The Journal of Supercomputing (2024) 80:12043–12061
https://doi.org/10.1007/s11227-024-05889-1

Data stream classification in dynamic feature space using feature mapping

Reza Sajedi[1] · **Mohammadreza Razzazi**[1]

Accepted: 4 January 2024 / Published online: 7 February 2024

Abstract
Stream learning in dynamic feature space has evolved into an immensely popular field. This problem assumes that each instance of the data stream may have different features, and the feature spaces of the classifier and the instances may also differ. Such assumptions are more relevant to real-world applications dealing with data streams, where dimensions are not predetermined and fixed. This study introduces a general algorithm for data stream classification in dynamic feature space using feature mapping. In contrast with the other studies, the proposed algorithm is not based on a specific classifier and can cooperate with any classifier best suited for an intended application. It discovers the relationship between the features and estimates the unavailable features previously observed by the classifier. This technique helps to exploit the full potential of the classifier. Furthermore, empirical experiments and comparisons with modern methods demonstrate that the proposed algorithm achieves higher accuracy.

Keywords Algorithm · Online machine learning · Varying feature space · Feature evolution

1 Introduction

In recent years, stream learning, a.k.a. online learning or incremental learning, has garnered significant attention. Prior to this, offline/batch learning algorithms were prevalent. However, in certain applications, such as stock market prediction, intrusion detection, and recommender systems, where regular model updates are essential, these traditional methods prove inefficient [19, 26, 27]. Infinite length,

✉ Reza Sajedi
r.sajedi@aut.ac.ir

Mohammadreza Razzazi
razzazi@aut.ac.ir

[1] Department of Computer Engineering, Amirkabir University of Technology, Tehran, Iran

concept drift, concept evolution, feature drift, and feature evolution are some of the well-known challenges that stream learning deals with. The phenomenon of concept drift occurs when the relation between the input data and the target variable changes over time [6] (data distribution variation), and the meaning of concept evolution is the emergence of new classes in the data stream [22]. In feature drift, it is assumed that the set of features is fixed and finite. However, the relevance of the selected features may decrease over time, leading to a reduction in model efficiency [1]. On the other hand, feature evolution posits that each instance of the data stream may have different independent variables [21].

In this research, the main focus is on the challenge of feature evolution. This challenge is also known as dynamic feature space, varying feature space, incremental/decremental features, or capricious data stream. Many real-world applications face this challenge. For example, in document classification, each document may contain different words [17]. In environment monitoring systems or health-care systems which are based on the Internet of Things, some old sensors may be replaced with new sensors, and the feature space of instances may differ from the feature space of the classifier [13, 32]. In recommender systems, new items are continuously added [15, 23]. In general, any problem dealing with data stream may face this challenge.

This research presents a general algorithm for data stream classification in dynamic feature space using feature mapping (DCDF2M). The assumption is made that features can change in any order; thus, the features in each instance of the data stream may differ from those in the preceding or subsequent instance. Each instance may contain new features that the classifier has not seen before. Also, some features previously observed by the classifier may be unavailable. The proposed algorithm exploits the full potential of the classifier for prediction by discovering relationships between features and estimating unavailables. Importantly, the algorithm is general, meaning that it is not tied to a specific classifier and can cooperate with any high-performance classifier in a desired application. To empirically validate the algorithm, experiments were conducted on ten datasets, and the results were compared with two existing algorithms, generative learning with streaming capricious data (GLSC) [9] and online learning from varying features (OLVF) [2]. These algorithms share the same assumptions in problem definition. The findings demonstrate that the proposed algorithm achieves higher accuracy. The contributions of this study can be summarized as follows:

- Introducing a general algorithm for data stream classification in dynamic feature space using feature mapping.
- Using a baseline model to estimate unavailable features and proving an upper bound for the estimation error.
- Evaluating the algorithm through empirical experiments, comparing its performance with two state-of-the-art algorithms, and demonstrating higher accuracy.

The subsequent sections are structured as follows: Sect. 2 provides an overview of the relevant literature. In Sect. 3, the proposed algorithm is delineated. The evaluation scenario is explicated, and the results are scrutinized in Sect. 4. Lastly, Sect. 5 encapsulates the concluding remarks of this study.

2 Related work

Several studies have made restrictive assumptions about how the features can evolve. In [31], the concept of a trapezoidal data stream has been introduced, in which each instance of the data stream must contain all the features previously observed; that is, each new instance's length must be greater than or equal to the previous one.

In [11], the assumption is made regarding the presence of an overlapping period. During this short period, the instances must contain all the old and new features to learn the relation between these two feature spaces. After the overlapping period, all old features must disappear, and all new features must appear simultaneously. In the extended version of this study, the concept drift challenge is also considered [33]. Another version [12] is presented in a semi-supervised setting. In [20], which assumes the existence of the overlapping period, deep learning is used to discover the relationship between features.

Some other studies do not impose restrictive assumptions on how features change. These more realistic studies operate under the assumption that each instance in the data stream may possess different features compared to its preceding or subsequent instance. The current work also adopts such an assumption in the problem definition.

In [2], the OLVF algorithm is proposed, which learns to classify feature spaces and instances simultaneously. The algorithm dynamically projects the instance classifier and the training instance onto their shared feature subspace for prediction. The feature space classifier predicts the projection confidence for the given feature space. The instance classifier is updated by following the empirical risk minimization principle, and the strength of the constraints is scaled with the projection confidence. The distinction between OLVF and DCDF2M lies in the fact that OLVF is specifically tailored to the support vector machine (SVM) classifier, while DCDF2M is general and can cooperate with any classifier. OLVF solely utilizes the available features in each instance for classification. In contrast, DCDF2M employs the feature mapping technique to estimate the values of unavailable features, thereby homogenizing the feature space of the instance with that of the classifier.

In [8, 9], the GLSC algorithm has been introduced. Similar to DCDF2M, this algorithm is designed to estimate unavailable features by discerning relationships among features. Another variant of the algorithm has been presented within a semi-supervised setting [10]. Additionally, an alternative version has been proposed, accommodating features with mixed types rather than solely numerical values [29]. The distinction between GLSC and DCDF2M is as follows: GLSC is specifically tied to the logistic regression (LR) classifier, whereas DCDF2M is general. In GLSC, the relationships between pairs of features are not considered independent of each other. Conversely, in DCDF2M, the modeling of relationships is conducted by observing the condition of independence, resulting in improved performance in scenarios where the feature space undergoes substantial changes. In GLSC, it is

assumed that all features have a linear relationship with each other. In contrast, the DCDF2M algorithm allows for the modeling of polynomial relations.

