DriftGAN: Using historical data for Unsupervised Recurring Drift Detection

Christofer Fellicious
University of Passau
Passau, Bavaria, Germany
christofer.fellicious@uni-passau.de

Sahib Julka
University of Passau
Passau, Bavaria, Germany
sahib.julka@uni-passau.de

Lorenz Wendlinger
University of Passau
Passau, Bavaria, Germany
sahib.julka@uni-passau.de

Michael Granitzer
University of Passau
Passau, Bavaria, Germany
michael.granitzer@uni-passau.de

ABSTRACT

In real-world applications, input data distributions are rarely static over a period of time, a phenomenon known as concept drift. Such concept drifts degrade the model's prediction performance, and therefore we require methods to overcome these issues. The initial step is to identify concept drifts and have a training method in place to recover the model's performance. Most concept drift detection methods work on detecting concept drifts and signalling the requirement to retrain the model. However, in real-world cases, there could be concept drifts that recur over a period of time. In this paper, we present an unsupervised method based on Generative Adversarial Networks(GAN) to detect concept drifts and identify whether a specific concept drift occurred in the past. Our method reduces the time and data the model requires to get up to speed for recurring drifts. Our key results indicate that our proposed model can outperform the current state-of-the-art models in most datasets. We also test our method on a real-world use case from astrophysics, where we detect the bow shock and magnetopause crossings with better results than the existing methods in the domain.

CCS CONCEPTS

• **Computing methodologies** → *Feature selection*; **Anomaly detection**; **Neural networks**; Active learning settings.

KEYWORDS

Drift Detection, Unsupervised Drift Detection, Generative Adversarial Networks, Recurring Drift

1 INTRODUCTION

Machine Learning is ubiquitous in our lives and has the potential to revolutionize many industries by increasing efficiency, personalizing experiences, predicting outcomes, automating tasks, and improving decision-making. Machine learning often predicts different aspects of our lives, such as the prediction of energy consumption [22] or the fuel consumption of trucks [20]. For applications with continuous stream of data that span months or years, there is a good chance for the input data distribution to change over time. For example, in credit card fraud detection, customers' buying behavior changes over time [8]. Such changes in the input data distribution are known as *concept drifts* and adversely affect the model's prediction performance. To counter such performance degradation, we need mechanisms to identify the changes in the input data distribution and retrain the prediction model so that the prediction model adapts the changed input data distribution and improves the performance. In the most straightforward strategy, we assume changes happen regularly and train the model periodically. However, this naive strategy comes with a couple of significant disadvantages. First, model retraining is usually very costly in terms of time for data collection and computation, and it is even harder to determine the perfect retraining interval. Secondly, changes in distribution may not coincide with the periodic model retraining schedule, thereby rendering model update efforts inefficient and ineffective. Therefore, we need a mechanism to detect the changes in the underlying data distribution and retrain the model for optimal performance and utilization of computing resources [7].

However, detecting changes in data distributions and identifying previous data distributions hold the potential to improve classifiers, which is still relatively unexplored. In particular, there could be patterns that reappear later. For example, suppose we have a method to identify such past data distributions. In that case, identifying recurring changes in the distribution will help us bolster the training set and get the model up to speed quicker rather than waiting for the required number of instances from the real-world data stream. Currently, most concept drift algorithms only focus on identifying the changes in the distribution, but do not consider whether a similar data distribution previously occurred. Examples of repeating distributions are seasonal repeating customer shopping behavior, energy consumption, or changes in credit card fraud patterns.

In standard methods, we need to wait until enough input is acquired to train the model after every new concept drift. Furthermore,

this data also needs to be labelled, which increases the cost and energy expenditure. Contrary to standard methods, our framework keeps track of the previously seen data distributions, which allows us to augment the retraining dataset with the previously seen data of the same distribution. In real-world cases, this augmentation helps reduce the time required to acquire enough new data on the new distribution, thereby reducing the model's turnaround time. As a result, we also increase the performance of the model when compared to retraining the model from scratch with limited data. This time to obtain the labels and retrain the model is significant when labeled data is either sparse or only available with a significant time delay [4].

In this work, we extend drift detection by keeping track of all data distributions seen by the model. We show that using the data from the past concept drifts for retraining the model allows us to outperform the current state of the art. Furthermore, this memory of previous data distributions allows the model to adapt and reduce the turnaround time to previously identified concept drifts. Methodologically, we develop a multiclass-discriminator drift Generative Adversarial Network (DriftGAN). The DriftGAN contains a discriminator module that does not only distinguish between real and artificial examples but also discriminates to which previous drift the current example belongs. Since drifts can increase over time, the discriminator network can grow incrementally by adding more discriminator classes whenever we identify new data distributions. We evaluate our DriftGAN on eight unsupervised real-world drift detection datasets and compare it with existing unsupervised methods. Unsupervised drift detection does not rely on the performance of the label prediction model to detect drift - which is not readily available in most tasks but only based on changes in the input distribution. The results show that our method has a better performance when compared to other unsupervised methods that only detect whether drifts occur or not. We also adapted our method as a sampling strategy for the magnetometer data of the NASA MESSENGER spacecraft. Our results for the real-world application scenario also produce better results than the existing methods, which shows that our method can be adapted to different domains with very little to no changes.

We structure the rest of the paper as follows, section 2 describes the related work with supervised and unsupervised concept drift detection, section 3 discusses our approach to the problem with the necessary explanation on the identification of previously seen data distributions and the research gap we address. We apply our method on the magnetometer dataset from the NASA MESSENGER spacecraft which is explained in section 5. Then, section 6 describes our experimental results and the interpretation of the results. Finally, section 7 concludes our paper.

2 RELATED WORK

We can broadly classify the concept drift detection methods into Supervised and Unsupervised methods. Supervised methods have access to the input feature vector's label, making it easier to understand and detect data distribution changes. The problem becomes compounded with unsupervised learning methods as the data labels are unavailable to the model to decide whether the distribution has changed. Unsupervised learning methods are more useful in

real-world scenarios as data labels might not be readily available to the model and may require time or expertise to obtain.

2.1 Supervised Methods

The Drift Detection Method (DDM) is one of the earliest supervised methods in concept drift detection [7]. The method uses the error rate to determine whether a concept drift occurred. The idea behind this method is that the error rate increases when there is a change in input data distribution. The algorithm defines two thresholds based on the error rate. The first threshold is also lower, serving as a drift warning. The second threshold signals the drift. The retraining occurs in the window that starts from the drift warning.

The Adaptive Windowing (ADWIN) method uses different-sized windows to detect concept drifts [2]. ADWIN uses statistical tests on different windows and detects concept drifts based on the error rates, similar to DDM. Cabral and Barros proposed a method using Fisher's Exact Test [5]. This method analyzed two separate context windows and the assumption that if there was no drift, the error distribution should be equal between the two windows. They use the generalized contingency table from Fisher's Exact test to test the null hypothesis.

