ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/338590631

Asynchronous dual-pipeline deep learning framework for online data stream classification

Article *in* Integrated Computer Aided Engineering · January 2020

DOI: 10.3233/ICA-200617

CITATIONS
19

READS
363

4 authors:

Pedro Lara-Benítez
Universidad de Sevilla
10 PUBLICATIONS **50** CITATIONS
SEE PROFILE

Manuel Carranza-García
Universidad de Sevilla
11 PUBLICATIONS **105** CITATIONS
SEE PROFILE

Jorge García-Gutiérrez
Universidad de Sevilla
48 PUBLICATIONS **530** CITATIONS
SEE PROFILE

José C Riquelme
Universidad de Sevilla
271 PUBLICATIONS **3,668** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Big Data for Electrical Fraud Detection View project

Applied Machine Learning View project

All content following this page was uploaded by Pedro Lara-Benítez on 19 November 2020.

The user has requested enhancement of the downloaded file.

The final publication is available at IOS Press through http://dx.doi.org/10.3233/ICA-200617
IOS Press – Preprint

Asynchronous dual-pipeline deep learning framework for online data stream classification

Pedro Lara-Benítez [a,*], Manuel Carranza-García [a,**], Jorge García-Gutiérrez [a] José C. Riquelme [a]
[a] *Division of Computer Science, University of Sevilla, Spain*

Abstract. Data streaming classification has become an essential task in many fields where real-time decisions have to be made based on incoming information. Neural networks are a particularly suitable technique for the streaming scenario due to their incremental learning nature. However, the high computation cost of deep architectures limits their applicability to high-velocity streams, hence they have not yet been fully explored in the literature. Therefore, in this work, we aim to evaluate the effectiveness of complex deep neural networks for supervised classification in the streaming context. We propose an asynchronous deep learning framework in which training and testing are performed simultaneously in two different processes. The data stream entering the system is dual fed into both layers in order to concurrently provide quick predictions and update the deep learning model. This separation reduces processing time while obtaining high accuracy on classification. Several time-series datasets from the UCR repository have been simulated as streams to evaluate our proposal, which has been compared to other methods such as Hoeffding trees, drift detectors, and ensemble models. The statistical analysis carried out verifies the improvement in performance achieved with our dual-pipeline deep learning framework, that is also competitive in terms of computation time.

Keywords: Classification, Convolutional Neural Network, Data streaming, Deep Learning, Evaluation, Online Learning

1. Introduction

Knowledge discovery from data streams has recently gained importance due to the enormous amount of data that modern devices collect at high speed. Models that deal efficiently with streams of data to provide real-time predictions are necessary in many fields such as machine fault detection [1], electricity demand prediction [2], financial data prediction [3], computer security [4], and health care [5]. An important challenge in the streaming scenario is performing online classification since its specific characteristics prevent from using traditional batch-learning techniques [6]. Data stream models learn incrementally using incoming data that cannot be stored, and have to be ready to assign a label whenever a new instance arrives.

Over the last decades, many novel classification algorithms have been proposed to improve the accuracy of traditional methods [7]. Promising results have been obtained by adapting these existing techniques for the streaming context. However, the recent advent of deep neural networks (DNNs) as the state-of-the-art for many problems opens an interesting research direction to aim for a higher performance [8]. DNNs are particularly suitable for data streaming due to their incremental learning nature and their capacity for solving dynamic non-linear problems [9]. Nevertheless, their application to high-velocity streams presents limitations due to the high computational cost of their learning procedure. Therefore, there is little research on the use of DNNs for data streaming, which do not usually appear as high-performing models in the literature [10].

Our aim in this study is to develop a framework that can use complex DNNs for data streaming classification, in which maintaining a high processing

*Corresponding author. E-mail: plbenitez@us.es.
**Corresponding author. E-mail: mcarranzag@us.es.

© 2020 – IOS Press and the authors. All rights reserved

rate is essential. This paper proposes a novel Asynchronous dual-pipeline Deep Learning framework for data streaming (ADLStream) that is designed to deal with the specific requirements of this scenario. Training and classification processes work simultaneously in two separated layers, hence the system is always ready to provide predictions. At the same time, the other process constantly updates the model in order to adjust it to changes in the incoming data distribution. This division allows reducing the computation time needed to deal with the instances while maintaining a high accuracy on classification. ADLStream is a general framework that could be used with any kind of deep learning (DL) model, regardless of its architecture.

As a case study to validate the performance of the proposed framework in a complex environment we have used convolutional neural networks (CNNs), which are especially indicated for dealing with data that has spatial or temporal structure [11]. For the experiments, we have simulated as streams a large number of time-series datasets from the UCR repository [12]. The performance of our proposal is compared in terms of accuracy and processing time to other popular streaming techniques such as Hoeffding trees and ensemble models. Furthermore, we have carried out several experiments with artificial datasets to evaluate the robustness of ADLStream when the properties of the stream change significantly over time, which is known as concept drift [13].

The main contributions of this work can be summarised as follows:

- ADLStream, a novel deep learning framework for data streaming classification.
- Asynchronous dual-pipeline architecture that reduces processing time of DL networks for data streaming by splitting training and classification tasks.
- A thorough experimental study, comparing ADLStream to several streaming techniques over more than 30 datasets.
- An analysis of the effects of different concept drifts on the performance of ADLStream.

The rest of the paper is organised as follows: Section 2 presents a review on related work; in Section 3 the materials used and the methods proposed in the study are described; Section 4 presents the experimental setup designed; Section 5 reports and discusses the results obtained; Section 6 presents the conclusions and possible future work.

2. Related work

Learning from data streams presents several challenges that prevent from directly using traditional data mining algorithms. Classifiers designed for static datasets typically require to iterate several times over the instances, hence they are not suitable for dealing with data arriving at high speed. An effective data stream classification model should be able to extract the relevant information with just a single pass on the instances, and using a limited amount of time and memory, which increases the complexity of the learning procedure [14]. Another important aspect when dealing with streams is the type of learning framework, which depends on the availability of labels for the incoming examples. In this work, we follow a completely supervised learning framework, where the true class of all examples is always known posterior to classification. Therefore, the studies covered in this section all work under the same assumption, which is very common in the literature. However, there are also studies considering other scenarios, such as learning with delayed labelling or semi-supervised learning (i.e. only a fraction of incoming examples have labels) [15]. Regardless of the framework considered, the classical division of batch learning techniques into training and predicting phases has to be shifted to an online approach in which both tasks are interleaved, given that the stream may be infinite. Furthermore, the update of the models has to account for concept drifts since the data distribution may change over time. These variations in the boundaries between classes need to be detected in order to carry out efficient retraining of the models [16].

Considering the above-mentioned characteristics of data streaming, there have been efforts to adapt for this context many existing techniques such as support vector machines [17], k-nearest-neighbors [18], rule-based classification [19], and Bayesian classification [20]. However, due to their simplicity and fast processing rate, one of the most popular approaches has been to develop adaptive algorithms based on decision-tree methods [21]. The Very Fast Decision Tree (VFDT) method, based on the Hoeffding tree principle, was the earliest to be designed specifically for stream classification [22]. Hoeffding bounds help to incrementally build a model similar to what a batch learner would produce. They split a node only when there is statistical significance between the current best attribute and the others. Following this concept, new proposals such as Hoeffding Option Trees (HOT) were de-

signed. In HOT, each example can update a set of option nodes instead of a single leaf, leading to a representation of multiple trees as separate paths [23]. Other algorithms more suitable for time-changing data streams were later developed, such as Hoeffding Adaptive Trees (HAT) [24], that reduce the effect of past data by using sliding windows and replacing branches.

Furthermore, an important area of research on data streams has been the development of real-time monitoring techniques that detect concept drifts over incoming data [25]. There exist a wide variety of concept drifts, that are usually simplified in four types [26]: *abrupt* drift, when there is a sudden shift from one concept to another; *incremental* drift, which implies going through many different intermediate concepts while drifting to a new concept; *gradual* drift, when the stream oscillates between two concepts before drifting completely; and *recurrent* drift, that happens when the stream drifts to a previously seen concept. Additionally, in real-world streams the distribution of classes may change over time, leading to an imbalance that increases the difficulty of classification.

To deal with these problems, the concept drift detection method (DDM) was presented in [27], which helps control the accuracy of predictions of the learning model. DDM can be used with a wrapper on a classifier by creating a new model with recent examples whenever a significant change in the class distribution is detected. This technique proved to be independent of the underlying classification algorithm used, and other studies have developed similar proposals, such as Early Drift Detection Method (EDDM) [28] or a nonparametric method based on Hoeffding's bounds [29]. Although drift detection methods would allow using batch algorithms as the base learner, their combination still faces the computational cost of rebuilding models from scratch many times. Therefore, alternative approaches to handle concept drifts have considered using classifiers that are adaptable to change on data. This behaviour can be implemented by using a sliding window or using incremental or online learners, such as neural networks, which can keep their weights updated by processing each instance only once [30].

More recently, ensemble methods have gained relevance since they can improve the robustness of single classification models and allow an easier adaptation to variations in the distribution of the data [15]. Online approaches to traditional bagging and boosting algorithms were designed in [31], in which the incoming samples are weighted using the Poisson distribution for carrying out the model updating. Later, several studies have proposed modifications to these methods in order to improve randomization, such as: Adaptive-Size Hoeffding Trees (ASHT), that builds an ensemble of trees of different sizes [24]; ADWIN Bagging, that uses adaptative windows to detect concept drifts and eliminate ensemble members with poor performance [24]; and Leveraging Bagging, which increases resampling and uses output detection codes [32].

The latest studies on data stream classification have followed the approach of combining the ideas of ensemble models and concept drift detection. The Adaptive Random Forest (ARF) algorithm for classification of evolving data streams was proposed in [33]. ARF improves resampling methods to add diversity and uses adaptive operators to cope with concept drifts. Furthermore, the independence of its components allows for a parallel implementation that reduces processing time without degrading performance.

In [34] the authors propose an improvement applicable to any online ensemble that adds possible abstentions in the voting process. Only classifiers that perform above a certain confidence level are allowed to vote, which proved to be particularly useful in noisy data streams. The same authors of this work developed later the Kappa Updated Ensemble (KUE) [16]. This algorithm is driven by the Kappa statistic and uses weighted voting from a pool of classifiers to provide predictions. In KUE, each component deals with a random dimensionality, as opposed to ARF in which the subspace size is fixed.

