Compressed k-Nearest Neighbors Ensembles for Evolving Data Streams

Maroua Bahri, Silviu Maniu, Albert Bifet, Rodrigo Fernandes de Mello, Nikolaos Tziortziotis

Compressed k-Nearest Neighbors Ensembles for Evolving Data Streams

Maroua Bahri[1] and **Albert Bifet**[1 2] and **Silviu Maniu**[3 4 5] and **Rodrigo F. de Mello**[6] and **Nikolaos Tziortziotis**[7]

Abstract. The unbounded and multidimensional nature, the evolution of data distributions with time, and the requirement of single-pass algorithms comprise the main challenges of data stream classification, which makes it impossible to infer learning models in the same manner as for batch scenarios. Data dimensionality reduction arises as a key factor to transform and select only the most relevant features from those streams in order to reduce algorithm space and time demands. In that context, Compressed Sensing (CS) encodes an input signal into lower-dimensional space, guaranteeing its reconstruction up to some distortion factor ϵ. This paper employs CS on data streams as a pre-processing step to support a k-Nearest Neighbors (kNN) classification algorithm, one of the most often used algorithms in the data stream mining area – all this while ensuring the key properties of CS hold. Based on topological properties, we show that our classification algorithm also preserves the neighborhood (withing an ϵ factor) of kNN after reducing the stream dimensionality with CS. As a consequence, end-users can set an acceptable error margin while performing such projections for kNN. For further improvements, we incorporate this method into an ensemble classifier, Leveraging Bagging, by combining a set of different CS matrices which increases the diversity inside the ensemble. An extensive set of experiments is performed on various datasets, and the results were compared against those yielded by current state-of-the-art approaches, confirming the good performance of our approaches.

1 Introduction

Data streams are unbounded sequences of multidimensional observations made available along time, hence, it is impossible to maintain and process them using the main memory. In practice, mining tasks reduce time and space requirements while only processing relevant features out of those redundant streams, what corresponds to data summaries generally obtained as follows: (i) either by selecting only a subsample (*sampling*) of the input data along time; or (ii) by reducing/selecting data attributes via dimensionality reduction techniques, what turns out to work along features. Naturally, the choice of the technique depends on the problem being solved [13]. The dimensionality reduction process inherently arises when one deals with a large number of attributes, especially when data are sparse. In this scenario, this is achieved by selecting only the most relevant features, or by transforming them into a smaller set.

A common taxonomy organizes those approaches as follows: (i) *Feature selection*, which consists in selecting a subset of the input features, i.e., the most relevant, non-redundant, without operating any sort of data transformation [5, 27]; and (ii) *Feature extraction*, which consists in *transforming* the input attributes into a new set of features in some lower dimensional space [26]. The main drawback of feature selection is that the simple feature removal could lead to *data loss*, when one inadvertently performs such selection that is even more challenging when dealing with potentially infinite and high-dimensional data streams that boost the use of computational resources, most precisely the memory and the processing time. In order to address this limitation, we apply *Compressed Sensing* (CS) (also referred to as Compressed Sampling) which is a feature extraction strategy that provides theoretical lower and upper bounds on pairwise data transformations. This data reduction is highly relevant in the context of data streams mining since it helps to diminish resource demands while ensuring the quality of learning (e.g. *classification accuracy*), addressing the evolving data streams framework requirements and avoiding more than a single data pass. This last issue would lead an algorithm either to lose information from next observations, given some sub-sampling along time, or to have an exponential amount of main memory available to save the stream, thus making sure that all observations were processed properly.

CS is a technique that has attracted a lot of attention and is based on the concept that a data compression method has to deal with redundancy while transforming and reconstructing data [15]. The basic idea is to use orthogonal features, i.e. complementary features, to provably and properly represent data as well as reconstruct them from small number of samples. CS has been widely studied and used throughout different domains in the offline framework, such as image processing [31], face recognition [25], and vehicle classification [35].

Hence, we aim to find the best trade-off over three aspects: (i) the classifier accuracy: the proportion of correctly predicted instances; (ii) the memory usage: the cost of keeping the transformed data in main memory; and (iii) the overall processing time: comprising the data transformation, learning, and classification. All such aspects are strongly related, so the drastic reduction of time and space complexities would make our algorithm much faster than using all features. Of course, one should weigh the accuracy while assessing these factors.