2.2 Unsupervised Methods

Identifying concept drifts without the corresponding labels to the input feature vectors is comparatively more difficult. We can only rely on the input feature vectors to decide whether a concept drift occurs. Additionally, unsupervised concept drift detection cannot detect certain types of concept drifts such as changes in the drift label while the input data distribution is more or less constant. In such scenarios, it is almost impossible to identify a concept drift without the class labels. Hu et al. explain why detecting concept drifts while maintaining high performance is not trivial [13]. The paper explains high performance with labeled data is possible, but labeling data streams as mentioned earlier could be expensive and time-consuming. These drawbacks of supervised drift detection mean that even with all the drawbacks, unsupervised drift detection is more efficient in real-world scenarios to detect concept drifts.

The Kolmogorov-Smirnov is a nonparamtetric statistical test can determine whether two data points belong to the same distribution. The problem with the K-S test is its time complexity that grows with the number of samples. Reis et al. proposed a faster alternative to the K-S test using an incremental K-S test [6]. This implementation uses a custom tree data structure called "*treap*." The algorithm uses two windows, where the first is fixed and the second is sliding. The first window contains the original concepts the model was trained on, while the second one slides and checks whether a concept drift occurred. While the K-S test remains popular for univariate data and an expansion of the K-S test for multivariate data was proposed by Justel et al. [16].

Discriminative Drift Detector is an unsupervised method for detecting concept drifts using a linear regressor on two windows [10]. The basic idea is that there are two windows; the first is assumed to be from the old data distribution and given label **0**. A second window, a sequence of input data, is chosen and labeled as **1**. If the linear regressor can discriminate between both windows with a certain precision means that the two windows are different, and

thus a concept drift has occurred. The performance is measured using a secondary supervised Hoeffding Tree classifier. One Class Drift Detection(OCDD) is a method that uses a linear SVM and sliding windows to detect outliers in the incoming data stream [11]. When the outliers reach a threshold, the algorithm signals a concept drift. Currently, OCDD has the best performance in unsupervised drift detection.

Another method, ODIN, uses autoencoders to identify drifts and select trained classifiers based on the drift [26]. ODIN consists of four components; an encoder, a decoder, a latent discriminator and an image discriminator. Autoencoders generate latent representations from input feature vectors, and a decoder generates the initial feature vectors. The latent representation is passed to the latent discriminator for detecting inlier frames from outlier frames. The loss function is computed as the weighted sum of the loss of the latent discriminator loss, the image discriminator, and the standard reconstruction loss.

Methods like the K-S test are unsuitable for data with many features, as checking the data distribution across each feature is not feasible. All methods in the concept drift domain are adept at identifying supervised and unsupervised drifts. However, they do not consider the possibility of recurring drifts and use the historical data from previous drifts to retrain the model. Reusing previous data from the same distributions that occurred in the past will enable us to retrain the model faster and reduce the false predictions until a new model is trained.

Gulcan and Can proposed a method that addresses concept drift in multi label data streams [12]. The method Label Dependency Drift Detector(LD3) works alongside any multi-label classifier. LD3 works by exploiting the predicted label dependencies and if there is a significant change in the dependencies, a concept drift is signalled. The algorithm uses a threshold and two windows, one for new samples and one for old samples, along with a correlation window that keeps track of past correlations between the data points.

Jain et al. address the issue of concept drift in anomaly detection of network traffic [14]. The authors propose two techniques, an Error Rate based concept drift detection and Data Distribution based Concept Drift Detection. The method uses a K-Means Clustering for reducing the dataset size and upgrade the training dataset. The final classifier is a Support Vector Machine(SVM) for anomaly detection.

3 METHOD

Generative Adversarial Networks(GAN) are valuable for creating synthetic data. Goodfellow et al. introduced the concept of generative adversarial networks [9]. A GAN comprises a **Generator** and a **Discriminator**. The **Generator** creates synthetic data, and the **Discriminator** predicts whether the input data belongs to either the real or synthetic data created by the Generator. The training continues until the discriminator's loss drops below a certain threshold. Generative Adversarial Networks are used in a wide variety of domains such as for music generation [17], image super-resolution [27], or even medical image reconstruction [3]. We understand from the different domain applications that Generative

Adversarial Networks are good at generating different output distributions. Furthermore, this is the property we use in our method. section 3 describes the complete method. The code is available as open source[1].

3.1 Framework

We use a **_Generator_** $\theta_G(s)$ that accepts a sequence $s = < x_{t-k} \ldots x_t >$ with length k of input feature vectors x. The main aim of the Generator is to predict the next feature vector x_{n+1}. More precisely, in Equation 1 we define $\theta_G(x)$ as

$$\theta_G(s) : x_{n-k}, ..., x_n \longmapsto x_{n+1} \tag{1}$$

The **_Discriminator_** $\theta_D(x)$ is a multi-class dynamic discriminator that predicts whether an input data belongs to one of the input distributions from the data or is a synthetic data point generated by $\theta_G(s)$. We define a mapping function $\theta_D(x)$, which maps the incoming feature vectors to a set of identified input data distributions or an unidentified data distribution, as explained in Equation 2.

$$\theta_D(x) : x_i \longmapsto \Pi_1, ..., \Pi_n, \Pi_{n+1}, \tag{2}$$

where,
- x_i is an input feature vector
- Π_j is the $j - th$ input distribution with $j = 1$ indicating the current distribution and $j > 1$ being the $j-1$ past distribution detected by our GAN

Detecting Drifts: We map all incoming input feature vectors to one of the distributions. We signal a drift if all the contiguous inputs in a batch are mapped to any of the same data distributions other than the current data distribution(P_1). Equation 3 gives the formal description.

$$\forall x \in \delta, where \theta_D(x) \neq \Pi_{curr} \&$$
$$\forall (x_1, x_2) \in \delta, \theta_D(x_1) = \theta_D(x_2) \tag{3}$$

where,
- x is an input feature vector
- δ is the input data batch to θ_D

Dynamic Discriminator Extension: Initially, we start with a training dataset of size ρ and train the GAN so that θ_D learns to discriminate the real data from the data generated by θ_G. The current data distribution is the data distribution of the training dataset of size ρ. Data generated by θ_G represents the drifted data. θ_G can generate the next in the sequence from completely different input distributions, and can also generate data sequences that do not belong to any previously seen data distributions. This capability of θ_G allows us to set a class for previously unseen distributions in θ_D. We also label the data so that anything other than previously seen distributions is assigned to the same class. A concept drift can be either from a previously seen distribution, except for the first concept drift, or an entirely new drift. If a previously seen data distribution repeats, we use the data from that distribution to bolster the training of the secondary classifier. It is also possible to reuse only the old data to train the classifier or even the old classifier itself if it is stored. However, we do not choose to do so because there could be different data distributions in an extensive

---

[1]https://github.com/cfellicious/DriftGAN

dataset, and the space requirements might be infeasible for larger datasets.