3 Proposed algorithm

The task involves the classification of data streams in a dynamic feature space. The binary mode is considered, but it can be extended to multiclass using some techniques like one-versus-rest (OvR) [3]. At each round t, an instance $x_t \subset \mathbb{F} \times \mathbb{R}$ is received where $\mathbb{F} = \{f_1, f_2, f_3, \cdots\}$ denotes the set of all possible features, and $\mathbb{R}$ denotes the set of real numbers. After predicting the class label $\hat{y}_t = \mathcal{H}(x_t)$, the true label $y_t \in \{+1, -1\}$ is revealed. The classifier $\mathcal{H}$ is updated using the prediction result and ground truth. It will be demonstrated how to exploit the feature mapping technique to have a better prediction $\hat{y}_t = \mathcal{H}(x_t \cup x_t^{'})$ by estimating unavailable feature's values $x_t^{'}$ for each instance.

3.1 Data structure

In order to model the relationships between features and store data, a data structure is required. The graph is selected as the preferred data structure. It can be represented using either an adjacency matrix or an adjacency list. Due to the dynamic nature inherent in the problem, a preference may exist for the adjacency list. The choice between a directed or an undirected graph depends on the nature of the relationship between features. An undirected graph is deemed suitable for linear relationships between features, whereas a directed graph is considered more appropriate for mapping through polynomial regression. This choice arises from the fact that a two-sided map between a pair of features cannot be achieved using a single regression model when multiple coefficients need to be learned. Consequently, the directed graph is employed to address this general case. Features are treated as vertices and maps as edges, as illustrated in Fig. 1.

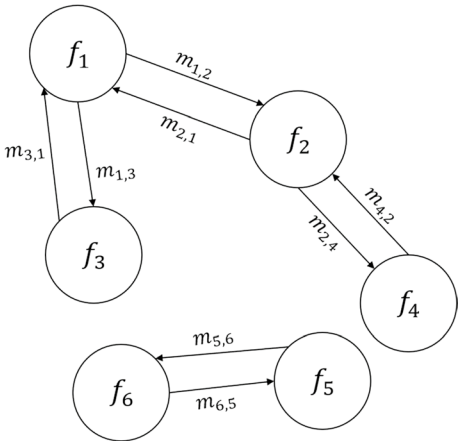

Fig. 1 A sample feature space modeled using a graph

Each feature (vertex) f has a property denoted by $\mathcal{I}_f$, which stores the list of incoming edges. It also preserves some other properties required for the normalization process. Each directed map (edge) $m_{i,j}$ which connects f_i to f_j has five properties:

- $\mathcal{N}m_{i,j}$: Number of observed instances in which both f_i and f_j are present
- $\mu_{m_{i,j}}$: Mean of values of f_j
- $\mathcal{M}_{m_{i,j}}$: Map function
- $\mathcal{E}m_{i,j}$: Sum of squared map errors
- $\bar{\mathcal{E}}m_{i,j}$: Sum of squared mean errors

3.2 Normalization

Due to the broad range of raw data values, correct functioning of objective functions is impeded in certain machine learning algorithms without normalization. Another consideration for the incorporation of normalization is its role in significantly expediting the convergence of gradient descent [14]. Feature standardization, a.k.a. Z-score normalization, involves assigning zero mean and unit-variance values to each feature. This approach has been widely employed for normalization across various machine learning algorithms [16]. To achieve this, the mean of each feature is subtracted from its respective values, and the result is then divided by the standard deviation of the feature. However, a challenge arises in the context of data streams, as it is impractical to ascertain the mean and standard deviation values before processing the entire dataset. To address this issue, normalization is performed by computing running statistics, commonly referred to as moving statistics. The underlying concept is to estimate the mean and update it as new values become available, applying the same principle to the standard deviation [28]. Consequently, the normalization process can be outlined as follows:

$$
\begin{aligned}
&\forall f \in \mathcal{S}_{x_t} : \\
&\quad \mathcal{N}_{t,f} = \mathcal{N}_{t-1,f} + 1 \\
&\quad \mu_{t,f} = \mu_{t-1,f} + \frac{x_{t,f} - \mu_{t-1,f}}{\mathcal{N}_{t,f}} \\
&\quad s_{t,f} = s_{t-1,f} + (x_{t,f} - \mu_{t-1,f}) \times (x_{t,f} - \mu_{t,f}) \\
&\quad \sigma_{t,f} = \sqrt{\frac{s_{t,f}}{\mathcal{N}_{t,f}}} \\
&\quad x_{t,f} = \frac{x_{t,f} - \mu_{t,f}}{\sigma_{t,f}}
\end{aligned}
\tag{1}
$$

Where $\mu_{t,f}$ is the mean, $s_{t,f}$ is the running sum of squares, and $\sigma_{t,f}$ is the running standard deviation of feature f at the moment t. The feature set of instance x_t is denoted by $\mathcal{S}_{x_t}$. The value associated with feature f in instance x_t is represented by $x_{t,f}$.

3.3 Feature mapping

Feature mapping involves a regression task wherein the relationship between pairs of features is learned. The association between two variables can be perceived as either linear or characterized by higher degrees, such as polynomials [7]. In this context, the general case is taken into consideration, specifically addressing polynomial regression. The univariate map function is defined as follows:

$$\mathcal{M}(x_i) = \sum_{d=0}^{\mathcal{D}} \theta_d x_i^d = \theta_0 + \theta_1 x_i + \theta_2 x_i^2 + \cdots + \theta_{\mathcal{D}} x_i^{\mathcal{D}} \tag{2}$$

Where $\mathcal{D}$ is the degree hyperparameter, and x_i is the value of feature f_i in instance x. The purpose of regression analysis is to learn the unknown parameters θ. These parameters are estimated by minimizing a cost/loss function, specifically squared loss (a.k.a. L2 loss) in this case:

$$J(\theta) = (\mathcal{M}(x_i) - x_j)^2 \tag{3}$$

The stochastic gradient descent (SGD) technique [18] is employed to minimize this cost function. To achieve this, it is necessary to calculate the gradient of the cost function:

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \frac{\partial J(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_{\mathcal{D}}} \end{bmatrix} = \begin{bmatrix} 2(\mathcal{M}(x_i) - x_j) \\ 2x_i(\mathcal{M}(x_i) - x_j) \\ 2x_i^2(\mathcal{M}(x_i) - x_j) \\ \vdots \\ 2x_i^{\mathcal{D}}(\mathcal{M}(x_i) - x_j) \end{bmatrix} \tag{4}$$

At each round t, upon receiving an instance x_t from the data stream, the maps between each pair of features within that instance are updated. Through this process, the evolving relationship between the features is incrementally learned. The update of each map involves the assignment of new values to its five properties. The new value of each property is contingent upon its prior value in the preceding round. The procedure for updating the maps is delineated in Eq. 5.

$$\begin{aligned} &\forall f_i, f_j \in \mathcal{S}_{x_t} : \\ &\quad \mathcal{N}_{t,m_{i,j}} = \mathcal{N}_{t-1,m_{i,j}} + 1 \\ &\quad \mu_{t,m_{i,j}} = \mu_{t-1,m_{i,j}} + \frac{x_{t,j} - \mu_{t-1,m_{i,j}}}{\mathcal{N}_{t,m_{i,j}}} \\ &\quad \bar{\mathcal{E}}_{t,m_{i,j}} = \bar{\mathcal{E}}_{t-1,m_{i,j}} + (x_{t,j} - \mu_{t-1,m_{i,j}}) \times (x_{t,j} - \mu_{t,m_{i,j}}) \\ &\quad \mathcal{E}_{t,m_{i,j}} = \mathcal{E}_{t-1,m_{i,j}} + (x_{t,j} - \mathcal{M}_{m_{i,j}}(x_{t,i}))^2 \\ &\quad \theta_{t,\mathcal{M}_{m_{i,j}}} = \theta_{t-1,\mathcal{M}_{m_{i,j}}} - \eta \nabla_\theta J(\mathcal{M}_{m_{i,j}}(x_{t,i})) \end{aligned} \tag{5}$$

After the maps have been updated, the estimation of unavailable features becomes possible through the utilization of the available ones. These features, missing in the instance x_t, have been observed in preceding instances and subsequently incorporated into the feature space. A learned relationship may exist between these unavailable features and the features present in the current instance. Consequently, by employing the maps established up to moment t, the estimation of unavailable features allows for the optimal utilization of the classifier in prediction.

The set of available features in the instance x_t is denoted by $\mathcal{S}_{x_t}$, and the set encompassing all features observed in prior instances up to moment t is represented by F_t. Therefore, the set of unavailable features in the instance x_t is derived by computing the difference between these two sets ($F_t - \mathcal{S}_{x_t}$). To estimate each unavailable feature $f_j \in F_t - \mathcal{S}_{x_t}$, the relevant maps from the available features $\mathcal{S}_{x_t}$ to f_j must be selected. The coefficient of determination, a widely recognized criterion in regression analysis [4], is employed for the selection of such maps. The calculation of this criterion for each map is as follows:

$$R^2_{m_{i,j}} = 1 - \frac{\mathcal{E}_{m_{i,j}}}{\bar{\mathcal{E}}_{m_{i,j}}} \qquad R^2_{m_{i,j}} \in (-\infty, 1] \tag{6}$$

The coefficient of determination measures the performance of the learned regression model compared with a baseline model that only reports the mean of the target variable, independent of any other variable. If the map error is less than the mean error, the coefficient of determination becomes greater than zero, indicating a better performance of the learned model compared with the baseline model. If the map error is greater than the mean error, the coefficient of determination becomes less than zero. In such cases, the baseline model performs better than the learned model, and using that map for proper estimation is not recommended. The presence of a negative coefficient of determination may signify the occurrence of under-fitting or a failure to converge within the regression model. This issue may arise due to various factors, including an inadequate number of instances for updating the map, improperly tuned hyperparameters, an abundance of noise or outlier, and the absence of a polynomial relationship between the features situated on either side of the map. Therefore, one of the necessary conditions for selecting a map will be as follows:

$$R^2_{m_{i,j}} > \mathcal{R} \qquad \mathcal{R} \in [0, 1) \tag{7}$$

Where $\mathcal{R}$ is a hyperparameter determining the coefficient of determination threshold for selecting a map. It is often assumed to be zero, but assigning higher values is also possible, which will be explained further.

Another consideration in the selection of a map involves establishing the minimum number of instances in which the features on both sides of the map are observed simultaneously. While a map may attain a high coefficient of determination with only one or two updates, there exists the potential for the presented instances to be noisy, and those features may not be observed simultaneously in subsequent instances. In such scenarios, the map with the highest coefficient of determination is employed to estimate each of the two unavailable features

in subsequent instances, leading to the possibility of inappropriate values. To address this concern, an additional necessary condition is defined for the selection of a map:

$$\mathcal{N}_{m_{i,j}} > \mathcal{C} \qquad \mathcal{C} \in \mathbb{W} \tag{8}$$

Where $\mathcal{C}$ is a hyperparameter that determines the threshold for selecting a map after the number of times it has been updated. $\mathbb{W}$ denotes the set of whole numbers.

Finally, the estimation of unavailable features is conducted under all the aforementioned conditions, as illustrated in Eq. 9. For each unavailable feature, the weighted average of the estimated values associated with the selected maps is computed. The coefficients of determination associated with each map serve as the weights, determining the contribution of each map to the estimation.

$$\begin{aligned} x_t^{'} = \bigcup_{f_j \in F_t - \mathcal{S}_{x_t}} (f_j, \frac{\sum_{m_{i,j} \in \mathcal{I}_{f_j}} R^2_{m_{i,j}} \times \mathcal{M}_{m_{i,j}}(x_{t,i})}{\sum_{m_{i,j} \in \mathcal{I}_{f_j}} R^2_{m_{i,j}}}) \\ s.t. \quad f_i \in \mathcal{S}_{x_t}, \quad R^2_{m_{i,j}} > \mathcal{R}, \quad \mathcal{N}_{m_{i,j}} > \mathcal{C} \end{aligned} \tag{9}$$

3.4 Theoretical analysis

As previously indicated, the estimation of unavailable features is accomplished using Eq. 9. If no appropriate map is identified to estimate an unavailable feature f_j, its value is implicitly considered zero. Given that the normalization is conducted based on the standard normal distribution, zero represents the mean of f_j up to the moment t. Consequently, if $\mathcal{C}$ is sufficiently large, in accordance with Eq. 10, the mean error becomes an upper bound for the estimation error. Thus, the performance of the proposed algorithm in feature estimation will never deteriorate beyond the performance of the baseline model.