The main contributions of this paper are the following:

- *Compressed-k-Nearest Neighbors (CS-kNN)*: a new kNN algorithm to support data stream classification. Our main focus consists in improving its resource usage by compressing input streams using CS while theoretically guaranteeing close approximation to the accuracy that would be obtained using the original stream;

[1] Télécom Paris, IP-Paris, Paris, France, email: {maroua.bahri, albert.bifet}@telecom-paris.fr
[2] University of Waikato, Hamilton, New Zealand
[3] Université Paris-Saclay, LRI, CNRS, Orsay, France, email: silviu.maniu@lri.fr
[4] Inria, Paris, France
[5] ENS DI, CNRS, École Normale Supérieure, Université PSL, Paris, France
[6] Universidade de São Paulo, Brazil, email: mello@icmc.usp.br
[7] Tradelab, France, email: ntziorzi@gmail.com

- *Bagging with Compressed Sensing (CSB)*: an ensemble technique based on Leveraging Bagging [8] where we combine several CS-kNN instances built upon different CS independent matrices;
- *Empirical results*: we present experiments to show the abilities of our proposals in obtaining a good trade-off among the three axes (accuracy, memory and time) against several popular baselines;

The remainder of this work is organized as follows. We present the related work for dimensionality reduction and streaming classification algorithms in Section 2.2. Section 3 provides the basics of compressed sensing, followed by its application in conjunction with the kNN algorithm for evolving data streams. Section 4 discusses the experimental results performed on both synthetic and real datasets followed by the concluding remarks in Section 5.

2 Preliminaries and Related work

2.1 Notation

In the following, we assume a data stream X is a *sequence* of instances $x_1, x_2, \ldots, x_N$, where N denotes the number of available observations so far. Each instance x_i is composed of a vector of d attributes $x_i^1, \ldots, x_i^d$. The dimensionality reduction comprises the process of finding some transformation function (or *map*) $A : \mathbb{R}^d \to \mathbb{R}^p$, where $p \ll d$, to be applied on each instance x_i of X.

2.2 Related Work

There are two main different types of technique for addressing the dimensionality problem: (i) based on random projections, a.k.a. *data-independent*, that are not derived from data; and (ii) *data-dependent* strategies derived from data to achieve the transformation itself. Among data-dependent techniques, we mention feature extraction based on component analysis where several variations have been proposed to handle evolving distributions such as IPCA [38] and IKPCA [19]. The most popular and straightforward technique is the Principal Component Analysis (PCA) [20] which aims to find a lower-dimensional basis in which the sum of square distances between the original data and their projections is minimized, i.e. being as close as possible to zero while maximizing the variances.

Random Projection (RP) is a cost-efficient alternative to PCA since it is data-independent and satisfies the Johnson-Lindenstrauss (JL) lemma [21]: Let $\epsilon \in [0, 1]$, $X = \{x_1, ..., x_N\} \in \mathbb{R}^d$. Given a number $p \geq \log(N/\epsilon^2)$, $\forall x_i, x_j \in X$ there is a linear map $A : \mathbb{R}^d \to \mathbb{R}^p$ such that:

$$(1 - \epsilon)\|x_i - x_j\|_2^2 \leq \|Ax_i - Ax_j\|_2^2 \leq (1 + \epsilon)\|x_i - x_j\|_2^2. \quad (1)$$

The JL lemma (1) asserts that N instances from an Euclidean space can be projected into a lower dimensional space of $\mathcal{O}(\log N/\epsilon^2)$ dimensions under which pairwise distances are preserved within a multiplicative factor $1 \pm \epsilon$.

As detailed in the following section, random projection matrices have been used before in conjunction with compressed sensing [37]. This approach applies a random linear transformation on vectors, changing their original space and leading to significant results, outperforming PCA. For example, given data in a 10^4 dimensional space, two random projections will give a perfect recovery while two PCA projections will only recover data with a probability equals to $2/10^4$. In short, RP achieves good performance with few projections whereas the PCA performance increases linearly with the number of output dimension which makes it slower [37].

Another well-known technique is the Hashing Trick (HT) [36], also known as feature hashing. It has been used to make the analysis of sparse and large data tractable in practice by mapping sparse instances or vectors into a lower feature space using a hash function. Given a list of keys that represents a set of features from the input instances, it calculates then the hash function for each key, which will ensure its mapping to a specific cell of a fixed size vector that constitutes the new compressed instance.

