A Sketch-Based Naive Bayes Algorithms for Evolving Data Streams

Maroua Bahri
LTCI, Télécom ParisTech
Université Paris-Saclay
Paris, France
maroua.bahri@telecom-paristech.fr

Silviu Maniu
LRI, Université Paris-Sud
Université Paris-Saclay
Orsay, France
silviu.maniu@lri.fr

Albert Bifet
LTCI, Télécom ParisTech
Université Paris-Saclay
Paris, France
albert.bifet@telecom-paristech.fr

***Abstract*—A well-known learning task in big data stream mining is classification. Extensively studied in the offline setting, in the streaming setting – where data are evolving and even infinite – it is still a challenge. In the offline setting, training needs to store all the data in memory for the learning task; yet, in the streaming setting, this is impossible to do due to the massive amount of data that is generated in real-time. To cope with these resource issues, this paper proposes and analyzes several evolving naive Bayes classification algorithms, based on the well-known count-min sketch, in order to minimize the space needed to store the training data. The proposed algorithms also adapt concept drift approaches, such as ADWIN, to deal with the fact that streaming data may be evolving and change over time. However, handling sparse, very high-dimensional data in such framework is highly challenging. Therefore, we include the hashing trick, a technique for dimensionality reduction, to compress that down to a lower dimensional space, which leads to a large memory saving.**

We give a theoretical analysis which demonstrates that our proposed algorithms provide a similar accuracy quality to the classical big data stream mining algorithms using a reasonable amount of resources. We validate these theoretical results by an extensive evaluation on both synthetic and real-world datasets.

***Index Terms*—Data stream classification, Naive Bayes, Count-min sketch, Hashing trick, Concept drift**

I. Introduction

The last few decades have seen a tremendous pace in the pervasiveness of technology, including, more and more, systems and applications that generate *streams* of data: sequences of instances that are for all purposes unbounded, but also sporadic and transient.

Among one of the most popular tasks in data mining and machine learning is *classification* [1]. When dealing with big data streams, classification algorithms for static datasets have been proved to be of limited effectiveness. Thus, an important number of classification algorithms [2] have been suggested to deal with evolving and streaming data (e.g., decision trees [3]–[6], naive Bayes [7], [8], bagging [9]).

The difference between classifiers for statics datasets (batch classifiers) and data stream classifiers (also know as *online* classifiers) resides in the way how learning and prediction are performed. Unlike batch classification, online classifiers must deal with the data *incrementally* and only use one-pass processing. Moreover, and most importantly, they must use a limited amount of time (to allow analysis of each instance without delay), and a limited amount of space (not needing to store a huge amount of data for the prediction task).

Several techniques have been proposed to cope with evolving data stream and its challenges previously mentioned. Among others, sketching algorithms or sketches, a class of specialized algorithms that can produce approximate results efficiently with mathematically proven error bounds. They are useful to process fast data using fewer resources.

Our contributions: In this paper, we focus on these important challenges in classification over data streams, by using sketching techniques for data streams. Sketching techniques are well-known for keeping small, but approximate, synopses of data.

The main focus of our work attempts to extend the stream naive Bayes algorithm to deal with massive data by keeping the fractional counts of the amount of data seen so far in a count-min sketch. Our main contributions can be summarized as follows:

- Our first proposed algorithm, named the *Sketch Naive Bayes SketchNB* algorithm (Section IV-A), stores data with high-quality approximations in the sketch which allows both fast predictions and uses a minimal amount of space for training.
- Our second algorithm, the *Adaptive Sketch Naive Bayes AdaSketchNB* algorithm (Section IV-B), extends the SketchNB algorithm to deal with evolving data using a concept drift mechanism, namely ADWIN [10].
- Finally, in the challenging context of evolving data streams, many domains generate very high attribute dimensional data, so, we proposed a third contribution to the two stated algorithms, $SketchNB_{HT}$ and $AdaSketchNB_{HT}$ (Section IV-C), that aims to address high dimensionality using the hashing trick technique [11] (Section II-C2).

The remainder of this paper is organized as follows. In Section II, we present the main components of our contributions. Section III reviews related work. In Section IV, we present our approaches in classifying streaming data. Section V discusses the different experiments performed on both artificial and real datasets. Finally, we draw our conclusion in Section VI.

II. Background

A. Notation

We define here the notations that will be used throughout this paper. We assume that the data stream S contains an infinite number of *instances* $X_1, X_2, \ldots, X_N, \ldots$, where each instance is a vector containing a integral *attributes*, denoted by $X_i = (x_i^1, x_i^2, \ldots, x_i^a)$. We will denote by N the number of instances encountered thus far in the stream. The *classification problem* is to assign each instance X_i to a *class* $c_j \in C$.

B. Naive Bayes Classifier

One of the most often used classifiers is naive Bayes [12]. It uses the assumption that the attributes are all independent of each other and w.r.t. the class label uses Bayes's theorem to compute the posterior probability of a class given the evidence (the training data). This assumption is obviously not always true in practice, yet the naive Bayes classifier has a surprisingly strong performance in real-world scenarios.

Using Bayes's Theorem one can compute the probability of each class:

$$P(C \mid A_1, \ldots, A_a) = \frac{P(A_1, \ldots, A_a \mid C) \cdot P(C)}{P(A_1, \ldots, A_a)} \quad (1)$$

where $P(C|A_1, \ldots, A_a)$ is the posterior probability of the target class given the attributes, $P(C)$ is the prior probability of the class, $P(A_1, \ldots, A_a|C)$ is the likelihood, and $P(A_1, \ldots, A_a)$ is the prior probability of attributes.

Once the probabilities are computed, the class having the highest probability is chosen as the predicted class.

The training in naive Bayes is straightforward. To compute class probabilities, we estimate them as a fraction of the instances seen thus far, as follows:

- Estimate $P(C)$ as the fraction of records having $C = c_j$,

$$P(C = c_j) = \frac{Count(C = c_j)}{N}$$

- Estimate $P(X = A_1, ..., A_a|C)$ as the fraction of records with $C = c_j$ for which $X = A_1, ..., A_a$,

$$P(X = A_i|C = c_j) = \frac{Count(X = A_i \wedge C = c_j)}{Count(C = c_j)}$$

We use this attribute independence assumption to construct our naive Bayes algorithm with count-min sketch.

C. Data Summarization Techniques

This section states the two key components of our solutions for mining data streams.

1) Count-Min Sketch: Applications can now generate data at rates and volumes which cannot be reasonably stored. To cope with the vast scale of information, one way is to use *synopsis* techniques [13]–[15]. Among them, the Count-Min Sketch (CMS) [16] which is a generalization of Bloom filters [17] used for counting items of a given type, using approximate counts that are theoretically sound.