Other studies have recently proposed novel drift detection methods: [35] presents a methodology to detect different concept drifts by selecting dynamically the most competent ensemble member to classify each incoming example; and [36] develops the Enhanced Concept Profiling Framework (ECPF), which focuses on improving speed by reusing previously trained classifiers when recurrent concept drifts are detected.

With regard to the application of DL techniques to data stream classification, few studies have presented deep neural network models for this context. Existing work is limited to the use of MultiLayer Perceptron (MLP) [37] and ensemble methods using them as the base classifiers [38]. Other proposals have considered placing traditional classifiers on top of simple Deep Belief Networks (DBNs) [10]. Although more sophisticated DL models are currently state-of-the-art for many problems in the batch setting, they have not been explored in the data streaming literature. Their high computational complexity has so far been a severe restriction to make them suitable for a high-velocity

stream scenario. In particular, architectures such as recurrent or convolutional networks have provided better performance than MLPs or ensemble models with grid-like data such as images or time series [39]. The above mentioned related works mostly validate their proposals using data without an inner temporal or spatial dependence. Therefore, building DL models that deal efficiently with this kind of data in the streaming context is a research area that has yet to be addressed.

Furthermore, there are recent proposals that develop asynchronous frameworks to reduce processing time in several domains: in [40] a system using coupled deep neural networks is proposed for reinforcement learning; and a genetic programming rule-based classifier for data streaming that can run asynchronously is presented in [41]. This trend calls for the development of similar proposals using deep learning, since the performance of popular streaming ensemble models can be enhanced by complex DL models if they have an adequate processing rate.

3. Materials and methods

3.1. Description of datasets

A total of 29 different time-series datasets have been used for this study, which have been obtained from the public UCR repository [12]. These datasets have already been used in the literature to simulate streams for different applications such as stream clustering [42], anomaly detection [43] or density estimation of data streams [44]. All the datasets considered are composed of instances that are one-dimensional time series, hence they have an inner grid-like structure. The length indicates the number of points that each individual instance arriving at the stream has. The particular characteristics of each dataset are presented in Table 1. Only those that have a minimum of 1000 instances have been used, in order to reproduce a realistic streaming scenario. As can be seen, the datasets are different in terms of the length of the series and the number of classes considered. They cover six different domains that are the following:

- Sensor: Readings from sensors in areas such as process control measurement (Wafer), weather monitoring (MoteStrain), car engines (Ford), human voice recognition (Phoneme), or animal sounds (InsectWingBeat).
- ECG: Electrocardiogram records for tasks such as detecting heart problems (TwoLeadECG) or identifying different people (CinCEGTorso).
- Motion: Captures of gestures generated from accelerometers (UWaveGestureLibrary) and digital pen traces (Pendigits).
- Image: Outlines of images that are mapped onto a one-dimensional series, such as faces (FaceAll), hands (HandOutlines) or shapes (ShapesAll).
- Device: Data from daily electrical power consumption (ElectricDevices).
- Simulated: Artificially generated time series for problems such as signal processing (Mallat) or pattern recognition (TwoPatterns).

A more detailed explanation of each type with figures can be found at [12]. The use of time-series data allows us to evaluate the behaviour of complex and time-consuming models in our framework, such as CNNs. Moreover, the variability of the datasets selected is essential in order to prove the capacity of generalisation of our proposal.

Table 1
Datasets used for the study.

#	Dataset	Instances	Length	Classes	Type
1	TwoPatterns	5000	128	4	SIMULATED
2	CinCECGtorso	1420	1639	4	ECG
3	TwoLeadECG	1162	82	2	ECG
4	Wafer	7164	152	2	SENSOR
5	Pendigits	10992	16	10	MOTION
6	FacesUCR	2250	131	14	IMAGE
7	Mallat	2400	1024	8	SIMULATED
8	FaceAll	2250	131	14	IMAGE
9	Symbols	1020	398	6	IMAGE
10	ItalyPowerDemand	1096	24	2	SENSOR
11	ECG5000	5000	140	5	ECG
12	MoteStrain	1272	84	2	SENSOR
13	NonInvasiveFetalECGThorax1	3765	750	42	ECG
14	NonInvasiveFetalECGThorax2	3765	750	42	ECG
15	SwedishLeaf	1125	128	15	IMAGE
16	FordA	4921	500	2	SENSOR
17	Yoga	3300	426	2	IMAGE
18	UWaveGestureLibraryX	4478	315	8	MOTION
19	FordB	4446	500	2	SENSOR
20	ElectricDevices	16637	96	7	DEVICE
21	UWaveGestureLibraryY	4478	315	8	MOTION
22	UWaveGestureLibraryZ	4478	315	8	MOTION
23	HandOutlines	1370	2709	2	IMAGE
24	InsectWingbeatSound	2200	256	11	SENSOR
25	ShapesAll	1200	512	60	IMAGE
26	MedicalImages	1141	99	10	IMAGE
27	PhalangesOutlinesCorrect	2658	80	2	IMAGE
28	ChlorineConcentration	4307	166	3	SIMULATED
29	Phoneme	2110	1024	39	SENSOR

3.2. ADLStream framework

The asynchronous dual-pipeline framework presented in this section aims to provide a general deep learning-based architecture to achieve high performance in data streaming classification. This novel framework improves processing rate and allows to use efficiently deep learning models, such as convolutional or recurrent networks, for data arriving at high speed. Similarly to most of the existing literature on data streaming, in this work we consider a fully supervised scenario in which the labels are immediately available for all processed examples and can be used to update the model [15]. Accordingly, the data stream can be defined as a sequence of labelled instances $\{x_t, y_t\}$ for $t = 1, 2, ..., T$. The task of the proposed framework is to provide quick predictions $\hat{y}_t$ for each incoming example x_t based on what it has learned from the data that it has seen so far. Later, when the real labels are available (immediately after the example has been processed), they are used to update the model.

The design of the ADLStream system is fully illustrated in Fig. 1, in which it can be seen that predicting and training phases are separated into two different layers. This split allows making predictions at any time while keeping the DL model constantly updated, and reduces the computation time compared to the traditional sequential scheme. The logic behind the complete system is described in Algorithm 24. Examples arriving online (i.e. instance by instance) from the stream are sent to both processes that work asynchronously. When the predicting process receives an instance, it is instantly classified using a previously trained model. Given that DNNs are significantly faster for predicting compared to the training procedure, the prediction layer is always ready to immediately classify incoming data. In contrast, the training layer is more time-consuming and works by saving the instances received and grouping them in batches. Once a specific number of batches are collected, they are fed to the DL model in order to carry out the training procedure through back-propagation. The new set of weights obtained when the training is completed is passed to the predicting process to maintain both models updated. With this approach, it is assured that each individual example is tested before it is used to train the network. More specifically, at least $s \times b$ instances are classified before updating the model with these instances, where s is the batch size and b is the number of batches fed in a training iteration.

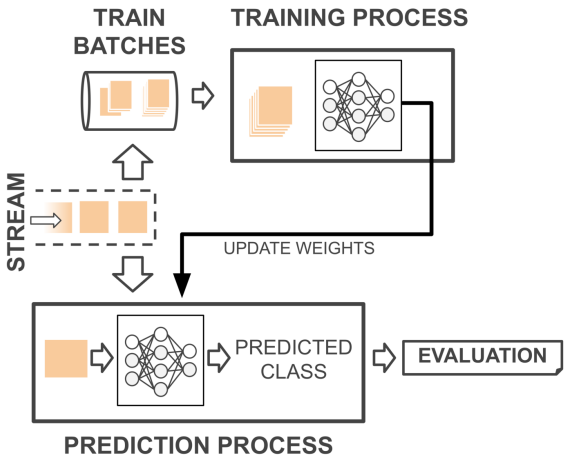

Fig. 1. ADLStream framework proposed in the study to perform classification over data streams.

Although ideally both layers could deal concurrently with all received instances, the significant difference between training and predicting execution time in DNN models poses problems given the high rate of arrival of examples. The capacity of the system for processing all data and re-training the model would depend on several factors such as the stream speed, data topology or computer specifications. However, the optimal situation in which every instance is used for training would imply that an increasingly larger queue of awaiting instances would be formed in the training layer. To solve this problem we propose to specify a limited number of instances to train with, while the rest are discarded in order not to overload the buffer. When the number of examples in the queue reaches a certain value $m = s \times b$ (batch size times number of batches fed), the system uses a sampling technique to select the training set for each iteration.

In the streaming scenario, the distribution of the incoming data tends to evolve over time, which implies that recent instances are more relevant to describe the state of the stream than older ones [45]. A first approach to deal with this issue could be to use a sliding window with the most recent instances [46]. However, with this solution, useful information about the distant historical behaviour of the stream can be lost. For this reason, a common approach is to consider a weighted sampling algorithm to regulate the choice of instances from the stream. In our proposed framework, a biased reservoir sampling method using the A-Chao algorithm is implemented [47]. This technique maintains a reservoir of samples with associated weights

Algorithm 1: ADLStream framework

Arguments:

Stream: Stream to be analysed
M: Deep learning model
Q: Training sample sequence
W: Sequence of weights of the training sample sequence
s: Batch size
b: Number of batches fed
e: Instance
λ: Decay factor for weight update

```
1  Procedure Main(Stream):
2      s ← initialise batch size
3      b ← initialise number of batches fed to
4            training model
5      m ← s × b
6      M ← initialise deep learning model
7      Q ← initialise empty sequence
8      W ← initialise empty sequence
9      for e in the first s instances in Stream do
10         Q.append(e)
11         W.append(1)
12     end
13     M ← M.train(Q)
14     Asynchronous P1
15         for instance e in Stream do
16             o ← M.predict(e)
17             Q ← sampling(Q, W, e, m)  [Alg. 2]
18         end
19     Asynchronous P2
20         while P1 is not finished do
21             if Q.size >= s then
22                 M ← M.train(Q)
23             end
24         end
```

Algorithm 2: Sampling

Input : Q : Training sample sequence
W : Sequence of weights of the training sample sequence
e: Instance
m: Maximum size of sample sequence

Output: Q : Updated sample sequence
W : Updated sequence of weights

```
1  w_e ← 1
2  if Q.size < m then
3      Q.append(e)
4      W.append(w_e)
5  else
6      Calculate probability p_e = w_e/(∑_{i=1}^{m} w_i)
7      Decide randomly if item e will be inserted
         into the sample with a probability of p_e
8      if Yes then
9          Delete a random item from the sample
10         Q.append(e)
11         W.append(w_e)
12     end
13 end
14 Update sample weights with decaying factor
     (W ← W × λ)
```

and performs probabilistic insertions and deletions on arrival of new stream points. As explained in Algorithm 2, when a new item is examined, its relative weight is used to randomly decide whether it will be inserted into the reservoir. In case it is selected, one random item inside the reservoir is deleted and the new instance is added. The weights of the instances belonging to the reservoir are updated each iteration with a decaying factor, which is fixed to $\lambda = 0.98$.