Those techniques could be combined to several machine learning algorithms while dealing with evolving data streams. Besides, an important issue to address in the data stream scenario is the computational efficiency of classifiers because of the potentially infinite nature of evolving data streams. Quite a number of classification algorithms for static datasets, that have already been thoroughly studied, proved to be of limited effectiveness when dealing with big data streams. Therefore, some of them have been extended to handle evolving data streams [6, 3], among others, Self-Adjusting Memory kNN (sam$k$NN) [28] which uses a dual-memory model to capture the drift in data streams. Hoeffding tree [14], also known as Very Fast Decision Tree (VFDT), which is an incremental, anytime decision tree induction algorithm that uses the Hoeffding bound to select the optimal splitting attributes. On the other hand, there exists ensemble learners which are popular when learning from evolving data streams because they achieve a high learning performance, such as Leveraging Bagging (LB) [8] and Adaptive Random Forest (ARF) [18]. However, their major drawback is the high computational demand.

3 Compressed Sensing Classification

3.1 Basic Notions of Compressed Sensing

The goal of compressed sensing, given a sparse vector $x \in \mathbb{R}^d$, is to measure $y \in \mathbb{R}^p$ and then reconstruct x, for $p \ll d$, as follows:

$$y = Ax, \quad (2)$$

where $A \in \mathbb{R}^{p \times d}$ is called a *measurement* (*sampling* or *sensing*) matrix. A is used to transform instances from high-dimensional space (x vectors) to a lower dimensional space (y vectors). Three concepts are crucial to the recovery of the stream with high probability [15]:

Sparsity: CS exploits the fact that data may be sparse and hence compressible in a concise representation. For an instance X with support $\text{supp}(X) = \{l : x^l \neq 0\}$, we define the ℓ_0-norm $\|X\|_0 = |supp(X)|$, so X is s-sparse if $\|X\|_0 \leq s$. The implication of sparsity is important to remove irrelevant data without much loss.

Restricted Isometry Property (RIP): A is said to respect RIP if there exists $\epsilon \in [0, 1]$ such that:

$$(1 - \epsilon)\|x\|_2^2 \leq \|Ax\|_2^2 \leq (1 + \epsilon)\|x\|_2^2. \quad (3)$$

This property holds for all s-sparse data $x \in \mathbb{R}^d$.

CS relies on the aforementioned principles that provide, with high probability, a good data reconstruction from a limited number of incoherent and possibly noisy measurements. Mathematically, the decompression of the data that obeys the linear relation in Equation (2) consists in approximating the error by ℓ_1-norm minimization that provides a convex relaxation and when data are sparse, the recovery via ℓ_1-minimization is provably exact [34]:

$$\arg\min_{x \in \mathbb{R}^d} \|x\|_1 \quad \text{s.t.} \quad y = Ax.$$

The goal is to find an efficient representation for each instance such that the sum of their reconstruction errors is minimized. The

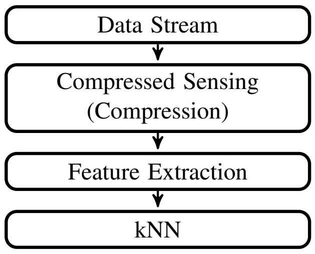

Figure 1: Compressed kNN Scheme.

RIP guarantees the proper computation of the above-mentioned recovery problem [10].

3.2 Construction of Sampling (Sensing) Matrices

The RIP is both a necessary and sufficient condition for an efficient data recovery. Randomization is a key ingredient in the construction of most of RIP matrices used in the CS transformation process [11]. In what follows, we cite examples of CS matrices:

- *Fourier matrix* is obtained by applying Fourier transform on data and thereafter selecting uniformly at random p rows from a d dimensional Fourier matrix;
- *Random Gaussian matrix* is generated randomly from a Gaussian distribution having independent and identically distributed (i.i.d) entries with zero mean and variance one: $A_{i,j} \sim \mathcal{N}(0, 1)$;
- *Random Bernoulli matrix* has entries which are randomly sampled from a Bernoulli distribution with equal probability: $A_{i,j} \in \{1/\sqrt{p}, -1/\sqrt{p}\}$.