One of the main issues with Generative Adversarial Networks is the mode collapse. Mode collapse occurs when the Generator learns only one of the input modes instead of generating generalized data. We counter the mode collapse problem by having a random mini-batch input to the Generator (θ_G) and the Discriminator(θ_D). Using mini batches forces the generator network to learn to generalize better rather than falling back to a single mode in the input distribution. We also standardize the input data by the mean and standard deviation, which compresses the input feature vector. ODIN implemented a variation of the standardization method [26].

3.2 Network Architectures

For our framework, we need two networks, one for the **Generator**(θ_G) and one for the **Discriminator**(θ_D). The Generator predicts the input feature vector in the sequence, and therefore we can get away with using a smaller network to learn the function. On the other hand, the multi-class Discriminator requires much more expressive power and therefore has a slightly larger network when compared to the Generator. The symbols N, P, R, S represent the number of neurons in each layer. In our case, N equals the sequence length multiplied by the input feature vector size, and P and R are constants of 128 and 4096, respectively. S is the output size equal to the size of a single input feature vector.

The $Discriminator(\theta_D)$ is a slightly larger network, and the size of the top layer of the network is incremented by one every time a previously unseen drift is detected. The first three layers do not change during the execution. We have two layers of 1024 neurons, each with the final layer having a Sigmoid non-linearity to the final layer. The number of neurons on the final layer at any point depends on the number of unique drifts seen by the discriminator plus one neuron as a generalization of all the unseen drifts.

The main objective on the design of the GAN is to have the networks as small as possible as the GAN is not a static part of the detection process. Every time a concept drift is detected, the GAN undergoes training and having a larger model that could capture and learn more information could be detrimental as it takes longer to train. Therefore, we avoid using larger and more complex networks and go for the simplest methods to speedup the training. The primary reason that small networks work well in our experiment setting is due to the fact that we do not use noise as the latent vectors to θ_G but rather a sequence of real input. This allows some leeway with having a smaller network to SACadapt to the inputs to only predict the next in the sequence.

3.3 Training

We start with an initial window of size ρ where the input feature vectors of that window are assumed to belong to the same distribution. The Discriminator, at this point, has only two output neurons, one for the current distribution of the window and one for unseen distributions. We use the Adadelta optimizer for both the Generator and discriminator. Furthermore, we use the CrossEntropyLoss function for the Discriminator. As for the Generator, the loss is a Mean Squared Error loss based on the actual $n + 1$ vector in the sequence added to the CrossEntropyLoss of the Discriminator. These additions of loss values allow the Generator to fool the Discriminator better while also better predicting the next in-sequence input vector.

4 ACTIVE LEARNING

We apply our approach to a real use case from planetary science to demonstrate its applicability as an improved sampling strategy.

4.1 Use case description

SJ: You can massively shorten the introduction and refer to this image describing the use case

(a)

(b)
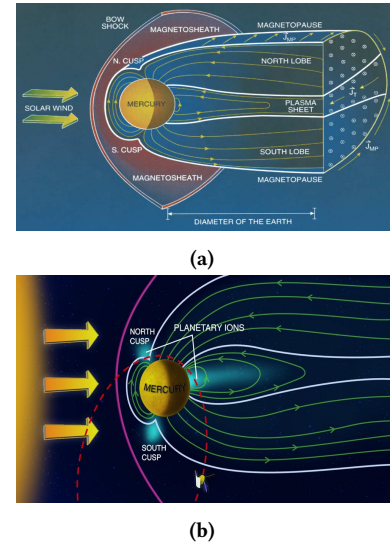
Figure 1: (a) Schematic view of Mercury's magnetic conditions [23].The bow shock slows down the approaching solar wind to subsonic speeds. The magnetopause further acts as an obstacle. (b) A typical MESSENGER orbit path: the spacecraft passed from the *interplanetary magnetic field* (IMF) through bow shock, magnetosheath, magnetopause and magnetosphere regions of Mercury and then through the same sequence in reverse [28].

The *magnetosphere* of a planet is the area surrounding it where the planet's magnetic field is stronger than that of the surrounding interplanetary space. The *magnetopause* is the outer boundary of this region, while the *magnetosheath* lies above it and extends to the bow shock. The *bow shock* is a shock wave that decelerates the solar wind and deflects it around the planet's magnetospheric cavity (C.f Figure 3). Studying magnetic fields of planets has been a long standing interest in planetary science. For Mercury, the innermost planet in our solar system, the magnetic field is peculiar: one, it is extremely weak– about 1 % that of the Earth, second, due to its close proximity to the sun, is heavily influenced by the varying solar wind conditions, and thus very dynamic. Studying such magnetospheres can yield interesting insights into more complex magnetospheres such as the Earth's. In such pursuit, NASA launched a spacecraft called MESSENGER to orbit Mercury and conduct long-term empirical studies[24]. MESSENGER collected

Table 1: Hyperparameters used in the code, their symbols and explanations

Symbol	Hyperparameter	Explanation
ρ	Window Size	Number of training instances for drifts
δ	Batch size	Number of testing instances that should agree for a drift
W	Data Window	Window of sequential input feature vectors of size ρ
δ	Data Batch	A batch of data for testing
$\Pi(x)$	Input distribution	Represents the input distribution of x
Π_{curr}	Current distribution	Drift not signalled as long as input in this distribution
ω	Seen distributions	Set of seen input distributions $\Pi(x_n) \in \omega$
μ	Mean	Mean of the input feature vector
σ	Standard Deviation	Standard deviation of the input feature vector
λ	Fraction	Percentage of historical data to be used for training

---

Algorithm 1: GAN-based Drift Detection. Table 1 explains the hyperparameters

---

FindDrifts(ρ, b) $\rightarrow$ Window size and batch size respectively ;
$W = data[1 : \rho]$ $\rightarrow$ Obtain ρ samples for training the GAN;
$\theta_G, \theta_D = \text{TrainGan}(W)$ $\rightarrow$ Train the GAN;
$\Pi_{curr} = \Pi(W) \rightarrow$ Set the current distribution;
$\omega = \Pi_{curr} \rightarrow$ Add curr distribution to seen distributions **while** *isDataBatchAvailable* **do**
    $W = x_{n-b}..x_b \rightarrow$ Obtain new batch ;
    $\delta = \forall x \in W(x - \mu(x))/\sigma(x)$;
    if $\forall x \in \delta, where\theta_D(x) \neq \Pi_{curr}$ & $\forall(x_1, x_2) \in \delta \; \theta_D(x_1) = \theta_D(x_2)$ **then**
        Batch prediction consensus on input not belonging to current data distribution, Drift Detected ;
        if $\Pi(\delta) \notin \omega$ **then**
            New input distribution detected ;
            $\omega = \omega \cup \Pi(\delta)$;
            Retrain θ_D using θ_G with the added new input distribution $\Pi(\delta)$;
    $\Pi_{curr} = \Pi(\delta)$

---

over 8000 incidents of bow shock and magnetopause crossings during its four-year journey from 2011 to 2015, due to the relatively small size of Mercury's magnetosphere compared to Earth's.