Maintaining a completely unbiased sample is not practical given that the evolution of the stream may lead to a reservoir filled with past irrelevant history. Therefore, it is desirable to bias the sampling to represent more recent behaviour of the stream [48]. Accordingly, newly arrived items are assigned more weight in order to increase their probability of belonging to the random sample.

With this reservoir approach, the framework keeps a fixed-size window containing a representative sample of the recent instances, which are fed to the training layer whenever an iteration is completed. In the experimental study, a grid search with different combinations has been performed in order to find suitable values for s and b, which are the parameters that define the size of the reservoir sampling window.

3.2.1. CNN Architecture

The ADLStream framework can be used with any type of DL architecture, but as a case study for this work we have used CNNs, which are particularly suitable for dealing with temporal data [49]. The objective of the selected DL model is to perform 1D convolution over the time-series examples in order to automatically extract abstract features that represent the internal structure of the data [50]. The proposed 1D-CNN architecture, illustrated in Fig. 2, is inspired by the study presented in [51]. As can be seen, the network is composed of a block of several one-dimensional dilated convolutional layers with pooling followed by a fully-connected Multi-Layer Perceptron (MLP), which performs the classification.

In the convolutional layers, the one-dimensional input array is convolved with filters of kernel size k in order to create several features maps that are passed through a non-linear activation function. The use of dilation in the convolution by skipping certain elements in the input allows expanding the receptive field of the network exponentially. This expansion generally improves performance since there is no loss of resolution, while it maintains the same computation and memory costs [52]. The complete convolution process can be formulated as follows [53]:

$$a_{ln}^{x} = g\left(\sum_{m}\sum_{k=0}^{K_l-1} w_{lnm}^{k} a_{(l-1)m}^{(x+(k\times d))} + b_{ln}\right) \qquad (1)$$

where a_{ln}^{x} is the value of a neuron at position (x) of the n-th feature map in the l-th layer; m indexes over the set of features maps in the $(l-1)$-th layer connected to the current feature map; K_l is the width of the convolutional kernel; w_{lnm}^{k} stands for the weight of position (k) connected to the m-th feature map; d is the dilation factor of the convolution; and b_{ln} is the bias term. Rectified Linear Units (ReLU) layers are used to compute the non-linear activation function ($g(x) = max(0, x)$) on the obtained feature maps in order to improve the network convergence speed [54]. After the convolution, max-pooling layers are used to progressively reduce the spatial size of the feature maps. Input maps are down-sampled by selecting the maximum activation value for each non-overlapping neighbourhood of fixed size. The reduction of parameters achieved with the pooling operation decreases the computation cost of the network and controls over-fitting by selecting superior invariant features [55]. The last block of the network is composed of dense layers that have full connections to all activations in the previous layer. These layers combine all the convoluted feature maps to create a flattened feature vector. Finally, the classification of the instances is performed according to the softmax output probabilities of the last dense layer.

Table 2

CNN architecture. The values of f and c are the number of features of the instances and the number of classes respectively.

Layer	Type	Neurons & # Maps	Kernel
0	Input	f neurons	
1	Convolutional + ReLU	$f \times 32$	7
2	Max-Pooling	$f/2 \times 32$	2
3	Convolutional + ReLU	$f/2 \times 64$	5
4	Max-Pooling	$f/4 \times 64$	2
5	Convolutional + ReLU	$f/4 \times 128$	3
6	Max-Pooling	$f/8 \times 128$	2
7	Dense + ReLU	512 neurons	
9	Dropout	512 neurons	
10	Dense + ReLU	128 neurons	
11	Dropout	128 neurons	
12	Softmax	c neurons	

Regarding the specific number of layers and feature maps used in our model, a more detailed description of the proposed architecture is presented in Table 2. Similarly to well known CNN architectures such as VGG [56], the number of filters in consecutive convolutional layers is increased in order to extract more detailed features from the richer representations obtained. Therefore, the convolutional layers have 32, 64, and 128 filters respectively. Due to the decreasing spatial resolution of the max-pooled feature maps, the kernel size of the convolutional layers are set to 7, 5 and 3 respectively, and with dilation rate 3. The features extracted from the convolutional block are then transferred to the fully connected block that has two dense layers of 512 and 128 neurons, and a softmax layer that has as many neurons as the number of classes considered. Furthermore, dropout layers with a small rate (0.2) are used after the fully connected layers. Dropout has proven to be an effective regularisation method since it enhances the capacity of generalisation of the network by deactivating different neurons on each training iteration [57]. Moreover, another element that we have considered to prevent overfitting is not to use a very large batch size. Deep networks converge to sharp minimizers when trained using large batches and they lose generalisation abilities [58]. For the implementation of the proposed convolutional network, the Keras framework has been used [59].

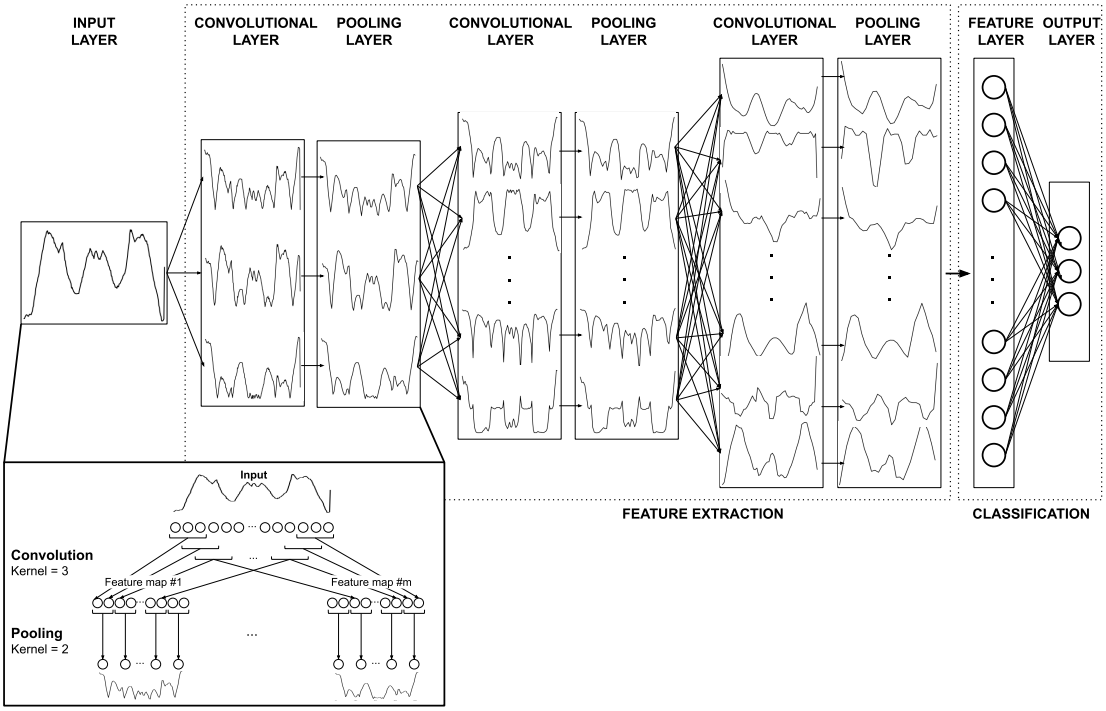

Fig. 2. CNN Architecture for time-series data streaming classification.

4. Experimental setup

This section presents the comparative study carried out to evaluate the performance of the ADLStream framework. The experimental process is based on a statistical analysis, with the results obtained for all datasets, that compares our proposal with several state-of-the-art algorithms for data stream classification. For simulating the streaming, the Apache Kafka platform has been used, since it has emerged as the best stream-processing tool in terms of efficiency of data management [60]. The Kafka server allows reproducing a real data streaming scenario in which instances are constantly arriving, hence the evolution over time of the accuracy of the models can be analysed.

4.1. Models used for comparison

Numerous existing techniques have been considered for the study, with the aim of fully covering all families of algorithms that have been proposed in the literature for this problem. Table 3 presents the different classifiers that have been evaluated, grouped by family, and with the abbreviations that would be used throughout the paper. All selected models are implemented in popular open-source frameworks such as MOA [61] and Scikit-learn [62].

4.2. Evaluation procedure

Unlike in traditional batch learning setup, cross-validation cannot be used as the evaluation technique in a streaming setup since unlimited data tends to make it too expensive computationally. In the online streaming setting, the objective is to capture how the accuracy evolves over time, hence one possible alternative is to perform holdout evaluation over independent sets periodically. Although ideally holdout could provide the best estimation of the accuracy on recent data, it is not practical for real scenarios where obtaining sufficient recent data for testing may be challenging. Therefore, the most extended solution is to maximise the use of available data with an interleaved test-then-train approach [63]. With this method, every instance is used to test the model before it is used for training. This implies that the model is always tested with unseen examples, which allows to incrementally update the accuracy taking into account all incoming instances. However, since streaming learning algorithms are supposed to evolve over time due to changes in the data distribution, more recent instances should be given more importance in order to provide a reliable error estimation. Accordingly, the predictive sequential (or prequential)

Table 3
Classifiers used for the comparative study.

