Efficient Batch-Incremental Classification Using UMAP for Evolving Data Streams[*]

Maroua Bahri[1,3], Bernhard Pfahringer[2], Albert Bifet[1,2], and Silviu Maniu[3,4,5]

[1] LTCI, Télécom Paris, IP-Paris, Paris, France
{maroua.bahri,albert.bifet}@telecom-paris.fr
[2] University of Waikato, Hamilton, New Zealand bernhard@waikato.ac.nz
[3] Université Paris-Saclay, LRI, CNRS, Orsay, France
silviu.maniu@lri.fr
[4] Inria, Paris, France
[5] ENS DI, CNRS, École Normale Supérieure, Université PSL, Paris, France

Abstract. Learning from potentially infinite and high-dimensional data streams poses significant challenges in the classification task. For instance, k-Nearest Neighbors (kNN) is one of the most often used algorithms in the data stream mining area that proved to be very resource-intensive when dealing with high-dimensional spaces. Uniform Manifold Approximation and Projection (UMAP) is a novel manifold technique and one of the most promising dimension reduction and visualization techniques in the non-streaming setting because of its high performance in comparison with competitors. However, there is no version of UMAP that copes with the challenging context of streams. To overcome these restrictions, we propose a batch-incremental approach that pre-processes data streams using UMAP, by producing successive embeddings on a stream of disjoint batches in order to support an incremental kNN classification. Experiments conducted on publicly available synthetic and real-world datasets demonstrate the substantial gains that can be achieved with our proposal compared to state-of-the-art techniques.

Keywords: Data stream · k-Nearest Neighbors · dimension reduction · UMAP.

1 Introduction

With the evolution of technology, several kinds of devices and applications are continuously generating large amounts of data in a fast-paced way as *streams*. Hence, the data stream mining area has become indispensable and ubiquitous in many real-world applications that require real-time –or near real-time– processing, e.g., social networks, weather forecast, spam filters, and more. Unlike traditional datasets, the dynamic environment and the tremendous volume of data streams make them impossible to store or to scan multiple times [12].

Classification is an active area of research in data stream mining field where several researchers are paying attention to develop new – or improve existing– algorithms [14]. However, the dynamic nature of data streams has outpaced the capability of traditional classification algorithms to process data streams. In this context, a multitude of supervised algorithms for static datasets that have been widely studied in the offline processing, and proved to be of limited effectiveness on large data, have been extended to work within a streaming framework [3,5,11,18]. Data stream mining approaches can be divided into two main types [23]: (i) *instance-incremental* approaches which update the model with each instance as soon as it arrives, such as Self-Adjusting Memory kNN (sam$k$NN) [18], and Hoeffding Adaptive Tree (HAT) [4]; and (ii) *batch-incremental* approaches which make no change/increment to their model until a batch is completed, e.g., support vector machines [10], and batch-incremental ensemble of decision trees [15]. Nevertheless, the high dimensionality of data complicates the classification task for some algorithms and increases their computational resources, most notably the k-Nearest Neighbors (kNN) because it needs the entire dataset to predict the labels for test instances [23]. To cope with this issue, a promising approach is *feature transformation* which transforms the input features into a new set of features, containing the most relevant components, in some lower-dimensional space.

In attempt to improve the performance of kNN, we incorporate a batch-incremental feature transformation strategy to tackle potentially high-dimensional and possibly infinite batches of evolving data streams while ensuring effectiveness and quality of learning (e.g. *accuracy*). This is achieved via a new manifold technique that has attracted a lot of attention recently: Uniform Manifold Approximation and Projection (UMAP) [21], built upon rigorous mathematical foundations, namely Riemannian geometry. To the best of our knowledge, no incremental version of UMAP exists which makes it not applicable on large datasets. The main contributions are summarized as follows:

- *Batch-Incremental UMAP*: a new batch-incremental novel manifold learning technique, based on extending the UMAP algorithm to data streams.
- *UMAP-kNearest Neighbors (UMAP-kNN)*: a new batch-incremental kNN algorithm for data streams classification using UMAP.
- *Empirical experiments*: we provide an experimental study, on various datasets, that discusses the implication of parameters on the algorithms performance;

The paper is organized as follows. Section 2 reviews the prominent related work. Section 3 provides the background of UMAP, followed by the description of our approach. In Section 4 we present and discuss the results of experiments on diverse datasets. Finally, we draw our conclusions and present future directions.