However, since supervised learning typically requires a large number of samples for training, they also proposed an active learning approach to select the most informative orbits using an entropy criterion. The underlying notion here is that orbit similarity can be exploited to select the orbits with maximum unseen variation. We take a step further and introduce a more sophisticated active learning method that further reduces the lower bound on required training samples by a significant margin. We evaluate this approach against the existing technique and random and demonstrate a distinct superiority. Preparing annotations in planetary science applications is a labour intense and costly endeavour– one that can be greatly ameliorated with Active learning methods.

The dataset we use for our experiments in this work contains magnetic field measurements from the MESSENGER magnetometer instrument. The data is already preprocessed with removal of calibration signals, enrichment with Mercury position data, and split according to UTC day boundaries and Mercury apoapsis. This cleaned and prepared dataset introduced in [15] contains 2776 orbits and is ready for Machine Learning modelling. The final annotations

that have been derived from the work of [21] have finally a class distribution given in in tab:labels. The boundary classes of interest are the Bow shock (SK) and Magnetopause (MP) boundaries. SJ: imho all tables should be booktabs

Table 2: Class labels with their abbreviations and frequency of occurence. The boundary classes are highly underrepresented. Classification involves accurately detecting these relatively rare events.

label	magnetic region	share
0	interplanetary magnetic field (IMF)	64.8 %
1	bow shock crossing (**SK**)	3.7 %
2	magnetosheath (MSh)	14.8 %
3	magnetopause crossing (**MP**)	2.3 %
4	magnetosphere (MSp)	14.4 %

We follow the assumption that the orbits in the dataset can be grouped into a sets of similar orbits based on their crossing locations and signatures. Upon visual inspection of orbits [CLEVERREF NOT WORKING], it becomes evident that adjacent orbits are likely to be similar to one another with respect to bow shock and magnetopause

crossing signatures, while orbits further apart tend to be more different. This pattern appears to have a cyclical nature, where patterns repeat periodically. We exploit this characteristic present in the data, to identify groups with recurring patterns. The hypothesis here is that by identifying groups, we can constrain our sampling to include a fixed size of orbits from each during training to obtain a generalisable performance with a significantly lesser amount of training data. This would optimise the sampling strategy with entropy alone [15] Orbit characteristics can be based on several factors, for instance the z-component of the magnetic field in IMF, denoted as B_{z-IMF}. The solar-wind–magnetosphere interaction is particularly sensitive to B_{z-IMF} [1], as it determines the state of the magnetosphere, specifically the amount of incoming energy to the magnetosphere and the magnetic flux magnitude on the open field lines. The cosine of Mercury azimuth angle, denoted as $cos\alpha$ [25], indicates whether the direction of shock normal is quasi-perpendicular or quasi-parallel to the direction of IMF. The shock conditions principally determine the magnetic signature during a bow shock boundary. We use these similarity in features to prepare a first set of groupings for use as labels to train the drift classifier. This classifier then classifies the next stream of orbital data into either of the previously seen groups or a new one. If an orbit is identified to have an unseen distribution, it is assigned a new label.
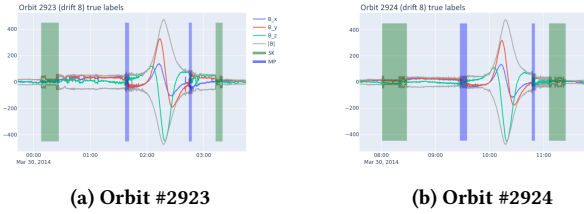

(a) Orbit #2923   **(b) Orbit #2924**

Figure 2: Orbits adjacent to one another are similar in crossing signatures and locations.

NOTE: We change the algorithm to expect recurring drifts.

The drift detection algorithm can be formally defined as detect_drifts() (C.falg: alg1) with parameters d_{gen} (generator label), O (sequential set of unlabelled orbits), $\mathcal{D}_{init}$ (hash table of initial orbit groupings with orbit numbers and drift labels):

We use the catalogue of detected drifts and sample orbits from them incrementally in an Active learning setting. Within each drift group we rank all the orbits according to an informativeness measure as used in [15]. As our classification model has a series of Multinoulli distributions for output, we measure the orbit uncertainty as a function of the output probabilities using *Shannon entropy* Consider the training set $\mathcal{T} \subseteq \mathbb{R}^{d \times w} \times \{IMF, SK, MSh, MP, MSp\}^{w+f}$ with number of features $d \in \mathbb{N}$, window size $w \in \mathbb{N}$. Given a model prediction $\hat{Y} = [\hat{\boldsymbol{y}}^{(1)}, \ldots, \hat{\boldsymbol{y}}^{(w)}] \in [0,1]^{5 \times (w)}$, we define its uncertainty as

$$\mathfrak{u}(\hat{Y}) := \max_j H(\hat{\boldsymbol{y}}^{(j)}) = -\min_j \sum_{i=1}^{5} y_i^{(j)} \log(y_i^{(j)}),$$

where $H : \triangle^4 \to \mathbb{R}$ is the Shannon entropy on the standard 4-simplex.

To achieve this on the orbit level, we must reduce the individual window uncertainties to a single orbit score. In contrast to the

---

Algorithm 2: detect_drifts($d_{gen}, O, \mathcal{D}_{init} : \mathbb{R}^{d \times w} \to \mathbb{R}^{d_{gen}}$) :

$\mathcal{D} := \mathcal{D}_{init}$ Hash map of orbits and drift labels. $gen :=$ Generator() $: \mathbb{R}^{d \times w} \to \mathbb{R}^{d \times w}$ Generator. $dis :=$ Discriminator() $: \mathbb{R}^{d \times w} \to \mathbb{R}^{|\mathcal{D}|}$ Discriminator. $\mathcal{T} :=$ create_training_dataset($O, \mathcal{D}$) Labelled training orbits. $\mathcal{T}_{gen} :=$ equalise_and_concatenate($\mathcal{T}$) Training data for generator. $dis :=$ train_discriminator($\mathcal{T}, dis, gen, d_{gen}$) Initial training. $gen :=$ train_generator($\mathcal{T}_{gen}, dis, gen, d_{gen}$) $i := 0$ Current orbit number. **while** $i < |O|$ **do**-_