Classifier	Abbreviation	Family
Majority Class	MC	Baseline
Active Classifier	AC	Bayesian classifiers
Naive Bayes	NB	
Bernoulli Naive Bayes	BNB	
Decision Stump	DST	Decision Tree
Hoeffding Tree	HT	
Hoeffding Adaptive Tree	HAT	
Hoeffding Option Tree	HOT	
Adaptive HOT	ADHOT	
Random Hoeffding Tree	RHT	
Adaptive-Size HT	ASHT	
Stochastic Gradient Descent	SGD	Function classifiers
Stochastic Pegasos	SPEG	
PassiveAggressiveClassifier	PAC	
Perceptron	P	
Single Classifier Drift	SCD	Drift classifiers
Incremental Online Bagging	BA	Meta classifiers Bagging
Bagging Adaptive-Size HT	B-ASHT	
Bagging using Adwin	BA-AD	
Leveraging Bagging	LBAG	
Adaptive Random Forest	ARF	
Kappa Updated Ensemble	KUE	
Incremental Online Boosting	BO	Meta classifiers Boosting
Online Coordinate Boosting	OCBO	
Boosting using Adwin	BO-AD	
Multi Layer Perceptron	MLP	Neural Networks
Our proposal	**ADLStream**	

evaluation method implements this idea of decreasing the relevance of past examples for the evaluation.

[64] proposes the use of a forgetting mechanism for computing prequential accuracy by using sliding windows or decaying factors. When using a sliding window of size ω, the prequential accuracy is computed over the ω most recent instances. If a fading factor is used instead, the previous errors are weighted using a decay factor α. Both approaches present the problem of selecting the optimal value for the parameters, but using a decaying factor provides an advantage. While sliding windows need to keep the last ω instances in memory, the fading factors are memory-less and allow to update the prequential accuracy at time i by using just the error at the previous step $(i-1)$. Therefore, in our study, we have decided to use the decaying factor version with α of 0.99, which is a commonly used value [65]. The process of computing the prequential accuracy can be formulated as shown in Equation 2, where L is the loss function and o and y are the real and expected output respectively.

$$P_\alpha(i) = \frac{\sum_{k=1}^{i} \alpha^{i-k} L(y_k, o_k)}{\sum_{k=1}^{i} \alpha^{i-k}} = \\ = L(y_i, o_i) + \frac{1}{\alpha} P_\alpha(i-1) \quad (2)$$

With regard to the specific measure used for evaluation, the standard accuracy is not the most appropriate option for the streaming context, since the number of instances for each class can change and lead to an imbalanced distribution [66]. Therefore, the Kappa statistic is a more reliable measure for estimating the performance of data streaming classification algorithms. Equation 3 presents how to compute the Kappa value, where p_0 is the prequential accuracy and p_c is the hypothetical probability of chance agreement.

$$k = \frac{p_0 - p_c}{1 - p_c} \quad (3)$$

Once the accuracy values for each method have been obtained, it is necessary to carry out a statistical analysis in order to correctly compare the performance of different classifiers. Since our study compares multiple classifiers over multiple datasets, the Friedman test is the recommended method [67]. This non-parametric test allows to detect global differences and provides a ranking of the algorithms. In the case of obtaining a p-value below the significance level (0.05), the null hypothesis (all algorithms are equivalent) can be rejected, and we can proceed with a post-hoc analysis in order to perform a pair-wise comparison. In the post-hoc step, each technique is compared to the best method (control algorithm) in order to determine whether there is statistical significance between their performance. In our study, the Holm's procedure will be used, with the aim of rejecting the null hypothesis for each pair. Obtaining satisfactory p-values in this test would allow us to verify the hypothesis of improved performance of the ADLStream framework over the rest of the methods considered. The online STATService [68] tool has been used for conducting the statistical analysis.

5. Results

In this section, we present and discuss the results obtained from the experiments carried out. The prequential Kappa results of all methods considered are reported, followed by the statistical evaluation performed. Moreover, given the importance of the speed in a data streaming scenario, the computation time of all algorithms is also analysed. Finally, we present a study on the effect of different concept drifts. For all tests, we have used a computer with an Intel Core i7-770K CPU and two NVIDIA GeForce GTX 1080 8GB GPU.

5.1. Parameter optimisation

Instead of setting arbitrary values, a grid search has been conducted to select the training hyper-parameters of the ADLStream framework, as it was mentioned in Section 3.2. Fig. 3 illustrates the grid search performed with values ranging from 10 to 120 for both parameters, the batch size (s) and the number of batches fed in each training iteration (b). The highest average accuracy considering all the datasets studied has been obtained with values $s = 90$ and $b = 60$, which has been set as a general configuration for all experiments. As can be seen, both accuracy and processing time tend to stabilise for a batch size higher than 30, regardless of the number of batches fed. Very similar results are obtained for all combinations with s above 30, which shows that there is flexibility on the choice of the parameters of the proposed framework. For values of s below 30, the performance decreases significantly, which indicates that a very small batch size could prevent the model from adapting properly to the evolution of the stream. Since instances are only used once, the update could focus excessively on the specific characteristics of just a few examples, being unable to learn about the changes in the overall data distribution.

5.2. Prequential evaluation results

In order to examine the behaviour of each method over each dataset, the prequential Kappa statistics using decaying factors are collected and analysed. Fig. 4 presents a heat map illustrating the results of all techniques. In the map, the methods are ordered with regard to their average overall accuracy, hence best models are at the left-hand side of the figure. As can be seen, our ADLStream proposal achieves a high level of accuracy for almost all the datasets considered. These

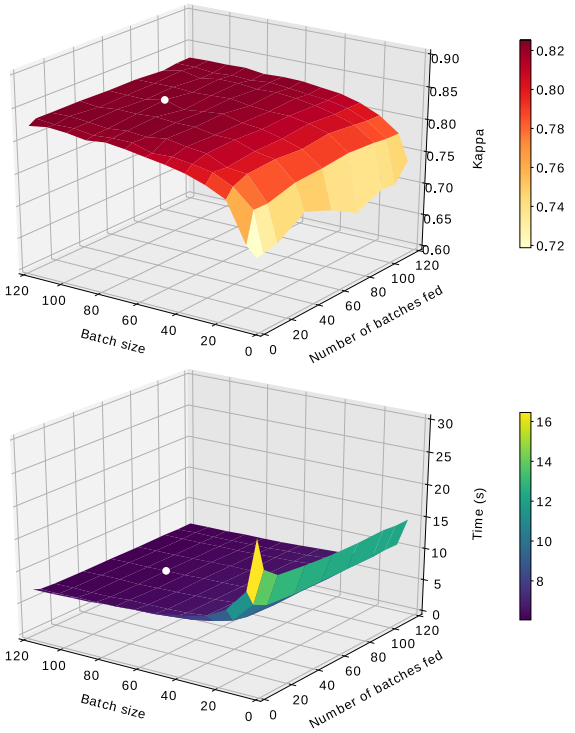

Fig. 3. Kappa and processing time results obtained with the ADLStream framework depending on batch size and the number of batches fed. The white dot represents the chosen values for the parameters.

results demonstrate the suitability of the use of CNNs for the time-series data streaming scenario. Moreover, the high-quality results obtained by the Multi-Layer Perceptron (MLP) also demonstrates the power of neural network-based techniques compared to the rest of classifiers. The adaptive random forest method (ARF) was the best ensemble model, obtaining the second position in average performance. Other methods with good results are those related to the Bayesian (NB) and drift classifiers (SCD) families. Concerning the decision trees family, the adaptive version ASHT provides the higher accuracy, closely followed by the standard HT. Lastly, the rest of the ensemble models (bagging and boosting) and the function classifiers are the classifiers with the poorest performance.

To provide a more detailed analysis, the results obtained with a subset of the top-performing classifiers of different families are reported in Table 4 and illustrated with a box-plot in Fig. 5. Our proposal outperforms the rest of the methods for almost all datasets, and there are particular cases that are worth mentioning. In the last four datasets, the methods from the lit-

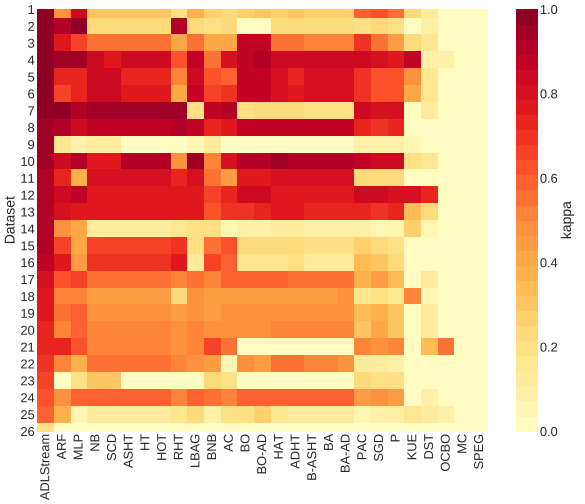

Fig. 4. Heat map of the prequential Kappa results obtained for each method over each dataset. The numbering of the datasets corresponds to the ordering presented in Table 1.

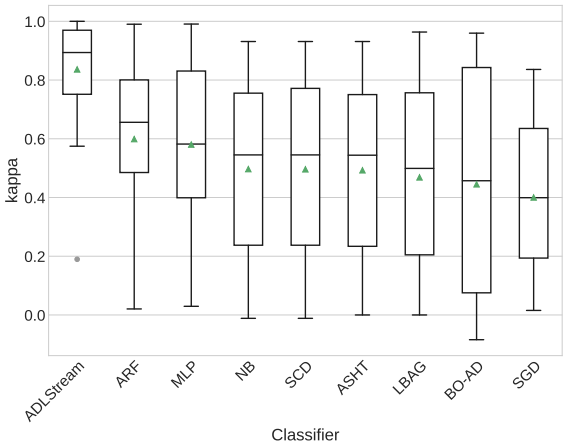

Fig. 5. Box plot summarising the Kappa statistics results obtained with the top-performing methods of each family.

erature struggle to achieve good performance. As an example, for the *ChlorineConcentration* dataset, ADLStream leads the ranking with an accuracy of 0.948 while the second method achieves just 0.149. On the other hand, the tree-based ensemble ARF got a better result than ADLStream in two datasets (*Mallat* and *HandOutline*). However, the difference in accuracy between both methods in these datasets is not significant as it is less than 0.02.

Another important aspect is that ADLStream using CNNs stands as the most reliable method since it shows less variability of results, as can be seen in the box-plot. A further visual comparison between the top-performing techniques is shown in Fig. 6. The increase in performance for all datasets achieved with the novel proposed method proves its capacity of generalisation. Using a constant architecture and parametrisation, ADLStream has achieved a high level of accuracy for all cases, regardless of the different characteristics of the streams in terms of length of the series and number of classes. In ADLStream, the CNN is trained a lower number of times and with fewer instances than the rest of methods, due to the grouping in batches and the examples discarded. Nevertheless, this fact does not have a detrimental effect on the performance, since CNNs can rapidly converge to an optimal set of weights even using fewer data.

