Disentangled Latent Space-based Drift Detection for Efficient Model Monitoring in Edge MLOps

Lara Luys[1], Mathias Verbeke[1]

[1]KU Leuven Bruges Campus, Spoorwegstraat 12, 8200 Sint-Michiels, Belgium

Abstract

Machine Learning Operations (MLOps) is a paradigm created to get machine learning (ML) algorithms out of the R&D environments and into production. A key principle of MLOps is monitoring, needed in ML projects where changes in the model environment can cause the performance to degrade. It is therefore important to detect these drifts as soon as possible, preferably without the use of difficult-to-obtain labels. This is done through unsupervised drift detection techniques. At the same time, the field of ML on resource-constrained edge devices has been gaining popularity for industrial use cases since edge ML does not require a stable internet connection, is more private and creates less latency. On these edge devices, however, there is a limited amount of memory and processing power. Although various unsupervised drift detection techniques exist, most are computationally expensive rendering them unusable on these resource-constrained devices. This paper proposes a novel drift detection technique that leverages the disentangled latent space of an autoencoder, the Latent Space Drift Detector (LSDD). LSDD contains three main phases. First, an offline training phase where the autoencoder is trained, creating a disentangled latent space. The decoder is removed and replaced with new classification task layers trained on the same dataset. The disentanglement creates clusters in the latent space for the different classes. The new model is deployed on the edge device in the offline drift detection phase where the centroids in the latent space are used to compare the samples coming from the input stream. A large distance between the current sample and the centroid implies drift. If drift is detected, the offline model adaptation phase is started where the sample is compared to previously known concepts. If the sample belongs to one of these concepts, the old model is redeployed, otherwise the task layers are retrained using data from the newly detected concept, updating the latent space centroids. In comparison to the Discriminative Drift Detector (D3), LSDD shows superior performance in terms of drift detection accuracy, memory usage and CPU performance.

Keywords

Concept drift detection, autoencoder, latent space, MLOps

1. Introduction

Recent advancements in machine learning (ML) have catalyzed its adoption across industrial sectors, where R&D divisions are systematically designing and developing ML models to enhance operational efficiency and product development. However, these ML projects often do not make it into production [1], [2]. Transitioning an ML model into production is a challenging step made difficult by aspects like poor communication, lack of versioning and pipelining, and the dynamic environments in which these ML models are deployed. To solve these problems, the concept of machine learning operations (MLOps) emerged [3, 4]. MLOps is a paradigm for developing and deploying ML applications into production in an automated and collaborative way. An important aspect of MLOps is dealing with the dynamic environments in which these models need to be integrated. When an ML model is trained, data with certain characteristics is often gathered from an R&D environment, based on a set of assumptions. However, when deploying the model, its environment can have different properties, or the properties can change over time. This can cause distribution changes in the input data of the ML model, called *data drift*, which in turn can cause the model performance to decrease since the learned concept is not relevant anymore, referred to as *concept drift* [5]. It is important to detect these drifts as soon as possible, such that the model can be adapted to its new environment. This is done through the use of drift detection techniques. The field of drift detection is extensive with the two main branches being

Workshop on Machine Learning Operations. October 25, 2025, Bologna, Italy

lara.luys@kuleuven.be (L. Luys); mathias.verbeke@kuleuven.be (M. Verbeke)

0009-0008-6470-9269 (L. Luys); 0000-0001-8297-6071 (M. Verbeke)

supervised and unsupervised techniques [5, 6, 7]. Supervised drift detection techniques use the true labels of the ML model to detect the drift through techniques using the error rate and accuracy of the ML model. However, this does not compare to real-life implementations where the creation of true labels is expensive and time-consuming, making them unavailable in most real-world implementations. A second main set of drift detection techniques leverages unsupervised detection strategies. These techniques implement statistical tests and ensemble methods to decide whether there is a drift in the data distribution. The main downside with these techniques is that they are computationally expensive making it difficult to detect drift in real-time situations [8].

At the same time, a lot of ML implementations require a model to be run on an embedded device at the network edge [9]. This can be motivated by the lack of a stable connection, e.g., for condition monitoring services that are running on industrial machinery in the field, data confidentiality as in production environments or healthcare systems, or latency that needs to be avoided in control applications. MLOps techniques including drift detection are highly relevant in these edge device settings. They are, however, limited through the constraints of the edge devices themselves. Unlike cloud applications, the available processing power and memory usage on the edge is constrained. Since most drift detection techniques are computationally expensive, these limitations cause a challenge. To address this, this paper introduces a novel drift detection technique, using clustering in the latent space of an autoencoder, called Latent Space Drift Detector (LSDD), which is computationally less expensive than other drift detection techniques whilst still producing a good drift detection result. An autoencoder is a neural network (NN) which first creates an embedded representation of the input data, called the latent space (LS), and using this embedding tries to reconstruct the input. The main idea of LSDD is to use the encoder of an autoencoder to create a smaller disentangled LS version of the input data which then uses a centroid-based clustering technique to detect drift. An LS is disentangled when each dimension in the LS vector has its own interpretation of the data independent from the other dimensions [10]. Ideally, these are also interpretable by humans. For example, when working with image data of an arrow, one dimension might change the arrow direction whilst the other changes its color. This type of LS will make it possible to more easily detect when the data distribution has changed. The contributions of this paper are as follows: (1) The introduction of a novel drift detection technique based on a disentangled autoencoder with dynamic updating, suitable for deployment on resource constraint edge devices called LSDD. (2) The introduction of a new drift detection dataset collected from a real-life motor bearing setup. (3) A thorough empirical evaluation, on this dataset in comparison with a state-of-the-art unsupervised drift detection technique in terms of drift detection accuracy, memory usage and processing power.