  $d := dis(O_i)$ Get most probable drift label from discriminator.

  if $d \in [\mathcal{D}_1 : \mathcal{D}_{|j|}]$ **then**

    reset_top_layer(dis) Reset the top Linear layer.

  else

    update(dis) Reset the top Linear layer and increase its output size by 1. $d_{gen} = d_{gen} + 1$

  $n = 1$ Number of adjacent orbits in a drift.

  if $d \neq \mathcal{D}[i-1]$ **then**

    $n = 14$

  $j := 0$

  while $j < n$ **do**

    $\mathcal{D}[O_{i+j}] := d$ Assign drift label to orbits. $j := j+1$ $\mathcal{T} :=$ create_training_dataset($O, \mathcal{D}$) Dataset with new drift. $\mathcal{T}_{gen} :=$ equalise_and_concatenate($\mathcal{T}$) $dis :=$ train_discriminator($\mathcal{T}, dis, gen, d_{gen}$) $gen :=$ train_generator($\mathcal{T}_{gen}, dis, gen, d_{gen}$) $i := i + n$ Move to the next unlabelled orbit. return $\mathcal{D}$

---

approach in [15] who calculate the entropy for only the windows including the crossing labels, we remain agnostic to it and use the entire region to calculate the orbit uncertainty $\hat{f}_{\boldsymbol{\theta}} : \mathbb{R}^{d \times w} \to \mathbb{R}^{d \times (w+f)}$ for our task by averaging over all the windows $\widetilde{\mathcal{D}_o}$ as:

$$\mathfrak{U}_{\hat{f}_{\boldsymbol{\theta}}}(\widetilde{\mathcal{D}_o}) := \frac{1}{|\widetilde{\mathcal{D}_o}|} \sum_{(X, \boldsymbol{y}) \in \widetilde{\mathcal{D}_o}} \mathfrak{u}(\hat{f}_{\boldsymbol{\theta}}(X))$$

Using this uncertainty measure, we formulate our active learning procedure in algorithm 4.

SJ: NOTE: Include the increment function $\partial(n)$ to express how many orbits are added to the training set

This algorithm can be written down formally as sample_orbits() with parameters Ω (set of all orbits), $\mathcal{D}$ (set of all drift labels) and m (maximum number of training orbits per drift, hyperparameter) in the following way:

To have an augmented set of fixed shaped input vectors and extract aggregated features from them, we use a sliding window, with a stride of size one. This is done to ensure translation equivariance i.e.the position of the crossing should not matter to the model. Hence, the model's input is a window of $w \in \mathbb{N}$ successive time steps, each comprising of $d \in \mathbb{N}$ scalar features. The input windows can thus be expressed as:

$$X := \begin{bmatrix} \boldsymbol{x}^{(1)} & \boldsymbol{x}^{(2)} & \cdots & \boldsymbol{x}^{(w)} \end{bmatrix} \in \mathbb{R}^{d \times w}$$

Consequently, the target output matrix of one-hot vectors is $Y \in \mathbb{R}^{5 \times (w)}$.

Algorithm 3: sample_orbits($\Omega, \mathcal{D}, m$) :

$\mathcal{T} := \varnothing$ Set of training orbits. $\mathcal{S} := \varnothing$ Set of testing orbits.
$d = 1$ Current drift label. **while** $d \leq |\mathcal{D}|$ **do-**⌐
$\mathcal{S} := \mathcal{S} \cup$ random.sample($\Omega_d, |\Omega_d|//5$) Randomly sample 20%
as SJ: validation/testing data. $E :=$ hash_table() **for**
⌐ $\omega \in \Omega_d \backslash \mathcal{S}$ **do**
Get entropy for all non-testing orbits with current drift.
$E[\omega] :=$ entropy(ω) $\mathcal{T} := \mathcal{T} \cup$ top_k(E, m) Add m most
uncertain orbits. $d := d + 1$ Move to the next drift. return $\mathcal{T}, \mathcal{S}$

Formally, the classification task is framed as a multi-dimensional multi-class classification: Given the window X, we predict a sequence of magnetic region probabilities, where each column sums up to one:

$$\hat{Y} := \begin{bmatrix} p_{1,1} & \cdots & p_{1,w} \\ \vdots & & \vdots \\ p_{5,1} & \cdots & p_{5,w} \end{bmatrix} \in [0,1]^{5 \times (w)}$$

We pass these normalised through a neural network comprised of a convolutional block and a recurrent block, referred to as CRNNSJ: cite, with the final activations represented as: $\gamma_\theta : p_{ij} = \gamma_\theta(X)$ where γ is the CRNN model, with θ as its parameters.

Table 3: CRNN performance our approach vs benchmark. Trained on 278 sampled orbits and tested on 97 orbits.ALD denotes Active Learning with Drift detection (our approach). AD denotes Active Learning without drift detection. All values are rounded to two decimals. Best values are marked in bold.

	IMF		SK		MSh		MP		MSp		Average	
|=|=|=|=|=|=|=|=|=|=|=|=|=|	ALD	AL	ALD	AL	ALD	AL	ALD	AL	ALD	AL	ALD	AL
F1	**0.98**	0.93	**0.66**	0.5	**0.86**	0.86	0.626	**0.64**	**0.99**	0.93	**0.82**	0.78
Recall	**0.97**	0.95	**0.73**	0.71	**0.85**	0.84	0.645	**0.76**	**0.95**	0.89	**0.83**	0.83
Precision	**0.98**	0.92	**0.59**	0.39	0.88	**0.88**	**0.61**	0.56	**0.98**	0.93	**0.80**	0.74

Table 4: CRNN performance Random Sampling. The number of training orbits is 278, like in table 2 to compare active learning against random sampling

	IMF	SK	MSh	MP	MSp
==|=|=|=|= **F1**	0.718	0.397	0.518	0.457	0.780
Recall	0.592	0.442	0.531	0.494	0.959
Precision	0.912	0.361	0.506	0.425	0.658

5 DETECTION OF BOW SHOCK AND MAGNETOPAUSE CROSSINGS AROUND MERCURY

In this section, we look at an application scenario involving the magnetosphere readings of Mercury. We apply our approach to a real use case from planetary science to demonstrate the applicability as an improved sampling strategy.
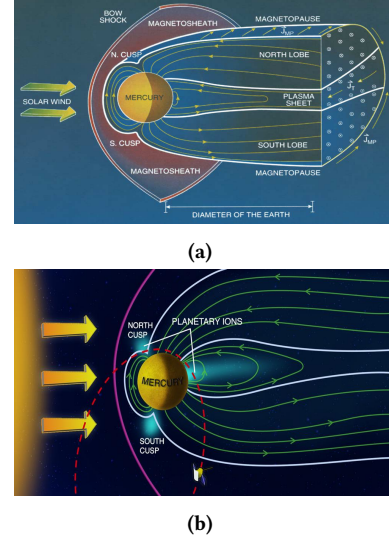
(a)

(b)

Figure 3: (a) Schematic view of Mercury's magnetic conditions [23].The bow shock slows down the approaching solar wind to subsonic speeds. The magnetopause further acts as an obstacle. (b) A typical MESSENGER orbit path: the spacecraft passed from the *interplanetary magnetic field* (IMF) through bow shock, magnetosheath, magnetopause and magnetosphere regions of Mercury and then through the same sequence in reverse [28].

