benchmarks datasets, with different types of artificially induced drift, demonstrates that the proposed framework can reliably detect drifts. Furthermore, in our studies we empirically demonstrated that the investigated drift detectors combined with the proposed framework, consistently outperform the other state-of-the-art unsupervised drift detection approaches.**

***Index Terms*—Concept Drift, Unsupervised, Deep Learning, Embedding Representation, Clustering**

## I. Introduction

In the context of data-driven machine learning, and data mining applications, the dataset used for calibrating or training the models plays a central role. Its statistical properties define the model's behavior and introduce implicit or explicit assumptions. In many situations, it is assumed that the distribution of the data streams is stationary, i.e. not changing over time, which is a valid assumption if the distribution of the data used to calibrate/train the model is the same as later during production use. However, due to various reasons, such as aging or slight system re-configurations, this assumption is not met in many real-world applications [1].

In literature, this phenomenon is usually referred to as *concept drift* [1] and it has received a lot of attention in recent years [2]–[4], because unexpected drifts can have a strongly negative impact on the model performance. Concept drift is defined as the change in the joint distribution of a set of input variables $\boldsymbol{x}$ and target variable $\boldsymbol{y}$ over time, i.e. $P_{t_0}(\boldsymbol{x}, \boldsymbol{y}) \neq P_{t_1}(\boldsymbol{x}, \boldsymbol{y})$ where $t_0$ and $t_1$ are usually *training* and *testing* time. We focus on classification tasks, where the output variables $\boldsymbol{y}$ represent class labels. Two fundamentally different categories exist for concept drift. *Virtual drift* [1] refers to changes in the distribution of the input data $\boldsymbol{x}$, without affecting the distribution of the classification labels, i.e. $P_{train}(\boldsymbol{x}) \neq P_{test}(\boldsymbol{x})$ but $P_{train}(\boldsymbol{y}|\boldsymbol{x}) = P_{test}(\boldsymbol{y}|\boldsymbol{x})$. *Real concept drift* refers to any change in the distribution $P(\boldsymbol{y}|\boldsymbol{x})$ which affects the classification labels. Another crucial aspect for concept drift is given by the speed of the changes. A *gradual concept drift* characterizes the case where the transition occurs smoothly over time, while for a *step concept drift* the switch between two contexts occurs abruptly.

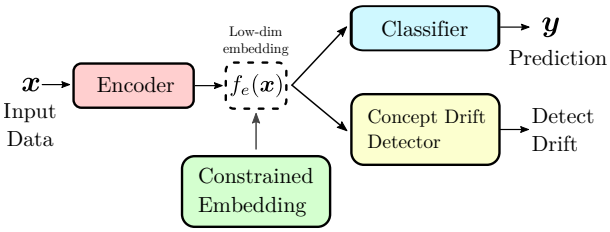


Fig. 1: Proposed Concept Drift architecture.

Concept drift detection is important for many real-world classification tasks, such as anomaly detection, fraud detection or monitoring the electricity load profiles of an industrial facility [5]–[7], as it can provide additional information to improve the performance. Interestingly, the majority of concept drift detectors operate in a *supervised* fashion and require immediate access to ground-truth labels during classifier training *and* testing/inference. But for many of the aforementioned applications, the acquisition of true labels is intractable or at least very expensive. Therefore, labels are possibly only available for training and testing where they can be utilized for building the classifier model, but they are typically not available during inference in the real application scenario. These type of situations are addressed by *unsupervised* or *semi-supervised* drift detection methods, which is a much less explored research area [2]. There exist fully unsupervised drift detection methods which aim for anomalies of an underlying data distribution, but do not take into account labels available during training, thus they are sensitive virtual drifts [8].

In this work, we propose a task-sensitive concept drift detection framework, which can be applied to any dataset in a classification problem, and the labels are only necessary

during training of the model. We learn a meaningful low-dimensional embedding representation of the input data, which is best suited for the classification task. Additionally, we employ a constraint in that embedding representation, which forces the latent representation to have small intra-class and large inter-class distances, and hence it optimally represents the given class structure. After training, concept drifts can be detected by monitoring the distances of incoming new data samples to the learned class centroids in the low-dimensional embedding space, instead of in the high-dimensional input space. The learned representation is highly sensitive to the trained classification task, but does not need true class labels during inference. The actual detection of the drift can be performed by analyzing the distance statistics derived from the embedding representation with any state-of-the-art (SotA) drift detection method. The proposed concept drift architecture is reported in Fig. 1.

In addition, we provide a number of minor improvements: (i) two simple drift detection methods based on the Exponential Moving Average (EMAD), and a modified $Z$-Score (ZSD); (ii) a novel evaluation metric ($H$-score), which accumulates three SotA drift detection metrics into one performance measure; (iii) an extensive evaluation of the proposed framework including ablation studies on the introduced hyperparameters on various datasets for different types of induced concept drift.

## II. Related work

The studies on the concept drift can be divided in two groups [3]: *performance-based* and *data distribution-based* approaches. Performance-based techniques aims at tracking changes in the error rate of a model. These techniques require access to the ground-truth labels. Popular algorithms are Drift Detection Model (DDM) [9] and Adaptive Windowing (ADWIN) [10]. Data-distribution based approaches monitor distribution changes in the data, and are based on statistical tests for distribution similarity. An example are Hellinger Distance Drift Detection Model (HDDDM) [11] and Incremental Kolomogorow-Smirnov test (IKS) [8]. Those algorithms can work with unlabeled data, but they are quite sensitive to data changes and, considering solely input data, bears the risk of detecting drifts in features that may be not important for the prediction model, i.e. wrongly detect virtual drifts. The proposed framework targets to only detect real drifts, i.e. changes in the input data that also have an impact on the classification results.

There have been few tentative attempts to develop unsupervised concept drift detection methods by using neural networks. In [12], the authors use the reconstruction error of an autoencoder to detect changes in the data. In [13], they use a contrastive loss to learn a low-dimensional embedding of data. Both of those methods, are sensitive to virtual drifts because they rely solely on raw input data and ignore the classification task. The internal representation learned by neural network is often utilized also for other purposes, rather than only fulfilling the main classification task. In [14] clustering is applied to the learned representation of the data, rather than in the original space. The authors of [15] and [16] propose to detect drift by using the embedding of a neural network classifier. This method can correctly ignore virtual drifts, but it needs to compute a reference statistic on the training data, which could be difficult to compute while working with large datasets. In contrast, we use very simple distance statistics in the embedding representation and derive the reference statistics from a small data sample with an additional simple update rule to detect drifted samples.

## III. Proposed Framework

### A. Constrained Embedding Representation

We assume that the dataset consists of $n$ data samples $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ with $i = 1, \ldots, n$, where $\boldsymbol{x}_i \in \mathbb{R}^q$ represents $q$-dimensional raw input data and $\boldsymbol{y}_i \in \{0,1\}^k$ the corresponding ground truth label, indicating to which of the $k$-classes the sample belongs (one-hot encoding).