CMS consists of a two-dimensional array of $w \cdot d$ cells of counters, having a width w of columns, and a depth d of rows.

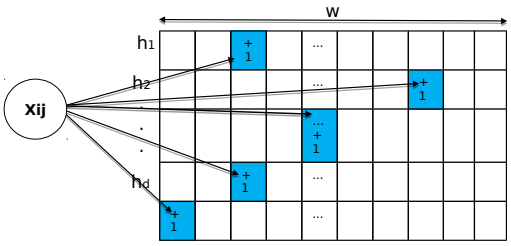

Fig. 1: Count-min sketch.

w and d are controlled by the approximation parameters ϵ and δ, such that, with probability $1 - \delta$, the approximate counts obtained from the sketch are within an absolute error ϵ of the true counts. Each row in d is a different hash function $h_1, h_2, \ldots, h_d$; each h_i is used to determine in which of the w counters on row i a count is incremented. Figure 1 illustrates the CMS.

Depending on the ϵ and δ parameters, the CMS should be initialized using the following dimensions: $d = \frac{e}{\epsilon}$ and $w = \ln \frac{1}{\delta}$, with all cells set to 0.

Each time a new instance arrives in the data stream S, and for each attribute and class, each hash function h_i is applied and the corresponding counter (in the range $1, \ldots, w$) is incremented. When an instance needs to be classified, the corresponding counts using the same hash functions need to be retrieved, i.e., when training the classifier, one needs to look at all cells for the attribute and the class being estimated. This is done by taking the minimum overall values in the corresponding cells. This is because, since each cell has been incremented each time an attribute has been seen, each cell represents an upper bound on the actual value.

Assuming a data stream with N arrivals, let c_i be the true count of an item being estimated. It has been shown in [16] that the estimated count for an item i is at least c_i since all the inserts are non-negative, and due to collisions, the counts can be over-estimated to at most $c_i + \epsilon \cdot N$ with probability at least $1 - \delta$, i.e., an upper bound to the estimate.

2) Hashing Trick: Hashing Trick (HT), also known as feature hashing [11], is another popular data summarization technique for dimensionality reduction. It is used when dealing with a massive number of features, i.e., sparse high-dimensional data, and we want to compress that down to a few features. To do so, hashing trick has been used to make the analysis of sparse and large data tractable in practice. The idea behind the hashing trick is presented in Figure 2, where the sparse feature values are mapped into a lower dimensional feature space; we then have a space-efficient way of vectorizing features, from the original input space consisting in a dimensions to a new vector of m dimensions, where $m \ll a$. If we have a list of keys that represent features from the input instance, then, for each key we can calculate the hash function. After applying the hash function, each key is mapped to a specific cell of a fixed size hash table that constitutes a much lower-dimensional representation of the input vector. So, these cells in the vector store frequencies which are the keys'

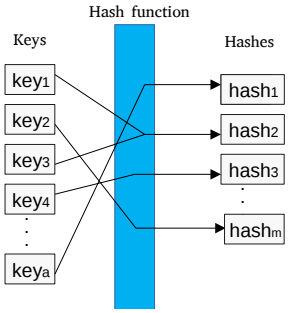

Fig. 2: Hashing trick.

counts, i.e., its index and its position in the feature vector.

An important point to make is that, generally, the quality of models changes when the size of the hash table increases. Usually, the larger the hash table size is, the better is the model. However, an optimal point can be picked which guarantees almost perfect model, while the output dimension size is not to be very large.

The hashing trick has the appealing properties of being very fast, simple, and sparsity preserving. An important advantage to point out is that this technique is very memory efficient because the feature vector size is limited, making it a clear candidate for using, especially for online learning on streams.

III. Related Work

The problem of classification has been studied for both offline and online settings. An important issue to address in the data stream scenario is the computational efficiency of classifiers because of the potentially infinite nature of data stream. Quite a number of classification algorithms for static datasets that have already been thoroughly studied have been extended to handle evolving distributions.

A well-known decision tree learner for the data stream is the Hoeffding tree algorithm [3], also known as Very Fast Decision Tree (VFDT). It is an incremental, anytime decision tree induction algorithm that uses the Hoeffding bound to select the optimal splitting attributes.

However, this learner assumes that the distribution generating instances does not change over time. So, to cope with an evolving data stream, a drift detection algorithm is usually coupled with it. In [18] an adaptive algorithm was proposed, Hoeffding Adaptive Tree (HAT), extending the VFDT to deal with concept drifts. It uses ADWIN, a change detector and estimator, to monitor the performance of branches on the tree and to replace them with new branches when their accuracy decreases if the new branches are more accurate [18]. These algorithms require more memory with the growth of tree expansion, also, waste computational speed due to the time spent in choosing the optimal attribute to split.

K-Nearest Neighbors (KNN) is another algorithm that has been adapted to the data stream setting. It does not require any work during training but it uses the entire dataset to predict the class labels for test examples. The challenge with adapting KNN to the stream setting is that it is not possible to store the entire stream for the prediction phase. An envisaged solution to solve this issue is to manage the examples that are remembered so that they fit into limited memory and to merge new examples with the closest ones already in memory. Yet, searching for the nearest neighbors still costly in terms of time and memory [19].

Little research has focused on using efficient data structures designed to reduce memory usage such as CMS with standard classifiers. Kveton et al. [20] proposed three graphical model sketches algorithms that estimate the marginal and the joint probabilities within a Bayesian network. Authors experimented with the special case of Bayesian networks, naive Bayes. After analyzing them, it was proved that GM-FactorSketch is the best approximation. The main idea of the algorithm is to use $2a-1$ sketch tables, one for each variable and one for each variable-parent pair in the graph. Given a test example, it retrieves the approximated count for each attribute from each corresponding sketch tables to compute the conditional probabilities and to predict thereafter the class label. None of the above-mentioned algorithms are efficient in terms of memory with large datasets, as evidenced in our experimental section.

IV. Sketch-Based Naive Bayes Algorithms

The main idea behind the sketch-based algorithms is the application of CMS for memory efficiency.

As discussed in the previous section, sketch-based techniques summarize massive data streams using limited space by using multiple hash functions to decrease the probability of having wrong counts due to collisions.

A. SketchNB Algorithm

Our first contribution in this paper attempts to adapt the CMS to the classic naive Bayes classifier by leveraging its strong theoretical guarantees.

The independence assumption in naive Bayes means that each attribute can be counted separately; this simplifies the learning for a large number of attributes, and allows us to use the sketch efficiently.

During the classification process, the sketch table will be used in two steps:

- *Learning*: updating the sketch table for each attribute each time a new instance arrives.
- *Prediction*: retrieving the counts of a given instance using the CMS, and use them to compute the naive Bayes probability (see equation 1).

