Effective Weighted k-Nearest Neighbors for Dynamic Data Streams

Maroua Bahri

Effective Weighted k-Nearest Neighbors for Dynamic Data Streams

Maroua Bahri
MiMove, Inria Paris

Paris, France
maroua.bahri@inria.fr

***Abstract*—Many real-world applications involve classification from evolving data streams. However, learning in such environment requires algorithms able to learn and predict from potentially unbounded data that are constantly changing. For this to happen, stream algorithms should restrict the storage to a part of – and/or synopsis information from – the stream using efficient and accurate manners and strategies, such as window models and summarization techniques (e.g., sampling, sketching, dimensionality reduction). In this work, we focus on the k-Nearest Neighbors (kNN) where most of the existing approaches for data streams consider that instances have the same weight from the start to the finish of the processing task.**

In a streaming data scenario, it is often the case that the most recent elements from the data stream are the more relevant ones. Taking into account that the most recent instances are more relevant, we propose a novel kNN approach that stores instances in a sliding window and weighs them according to their arrival time (i.e position on the window) using an adjusted weight function. The empirical results on comprehensive real and synthetic datasets indicate the effectiveness and efficiency of our proposed approach in comparison with state-of-the-art algorithms.

***Index Terms*—Data stream classification, k-nearest neighbors, sliding window, weighting vote.**

I. Introduction

The evolution of technology has invaded our lives in multiple domains and changed the way in which we generate and consume data. Several emerging applications and devices produce and accumulate a massive volume of data at an ever increasing rate in the form of streams. Hence, the use of data stream mining has become common in different domains, such as traffic management, health monitoring, social networks, and Internet of Things. The latter is a good example where multiple connected devices (things) and sensors are interacting with each other yielding to huge amounts of data generated continuously and online as *streams* [1], [2].

The application of machine learning in the classical offline (or static) setting assumes that data can be randomly accessed multiple times without strict processing time or memory constraints. Classification is a prevalent machine learning task that creates a model using labelled data (i.e., training dataset) and can accurately predict the class labels of unlabelled, previously unseen data. Commonly, algorithms designed to be trained on static data make multiple passes over the training data, each of them refining the model further [3].

However, when applied on streams, static algorithms fail to process a potentially infinite sequence of data because of its evolving nature that requires methods to adapt automatically. A data stream may evolve over time experiencing concept drift, i.e., changes in the learned model. Dealing with concept drift is one of the most important challenges to a classifier. Moreover, the algorithm must be able to handle the stream using limited resources, because the data stream flood continuously which makes it impossible to be stored in the memory and processed in an offline fashion. Under these constraints, an efficient and effective algorithm should be able to obtain a high – relatively good – accuracy and use reasonable amounts of resources, in terms of memory and time [4]. Many real-world applications involve classification from evolving data streams. Filtering spam emails is a good example for classification where we predict if an email is a spam or not based on the text contents (attributes).

To cope with the data stream resource challenges (e.g., memory and time constraints) and address the stream framework requirements while processing evolving data, stream algorithms use well-established manners [4], [5]. The latter techniques include, but not limited to, *one-pass* processing where instances should be processed only once, *sliding window* [6] of a fixed size, where only the most recent instances from the stream are stored, and *dimensionality reduction* [7] to reduce the number of attributes of data.

The k-Nearest Neighbors (kNN) is a well-known algorithm that has been adapted to the stream setting by maintaining a sliding window of a fixed size since it is impossible to store the entire stream in memory [8]. Despite the fact that this stream version of kNN is limited by the size of the moving window, previous empirical studies [8], [9] and our analysis (in Section IV) show that the standard stream kNN is still computationally expensive. Besides, drift detectors monitor the data stream and update the model when a change is detected. On the other hand, fading factors smoothly phase out older instances in favor of the new by using a weighting function or a sliding window of instances. The simplest sliding window is of a fixed size, where the oldest instance is removed from the tail and the newest one is added to the top, which makes it suitable for gradual and incremental forms of drifts.