This paper is organized as follows. First, the background and related work are discussed in Sections 2 and 3. The method and experimental setup are discussed in Sections 4 and 5, after which we discuss the experimental results in Section 6. Section 7 concludes the paper and outlines directions for future work.

2. Background

2.1. Drift

Before discussing drift detection techniques, the different types of drift need to be understood. As mentioned before, there is data drift, where the distribution of the input data of an ML model has changed, and concept drift, where the true label distribution has changed in such a way that the learned ML model concept is not relevant anymore. In a lot of cases, data drift initiates concept drift. However, in some situations a change in the data does not cause a drift in concept which is called virtual drift. Another way to categorize drift is by its timing as shown in Figure 1 [11, 5]. First, there is sudden drift, which is an abrupt change in the data distribution where one concept is quickly replaced by the other. Second, there is gradual drift. Here the concept is slowly switched out over time. The second concept is first seen for short bursts and is then slowly seen for larger amounts of time until it has fully replaced the previous concept. Next, there is incremental drift where the first concept incrementally changes over time until it is an entirely new concept. Lastly, there is reoccurring drift. This means that

a certain concept can be replaced by another and then later reappear. This is linked to data seasonality where e.g. a situation in winter can be a certain concept and a situation in summer another one. In real-world situations all these different types of drift can occur, due to which it is important to create a drift detection algorithm which is able to deal with all these different types of drift.

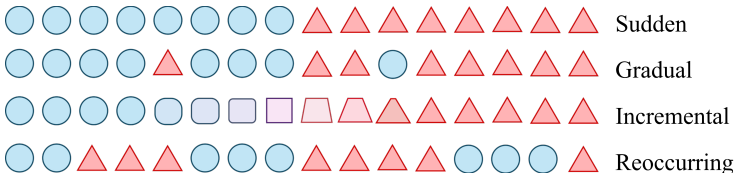

Figure 1: Different types of drift through time

2.2. Autoencoders

The type of autoencoder (AE) has a big effect on its ability to create good disentanglement in the LS. Three different types of AE are discussed to highlight this effect. The first one being the conventional AE. This NN tries to reconstruct its inputs using an encoder and a decoder, creating an embedded version of the input data in the LS. The reconstruction loss function consists of the mean absolute error between the predictions of the model and its inputs:

$$MAE - loss = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

with y_i being the input data and $\hat{y}_i$ being the predicted value.

The second type of AE is the β-variational AE (β-VAE) [12]. The VAE tries to fit the LS of the training data onto a Gaussian curve, which creates the possibility to generate new outputs from a random embedding in the LS. This is done by adding a variational loss to the reconstruction loss of the autoencoder. The β-VAE adds a small change to the variational loss function, a β hyperparameter to control its effect during training. A lower β gives more focus on the reconstruction of the data and a higher β makes the variational loss more important. This hyperparameter, when chosen right, creates a better disentanglement in the LS. However, it is known that finding the correct balance between these losses is a very difficult task to achieve.

$$loss = MAE + \beta * Loss_{Variational}$$

The third type of AE is the orthogonal AE (OAE) [13]. This type forces orthogonality onto the LS using the loss function. Next to the reconstruction loss, the orthogonal loss is added which forces the multiplication of the embedding vector z with its transformed version to be the unit matrix. This will also create disentanglement in the LS. Again, there is the possibility to control the effects of the reconstruction loss during training with a hyperparameter λ.

$$loss = MAE + \lambda * Loss_{Orthogonal}$$

In this paper the choice has been made to work with the OAE since it is easier to create a disentangled LS without difficult balancing tasks. This disentanglement is important to achieve since this creates the clear, separable clusters between different concepts which makes it possible to detect drift.

3. Related Work

3.1. Supervised drift detection

As mentioned, drift detection techniques can generally be split up into two separate branches. The first one being the supervised drift detection techniques which are well-researched and include some

well-known methods. The first one is the Drift Detection Method (DDM) [14]. This method uses the true labels of the data to calculate the error rate of the model. They then implement a hypothesis test to estimate the probability that the current error rate's mean differs from a true mean. Using this test two thresholds are defined, the warning level and the drift level. When both levels have been exceeded, drift is detected since the current error rate is significantly different from the previous one. DDM has been the basis for a lot of other drift detection algorithms. For example, Early Drift Detection Method (EDDM) [15], where the distance between the classification errors of two samples is used to detect drift. This method performs better at detecting gradual changes compared to DDM, but worse at sudden drift detection. Other variations of the DDM are the Hoeffding Drift Detection Method (HDDM), which uses Hoeffding's inequality to detect statistically significant changes, Fast Hoeffding Drift Detection Method (FHDDM), which is an improved version of HDDM, and Reactive Drift Detection Method (RDDM), which creates a buffer of data at the warning level to instantly retrain once the drift level is exceeded.[16, 17, 18]. Another well-known error-rate based drift detection method is adaptive windowing (ADWIN) [19]. It uses automatically resizable windows to capture the drift in the error rate. This is done using the means of these two windows. If the means are constant, the size of the window will stay bigger. When the difference becomes larger, the size will shrink, thus detecting drift more quickly. Drift is detected using a statistical test. There are also some algorithms that tried to improve on ADWIN, including the SEED drift detection algorithm [20] which uses the rate changes or volatility to detect drift. Although these techniques and other supervised techniques are well-researched and widely used, they assume that true labels are available shortly after the prediction of the sample. This does not work in most real-world use cases where the availability of true labels is limited.