We train a neural network that consists of an *encoder* part $f_e$, which transforms the input data $\boldsymbol{x}_i$ into a low-dimensional embedding representation, $\boldsymbol{e}_i = f_e(\boldsymbol{x}_i) \in \mathbb{R}^d$ with $d \ll q$. A *classifier* neural network $f_c$ takes this low dimensional embedding and produces a prediction of the class labels as output, $\boldsymbol{p}_i = \mathrm{softmax}(f_c(\boldsymbol{e}_i))$. The loss function for the classification task is then given by the cross entropy,

$$\mathcal{L}_c = -\frac{1}{n}\sum_{i=1}^{n} \boldsymbol{y}_i^T \cdot \log(\boldsymbol{p}_i). \tag{1}$$

Additionally, we include a *constrained embedding module* $f_{ce}$, [7] which forces the latent representation to have small intra-class and large inter-class distances. We first initialize the $k$ centroids $\boldsymbol{C}_j \in \mathbb{R}^d$ with $j = 1, \ldots, k$, in the $d$-dimensional embedding space randomly. Then, we iteratively adapt $\boldsymbol{C}_j$ by minimizing the function:

$$\mathcal{L}_{ce} = \frac{1}{n}\sum_{i=1}^{n}\Bigg[\underbrace{\left\|f_e(\boldsymbol{x}_i) - \boldsymbol{C}_{\boldsymbol{y}_i}\right\|_2^2}_{\text{intra-class}} + \underbrace{\log\sum_{j=1}^{k}\exp\left(-\left\|f_e(\boldsymbol{x}_i) - \boldsymbol{C}_j\right\|_2\right)}_{\text{inter-class}}\Bigg] + \ell_{reg}, \tag{2}$$

where $\boldsymbol{C}_{\boldsymbol{y}_i}$ indicates the centroid representing the ground truth class label of sample $i$. The regularization term $\ell_{reg} = -\sum_l^k \min_{l \neq j} \log\left\|\boldsymbol{C}_l - \boldsymbol{C}_j\right\|_2$ aims to create well separated embedding for different classes.

The final total loss function is given by the sum of those contributions: $\mathcal{L} = \mathcal{L}_c + \mathcal{L}_{ce}$. During training, all parameters of the neural networks, as well as the constraint module, are learned and the procedure is summarized in Algorithm 1.

The additional computational complexity of the proposed constraint embedding is negligible compared to the encoding and classification neural networks. Only $k$ additional parameter have to be learned for the constraining module and the the distances in (2) are computed in the low-dimensional embedding space, which is computationally feasible.

### B. Task-Sensitive Concept Drift Detection Framework

In order to detect drifting samples during inference, without access to ground-truth labels, we take the fully trained classifier $f_c$ with the constrained embedding $f_{ce}$, described in the previous section, and analyze the data statistics in the embedding space. As the constrained embedding representation is trained with the classifier, we expect that the drift detector is able to detect drifts only if it is affecting the classification task, i.e. real task-sensitive drift. Since the embedding space is tailored toward a compact representation of classes, we rely on particularly simple distance-based statistics to describe samples within the embedding space.

Let $\boldsymbol{x}_i \in \boldsymbol{X}^{test}$ be a sample of a data stream, the idea is to calculate the Euclidean distances of each sample to the class centroids $\boldsymbol{C}_j$ in the latent space:

$$\boldsymbol{d}_{i,j} = \|f_e(\boldsymbol{x}_i) - \boldsymbol{C}_j\|_2 \,. \tag{3}$$

The centroids $\boldsymbol{C}_j$ are fixed after the training and the distances are expected to be small w.r.t. the centroid of the predicted class, and large to the centroids representing the other classes. Therefore, we extract different statistical feature vectors $\boldsymbol{m}_i$ from the distance statistics of sample $i$ w.r.t. all centroids, $\{\boldsymbol{d}_{i,j}\}_{j=1}^{k}$, and use them for drift detection.

The actual drift detection is done by comparing the feature statistics of each sample to a reference distribution, $\{\boldsymbol{m}^{ref}\}$, which is calculated from some small reference data set and which thereby defines the non-drifted data. Then, the statistics $\boldsymbol{m}_i$ for the sample $i$, is compared against $\{\boldsymbol{m}^{ref}\}$ by a change detection method $\mathcal{DM}$:

$$o_i = \mathcal{DM}(\boldsymbol{m}_i, \{\boldsymbol{m}^{ref}\}), \tag{4}$$

where $o_i \in [0, 1]$ is a binary flag that signals if the test sample is significantly different from the reference distribution. Note that $\mathcal{DM}$ can be any unsupervised change detection method.

Finally, in order to reduce the ratio of false detections, we declare that a drift happened only if the number of raw detections $o_i$ in the last $w$ samples is higher than a detection threshold $r$. Precisely, we report a drift at the time $i$ if

$$\frac{1}{w}\sum_{t=0}^{w-1} o_{i-t} > r. \tag{5}$$

In case no drift is detected, the reference statistics $\{\boldsymbol{m}^{ref}\}$ is updated with the test sample statistic $\boldsymbol{m}_i$ of undrifted samples, i.e. when $o_i = 0$. The pseudo-code of the proposed concept drift detection framework is reported in Algorithm 2.

Recall that the distances of (3), and the statistics are calculated on the $d$-dimensional embedding space, hence its computational cost is relatively small, when the number of classes $k$ and dimensions $d$ are not high (O($dk$)).

### C. Unsupervised Drift Detection Methods

The proposed drift detection framework is flexible to be used with any unsupervised change detection algorithm. We propose two efficient novel detectors, tailored for our specific representation and allow for continuous self-calibration of their meta-parameters.

**Algorithm 1:** Training with constrained embedding

**Require:** Data $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_n$, training model $\mathcal{M}$: classifier $f_c$, encoder $f_e$, constraint module $f_{ce}$
**Output:** Trained model $\mathcal{M}$: $f_c$, $f_e$, $f_{ce}$, centroids $\{\boldsymbol{C}_j\}_k$

```
{C_j} ← random initialization                    // Initialize the centorids
for training epoch t = 0 to t_end do
    Fetch mini-batch data {(x_i, y_i)}_b at current epoch t
    e_i = f_e(x_i)                               // Embedding forward pass
    p_i = softmax(f_c(e_i))                      // Classifer forward pass
    L_c(p_i, y_i) ← Eq. 1                        // Classification loss
    L_ce(e_i, {C_j}) ← Eq. 2                     // Constrained embedding loss
    Update M by SGD on L_c + L_ce                // Backward pass
end
```

**Algorithm 2:** Task-sensitive drift detection framework

**Require:** Trained encoder $f_e$, centroids $\{\boldsymbol{C}_j\}_k$, reference data $\{\boldsymbol{X}\}^{ref}$, test data $\{\boldsymbol{X}\}^{test}$, unsupervised detector $\mathcal{DM}$, detection history size $w$, detection threshold $r$
**Output:** Drift detection index $i_d$

```
i = 0 , i_d = ∞                                  // Init sample and drift index
/* Calculate reference statistics                                          */
for x_r in {X^ref} do
    d_{r,j} = ||f_e(x_r) − C_j||_2                // Distances to centroids
    {m^ref} ← get statistical features of {d_{r,j}}_{j=1}^k   // Reference
      statistics
end
/* Begin Concept Drift detection                                           */
for x_i in {X^test} do
    d_{i,j} = ||f_e(x_i) − C_j||_2                // Distances to centroids
    m_i ← get statistical features of {d_{i,j}}_{j=1}^k   // Get statistics
    o_i = DM(m_i, {m^ref})                       // Compare statistics
    if (1/w) Σ_{t=0}^{w−1} o_{i−t} > r then
        i_d = i                                  // Drift detected, end algorithm
        break
    else if o_i = 0 then
        {m^ref} ← m_i                            // Update reference statistics
    end
    i = i + 1
end
```