Let us start by discussing the learning process using the CMS, as described in Algorithm 1. Once an instance $X_i = (x_i^1, x_i^2 ... x_i^a)$ is received, the classification algorithm starts by updating the sketch with the counts of the attributes value by inserting each of the attribute value as $\langle a, x_i^a, c \rangle$. The sketch table will thus contain the counts of the attributes values, using each of the d hash functions a times according to the number of attributes, so $O(d \cdot a)$ times in total.

Figure 1 shows the updating process of the sketch table where an attribute value x_i^j is mapped to one counter in each row using hash functions. Each of those counts gets incremented whenever a particular similar attribute value in the same class is seen. Therefore, each of those numbers is going to be an upper bound. Since all of them are going to be an upper bound, only the minimum can be taken for the prediction phase using the same hash functions used during the update process.

Algorithm 1 Learning phase: SketchNB Updates

procedure UPDATESKETCH(labeled data stream: S, epsilon: ϵ, delta: δ)

Create the sketch with $w \cdot d$, $\triangleright$ $w = \left\lceil \frac{e}{\epsilon} \right\rceil$, $d = \left\lceil ln\frac{1}{\delta} \right\rceil$

while There are instances in S **do**

For each attribute value x_i^k from instance X_i in S,

Increment the corresponding cells by 1 using d hash functions in the range $[1..w]$.

Obverse that Algorithm 2 assumes that we have one stream used to present training instances, and a stream $S\prime$ for predictions; this works by presenting, at each timestamp j, an unlabeled instance X_j. To predict the class, we need to use equation (1), the counts in the CMS table, and thus compute an estimation of the class having the highest probability.

Algorithm 2 Prediction phase: Sketch Estimations

procedure ESTIMATESKETCH(data stream: $S\prime$, count-min sketch)

For each attribute value from the received test instance in $S\prime$,

Retrieve the counts for all the a attributes given the attribute value and attribute index for each class label in C,

Compute the probabilities for all the classes using Bayes equation (1),

Report the class label to the instance with the highest probability.

To determine the efficiency of the proposed SketchNB algorithm, we need to analyze the behavior of the sketch table since we are retrieving approximate counts from it. It has been shown that for all the inserts, the counts are non-negative and may be over-estimated because of collisions [16].

Let $f_j(x_i^k)$ be the fractional count of the attribute value x_i^k from the instance X_i in the jth class. Its estimated fractional count is denoted by $\hat{f}_j(x_i^k)$.

The estimate $\hat{f}_j(x_i^k)$ has the following guarantees: $f_j(x_i^k) \leq \hat{f}_j(x_i^k)$; and with probability at least $(1-\delta)$,

$$\hat{f}_j(x_i^k) \leq f_j(x_i^k) + N \cdot a \cdot \epsilon \tag{2}$$

Using one sketch table with size $\left\lceil \frac{e}{\epsilon} \times ln\frac{1}{\delta} \right\rceil$, after the processing of N a-dimensional data in the stream, the counts are over-estimated to within $N \cdot a \cdot \epsilon$ of their true values. Since we are using naive Bayes, as described in the second step of Algorithm 2, for each incoming instance, each class c from the set C, and each attribute a, we are doing multiple extractions at the same time by retrieving $m = C \cdot a$ counts:

Theorem 1. With probability at least $1 - \Delta = (1-\delta)^m$ we have:

$$\bigwedge_{k=1}^{m} (\hat{f}_j(x_i^k) \leq f_j(x_i^k) + Na\epsilon) = \text{True} \tag{3}$$

This means that we need to set in the sketch,

$$\delta = 1 - \sqrt[m]{1-\Delta} \tag{4}$$

or

$$d = \ln \frac{1}{1 - \sqrt[m]{1-\Delta}} \tag{5}$$

The above result will be used to set a crucial parameter to construct the sketch which is δ in order to fix the depth of the sketch table. The obtained δ will lead to a deeper sketch that would be able to maintain the entire stream. Consequently, we will obtain a more accurate estimation of counts by avoiding collisions.

We would like to determine the accuracy of SketchNB algorithm. The process of this algorithm is described in the pseudocode of Algorithm 2. It consists on retrieving, from the sketch table, the fractional counts for each attribute value so one can apply the naive Bayes equation for each class label. In order for this to happen, we need to compute the product of the estimated fractional counts for each hash function in $1, \ldots, \ln \frac{1}{\delta}$.

Theorem 2. We have:

$$\prod_{k=1}^{a} \hat{f}_j(x_i^k) \leq \prod_{k=1}^{a} (f_j(x_i^k) + Na\epsilon) \tag{6}$$

Then, with probability at least $(1-\Delta)$,

$$\prod_{k=1}^{a} \hat{f}_j(x_i^k) \leq \prod_{k=1}^{a} f_j(x_i^k) + \sum_{p=1}^{a} \frac{a!}{p!(a-p)!} (\epsilon\prime)^p \tag{7}$$

Proof. Let $\epsilon\prime = Na\epsilon$, and $\hat{f}_j(x_i^k)$ be the estimated fractional count of the attribute value x_i^k in the jth cluster. Given an instance X_i:

$$\hat{f}_j(x_i^1) \cdot \hat{f}_j(x_i^2) \ldots \cdot \hat{f}_j(x_i^a) \leq (f_j(x_i^1) + \epsilon\prime) \cdot (f_j(x_i^2) + \epsilon\prime) \tag{8}$$

$$\begin{aligned} &\ldots \cdot (f_j(x_i^a) + \epsilon\prime) \\ &= f_j(x_i^1) \ldots f_j(x_i^a) + f_j(x_i^1) \cdot \epsilon\prime \\ &+ f_j(x_i^2) \cdot \epsilon\prime \ldots + f_j(x_i^a) \cdot \epsilon\prime \\ &+ f_j(x_i^1) \cdot f_j(x_i^2) \cdot \epsilon\prime \ldots \end{aligned} \tag{9}$$

Since all the frequencies are non-negative, we know that the range of possible fractional counts is $[0, 1]$, i.e. at most 1, thus, 1 will be an upper bound to the fractional counts. So, with certainty we know that $\hat{f}_j(x_i^k) \cdot \epsilon\prime \leq \epsilon\prime$, and any product

of the estimated fractional counts is always less than 1. So, by applying Pascal's triangle, we obtain:

$$\begin{aligned}\prod_{k=1}^{a}\hat{f}_j(x_i^k) &\leq \prod_{k=1}^{a} f_j(x_i^k) + f_j(x_i^1)\cdot\epsilon\prime + f_j(x_i^2)\cdot\epsilon\prime ... \\ &+ f_j(x_i^a)\cdot\epsilon\prime + f_j(x_i^1)\cdot f_j(x_i^2)\cdot\epsilon\prime ... \\ &\leq \prod_{k=1}^{a} f_j(x_i^k) + \sum_{p=1}^{a}\frac{a!}{p!(a-p)!}(\epsilon\prime)^p \\ &= \prod_{k=1}^{a} f_j(x_i^k) + \sum_{p=1}^{a} C_a^p(\epsilon\prime)^p \qquad (10)\end{aligned}$$