In this paper, we propose and efficient and effective kNN approach for data streams that is robust to the presence of

concept drift. We also investigate the predictive performance improvements of this proposed approach compared to well-known baselines. The main contributions of this work are as follows: (i) an adaptation of the existing kNN algorithm for evolving data streams which uses a sliding window to maintain the recent observations from the stream; (ii) an effective weighting strategy to adjust the weight of instances as data flow which makes the proposed approach work adaptively to evolving data and handle concept drifts; and (iii) an experimental study[1] is provided where we assess the performance of our proposed approach and compare it against state-of-the-art algorithms based on a diverse set of real and synthetic datasets.

The remainder of this work is organized as follows. In Section II, we discuss relevant work in stream classification and neighborhood-based approaches for comparison. Section III describes the weighting adjustment strategy and the proposed kNN approach for data streams. Section IV presents the experimental results. We conclude the paper and pose directions for future work in Section V.

II. Related Work

Several classification algorithms have been thoroughly studied and used with evolving data streams, mostly derived from the traditional algorithms for the offline setting [4], [10], [11], e.g., decision trees [12]–[14], naive Bayes [15], [16], k-Nearest Neighbors (kNN) [8], [17], [18], and the ensemble-based methods [11], [19]–[21]. Naive Bayes (NB) [16] is the simplest classifier that updates counters with each observation and uses the assumption "all the attributes are independent of each other given the class label". In order to make prediction, the NB classifier uses the Bayes theorem using the stored counters which makes it useful with massive data streams. This naive assumption between attributes does not always hold in practice, which can lead to (potentially) bad results.

The Hoeffding Tree (HT) – also known as very fast decision tree – algorithm is a common algorithm for classification that consists in a streaming adaptation of the well-known static decision tree algorithm [12]. It is a decision tree method that is capable of learning from evolving data streams incrementally by using the Hoeffding bound to choose the optimal splitting attributes. Hoeffding tree has been widely used as a base learner to ensemble-based methods, such as bagging and adaptive random forest [19], [22].

Unlike naive Bayes and HT, the kNN algorithm does not learn any model, since it maintains all instances in order to find the neighbors for every test instance. A basic implementation consists in keeping a moving window that stores the most recent instances from the stream. Self-Adjusting Memory kNN (Sam$k$NN) [18] algorithm is another kNN variation that builds an ensemble of models to deal with concept drifts. For this to happen, the Sam$k$NN algorithm uses two memories: short-term memory to target current concept, the long-term memory to keep track about the past concepts.

[1]The source code and datasets employed in our analysis can be found at https://bit.ly/3gqDopQ.

In [8], authors proposed a Probabilistic Adaptive Window (PAW) which is based on the approximate counting of Morris. PAW includes older instances as well as the most recent ones from the stream, and therefore maintains somewhat information about past concept drifts and adapts to new ones. PAW has been used with the kNN as a window to maintain instances. In order to add an explicit change detection mechanism to the aforementioned kNN approach, authors in [8] used, on top of PAW, ADWIN [23], a change detector that keeps a variable-length window of recently seen instances to handle concept drifts in the distribution.

Instance Based Learner on Streams (IBLStreams) has been proposed in [24] as an extension of MOA [25]. It consists of a learning algorithm for stream classification and regression problems, and optimizes the composition and size of the case base autonomously by adding to the case base each new instance available from the stream. Besides, IBLStreams checks whether other instances might be removed, either because they have become redundant or they are outliers. To do so, a set k of instances within the neighborhood of a given instance are considered as candidates. The prediction for a new test instance is therefore achieved by combining the outputs of its neighbors.

These aforementioned single algorithms serve the purpose of common baselines since they are used for comparison in the data stream classification.

III. Weighted kNN

A. Preliminaries

A data stream is potentially an unbounded sequence of data $S = x_1, x_2, \ldots, x_t, \cdots$ that arrives continuously, where each observation is composed of a vector of d attributes, $x_i \in \mathbb{R}^d$. Let y be an open-ended sequence of the corresponding class labels, such that each instance x_i has a class label $y \in C$, where $C = \{c_1, \cdots, c_l\}$ is a finite set of possible class labels and $l > 1$. We focus on data stream classification which aims to predict a target class label given a new arrived unlabeled instance. Unlike the traditional batch setting where an algorithm generates a model based on a training set, data stream classification is usually evaluated in the incremental setting called *Interleaved Test-Then-Train* where instances are firstly used for prediction (testing) before being used for learning (training).