The *magnetosphere* of a planet is the area surrounding it where the planet's magnetic field is stronger than that of the surrounding interplanetary space. The *magnetopause* is the outer boundary of this region; while the *magnetosheath* lies above it and extends to the bow shock. The *bow shock* is a shock wave that decelerates the solar wind and deflects it around the planet's magnetospheric cavity as shown in Figure 3. The magnetic field of Merucury is highly variable due to its closest proximity to the sun, and a weak magnetic core. However, due to its global nature and similarity to Earth's magnetic field, it has gathered increasing attention from researchers. To this end, NASA launched a spacecraft called MESSENGER to orbit Mercury and conduct long-term empirical studies [24]. MESSENGER collected over 4000 orbits' worth of data.

We need to classify whether the spacecraft is in interplanetary space, bow shock crossing(SK), magnetosheath (MSh), magnetopause crossing(MP) or in Mercury's magnetosphere(MSp). MESSENGER spends most of its time in interplanetary space and the other events are comparatively rare as shown in Table 5. The magnetic field of Mercury is very dynamic owing to the presence of solar winds and is considerably affected by the solar activity. Therefore, identifying such events is difficult. Supervised learning typically requires a large number of samples for training but labelling all the orbits and events is extremely time-consuming and expensive as we require experts to label the data by hand.

The dataset we use for our experiments in this work contains magnetic field measurements from the MESSENGER magnetometer instrument. The data is already preprocessed with removal of calibration signals, enrichment with Mercury position data, and split according to UTC day boundaries and Mercury apoapsis. This cleaned and prepared dataset introduced in [15] contains 2776 orbits and is ready for Machine Learning modelling. The boundary classes of interest are the Bow shock (SK) and Magnetopause (MP) boundaries Table 5. We assume that the crossing signatures fol-

Table 5: Class labels with their abbreviations and frequency of occurrence.

label	magnetic region	share
0	interplanetary magnetic field (IMF)	64.8 %
1	bow shock crossing (**SK**)	3.7 %
2	magnetosheath (MSh)	14.8 %
3	magnetopause crossing (**MP**)	2.3 %
4	magnetosphere (MSp)	14.4 %

low a recurring drift theme due to cylical weather and solar wind conditions, and thus test our approach on this use case where we apply our proposed drift detection algorithm to select the most informative orbits for training and compare that to a equivalent sampling/active learning strategy with random sampling. Preparing annotations in planetary science applications is a labour intense and costly endeavour– one that can be greatly ameliorated with Active learning methods.

5.1 Using DriftGAN for sampling orbits

We follow the assumption that the orbits in the dataset can be grouped into a sets of similar orbits based on their crossing locations and signatures. Upon visual inspection of orbits(Figure 4), it becomes evident that adjacent orbits are likely to be similar to one another with respect to bow shock and magnetopause crossing signatures, while orbits further apart tend to be more different. This pattern appears to have a cyclical nature, where patterns repeat periodically. We exploit this characteristic present in the data, to identify groups with recurring patterns. The hypothesis here is that by identifying groups, we can constrain our sampling to include a fixed size of orbits from each during training to obtain a generalized performance with a significantly lesser amount of training data. This would optimise the sampling strategy with entropy alone [15] We use B_{z-IMF} and $cos\alpha$ to prepare an initial catalogue of similar orbit groups and use those labels to train our drift detectino algorithm. These two features are recognised in literature to maximally influence the orbit charactersitics Orbit characteristics can be based on several factors, for instance the z-component of the magnetic field in IMF, denoted as B_{z-IMF}. The solar-wind–magnetosphere interaction is particularly sensitive to B_{z-IMF} [1], as it determines the state of the magnetosphere, specifically the amount of incoming energy to the magnetosphere and the magnetic flux magnitude on the open field lines. The cosine of Mercury azimuth angle, denoted as $cos\alpha$ [25], indicates whether the direction of shock normal is quasi-perpendicular or quasi-parallel to the direction of IMF. The shock conditions principally determine the magnetic signature during a bow shock boundary. We use B_{z-IMF} and $cos\alpha$ to prepare an

initial catalogue of similar orbit groups and use those labels to train our drift detection algorithm. These two features are recognised in literature to maximally influence the orbit .
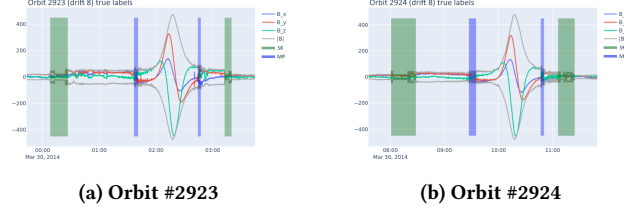

(a) Orbit #2923   (b) Orbit #2924

Figure 4: Orbits adjacent to one another are similar in crossing signatures and locations.

This algorithm can be written down formally as $sample\_orbits()$ with parameters Ω (set of all orbits), $\mathcal{D}$ (set of all drift labels) and m (maximum number of training orbits per drift, hyperparameter) as given in algorithm 4.

Algorithm 4: $\texttt{sample\_orbits}(\Omega, \mathcal{D}, k)$:

$\mathcal{T} := \varnothing \rightarrow$Set of training orbits;
$\mathcal{S} := \varnothing \rightarrow$Set of testing orbits;
$d = 1 \rightarrow$Current drift label.;
while $d \leq |\mathcal{D}|$ **do**
    $\mathcal{S} := \mathcal{S} \cup \texttt{random.sample}(\Omega_d, |\Omega_d|/5) \rightarrow$ Randomly sample 20% as testing data;
    $E := \texttt{hash\_table}()$;
    for $\omega \in \Omega_d \backslash \mathcal{S}$ **do**
       Get entropy for all non-testing orbits with current drift.;
       $E[\omega] := \texttt{entropy}(\omega)$;
    $\mathcal{T} := \mathcal{T} \cup \texttt{top\_k}(E, k) \rightarrow$Add k most uncertain orbits;
    $d := d + 1 \rightarrow$Move to the next drift;
return $\mathcal{T}, \mathcal{S}$;

---

6 RESULTS

We evaluate our approach on different real-world datasets and compare it with the current state of the art. We also use our approach to predict the incoming data distributions and predict on the different magnetic events of the MESSENGER spacecraft while orbiting Mercury.