This completes the proof. □

This observation shows that, with probability $1-\Delta$, the quantity of error due to collisions is at worst equal to $\sum_{p=1}^{a} C_a^p(\epsilon\prime)^p$ which leads to the following corollary:

Corollary 1. Let E be big epsilon and N be the size of the stream. An immediate result from equation (10) is the following:

$$\epsilon = \frac{\sqrt[a]{aNE+1}-1}{aN} \qquad (11)$$

or

$$w = \frac{eaN}{\sqrt[a]{aNE+1}-1} \qquad (12)$$

Proof. Let $\epsilon\prime = aN\epsilon$. After processing N a-dimensional instances in the stream, the quantity of error caused by collisions in Theorem 2 must be the same as aNE according to Theorem 1.

$$\begin{aligned}\sum_{p=1}^{a} C_a^p(aN\epsilon)^p &= aNE \\ C_a^0(aN\epsilon)^0 + \sum_{p=1}^{a} C_a^p(aN\epsilon)^p &= aNE+1 \\ \sum_{p=0}^{a} C_a^p(aN\epsilon)^p &= aNE+1 \\ \sum_{p=0}^{a} C_a^p(aN\epsilon)^p 1^{a-p} &= aNE+1 \\ (aN\epsilon+1)^a &= aNE+1 \\ aN\epsilon+1 &= \sqrt[a]{aNE+1} \\ \epsilon &= \frac{\sqrt[a]{aNE+1}-1}{aN}\end{aligned}$$

This completes the proof. □

Our proposed SketchNB algorithm possesses strong theoretical guarantees when setting the depth and width of the sketch using equations (11) and (4) respectively. This means that, in order to keep the guarantees overall attributes and classes, we need to set a much deeper and wider sketch table than for only one counter.

Moreover, equations (11) and (12) assume the size of the stream is known. In real-world scenarios, where the stream can be infinitely increasing, this means that our sketch, along with the hash functions, will need to increase with the size of the stream.

The theoretical parameters derived by us here provide good results, but they can lead to a relatively large sketch; in some cases, this may not be desirable due to space reasons. We can however perform some optimizations to the space needed. The size of the stream, N, can be too large and even infinite; instead, it can simply be a "sliding window" over which error guarantee is provided. To achieve this, we introduce a scaling constant b to the computations, in the following manner. First, we can set ϵ as follows:

$$\epsilon = b\cdot\frac{\sqrt[a]{aNE+1}-1}{aN},$$

then choose the width w as follows:

$$w = \frac{eaN}{b(\sqrt[a]{aNE+1}-1)}. \qquad (13)$$

Note that the depth d still remains the same: it depends only on the parameter Δ. It is also necessary to point out that $b>1$. When a increases, the sketch table size increases accordingly. A higher value of b reduces the width of the sketch table. In this work, b will be picked up experimentally..

B. AdaSketchNB Algorithm

One crucial issue when dealing with a very large stream is the fact that the underlying distribution of the data can change at any moment, a phenomenon known as *concept drift*, and we direct the reader to [21] for a survey of this concept.

A widely popular algorithm that handles concept drift is ADaptive WINdowing (ADWIN) [10], a change detector and estimator, also used in a few machine learning algorithms such as Hoeffding adaptive tree [18] and leveraging bagging [9].

The main idea of ADWIN is to maintain a variable-length window W with the most recently seen instances with the property that the window has the maximal length statistically consistent with the hypothesis "there has been no change in the average value inside the window" [10].

In our context, it is also important to incorporate a change detector, and we choose ADWIN here due to its strong theoretical guarantees and good practical results.

Against this background, we propose a second algorithm, AdaSketchNB described in Algorithm 3, that incorporates to the SketchNB algorithm an adaptive strategy using ADWIN change detector in the learning phase, for the entire sketch table, in order to track drifts. It works on the data distribution by detecting the change in the class label.

Given a stream S, when a new training instance $X_i = (x_i^1, x_i^2 ... x_i^a)$ arrives, we compare the predicted class label (under our current model) and the true class label. Then, we update the sliding window W by adding 1 if this comparison holds true, 0 otherwise. Once a change is detected in the distribution, i.e., the newly generated instances changed from the original class distribution of the classifier built up to now,

Algorithm 3 Learning phase: AdaSketchNB Updates

procedure UPDATESKETCH(labeled data stream: S, Sliding window: W, epsilon: ϵ, delta: δ)
begin
Create the sketch with $w \cdot d$, $\triangleright$ $w = \left\lceil \frac{e}{\epsilon} \right\rceil$, $d = \left\lceil ln\frac{1}{\delta} \right\rceil$
while there are instances in S **do**
For each attribute value x_i^k from instance X_i in S,
Increment the corresponding cells by 1 using d hash functions in the range $[1..w]$,
if true class = predicted class **then**
Add 1 to W,
else
Add 0 to W.
if a change is detected **then**
Reset the sketch table and W.

the sketch table and W will be re-initialized to learn the new model, corresponding to the new distribution. The prediction phase remains the same as the basic SketchNB algorithm.

C. SketchNB$_{HT}$ and AdaSketchNB $_{HT}$ Algorithms

It is an undeniable fact that the sketches summarize massive data streams within a limited space, but hashing trick can be used for further optimization with large datasets.

Our third contribution, SketchNB$_{HT}$ and AdaSketchNB$_{HT}$ algorithms, attempts to enhance the SketchNB and the AdaSketchNB in terms of memory and processing time while maintaining high accuracy. Indeed, it could perform better by integrating the hashing trick technique.

In fact, to make the analysis of high-dimensional and large datasets tractable, firstly, we adjust the size of the input dimension by applying the hashing trick algorithm to instances one by one. We obtain thereafter instances with significantly smaller dimension than the original one, then we process them by either the update procedure in Algorithm 1 or 3 if we are dealing with SketchNB$_{HT}$ or AdaSketchNB$_{HT}$ respectively.

By applying the hashing trick on an input vector, we are supposed to obtain a numerical representation (Section II-C2) which prevents it from fitting to the sketch table. Instead, we are obtaining a binary vector by updating a cell only once, i.e., we do not increment the cells, we just put 1 if a cell contains 0, and if it contains already 1, it remains the same. Thus, we get a discretized representation able to fit in the sketch table and also avoid wrong values due to collisions.

Then, we store the instances in the sketch table in the same way as the SketchNB algorithm. So, instead of updating a times the sketch table, we update it henceforth only m times, where $m \ll a$.