5.3. Statistical analysis

The Kappa statistic results obtained from the experiments have to be analysed through a statistical test to correctly verify the hypothesis of improved performance of our proposal. The global ranking obtained from applying the Friedman Test is presented in Table 5. As expected, the ADLStream model leads the ranking with a high difference in score with respect to the second method, which is ARF. MLP ranks in the third position and has a similar behaviour compared to NB and SCD. Finally, decision trees and ensembles obtain a lower score given their poorer performance. The null hypothesis can be rejected since the p-value obtained (<0.001) is below the significance level (0.05). Therefore, we can proceed with the post-hoc analysis to perform a pair-wise comparison.

For carrying out the Holm's procedure, ADLStream is set as the best method and compared to the rest of the algorithms individually. The results obtained for this step are displayed in Table 6, which reports the adjusted α and p-values for each method. As can be seen, all null hypothesis can be rejected since the p-values are always below the corresponding significance level. Accordingly, it can be stated that there is a statistical significance in the differences between the performance of ADLStream and the other methods considered. Therefore, we can confirm our initial hypothesis that the proposed DL framework provides an important increase in accuracy for our data streaming classification study.

Table 4
Prequential Kappa results for the best classifiers of different families over all datasets.

#	Dataset	ADLStream	ARF	MLP	NB	SCD	ASHT	LBAG	BO-AD	SGD
1	TwoPatterns	**1.000**	0.489	0.832	0.290	0.290	0.290	0.380	0.304	0.615
2	CinCECGtorso	**0.997**	0.924	0.982	0.237	0.237	0.234	0.225	0.000	0.222
3	TwoLeadECG	**0.996**	0.762	0.643	0.545	0.545	0.544	0.537	0.858	0.559
4	Wafer	**0.997**	0.982	0.991	0.194	0.192	0.356	0.963	0.960	0.542
5	pendigits	**0.986**	0.950	0.938	0.824	0.784	0.850	0.867	0.909	0.800
6	FacesUCR	**0.974**	0.656	0.723	0.842	0.842	0.777	0.861	0.874	0.635
7	Mallat	0.970	**0.990**	0.911	0.931	0.931	0.931	0.205	0.216	0.787
8	FaceAll	**0.977**	0.735	0.729	0.829	0.829	0.736	0.849	0.869	0.638
9	Symbols	**0.950**	0.893	0.831	0.890	0.890	0.889	0.887	0.890	0.709
10	ItalyPowerDemand	**0.939**	0.841	0.898	0.778	0.778	0.916	0.932	0.918	0.836
11	ECG5000	**0.895**	0.856	0.877	0.750	0.772	0.750	0.752	0.843	0.833
12	MoteStrain	**0.895**	0.800	0.781	0.755	0.755	0.754	0.757	0.717	0.698
13	NonInvasiveFetalECGThorax1	**0.893**	0.668	0.399	0.650	0.650	0.650	0.191	0.215	0.215
14	NonInvasiveFetalECGThorax2	**0.888**	0.763	0.462	0.681	0.681	0.681	0.131	0.161	0.292
15	SwedishLeaf	**0.897**	0.731	0.392	0.799	0.799	0.798	0.794	0.777	0.241
16	FordA	**0.832**	0.211	0.429	-0.012	-0.012	0.104	0.058	0.045	0.050
17	Yoga	**0.894**	0.485	0.407	0.129	0.129	0.126	0.208	0.075	0.042
18	UWaveGestureLibraryX	**0.798**	0.626	0.644	0.569	0.569	0.569	0.569	0.575	0.453
19	FordB	**0.808**	0.128	0.338	0.046	0.046	0.067	0.032	0.008	0.021
20	ElectricDevices	**0.768**	0.526	0.526	0.456	0.456	0.456	0.468	0.457	0.194
21	UWaveGestureLibraryY	**0.752**	0.542	0.582	0.469	0.469	0.469	0.469	0.476	0.378
22	UWaveGestureLibraryZ	**0.720**	0.528	0.572	0.504	0.504	0.504	0.507	0.500	0.399
23	HandOutlines	0.717	**0.720**	0.634	0.533	0.533	0.533	0.530	-0.084	0.475
24	InsectWingbeatSound	**0.632**	0.473	0.576	0.587	0.587	0.587	0.589	0.588	0.465
25	ShapesAll	**0.696**	0.512	0.372	0.566	0.566	0.565	0.499	0.456	0.120
26	MedicalImages	**0.669**	0.020	0.191	0.307	0.307	0.020	0.020	0.020	0.199
27	PhalangesOutlinesCorrect	**0.575**	0.377	0.060	0.134	0.134	0.133	0.245	0.277	0.072
28	ChlorineConcentration	**0.948**	0.149	0.082	0.122	0.122	0.001	0.063	0.001	0.099
29	Phoneme	**0.190**	0.032	0.029	0.003	0.003	0.000	0.000	0.000	0.015

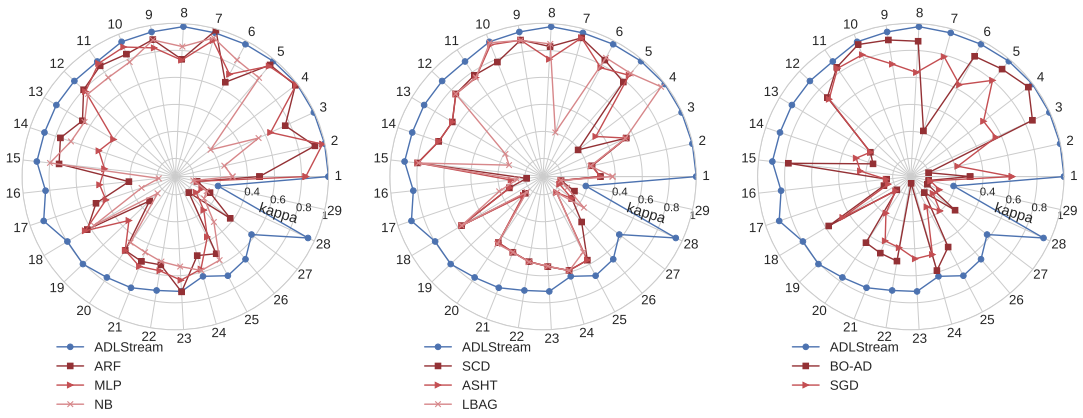

Fig. 6. Radar plot comparing the performance of different techniques with our proposed ADLStream framework.

Table 5
Friedman Test Ranking.

Friedman Test Ranking	
ADLStream	1.069
ARF	3.534
MLP	4.414
NB	5.362
SCD	5.397
BO-AD	5.810
LBAG	5.845
ASHT	6.155
SGD	7.414

Table 6
Holm's post-hoc analysis.

PostHoc Analysis			
Classifier	p	z	Holm's α
SGD	<0.001	8.822	0.006
ASHT	<0.001	7.072	0.007
LBAG	<0.001	6.641	0.008
BO-AD	<0.001	6.593	0.010
SCD	<0.001	6.017	0.013
NB	<0.001	5.969	0.017
MLP	<0.001	4.651	0.025
ARF	0.001	3.428	0.050

5.4. Computation time analysis

Given the high rate of arrival of instances in the streaming scenario, analysing the average time that each method takes to process an instance (hereinafter referred as processing time) is essential. The processing time for the set of best classifiers of each family over each dataset is reported in Table 7. As can be seen in the comparison shown in Fig. 7, the processing time of the proposed ADLStream model is competitive with respect to the rest of the algorithms considered. Logically, simpler models such as MLP, NB or HT have a higher processing rate, but the small increase in processing time is compensated with significantly higher accuracy, as it was seen in Section 5.2. Furthermore, the processing time of ADLStream is lower than other ensemble methods such as LBAG and BO-AD, which are considered state-of-the-art ensemble techniques in the data streaming literature [69].

5.5. Comparison between sequential and dual-pipeline approach

In this subsection, we present a comparison with a sequential approach, in order to illustrate the importance of the asynchronous dual-pipeline architecture designed for the DL model. The novel ADLStream framework introduced in this study aims to tackle the processing time problem of complex DNNs models in the data streaming context. Table 8 presents a comparison in terms of time and performance between a sequential CNN and the proposed ADLStream. In the sequential scheme, which is significantly slower, every instance is used to train the model after it has been given a prediction. Therefore, all examples are classified with a model recently updated with all instances seen so far. In contrast, in the dual-pipeline framework, train and test phases work concurrently, obtaining an average speed-up of 42 times faster than the sequential model. Thanks to the parallelisation, the time to process an instance corresponds just to the prediction time, while the model is trained in a separate layer as many times as possible. Depending on the speed of the data stream and the available computing resources, ADLStream may classify more instances with a non-updated model. Theoretically, this fact could produce a great difference in performance between both approaches. However, the results presented in Table 8 do not show that the sequential approach, in which the model is re-trained more times, outperforms significantly the proposed ADLStream system.

5.6. Concept drift analysis

Given the evolving nature of data streams, we also consider important to carry out experiments that evaluate the effect of concept drifts on the performance of ADLStream. For this purpose, we have created a set of 15 data streams, with a million instances each, using different generators (RBF, RandomTree, Agrawal, SEA, LED) from MOA [63]. These streams cover the main types of concept drifts (gradual, abrupt, incremental and recurrent drift) with different speeds and class-imbalance drifts. Table 9 provides a more detailed description regarding the number of attributes, number of classes, imbalance rate (IR) and type of drift of each dataset.

As stated in previous sections, the ADLStream framework can be used with any DL model. The model should be selected depending on the characteristics of the data so that maximal accuracy can be obtained.

Table 7
Processing time in milliseconds for the best methods of different families over all datasets.