More formally, the classifier builds a model $\mathcal{M}$ and given an instance x_t, the learning aims to predict its discrete class y_t from a set C, st.

$$\hat{y} = \mathcal{M}_{t-1}(x_t)$$

, where $\mathcal{M}_{t-1}$ is the last learned model. Afterwards, once the true class label y_t is revealed and before proceeding with the next incoming instance, the algorithm updates its model and generates the new one $\mathcal{M}_t$ using the current tuple (x_t, y_t) as

$$\mathcal{M}_t = train(\mathcal{M}_{t-1}, x_t, y_t).$$

Fig. 1: The nearest neighbors to a test instance (with $k = 3$). The red point represents the unlabeled instance and the yellow ones are the nearest neighbors in the window according to a distance metric.

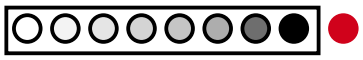

Fig. 2: The nearest neighbors have different weights that depend on their position (proportional to their age) in the window. The white point represents the oldest instance in the window with the lowest weight, while the black one represent the most recent with the highest weight.

B. Algorithm

Window models are a very popular way to keep instances in the memory for data streams learning. For instance, the sliding window with a fixed size that moves forward as time progresses [6]. This window maintains the most recent observations from the stream consisting of a smaller number than the real size of the huge stream which is potentially infinite. Several stream mining algorithms uses windows, e.g., the kNN and SAMkNN algorithms.

The kNN is one of the most popular non-parametric algorithms used for classification. In the offline setting, the kNN algorithm stores all the instances in a dataset and searches for the k closet neighbors to an unlabeled test instance by computing a distance metric (e.g., the Euclidean distance). The prediction is therefore made by taking the most frequent class label over the k-nearest neighbors in the dataset. kNN has already been applied in the streaming scenario where the basic algorithm maintains a fixed size sliding window of the most recent data instances whilst older ones are constantly deleted as the stream progresses. Label prediction of a new instance is therefore obtained by taking the majority vote of its k closest instances inside the sliding window. As mentioned before, the sliding window is indeed an efficient technique to make some static algorithms applicable on evolving data streams. However, during the prediction and when $k > 1$, two neighbor instances to the test instance (based on a distance metric) can be the most recent and oldest instances in the window that belong to two different concepts (see example in Fig. 1 of a window of size 8). In this situation where all the instances have the same weight (=1), the prediction would impact the final accuracy of the algorithm. To overcome this disadvantage, we propose the Weighted kNN, abbreviated WkNN, that is able to cope with concept drifts and can be easily applied in practice without any parametrization.

In the following, we describe the main idea of our WkNN approach which is derived from the traditional kNN for data streams that uses a fixed size sliding window. We employ a weighted function that assigns a weight to instances based on their position inside the window. This weighting strategy addresses the question how much a neighbor matters relative to other nearest neighbors taking into account their arriving order. The time-dependent weight is therefore adjustable with the stream progression, whenever a new instance enters the window, we update the weight of all the instances using the following formula:

$$w_i = 2 - \frac{(s-1)}{(w-1)}, \tag{1}$$

where w is the size of the window and s is the number of steps back from the current time that an instance was seen. So, as depicted in Fig. 2, the most recent instance "youngest", $s = 1$, would have a weight of 2. Similarly, the "oldest" instance that is about to be removed from the window, with $s = w$ (w steps back from the current time), would have a weight of 1. Hence, the weight would gradually go from 2 to 1 in order to avoid having instances with a zero weight (in the case from 1 to 0), and thus neighbor instances that are ignored. In this way the window can be maintained at a fixed size and kept up-to-date. The problems of data stream's infinite length and concept drift can correspondingly be well addressed.

The kNN approaches generally use the Euclidean distance metric which is often chosen for computing the dissimilarities of the nearest neighbors. Let us consider a window W, the Euclidean distance between pairs of instances, x_i and x_j, is computed as follows:

$$D_{x_j}(x_i) = \sqrt{\|x_i - x_j\|^2}. \tag{2}$$

Likewise, the k-nearest neighbors distance is defined as follows:

$$D_{W,k}(x_i) = \min_{\binom{W}{k}, x_j \in W} \sum_{j=1}^{k} D_{x_j}(x_i), \tag{3}$$

where $\binom{W}{k}$ stands for the subset of k-nearest neighbors to the instance x_i in W.