*1) **Exponential Moving Average Detector (EMAD)**:* It monitors the exponentially weighted running statistics of the distance of each sample to its closest centroid. In this case the feature vector $m_i$ just has one entry and is defined as

$$m_i = \min_j \boldsymbol{d}_{i,j} \,. \tag{6}$$

For the change detector we calculate the exponential moving averages of the feature and its variance,

$$\mu_i = \lambda \cdot \mu_{i-1} + (1 - \lambda) \cdot m_i \tag{7}$$

$$\sigma_i = \lambda \cdot \sigma_{i-1} + (1 - \lambda) \cdot (m_i - \mu_i)^2 \tag{8}$$

with the forgetting factor $\lambda = 0.95$ and $i$ the time stamp.

We detect a change when the actual average $\mu_i$ is greater than a threshold $\beta$, i.e. $\mu_i > \beta$. Here, $\beta$ is an adaptive threshold which is initialized with the reference-set $\boldsymbol{X}^{ref}$ statistic,

$$\beta = \mu_{\boldsymbol{m}_r} + \sigma_{\boldsymbol{m}_r}. \tag{9}$$

Note that we use $\mu + \sigma$ as threshold as we only flag when the actual metric is *significantly greater* than the reference (not when it is smaller). When no change is detected, the threshold $\beta$ is updated with the current statistics: $\beta = \mu_i + \sigma_i$.

*2) **Z-Score Detector (ZSD)**:* The feature is given by the distance of each sample to its closest centroid $m_i$, Eq. (6). The feature is transformed to a $z$-score $z$, using the mean ($\mu$) and standard deviation ($\sigma$) of the reference statistic $\boldsymbol{m}_r$,

$$z_i = \frac{m_i - \mu_{\boldsymbol{m}_r}}{\sigma_{\boldsymbol{m}_r}} \tag{10}$$

We then compute a $p$-value using the cumulative distribution function of the normal distribution $\Phi$ as $p_i = \Phi(z_i)$. A change is detected if the $p$-value is below a confidence level $\alpha$, i.e. $p_i < \alpha$ with $\alpha = 0.05$. When no change is detected, the $m_i$ is used to update the reference statistic $\boldsymbol{m}_r$, and the $\mu_{\boldsymbol{m}_r}$ and $\sigma_{\boldsymbol{m}_r}$ are dynamically updated with (7) and (8) respectively.

## IV. Experimental setup

### A. Comparative methods

We also report results using state-of-the-art unsupervised detection methods which are briefly describe in the following.

*1) **Hellinger Distance Drift Detector Method (HDDDM)**:* It monitors the Hellinger distance to detect a drift between two multivariate distributions [11]. Since HDDDM is a batch-based method, we modified this approach in the same way as [17], such that a batch is defined incrementally by a sliding window of $5w$ samples, with a stride of one sample. Additionally, to detect a change, we directly use the Hellinger distance $\delta_H$: $\delta_H > \beta$, with $\beta = \mu + \sigma$, as in [11]. The threshold is initialized with the first $5w$ samples to calculate the statistics $\mu$ and $\sigma$.

We do experiment with HDDDM on both, the raw input data $\boldsymbol{x}$, and on the network low-dimensional embedding $f_e(\boldsymbol{x})$. In the remainder of the paper, the methods are referred as HDDDM$_I$ and HDDDM$_E$ respectively.

*2) **Incremental Kolmogorov–Smirnov test (IKS)**:* It is an online variant of the Kolmogorow-Smirnov (KS) test [8]. The principle behind IKS detector is to apply the KS test on each individual feature. The detection of a change in a single feature may be sufficient to trigger the presence of a concept drift. The window size used is $5w$ and the confidence level $\alpha$ is 0.01. First, we compute the pairwise distances $\boldsymbol{d}_{i,j}$ with (3). Then, we create the feature vectors $\{\boldsymbol{m}^{ref}\}$ and $\boldsymbol{m}$, where we use the *mean*, *standard deviation*, *maximum* and *minimum* as statistical features, respectively on the reference and test-set. We use those feature vectors derived from the embedding representation with the IKS detection method.

### B. Datasets and preprocessing

We conduct our experiments on two synthetic and seven real-world datasets. A summary of the characteristics of the used datasets is reported in Table I. The synthetic datasets *RBF* and *MovingRBF* are created using the `make_classification` method of the scikit-learn Python toolbox [18]. For both datasets, we create four clusters with unit variance in a ten dimensional feature space. For the *RBF* dataset we add ten more features to each feature vector, where five are redundant and five are just white noise.

Additionally, we consider seven real-world datasets which are already utilized in others drift detection works [17], [19].

TABLE I: Characteristics of datasets used.

| Dataset | #Instances | #Attributes | #Classes |
|---|---|---|---|
| *adult* | 48842 | 65 | 2 |
| *bank* | 45211 | 48 | 2 |
| *digits08* | 1499 | 16 | 2 |
| *digits17* | 1557 | 16 | 2 |
| *musk* | 6598 | 166 | 2 |
| *phishing* | 11055 | 46 | 2 |
| *wine* | 6497 | 12 | 2 |
| *RBF* | 10000 | 20 | 4 |
| *MovingRBF* | 10000 | 10 | 4 |

All the datasets were processed to have only numeric and binary values, and normalized to have each attribute zero mean and unit standard deviation.

The initial 50% of each dataset is assumed to be labeled, and is used to train the model. The last half of the data is considered as an unsupervised data stream, and it is used for testing. The first 25% of the test data is used as reference dataset to initialize the statistics for the drift detectors. A concept drift is induced in the last 50% of the test-stream.

### C. Inducing concept drift

By controlling the location and nature of the drift, it is possible to evaluate the drift detection capabilities of different methods. The drift induction process is a standard benchmark framework for drift detection [17]. The features are corrupted based on their importance to the classification task to create a task-dependent drift. First, the features are ranked based on their information gain [19], and then either the most or least informative features are selected, by choosing the top 25% (*most informative*) or the bottom 25% (*least informative*) of features from the ranked list.

We focus on both *step* and *gradual* drifts. The *step* drift is induced by randomly shuffling the values of a subset of features [19]. This approach ensures that feature drifts are induced while also maintaining the original data properties of the dataset. The *gradual* drift is induced by corrupting a subset of features with noise $\eta \sim \mathcal{N}(\mu, \sigma^2)$. While $\mu = 1$, the value of $\sigma$ increases over time, from $\sigma = 0$ at 50% to $\sigma = 2$ after 75% of the data stream. This corruption strategy simulates a gradual mean shift of the features. Differently from [19], we corrupt the samples of all classes, instead of picking only one class. In this way, the drift has bigger impact on the classification task, and also could be relevant when modifying the least important features.