#	Dataset	**ADLStream**	**ARF**	**MLP**	**NB**	**SCD**	**ASHT**	**LBAG**	**BO-AD**	**SGD**
1	TwoPatterns	1.587	0.258	0.094	0.029	0.049	0.065	1.309	0.520	0.056
2	CinCECGtorso	2.296	1.287	0.867	0.380	0.700	0.851	15.543	7.299	0.242
3	TwoLeadECG	1.903	0.155	0.079	0.010	0.017	0.023	0.468	0.362	0.024
4	Wafer	1.459	0.087	0.109	0.052	0.035	0.034	0.754	0.591	0.033
5	pendigits	1.241	0.326	0.071	0.009	0.017	0.017	0.332	0.236	0.080
6	FacesUCR	1.749	0.817	0.097	0.092	0.175	0.206	3.334	2.820	0.112
7	Mallat	1.938	1.073	0.629	0.482	0.932	1.068	19.527	11.152	0.235
8	FaceAll	1.640	0.687	0.095	0.098	0.182	0.213	3.370	2.816	0.113
9	Symbols	1.940	0.654	0.402	0.129	0.246	0.315	6.623	3.278	0.170
10	ItalyPowerDemand	1.971	0.095	0.084	0.003	0.005	0.009	0.149	0.154	0.025
11	ECG5000	1.401	0.194	0.096	0.037	0.070	0.079	1.431	1.045	0.062
12	MoteStrain	2.037	0.139	0.079	0.009	0.017	0.025	0.443	0.400	0.024
13	NonInvasiveFetalECGThorax1	1.890	6.312	0.596	2.497	3.972	4.852	80.896	59.052	0.820
14	NonInvasiveFetalECGThorax2	1.911	6.116	0.551	2.418	4.571	4.760	85.372	59.077	0.740
15	SwedishLeaf	2.048	0.729	0.094	0.095	0.186	0.230	4.464	2.546	0.123
16	FordA	1.975	0.616	0.668	0.111	0.214	0.106	4.156	2.036	0.106
17	Yoga	1.705	0.335	0.430	0.051	0.090	0.096	1.925	1.183	0.073
18	UWaveGestureLibraryX	1.580	0.735	0.440	0.147	0.263	0.305	6.929	3.004	0.183
19	FordB	1.648	0.613	0.496	0.129	0.214	0.163	4.141	1.248	0.088
20	ElectricDevices	1.403	0.296	0.095	0.033	0.064	0.073	1.320	1.053	0.076
21	UWaveGestureLibraryY	1.650	0.749	0.339	0.150	0.281	0.307	6.797	2.875	0.158
22	UWaveGestureLibraryZ	1.556	0.713	0.550	0.133	0.265	0.311	6.626	2.860	0.198
23	HandOutlines	3.323	1.606	1.480	0.331	0.641	0.882	15.734	12.670	0.254
24	InsectWingbeatSound	1.651	0.839	0.130	0.148	0.293	0.339	5.909	3.988	0.115
25	ShapesAll	2.191	6.715	0.383	2.444	4.223	4.826	86.712	58.505	0.741
26	MedicalImages	1.887	0.228	0.127	0.049	0.099	0.062	0.992	0.693	0.119
27	PhalangesOutlinesCorrect	1.791	0.220	0.079	0.009	0.019	0.022	0.493	0.541	0.023
28	ChlorineConcentration	1.409	0.183	0.099	0.027	0.053	0.037	0.679	0.324	0.050
29	Phoneme	1.821	5.531	0.581	2.905	5.937	3.098	52.109	20.749	0.753

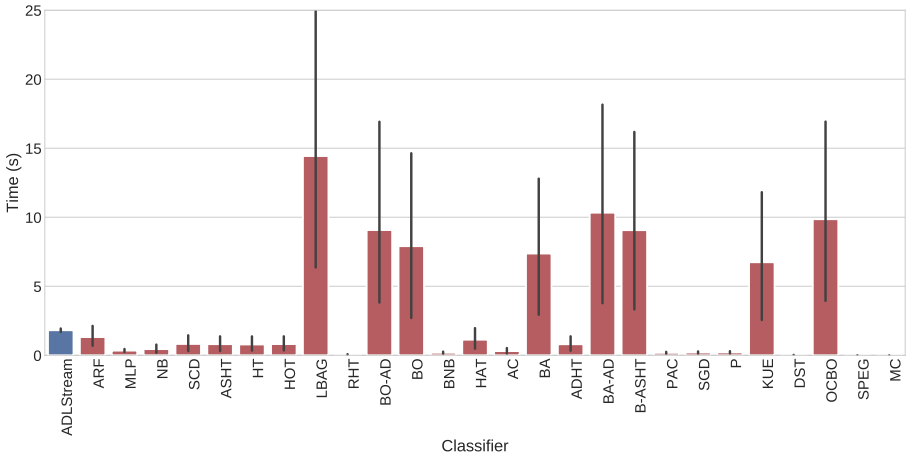

Fig. 7. Bar plot comparing the performance in terms of processing time of different techniques with the proposed ADLStream framework.

Table 8
Accuracy and processing time results of the ADLStream framework and a sequential CNN model.

#	Dataset	Time (ms)		Kappa	
		ADLStream	**CNN**	**ADLStream**	**CNN**
1	TwoPatterns	**1.587**	34.131	**1.000**	0.998
2	CinCECGtorso	**2.296**	292.087	**0.997**	0.721
3	TwoLeadECG	**1.903**	22.876	0.996	0.996
4	Wafer	**1.459**	38.484	**0.997**	0.874
5	pendigits	**1.241**	11.657	**0.986**	0.952
6	FacesUCR	**1.749**	31.201	**0.974**	0.958
7	Mallat	**1.938**	189.533	0.970	**0.973**
8	FaceAll	**1.640**	31.147	**0.977**	0.954
9	Symbols	**2.128**	70.166	0.950	**0.961**
10	ItalyPowerDemand	**1.971**	12.262	0.939	**0.943**
11	ECG5000	**1.401**	34.379	**0.895**	0.885
12	MoteStrain	**2.037**	22.152	0.895	**0.911**
13	NonInvasiveFetalECGThorax1	**1.890**	144.901	**0.893**	0.831
14	NonInvasiveFetalECGThorax2	**1.911**	144.751	**0.888**	0.884
15	SwedishLeaf	**2.048**	28.329	0.897	**0.908**
16	FordA	**1.975**	99.979	**0.832**	0.806
17	Yoga	**1.705**	85.625	**0.894**	0.813
18	UWaveGestureLibraryX	**1.580**	66.484	**0.798**	0.742
19	FordB	**1.648**	100.459	**0.808**	0.732
20	ElectricDevices	**1.403**	27.304	0.768	**0.808**
21	UWaveGestureLibraryY	**1.650**	66.501	**0.752**	0.672
22	UWaveGestureLibraryZ	**1.556**	66.488	**0.720**	0.664
23	HandOutlines	**3.323**	453.426	0.717	**0.814**
24	InsectWingbeatSound	**1.651**	53.232	**0.632**	0.591
25	ShapesAll	**2.191**	90.354	**0.696**	0.653
26	MedicalImages	**1.887**	23.932	**0.669**	0.610
27	PhalangesOutlinesCorrect	**1.791**	23.419	0.575	**0.662**
28	ChlorineConcentration	**1.409**	39.228	**0.948**	0.942
29	Phoneme	**1.821**	186.394	**0.190**	0.150

However, these experiments aim to evaluate how the asynchronous approach of the framework recovers from concept drifts. Therefore, we have decided to keep using the model detailed in 3.2.1 even though CNNs may not be optimal for these type of datasets.

Table 10 presents the Kappa accuracy obtained with the top six techniques over the different concept drift datasets. As can be seen, ADLStream obtains very competitive results, similar to those obtained by the literature methods and even better in seven cases. This implies that ADLStream also leads the performance ranking for these experiments, as it is displayed in Table 11. However, the subsequent post-hoc analysis (Table 12) concludes that there is not a significant difference between the performance of ADLStream, KUE, BA-AD and HAT. Taking into account that we have not explicitly designed a concept drift detection method, the performance obtained is comparable to other state-of-the-art classifiers. These results demonstrate the robustness of ADLStream to deal with different types of drifts in the incoming data distribution. The model can adapt to changes by relying on the incremental learning nature of neural networks, which is found to be very helpful for these situations. Furthermore, the fact that we have used CNNs, which are not particularly suitable for data without a grid-like structure, further supports the strength of our proposal.

Table 9
Datasets used for drift analysis.

Dataset	Attrs	Classes	IR	Drift type
RBFi-slow	20	3	3	
RBFi-fast	20	3	3	Incremental
LED-4	7	10	1	
RTGa	20	3	4 to 2	
RTGa3	20	3	4 to 30	
ARGWa-F1F4	9	2	2 to 1	Abrupt
ARGWa-F2F5F8	9	2	1 to 50	
SEAa-F2F4	3	2	1	
RTGg	20	3	4 to 2	
RTGg3	20	3	4 to 30	
ARGWg-F1F4	9	2	2 to 1	Gradual
ARGWg-F2F5F8	9	2	1 to 50	
SEAg-F2F4	3	2	1	
ARGWa-F3F6F3F6	9	2	1	Recurrent
ARGWg-F3F6F3F6	9	2	1	

Additionally, we provide a visual comparison of the reaction of different classifiers to concept drifts. Figure 8 shows the evolution of the prequential Kappa metric with the progress of the streams. In the particular case of the fast incremental drift dataset (*RBFi-fast*), the concept continuously changes faster than the adaptive capability of the CNN. However, it can be seen that, for the rest of datasets, ADLStream is able to recover satisfactorily from the drifts. In general, the figures show that our proposal offers a similar performance, and even better in some cases, than other popular models without the need for any explicit drift detection mechanism.

6. Conclusions

In this paper, a novel asynchronous dual-pipeline deep learning framework for data stream classification is presented. The proposed system has two separate layers for training and testing that work simultaneously, in order to provide quick predictions and perform frequent updates of the model. This architecture alleviates the computational cost problem of complex deep learning models for the data streaming scenario, in which speed is essential. The results obtained using a large number of time-series datasets showed that the ADLStream framework outperforms other state-of-the-art techniques in the literature, such as Hoeffding trees or ensemble methods. Furthermore, in terms of processing time, our proposal was also found to be competitive and even faster to other extensively used bagging and boosting methods.

We also aimed to illustrate the importance of the dual-pipeline architecture for a real-time environment by comparing its performance and processing time with a sequential approach. It was seen that the layers working asynchronously provided a decisive time reduction to deal with data arriving at high speed, while maintaining a very similar predictive accuracy. In addition, other aspects with regard to the training procedure of the system were analysed, such as the impact of the batch size and the maximum number of batches used for updating the model. Furthermore, a study on the behaviour of ADLStream over datasets that simulate different concept drifts was carried out. For these experiments, our proposal showed a performance comparable to other state-of-the-art techniques. These results proved the capacity of our framework to adapt to changes in the data, without the need for any explicit drift detection method. In conclusion, our study demonstrated that deep learning is a very powerful solution for performing online classification with data of different characteristics. To the best of our knowledge, there are no scientific papers that adapt complex deep neural networks for data streaming. Therefore, the positive results of the experimental analysis carried out could be helpful in order to give further importance to the application of deep learning models in the streaming literature.