The main differences between the classical kNN and our WkNN are: (i) our approach assigns an updatable weight to each instance in the window while the vanilla kNN gives an equal weight of 1 to all the instances; and (ii) the kNN prediction process is based on equally weighted voting while ours utilizes the weights to make predictions/votes and gives more importance to recent instances. To make prediction, we sum the weights of the k nearest neighbors from each class label and select the label with the highest sum as follows:

$$\hat{y} = \arg\max_{y \in C} \sum_{(x_j, y_j) \in D_{W,k}(x)} w_j * I(y = c_j), \tag{4}$$

where y is a class label in the set of labels C, x_j is one of the nearest neighbors to x in $D_{W,k}(x)$ from Equation (3), $I(y = c_j)$ is an indicator function that returns 1 if the class of y_j of the neighbor x_j is the same as y, and 0 otherwise. Hence, this weighted voting strategy makes sense only for $k > 1$.

Algorithm 1 WkNN algorithm.
Symbols: S: data stream; C: set of class labels; k: number of neighbors; W: sliding window; w: maximum window size.

1: **function** WkNN(S, w, k)
2: $W \leftarrow \emptyset$
3: **while** $HasNext(S)$ **do**
4: $(x, y) \leftarrow Next(S)$
5: $\hat{y} \leftarrow Predict(x)$ ▷ Using Equation (4)
6: **if** $size(W) \geq w$ **then**
7: $Remove(W[0])$ ▷ Delete the oldest instance in W
8: **end if**
9: $W \leftarrow Add(x, y, 2)$ ▷ Add the instance of weight 2 (most recent one)
10: **for all** $(x_i, y_i) \in W$ **do** ▷ $i \in [0, w-1]$
11: $w_i \leftarrow 2 - \frac{w-i-1}{w-1}$
12: $Update(x_i, y_i, w_i)$
13: **end for**
14: **end while**
15: **end function**

The overall pseudocode for the Weighted kNN (WkNN) is presented in Algorithm 1. The WkNN is focused on the well-known Test-Then-Train setting [26], where every instance is used first for testing (prediction) and then for training. Nevertheless, the WkNN approach does not build a model, instead, the training phase consists of maintaining instances inside a fixed size window.

When a new instance arrives from the stream, the prediction of its label is made by taking the class label with the highest sum of weights over the nearest neighbors (line 5, Algorithm 1) retrieved from W using the kNN distance (Equation (3)). We assume that the true class label y of the current instance is available before the next instance appears, thus, it can be maintained with the corresponding instance in the window. In order to maintain a window that slides over the stream with the same size, we start by adding the first w instances from the stream into the sliding window and whenever the window is filled, the oldest instance (with a weight of 1) is removed and the new one is added to it with the highest weight 2 (line $6-9$, Algorithm 1). For each new instance, we update the weights of all the instances inside the window according to the weighting function introduced in Equation (1), (line $10-12$, Algorithm 1).

IV. Experimental Study

In this section, we first describe the test collections and configuration of each considered method. We then look at the performance offered by our proposed WkNN approach against its competitors (presented in Section II). The performance is evaluated in terms of three main dimensions: (i) the predictive performance or accuracy obtained as the final percentage of instances correctly classified; (ii) the memory (MB) used to keep the instances in the sliding window; and (iii) the time (seconds) required to learn (keep the most recent instance in kNN) and predict from data.

TABLE I: The algorithms we consider and their parameters.

Abbr.	Classifier	Parameters
NB	Naive Bayes	
kNN	k-Nearest Neighbors	$w = 1000, k = 10$
Sam$k$NN	Self adjusting memory kNN	$w = 1000, k = 10$
kNN$_W$	kNN with PAW	$w = 1000, k = 10$
kNN$_W^A$	kNN with PAW + ADWIN	$w = 1000, k = 10$
WkNN	Weighted kNN	$w = 1000, k = 10$
IBLS	Instance Based Learner on Streams	$w = 1000, k = 10$

TABLE II: Overview of the datasets.