In other words, the hashing trick is treated as an internal preprocessing and transforming instances within the SketchNB and AdaSketchNB algorithms. Using this manner, we guarantee that out of memory error that may occur with standard classifiers for data streams will not happen using the hashing trick and the sketches.

Table I: Overview of the datasets

Dataset	#Instances	#Attributes	#Classes	Type
SEA	1,000,000	3	2	Synthetic
RBF	1,000,000	10	5	Synthetic
LED	1,000,000	24	10	Synthetic
LED$_g$	1,000,000	24	10	Synthetic
AGR	1,000,000	9	2	Synthetic
HYP	1,000,000	10	2	Synthetic
Tweets$_1$	10,000	1,000	2	Synthetic
Tweets$_2$	10,000	10,000	2	Synthetic
KDD99	494,020	41	23	Real
Cover	581,012	54	7	Real
Elec	45,312	8	2	Real
Poker	829,201	10	10	Real
Enron	1,702	1,054	2	Real
IMDB	120,919	1,001	2	Real
CNAE	1,080	856	9	Real

V. EXPERIMENTS

We conduct several experiments to assess the classification performance of the aforementioned proposals, SketchNB, AdaSketchNB, SketchNB$_{HT}$ and AdaSketchNB$_{HT}$ algorithms. To evaluate them, we are interested in three main dimensions; the accuracy as the final percentage of instances correctly classified, the memory (bytes), and the time (seconds) required to learn and predict from data.

A. Datasets

We use 8 synthetic and 7 real world datasets on our experiments, where 5 of them contain high-dimensional data. The synthetic datasets created using the data generators provided by MOA [22] include drifts. In the case of real datasets, we do not know whether a drift exists or not, but we still evaluate it using the ADWIN variants of the algorithms.

Table I presents a short description of each dataset, and further details are provided in what follows.

SEA. The SEA Generator proposed by [23]. It is generated with 3 attributes and 2 decision classes with concept drift and simulates 3 gradual drifts.

RBF. The RBF (Radial Basis Function) generator creates centroids at random positions, and each one has a standard deviation, a weight and a class label. This dataset simulates drift by moving the centroids with constant speed.

LED. The LED generator originates from the CART book [24] and simulates concept drifts. It produces 24 attributes, of which 17 are irrelevant. The goal is to predict the digit displayed on the LED display. We generate also LED$_g$ that simulates 3 gradual drifts.

AGR. The AGRAWAL generator [25] creates data stream with 9 attributes and 2 classes. A factor is used to change the original value of the data. AGR is used to simulate 3 gradual drift in the generated stream.

HYP. The HYPERPLANE generator [26] used to generate streams with gradual concept drift by changing the values of its weights. We parameterize HYP with 10 attributes and a magnitude of change equals to 0.001.

Tweets. Tweets is a text generator that simulates sentiment analysis on tweets, where messages can be classified into two categories depending on whether they convey positive or negative feelings. Tweets_1 and Tweets_2 produce 1,000 and 10,000 attributes respectively.

KDD99. KDD cup'99[1] for network intrusion detection. This dataset contains 41 attributes and 23 classes. It has been often used to evaluate big data streams algorithms' performance.

Cover. The forest covertype dataset obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. It contains 54 attributes and 7 classes.

Elec. The Electricity market dataset described firstly by [27]. In this marked the prices changes every 5 minutes and are affected by demand, supply, season, weather and time. It contains two possible class labels identifying the changes of the price relative to a moving average of the last 24h.

Poker. The Poker hand dataset consists of 829,201 instances and 10 attributes. Each instance of the Poker-Hand dataset is an example of a hand consisting of five playing cards.

Enron. The Enron corpus dataset is a large set of email messages that was made public during the legal investigation concerning the Enron corporation [28]. This cleaned version of Enron consists of 1,702 instances and 1,054 attributes.

IMDB. IMDB[2] movie reviews dataset was first proposed for sentiment analysis [29], where reviews have been preprocessed, and each review is encoded as a sequence of word indexes (integers).

CNAE. CNAE is the national classification of economic activities dataset, initially used in [30]. 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.

Handling continuous attributes in data stream classifiers is a bit tricky; one way to deal with this issue is to consider that the datasets will follow a series of Gaussian distributions. We use a simpler way here to preprocess the datasets and transform all numeric attributes into discrete attributes. The discretization was performed using WEKA [31], where each numerical attribute was discretized to an equal-width histogram having 10 bins.

B. Results

The experiments were performed, implemented and evaluated in JAVA using the MOA framework [22] and the datasets explained in Section V-A. They were conducted on a machine equipped with an Intel Core i5 CPU speed of 2.60GHz processor and 4 GB of main memory.

Despite theoretical bounds on the size of the sketch, we need to optimize the sketch table parameters to allow better space usage. To do so, we use both of the synthetic and real datasets to parameterize the sketch table size for each dataset, by controlling the parameter b.

[1]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[2]http://waikato.github.io/meka/datasets/

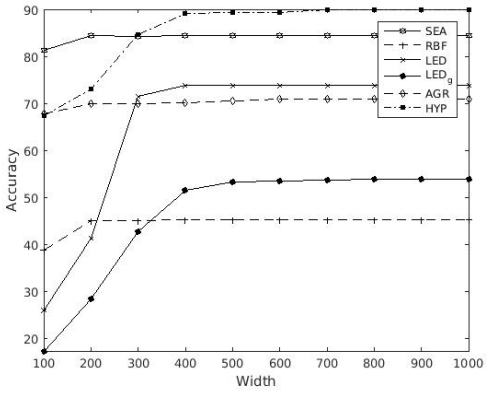

Fig. 3: Classification accuracy with different size of the sketch for different synthetic datasets.

Table II: Accuracy (%) comparison of SNB, GMS, NB, KNN, ASNB, AdNB, and HAT. Bold values indicate the best results per dataset.