3.2. Unsupervised drift detection

Most unsupervised techniques can be split up into four steps [7, 21]. The first step is to accumulate data. This includes the selection of a reference window and a detection window. The reference window can be fixed or sliding but is assumed to contain stable data of the known concept. The detection window, which can be batched or online, contains the possibly drifted data. When the detection window uses batched data it sometimes undergoes instance selection to review only a part of the window. The second step is the data modeling step where the windowed data is represented by a different, often smaller representation. This includes statistical calculations like the standard deviation used in UDetect [22], a dimensionality reduction technique as in the pre-clustering based technique of Wan and Wang [23], and the predictions of an ML model like the linear classifier used by the Discriminative Drift Detector (D3) [24]. The third step is the dissimilarity measure which is used to compare the two windows. A common choice is the use of distance measures and divergence metrics like the Frechét distance used in the DriftLens framework [8] and the Kolmogorov-Smirnov (KS) test used in the Incremental Kolmogorov-Smirnov (IKS) detection method [25]. Other dissimilarity measures include k-means clustering[23] and the log likelihood [26]. The fourth and final step is the drift criterion. This is the criterion used to decide drift. For example, the use of a threshold on the dissimilarity metric [27, 24] or the rejection or acceptance of a hypothesis test [23], among others. In some cases, steps are combined or removed all together like the online-based methods defined by the taxonomy of [7] where the dissimilarity measure is implemented directly onto the windows leaving out the data modeling step.

Although most drift detection techniques use statistical tests or classical ML methods, there are some that utilize NNs. An example is DNN+AE-DD [28], which trains a deep learning model on a reference dataset and an AE on the embeddings of the last hidden layer of this model. If the error rate of this AE is located outside of two thresholds, drift is detected. Fellicious et. al. use a Generative Adversarial Network (GAN) [29]. The GAN consists of a generator, which generates slightly out of distribution data, and a discriminator, which identifies the difference between the generated and original data, detecting drift. Since this process takes a long time, the authors switch to a NN approach after the first drift detection. The NN is trained on the original and drifted data to predict the probability of the input data being drift. The last specific NN-based approach is the model uncertainty unsupervised drift detector (MU-UDD) [30]. This detector was created to detect real-time drift in network traffic

classification. Using the model output probabilities they create a Dirichlet distribution and calculate its entropy, quantifying model uncertainty. They do this for a sliding window and calculate a weighted cumulative sum of the differences between the mean entropy and the entropy of that sample. If this sum is larger than a certain threshold, drift is detected.

We will limit our in depth discussion to techniques that focus more on the improvement of the realtime aspect of the drift detection methods. The first technique is the IKS drift detection technique by Reis et al. [25]. They first propose an incremental version of the normal KS test which also works a lot faster. It subsequently applies this new test for drift detection. They work with a stagnant reference window and a sliding detection window of fixed size. At each step they perform the IKS test on two samples, one from each of these two windows. Drift is detected when the null hypothesis of the IKS test is rejected. The results of their experiments show that the IKS performs a lot faster than the original KS test. Although the IKS performs a lot faster, it still performs a complex data distribution comparison at each time step, which can still be computationally expensive.

In 2021, Souza et. al. created an efficient unsupervised image based drift detector (IBDD) [31] which is able to detect drift in a high-dimensional fast datastream. They create an image based on the feature values from a certain window of data for both a static reference window and the current detection window. From this image they calculate the mean-squared deviation metric which shows the difference between these two images. Drift is detected when this metric is above or below calculated thresholds.

The next technique, discriminative drift detector (D3), avoids the need for these distribution comparisons by creating an ML model which tries to separate the detection window and the sliding reference window [24]. The authors train a Hoeffding Tree which is updated at each step in the data stream. The reference window gets a label 0 and the detection window is represented by label 1. The trained Hoeffding Tree classifier runs on these windows and looks at the area under the curve (AUC) to see how well the model separates the two measures. A small AUC implies a bad model separation and a small difference between two windows meaning no drift is detected. On the other hand, if the AUC is bigger than this threshold, the two distributions can be separated easily by the classifier, implying drift. If this is the case, the reference window is emptied and filled with new current data. When compared to EDDM, DDM and ADWIN, this drift detection method generally outperforms these techniques without the use of any data labels. However, the downside of this method is that the discriminative model needs to be updated at each step, which uses a lot of memory and processing power.

The method proposed by Greco S. et. al., called DriftLens, tries to speed up the drift detection process [8]. They use the embeddings created by the used ML model itself which are reduced through Principle Component Analysis (PCA). Next, they calculate the overall and per-label distribution for these embeddings and use the Frechét distance to detect whether the detection distributions differs from a reference distribution. If the distance is larger than a previously defined threshold, drift is detected. This method is compared to other statistical drift detection techniques, and is proven to be at least five times faster whilst creating generally better drift prediction results. However this technique only works with batched data to create the embedded distribution of the detection window. It is therefore still needed to save multiple batches of data which, on a smaller edge device, could cause a problem.

The last drift detection algorithm, introduced by Yamada T. and Matsutani H [32], is a lightweight drift detector which uses k-means clustering. A discriminative model first produces the output class labels of the data. This model comprises of an online sequential extreme machine learning model (OS-EML) for each class, trying to reconstruct the class input data. Then a k-means inspired drift detection algorithm is used on the input data itself. First, for each class a centroid is calculated on the training data. Then for each new sample the centroid corresponding with the discriminative model prediction is updated. When the distance between these two centroids changes significantly, drift is detected. This model performs well in terms of accuracy on the evaluated dataset but with a much smaller memory size. It also has a much larger detection delay in comparison to the other methods. Another aspect is that working with the original input data, does not scale well to higher-dimensional data where each feature needs to be saved on the edge device. Working with high-dimensional data can also introduce the curse of dimensionality where the calculation of distances become less effective in high-dimensional data. The proposed LSDD algorithm uses an AE to create a disentangled LS. This type of NN can be deployed

on the edge using a relatively small amount of memory and perform fast inference, compared to most unsupervised techniques. Using the LS also creates a smaller dimensionality which makes it possible to work with higher-dimensional input data. The technique also avoids complex data distribution comparisons and retraining of models at each sample, creating a low memory and fast model suitable for resource-constrained edge devices. Lastly, this technique is able to detect reoccurring drift, avoiding unnecessary retraining of previously known concepts.