2   Related Work

Dimensionality reduction (DR) is a powerful tool in data science to look for hidden structure in data and reduce the resources usage of learning algorithms.

The problem of dimensionality has been widely studied [25] and used throughout different domains, such as image processing and face recognition. Dimensionality reduction techniques facilitate the classification task, by removing redundancies and extracting the most relevant features in the data, and permits a better data visualization. A common taxonomy divides these approaches into two major groups: *matrix factorization* and *graph-based* approaches.

Matrix factorization algorithms require matrix computation tools, such as Principal Components Analysis (PCA) [16]. It is a well-known linear technique that uses singular value decomposition and aims to find a lower-dimensional basis by converting the data into features called principal components by computing the eigenvalues and eigenvectors of a covariance matrix. This straightforward technique is computationally cheap but ineffective with data streams since it relies on the whole dataset. Therefore, some incremental versions of PCA have been developed to handle streams of data [13,24,26].

Graph/Neighborhood-based techniques are leveraged in the context of dimension reduction and visualization by using the insight that similar instances in a large space should be represented by close instances in a low-dimensional space, whereas dissimilar instances should be well separated. t-distributed Stochastic Neighbor Embedding (tSNE) [20] is one of the most prominent DR techniques in the literature. It has been proposed to visualize high-dimensional data embedded in a lower space (typically 2 or 3 dimensions). In addition to the fact that it is computationally expensive, tSNE does not preserve distances between all instances and can affect any density–or distance– based algorithm and hence conserves more of the local structure than the global structure.

3   Batch-Incremental Classification

In the following, we assume a data stream S is a *sequence* of instances $X_1, \ldots, X_N$, where N denotes the number of available observations so far. Each instance X_i is composed of a vector of d attributes $X_i = (x_i^1, \ldots, x_i^d)$. The dimensionality reduction of S comprises the process of finding a low-dimensional presentation $S' = Y_1, \ldots, Y_N$, where $Y_i = (y_i^1, \ldots, y_i^p)$ and $p \ll d$.

3.1   Prior Work

Unlike tSNE [20], UMAP has no restriction on the projected space size making it useful not only for visualization but also as a general dimension reduction technique for machine learning algorithms. It starts by constructing open balls over all instances and building simplicial complexes. Dimension reduction is obtained by finding a representation, in a lower space, that closely resembles the topological structure in the original space. Given the new dimension, an equivalent fuzzy topological representation is then constructed [21]. Then, UMAP optimizes it by minimizing the cross-entropy between these two fuzzy topological representations. UMAP offers better visualization quality than tSNE by preserving more of the global structure in a shorter running time. To the best of our knowledge, none

of these techniques has a streaming version. Ultimately, both techniques are
essentially transductive[6] and do not learn a mapping function from the input
space. Hence, they need to process all the data for each new unseen instance,
which prevents them from being usable in data streams classification models.

(a)	(b)	(c)

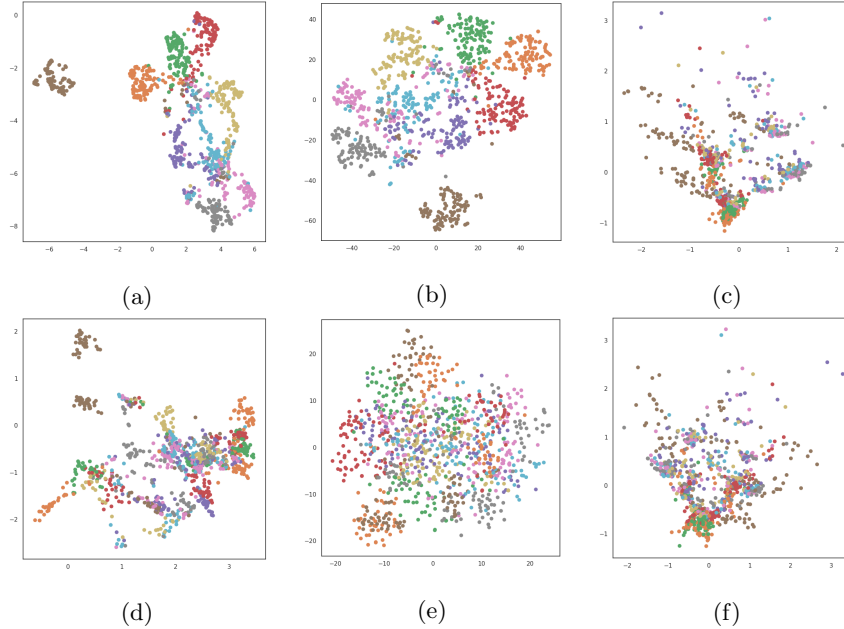|		
(d)	(e)	(f)

Fig. 1: Projection of CNAE dataset in 2-dimensional space. Offline: (a) UMAP,
(b) tSNE, and (c) PCA. Batch-incremental: (d) UMAP, (e) tSNE, and (f) PCA.