Dataset	#Inst	#Att	#Classes	Type	MF label	LF label
SEA$_a$	1,000,000	3	2	S	57.55	42.45
SEA$_g$	1,000,000	3	2	S	57.55	42.45
AGR$_a$	1,000,000	9	2	S	52.83	47.17
AGR$_g$	1,000,000	9	2	S	52.83	47.17
Nomao	34,465	119	2	R	71.44	28.56
Poker	829,201	10	10	R	41.55	2.00
Spam	9,324	39,916	2	R	74.40	25.60
Covtype	581,012	54	7	R	48.76	0.47
CNAE	1,080	856	9	R	12.00	12.00
Har	10,299	561	6	R	19.44	14.06

MF and LF labels (in %) stands for the Most and Less Frequent class label, respectively.

A. Experimental setup

a) Methods parameters: All the methods evaluated in this paper were implemented in Java within the Massive Online Analysis (MOA) framework[2] [25]. For comparison, we used the baseline implementations of the original authors available in MOA.

Based on previous studies, such as [8], [17], [18], a bigger window size will increase the use of resource computations (i.e., memory and time) and a smaller size will impact the accuracy that may potentially decrease. Therefore, for a good tradeoff, we select $w = 1000, k = 10$ for all the kNN-based algorithms based on the empirical results reported in [8], [17]. These algorithms and their corresponding parameterization are displayed in Table I.

b) Datasets: In our experiments, we used a diverse set of both synthetic (S) and real (R) datasets in different scenarios. For this paper, we used data generators and real data, where most of them have been thoroughly used in the literature to assess the performance of data stream classification algorithms. Table II shows an overview of the datasets used while further details are provided in the rest of this section.

SEA The SEA Generator proposed by [27] is generated with 3 attributes, where only 2 are relevant, and 2 decision classes.

[2]In order to improve the reproducibility of our approach, the source code and datasets employed in our analysis can be found at https://bit.ly/3gqDopQ.

TABLE III: Accuracy comparison (%).

Dataset	WkNN	kNN	Sam$k$NN	kNN$_W$	kNN^A_W	IBLS	NB
SEA$_a$	86.79	86.79	85.48	**87.17**	**87.17**	86.04	85.37
SEA$_g$	86.56	86.54	85.21	**86.93**	**86.93**	85.77	85.37
AGR$_a$	**67.46**	62.44	67.38	64.87	64.88	64.16	65.73
AGR$_g$	**66.11**	61.26	65.70	63.60	63.54	62.86	65.75
Nomao	96.36	96.09	**96.68**	96.03	96.23	72.97	86.86
Poker	**80.65**	69.34	78.31	66.81	68.78	75.03	59.55
Spam	82.39	77.71	**92.69**	70.09	80.15	90.61	74.57
Covtype	92.05	**92.22**	89.22	91.89	91.89	88.63	60.52
CNAE	79.54	71.75	**81.57**	69.07	69.07	16.57	55.92
Har	92.73	92.33	87.57	**93.07**	93.03	86.73	73.36
Overall ∅	**83.10**	79.65	82.95	78.95	80.17	72.92	71.14

Bold and underlined values indicate the best and the second best results per dataset, respectively.

TABLE IV: Processing time comparison (s).

Dataset	WkNN	kNN	Sam$k$NN	kNN$_W$	kNN^A_W	IBLS
SEA$_a$	**50.78**	103.22	106.14	257.83	412.63	113.88
SEA$_g$	**52.45**	104.73	103.89	208.56	436.85	106.71
AGR$_a$	**65.40**	231.82	78.63	406.85	747.15	510.13
AGR$_g$	**67.05**	233.77	75.91	410.18	851.20	388.60
Nomao	**8.09**	66.49	29.62	99.05	189.47	77.19
Poker	**54.23**	207.74	95.17	374.72	541.89	163.23
Spam	**9747.11**	9820.15	18088.63	14152.23	16837.96	88393.01
Covtype	**84.43**	325.15	193.70	760.72	797.75	527.27
CNAE	21.33	17.76	2.49	17.30	31.24	**1.65**
Har	**9.90**	107.99	21.09	227.58	249.70	64.89
Overall ∅	**1654.18**	1757.57	3071.95	2605.43	3108.00	9979.92

SEA$_a$ simulates three abrupt drifts while SEA$_g$ simulates 3 gradual drifts.