Only in the *MovingRBF* dataset, we induce the drift by randomly re-sampling the location of the class centroids, and then moving them from the initial to this final position when generating the data, either in a step or gradual manner. In this way, we directly affect the position of the class centroids, while in the other drift induction strategies, it is a consequence of the corruption of the attributes.

### D. Implementation details

We use the same architecture and hyper-parameters for all experiments. The encoder is a multilayer perceptron (MLP) neural network. It consists of three hidden layers of 256,

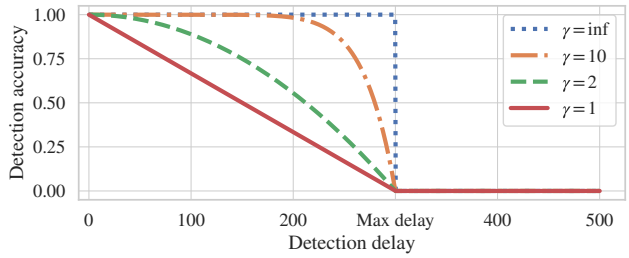


Fig. 2: Penalized drift detection accuracy $\widehat{DA}$.

64 and 3 neurons. Hence, the embedding representation has dimension $d = 3$ ($f_e(\boldsymbol{x}) \in \mathbb{R}^3$). The classifier is a single fully connected layer with the number of output neurons equal to the number of classes. The ReLU non-linear function is used for the hidden layers and the softmax activation for the output. The optimizer used is SGD with 0.9 of momentum. Also, a dropout of 0.25 and $L_2$ norm penalization with weight 0.001 is employed in order to reduce generalization error.

Unless explicitly reported, the default value of the drift detection history, maximum delay and detection ratio are $w = 50$, $\delta_{max} = 6w$, and $r = 0.25$ respectively. The source code used in the experiments is publicly available on `http://github.com/Castel44/TSDD`.

### E. Evaluation measures

A dataset exhibits *real drift*, if the performance of the classifier is significantly degraded by the drift. With the generalization error defined as $GE = |Acc_{Train} - Acc_{Valid}|$, we attribute a real drift to a dataset if the classification accuracy drops below the validation accuracy with the generalization error as margin, i.e. $Acc_{Test} < Acc_{Valid} - GE$. If the performance after the drift does not decrease below this threshold, a *virtual drift* occurs which should not be detected.

The performance of drift detectors are usually assessed by Detection Accuracy $DA$, the False Alarm Rate (False Positive Rate $FPR$) and the Detection Delay $\delta$ [4], [19]. The $DA$ is a binary value that signals the correct detection of the drift event. Note that, in case of virtual drift, the goal is to not have any detection, i.e. any detection is considered a false positive. In case of real drift, we account for the detection delay by penalizing $DA$ with a term proportional to the delay $\delta$,

$$\widehat{DA} = \begin{cases} DA - \left(\frac{\delta}{\delta_{max}}\right)^{\gamma} & \text{for real drift} \\ DA & \text{for virtual drift} \end{cases}, \tag{11}$$

where $\delta_{max}$ is the maximum acceptable delay which is set to $\delta_{max} = 300$ samples and $\gamma > 0$ a free parameter. The penalization allows for the discrimination between two detected drifts which have different delay. Drifts which are detect earlier, have higher. i.e. better, penalized $DA$ values than drifts which are detected later, see Fig 2. The free parameter $\gamma$ allows to adjust the strength of the penalization as function of the delay. Smaller values of $\gamma$ lead to larger suppression of the detection accuracy for larger delays. In our studies, we set $\gamma$=2.

In order to characterize the performance with a single metric, we propose an accumulation of those performance values. With the true negative rate $TNR$ = 1-$FPR$, we define the metric $H$ as the harmonic mean between $\widehat{DA}$ and $TNR$,

$$H = 2 \cdot \frac{\widehat{DA} \cdot TNR}{\widehat{DA} + TNR} \tag{12}$$

The $H$ metric is continuous and bounded $H \in [0, 1]$. $H = 1$ indicates perfect detection and no false alarms, and $H = 0$ means that either the drift has not been detected or the false alarm rate is 1.

All experiments have been repeated 10 times with different random seeds. In order to assess the statistical significance of the results, we use the Friedman non-parameteric test with 0.05 confidence level, followed by Nemenyi post-hoc test. The results are visualized with the critical difference diagram [20], where a thick horizontal line shows the algorithms that are not significantly different in terms of $H$-score. This evaluation approach allows the simultaneous comparison of different methods considering several data sets.

## V. Results and discussion

In our experiments, the goal is to pinpoint the single drift event, in case of *real concept drift*, and to not detect any change in case of *virtual drift*, where the changes in the data does not have an impact on the predictions. The effect of corrupting the most and least informative feature is reported in Table II, where the similarity of train and validation accuracy indicates an initial static dataset, and the effect of drift is reported by a drop of the accuracy in the test-set. We report real drift in all the datasets when the most informative features are corrupted. Instead, the corruption of the least informative features induces real drift only on *digits08* (only step drift) and *phishing* datasets. While, the others, reports virtual drift.

We empirically validate the effect of the constraint embedding module on the latent representation by using the Generalized Variance (GV) [21] of the validation data around its class centroids. The GV is proportional to the area of the ellipsoid in the $d$-dimensional space [22], and it includes meaningful information about the data sparsity. The effect of the constraining embedding module is to reduce the GV of the latent representation by 63.31% to 99.95% depending on the dataset. Therefore, the constraint module leads to a strong concentration of data samples around the centroids as desired.

We compare the performance of the proposed drift detection framework with the algorithms EMAD and ZSD against different SotA detection algorithms. The performance is evaluated by using the proposed constrained embedding module (indicated by ✓), which returns the class centroids at the end of the training, and compared to the case where the constraints module was not used during training (indicated by ✗), where the class centroids need to be calculated after the training, as the average of the training data of each class. Additionally, we also compare to the SotA for unsupervised concept drift detection, with an analysis directly on the input data distribution with the HDDDM$_I$ algorithm.

We report the drift detection results for a representative subset of the all the studied datasets in Table III. The result of all datasets can be found in the supplementary material [23]. By analyzing all the results, we observe that the proposed framework, with the inclusion of the constraint embedding module, outperforms the SotA counterpart in terms of $H$-score in 79 experiments, has the same performance level in 30, and is worse in 27 out of 136 experiments. Moreover, the proposed framework always shows comparable or better performance than the unsupervised drift detection on the raw input data, HDDDM$_I$, which is not able to deal with virtual drifts as expected. All methods successfully detect the drift on the synthetic dataset *MovRBF*.

In the Table III, we can observe the advantage of the cumulative measure $H$, which accounts for all three performance measures *DA*, *TNR* and *Delay*. For example, comparing the performance of the *IKS* algorithm on the *wine* dataset with step drift on the most important features, the performance is almost perfect for *DA* and *TNR*, but the delay is rather large when trained without the constraints on the embedding (indicated with ✗). This is reflected in a reduced performance measure $H$. In comparison, including the constraint (✓) also gives almost perfect performance on *DA* and *TNR*, but reduces the delay considerably, which is reflected in an improved $H$ measure. In that way, the $H$ measure gives a more comprehensive picture of the performance as relying on one single metrics. Also, from that example, it is clear that the inclusion of the constrained embedding module improves the performance.