Figure 1 shows the projection of CNAE dataset (see Table 1) into 2-dimensions
in an offline/online fashions where each color represents a label. In Figure 1a, we
note that UMAP offers the most interesting visualization while separating classes
(9 classes). The overlap in the new space, for instance with tSNE in Figure 1b,
can potentially affect later classification task, notably distance-based algorithms,
since properties like global distances and density may be lost. On the other hand,
linear transformation, such as PCA, cannot discriminate between instances which
prevents them from being represented in the form of clusters (Figure 1c). To
motivate our choice, we project the same dataset using our batch-incremental
strategy (more details in Section 3.2). Figure 1d illustrates the change from the
offline UMAP representation; this is not as drastic as the ones engendered by
tSNE and PCA (Figures 1e and 1f, respectively) showing their limits on capturing
information from data that arrives in a batch-incremental manner.

---

[6] Transductive learning consists on learning on a full given dataset (including unknown
label), but prediction is only made on the known set of unlabeled instances from the
same dataset. This is achieved by clustering data instances.

3.2   Algorithm Description

A very efficient and simple scheme in supervised learning is *lazy learning* [1]. Since lazy learning approaches are based on distances between every pair of instances, they unfortunately have a low performance in terms of execution time. The k-Nearest Neighbors (kNN) is a well-known lazy algorithm that does not require any work during training, so it uses the entire dataset to predict labels for test instances. However, it is impossible to store an evolving data stream which is potentially infinite – nor to scan it multiple times – due to its tremendous volume. To tackle this challenge, a basic incremental version of kNN has been proposed which uses a fixed-length window that slides through the stream and merges new arriving instances with the closest ones already in the window [23].

To predict the class label for an incoming instance, we take the majority class labels of its nearest neighbors inside the window using a defined distance metric (Equation 2). Since we keep the recent arrived instances inside the sliding window for prediction, the search for the nearest neighbors is still costly in terms of memory and time [3,7] and high-dimensional streams require further resources.

Given a window w, the distance between X_i and X_j is defined as follows:

$$D_{X_j}(X_i) = \sqrt{\|X_i - X_j\|^2}. \tag{1}$$

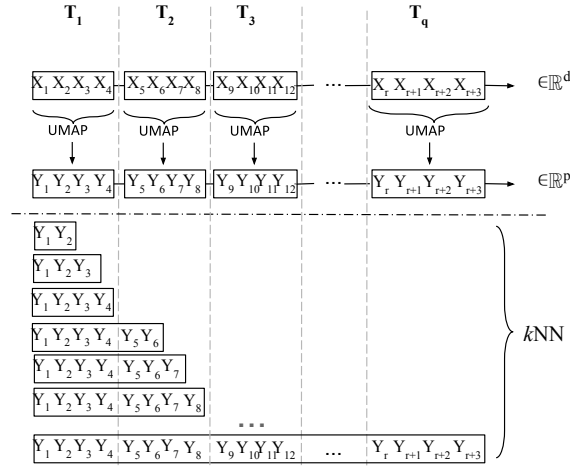Similarly, the k-Nearest Neighbors distance is defined as follows:

$$D_{w,k}(X_i) = \min_{\binom{w}{k}, X_j \in w} \sum_{j=1}^{k} D_{X_j}(X_i), \tag{2}$$

where $\binom{w}{k}$ denotes the subset of the kNN to X_i in w.

When dealing with high-dimensional data, a pre-processing phase before applying a learning algorithm is imperative to avoid the curse of dimensionality from a *computational* point of view. The latter may increase the resources usage and decrease the performance of some algorithms, such as kNN. The main idea to mitigate this curse consists of using an efficient strategy with consistent and promising results such as UMAP.

Since UMAP is a transductive technique, an instance-incremental learning approach that includes UMAP does not work because the entire stream needs to be processed for each new incoming instance. By doing it this way, the process will be costly and will not respond to the streaming requirements. To alleviate the processing cost considering the framework within which several challenges shall be respected, including the memory constraint and the incremental behavior of data, we adopt a batch-incremental strategy. In the following, we introduce the procedure of our novel approach, batch-incremental UMAP-kNN.