For the data-independent random matrices, it has been proved that any matrix A satisfying the JL lemma (1) will also satisfy the RIP in CS with high probability if $p = \mathcal{O}(s \log(d))$ [1]. A comparison of the results obtained with these matrices and an explanation of the choice of the matrix used in this work are provided in the following.

3.3 Compressed Classification Using kNN

The k-Nearest Neighbors (kNN) is one of the most often used algorithms that has been adapted to the stream setting [32]. It does not require any work during training but its offline version uses the entire dataset to predict class labels for test instances. The challenge with adapting kNN to evolving data streams lies in the impracticality of storing the entire stream for prediction. To tackle this issue, an adaptive solution fits new instances once they arrive into a fixed-length window and merge them with the closest ones already in memory. The prediction of the class label for an instance is therefore made by taking the majority vote of its nearest neighbors, using a defined distance metric. Yet, the search for the nearest neighbors is still costly in terms of time and memory [9, 3]. Thus, for high-dimensional data, a dimension reduction is imperative to avoid the curse of dimensionality which may increase the use of computational resources. The main idea to mitigate this drawback and improve kNN's performance is to use a simple strategy with relevant properties such as CS.

We focus on the analysis of an infinite stream of instances $x_i \in \mathbb{R}^d$ from which we wish to construct a low dimensional space $\mathbb{R}^p$, where $p \ll d$. We assume that instances are s-sparse in some basis, so we can use CS with a RIP matrix and work in a lower dimension of $\mathcal{O}(s \log(d))$. It is important to perform such reduction because it is related to the number of dimensions and independent of the stream

Algorithm 1 Compressed-kNN algorithm. **Symbols:** $X = \{x_1, x_2, \ldots\}$: stream; $C = \{c_1, c_2, \ldots\}$: set of labels; w: window; k: the neighborhood size; p: the target dimension; S: subset of w.

```
1: function CS-$k$NN($X, w, k, p$)
2:     Init $w \leftarrow \emptyset$
3:     for all $x_i \in X$ do
4:         $y_i \leftarrow$ CS($x_i$)                        ▷ apply CS
5:         for all $y_j \in w$ do                         ▷ $\forall\, j \neq i$
6:             compute $D_{y_j}(y_i)$                     ▷ Equation (5)
7:         $c \leftarrow \max_{c \in C} D_{S,k}(y_i)$          ▷ Equation (6)
8:         $w \leftarrow y_i$        ▷ maintain the compressed $x_i$ in $w$
```

size, making it useful in applications for data streams where the size is unknown. This transformation can lead to information loss, except if the sensing matrix respects the RIP, then with high probability, the information loss is minimal, and the original signal can be recovered.

Figure 1 presents the main flow of the proposed approach combining the simplicity of kNN and the strong properties of CS to obtain the compressed kNN classifier, called CS-kNN in the following. Fundamentally, CS is composed of two phases: (i) the *compression* phase, where the data are projected onto a low-dimensional space; and (ii) the *decompression* phase, where the data are recovered. Nevertheless, the compressed nature of CS makes the paradigm a better fit to classification than the reconstruction. In this paper, we are only concerned with the first stage, so the extracted features from the high-dimensional space are fed to kNN classifier which predicts target class labels. This does not prevent, however, the guarantees over the recovery to hold true. Algorithm 1 shows the pseudo-code of the CS-kNN. We apply CS on each instance x_i of the stream (lines 3-4), then we apply kNN by computing the distance of each y_j in w with y_i (line 6) and thus report the most frequent class label to y_i (line 7). Finally, we feed the compressed version y_i to w (line 8).

In this work, we opt to make kNN more efficient in terms of memory and speed taking into account the online aspect of evolving data streams. Our approach consists of the CS application on high-dimensional data obtained by compressing every new arrived instance via solving Equation (2). In order that our approach works well, we need to use an effective sampling matrix that gives sufficiently good (or with minor loss in) accuracy and potentially leads to computational savings. In recent work [2], authors reviewed different sampling matrices performance where the experiment results show that Gaussian random matrices perform nicely.