4. Methodology

LSDD contains three phases. The first phase is the offline training phase. It is assumed that this phase runs on a machine with unlimited resources, like a server or the cloud. In this phase the AE and task model are trained, for which true classification labels of the task model should be available. The second phase is the online drift detection phase. Here the drift detection algorithm will detect drift on an online stream of data, assumed to run on a resource-constrained edge device with limited memory and processing power. In this step no labels are available. When drift is detected, the final phase is started, the offline model adaptation. In this step, the current sample is compared to the previous known concepts. If the sample belongs to one of these concepts, the corresponding concept is redeployed. Otherwise, data from the new concept is collected and send back to the server/cloud. There, the task layers are adapted to the new concept with a labeled version of this new data. The information in the LS is also updated with the new concept.

4.1. Offline training phase

The first step of the offline phase is to train an AE model on the training data. The goal is to create an encoder which has a disentangled LS. When this has been achieved, the decoder layers are removed and new task layers are added. These are then trained on the training data as well. Lastly, the centroids and variances of the different LS clusters are calculated.

In this version an OAE is used because this type of AE is consistently able to create a disentangled LS. When **training our AE** we use hyperparameter search to select a model with the best disentanglement. To do this the silhouette score of the training data classes in the LS is used as objective function. The silhouette score is most well known for its use in partitioning-based clustering algorithms to decide which hyperparameters result in the best clustering solution [33]. Its input is a clustering which contains data samples and their label. The silhouette score is used in this context to decide if an LS representation is disentangled. The idea is that if an LS is disentangled, the classes and concepts in that LS will be located in clear separable clusters. A new, drifted concept would be located in its own cluster and thus be easier to detect. The silhouette score of the known class labels in the LS is used as the objective function for the hyperparameter search. The encoder of the AE which has the best silhouette score will be further used. The silhouette score is calculated for each sample in a clustering using Equation:

$$Silhouette = \frac{(b-a)}{max(a,b)} \tag{1}$$

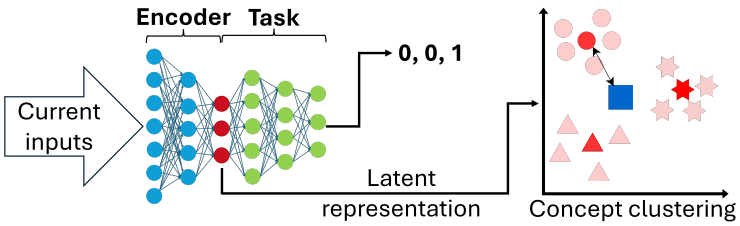

Figure 2: Overview of the drift detection method. The encoder and task model create an LS and a class label respectively. In the LS the new sample (blue square) is compared to the centroid of the output class cluster (dark red circle). The other classes of the same concept are also shown (red triangles and stars).

where a is the distance from that sample to the mean of its cluster and b is the distance to the mean of the nearest cluster of which the sample is not a part. If the silhouette score of the value is close to 1, the sample is located close to its own cluster centre and far away from any other clusters. This is the desired result. If the value is close to -1, the sample is located closer to another cluster than to the one it is assigned to, and if the value is close to 0, the distances to both its own cluster and the closest different one are similar meaning that there is overlap in the clusters themselves. Both those options should be avoided. To know if the whole clustering was done well, the mean of the silhouette score for each sample is used. This mean should be as close to 1 as possible, meaning that all clusters are located far away from each other and all samples in a cluster are located close to the centre of their cluster. We therefore want to maximize this silhouette score during our hyperparameter search.

Starting from the selected AE, the **decoder is removed**. Then, new NN layers are added to the encoder. The number of layers depends on the difficulty of the task and the size of the edge device. These layers are trained on the labeled training data to perform a classification task. Only the task layers are updated. The encoder itself is not changed meaning that the task inputs are the embedded representations of the input data. If the found models would happen to be too large to run on the wanted edge device, it is possible to use techniques like quantization to reduce the model memory size.

The last step is to **calculate the means and variances** of the embeddings for each class label of the training data. For this inference is done on the encoder with the training data, saving the resulting embedding values. With a 2-dimensional embedding space this results in four values for each class: two for the 2D-mean representation and two for the 2D-variance representation. These values are stored on the edge device where the encoder and task layers are deployed. This finishes the setup which can be deployed on the edge device. The concept is visually represented in Figure 2

4.2. Online drift detection phase

Algorithm 1 Drift detection algorithm for 2D embedding space

$z \leftarrow Encoder(x)$
$y \leftarrow Task(z)$
$d(z_0, c_{y0}) \leftarrow |z_0 - c_{y0}|$
$d(z_1, c_{y1}) \leftarrow |z_1 - c_{y1}|$
if $d(z_0, c_{y0}) \geq v_{y0}$ or $d(z_1, c_{y1}) \geq v_{y1}$ **then** ▷ Check if current sample is located in current cluster
    $Drift \leftarrow True$
else
    $c_y \leftarrow \frac{c_y + z}{2}$
    $d(c_{y0}, c_{y,old_0}) \leftarrow |c_{y0} - c_{y,old_0}|$
    $d(c_{y1}, c_{y,old_1}) \leftarrow |c_{y1} - c_{y,old_1}|$
    if $d(c_{y0}, c_{y,old_0}) \geq v_{y0}$ or $d(c_{y1}, c_{y,old_1}) \geq v_{y1}$ **then** ▷ Check if current cluster has drifted
        $c_y \leftarrow c_{y,old}$
        $Drift \leftarrow True$
    else
        $Drift \leftarrow False$
    end if
end if