Step 1: *Partition of the stream.* During this step, we assume that data arrive in batches – or chunks – by dividing the stream into disjoint partitions $S_1, S_2, \ldots$ of size s. The first part of Figure 2 shows a stream of instances divided into batches, so instead of having instances available one at a time, they will arrive as a group of instances simultaneously, $S_1, \ldots, S_q$, where S_q is the qth chunk. A

T_1 | T_2 | T_3 | T_q

$X_1 X_2 X_3 X_4$ — $X_5 X_6 X_7 X_8$ — $X_9 X_{10} X_{11} X_{12}$ — $\cdots$ — $X_r X_{r+1} X_{r+2} X_{r+3}$ $\in \mathbb{R}^d$

UMAP   UMAP   UMAP   UMAP

$Y_1 Y_2 Y_3 Y_4$ — $Y_5 Y_6 Y_7 Y_8$ — $Y_9 Y_{10} Y_{11} Y_{12}$ — $\cdots$ — $Y_r Y_{r+1} Y_{r+2} Y_{r+3}$ $\in \mathbb{R}^p$

$Y_1 Y_2$

$Y_1 Y_2 Y_3$

$Y_1 Y_2 Y_3 Y_4$

$Y_1 Y_2 Y_3 Y_4$ $Y_5 Y_6$

$Y_1 Y_2 Y_3 Y_4$ $Y_5 Y_6 Y_7$

$Y_1 Y_2 Y_3 Y_4$ $Y_5 Y_6 Y_7 Y_8$

$\cdots$

$Y_1 Y_2 Y_3 Y_4$ $Y_5 Y_6 Y_7 Y_8$ $Y_9 Y_{10} Y_{11} Y_{12}$ $\cdots$ $Y_r Y_{r+1} Y_{r+2} Y_{r+3}$

kNN

Fig. 2: Batch-incremental UMAP-kNN scheme

simple example of data stream is a video sequence where at each instant we have a succession of images.

Step 2: *Data pre-processing.* We aim to construct a low-dimensional $Y_i \in p$, from an infinite stream of high-dimensional data $X_i \in d$, where $p \ll d$. As mentioned before, UMAP is unable to compress data incrementally and needs to transform more than one observation at a time because it builds a neighborhood-graph on a set of instances and then lays it out in a lower dimensional space [21]. Thus, our proposed approach operates on batches of the stream where a single batch S_i of data is processed at a time T_i. The two first steps in Figure 2 depict the application of UMAP on the disjoint batches. Once a batch is complete, throughout the second step, we apply UMAP on it independently from the chunks that have been already processed, so each $S_i \in \mathbb{R}^d$ will be transformed and represented by $S_i{'} \in \mathbb{R}^p$. This new representation is very likely devoid of redundancies, irrelevant attributes, and is obtained by finding potentially useful non-linear combinations of existing attributes, i.e. by repacking relevant information of the larger feature space and encoding it more compactly.

For UMAP to learn when moving from a batch to another, we seed each chunk's embedding with the outcome of the previous one, i.e., match the prior initial coordinates for instances in the current embedding to the final coordinates in the preceding one. This will help to avoid losing the topological information of the stream and to keep stability in successive embeddings as we transition from one batch to its successor. Afterwards, we use the compressed representation of the high-dimensional chunk for the next step that consists in supporting the incremental kNN classification algorithm.

Step 3: *kNN classification.* UMAP-kNN aims to decrease the computational costs of kNN on high-dimensional data stream by reducing the input space size using the dimension reducing UMAP, in a batch-incremental way. In addition to

the prediction phase of the kNN algorithm that, based on the neighborhood[7], UMAP operates on a k-nearest graph (topological representation) as well and optimizes the low-dimensional representation of the data using gradient descent. One nice takeaway is that UMAP, because of its solid theoretical backing as a manifold technique, keeps properties such as density and pairwise distances. Thus, it does not bias the neighborhood-based kNN performance.

This step consists of classifying the evolving data stream, where the learning task occurs on consecutive batches, i.e. we train incrementally kNN with instances becoming successively available in chunk buffers after pre-processing. Figure 2 shows the underlying batch-incremental learning scheme used which is built upon the divide-and-conquer strategy. Since UMAP is independently applied to batches, so once a chunk is complete and has been transformed in $\mathbb{R}^p$, we feed the half of the batch to the sliding window and we predict incrementally the class label for the second half (the rest of instances).