Dataset	Non-adaptive				Adaptive		
	SNB	GMS	NB	KNN	AdSNB	AdNB	HAT
SEA	**84.64**	70.83	**84.64**	69.95	**86.70**	**86.70**	**86.70**
RBF	44.66	32.62	**45.43**	35.14	47.14	45.91	**82.73**
LED	**73.94**	73.82	**73.94**	44.02	**73.90**	**73.90**	70.87
LED_g	54.02	**54.23**	54.02	43.18	72.71	**73.09**	72.60
AGR	70.95	61.67	**70.96**	68.19	79.98	82.53	**89.34**
HYP	**90.16**	81.93	**90.16**	64.18	90.88	**91.16**	81.32
Tweet_1	87.50	-	**89.26**	68.73	87.54	**89.26**	84.64
Tweet_2	74.42	-	**91.41**	77.72	74.42	**91.41**	85.38
Syn ∅	72.54	-	**75.23**	58.89	76.66	79.24	**81.73**
KDD99	89.72	97.43	95.51	**99.69**	91.17	**99.59**	98.93
Cover	62.08	50.69	62.86	**80.07**	**95.80**	83.17	86.56
Elec	66.64	63.90	66.53	**73.89**	71.70	**73.31**	72.48
Poker	56.81	56.43	58.84	**74.98**	69.22	74.58	**74.73**
Enron	75.50	-	77.44	**95.24**	84.61	85.61	**91.83**
IMDB	64.98	-	68.38	**70.42**	68.52	70.67	**70.71**
CNAE	56.30	-	**62.13**	62.13	56.30	62.13	**68.80**
Real ∅	67.43	-	70.24	**79.49**	76.76	78.43	**80.55**
O. ∅	70.16	-	**72.90**	68.50	76.71	78.86	**81.18**

In order to fix the value of the constant b in equation (13), we perform some experiments. We set the default values for different parameters to the sketch table, $\Delta = 0.1$, $E = 0.01$ and $N = 10^5$ for equations (5) and (12) to fix the sketch table size, and we set the default confidence bound to ADWIN.

Figure 3 illustrates the appropriate width for the first 6 synthetic datasets, i.e., what is the basic width that leads to an accurate model for each dataset. We notice that for the same number of attributes and classes, e.g., LED and LED_g, we obtain the same width that is able to maintain the entire data stream. Therefrom we can fix the value of the constant b for each dataset experimentally reported in Table IV.

With the parameters set, Tables II, III and IV present the results on all datasets. We compared SketchNB (SNB) and AdaSketchNB (AdSNB) classifiers to well-known state-of-the-art algorithms: the Naive Bayes (NB), the K-Nearest Neighbor (KNN) with $k = 100$, the GMFactorSketch (GMS) with default parameters $\epsilon = 0.01$, $\delta = 0.1$. We compared also to adaptive classifiers such as the Hoeffding Adaptive Tree (HAT), and the adaptive NB (AdNB) with ADWIN.

Table III: Memory (B) comparison of SNB, GMS, NB, KNN, ASNB, AdNB, and HAT. Bold values indicate the best results per dataset.

Dataset	Non-adaptive				Adaptive		
	SNB	GMS	NB	KNN	AdSNB	AdNB	HAT
SEA	5,568	49,312	**4,880**	58,280	7,832	**7,704**	2.48E6
RBF	18,824	226,072	**18,520**	100,920	**19,768**	20,688	3.61E6
LED	**23,568**	644,544	27,704	187,896	**22,792**	30,168	1.27E6
LED_g	**23,568**	644,544	27,704	187,896	**22,792**	30,528	623,856
AGR	19,360	201,000	**12,520**	92,760	**14,992**	15,344	3.85E6
HYP	20,624	225,952	**15,248**	100,824	**17,264**	17,568	925,600
$Tweet_1$	**569,880**	-	645,456	7.52E6	**582,144**	647,440	1.71E6
$Tweet_2$	**6.03E6**	-	6.55E6	7.5E7	**6.04E6**	6.39E6	1.23E7
Syn ∅	**839,598**	-	912,687	1.04E7	**840,600**	895,121	3.38E6
KDD99	**106,632**	1.32E6	123,496	303,200	108,456	61,088	**57,048**
Cover	**55,736**	1.71E6	60,488	377,832	61,440	46,344	**19,936**
Elec	14,776	178,464	**11,944**	88,712	14,560	**13,120**	42,432
Poker	**16,568**	223,952	19,800	98,760	18,608	**17,688**	278,528
Enron	**565,104**	-	688,752	7.13E6	534,264	690,400	**275,624**
IMDB	**1.41E6**	-	1.57E6	7.53E6	**1.37E6**	1.54E6	2.91E6
CNAE	**1.16E6**	-	1.58E6	1.58E6	1.23E6	1.58E6	**571,064**
Real ∅	**474,706**	-	578,263	2.44E6	**554,384**	563,117	593,306
O. ∅	**221,977**	-	270,343	6.69E6	**707,033**	740,186	2.08E6

Table IV: Time (sec) comparison of SNB, GMS, NB, KNN, ASNB, AdNB, and HAT. Bold values indicate the best results per dataset.

Dataset	Non-adaptive				Adaptive			b
	SNB	GMS	NB	KNN	AdSNB	AdNB	HAT	
SEA	1.75	2.08	**1.49**	10.91	2.87	**2.03**	7.75	675
RBF	6.62	9.65	**3.73**	31.2	9.6	**4.5**	17.67	7,191
LED	15.9	41.03	**5.67**	64.21	30.79	**7.28**	21.08	14,692
LED_g	15.93	43.45	**5.73**	68.83	29.23	**7.25**	23.09	14,692
AGR	4.09	5.94	**3.1**	26.05	5.47	**3.57**	9.12	3,678
HYP	4.51	6.42	**3.42**	27.79	4.11	**3.77**	9.8	5,448
$Tweet_1$	4.79	-	**4.17**	29.21	6.77	**2.89**	5.57	7,514,600
$Tweet_2$	73.44	-	**51.04**	385.23	91.43	**44.78**	65.93	6.48E8
Syn ∅	15.87	-	**9.79**	80.43	22.53	**9.51**	20	-
KDD99	40.02	90.75	**12.78**	51.66	64.42	**7.86**	18.11	15,074
Cover	18.92	50	**7.67**	64.42	32.74	**7.09**	19.55	59,675
Elec	0.33	0.42	**0.18**	1.06	0.35	**0.22**	0.67	4,555
Poker	6.87	12.3	**2.36**	22.03	11.67	11.67	**7.95**	4,859
Enron	0.98	-	**0.63**	6.07	1.29	**0.62**	0.97	15,340,000
IMDB	53.45	-	**27.07**	273.73	68.36	**32.89**	113.38	6,525,300
CNAE	1.38	-	**0.52**	0.67	2.26	**0.65**	1.71	7,231,800
Real ∅	17.41	-	**7.31**	59,95	25.8	**8.71**	23.2	-
O. ∅	16.54	-	**8.63**	70.87	24.06	**9.14**	21.49	-

It turns out that some classifiers are useless and could not process some datasets which are marked by the cells with "-".

We observe that the accuracy of SketchNB is almost the same when comparing to NB for all the datasets, even despite using probabilistic counts to estimate the fractional counts. In comparison with KNN, we notice that SketchNB is more accurate for all the datasets except the real ones. Such difference can be explained by the independence assumptions between attributes of NB.

Since GMS builds two sketch tables for each attribute, it is obvious that it will be more memory and time consuming because we are using only one big sketch. More than this, GMS cannot work with large datasets, e.g., $Tweet_1$.