$$\sum_{t=1}^{\infty} \sum_{f_j \in F_t - \mathcal{S}_{x_t}} \sum_{m_{i,j} \in \mathcal{I}_{f_j}} min(\mathcal{E}_{t,m_{i,j}}, \bar{\mathcal{E}}_{t,m_{i,j}}) \leqslant \sum_{t=1}^{\infty} \sum_{f_j \in F_t - \mathcal{S}_{x_t}} \sum_{m_{i,j} \in \mathcal{I}_{f_j}} \bar{\mathcal{E}}_{t,m_{i,j}} \tag{10}$$

3.5 Time and space complexity

The pseudocode of DCDF2M is demonstrated in Algorithm 1. In each iteration, upon receiving each instance x_t, adding new features to the feature space and normalization, which are specified in lines 4 and 5, takes $O(|x_t|)$. Updating the maps in line 6 is done in $O(|x_t|^2)$. Estimating unavailable features in line 7 takes $O(|F_t|^2)$. The

time complexity of label prediction and classifier update in lines 8–10 depends on the chosen classifier $\mathcal{H}$. If $\mathcal{H}$ is a linear classifier like logistic regression, this process is done in $O(|F_{t+1}|)$. Therefore, DCDF2M's time complexity will be $O(|F_t|^2 + |x_t|^2)$, provided that the chosen classifier $\mathcal{H}$ does not exceed this limit.

Learning the relationships between features and storing the maps requires $O(|F_t|^2)$ space. The amount of space consumed by the classifier $\mathcal{H}$ also varies. If we consider logistic regression for $\mathcal{H}$, its space complexity will be $O(|F_t|)$. Therefore, DCDF2M's space complexity will be $O(|F_t|^2)$, provided that the chosen classifier $\mathcal{H}$ does not exceed this limit.

Algorithm 1 DCDF2M

Input: Classifier $\mathcal{H}$, Hyperparameters $\mathcal{D}$, $\mathcal{R}$, $\mathcal{C}$.

1: Initialize feature set: $F_t = \varnothing$
2: **for** $t = 1$ **to** $t = \infty$ **do**
3: Receive instance: x_t
4: Add new observed features: $F_{t+1} = F_t \cup \mathcal{S}_{x_t}$
5: Perform normalization using Eq. 1
6: Update maps using Eq. 5
7: Estimate unavailable feature values using Eq. 9: $x_t^{'}$
8: Predict class label: $\hat{y}_t = \mathcal{H}(x_t \cup x_t^{'})$
9: Receive true class label: y_t
10: Update classifier $\mathcal{H}$ using: $y_t, x_t \cup x_t^{'}$

3.6 Pruning weak maps

As previously mentioned, should the performance of a map be inferior to that of the baseline model, it is precluded from utilization. The storage and updating of such maps in successive iterations prove futile, leading to escalated memory usage and execution time. The identification and pruning of these inefficient maps can be accomplished by validating a specific condition during the update step. Consequently, the following condition is defined:

$$R^2_{m_{i,j}} \leq \mathcal{R} \quad and \quad \mathcal{N}_{m_{i,j}} > \bar{\mathcal{C}} \qquad \bar{\mathcal{C}} \in \mathbb{W} : \bar{\mathcal{C}} \geq \mathcal{C} \tag{11}$$

Where $\bar{\mathcal{C}}$ is a hyperparameter that determines a threshold for pruning a weak map after the number of times it has been updated. In simpler terms, this condition states that if a map does not satisfy condition 7 after $\bar{\mathcal{C}}$ times updating, it can be pruned.

Sometimes, adequate resources may be unavailable to update and store all maps in high dimensions. In such cases, only the most essential maps can be preserved by assigning higher values to the hyperparameter $\mathcal{R}$. This technique can prove highly effective in accelerating algorithm execution and reducing memory usage.

4 Experiments

In this section, DCDF2M's performance is evaluated compared with two contemporary algorithms, GLSC and OLVF. To ensure a fair comparison of the results, the same evaluation scenario and metric is followed [2, 9]. The primary metric for evaluation is accuracy. It is important to note that the concern regarding the appropriateness of accuracy arises primarily in the context of imbalanced datasets. In this case, since the datasets are nearly balanced, this concern is not applicable.

In addition to accuracy, run time and memory usage are also evaluated. In stream mining, where real-time processing is crucial, understanding the necessary computational resources is imperative. Assessments of run time and memory usage offer valuable insights into the algorithm's efficiency and scalability. They contribute to a comprehensive evaluation of its practical utility and feasibility for deployment in resource-constrained environments. These considerations extend beyond task-specific metrics, providing a holistic perspective on the algorithm's performance and applicability in stream mining scenarios, as evidenced by their utilization in various studies [9, 25, 30].

To simulate a dynamic feature space, a parameter called removing ratio is defined, and experiments are performed for values of 0.25, 0.5, and 0.75. The removing ratio determines how many features should be removed from each instance. For example, when the removing ratio is equal to 0.5, half of the features of each dataset's instance are removed uniformly at random. The experiments are repeated 20 times for each dataset and removing ratio, and the mean of the results is reported. In each repetition, the order of the instances is shuffled.

As the proposed algorithm is designed to be general, three well-known classifiers have been selected for the hyperparameter $\mathcal{H}$: logistic regression (LR), Gaussian Naive Bayes (GNB), and Hoeffding tree (HT). This selection leads to the generation of three variants of DCDF2M. These classifiers have been employed using the River library, with default hyperparameters. The River library implements common data mining and machine learning algorithms with an online approach, professionally utilized for processing and analyzing data streams [24]. Other hyperparameters of DCDF2M are set as follows: $\mathcal{D} = 1$, $\mathcal{R} = 0.1$, $\mathcal{C} = 5$, and $\bar{\mathcal{C}} = 20$. These values were determined through a process of trial and error. The proposed algorithm exhibits insensitivity to hyperparameters, and it has been observed that utilizing these values uniformly across all datasets results in acceptable performance.