Given that kNN is adaptive, the main novelty of UMAP-kNN is in how it merges the current batch to previous ones. This is done by adding it to the instances from previous chunks inside the kNN window. Even if past chunks have been discarded, only some of them have been stored and maintained while the adaptive window scrolls. Thereafter, instances kept temporarily inside the window are going to be used to define the neighborhood and predict the class labels for later incoming instances. As presented in Figure 2, the intuitive idea to combine results from different batches is to use the half of each batch for training and the second half for prediction. In general, due to the possibility of having often very different successive embeddings, one would expect that this may affect the global performance of our approach. Thus, we adopt this scheme to maintain a stability over an adaptive batch-incremental manifold classification approach.

4   Experiments

In this section, we present a series of experiments carried out on various datasets based on three main results: the accuracy, the memory (MB), and the time (Sec).

4.1   Datasets

We use 3 synthetic and 6 real-world datasets from different scenarios that have been thoroughly used in the literature to evaluate the performance of data streams classifiers. Table 1 presents a short description of each dataset, and further details are provided in what follows.

Tweets. The dataset was created using the tweets text data generator provided by MOA [6] that simulates sentiment analysis on tweets, where messages can be classified depending on whether they convey positive or negative feelings. Tweets$_{1,2,3}$ produce instances of 500, 1,000 and 1,500 attributes respectively.

---

[7] The distances between the new incoming instance and the instances already available inside the adaptive window are computed in order to assign it to a particular class.

Har. Human Activity Recognition dataset [2] built from several subjects performing daily living activities, such as walking, sitting, standing and laying, while wearing a waist-mounted smartphone equipped with sensors. The sensor signals were pre-processed using noise filters and attributes were normalized.

CNAE. CNAE is the national classification of economic activities dataset [9]. Instances represent descriptions of Brazilian companies categorized into 9 classes. The original texts were pre-processed to obtain the current highly sparse data.

Enron. The Enron corpus dataset is a large set of email messages that was made public during the legal investigation concerning the Enron corporation [17]. This cleaned version of Enron consists of $1,702$ instances and $1,000$ attributes.

Table 1: Overview of the datasets

Dataset	#Instances	#Attributes	#Classes	Type
Tweets$_1$	1,000,000	500	2	Synthetic
Tweets$_2$	1,000,000	1,000	2	Synthetic
Tweets$_3$	1,000,000	1,500	2	Synthetic
Har	10,299	561	6	Real
CNAE	1,080	856	9	Real
Enron	1,702	1,000	2	Real
IMDB	120,919	1,001	2	Real
Nomao	34,465	119	2	Real
Covt	581,012	54	7	Real

IMDB. IMDB movie reviews dataset was proposed for sentiment analysis [19], where each review is encoded as a sequence of word indexes (integers).

Nomao. Nomao dataset [8] was provided by Nomao Labs where data come from several sources on the web about places (name, address, localization, etc).

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

4.2 Results and Discussions

We compare our proposed classifier, UMAP-kNN, to various commonly-used baseline methods in dimensionality reduction and machine learning areas. PCA [24], tSNE (fixing the perplexity to 30, which is the best value as reported in [20]), SAM-kNN (SkNN) [18]. We use HAT, a classifier with a different structure based on trees [4], to assess its performance with the neighborhood-based UMAP. For fair comparison, we compare the performance of UMAP-kNN approach with a competitor using UMAP as well in the same batch-incremental manner. Actually, incremental kNN has two crucial parameters: (i) the number of neighbors k fixed to 5; and (ii) the window size w, that maintains the low-dimensional data, fixed to 1000. According to previous studies such as [7], a bigger window will increase the resources usage and smaller size will impact the accuracy.

The experiments were conducted on a machine equipped with an Intel Core i5 CPU and 4 GB of RAM. All experiments were implemented and evaluated in Python by extending the Scikit-multiflow framework[8] [22].