To motivate our choice in the following, we perform experiments to assess different sampling matrices. For this, we first generate synthetic random Bernoulli and Gaussian matrices, and we also construct the Fourier matrices on some datasets (see the description in Table 2). Thus, for each dataset, we build projections for 5 different settings of the target dimension $\{10, 20, \ldots, 50\}$. Table 1 shows the results for kNN (with $k = 5$) along with the overall average over the different targeting dimensions for each matrix. We notice that with the random Bernoulli matrix, kNN performs worse on average, confirming previous studies [2, 30], compared to the Random Gaussian and Fourier matrices which are very close to the kNN, in terms of accuracy, using the whole data without projections. Nevertheless, Fourier transformation relies on data, i.e., it requires the presence of all instances which is unrealistic in the context of data streams. In [17], authors proposed a recursive scheme for using Fourier matrices with data streams which constructs successive windows and uses the measurement in the previous window to obtain the next one. However, this approach is expensive in terms of memory since it keeps data on windows and it is still not as accurate as using a Gaussian matrix.

Table 1: Accuracy (%) comparison of compressed sensing with Bernoulli, Gaussian, Fourier matrices, and the entire dataset.

Dataset	Bernoulli	Gaussian	Fourier	Whole data
Tweet$_1$	64.78	77.89	77.90	79.80
Tweet$_2$	64.59	77.17	79.53	79.20
Tweet$_3$	60.24	75.59	77.13	78.86
CNAE	27.04	64.59	58.49	73.33
Enron	95.88	95.97	95.91	96.18
Overall ∅	62.51	78.24	77.79	81.58

In this work, we want to use a data-independent matrix to ensure fast processing. Following these experiments, we focus on the Gaussian matrix which not only provides good accuracy but satisfies with high probability the RIP and therefore allows recovery of instances [4]. The matrix A in Equation (2) satisfies the RIP so x can be recovered with minimum error from y, i.e., y preserves the important information that x contains.

First, we need to bound the probability of error related to the estimated instance and its expected value using the Hoeffding inequality. Given $x_i, \forall i \in [1, N]$, where x_i^j are bounded by the interval $[a_j, b_j]$, then for any ϵ, the probability of error is upper bounded as follows:

$$P(|x_i - \hat{x_i}| \geq \epsilon) \leq 2 \exp\left(-\frac{2\epsilon^2}{\sum_{l=1}^{d}(b_l - a_l)^2}\right), \quad (4)$$

where $\hat{x_i}$ is the reconstructed instance and N can simply be the size of a "sliding window" over which the error guarantee is provided.

To make the link between the kNN and the use of RIP matrices, we point out that the JL lemma [21] asserts that a random projection preserves the distance between pairs of instances up to $1 \pm \epsilon$ guarantee with high probability. Similarly, the kNN algorithm is based on a function that measures the distances between instances to predict. Thus, we aim to provide theoretical guarantees on the connection between kNN and stream recovery by showing that the CS transformation using Gaussian matrix preserves the distance function and also approximately maintains the shape –in the neighborhood sense– as the original space, based on the concept of *persistent homology*.

Persistent homology [16] is one of the main tools used to extract information from topological features of a space at different scales for an effective shape description. Given a dataset in some metric space, computing the persistent homology naturally involves nearest neighbors since we are constructing the topological space by building open balls around instances. In this regards, it has been shown in [33] that the persistent homology of a distance such as in the JL lemma (1) is $(1 \pm \epsilon)$-preserved under random projection into $\mathcal{O}(\log N/\epsilon^2)$ dimensions. The basic idea in [33] consists in preserving the radius of the minimum enclosing open ball of data up to a factor of $(1 \pm 4\epsilon)$.

In the following, we deal with the Euclidean distance function in both kNN and data reconstruction guarantees. Given a window w, the distance between instances x_i and x_j is defined as follows:

$$D_{x_j}(x_i) = \sqrt{\|x_i - x_j\|^2}. \quad (5)$$

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

$$D_{w,k}(x_i) = \min_{\binom{w}{k}, x_j \in w} \sum_{j=1}^{k} D_{x_j}(x_i), \quad (6)$$

where $\binom{w}{k}$ denotes the subset of w of size k, i.e., the k-nearest neighbors to the instance x_i in w.

CS random matrices satisfy RIP, so we need to show that our matrix preserves the neighborhood for kNN without significant loss through the JL lemma (1). This would allow us to conserve distances among instances and finally ensure distance-preservation among all neighbors. In [4], authors have indeed established a connection between the expressions in (1) and (3), and proved that the JL lemma implies the RIP for s-sparse data within an ϵ-multiplicative factor. A converse result has been proved in [24] wherein matrices having the RIP respect the JL lemma, i.e., preserve the distances in the transformation between any pairs of instances up to a $(1 \pm \epsilon)$-factor with target dimension in $\mathcal{O}(s \log(d))$.