Future work should study the suitability of the framework when the label for each instance is not always immediately available. In many applications, obtaining the ground truth is a time-consuming process that often has to be done manually. Therefore, it is important to consider other scenarios such as semi-supervised learning. Moreover, future studies should consider the application of other deep learning architectures such as Long-Short Term Memory, Temporal Convolutional Networks, and others available in the DeepLearning4J library, which can be used with MOA. More powerful networks, together with a more sophisticated hyper-parameter search and advanced sampling techniques, could enhance the performance of data stream classification. Furthermore, another interesting research direction is to develop an ensemble framework containing a pool of deep learning models with different update criteria and abstaining mechanisms.

Table 10
Kappa accuracy of the top 6 classifiers for concept drift datasets.

Dataset	**ADLStream**	**KUE**	**BA-AD**	**HAT**	**ARF**	**BO**
RBFi-slow	**0.955**	0.881	0.888	0.792	0.925	0.879
RBFi-fast	0.474	0.351	0.494	0.423	**0.779**	0.399
LED-4	0.688	0.709	**0.710**	0.707	0.709	0.707
RTGa	0.848	**0.912**	0.816	0.872	0.820	0.854
RTGa3	0.766	0.820	0.791	**0.826**	0.703	0.594
ARGWa-F1F4	**0.855**	0.844	0.849	0.849	0.766	0.760
ARGWa-F2F5F8	**0.796**	0.713	0.704	0.584	0.504	0.511
SEAa-F2F4	0.789	0.784	0.779	0.779	**0.790**	0.774
RTGg	0.804	**0.878**	0.848	0.813	0.752	0.809
RTGg3	0.661	**0.737**	0.716	0.685	0.492	0.526
ARGWg-F1F4	**0.830**	0.792	0.795	0.731	0.690	0.704
ARGWg-F2F5F8	**0.693**	0.582	0.481	0.392	0.294	0.444
SEAg-F2F4	0.786	0.782	0.777	0.777	**0.788**	0.774
ARGWa-F3F6F3F6	**0.866**	0.812	0.828	0.838	0.736	0.729
ARGWg-F3F6F3F6	**0.770**	0.712	0.695	0.601	0.599	0.607

Table 11
Friedman Test Ranking for concept drift experiments.

Friedman Test Ranking	
ADLStream	2.467
KUE	2.733
BA-AD	3.067
HAT	3.533
ARF	4.200
BO	5.000

Table 12
Holm's post-hoc analysis for concept drift experiments.

PostHoc Analysis			
Classifier	p	z	Holm's α
BO	<0.001	3.708	0.010
ARF	0.011	2.537	0.013
HAT	0.118	1.561	0.017
BA-AD	0.380	0.878	0.025
KUE	0.696	0.390	0.050

Funding

This research has been funded by the Spanish Ministry of Economy and Competitiveness under the project TIN2017-88209-C2-2-R

Acknowledgements

We are grateful to NVIDIA for their GPU Grant Program that has provided us high quality GPU devices for carrying out the study.

References

[1] Manco G, Ritacco E, Rullo P, Gallucci L, Astill W, Kimber D, et al. Fault detection and explanation through big data analysis on sensor streams. Expert Systems with Applications. 2017;87:141 – 156.

[2] Khan I, Huang JZ, Ivanov K. Incremental density-based ensemble clustering over evolving data streams. Neurocomputing. 2016;191:34–43.

[3] Li Q, Huang Y, Tian X, Li F, Lian G. Integrated financial data stream mining based on memory curve. Journal of Interdisciplinary Mathematics. 2017;20(4):1059–1071.

[4] Puthal D, Nepal S, Ranjan R, Chen J. A dynamic prime number based efficient security mechanism for big sensing data streams. Journal of Computer and System Sciences. 2017;83(1):22–42.

[5] Althouse BM, Scarpino SV, Meyers LA, Ayers JW, Bargsten M, Baumbach J, et al. Enhancing disease surveillance with

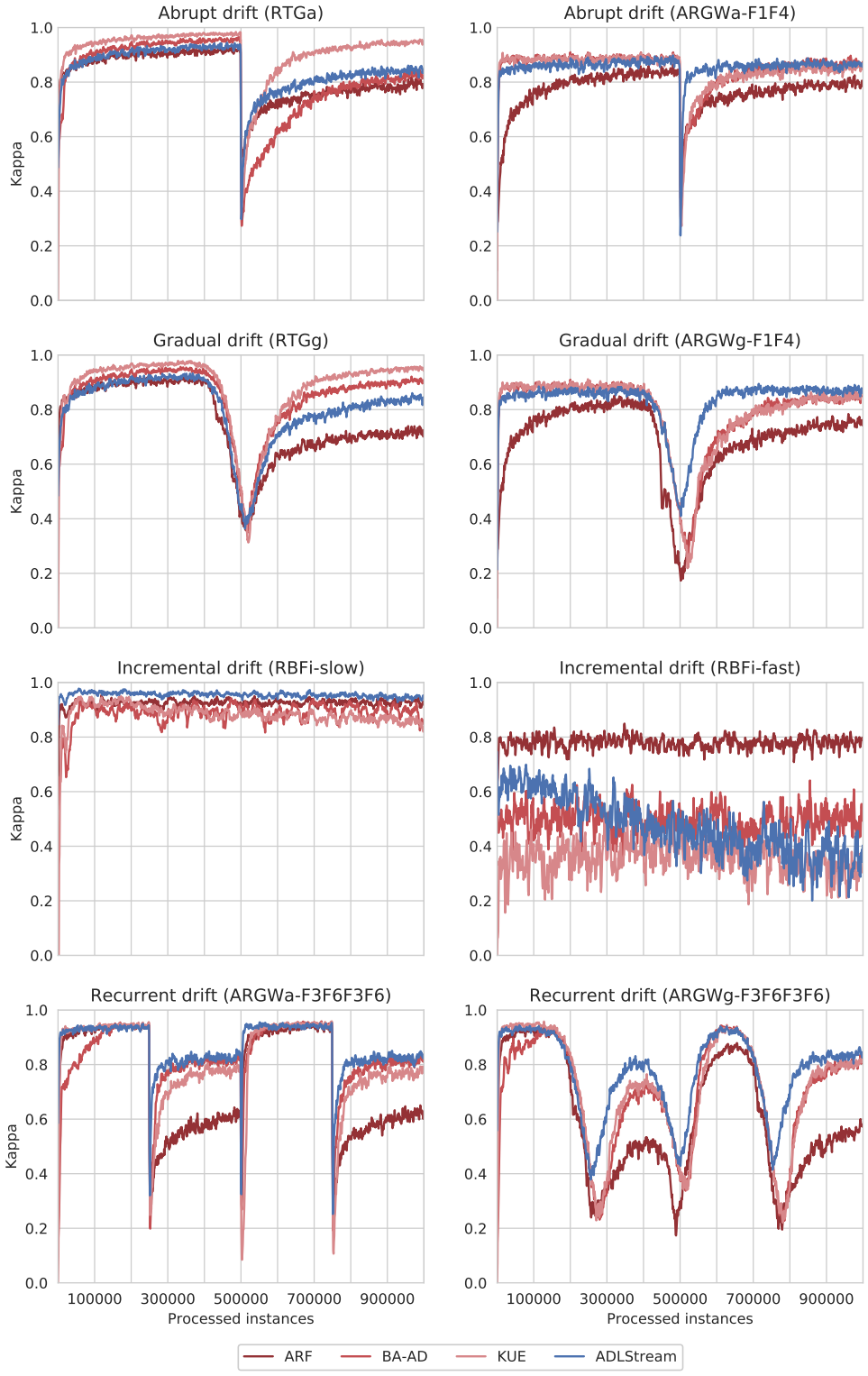

Fig. 8. Prequential Kappa evolution for ARF, BA-AD, KUE and ADLStream over two datasets of each drift type.

novel data streams: challenges and opportunities. EPJ Data Science. 2015;4(1):1–8.

[6] Wang D, Wu P, Zhao P, Wu Y, Miao C, Hoi SCH. High-Dimensional Data Stream Classification via Sparse Online Learning. In: 2014 IEEE International Conference on Data Mining; 2014. p. 1007–1012.

[7] Rafiei MH, Adeli H. A New Neural Dynamic Classification Algorithm. IEEE Transactions on Neural Networks and Learning Systems. 2017;28(12):3074–3083.

[8] Acharya UR, Oh SL, Hagiwara Y, Tan JH, Adeli H. Deep convolutional neural network for the automated detection and diagnosis of seizure using EEG signals. Computers in Biology and Medicine. 2018;100:270–278.

[9] Aggarwal CC. A Survey of Stream Classification Algorithms. In: Data Classification: Algorithms and Applications; 2014. .

[10] Read J, Perez-Cruz F, Bifet A. Deep Learning in Partially-labeled Data Streams. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing. SAC '15. New York, NY, USA: ACM; 2015. p. 954–959.

[11] Yao H, Tang X, Wei H, Zheng G, Li Z. Revisiting Spatial-Temporal Similarity: A Deep Learning Framework for Traffic Prediction; 2018.

[12] Dau HA, Bagnall AJ, Kamgar K, Yeh CM, Zhu Y, Gharghabi S, et al. The UCR Time Series Archive. CoRR. 2018;abs/1810.07758.

[13] Xiaofeng L, Weiwei G. Study on a classification model of data stream based on concept drift. International Journal of Multimedia and Ubiquitous Engineering. 2014;9(5):363–372.

[14] Jani R, Bhatt N, Shah C. A Survey on Issues of Data Stream Mining in Classification. In: Satapathy SC, Joshi A, editors. Information and Communication Technology for Intelligent Systems (ICTIS 2017) - Volume 1. Cham: Springer International Publishing; 2018. p. 137–143.

[15] Krawczyk B, Minku LL, Gama J, Stefanowski J, Woźniak M. Ensemble learning for data stream analysis: A survey. Information Fusion. 2017;37:132 – 156.