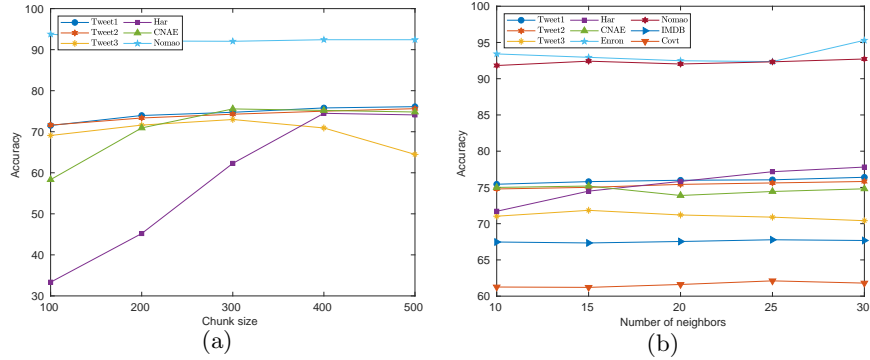
[8] https://scikit-multiflow.github.io/

Fig. 3: (a) Varying the chunk size. (b) Varying the neighborhood size for UMAP.

Figure 3a depicts the influence of the chunk size on the accuracy using UMAP-kNN with some datasets. Generally, fixing the chunk size imposes the following dilemma: choosing a small size so that we obtain an accurate reflection of the current data or choosing a large size that may increase the accuracy since more data are available. The ideal would be to use a batch with the maximum of instances to represent as possible the whole stream. In practice, the chunk size needs to be small enough to fit in the main memory otherwise the running time of the approach will increase. Since UMAP is relatively slow, we choose small chunk sizes to overcome this issue with UMAP-kNN. Based on the obtained results, we fix the chunk size to 400 for the best trade-off accuracy-memory.

We investigate the behavior of a crucial parameter that controls UMAP, number of neighbors, via the classification performance of our approach. Based on the size of the neighborhood, UMAP constructs the manifold and focuses on preserving local and global structures. Figure 3b shows the accuracy when the number of neighbors is varied on diverse datasets. We notice that for all datasets, the accuracy is consistently the same with no large differences, e.g. Har. Since a large neighborhood leads to a slower learning process, in the following we fix the neighborhood size to 15.
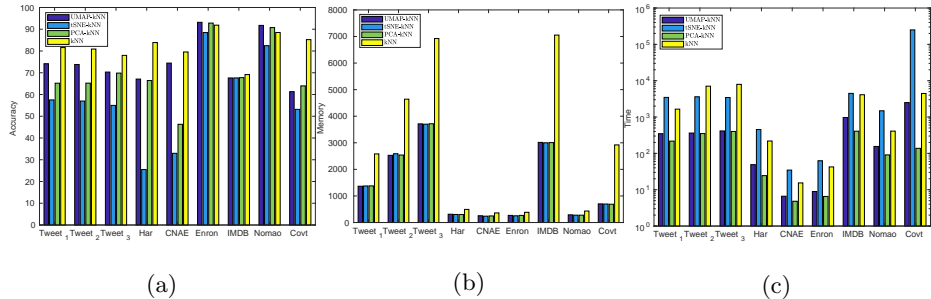

Fig. 4: Comparison of UMAP-kNN, tSNE-kNN, PCA-kNN, and kNN (with the entire datasets) while projecting into 3-dimensions: (a) Accuracy. (b) Memory.

tSNE is a visualization technique, so we are limited to project high-dimensional data into 2 or 3 dimensions. In order to evaluate the performance of our proposal in a fair comparison against each of tSNE-kNN and PCA-kNN, we project data into 3-dimensional space. We illustrate in Figure 4a that UMAP-kNN makes significantly more accurate predictions beating consistently the best performing baselines (tSNE-kNN and PCA-kNN) notably with CNAE and the tweets datasets. Figure 4b depicts the quantity of memory needed by the three algorithms which is practically the same for some datasets. Compared to kNN that uses the whole data without projection, we notice that UMAP-kNN consumes much less memory whilst sacrificing a bit in accuracy because we are removing many attributes. Figure 4c shows that our approach is consistently faster than tSNE-kNN because tSNE computes the distances between every pair of instances to project. But PCA-kNN is a bit faster thanks to the simplicity of PCA. But with this trade-off our approach performs good on almost all datasets.

Table 2: Comparison of UMAP-kNN, PCA-kNN, UMAP-SkNN, and UMAP-HAT.