Table 1 Specifications of the experiment environment

CPU	Intel(R) Core(TM) i7-4800MQ 2.70GHz
RAM	8GB
OS	Windows 10 64-bit
Lang	Python 3.10.2
Lib	River 0.15.0

It is important to note that conducting the experiment in different environments may yield varying results, exerting a significant influence on execution time. Factors such as the processor, memory, operating system, programming language, and libraries in the experiment environment play a crucial role in shaping the outcomes. The specifications of the experiment environment employed in this research are detailed in Table 1.

4.1 Datasets

Experiments are conducted on ten datasets selected from the UCI repository [5]. These datasets, sourced from various applications, are commonly employed for binary classification tasks. Notably, these datasets align with those utilized in two recent studies [2, 9]. Their deliberate selection aims to ensure a fair comparison between the proposed algorithm and the aforementioned studies. Additional information regarding the number of features and instances in each dataset is provided in Table 2.

4.2 Results

In this subsection, the experimental results are delineated with respect to accuracy, run time, and memory usage individually. The performance of the proposed algorithm is evaluated through a comprehensive analysis and discussion of these results, making comparisons with the GLSC and OLVF algorithms.

4.2.1 Accuracy

In Table 3, the accuracy of algorithms is reported for three different removing ratios in each dataset. Overall, there was no significant difference in the accuracy values between the two algorithms, GLSC and OLVF. However, DCDF2M's superiority

Table 2 Characteristics of the datasets used in experiments

Dataset	Features	Instances
Diabetes	8	768
Magic	10	19020
German	20	1000
Svmguide3	22	1243
WDBC	30	569
Ionosphere	33	351
Spambase	57	4601
Splice	60	3190
A8A	123	22696
DNA	180	3186

Table 3 Accuracy of the algorithms in each dataset using three different removing ratios

Alg./Ds	Diabetes (0.25)	Diabetes (0.50)	Diabetes (0.75)
DCDF2M (LR)	**0.727 ± 0.011**	0.698 ± 0.007	0.661 ± 0.007
DCDF2M (GNB)	0.724 ± 0.012	**0.700 ± 0.009**	**0.668 ± 0.014**
DCDF2M (HT)	0.719 ± 0.012	0.691 ± 0.009	0.654 ± 0.012
GLSC	0.690 ± 0.013	0.661 ± 0.015	0.619 ± 0.015
OLVF	0.699 ± 0.014	0.677 ± 0.010	0.621 ± 0.016
Alg./Ds	Magic (0.25)	Magic (0.50)	Magic (0.75)
DCDF2M (LR)	**0.765 ± 0.002**	**0.735 ± 0.003**	**0.707 ± 0.002**
DCDF2M (GNB)	0.722 ± 0.003	0.710 ± 0.003	0.698 ± 0.005
DCDF2M (HT)	0.749 ± 0.007	0.705 ± 0.005	0.691 ± 0.006
GLSC	0.730 ± 0.013	0.699 ± 0.003	0.649 ± 0.003
OLVF	0.745 ± 0.002	0.696 ± 0.003	0.649 ± 0.003
Alg./Ds	German (0.25)	German (0.50)	German (0.75)
DCDF2M (LR)	**0.724 ± 0.009**	**0.708 ± 0.010**	**0.699 ± 0.002**
DCDF2M (GNB)	0.674 ± 0.035	0.638 ± 0.032	0.638 ± 0.022
DCDF2M (HT)	0.698 ± 0.004	0.697 ± 0.003	0.697 ± 0.004
GLSC	0.622 ± 0.016	0.611 ± 0.013	0.571 ± 0.015
OLVF	0.645 ± 0.009	0.616 ± 0.015	0.575 ± 0.015
Alg./Ds	Svmguide3 (0.25)	Svmguide3 (0.50)	Svmguide3 (0.75)
DCDF2M (LR)	**0.782 ± 0.007**	**0.771 ± 0.005**	**0.767 ± 0.005**
DCDF2M (GNB)	0.781 ± 0.014	0.761 ± 0.020	0.733 ± 0.026
DCDF2M (HT)	0.777 ± 0.007	0.767 ± 0.005	0.763 ± 0.003
GLSC	0.590 ± 0.020	0.594 ± 0.020	0.592 ± 0.015
OLVF	0.620 ± 0.013	0.612 ± 0.009	0.610 ± 0.012
Alg./Ds	WDBC (0.25)	WDBC (0.50)	WDBC (0.75)
DCDF2M (LR)	**0.947 ± 0.006**	**0.931 ± 0.008**	0.889 ± 0.010
DCDF2M (GNB)	0.923 ± 0.007	0.914 ± 0.006	0.890 ± 0.009
DCDF2M (HT)	0.921 ± 0.008	0.913 ± 0.008	0.883 ± 0.017
GLSC	0.931 ± 0.007	0.925 ± 0.009	**0.897 ± 0.013**
OLVF	0.928 ± 0.010	0.916 ± 0.012	0.896 ± 0.013
Alg./Ds	Ionosphere (0.25)	Ionosphere (0.50)	Ionosphere (0.75)
DCDF2M (LR)	0.815 ± 0.010	0.780 ± 0.021	0.712 ± 0.018
DCDF2M (GNB)	**0.828 ± 0.014**	**0.803 ± 0.020**	**0.754 ± 0.019**
DCDF2M (HT)	0.818 ± 0.018	0.783 ± 0.027	0.736 ± 0.020
GLSC	0.673 ± 0.027	0.704 ± 0.018	0.671 ± 0.027
OLVF	0.698 ± 0.011	0.685 ± 0.016	0.668 ± 0.021
Alg./Ds	Spambase (0.25)	Spambase (0.50)	Spambase (0.75)
DCDF2M (LR)	**0.886 ± 0.003**	**0.853 ± 0.007**	0.787 ± 0.005
DCDF2M (GNB)	0.675 ± 0.020	0.669 ± 0.013	0.644 ± 0.015

Table 3 (continued)