In Fig. 3 we report the critical differences based on the $H$-score of the investigated algorithms accumulated over all the datasets and drift types. The algorithms using the proposed constrained embedding framework (subscript ✓) are in all cases, except for HDDDM$_E$, significantly better than the corresponding method without the constraint embedding (subscript ✗). However, there is no single detector that outperform all the others since between the proposed EMAD$_✓$, ZSD$_✓$ and IKS$_✓$ the difference is not statistically significant.

For the experiments on datasets showing real drift as reported in Fig. 4 the trend is even clearer. All approaches using the constraints on the embedding are significantly better than the corresponding approaches without the constraints. Only HDDDM$_I$, working on the raw input, shows a similar performance as the proposed constrained embedding approaches. However, the performance of HDDDM$_I$ is the worst on the datasets showing virtual drift as it can be expected (see Fig. 5). For the other approaches working on the low dimensional embedding, there is no difference between methods with and without constrains, because virtual drift does not affect the model performance and the latent representation substantially.

Finally, we report ablation studies on the hyperparameters introduced with the proposed framework, in particular the drift detection history size $w$ and the detection threshold ratio $r$. In Fig. 6, we report the average rank of the selected hyperparameter combination $(w, r)$. For very small $r \approx 0$ a drift is reported as soon as a single changed data sample is detected, which leads to many false positives for small $r$ and the corresponding low performance. With a large detection history and detection threshold, i.e. ($w = 100$ and $r = 0.5$), the performance are also low, because of the large delay for the detection. The best results are achieved with a detection window of 50 samples and a detection threshold ratio from 0.1 to 0.5.

TABLE II: Classification accuracies of train, validation, and test-set when the most and least informative features are corrupted. Cases of virtual drift are highlighted.

| Dataset | Train | Valid | *Step drift* | | *Gradual drift* | |
|---|---|---|---|---|---|---|
| | | | *Most* | *Least* | *Most* | *Least* |
| *adult* | 88.8±0.1 | 86.0±0.2 | 77.4±3.4 | **84.9±0.1** | 72.3±0.8 | **84.8±0.1** |
| *bank* | 95.3±0.1 | 91.5±0.2 | 77.4±2.8 | **90.7±0.1** | 81.2±0.5 | **89.5±0.3** |
| *digits08* | 100.0±0.0 | 99.6±0.3 | 96.3±3.2 | 96.7±1.2 | 98.5±0.6 | **99.4±0.1** |
| *digits17* | 100.0±0.0 | 100.0±0.0 | 97.5±0.5 | **99.1±0.4** | 90.9±0.7 | **99.4±0.2** |
| *musk* | 100.0±0.0 | 98.8±0.1 | 93.6±1.6 | **98.1±0.4** | 80.6±0.2 | **89.4±1.0** |
| *phishing* | 98.8±0.0 | 95.0±0.4 | 76.7±4.6 | 90.4±0.5 | 88.9±0.4 | 90.8±0.3 |
| *wine* | 100.0±0.0 | 100.0±0.0 | 80.8±5.7 | **100.0±0.0** | 89.7±0.4 | **100.0±0.0** |
| *RBF* | 99.9±0.1 | 99.1±0.5 | 93.5±3.4 | **99.1±0.6** | 90.8±4.1 | **98.9±0.6** |
| *MovRBF* | 100.0±0.0 | 100.0±0.0 | 49.9±9.0 | - | 52.4±7.7 | - |

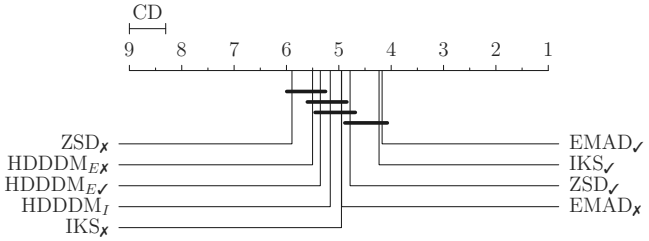


Fig. 3: Critical difference diagram based on the $H$-score accumulated over all datasets and drift types showing pairwise statistical difference comparison of different drift detectors.

## VI. Conclusion and future work

In this work, we presented a novel semi-supervised task-sensitive concept drift detection framework, which is based on learning a constrained low-dimensional embedding representation. The constraint imposed during the learning of the classification task leads to well separated clusters for each class, and allows for a robust detection of drift in incoming novel data samples. Any unsupervised change detection algorithm can be used for analyzing the statistics of the distances between incoming data samples and the centroids in the low-dimensional embedding representation, and thereby detecting drifted individual samples. We also introduced two simple detection methods, EMAD and ZSD, which are based on the exponential moving average and on a modified $z$-score, respectively. To compare the performance of different drift detection algorithms, we introduced a bounded and continuous metric $H$, which accumulates the detection accuracy, delay and false positive rate into one value.

We conducted experiments on nine benchmarks datasets where different types of drift have been induced. The results indicated that the proposed framework is capable of robustly detecting real concept drifts, while properly ignoring virtual drift, where the classification accuracy is not affected by the drift. Furthermore, the proposed change detectors EMAD and

TABLE III: Drift detection results. The presence (or not) of the constraint module is reported by ✓ (or ✗). Highlighted is the best performing in the comparison with and without constraint module.