	Accuracy (%)			
Dataset	UMAP-kNN	PCA-kNN	UMAP-SkNN	UMAP-HAT
Tweets$_1$	75.71	69.89	75.37	66.47
Tweets$_2$	75.16	69.21	74.40	61.27
Tweets$_3$	71.01	70.81	70.47	66.98
Har	75.30	70.50	64.09	84.89
CNAE	76.67	67.41	75.18	40.18
Enron	92.24	93.41	91.89	91.77
IMDB	67.38	67.28	67.43	64.52
Nomao	91.92	91.13	91.63	83.75
Covt	61.29	66.73	53.08	55.43
	Memory (MB)			
Dataset	UMAP-kNN	PCA-kNN	UMAP-SkNN	UMAP-HAT
Tweets$_1$	1366.71	1354.24	1373.15	2738.32
Tweets$_2$	2530.30	2518.76	2532.95	4891.23
Tweets$_3$	3706.99	3706.55	3722.68	7144.77
Har	311.58	310.48	312.84	381.49
CNAE	254.17	246.94	260.29	262.52
Enron	269.00	267.31	271.56	288.74
IMDB	3012.85	3013.28	3018.04	7471.64
Nomao	289.81	285.50	290.60	508.50
Covt	700.69	689.97	704.46	3788.54
	Time (Sec)			
Dataset	UMAP-kNN	PCA-kNN	UMAP-SkNN	UMAP-HAT
Tweets$_1$	558.56	217.44	1396.32	2163.14
Tweets$_2$	616.50	350.63	908.59	3453.21
Tweets$_3$	667.43	400.62	1066.98	6273.19
Har	75.20	24.37	77.99	82.47
CNAE	8.89	4.81	13.17	19.78
Enron	12.80	9.52	17.26	32.84
IMDB	715.68	407.60	1038.77	4691.07
Nomao	248.79	20.46	327.36	228.00
Covt	2311.21	137.62	3756.41	2297.01

In addition to its good classification performance in comparison with competitors, the batch-incremental UMAP-kNN did a better job of preserving density by capturing both of global and local structures, as shown in Figure 1d. The fact that UMAP and kNN are both neighborhood-based methods arises as a key element in achieving a good accuracy. UMAP has not only the power of

visualization but also the ability to reduce the dimensionality of data efficiently which makes it useful as pre-processing technique for machine learning.

Table 2 reports the comparison of UMAP-kNN against state-of-the-art classifiers. We highlight that our approach performs better on almost all datasets. It achieves similar accuracies to UMAP-SkNN on several datasets but in terms of resources, the latter is slower because of its drift detection mechanism. UMAP-kNN has a better performance than PCA-kNN, e.g. the Tweets datasets at the cost of being slower. We also observe the UMAP-HAT failed to overcome our approach (in terms of accuracy, memory, and time) due to the integration of a neighborhood-based technique (UMAP) to a tree structure (HAT).

Figure 5 reports detailed results for Tweet$_1$ dataset with five output dimensions. Figure 5a exhibits the accuracy of our approach which is consistently above competitors whilst ensuring stability for different manifolds. Figures 5b and 5c show that kNN-based classifiers use much less resources than the tree-based UMAP-HAT. We see that UMAP-kNN requires less time than UMAP-HAT and UMAP-SkNN to execute the stream but PCA-kNN is fastest thanks to its simplicity. Still, the gain in accuracy with UMAP-kNN is more significant.
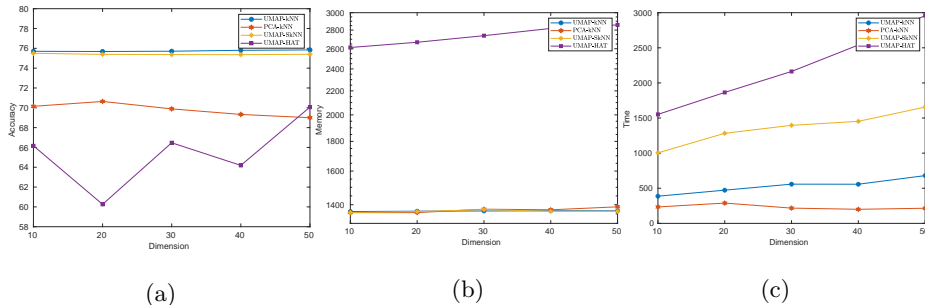

| (a) | (b) | (c) |

Fig. 5: Comparison of UMAP-kNN, PCA-kNN, UMAP-SkNN, and UMAP-HAT over different output dimensions on Tweet$_1$: (a) Accuracy. (b) Memory. (c) Time.

5 Concluding Remarks and Future Work

In this paper, we presented a novel batch-incremental approach for mining data streams using the kNN algorithm. UMAP-kNN combines the simplicity of kNN and the high performance of UMAP which is used as an internal pre-processing step to reduce the feature space of data streams. We showed that UMAP is capable of embedding efficiently data streams within a batch-incremental strategy in an extensive evaluation with well-known state-of-the-art algorithms using various datasets. We further demonstrated that the batch-incremental approach is just as effective as the offline approach in visualization and its accuracy outperforms reputed baselines while using reasonable resources.