Application to persistent homology. Now we want to prove, based on the aforementioned result [24] derived from Equation (3), that CS preserves as well the distances between all instances up to $(1 \pm \epsilon)$-error and not only distances between pairs of instances. In other words, we prove that, given a RIP matrix, the resulting compressed instances preserve the kNN neighborhood of the data.

Theorem 1 *Given a set of instances in a sliding window $w = \{x_i\}$, $i \in [1, N]$ and $\epsilon \in [0, 1]$, if there exists a transformation matrix $A : \mathbb{R}^d \to \mathbb{R}^p$ having the RIP, such that $p = \mathcal{O}(s \log(d))$, where s is the sparsity of data, then $\forall x_i \in w$:*

$$(1 - \epsilon) D^2_{w,k}(x) \leq D^2_{w,k}(Ax) \leq (1 + \epsilon) D^2_{w,k}(x). \quad (7)$$

Proof. Assume that $x_1, x_2, \cdots, x_k$ are the k-nearest neighbors to an instance $t \in w$. We have:

$$(1 - \epsilon)\|t - x_i\|^2 \leq \|At - Ax_i\|^2 \leq (1 + \epsilon)\|t - x_i\|^2.$$

By summing these inequalities k times, we obtain:

$$(1 - \epsilon)\sum_{i=1}^{k}\|t - x_i\|^2 \leq \sum_{i=1}^{k}\|At - Ax_i\|^2 \leq (1 + \epsilon)\sum_{i=1}^{k}\|t - x_i\|^2.$$

The distance of At to its k-nearest neighbors in w is minimal, so we have the lower bound as follows:

$$D^2_{w,k}(At) \leq \sum_{i=1}^{k}\|At - Ax_i\|^2.$$

For the upper bound, we have:

$$D^2_{w,k}(At) \leq \sum_{i=1}^{k}\|At - Ax_i\|^2 \leq (1 + \epsilon)\sum_{i=1}^{k}\|t - x_i\|^2,$$

$$D^2_{w,k}(At) \leq \sum_{i=1}^{k}\|At - Ax_i\|^2 \leq (1 + \epsilon) D^2_{w,k}(t).$$

Assume that $Az_1, Az_2, \cdots, Az_k$ are the k-nearest neighbors to At, where $z_1, z_2, \cdots, z_k \in w$. So we have:

$$(1 - \epsilon)\sum_{i=1}^{k}\|y - z_i\| \leq \sum_{i=1}^{k}\|At - Az_i\|^2 = D^2_{w,k}(At).$$

Given the fact that $x_1, x_2, \cdots, x_k$ are the k-nearest neighbors to t, we found the lower bound as follows:

$$D^2_{w,k}(t) = \sum_{i=1}^{k}\|t - x_i\|^2 \leq \sum_{i=1}^{k}\|t - z_i\|^2.$$

This completes the proof. □

So far, we demonstrated that the CS-kNN linked to geometrical properties by achieving homology preservation while being scale-invariant in terms of distances, captures the neighborhood up to some (ϵ)-divergence between the original and the compressed instances.

Bagging CS-kNN. Another application of this framework is to use the CS in an ensemble method which applies CS-kNN as a base learner under Leveraging Bagging (LB) [8], denoted CS-kNN^{LB}.

To increase the diversity inside this LB ensemble, in addition to sampling with the Poisson distribution (λ), where $\lambda \geq 1$, we can use several random matrices by generating a different CS matrix for each ensemble member (CS-kNN) instead of using only one for all the learners (the case of CS-kNN^{LB}). We refer to the aforementioned approach in the following as Compressed Sensing Bagging Ensemble (CSB), (CSB-kNN). The properties assessing the neighborhood preservation for CS-kNN hold also for the CSB-kNN.

4 Experimental Results

We conduct extensive experiments to evaluate the performance of our proposals. We are interested in three main results: the accuracy, the memory (megabytes), and the time (seconds).

4.1 Data

We use 4 synthetic and 5 real-world datasets from different scenarios. Table 2 presents a short description of each dataset, and further details are provided in what follows.