AGR The AGRAWAL generator [28] creates data stream with 9 attributes and 2 classes. A perturbation factor is used to add noise to the data, both AGR$_a$ and AGR$_g$ includes 10% perturbation factor. AGR$_a$ simulates 3 abrupt drifts in the generated stream while AGR$_g$ simulates 3 gradual drifts.

Nomao It is a large dataset that has been provided by Nomao Labs [29]. This dataset contains data that arrive from multiple sources on the web about places (name, website, localization, address, fax, etc $\cdots$).

Poker The poker-hand dataset[3] consists of 829,201 instances and 10 attributes describing each hand. The class indicates the value of a hand.

Spam The spam corpus is the result of a text mining on an online news dissemination system which intends on creating an incremental filtering of e-mails classifying them as spam or not [30]. Each attribute represents the presence of a word in the instance (an e-mail).

Covtype The forest covertype dataset obtained from US forest service resource information system data where each class label presents a different cover type.

CNAE CNAE is the national classification of economic activities dataset, initially used in [31]. It contains 1,080 instances, each of 856 attributes, representing descriptions of Brazilian companies categorized into 9 classes. The original texts were preprocessed to obtain the current highly sparse dataset.

Har Human Activity Recognition dataset [32] built from several subjects performing daily living activities, such as walking, walking upstairs/downstairs, sitting, standing and laying, while wearing a waist-mounted smartphone equipped with sensors. The sensor signals were preprocessed using noise filters and attributes were normalized and bounded within $[-1, 1]$.

[3]https://archive.ics.uci.edu/ml/datasets/Poker+Hand

B. Results

We assess the predictive performance through accuracy using the Test-Then-Train evaluation methodology [26], where every instance is used for prediction and then used for training. For fair comparison, we used the configurations previously stated in Table I of the baselines presented in Section II.

Table III reports the accuracy comparison, the most common quality measure in the stream classification field [4], that is measured as the final percentage of correctly classified instances over the Test-Then-Train evaluation.

To assess the benefits in terms of resources where small values are desirable, Table IV reports the time cost, in terms of seconds (s), of the classification algorithms which consists in the time used to make prediction. For fair comparison, we assess the performance of our proposal against the neighborhood-based competitors using the same values for k and w (without NB).

Figure 3 depicts the memory performance of our proposed approach against its competitors by taking the average performance over of all the datasets.

C. Discussions

In Table III, we notice that WkNN is effective on all the datasets, even when it achieves the second highest predictive performance (underlined results), the difference with the competitors is not huge and mostly after the decimal point. We obtain the most accurate result on the overall average of all the datasets. Our approach performs much better than the baseline naive Bayes because of the naive assumption between features of the latter that is not always true and does not hold with many datasets. We also observe that the weighted kNN approach defeats the standard kNN with almost all the datasets except for Covertype where the difference is small (0.17%). On the overall average, with WkNN, we gain around 4% in accuracy in comparison with the vanilla kNN thanks to the weighting function that seems to be more appropriate than considering instances have the same weight independently from their age in the window.

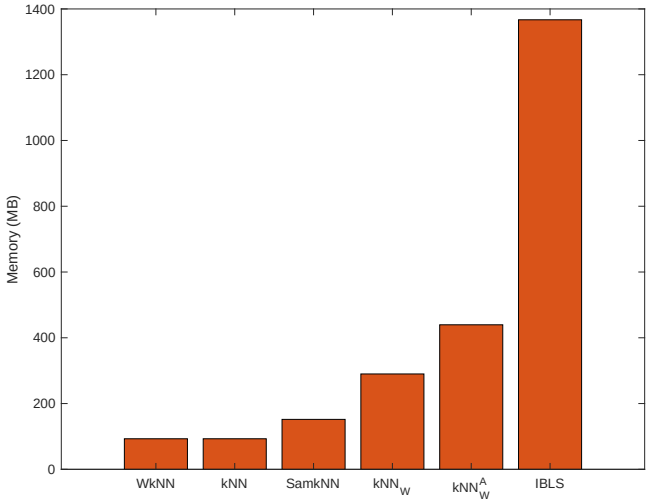

Fig. 3: Memory usage (MB).