Alg./Ds	Spambase (0.25)	Spambase (0.50)	Spambase (0.75)
DCDF2M (HT)	0.694 ± 0.013	0.671 ± 0.013	0.632 ± 0.014
GLSC	0.873 ± 0.006	0.846 ± 0.005	**0.797 ± 0.007**
OLVF	0.843 ± 0.006	0.806 ± 0.007	0.752 ± 0.009
Alg./Ds	Splice (0.25)	Splice (0.50)	Splice (0.75)
DCDF2M (LR)	0.747 ± 0.006	0.695 ± 0.007	0.621 ± 0.009
DCDF2M (GNB)	**0.813 ± 0.007**	**0.761 ± 0.007**	**0.669 ± 0.008**
DCDF2M (HT)	0.812 ± 0.007	0.760 ± 0.008	0.664 ± 0.014
GLSC	0.708 ± 0.011	0.677 ± 0.013	0.618 ± 0.009
OLVF	0.750 ± 0.004	0.697 ± 0.006	0.626 ± 0.008
Alg./Ds	A8A (0.25)	A8A (0.50)	A8A (0.75)
DCDF2M (LR)	**0.825 ± 0.001**	**0.811 ± 0.002**	**0.787 ± 0.001**
DCDF2M (GNB)	0.347 ± 0.021	0.375 ± 0.019	0.415 ± 0.026
DCDF2M (HT)	0.759 ± 0.003	0.758 ± 0.003	0.758 ± 0.002
GLSC	0.707 ± 0.002	0.699 ± 0.003	0.675 ± 0.003
OLVF	0.695 ± 0.001	0.686 ± 0.002	0.652 ± 0.003
Alg./Ds	DNA (0.25)	DNA (0.50)	DNA (0.75)
DCDF2M (LR)	**0.878 ± 0.004**	**0.827 ± 0.005**	0.735 ± 0.005
DCDF2M (GNB)	0.860 ± 0.005	0.814 ± 0.006	0.721 ± 0.007
DCDF2M (HT)	0.836 ± 0.020	0.782 ± 0.034	0.713 ± 0.019
GLSC	0.803 ± 0.012	0.782 ± 0.009	0.718 ± 0.005
OLVF	0.865 ± 0.003	0.821 ± 0.004	**0.740 ± 0.006**

The highest accuracy value associated with the best performing algorithm for each dataset and removing ratio is indicated in bold

over these two algorithms is quite evident. When the removing ratio was equal to 0.25 or 0.5, at least one of the variants of DCDF2M outperformed both algorithms in all datasets.

When the removing ratio was 0.75, at least one of the DCDF2M's variants had better performance in seven datasets. In the *WDBC* and *Spambase* datasets, the GLSC algorithm had the best performance, while the OLVF algorithm performed better in the *DNA* dataset. However, their difference compared with the best variant of DCDF2M was negligible, with a margin of only about 1%.

Among the DCDF2M's variants, LR had the best performance in most cases. Only in the *Ionosphere* and *Splice* datasets did the GNB and HT variants perform better.

On average, considering the results of all datasets and different removing ratios, the overall accuracy of DCDF2M is 7.1% higher than the average accuracy of GLSC and OLVF. For a more convenient comparison, the accuracy results are also represented using box plot in Fig. 2.

Moreover, the progressive accuracy of the algorithms in six datasets with a removing ratio of 0.5 is demonstrated in Fig. 3. These graphs depict the accuracy

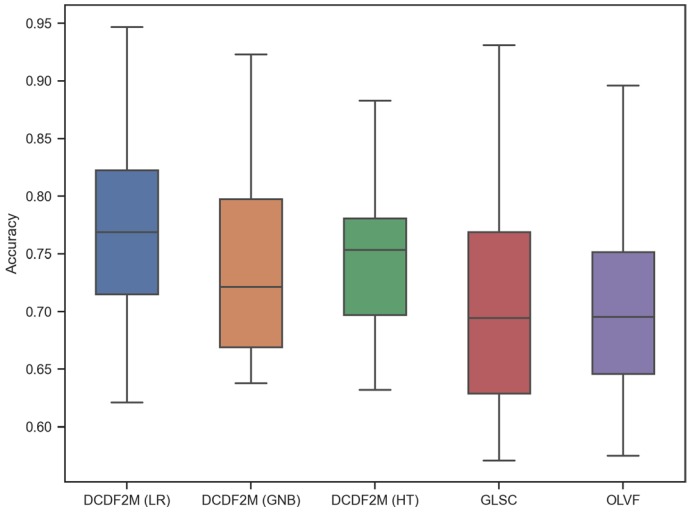

Fig. 2 Representation of the accuracy results using box plot

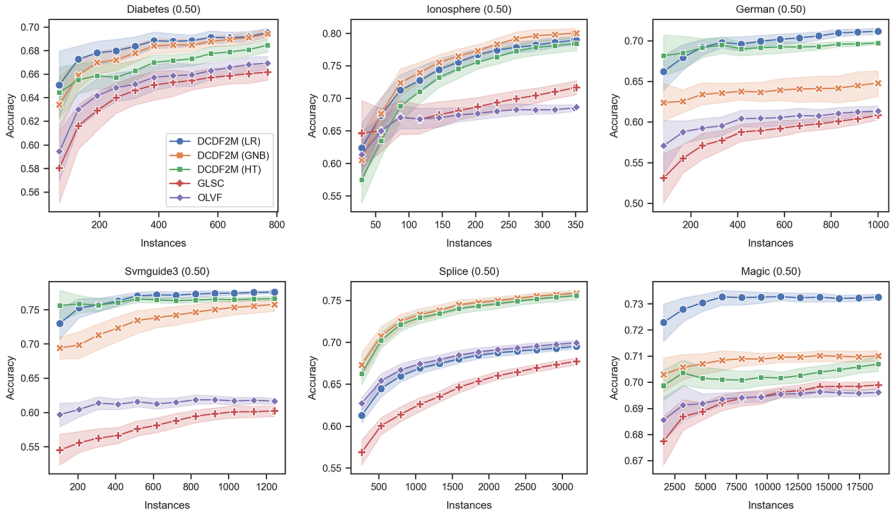

Fig. 3 Progressive accuracy of the algorithms in six datasets with a removing ratio of 0.5

of different algorithms from the initial reception of the first instance from the data stream to the final instance. As observed, in most datasets, accuracy gradually increases with the acquisition of more instances. In the *Magic* dataset, which contains a relatively large number of instances, the accuracy of most algorithms reaches a constant value after a certain point, displaying minimal further change.

This occurrence illustrates the convergence and stability of data distribution throughout the experiment. In the context of the classification problem, if it can be ensured that the data distribution remains constant, the number of instances can be restricted, and the model can be updated only when the distribution of the data undergoes changes. This approach is particularly relevant as labeling each instance is typically an expensive process. The aforementioned considerations are primarily associated with topics such as active learning and concept drift, which fall outside the scope of this research. However, acknowledging these aspects can stimulate valuable contemplation for the refinement of the proposed algorithm and the initiation of future research in this domain.