Table 2: 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
RBF	1,000,000	200	10	Synthetic
CNAE	1,080	856	9	Real
Enron	1,702	1,000	2	Real
IMDB	120,919	1,001	2	Real
Spam	9,324	39,916	2	Real
Covt	581,012	54	7	Real

Tweets. Tweets was created using the text data generator provided by MOA [7]. It simulates sentiment analysis on tweets, where messages can be classified into two categories depending on whether they convey positive or negative feelings. Tweets$_1$, Tweets$_2$, and Tweets$_3$ produce instances of 500, $1,000$, and $1,500$ attributes, respectively.

RBF. The Radial Basis Function generator creates centroids at random positions, and each one has a standard deviation, a weight and a class label.

CNAE. CNAE is the national classification of economic activities dataset, initially used in [12]. Instances represent descriptions of Brazilian companies categorized into 9 classes. The original texts were pre-processed to obtain the current highly sparse dataset.

Enron. The Enron corpus is a cleaned version of a large set of emails that was made public during the legal investigation concerning the Enron corporation [23].

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

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 [22]. Each attribute represents the presence of a word in the instance (an e-mail).

Covt. The forest covertype dataset obtained from US Forest Service Region 2 Resource Information System (RIS) data.

4.2 Results

The experiments were conducted on a machine equipped with an Intel Core i5 CPU and 4GB of main memory. They were implemented and evaluated in Java by extending the MOA framework [6, 7]. We used the online evaluation setting for Test-Then-Train method, where each instance is used first for testing and then for training.

Results with non-ensemble methods. We compare the performance of our proposed classifier, CS-kNN, to commonly-used techniques in the literature; self-adjusting memory kNN (CS-sam$k$NN) with CS, kNN using hashing trick (HT-kNN), principal component analysis (PCA-kNN), and the standard kNN without projection as well (using the entire data). The streaming kNN has two principal parameters: the number of neighbors k and the window size w.

Table 3: Performance of kNN with different window sizes.

	Accuracy (%)		
	w=100	w=1000	w=5000
Overall ∅	75.48	80.33	82.44
	Time (sec)		
	w=100	w=1000	w=5000
Overall ∅	537.76	6592.72	20028.01
	Memory (MB)		
	w=100	w=1000	w=5000
Overall ∅	46.85	269.65	2000

Table 3 presents the results for distinct sizes of w and shows that; for shorter windows ($w = 100$), the accuracy degrades, while for bigger windows the accuracy slightly increases. On the other hand, the processing time and memory usage increase as well. Therefore this parameter selection implies an accuracy-time-memory trade-off. The following experiments are performed with $w = 1000$ for kNN, because using a greater window size yields indeed to a better accuracy but the resource consumption is more significant.

Tables 4, 5, and 6 report the final accuracies, memory consumption, and speed of the classification models in a 40-dimensional space after the projections based on two setups of $k = 5, 11$. We choose 40 dimensions because we noticed that, starting from this size of space, improvements are statistically insignificant as showed in Figure 3. The latter illustrates a detailed comparison with five different values of output dimension $(10, 20, \cdots, 50)$ on Tweet$_2$ and Enron datasets.

Results with ensemble methods. We compare the proposed LB with CS-kNN as a base learner (CS-kNN^{LB}) and the CSB-kNN with a different CS matrix for each learner, both using 10 learners (the size of ensemble) and $k = 5$, against popular ensemble methods such as the adaptive random forest (ARF) [18] and leveraging bagging using Hoeffding tree [14] as base learner (HTree^{LB}), with 30 and 10 ensemble members, respectively. Tables 7, 8 and 9 display the performance of the ensembles. In this evaluation, each of the ensemble member uses the same CS matrix to perform the reduction into 40 dimensions, except CSB-kNN which sets up a different matrix for each member in attempt to assess the ensemble diversity impact.

4.3 Discussion

Our proposed CS-kNN has more accurate results (Table 4) than HT-kNN for all datasets and it is slightly outperformed by CS-sam$k$NN, the standard kNN (without projection) and PCA-kNN; this quite a

Table 4: Accuracy (%) comparison of CS-kNN, CS-sam$k$NN, HT-kNN, PCA-kNN, and kNN over the whole dataset.