Besides, our approach is very competitive to the neighborhood-based baselines even against those coupled with an explicit drift detection mechanism, notably the Sam$k$NN and the kNN^A_W, due to the weighted vote adjustment strategy that assigns more weight to new instances. Compared to Sam$k$NN, we can see that better results have been obtained with the synthetic datasets SEA$_a$, SEA$_g$, AGR$_a$, and AGR$_g$ that contain abrupt *(a)* and gradual *(g)* drifts, simulated using MOA. However, the WkNN is largely defeated by Sam$k$NN on the Spam dataset that represents a gradual drift [33] thanks to its sophisticated cleaning process that could decide to remove instances from the short-term and the long-term memories as the distribution may change [18]. Similarly, better predictive performance has been obtained than kNN$_W$ and kNN^A_W where we are slightly defeated on the SEA datasets. For these approaches with the latter data, the difference is less than 0.7%. IBLS is also outperformed by our proposal on all the datasets because basically it keeps track of recent instances from the stream and cannot adapt well to evolving concepts, and thus the prediction performance is affected.

Table IV shows that our approach is faster than the other methods mainly because of its simpler implementation under the MOA software. Moreover, Sam$k$NN uses a dual memory (short-term and long-term memories) to maintain models for current and past concepts which makes the prediction and drift detection times more important. Besides, kNN^A_W uses ADWIN on top of the sliding window to detect the drifts making it less efficient in terms of resources. One exception has occurred with the CNAE data simply because it is a small dataset that has the size of only one window. IBLS and Sam$k$NN are faster on this dataset because (i) IBLS removes instances autonomously and (ii) Sam$k$NN has a dual window that allows it to compute pairwise distances in two small windows which leads to faster computations than a big one (e.g., of $w = 1000$).

In Fig. 3, we observe that close amounts of memory are used by WkNN and kNN since both use the same size of window $w = 1000$ and do not utilize other data structures. On the other hand, more memory is used by kNN$_W$ that stores, other than the probabilistic approximate window, statistics related to the probability of insertion of a new instance to the window, and the kNN^A_W, on top of that, ADWIN, to handle drifts. Sam$k$NN also uses more memory because of its drift detection mechanism that needs to maintain two memories. Similarly, IBLStreams is the most memory-consuming due to its sophisticated strategy that aims at building an autonomous classifier.

To sum up, the proposed approach offers the best trade-off accuracy-resources by obtaining accurate performance on almost all the datasets and being space and time-efficient compared to the baselines.

V. Conclusions

We proposed the weighted kNN, a neighborhood-based approach for data stream classification that uses a weighted vote strategy to weight the instances according to their position / age in the sliding window. These weights are used during prediction where more interest is given to the most recent instances in the test data neighborhood. We discussed and demonstrated how the adjust weight function tends to improve the predictive performance of the standard kNN even for datasets with concept drifts using feasible resources. As future work for this project, we plan to investigate the use of sophisticated weighting strategies to see if that could improve further the accuracy taking into account the eventual presence of different types of concept drift.

References

[1] Z. Caiming and C. Yong, "A review of research relevant to the emerging industry trends: industry 4.0, iot, blockchain, and business analytics," *Journal of Industrial Integration and Management*, pp. 165–180, 2020.

[2] L. Da Xu, W. He, and S. Li, "Internet of things in industries: A survey," *IEEE Transactions on industrial informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[3] D. J. Hand, H. Mannila, and P. Smyth, *Principles of data mining*. MIT press, 2001.

[4] M. Bahri, A. Bifet, J. Gama, H. M. Gomes, and S. Maniu, "Data stream analysis: Foundations, major tasks and tools," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 11, no. 3, p. e1405, 2021.

[5] C. C. Aggarwal and S. Y. Philip, "A survey of synopsis construction in data streams," in *Data Streams*. Springer, 2007, pp. 169–207.

[6] W. Ng and M. Dash, "Discovery of frequent patterns in transactional data streams," in *Transactions on large-scale data-and knowledge-centered systems II*. Springer, 2010, pp. 1–30.