| Dataset | Detector | ✗/✓ | *Step drift: most important* | | | | *Step drift: least important* | | | | *Gradual drift: most important* | | | | *Gradual drift: least important* | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | *DA* | *TNR* | *Delay* | *H* | *DA* | *TNR* | *Delay* | *H* | *DA* | *TNR* | *Delay* | *H* | *DA* | *TNR* | *Delay* | *H* |
| *digits08* | ZSD | ✗ | **1.00±0.00** | 0.94±0.14 | 11.80±16.34 | **0.96±0.09** | 0.50±0.53 | 0.99±0.04 | - | 0.49±0.51 | 0.60±0.52 | 0.99±0.04 | 14.20±25.53 | 0.60±0.52 | 0.30±0.48 | 1.00±0.00 | 78.70±47.96 | 0.30±0.48 |
| | | ✓ | 0.90±0.32 | **1.00±0.00** | 7.10±3.96 | 0.90±0.32 | **1.00±0.00** | 1.00±0.00 | - | **1.00±0.00** | 0.60±0.52 | 1.00±0.00 | **9.10±2.38** | 0.60±0.52 | **0.60±0.52** | 1.00±0.00 | **19.00±46.24** | **0.60±0.52** |
| | EMAD | ✗ | 0.90±0.32 | 1.00±0.00 | 23.00±38.27 | 0.90±0.32 | 0.50±0.53 | 1.00±0.00 | - | 0.50±0.52 | 0.50±0.53 | 1.00±0.00 | 12.20±29.35 | 0.50±0.53 | 0.60±0.52 | 1.00±0.00 | **0.00±0.00** | **0.60±0.52** |
| | | ✓ | 0.90±0.32 | 1.00±0.00 | **9.50±3.54** | 0.90±0.32 | **1.00±0.00** | 1.00±0.00 | - | **1.00±0.01** | 0.60±0.52 | 1.00±0.00 | 12.00±2.62 | 0.60±0.52 | 0.40±0.52 | 1.00±0.00 | 99.30±47.51 | 0.39±0.50 |
| | IKS | ✗ | 0.60±0.52 | 1.00±0.00 | 92.10±60.28 | 0.60±0.51 | 0.60±0.52 | 1.00±0.00 | - | 0.57±0.49 | 0.60±0.52 | 1.00±0.00 | **76.50±7.92** | 0.60±0.51 | 0.30±0.48 | 1.00±0.00 | 105.30±20.68 | 0.30±0.48 |
| | | ✓ | **0.90±0.32** | 1.00±0.00 | **41.20±9.89** | **0.90±0.32** | **1.00±0.00** | 1.00±0.00 | - | **0.99±0.00** | 0.70±0.48 | 1.00±0.00 | 108.30±96.02 | 0.64±0.45 | **0.70±0.48** | 1.00±0.00 | **20.70±65.46** | **0.68±0.47** |
| | $HDDDM_E$ | ✗ | 0.50±0.53 | 1.00±0.00 | 145.00±0.00 | 0.47±0.50 | 0.70±0.48 | 1.00±0.00 | - | 0.63±0.44 | 0.40±0.52 | 1.00±0.00 | 131.00±46.06 | 0.38±0.48 | **0.50±0.53** | 1.00±0.00 | 145.30±59.23 | **0.48±0.50** |
| | | ✓ | **0.80±0.42** | 1.00±0.00 | 130.60±45.89 | **0.75±0.40** | **1.00±0.00** | 1.00±0.00 | - | **0.94±0.00** | 0.50±0.53 | 1.00±0.00 | 145.90±59.64 | 0.45±0.48 | 0.30±0.48 | 1.00±0.00 | 145.00±0.00 | 0.28±0.45 |
| | $HDDDM_I$ | - | 0.50±0.53 | 1.00±0.53 | 158.00±21.28 | 0.45±0.48 | 0.40±0.52 | 1.00±0.00 | - | 0.40±0.52 | 0.50±0.53 | 1.00±0.53 | 0.00±0.00 | 0.50±0.53 | 0.60±0.52 | 1.00±0.00 | 0.00±0.00 | 0.60±0.52 |
| *phishing* | ZSD | ✗ | **1.00±0.00** | 0.08±0.04 | **0.00±0.00** | 0.14±0.07 | 0.50±0.53 | 0.44±0.29 | 261.10±331.56 | 0.30±0.38 | 0.70±0.48 | **0.39±0.17** | 201.80±203.94 | 0.28±0.26 | 0.40±0.52 | **0.58±0.20** | 273.30±377.38 | 0.33±0.43 |
| | | ✓ | 0.90±0.32 | **0.77±0.21** | 80.80±155.72 | **0.82±0.28** | 0.70±0.48 | 0.13±0.07 | 0.00±0.00 | 0.19±0.16 | **1.00±0.00** | 0.13±0.07 | **0.00±0.00** | 0.22±0.11 | **0.90±0.32** | 0.13±0.06 | **0.00±0.00** | **0.22±0.12** |
| | EMAD | ✗ | 0.10±0.32 | 1.00±0.00 | **6.30±19.92** | 0.10±0.31 | 0.30±0.48 | 1.00±0.00 | 0.00±0.00 | 0.30±0.48 | 0.00±0.00 | 1.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.10±0.32 | 1.00±0.00 | 0.00±0.00 | 0.10±0.32 |
| | | ✓ | **0.70±0.48** | 1.00±0.00 | 197.50±214.28 | **0.66±0.46** | **0.70±0.48** | 1.00±0.00 | 0.00±0.00 | **0.70±0.48** | **0.20±0.42** | 1.00±0.00 | 0.00±0.00 | **0.20±0.42** | **0.90±0.32** | 1.00±0.00 | 0.00±0.00 | **0.90±0.32** |
| | IKS | ✗ | 1.00±0.00 | 0.71±0.09 | 9.20±15.96 | 0.83±0.06 | 0.30±0.48 | 0.66±0.05 | **0.00±0.00** | 0.24±0.38 | 0.70±0.48 | 0.68±0.09 | 135.25±56.36 | 0.55±0.38 | 0.20±0.42 | 0.70±0.07 | **0.00±0.00** | 0.17±0.36 |
| | | ✓ | 1.00±0.00 | 0.78±0.12 | 16.60±35.15 | 0.87±0.08 | **0.60±0.52** | 0.73±0.14 | 44.80±39.99 | **0.51±0.44** | 0.70±0.48 | 0.73±0.17 | 149.40±53.25 | 0.50±0.36 | **0.70±0.48** | 0.76±0.08 | 186.20±438.38 | **0.59±0.41** |
| | $HDDDM_E$ | ✗ | 1.00±0.00 | 0.82±0.04 | 111.10±39.41 | 0.87±0.05 | 0.70±0.48 | 0.81±0.03 | 403.30±606.53 | 0.63±0.43 | 0.30±0.48 | 0.78±0.04 | 558.20±402.23 | 0.06±0.17 | **0.90±0.32** | 0.81±0.06 | 164.80±408.80 | **0.80±0.28** |
| | | ✓ | 1.00±0.00 | 0.78±0.03 | 132.40±42.85 | 0.83±0.06 | 0.40±0.52 | 0.77±0.03 | 146.20±386.87 | 0.32±0.42 | 0.20±0.42 | 0.79±0.03 | 443.30±349.13 | 0.16±0.34 | 0.20±0.42 | 0.75±0.04 | 369.90±593.49 | 0.17±0.35 |
| | $HDDDM_I$ | - | 1.00±0.00 | 0.84±0.00 | 14.30±2.50 | 0.91±0.00 | 0.30±0.48 | 0.84±0.00 | 17.70±4.00 | 0.27±0.44 | 0.80±0.42 | 0.84±0.42 | 12.00±0.00 | 0.73±0.38 | 0.10±0.32 | 0.84±0.00 | 11.00±0.00 | 0.09±0.29 |
| *wine* | ZSD | ✗ | 1.00±0.00 | 0.67±0.13 | 19.30±21.97 | 0.80±0.10 | 0.60±0.52 | 0.72±0.19 | - | 0.55±0.47 | 1.00±0.00 | 0.79±0.10 | 18.50±12.26 | 0.88±0.06 | 0.10±0.32 | 0.63±0.18 | - | 0.09±0.30 |
| | | ✓ | 1.00±0.00 | **1.00±0.00** | 28.30±13.15 | **1.00±0.01** | **1.00±0.00** | **1.00±0.00** | - | **1.00±0.00** | 1.00±0.00 | **1.00±0.00** | 99.80±11.43 | **0.98±0.01** | **1.00±0.00** | **1.00±0.00** | - | **1.00±0.00** |