[16] Cano A, Krawczyk B. Kappa Updated Ensemble for drifting data stream mining. Machine Learning. 2019;Available from: https://doi.org/10.1007/s10994-019-05840-z.

[17] Lu Y, Boukharouba K, Boonært J, Fleury A, Lecœuche S. Application of an incremental SVM algorithm for on-line human recognition from video surveillance using texture and color features. Neurocomputing. 2014;126:132 – 140. Recent trends in Intelligent Data Analysis Online Data Processing.

[18] Ramírez-Gallego S, Krawczyk B, García S, Wózniak M, Benítez JM, Herrera F. Nearest neighbor classification for high-speed big data streams using spark. IEEE Transactions on Systems, Man, and Cybernetics: Systems. 2017;47(10):2727–2739.

[19] Pratama M, Anavatti SG, Er MJ, Lughofer ED. pClass: An Effective Classifier for Streaming Examples. IEEE Transactions on Fuzzy Systems. 2015;23(2):369–386.

[20] Lin QM, Xiao Y, Ye N, Wang RC. A method of cleaning RFID data streams based on Naive Bayes classifier. International Journal of Ad Hoc and Ubiquitous Computing. 2016;21(4):237–244.

[21] Nguyen HL, Woon YK, Ng WK. A Survey on Data Stream Clustering and Classification. Knowledge and Information Systems. 2014 12;45.

[22] Domingos P, Hulten G. Mining High-speed Data Streams. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '00. New York, NY, USA: ACM; 2000. p. 71–80.

[23] Pfahringer B, Holmes G, Kirkby R. New Options for Hoeffding Trees. In: Proceedings of the 20th Australian Joint Conference on Advances in Artificial Intelligence. AI'07. Berlin, Heidelberg: Springer-Verlag; 2007. p. 90–99.

[24] Bifet A, Gavaldà R. Adaptive Learning from Evolving Data Streams. In: Adams NM, Robardet C, Siebes A, Boulicaut JF, editors. Advances in Intelligent Data Analysis VIII. Berlin, Heidelberg: Springer Berlin Heidelberg; 2009. p. 249–260.

[25] Tennant M, Stahl F, Rana O, Gomes JB. Scalable real-time classification of data streams with concept drift. Future Generation Computer Systems. 2017;75:187 – 199.

[26] Gama Ja, Žliobaitė Ie, Bifet A, Pechenizkiy M, Bouchachia A. A Survey on Concept Drift Adaptation. ACM Comput Surv. 2014 Mar;46(4):44:1–44:37.

[27] Gama J, Medas P, Castillo G, Rodrigues P. Learning with Drift Detection. vol. 8; 2004. p. 286–295.

[28] Baena-García M, Campo-Ávila J, Fidalgo-Merino R, Bifet A, Gavald R, Morales-Bueno R. Early Drift Detection Method. 2006 01;.

[29] Frías-Blanco I, Del Campo-Ávila J, Ramos-Jiménez G, Morales-Bueno R, Ortiz-Díaz A, Caballero-Mota Y. Online and non-parametric drift detection methods based on Hoeffding's bounds. IEEE Transactions on Knowledge and Data Engineering. 2015;27(3):810–823.

[30] Vicente R, Kinouchi O, Caticha N. Statistical Mechanics of Online Learning of Drifting Concepts : A Variational Approach. Mach Learn. 1998 02;32.

[31] Oza NC. Online bagging and boosting. In: 2005 IEEE International Conference on Systems, Man and Cybernetics. vol. 3; 2005. p. 2340–2345 Vol. 3.

[32] Bifet A, Holmes G, Pfahringer B. Leveraging Bagging for Evolving Data Streams. In: Balcázar JL, Bonchi F, Gionis A, Sebag M, editors. Machine Learning and Knowledge Discovery in Databases. Berlin, Heidelberg: Springer Berlin Heidelberg; 2010. p. 135–150.

[33] Gomes HM, Bifet A, Read J, Barddal JP, Enembreck F, Pfharinger B, et al. Adaptive random forests for evolving data stream classification. Machine Learning. 2017;106(9):1469–1495.

[34] Krawczyk B, Cano A. Online ensemble learning with abstaining classifiers for drifting and noisy data streams. Applied Soft Computing. 2018;68:677 – 692.

[35] Pinagé F, dos Santos EM, Gama J. A drift detection method based on dynamic classifier selection. Data Mining and Knowledge Discovery. 2019;.

[36] Anderson R, Koh YS, Dobbie G, Bifet A. Recurring concept meta-learning for evolving data streams. Expert Systems with Applications. 2019;138:112832.

[37] Ghazikhani A, Monsefi R, Sadoghi Yazdi H. Online neural network model for non-stationary and imbalanced data stream classification. International Journal of Machine Learning and Cybernetics. 2014;5(1):51–62.

[38] Zhang Y, Yu J, Liu W, Ota K. Ensemble Classification for Skewed Data Streams Based on Neural Network. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems. 2018;26(05):839–853.

[39] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2016. .

[40] Mnih V, Badia AP, Mirza M, Graves A, Harley T, Lillicrap TP, et al. Asynchronous Methods for Deep Reinforcement Learning. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48. ICML'16. JMLR.org; 2016. p. 1928–1937.

[41] Cano A, Krawczyk B. Evolving rule-based classifiers with genetic programming on GPUs for drifting data streams. Pattern Recognition. 2019;87:248 – 268.

[42] Laurinec P, Lucká M. Interpretable multiple data streams clustering with clipped streams representation for the improvement of electricity consumption forecasting. Data Mining and Knowledge Discovery. 2019;33(2):413–445.

[43] Jiang R, Fei H, Huan J. A family of joint sparse PCA algorithms for anomaly localization in network data streams. IEEE Transactions on Knowledge and Data Engineering. 2013;25(11):2421–2433.

[44] Boedihardjo AP, Lu CT, Wang B. A framework for exploiting local information to enhance density estimation of data streams. ACM Transactions on Knowledge Discovery from Data. 2014;9(1).

[45] Tabassum S, Gama J. Biased Dynamic Sampling for Temporal Network Streams. In: Aiello LM, Cherifi C, Cherifi H, Lambiotte R, Lió P, Rocha LM, editors. Complex Networks and Their Applications VII. Cham: Springer International Publishing; 2019. p. 512–523.

[46] Babcock B, Datar M, Motwani R. Sampling from a Moving Window over Streaming Data. In: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms. SODA '02. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics; 2002. p. 633–634. Available from: http://dl.acm.org/citation.cfm?id=545381.545465.

[47] Efraimidis PS. In: Zaroliagis C, Pantziou G, Kontogiannis S, editors. Weighted Random Sampling over Data Streams. Cham: Springer International Publishing; 2015. p. 183–195.

[48] Aggarwal CC. On Biased Reservoir Sampling in the Presence of Stream Evolution. In: Proceedings of the 32Nd International Conference on Very Large Data Bases. VLDB '06. VLDB Endowment; 2006. p. 607–618.

[49] Acharya UR, Oh SL, Hagiwara Y, Tan JH, Adeli H, Subha DP. Automated EEG-based screening of depression using deep convolutional neural network. Computer Methods and Programs in Biomedicine. 2018;161:103 – 113.

[50] Zhao B, Lu H, Chen S, Liu J, Wu D. Convolutional neural networks for time series classification. Journal of Systems Engineering and Electronics. 2017;28(1):162–169.

[51] Zheng Y, Liu Q, Chen E, Ge Y, Zhao JL. Exploiting multi-channels deep convolutional neural networks for multivariate time series classification. Frontiers of Computer Science. 2016 Feb;10(1):96–112.

[52] Yu F, Koltun V. Multi-Scale Context Aggregation by Dilated Convolutions. In: 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings; 2016. .

[53] Chen Y, Jiang H, Li C, Jia X, Ghamisi P. Deep Feature Extraction and Classification of Hyperspectral Images Based on Convolutional Neural Networks. IEEE Transactions on Geoscience and Remote Sensing. 2016 Oct;54(10):6232–6251.

[54] Nair V, Hinton GE. Rectified linear units improve Restricted Boltzmann machines. In: ICML 2010 - Proceedings, 27th International Conference on Machine Learning; 2010. p. 807–814.

[55] Álvaro Arcos-García, Álvarez García JA, Soria-Morillo LM. Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods. Neural Networks. 2018;99:158 – 165.

[56] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings; 2015. .

[57] Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov R. Improving neural networks by preventing co-adaptation of feature detectors. CoRR. 2012;abs/1207.0580.

[58] Keskar NS, Mudigere D, Nocedal J, Smelyanskiy M, Tang PTP. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. CoRR. 2016;abs/1609.04836. Available from: http://arxiv.org/abs/1609.04836.

[59] Chollet F. Keras. GitHub; 2015. Accessed 29 April 2019. https://github.com/fchollet/keras.

[60] Fernández-Rodríguez JY, Álvarez García JA, Fisteus JA, Luaces MR, Magaña VC. Benchmarking real-time vehicle data streaming models for a smart city. Information Systems. 2017;72:62 – 76.

[61] Bifet A, Holmes G, Kirkby R, Pfahringer B. MOA: Massive Online Analysis. Journal of Machine Learning Research. 2010;11:1601–1604.

[62] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. 2011;12:2825–2830.

[63] Bifet A, Gavaldà R, Holmes G, Pfahringer B. Machine Learning for Data Streams with Practical Examples in MOA. MIT Press; 2018.

[64] Gama J, Sebastião R, Rodrigues PP. On evaluating stream learning algorithms. Machine Learning. 2013 Mar;90(3):317–346.

[65] Gama J, Sebastião R, Rodrigues P. Issues in evaluation of stream learning algorithms; 2009. p. 329–338.

[66] Bifet A, de Francisci Morales G, Read J, Holmes G, Pfahringer B. Efficient Online Evaluation of Big Data Stream Classifiers. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '15. New York, NY, USA: ACM; 2015. p. 59–68.

[67] García S, Herrera F. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. Journal of Machine Learning Research. 2008;9:2677–2694.

[68] Parejo J, García J, Ruiz-Cortés A, Riquelme JC. STATService: Herramienta de análisis estadístico como soporte para la investigación con Metaheurísticas. Actas del VIII Congreso Espanol sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados. 2012;.

[69] de Barros RSM, de Carvalho Santos SGT. An overview and comprehensive comparison of ensembles for concept drift. Information Fusion. 2019;52:213 – 244.

View publication stats