[7] M. Bahri, A. Bifet, S. Maniu, and H. M. Gomes, "Survey on feature transformation techniques for data streams," in *International Joint Conference on Artificial Intelligence*, 2020.

[8] A. Bifet, B. Pfahringer, J. Read, and G. Holmes, "Efficient data stream classification via probabilistic adaptive windows," in *Symposium On Applied Computing (SIGAPP)*. ACM, 2013, pp. 801–806.

[9] J. Read, A. Bifet, B. Pfahringer, and G. Holmes, "Batch-incremental versus instance-incremental learning in dynamic and evolving data," in *Intelligent Data Analysis (IDA)*. Springer, 2012.

[10] J. Gama, *Knowledge discovery from data streams*. CRC Press, 2010.

[11] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet, "A survey on ensemble learning for data stream classification," *Computing Surveys (CSUR)*, vol. 50, no. 2, p. 23, 2017.

[12] P. Domingos and G. Hulten, "Mining high-speed data streams," in *SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2000.

[13] J. Gama, R. Fernandes, and R. Rocha, "Decision trees for mining data streams," *Intelligent Data Analysis (IDA)*, vol. 10, no. 1, pp. 23–45, 2006.

[14] J. Gama, R. Rocha, and P. Medas, "Accurate decision trees for mining high-speed data streams," in *SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2003, pp. 523–528.

[15] M. Bahri, S. Maniu, and A. Bifet, "Sketch-based naive bayes algorithms for evolving data streams," in *International Conference on Big Data*. IEEE, 2018, pp. 604–613.

[16] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[17] M. Bahri, A. Bifet, S. Maniu, R. de Mello, and N. Tziortziotis, "Compressed k-nearest neighbors ensembles for evolving data streams," in *European Conference on Artificial Intelligence (ECAI)*. IEEE, 2020.

[18] V. Losing, B. Hammer, and H. Wersing, "Knn classifier with self adjusting memory for heterogeneous concept drift," in *International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 291–300.

[19] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem, "Adaptive random forests for evolving data stream classification," *Machine Learning*, pp. 1–27, 2017.

[20] H. M. Gomes, J. Read, and A. Bifet, "Streaming random patches for evolving data stream classification," in *International Conference on Data Mining (ICDM)*. IEEE, 2019.

[21] N. C. Oza and S. Russell, "Experimental comparisons of online and batch versions of bagging and boosting," in *SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2001, pp. 359–364.

[22] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2010, pp. 135–150.

[23] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *International Conference on Data Mining (ICDM)*. SIAM, 2007, pp. 443–448.

[24] A. Shaker and E. Hüllermeier, "Iblstreams: a system for instance-based classification and regression on data streams," *Evolving Systems*, vol. 3, no. 4, pp. 235–249, 2012.

[25] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: Massive online analysis," *Journal of Machine Learning Research (JMLR)*, vol. 11, no. May, pp. 1601–1604, 2010.

[26] A. P. Dawid, "Present position and potential developments: Some personal views statistical theory the prequential approach," *Journal of the Royal Statistical Society: Series A (General)*, vol. 147, no. 2, pp. 278–290, 1984.

[27] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," in *SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2001, pp. 377–382.

[28] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: A performance perspective," *Transactions on Knowledge and Data Engineering (TKDE)*, vol. 5, no. 6, pp. 914–925, 1993.

[29] L. Candillier and V. Lemaire, "Design and analysis of the nomao challenge active learning in the real-world," in *ALRA, Workshop ECML-PKDD*. sn, 2012.

[30] I. Katakis, G. Tsoumakas, E. Banos, N. Bassiliades, and I. Vlahavas, "An adaptive personalized news dissemination system," *Journal of Intelligent Information Systems*, vol. 32, no. 2, pp. 191–212, 2009.

[31] P. M. Ciarelli and E. Oliveira, "Agglomeration and elimination of terms for dimensionality reduction," in *International Conference on Intelligent Systems Design and Applications*. IEEE, 2009, pp. 547–552.

[32] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *IWAAL*. Springer, 2012, pp. 216–223.

[33] I. Katakis, G. Tsoumakas, and I. Vlahavas, "Tracking recurring contexts using ensemble classifiers: an application to email filtering," *Knowledge and Information Systems*, vol. 22, no. 3, pp. 371–391, 2010.