In the online phase, the encoder and task layers are located on a resource-constrained edge device. First an algorithm will decide if a sample has drifted. To make sure that true drift is detected and the algorithm has not just detected an anomaly, an extra check is done. The **drift detection algorithm**, shown in Algorithm 1, starts with the arrival of a new input sample x containing a certain number of input features. The encoder transforms the input into its disentangled representation. For simplicity, the LS is two-dimensional, although this algorithm also works for higher-dimensional representations. This representation is used as input for the task layers which produce a label. The embedding values

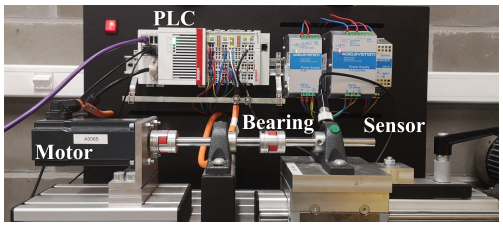

Figure 3: Motor bearing setup used for the collection of the motor bearing dataset.

are compared to the centroid of the class corresponding to this label. This is done for both dimensions. When this distance is greater than the cluster variance for that dimension, drift is detected. Otherwise, the mean of the cluster is recalculated using the new embedding value. If this is the case, the distances per dimension between the new centroid and the original one are calculated. If this distance is greater than the variance, drift is detected as well. When all distances are smaller than their respective variances, the new mean is used as the centroid for new samples and the old mean is stored to use for future comparisons of the centroids themselves. The first comparison is able to detect sudden drift and the second comparison is able to detect gradual drift. When gradual drift occurs, the centroid will slightly move with each recalculation, which would go unnoticed by the first comparison. Comparing the new centroid to the original one makes sure that this drift is detected as well.

Sometimes noise in the input data can cause the model to misbehave for a single sample, called an anomaly or outlier. To be able to **differ between outliers and real drift detection**, a boolean vector of the last N samples is kept. Each value represents whether drift was detected (True) or not (False). If the percentage of True values in this vector is larger than a threshold, γ, true drift has been detected. Both N and γ are user-defined parameters, deciding the sensitivity of the drift detection technique.

4.3. Offline model adaptation phase

The first time true drift has been detected, the retraining process will start. This process starts by collecting data from the new concept. When a user defined window of data has been collected, it is sent to the server or cloud to be labeled and used to retrain the task layers. The embedded values of this new data are also created to calculate the mean and variance of the new concept cluster. These are stored next to the other centroids and variances of the first concept. This new cluster is assumed to be the new current concept and its mean and variance are used for comparison until the moment true drift is detected again. When more than a single concept is known, a new check will be added once drift is detected. The most recent sample will be compared to the other known means and variances, using the prediction of the input sample by the old model. If, in one of those comparisons, the distance to the centroid is smaller than the variance, it belongs to this known concept which means that reoccurring drift is happening. Depending on the size of the edge device this check can be done on the edge device itself or on the cloud. If it is possible to store the new task layers on the edge device itself, the full check can be done on the edge and the used model can simply be switched out without a cloud connection. If the task models are too large, they are stored on the cloud where this check is done as well. If the concept is known, the model corresponding to that concept is redeployed on the edge.

5. Experimental setup

LSDD is compared to the D3 technique as it was one of the best performers in the comparison by Lukats et. al. [5]. The two techniques will be compared in terms of drift detection effectiveness and resource usage on a real-world dataset.

5.1. Motor bearing dataset

A motor-bearing setup, seen in Figure 3, is used for the creation of a new condition monitoring dataset, representing a real life common use case of ML on the edge. In the situation of condition monitoring it is important to detect errors as soon as possible, without a large latency. It is therefore ideal to run this model and drift detection method locally on the motor setup instead of sending data to the cloud and back, making it a relevant use case for drift detection on the edge. This setup contains a motor controlled by a Beckhoff PLC. Connected to the motor is a bearing which can be switched out between three different types of bearings; an ideal bearing, a damaged bearing, and a bearing which is misaligned. There is also a vibration sensor connected to the bearing which is able to collect the vibration data when the motor is running. The task of this setup is to detect the type of bearing running on the motor through the vibration data. To simulate drift, the speed of the motor can be changed running from 10 revolutions/second to 30 revolutions/second. This gives us two possible concepts. For each type of bearing, data is collected for 10 minutes with a sampling rate of 10 kHz. This gives a dataset containing 6.000.000 data samples for each class in a concept. For the damaged bearing at 30 rev/s and the misaligned bearing at 10 rev/s the data collecting process was cut short creating a dataset of 3.589.500 and 4.054.000 samples, making the training classes slightly imbalanced in the misaligned case. Before training the AE, some data preprocessing steps are executed. To create a slightly smaller dataset, the mean of every non-overlapping 100 samples is taken. This value is then normalized using min-max normalization. The dataset is then split into a training, validation and test set with a 0.95, 0.025 and 0.025 ratio. Because of the fact that once drift is detected, data is collected to retrain the model, it is necessary to have all classes represented in that collected data, to make sure the model learns a representation of all three classes. Therefore, for each concept, the data are shuffled. The two concepts are then split and put behind each other to create sudden drift between the two concepts. The data starts with the first concept of 10 rev/s then switches to 30 rev/s, back to 10 rev/s and then back again. This creates three moments of sudden drift of which two are reoccurring drift.

5.2. Metrics

There are three main aspects to compare between LSDD and the state of the art. These are the effectiveness of the drift detection method, the memory usage and the processing power used by the method. The metrics used during these experiments are discussed in the following sections.