| | EMAD | ✗ | 0.00±0.00 | **1.00±0.00** | **0.00±0.00** | 0.00±0.00 | 1.00±0.00 | 1.00±0.00 | - | 1.00±0.00 | 1.00±0.00 | 1.00±0.00 | 68.30±64.54 | 0.96±0.12 | 1.00±0.00 | 1.00±0.00 | - | 1.00±0.00 |
| | | ✓ | **1.00±0.00** | 0.95±0.02 | 13.50±5.64 | **0.97±0.01** | 1.00±0.00 | 0.95±0.02 | - | 0.99±0.01 | 1.00±0.00 | 0.95±0.02 | 41.30±28.06 | 0.97±0.02 | 1.00±0.00 | 0.95±0.03 | - | 0.97±0.03 |
| | IKS | ✗ | 0.90±0.18 | 1.00±0.00 | 224.00±54.86 | 0.68±0.21 | 0.30±0.48 | 1.00±0.00 | - | 0.30±0.48 | 1.00±0.00 | 1.00±0.00 | 58.40±8.82 | 1.00±0.00 | 0.30±0.48 | 1.00±0.00 | - | 0.30±0.48 |
| | | ✓ | 1.00±0.00 | 1.00±0.00 | **94.20±18.60** | **0.98±0.01** | **0.90±0.32** | 1.00±0.00 | - | **0.90±0.32** | 1.00±0.00 | 1.00±0.00 | 61.70±25.49 | 0.99±0.01 | **1.00±0.00** | 1.00±0.00 | - | **1.00±0.00** |
| | $HDDDM_E$ | ✗ | 1.00±0.00 | 0.88±0.05 | 39.60±16.85 | 0.93±0.03 | 0.50±0.53 | 0.88±0.06 | - | 0.47±0.50 | 1.00±0.00 | 0.88±0.04 | 42.60±9.99 | 0.93±0.02 | 0.30±0.48 | 0.92±0.06 | - | 0.29±0.47 |
| | | ✓ | 1.00±0.00 | 0.89±0.08 | 23.40±13.75 | 0.94±0.05 | 0.70±0.48 | 0.90±0.06 | - | 0.66±0.46 | 1.00±0.00 | 0.85±0.05 | 37.80±11.35 | 0.92±0.03 | 0.40±0.52 | 0.89±0.06 | - | 0.37±0.48 |
| | $HDDDM_I$ | - | 1.00±0.00 | 0.88±0.00 | 22.50±2.42 | 0.93±0.00 | 0.00±0.00 | 0.88±0.00 | - | 0.00±0.00 | 1.00±0.00 | 0.88±0.00 | 44.90±1.97 | 0.93±0.00 | 0.00±0.00 | 0.88±0.00 | - | 0.00±0.00 |
| *RBF* | ZSD | ✗ | 0.90±0.32 | 0.06±0.04 | **0.00±0.00** | 0.10±0.08 | 0.10±0.32 | 0.08±0.06 | - | 0.00±0.01 | 1.00±0.00 | 0.10±0.10 | 3.80±9.81 | 0.17±0.16 | 0.10±0.32 | 0.08±0.08 | - | 0.01±0.03 |
| | | ✓ | **1.00±0.00** | **0.51±0.27** | 17.40±38.19 | **0.64±0.21** | **0.40±0.52** | **0.56±0.30** | - | **0.31±0.42** | 1.00±0.00 | **0.49±0.25** | 4.20±6.81 | **0.62±0.24** | 0.20±0.42 | **0.51±0.24** | - | **0.14±0.29** |
| | EMAD | ✗ | 0.10±0.32 | 1.00±0.00 | **0.00±0.00** | 0.10±0.32 | 0.90±0.32 | 1.00±0.00 | - | 0.90±0.32 | 0.60±0.52 | 1.00±0.00 | 190.20±163.24 | 0.45±0.48 | 0.90±0.32 | 1.00±0.00 | - | 0.90±0.32 |
| | | ✓ | **0.90±0.32** | 1.00±0.00 | 90.20±87.50 | **0.89±0.31** | 0.80±0.42 | 1.00±0.00 | - | 0.80±0.42 | **1.00±0.00** | 1.00±0.00 | **47.00±48.84** | **0.99±0.02** | 0.90±0.32 | 1.00±0.00 | - | 0.90±0.32 |
| | IKS | ✗ | 0.60±0.52 | 1.00±0.00 | 259.90±128.96 | 0.37±0.40 | 0.80±0.42 | 1.00±0.01 | - | 0.80±0.42 | 0.90±0.32 | 1.00±0.00 | 207.70±142.06 | 0.74±0.37 | **0.90±0.32** | 0.99±0.04 | - | **0.89±0.31** |
| | | ✓ | **0.90±0.32** | 0.99±0.04 | 169.00±120.60 | **0.83±0.30** | 0.80±0.42 | 0.99±0.03 | - | 0.79±0.42 | **1.00±0.00** | 1.00±0.00 | **85.40±33.56** | **0.98±0.02** | 0.50±0.53 | 1.00±0.01 | - | 0.50±0.53 |
| | $HDDDM_E$ | ✗ | 0.90±0.32 | 0.88±0.07 | 100.40±50.92 | 0.81±0.29 | 0.80±0.42 | 0.88±0.05 | - | 0.68±0.41 | 1.00±0.00 | 0.88±0.09 | 66.10±34.42 | 0.93±0.05 | 0.60±0.52 | 0.89±0.08 | - | 0.57±0.49 |
| | | ✓ | **1.00±0.00** | 0.87±0.08 | 82.20±24.98 | **0.92±0.05** | 0.50±0.53 | 0.90±0.10 | - | 0.47±0.49 | 1.00±0.00 | 0.91±0.07 | 50.10±25.91 | 0.95±0.04 | 0.50±0.53 | 0.93±0.09 | - | 0.46±0.49 |
| | $HDDDM_I$ | - | 0.90±0.32 | 0.89±0.32 | 165.10±122.68 | 0.65±0.37 | 0.70±0.48 | 0.89±0.08 | - | 0.65±0.45 | 1.00±0.00 | 0.89±0.00 | 41.80±30.26 | 0.94±0.04 | 0.10±0.32 | 0.89±0.08 | - | 0.10±0.31 |
| *MovRBF* | ZSD | ✗ | 1.00±0.00 | 0.82±0.18 | 2.50±2.07 | 0.89±0.11 | - | - | - | - | 1.00±0.00 | 0.86±0.00 | **5.80±4.05** | 0.89±0.22 | - | - | - | - |
| | | ✓ | 1.00±0.00 | **1.00±0.00** | 4.10±0.32 | **1.00±0.00** | - | - | - | - | 1.00±0.00 | **1.00±0.00** | 65.40±40.07 | **0.99±0.01** | - | - | - | - |
| | EMAD | ✗ | 1.00±0.00 | 1.00±0.00 | 6.20±1.75 | 1.00±0.00 | - | - | - | - | 1.00±0.00 | 1.00±0.00 | 23.10±28.78 | **1.00±0.01** | - | - | - | - |
| | | ✓ | 1.00±0.00 | 1.00±0.00 | 4.00±0.00 | 1.00±0.00 | - | - | - | - | 1.00±0.00 | 1.00±0.00 | 17.70±9.87 | 1.00±0.00 | - | - | - | - |
| | IKS | ✗ | 1.00±0.00 | 1.00±0.00 | 57.20±34.52 | 0.99±0.01 | - | - | - | - | 1.00±0.00 | 1.00±0.00 | 77.70±30.74 | 0.99±0.02 | - | - | - | - |
| | | ✓ | 1.00±0.00 | 0.99±0.04 | **29.30±15.17** | 0.99±0.02 | - | - | - | - | 1.00±0.00 | 1.00±0.00 | 50.80±22.46 | 1.00±0.00 | - | - | - | - |
| | $HDDDM_E$ | ✗ | 1.00±0.00 | 1.00±0.00 | 7.40±1.84 | 1.00±0.00 | - | - | - | - | 1.00±0.00 | 1.00±0.00 | 26.70±25.45 | 1.00±0.00 | - | - | - | - |
| | | ✓ | 1.00±0.00 | 1.00±0.00 | 5.90±0.57 | 1.00±0.00 | - | - | - | - | 1.00±0.00 | 1.00±0.00 | 36.30±32.09 | **1.00±0.01** | - | - | - | - |
| | $HDDDM_I$ | - | 1.00±0.00 | 1.00±0.00 | 10.20±5.69 | 1.00±0.00 | - | - | - | - | 1.00±0.00 | 1.00±0.00 | 38.50±16.28 | 1.00±0.00 | - | - | - | - |