4.2.2 Run time

As previously mentioned, DCDF2M's time complexity is $O(|F_t|^2 + |x_t|^2)$. Time complexities of GLSC and OLVF are $O(|x_t|^2 \times |F_t|)$ and $O(|F_t| + |x_t|)$, respectively. The run time of the algorithms in the experimental test, reported in Table 4, is also as expected.

The OLVF algorithm had the fastest run time compared with all other algorithms, while the GLSC algorithm had the longest. On average, DCDF2M's run time was lower than GLSC. In GLSC, edges were created between all previously observed features with each new feature observed, whereas in DCDF2M, edges were only created between pairs of features that were observed simultaneously. Additionally, if some pairs of features were found to be unrelated, the edges were removed, leading to a reduction in the run time of DCDF2M compared with GLSC.

Among the different DCDF2M's variants, LR had the lowest run time. The results of the run time are also represented using box plot in Fig. 4.

Table 4 Run time of the algorithms (Seconds)

Ds./Alg	DCDF2M (LR)	DCDF2M (GNB)	DCDF2M (HT)	GLSC	OLVF
Diabetes	0.15	0.23	0.21	0.19	0.05
Magic	3.52	5.54	5.52	5.40	1.13
German	0.49	0.66	0.63	0.59	0.10
Svmguide3	0.71	0.94	0.89	0.70	0.11
WDBC	0.50	0.59	0.62	0.54	0.07
Ionosphere	0.47	0.57	0.60	0.41	0.05
Spambase	3.64	4.48	4.81	11.18	0.80
Splice	3.62	4.42	4.75	8.61	0.57
A8A	35.11	44.81	46.33	233.40	7.71
DNA	32.34	34.34	36.00	69.45	1.53

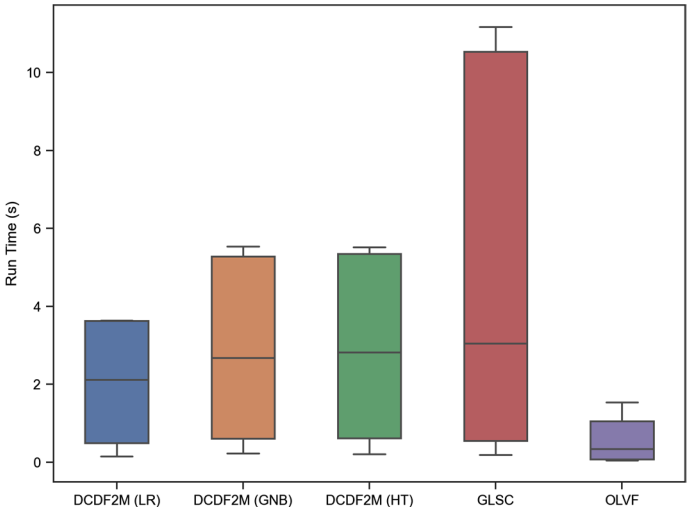

Fig. 4 Representation of the run time results using box plot

Table 5 Memory usage of the algorithms (KB)

Ds./Alg	DCDF2M (LR)	DCDF2M (GNB)	DCDF2M (HT)	GLSC	OLVF
Diabetes	17	67	79	12	7
Magic	42	104	772	14	8
German	77	202	224	40	13
Svmguide3	107	238	335	43	14
WDBC	127	315	345	90	20
Ionosphere	141	347	380	103	20
Spambase	450	808	1682	303	37
Splice	475	852	909	326	38
A8A	1950	2723	9837	1319	75
DNA	5516	6653	6823	3234	119

4.2.3 Memory usage

The algorithms' memory usage is reported in Table 5. OLVF had the lowest memory usage, as its space complexity is $O(|F_t|)$. The space complexity of DCDF2M and GLSC is $O(|F_t|^2)$. The experimental results indicated that the memory consumption of GLSC is lower than DCDF2M, as DCDF2M stores five properties for each pair of features, whereas GLSC only stores one weight for each pair.

Among the different DCDF2M's variants, LR had the lowest memory usage, while HT had the highest. The results are also represented using box plot in Fig. 5.

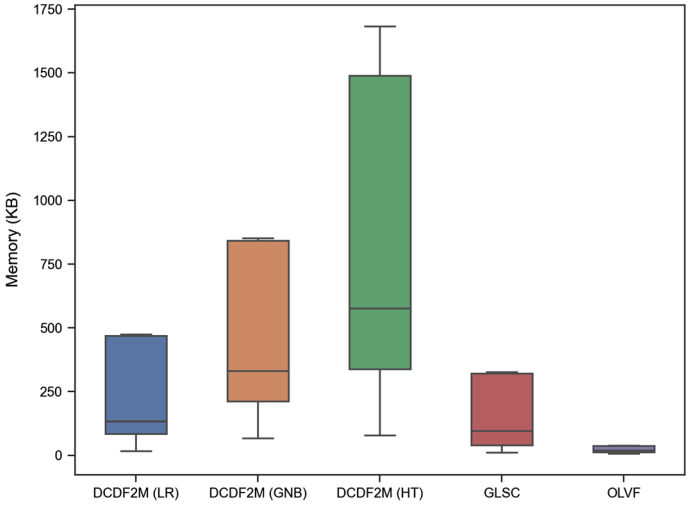

Fig. 5 Representation of the memory usage results using box plot

5 Conclusion

In this research, the problem of data stream classification in dynamic feature space, where changes in features can occur in an arbitrary order, was investigated. An algorithm named DCDF2M was introduced, employing a feature mapping technique to homogenize the feature spaces and harness the full potential of the classifier. This algorithm, characterized by its generality, is not contingent upon a specific classifier, allowing users the flexibility to select the most suitable classifier for their intended application. Empirical evaluation of the algorithm involved the generation of three variants using different classifiers. Experiments were conducted on ten datasets, demonstrating its superiority over OLVF and GLSC, the latest algorithms that share the same assumptions about the problem. On average, the algorithm exhibited a 7.1% improvement in accuracy. In terms of execution time, this algorithm falls within the mid-range and consumes more memory compared to its competitors. To summarize, Table 6 provides a comparison of the key characteristics of the algorithms.