We would like to pursue our promising approach further to enhance its runtime performance by applying a fast dimension reduction before using of UMAP. Another area for future work could be the use of a different mechanism, such as the application of UMAP for each incoming data inside a sliding window. We believe that this may be slow but will be suited for instance-incremental learning.

References

1. Aha, D.W.: Lazy learning. Springer (2013)
2. Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.L.: Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In: IWAAL. pp. 216–223. Springer (2012)
3. Bahri, M., Maniu, S., Bifet, A.: A sketch-based naive bayes algorithms for evolving data streams. In: International Conference on Big Data. pp. 604–613. IEEE (2018)
4. Bifet, A., Gavaldà, R.: Adaptive learning from evolving data streams. In: International Symposium on Intelligent Data Analysis. pp. 249–260. Springer (2009)
5. Bifet, A., Gavaldà, R., Holmes, G., Pfahringer, B.: Machine Learning for Data Streams: with Practical Examples in MOA. MIT Press (2018)
6. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: Moa: Massive online analysis. JMLR **11**(May), 1601–1604 (2010)
7. Bifet, A., Pfahringer, B., Read, J., Holmes, G.: Efficient data stream classification via probabilistic adaptive windows. In: SIGAPP. pp. 801–806. ACM (2013)
8. Candillier, L., Lemaire, V.: Design and analysis of the nomao challenge active learning in the real-world. In: ALRA, Workshop ECML-PKDD. sn (2012)
9. Ciarelli, P.M., Oliveira, E.: Agglomeration and elimination of terms for dimensionality reduction. In: ISDA. pp. 547–552. IEEE (2009)
10. Cortes, C., Vapnik, V.: Support-vector networks. ML **20**(3), 273–297 (1995)
11. Domingos, P., Hulten, G.: Mining high-speed data streams. In: SIGKDD. pp. 71–80. ACM (2000)
12. Gama, J., Sebastião, R., Rodrigues, P.P.: Issues in evaluation of stream learning algorithms. In: SIGKDD. pp. 329–338. ACM (2009)
13. Günter, S., Schraudolph, N.N., Vishwanathan, S.: Fast iterative kernel principal component analysis. JMLR **8**(8), 1893–1918 (2007)
14. Hand, D.J., Mannila, H., Smyth, P.: Principles of data mining. MIT press (2001)
15. Holmes, G., Kirkby, R.B., Bainbridge, D.: Batch-incremental learning for mining data streams (2004)
16. Hotelling, H.: Analysis of a complex of statistical variables into principal components. Journal of Educational Psychology **24**(6), 417 (1933)
17. Klimt, B., Yang, Y.: The enron corpus: A new dataset for email classification research. In: ECML. pp. 217–226. Springer (2004)
18. Losing, V., Hammer, B., Wersing, H.: Knn classifier with self adjusting memory for heterogeneous concept drift. In: ICDM. pp. 291–300. IEEE (2016)
19. Maas, A.L., Daly, R.E., Pham, P.T., Huang, D., Ng, A.Y., Potts, C.: Learning word vectors for sentiment analysis. In: ACL-HLT. pp. 142–150. Association for Computational Linguistics (2011)
20. Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. JMLR **9**, 2579–2605 (2008)
21. McInnes, L., Healy, J., Melville, J.: Umap: Uniform manifold approximation and projection for dimension reduction. arXiv preprint arXiv:1802.03426 (2018)
22. Montiel, J., Read, J., Bifet, A., Abdessalem, T.: Scikit-multiflow: a multi-output streaming framework. JMLR **19**(1), 2915–2914 (2018)
23. Read, J., Bifet, A., Pfahringer, B., Holmes, G.: Batch-incremental versus instance-incremental learning in dynamic and evolving data. In: IDA. pp. 313–323 (2012)
24. Ross, D.A., Lim, J., Lin, R.S., Yang, M.H.: Incremental learning for robust visual tracking. IJCV **77**(1-3), 125–141 (2008)
25. Sorzano, C.O.S., Vargas, J., Montano, A.P.: A survey of dimensionality reduction techniques. arXiv preprint arXiv:1403.2877 (2014)
26. Weng, J., Zhang, Y., Hwang, W.S.: Candid covariance-free incremental principal component analysis. TPAMI **25**(8), 1034–1040 (2003)