In order to simulate the proposed change detector classifier AdaSketchNB, we compare against the AdNB and the HAT. In Table II, on overall average, we observe that AdaSketchNB achieves, practically, the same accuracy as AdNB whilst using a less amount of resources (see Table III). In comparison with the HAT, the gain in memory exceeds the slight loss in accuracy. This is due to using, in addition to the sketch, the ADWIN change detector and estimator [10].

Table V: Accuracy (%) comparison of SNB_{HT}, GMS_{HT}, NB_{HT}, KNN_{HT}, $ASNB_{HT}$, $AdNB_{HT}$, and HAT_{HT}. Bold values indicate the best results per dataset.

Dataset	Non-adaptive				Adaptive		
	SNB_{HT}	GMS_{HT}	NB_{HT}	KNN_{HT}	$ASNB_{HT}$	$AdNB_{HT}$	HAT_{HT}
$Tweet_1$	77.99	61.84	**78.80**	68.04	77.99	**78.80**	77.53
$Tweet_2$	79.25	74.76	**79.66**	75.27	79.25	**79.66**	78.96
Enron	70.22	-	71.69	**89.54**	**83.14**	76.11	-
IMDB	68.40	-	69.98	**70.27**	69.73	69.55	**70.44**
CNAE	58.38	53.24	**64.20**	47.05	58.92	**64.20**	62.81
O ∅	70.85	-	**72.87**	70.03	**73.80**	73.67	-

To assess the benefits in terms of resources usage we observe the behavior of memory results is similar to the behavior of time results, i.e., when the memory usage increases, the time processing also increases. For some datasets, NB is more space-efficient than SketchNB (especially for datasets with a low number of attributes and classes,e.g., SEA, AGR) which is quite natural as simpler learners usually require less time for training and prediction. With large datasets (in terms of the number of attributes and classes), e.g. $Tweet_1$ and $Tweet_2$, SketchNB and AdaSketchNB consume fewer resources than NB and AdNB respectively due to the use of a space-saving structure, the CMS. In comparison with GMS, KNN, and HAT, the proposed algorithms are more space and time efficient.

Despite the gain with SketchNB and AdaSketchNB classifiers over different datasets, still wholly not satisfying. Therefore, we proposed $SketchNB_{HT}$ (SNB_{HT}) and $AdaSketchNB_{HT}$ ($ASNB_{HT}$), a third contribution that consists on preprocessing internally instances of SketchNB and AdaSketchNB coupled with the hashing trick.

Figure 4 presents the results of these experiments, where each dataset is processed with 6 different output dimension. We report in Tables V, VI and VII the overall average accuracy, memory and processing time over the different values of dimension for each dataset.

For all datasets with different space dimension, the $SketchNB_{HT}$'s accuracy is similar to NB using a feasible amount of resources. E.g., Figure 6a depicts the ability of $SketchNB_{HT}$ and $AdaSketchNB_{HT}$ to achieve similar accuracy to NB and AdNB on $Tweet_1$, and in Figure 6b and 6c we can see that they lead to large memory savings in lower time processing.

Comparing the results from Figure 4 and its corresponding average (Tables V, VI and VII), $SketchNB_{HT}$ outperforms also GMS_{HT} and KNN_{HT} (excepts for accuracy with this latter). Nevertheless, the gain in memory and time is more interesting. Also with the adaptive classifiers, $AdaSketchNB_{HT}$ is more accurate than HAT_{HT} for some datasets and uses much less memory.

VI. Conclusion

In this paper, we presented SketchNB, AdaSketchNB, $SketchNB_{HT}$, and $AdaSketchNB_{HT}$ algorithms, new classifiers for big data streams. The SketchNB algorithm extends the

Table VI: Memory (B) comparison of SNB_{HT}, GMS_{HT}, NB_{HT}, KNN_{HT}, ASNB_{HT}, AdNB_{HT}, and HAT_{HT}. Bold values indicate the best results per dataset.

Dataset	Non-adaptive				Adaptive		
	SNB_{HT}	GMS_{HT}	NB_{HT}	KNN_{HT}	ASNB_{HT}	AdNB_{HT}	HAT_{HT}
Tweet_1	**3,648**	1,001,109	6,817	236,328	**5,472**	8,801	76,896
Tweet_2	**3,721**	1,001,109	6,817	236,328	**5,545**	8,801	62,404
Enron	**3,619**	-	7,760	265,373	**4,381**	9,341	-
IMDB	**4,528**	-	6,817	236,288	**5,580**	8,157	93,113
CNAE	**14,133**	1,001,165	19,623	236,848	**15,935**	21,103	23,689
O ∅	**5,930**	-	9,567	242,233	**7,382**	11,241	-

Table VII: Time (%) comparison of SNB_{HT}, GMS_{HT}, NB_{HT}, KNN_{HT}, ASNB_{HT}, AdNB_{HT}, and HAT_{HT}. Bold values indicate the best results per dataset.

Dataset	Non-adaptive				Adaptive		
	SNB_{HT}	GMS_{HT}	NB_{HT}	KNN_{HT}	ASNB_{HT}	AdNB_{HT}	HAT_{HT}
Tweet_1	**2.66**	3.03	3.16	3.59	**2.84**	2.84	2.97
Tweet_2	**29.72**	43.06	43.48	37.93	**29.63**	33.28	34.11
Enron	**0.42**	-	0.47	0.61	0.44	**0.41**	-
IMDB	14.59	-	**13.54**	25.57	14.38	**13.58**	14.75
CNAE	0.44	0.51	**0.40**	0.52	0.46	**0.41**	0.49
O ∅	**9.55**	-	12.2	13.64	9.56	**9.51**	-

naive Bayes classifier using the count-min sketch to reduce the memory needed. Then we proposed AdaSketchNB, an adaptive version of SketchNB to handle concept drift. Finally, we coupled SketchNB and its adaptive version with the hashing trick technique for further gain with high-dimensional data to obtain SketchNB_{HT} and AdaSketchNB_{HT}. We explained the learning process of these classifiers using sketches showing strong theoretical guarantees.

We compared these classifiers in an extensive evaluation with well-known classifiers showing that GMS, NB, HAT, and KNN were outperformed in memory by SketchNB and AdaSketchNB with large datasets. We showed also that using the hashing trick and the count-min sketch, SketchNB_{HT} and AdaSketchNB_{HT} obtain good results in terms of classification performance (accuracy, memory and time) when compared to other state-of-the-art classifiers.

REFERENCES

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

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

[3] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2000, pp. 71–80.

[4] J. Gama, R. Rocha, and P. Medas, "Accurate decision trees for mining high-speed data streams," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 523–528.

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

[6] 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.

[7] F. Klawonn and P. Angelov, "Evolving extended naive bayes classifiers," in *Proceedings of the sixth International Conference on Data Mining Workshops*. IEEE, 2006, pp. 643–647.