The current state of the art for unsupervised drift detection is from [10] on a public dataset. We use the same datasets used by the authors for the state-of-the-art. In order to facilitate a direct comparison to state-of-the-art, we use the same real-world datasets, the same metrics, and the secondary classifier for our experiments. Using the same datasets allows us for an apples-to-apples comparison.

The secondary classifier predicts the actual labels on the incoming data. The better the concept drift identification of our model, the better the performance of the secondary classifier, which is a supervised learning algorithm, a Hoeffding Tree Classifier, in our

case. We use a Hoeffding Tree Classifier as the secondary supervised classifier because of the support for streaming datasets. The main reason behind the choice is to enable better comparison with the state of the art as they use the same classifier as their secondary classifier.

Our comparison is based on the results of the current state-of-the-art in unsupervised concept drift detection published by [10]. We use accuracy as the metric for evaluating all the datasets as it is the most prevalent metric for analyzing the algorithm's performance. Most other publications also use the accuracy metric; hence, we use the same metric to evaluate our experiments for comparison. We use the implementation of scikit-learn [19]. In addition, we use the implementation of the scikit-multiflow library for the Hoeffding Tree classifier [18]. This library caters to analyzing streaming datasets. For the evaluation, we use the Interleaved-Test-Then-Train method outlined in the paper by [11]. This method tests every incoming feature vector and then adapts the classifier to the tested feature vector. We reset and retrain the Hoeffding tree classifier whenever a concept drift is detected.

6.1 Standard Drift Datasets

We compare our results with the state of the art but we also developed three baseline methods. The baseline methods are explained below.

6.1.1 Baseline Methods: In addition to comparing our results with state of the art, we also compared the experimental results with three additional methods that we call our baseline methods for the experiment. The baseline methods we selected are probable real-world scenarios where concept drift is not considered. The first method is one we see in most machine learning models where the model is trained on a training set and then never modified. The purpose of such a baseline is to provide an understanding of the performance if we never consider concept drift and let the model predict over the whole test set. We call this method *Initial Learn*, as the model learns on a small initial set of input feature vectors while predicting the rest.

In the following scenario, we retrain the model as specific intervals without considering concept drift. We can liken this to retraining a fraud prediction model every month or week so that the model is trained on the latest input data distributions. We call this method the *Regular Retrain* method, as the model is regularly updated without considering concept drift. From a computational point of view, this is the most expensive model as the model is retrained far more often than required, irrespective of the fact whether a concept drift is present in the input data or not.

The final baseline scenario is where the model is continuously updated with the input data distribution. We can see this as partially updating a model as soon as the labels to the data are obtained. Although such a scenario is very unlikely in a real-world scenario due to the issues with updating models in a production environment and attaining labels for each and every data point. As explained by [13], labelling data streams is expensive and requires quite a lot of effort to obtain. But such a baseline gives us an idea of how good a model could get with only partial updates. We call this baseline method the *Regular Update*.

We compare all the methods using the Hoeffding Tree classifier with all default parameters. The **Regular Retrain** method makes use of the reset method of the Hoeffding Tree classifier while the **Regular Update** method uses the partial fit method of the algorithm.

6.1.2 Results and Discussion: We set the training window size(ρ) to one hundred input training instances based on experimentation with different values. Also [10] found that having a window size of 100 provides a good balance between generalizing to the current distribution and mostly having only a single distribution within the window. A smaller window size does not adequately generalise features for the network to discriminate between the different contexts adequately.

Our method beats the state of the art in most datasets as seen in Table 6. For larger datasets, like Rialto, Airlines, Poker or Covertype(marked with * in the table), we only considered the first 50000 input instances. This is due to the sheer number of unique drifts that arise and the increased training time once a previously unseen drift is detected.

One of the best examples is for the **Airlines** dataset where our model did not detect a drift at all for the first 50000 input data instances. This also produces a better score compared to the other methods. But on the other hand, our model performs worse for the Electricity dataset. One reason for this could be that the dataset has a higher number of outliers within our window size(ρ).

6.2 Predictions on Magnetosphere data

We test our active learning method with and without drift detection and also compare it with random sampling on the data. From Table 7, we can see that our method performs well across all the categories except the detection of the Magnetosheath and Magnetopause. In the case of Magnetosheath, our method equals the active learning method. The only metric where the drift detection method varies significantly is the recall in the call of magnetopause. On an average, the active learning method with drift detection outperforms the method without drift detection as seen in Table 7. Both active learning methods are far superior when compared to random sampling. We cannot compare our drift detection framework with other methods such as D3 or OCDD because all these methods focus only on detecting drifts. They do not provide any information on whether the drift has happened before and thereby choose a classifier based on the distribution. Therefore, only the Active Learning method with and without drift detection is compared against each other.

7 CONCLUSION

Our method addresses the issue of identifying concept drifts that have occurred in the past, and we use that information to increase the training data for the secondary classifier. This memory of previous distributions allows our method to improve prediction accuracy, reduce the training time for the model, and reduce the effort in labelling data. Our model shows better performance when compared to other state-of-the-art models across the different publicly available datasets. Our method as a framework is easy to use and can be applied to different application domains with little to no modifications. We applied our method in an active learning loop to sample the most informative orbits to train a classifier. Our method

Table 6: comparison of accuracy values of different methods.

Dataset	Initial Train	Regular Update	Regular Retrain	OCDD	Our Method
Electricity	56.44	77.69	75.4	**86.22**	79.86
Poker	50.12	74.08	63.05	76.63	**76.99***
Cover Type	47.69	82.49	55.48	**88.36**	82.49*
Rialto	09.95	31.373	51.02	**66.68**	54.84
Spam	66.48	88.35	75.81	87.02	**89.28**
Phishing	83.49	90.26	89.18	90.56	**91.37**
Airlines	56.13	63.88	59.72	63.16	**65.59***
Outdoor	15.21	57.15	12.00	62.24	**62.57***

Table 7: CRNN performance our approach vs benchmark. Trained on 278 sampled orbits and tested on 97 orbits.ALD denotes Active Learning with Drift detection (our approach). AD denotes Active Learning without drift detection. All values are rounded to two decimals. Best values are marked in bold.