Table 6 Comparing the key characteristics of the algorithms

Characteristic/algorithm	OLVF	GLSC	DCDF2M
Average accuracy	0.716	0.711	0.785
Time complexity	$O(\|F_t\| + \|x_t\|)$	$O(\|x_t\|^2 \times \|F_t\|)$	$O(\|F_t\|^2 + \|x_t\|^2)$
Space complexity	$O(\|F_t\|)$	$O(\|F_t\|^2)$	$O(\|F_t\|^2)$
General	No	No	Yes
Feature mapping	No	Yes	Yes

As previously mentioned, data stream classification confronts various challenges such as concept drift, concept evolution, feature drift, and feature evolution (dynamic feature space). This research specifically addressed one of these challenges. However, it is evident that for the application of the proposed algorithm in real-world scenarios and the acquisition of reliable results, all challenges must be taken into consideration. The demonstrated superiority of the proposed algorithm in its basic state suggests potential for gradual development in the future research endeavors by addressing additional challenges. Incorporating each challenge into the algorithm can be explored as a distinct avenue for further research, aligning with common practices in the literature.

Author Contributions Authors contributed equally to this work.

Funding No funding was received for conducting this study.

Data Availability Statement All datasets are publicly available, and the sources are cited.

Declarations

Conflict of interest The authors have no competing interests to declare that are relevant to the content of this article.

Consent for publication The authors give full consent for publication.

Financial/non-financial interests The authors have no relevant financial or non-financial interests to disclose.

Ethics approval and consent to participate No ethical issue is involved. This research involves no human participants or animals.

References

1. Barddal JP, Gomes HM, Enembreck F (2015) A survey on feature drift adaptation. In: 2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI), IEEE, pp 1053–1060
2. Beyazit E, Alagurajah J, Wu X (2019) Online learning from data streams with varying feature spaces. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp 3232–3239
3. Bishop CM, Nasrabadi NM (2006) Pattern recognition and machine learning, vol 4. Springer
4. Draper NR, Smith H (1998) Applied regression analysis, vol 326. Wiley, New York
5. Dua D, Graff C (2017) UCI machine learning repository. http://archive.ics.uci.edu/ml
6. Gama J, Žliobaitė I, Bifet A et al (2014) A survey on concept drift adaptation. ACM Comput Surv 46(4):1–37
7. Hartley HO (1961) The modified gauss-newton method for the fitting of non-linear regression functions by least squares. Technometrics 3(2):269–280
8. He Y, Wu B, Wu D, et al (2019) Online learning from capricious data streams: a generative approach. In: International Joint Conference on Artificial Intelligence Main Track
9. He Y, Wu B, Wu D et al (2020) Toward mining capricious data streams: a generative approach. IEEE Trans Neural Netw Learn Syst 32(3):1228–1240
10. He Y, Yuan X, Chen S, et al (2021) Online learning in variable feature spaces under incomplete supervision. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp 4106–4114

11. Hou BJ, Zhang L, Zhou ZH (2017) Learning with feature evolvable streams. Adv Neural Inf Process Syst 30:1417–1427
12. Hou BJ, Yan YH, Zhao P, et al (2021) Storage fit learning with feature evolvable streams. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp 7729–7736
13. Hou C, Zhou ZH (2017) One-pass learning with incremental and decremental features. IEEE Trans Pattern Anal Mach Intell 40(11):2776–2792
14. Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning, PMLR, pp 448–456
15. Jakomin M, Bosnić Z, Curk T (2020) Simultaneous incremental matrix factorization for streaming recommender systems. Expert Syst Appl 160:113685
16. Joel G (2015) Data science from scratch. O'Reilly Media
17. Katakis I, Tsoumakas G, Vlahavas I (2006) Dynamic feature space and incremental feature selection for the classiflcation of textual data streams. In: Proceedings of ECML/PKDD-2006 International Workshop on Knowledge Discovery from Data Streams. Springer, pp 107–116
18. Kiefer J, Wolfowitz J (1952) Stochastic estimation of the maximum of a regression function. Ann Math Stat, pp 462–466
19. Li YF, Gao Y, Ayoade G, et al (2019) Multistream classification for cyber threat data with heterogeneous feature space. In: The World Wide Web Conference, pp 2992–2998
20. Lian H, Atwood J, Hou BJ et al (2022) Online Deep Learning from Doubly-Streaming Data. In: Proceedings of the 30th ACM International Conference on Multimedia (MM)
21. Masud MM, Chen Q, Gao J et al (2010a) Classification and novel class detection of data streams in a dynamic feature space. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, pp 337–352
22. Masud MM, Chen Q, Khan L, et al (2010b) Addressing concept-evolution in concept-drifting data streams. In: 2010 IEEE International Conference on Data Mining, IEEE, pp 929–934
23. Matuszyk P, Spiliopoulou M (2017) Stream-based semi-supervised learning for recommender systems. Mach Learn 106:771–798
24. Montiel J, Halford M, Mastelini SM et al (2021) River: machine learning for streaming data in python. J Mach Learn Res 22(1):4945–4952
25. Nakatani S (2022) Memory efficient stream processing for iot devices. In: 2022 International Conference on Algorithms, Data Mining, and Information Technology (ADMIT), IEEE, pp 129–139
26. Singh T, Kalra R, Mishra S et al (2022) An efficient real-time stock prediction exploiting incremental learning and deep learning. Evol Syst pp 1–19
27. Vinagre J, Jorge AM, Al-Ghossein M et al (2022) Preface to the special issue on dynamic recommender systems and user models. User Model User-Adap Inter 32(4):503–507
28. Welford B (1962) Note on a method for calculating corrected sums of squares and products. Technometrics 4(3):419–420
29. Wu D, Zhuo S, Wang Y, et al (2023) Online semi-supervised learning with mix-typed streaming features. In: Proceedings of the 37th AAAI Conference on Artificial Intelligence (AAAI)
30. Yang L, Shami A (2021) A lightweight concept drift detection and adaptation framework for IOT data streams. IEEE Intern Things Magaz 4(2):96–101
31. Zhang Q, Zhang P, Long G et al (2016) Online learning from trapezoidal data streams. IEEE Trans Knowl Data Eng 28(10):2709–2723
32. Zhang Y, Chen Y, Yu H et al (2021) A feature adaptive learning method for high-density SEMG-based gesture recognition. Proc ACM Interact Mob Wearable Ubiquitous Technol 5(1):1–26
33. Zhang ZY, Zhao P, Jiang Y, et al (2020) Learning with feature and distribution evolvable streams. In: International Conference on Machine Learning, PMLR, pp 11317–11327