[8] C. Salperwyck, V. Lemaire, and C. Hue, "Incremental weighted naive bays classifiers for data stream," in *Data Science, Learning by Latent Structures, and Knowledge Discovery*. Springer, 2015, pp. 179–190.

[9] 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.

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

[11] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 1113–1120.

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

[13] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 2002, pp. 346–357.

[14] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine, "Synopses for massive data: Samples, histograms, wavelets, sketches," *Foundations and Trends in Databases*, vol. 4, no. 1–3, pp. 1–294, 2012.

[15] G. Cormode, "Data sketching," *Queue*, vol. 15, no. 2, p. 60, 2017.

[16] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[17] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[18] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *International Symposium on Intelligent Data Analysis*. Springer, 2009, pp. 249–260.

[19] A. Bifet, B. Pfahringer, J. Read, and G. Holmes, "Efficient data stream classification via probabilistic adaptive windows," in *Proceedings of the 28th annual ACM symposium on applied computing*. ACM, 2013, pp. 801–806.

[20] B. Kveton, H. Bui, M. Ghavamzadeh, G. Theocharous, S. Muthukrishnan, and S. Sun, "Graphical model sketch," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2016, pp. 81–97.

[21] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 44, 2014.

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

[23] W. N. Street and Y. Kim, "A streaming ensemble algorithm (sea) for large-scale classification," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2001, pp. 377–382.

[24] W.-Y. Loh, "Classification and regression trees," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 14–23, 2011.

[25] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: A performance perspective," *IEEE transactions on knowledge and data engineering*, vol. 5, no. 6, pp. 914–925, 1993.

[26] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2001, pp. 97–106.

[27] M. Harries and N. S. Wales, "Splice-2 comparative evaluation: Electricity pricing," 1999.

[28] B. Klimt and Y. Yang, "The enron corpus: A new dataset for email classification research," in *European Conference on Machine Learning*. Springer, 2004, pp. 217–226.

[29] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *The 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*. Association for Computational Linguistics, 2011, pp. 142–150.

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

[31] G. Holmes, A. Donkin, and I. H. Witten, "Weka: A machine learning workbench," in *Proceedings of the second Australia and New Zealand Conference on Intelligent Information Systems,*. IEEE, 1994, pp. 357–361.

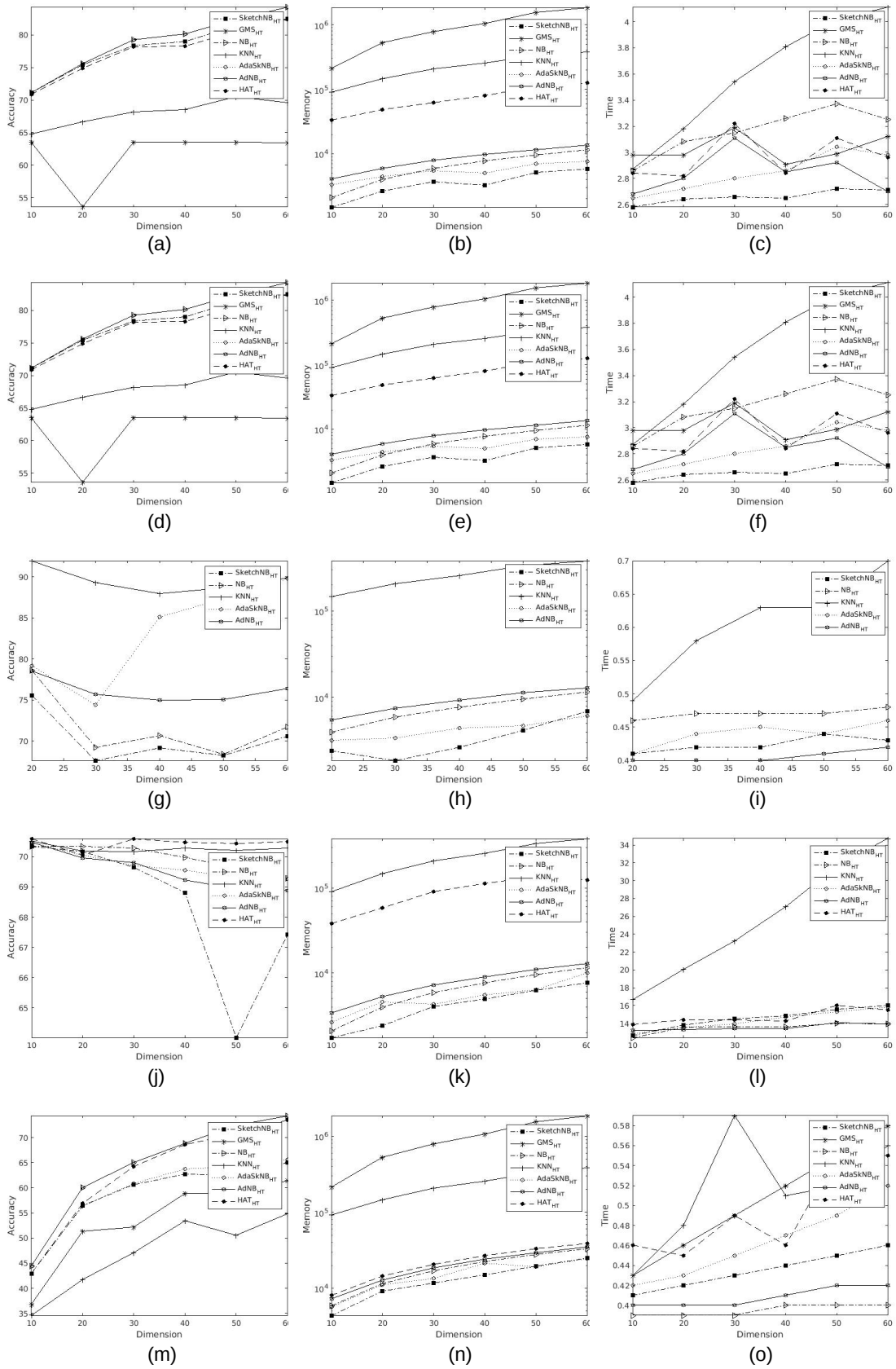

Fig. 4: Sorted plots of Accuracy, Memory and Time over the output dimension. **a** Accuracy Tweet$_1$. **b** Memory Tweet$_1$. **c** Time Tweet$_1$. **d** Accuracy Tweet$_2$. **e** Memory Tweet$_2$. **f** Time Tweet$_2$. **g** Accuracy Enron. **h** Memory Enron. **i** Time Enron. **j** Accuracy IMDB. **k** Memory IMDB. **l** Time IMDB. **m** Accuracy CNAE. **n** Memory CNAE. **o** Time CNAE.