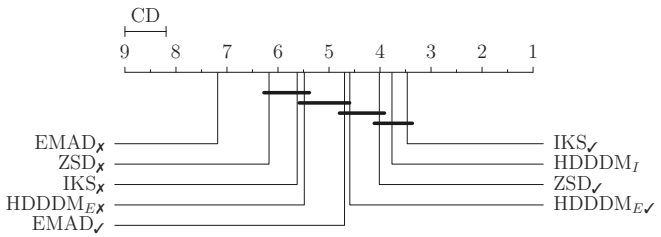


Fig. 4: Critical difference diagram based on the $H$-score accumulated over the datasets showing *real drift* only.

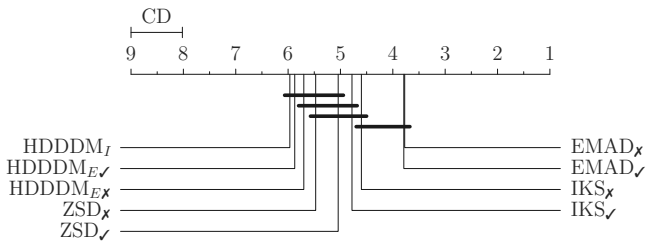


Fig. 5: Critical difference diagram based on the $H$-score accumulated over the datasets showing *virtual drift* only.

ZSD have shown performance comparable with other SotA algorithms. We thoroughly analyzed the dependence of the two introduced hyperparameters and identified a range of values for robust and good operation.

A drawback of the proposed framework is that it utilizes a $k$-means clustering in the embedding space with a single representative for each class and the Euclidean distances for calculating the distance statistics. This cannot capture multivariate or asymmetric class distributions well. Thus, we will investigate in future works an extension to incorporate more general characterizations of the clustering statistics.

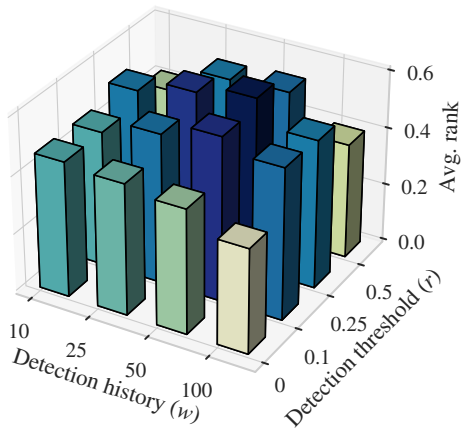


Fig. 6: Average performance rank when varying drift window size $w$ and drift detection threshold $r$ calculated for each dataset and algorithm combination (higher is better).

## References


[1] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys*, vol. 46, 2014.

[2] R. N. Gemaque, A. F. J. Costa, R. Giusti, and E. Santos, "An overview of unsupervised drift detection methods," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 10, 2020.

[3] H. Hu, M. Kantardzic, and T. S. Sethi, "No free lunch theorem for concept drift detection in streaming data classification: A review," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 10, 2020.

[4] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, 2019.

[5] I. Zliobaite, M. Pechenizkiy, and J. Gama, "An overview of concept drift applications," in *Big Data Analysis: New Algorithms for a New Society. Studies in Big Data*, 2016.

[6] A. Castellani, S. Schmitt, and S. Squartini, "Real-world anomaly detection by using digital twin systems and weakly supervised learning," *IEEE Transactions on Industrial Informatics*, vol. 17, 2021.

[7] A. Castellani, S. Schmitt, and B. Hammer, "Estimating the electrical power output of industrial devices with end-to-end time-series classification in the presence of label noise," *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2021.

[8] D. Reis, P. A. Flach, S. Matwin, and G. E. A. P. A. Batista, "Fast unsupervised online drift detection using incremental kolmogorov-smirnov test," *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[9] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," *Advances in Artificial Intelligence – SBIA*, 2004.

[10] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," *SIAM international conference on data mining*, 2007.

[11] G. Ditzler and R. Polikar, "Hellinger distance based drift detection for nonstationary environments," *IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments*, 2011.

[12] M. Jaworski, L. Rutkowski, and P. Angelov, "Concept drift detection using autoencoders in data streams processing," *International Conference on Artificial Intelligence and Soft Computing*, 2020.

[13] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "{CADE}: Detecting and explaining concept drift samples for security applications," *30th USENIX Security Symposium*, 2021.

[14] X. Li, H. Yin, K. Zhou, and X. Zhou, "Semi-supervised clustering with deep metric learning and graph embedding," *World Wide Web*, vol. 23, 2019.

[15] P. Dube and E. Farchi, "Automated detection of drift in deep learning based classifiers performance using network embeddings," *International Workshop on Engineering Dependable and Secure Machine Learning Systems*, 2020.

[16] S. Ackerman, P. Dube, and E. Farchi, "Sequential drift detection in deep learning classifiers," *arxiv:2007.16109*, 2020.

[17] T. S. Sethi and M. Kantardzic, "On the reliable detection of concept drift from streaming unlabeled data," *Expert Syst. Appl.*, vol. 82, 2017.

[18] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, 2011.

[19] T. S. Sethi, M. Kantardzic, and E. Arabmakki, "Monitoring classification blindspots to detect drifts from unlabeled data," *IEEE 17th International Conference on Information Reuse and Integration*, 2016.

[20] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, no. 1, 2006.

[21] S. Wilks, "Certain generalizations in the analysis of variance," *Biometrika*, vol. 24, 1932.

[22] T. W. Anderson, "An introduction to multivariate statistical analysis," Wiley New York, Tech. Rep., 1962.

[23] A. Castellani, S. Schmitt, and B. Hammer, "Supplementary material: Task-sensitive concept drift detector with constraint embedding," *arxiv:2108.06980*, 2021.