5.2.1. Algorithm effectiveness

To compare the accuracies, three different metrics are used. The first metric is the Missed Detection Rate (MDR) [21]. A concept is missed if the algorithm did not detect drift between the occurrence of the drift and the next one. A lower MDR implies a better drift detection. The second metric is the Mean Time To Detection (MTD). It shows the time it took between the occurrence of drift and the detection of it. A lower MTD means that the algorithm works faster. The last metric is the number of False Detections (FD). This number should be as small as possible, avoiding false alarms and unnecessary retraining. The best performing model is one with zero FD, detecting all drifts and as fast as possible.

5.2.2. Resources

Our method is created for its implementation on an edge device. It is therefore crucial that the method limits the use of processing power. The amount of processing power needed by the algorithm is often measured by the algorithm complexity, represented by the time needed to run the algorithm. Similar to [8] we are using the mean running time to detect drift on a single sample. The device on which these experiments are run is a laptop with a 13th Gen Intel® Core™ i7 processor and 32Gb of RAM. Another aspect to compare is the memory usage of the algorithm. Using the tracemalloc Python library, the peak memory is measured for monitoring the full dataset for drift without retraining the algorithm.

Table 1
Drift detection results of LSDD and D3 methods on the motor dataset

Algorithm	Parameters	MTD	MDR	FD
LSDD	N=13, γ=0.077, k=73	2.666	0	0
LSDD	N=10, γ=0.05, k=74	2.666	0	0
D3	w=82, ρ=0.544, threshold=0.6525, k=47	8.5	0.333	0

This way only the drift detection algorithm itself is measured. This is done 35 times for each algorithm from which the results are used to compare the algorithms.

6. Results and discussion

6.1. Drift Detection Accuracy

Both techniques, LSDD and D3, which is used as a benchmark, are first optimized on the dataset. This is done using multi-objective optimization in Optuna [34]. The metrics to minimize are the MTD, MDR and FD. The TPE sampler is used for optimization and Optuna runs for 300 trials. For D3, the reference window length, w, detection window length, ρ and threshold are used as hyperparameters. For LSDD, N and γ are the two main hyperparameters. In both cases the number of samples gathered for retraining, k, is also a hyperparameter since the detection method does not run while it is collecting data. The results given by the multi-objective optimization search are the pareto front. From those results, the ones with the lowest MDR are chosen and of those, the ones with the lowest FD are retained. This gives us two results for the LSDD technique and one for the D3 technique with the motor bearing dataset. In Table 1, it is shown that LSDD has a lower MTD and thus is able to detect drift faster than the D3 algorithm. It is also able to detect all three drifts in the dataset whilst D3 is only able to detect two out of the three. This is because the last two drifts were located close to each other. Because the LSDD technique can detect reoccurring drift and switch back to a previous model, it does not need to collect data and can instantly start evaluating new samples. The D3 algorithm does not make a distinction and thus always collects data. Therefore it takes longer to switch to a new model and by the time D3 can detect the last drift, the input stream has stopped. LSDD is thus able to detect drift as well as the state-of-the art technique for sudden drift, and can easily identify reoccurring drift which the D3 technique cannot, making the model adaptation process faster in those cases.

6.2. Resource usage

To measure the memory usage, the best performing parameters from Section 6.1 are used. Since the retraining steps are assumed to run on the cloud, they are omitted from the algorithm during these measurements. Only the model inference and the drift detection itself are measured. The algorithms are run on the dataset 35 times collecting the peak memory from the algorithm. In Figure 4, the boxplots of these peak memories are shown. The LSDD algorithm clearly has a smaller memory footprint using on average only 224Kb. Thus making it more suitable for resource-constrained edge devices.

The processing power is measured by the mean time needed to run the drift detection step of the algorithm. This excludes model inference and retraining from the measurements. Both algorithms are run on the motor bearing dataset for 35 times, this time collecting the mean time of the drift detection step. The boxplots of these means can be seen in Figure 4. These plots show that the LSDD is able to perform the drift detection a lot faster than the D3 algorithm, being on average 23 times faster.

7. Conclusion

In this paper the LSDD algorithm was introduced as a novel unsupervised drift detection algorithm for resource-constrained devices. It leverages the disentangled latent space of an OAE to create a clustered

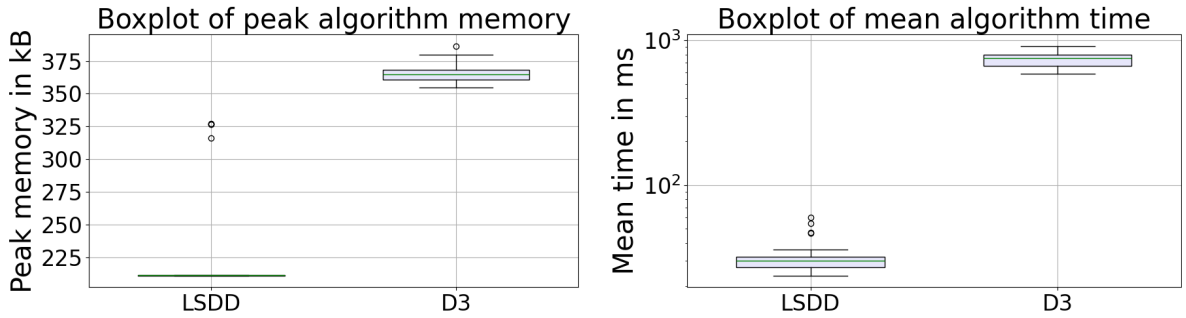

Figure 4: Boxplots of the the peak algorithm memory usage (left) and mean algorithm time (right) for both LSDD and D3, showing a lower memory usage and faster time for LSDD.

representation of the data. The experimental evaluation shows that LSDD was able to detect drift on a real-life dataset with better results compared to D3, a state-of-the-art drift detection technique. The algorithm also has a smaller memory and time footprint rendering it more effective for resource-constrained edge devices. Future work includes a comparison of the proposed algorithm on different, more complex datasets and to a broader selection of state-of-the art techniques for a more in-depth analysis. This analysis will also be done on different types of edge devices, to evaluate the resource usage in more depth. Furthermore, the algorithm will be enhanced with quantization on the encoder and task layers to create an algorithm that can run on even smaller edge devices like microcontrollers, and explore the effects on the drift detection technique. Lastly, the LSDD algorithm will be implemented in a full MLOps framework which also includes retraining and versioning for continual learning, effectively using the advantages of drift detection on the edge.