	IMF		SK		MSh		MP		MSp		Average	
	ALD	AL	ALD	AL	ALD	AL	ALD	AL	ALD	AL	ALD	AL
F1	**0.98**	0.93	**0.66**	0.5	0.86	0.86	0.63	**0.64**	**0.96**	0.93	**0.82**	0.78
Recall	**0.97**	0.95	**0.73**	0.71	**0.85**	0.84	0.64	**0.76**	**0.95**	0.89	0.83	0.83
Precision	**0.98**	0.92	**0.59**	0.39	0.88	0.88	**0.61**	0.56	**0.98**	0.93	**0.80**	0.74

Table 8: CRNN performance Random Sampling. The number of training orbits is 278, like in table 2 to compare active learning against random sampling

	IMF	SK	MSh	MP	MSp	Average
F1	0.718	0.397	0.518	0.457	0.780	0.574
Recall	0.592	0.442	0.531	0.494	0.959	0.604
Precision	0.912	0.361	0.506	0.425	0.658	0.572

performed vastly superior over random sampling and an active learning method. The only drawback of our framework is that the amount of training time increases with the increase in the number of unique drifts. Having to store the data of previous distributions in memory to retrain the model could also be memory inefficient. As for future work, we expect to use a Conditional Generative Adversarial Network that will enable us to simulate different input probability distributions effectively.

REFERENCES

[1] Brian Anderson and C.L. Johnson. 2013. A magnetic disturbance index for Mercury's magnetic field derived from MESSENGER Magnetometer data. *Geochemistry, Geophysics, Geosystems* 14 (09 2013), 3875–3886. https://doi.org/10.1002/ggge.20242

[2] Albert Bifet and Ricard Gavalda. 2007. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining.* SIAM, 443–448.

[3] Yuhua Chen, Feng Shi, Anthony G Christodoulou, Yibin Xie, Zhengwei Zhou, and Debiao Li. 2018. Efficient and accurate MRI super-resolution using a generative adversarial network and 3D multi-level densely connected network. In *International Conference on Medical Image Computing and Computer-Assisted Intervention.* Springer, 91–99.

[4] Kanishka Ghosh Dastidar, Michael Granitzer, Wissam Siblini, et al. 2022. The Importance of Future Information in Credit Card Fraud Detection. In *International Conference on Artificial Intelligence and Statistics.* PMLR, 10067–10077.

[5] Danilo Rafael de Lima Cabral and Roberto Souto Maior de Barros. 2018. Concept drift detection based on Fisher's Exact test. *Information Sciences* 442 (2018), 220–234.

[6] Denis Moreira dos Reis, Peter Flach, Stan Matwin, and Gustavo Batista. 2016. 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.* 1545–1554.

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

[8] Kanishka Ghosh Dastidar, Johannes Jurgovsky, Wissam Siblini, and Michael Granitzer. 2022. NAG: neural feature aggregation framework for credit card fraud detection. *Knowledge and Information Systems* 64, 3 (2022), 831–858.

[9] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial networks. *arXiv preprint arXiv:1406.2661* (2014).

[10] Ömer Gözüaçık, Alican Büyükçakır, Hamed Bonab, and Fazli Can. 2019. Unsupervised concept drift detection with a discriminative classifier. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management.* 2365–2368.

[11] Ömer Gözüaçık and Fazli Can. 2021. Concept learning using one-class classifiers for implicit drift detection in evolving data streams. *Artificial Intelligence Review* 54, 5 (2021), 3725–3747.

[12] Ege Berkay Gulcan and Fazli Can. 2023. Unsupervised concept drift detection for multi-label data streams. *Artificial Intelligence Review* 56, 3 (2023), 2401–2434.

[13] Hanqing Hu, Mehmed Kantardzic, and Tegjyot S Sethi. 2020. No Free Lunch Theorem for concept drift detection in streaming data classification: A review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 10, 2 (2020), e1327.

[14] Meenal Jain, Gagandeep Kaur, and Vikas Saxena. 2022. A K-Means clustering and SVM based hybrid concept drift detection technique for network anomaly detection. *Expert Systems with Applications* 193 (2022), 116510.

[15] Sahib Julka, Nikolas Kirschstein, Michael Granitzer, Alexander Lavrukhin, and Ute Amerstorfer. 2022. Deep Active Learning for Detection of Mercury's Bow Shock and Magnetopause Crossings. (2022).

[16] Ana Justel, Daniel Peña, and Rubén Zamar. 1997. A multivariate Kolmogorov-Smirnov test of goodness of fit. *Statistics & Probability Letters* 35, 3 (1997), 251–259.

[17] Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brébisson, Yoshua Bengio, and Aaron C Courville. 2019. Melgan: Generative adversarial networks for conditional waveform synthesis. *Advances in neural information processing systems* 32 (2019).

[18] Jacob Montiel, Jesse Read, Albert Bifet, and Talel Abdessalem. 2018. Scikit-multiflow: A multi-output streaming framework. *The Journal of Machine Learning Research* 19, 1 (2018), 2915–2914.

[19] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[20] Federico Perrotta, Tony Parry, and Luis C Neves. 2017. Application of machine learning for fuel consumption modelling of trucks. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 3810–3815.

[21] Lydia C Philpott, Catherine L Johnson, Brian J Anderson, and Reka M Winslow. 2020. The shape of Mercury's magnetopause: The picture from MESSENGER magnetometer observations and future prospects for BepiColombo. *Journal of Geophysical Research: Space Physics* 125, 5 (2020), e2019JA027544.

[22] Saleh Seyedzadeh, Farzad Pour Rahimian, Ivan Glesk, and Marc Roper. 2018. Machine learning for estimation of building energy consumption and performance: a review. *Visualization in Engineering* 6, 1 (2018), 1–20.

[23] James A Slavin. 2004. Mercury's magnetosphere. *Advances in Space Research* 33, 11 (2004), 1859–1874.

[24] Sean C Solomon, Ralph L McNutt, Robert E Gold, and Deborah L Domingue. 2007. MESSENGER mission overview. *The Messenger Mission to Mercury* (2007), 3–39.

[25] Torbjörn Sundberg, Scott A. Boardsen, James A. Slavin, Vadim M. Uritsky, Brian J. Anderson, Haje Korth, Daniel J. Gershman, Jim M. Raines, Thomas H. Zurbuchen, and Sean C. Solomon. 2013. Cyclic reformation of a quasi-parallel bow shock at Mercury: MESSENGER observations. *Journal of Geophysical Research: Space Physics* 118 (2013), 6457–6464. https://doi.org/10.1002/jgra.50602

[26] Abhijit Suprem, Joy Arulraj, Calton Pu, and Joao Ferreira. 2020. Odin: automated drift detection and recovery in video analytics. *arXiv preprint arXiv:2009.05440* (2020).

[27] Wenlong Zhang, Yihao Liu, Chao Dong, and Yu Qiao. 2019. Ranksrgan: Generative adversarial networks with ranker for image super-resolution. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 3096–3105.

[28] Thomas H Zurbuchen, Jim M Raines, James A Slavin, Daniel J Gershman, Jason A Gilbert, George Gloeckler, Brian J Anderson, Daniel N Baker, Haje Korth, Stamatios M Krimigis, et al. 2011. MESSENGER observations of the spatial distribution of planetary ions near Mercury. *Science* 333, 6051 (2011), 1862–1865.