Acknowledgments

This research is supported by Flanders Innovation & Entrepreneurship (VLAIO) through the TETRA project MLOps4ECM (HBC.2023.0063), and by Internal Funds KU Leuven (STG/21/057).

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] M. Loukides, AI adoption in the ensterprise 2022, Technical Report, O'Reilly, 2022. URL: https://www.oreilly.com/radar/ai-adoption-in-the-enterprise-2022/.

[2] S. Vohra, A. Vasal, P. Roussiere, P. Tanguturi, L. Guan, The art of AI maturity: advancing from practice to performance, Technical Report, Accenture, 2021. URL: https://www.accenture.com/us-en/insights/artificial-intelligence/ai-maturity-and-transformation?c=acn_glb_aimaturityfrompmediarelations_13124019&n=mrl_0622.

[3] M. Testi, M. Ballabio, E. Frontoni, G. Iannello, S. Moccia, P. Soda, G. Vessio, MLOps: A Taxonomy and a Methodology, IEEE Access 10 (2022) 63606–63618. doi:10.1109/ACCESS.2022.3181730, publisher: IEEE.

[4] D. Kreuzberger, N. Kuhl, S. Hirschl, Machine Learning Operations (MLOps): Overview, Definition, and Architecture, IEEE Access 11 (2023) 31866–31879. doi:10.1109/ACCESS.2023.3262138, arXiv: 2205.02302 Publisher: Institute of Electrical and Electronics Engineers Inc.

[5] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, G. Zhang, Learning under Concept Drift: A Review, IEEE Transactions on Knowledge and Data Engineering (2018) 1–1. URL: https://ieeexplore.ieee.org/document/8496795/. doi:10.1109/TKDE.2018.2876857.

[6] S. Agrahari, A. K. Singh, Concept Drift Detection in Data Stream Mining : A literature review, Journal of King Saud University - Computer and Information Sciences 34 (2022) 9523–9540. URL: https://linkinghub.elsevier.com/retrieve/pii/S1319157821003062. doi:10.1016/j.jksuci.2021.11.006.

[7] R. N. Gemaque, A. F. J. Costa, R. Giusti, E. M. Dos Santos, An overview of unsupervised drift detection methods, WIREs Data Mining and Knowledge Discovery 10 (2020) e1381. URL: https://wires.onlinelibrary.wiley.com/doi/10.1002/widm.1381. doi:10.1002/widm.1381.

[8] S. Greco, B. Vacchetti, D. Apiletti, T. Cerquitelli, Unsupervised Concept Drift Detection from Deep Learning Representations in Real-time, 2024. URL: http://arxiv.org/abs/2406.17813. doi:10.48550/arXiv.2406.17813, arXiv:2406.17813 [cs].

[9] M. G. Sarwar Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, F. Hussain, Machine Learning at the Network Edge: A Survey, ACM Computing Surveys 54 (2022). doi:10.1145/3469029, arXiv: 1908.00080 Publisher: Association for Computing Machinery.

[10] J. Cha, J. Thiyagalingam, Orthogonality-enforced latent space in autoencoders: An approach to learning disentangled representations, in: A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, J. Scarlett (Eds.), Proceedings of the 40th International Conference on Machine Learning, volume 202 of *Proceedings of Machine Learning Research*, PMLR, 2023, pp. 3913–3948. URL: https://proceedings.mlr.press/v202/cha23b.html.

[11] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Computing Surveys 46 (2014) 1–37. URL: https://dl.acm.org/doi/10.1145/2523813. doi:10.1145/2523813.

[12] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, A. Lerchner, beta-VAE: learning basic visual concepts with a constrained variational framework (2017).

[13] W. Wang, D. Yang, F. Chen, Y. Pang, S. Huang, Y. Ge, Clustering With Orthogonal AutoEncoder, IEEE Access 7 (2019) 62421–62432. URL: https://ieeexplore.ieee.org/document/8712494/. doi:10.1109/ACCESS.2019.2916030.

[14] J. Gama, P. Medas, G. Castillo, P. Rodrigues, Learning with Drift Detection, in: D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, A. L. C. Bazzan, S. Labidi (Eds.), Advances in Artificial Intelligence – SBIA 2004, volume 3171, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 286–295. URL: http://link.springer.com/10.1007/978-3-540-28645-5_29. doi:10.1007/978-3-540-28645-5_29, series Title: Lecture Notes in Computer Science.

[15] M. Baena-García, J. Campo-Ávila, R. Fidalgo-Merino, A. Bifet, R. Gavald, R. Morales-Bueno, Early Drift Detection Method (2006).

[16] I. Frias-Blanco, J. D. Campo-Avila, G. Ramos-Jimenez, R. Morales-Bueno, A. Ortiz-Diaz, Y. Caballero-Mota, Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds, IEEE Transactions on Knowledge and Data Engineering 27 (2015) 810–823. URL: http://ieeexplore.ieee.org/document/6871418/. doi:10.1109/TKDE.2014.2345382.

[17] P. Frasconi, N. Landwehr, G. Manco, J. Vreeken (Eds.), Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part II, volume 9852 of *Lecture Notes in Computer Science*, Springer International Publishing, Cham, 2016. URL: https://link.springer.com/10.1007/978-3-319-46227-1. doi:10.1007/978-3-319-46227-1.

[18] R. S. Barros, D. R. Cabral, P. M. Gonçalves, S. G. Santos, RDDM: Reactive drift detection method, Expert Systems with Applications 90 (2017) 344–355. URL: https://linkinghub.elsevier.com/retrieve/pii/S0957417417305614. doi:10.1016/j.eswa.2017.08.023.

[19] A. Bifet, Adaptive learning and mining for data streams and frequent patterns, SIGKDD Explor. Newsl. 11 (2009) 55–56. URL: https://doi.org/10.1145/1656274.1656287. doi:10.1145/1656274.1656287.

[20] D. T. J. Huang, Y. S. Koh, G. Dobbie, R. Pears, Detecting Volatility Shift in Data Streams, in: 2014 IEEE International Conference on Data Mining, IEEE, Shenzhen, China, 2014, pp. 863–868. URL: http://ieeexplore.ieee.org/document/7023414/. doi:10.1109/ICDM.2014.50.

[21] D. Lukats, O. Zielinski, A. Hahn, F. Stahl, A benchmark and survey of fully unsupervised concept drift detectors on real-world data streams, International Journal of Data Science and Analytics 19 (2025) 1–31. URL: https://link.springer.com/10.1007/s41060-024-00620-y. doi:10.1007/s41060-024-00620-y.

[22] S. A. Bashir, A. Petrovski, D. Doolan, A framework for unsupervised change detection in activity recognition, International Journal of Pervasive Computing and Communications 13 (2017) 157–175. URL: https://www.emerald.com/insight/content/doi/10.1108/IJPCC-03-2017-0027/full/html. doi:10.1108/IJPCC-03-2017-0027.

[23] J. S.-W. Wan, S.-D. Wang, Concept Drift Detection Based on Pre-Clustering and Statistical Testing, Journal of Internet Technology Volume 22 (2021) No.2 (2021).

[24] O. Gözüaçik, A. Büyükçakir, H. Bonab, F. Can, Unsupervised Concept Drift Detection with a Discriminative Classifier, in: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, ACM, Beijing China, 2019, pp. 2365–2368. URL: https://dl.acm.org/doi/10.1145/3357384.3358144. doi:10.1145/3357384.3358144.

[25] D. M. Dos Reis, P. Flach, S. Matwin, G. Batista, Fast Unsupervised Online Drift Detection Using Incremental Kolmogorov-Smirnov Test, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, San Francisco California USA, 2016, pp. 1545–1554. URL: https://dl.acm.org/doi/10.1145/2939672.2939836. doi:10.1145/2939672.2939836.

[26] L. I. Kuncheva, Change Detection in Streaming Multivariate Data Using Likelihood Detectors, IEEE Transactions on Knowledge and Data Engineering 25 (2013) 1175–1180. URL: http://ieeexplore.ieee.org/document/6060824/. doi:10.1109/TKDE.2011.226.

[27] O. Gözüaçik, F. Can, Concept learning using one-class classifiers for implicit drift detection in evolving data streams, Artificial Intelligence Review 54 (2021) 3725–3747. URL: https://link.springer.com/10.1007/s10462-020-09939-x. doi:10.1007/s10462-020-09939-x.

[28] L. Hu, Y. Lu, Y. Feng, Concept Drift Detection Based on Deep Neural Networks and Autoencoders, Applied Sciences 15 (2025) 3056. URL: https://www.mdpi.com/2076-3417/15/6/3056. doi:10.3390/app15063056.

[29] C. Fellicious, L. Wendlinger, M. Granitzer, Neural Network Based Drift Detection, in: G. Nicosia, V. Ojha, E. La Malfa, G. La Malfa, P. Pardalos, G. Di Fatta, G. Giuffrida, R. Umeton (Eds.), Machine Learning, Optimization, and Data Science, volume 13810, Springer Nature Switzerland, Cham, 2023, pp. 370–383. URL: https://link.springer.com/10.1007/978-3-031-25599-1_28. doi:10.1007/978-3-031-25599-1_28, series Title: Lecture Notes in Computer Science.

[30] M. Liu, P. Wang, Y. Ye, X. Chen, Model Uncertainty Based Unsupervised Real-Time Drift Detection in Network Traffic Classification, in: 2024 IEEE Cyber Science and Technology Congress (CyberSciTech), IEEE, Boracay Island, Philippines, 2024, pp. 82–87. URL: https://ieeexplore.ieee.org/document/10795723/. doi:10.1109/CyberSciTech64112.2024.00023.

[31] V. M. A. Souza, A. R. S. Parmezan, F. A. Chowdhury, A. Mueen, Efficient unsupervised drift detector for fast and high-dimensional data streams, Knowledge and Information Systems 63 (2021) 1497–1527. URL: https://link.springer.com/10.1007/s10115-021-01564-6. doi:10.1007/s10115-021-01564-6.

[32] T. Yamada, H. Matsutani, A Lightweight Concept Drift Detection Method for On-Device Learning on Resource-Limited Edge Devices, in: 2023 IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2023, Institute of Electrical and Electronics Engineers Inc., 2023, pp. 761–768. doi:10.1109/IPDPSW59300.2023.00128.

[33] K. R. Shahapure, C. Nicholas, Cluster Quality Analysis Using Silhouette Score, in: 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA), IEEE, sydney, Australia, 2020, pp. 747–748. URL: https://ieeexplore.ieee.org/document/9260048/. doi:10.1109/DSAA49011.2020.00096.

[34] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, 2019. URL: https://arxiv.org/abs/1907.10902. arXiv:1907.10902.