Dataset	CS-kNN		CS-sam$k$NN		HT-kNN		PCA-kNN		kNN	
	$k = 5$	$k = 11$	$k = 5$	$k = 11$	$k = 5$	$k = 11$	$k = 5$	$k = 11$	$k = 5$	$k = 11$
Tweet$_1$	78.82	78.88	76.02	74.31	73.77	73.14	80.43	79.43	79.80	78.17
Tweet$_2$	78.13	78.36	75.74	74.13	73.02	72.61	80.06	78.89	79.20	77.74
Tweet$_3$	76.75	76.16	73.03	72.56	72.40	72.36	81.93	82.38	78.86	77.73
RBF	98.90	97.31	99.87	99.78	19.20	19.20	99.00	97.86	98.89	97.33
CNAE	70.00	68.70	73.77	72.19	65.00	65.28	75.83	72.08	73.33	71.48
Enron	96.02	95.65	96.23	96.06	95.76	95.48	94.59	93.18	96.18	96.00
IMDB	69.86	72.32	74.29	74.53	69.65	72.03	70.57	72.81	70.94	72.51
Spam	85.39	81.01	91.34	90.48	83.82	80.63	96.00	94.66	81.17	77.32
Covt	91.36	89.92	90.47	87.71	77.18	76.59	91.55	90.16	91.67	90.30
Overall ∅	82.80	82.04	83.42	82.42	69.98	69.70	85.55	84.61	83.34	82.06

Table 5: Time (sec) comparison of CS-kNN, CS-sam$k$NN, HT-kNN, PCA-kNN, and kNN over the whole dataset.

Dataset	CS-kNN		CS-sam$k$NN		HT-kNN		PCA-kNN		kNN	
	$k = 5$	$k = 11$	$k = 5$	$k = 11$	$k = 5$	$k = 11$	$k = 5$	$k = 11$	$k = 5$	$k = 11$
Tweet$_1$	62.55	91.06	41.81	59.20	93.24	99.78	622.65	629.60	1198.78	1432.47
Tweet$_2$	107.48	112.97	74.92	99.77	120.83	127.95	705.71	712.84	2029.82	2502.92
Tweet$_3$	126.73	142.95	83.01	101.43	154.22	165.11	988.25	995.93	2864.55	3643.26
RBF	59.47	80.52	60.08	77.00	168.31	169.88	243.26	258.12	284.34	439.23
CNAE	0.87	0.92	0.56	0.63	0.95	1.02	3.97	4.14	32.19	35.04
Enron	1.58	1.63	1.31	1.57	1.81	1.90	7.21	7.28	86.08	91.99
IMDB	95.62	120.66	80.82	103.51	125.62	129.27	1686.88	1692.28	7892.96	8217.06
Spam	159.92	183.19	197.22	208.94	194.07	216.37	11329.91	14820.26	34231.45	35031.76
Covt	30.94	51.08	39.25	45.55	88.17	90.85	161.00	164.16	252.69	268.28
Overall ∅	71.68	87.22	64.33	75.29	105.25	111.42	1749.87	2142.73	5430.32	5740.22

Table 6: Memory (MB) comparison of CS-kNN, CS-sam$k$NN, HT-kNN, PCA-kNN, and kNN over the whole dataset.

Dataset	CS-kNN	CS-sam$k$NN	HT-kNN	PCA-kNN	kNN
Tweet$_1$	2.52	8.86	2.52	3.03	34.64
Tweet$_2$	2.52	10.48	2.52	5.97	70.97
Tweet$_3$	2.52	10.52	2.52	8.84	103.19
RBF	2.52	10.31	2.52	8.86	13.18
CNAE	2.52	10.22	2.52	3.09	61.37
Enron	2.52	9.84	2.52	3.51	70.60
IMDB	2.52	10.28	2.52	8.81	70.65
Spam	2.52	10.57	2.52	245.22	1476.11
Covt	2.52	9.96	2.52	3.02	3.47
Overall ∅	2.52	10,12	2.52	32.26	211.57

natural result since kNN processes the whole data stream and PCA-kNN formally tries to find a lower-dimensional space under which the sum of square distances – representing the error, between the original data and its projection – is minimized. CS-kNN is moderately less accurate than CS-sam$k$NN for some datasets containing drifts, because the latter deals with different types of concept drift which makes it stronger facing changes in data distributions.

To assess the benefits in terms of computational resources – where small values are desirable – Tables 5 and 6 point out the improvements of CS-kNN in terms of memory and time against CS-sam$k$NN, PCA-kNN, and kNN which are significant enough to justify relatively minor losses in accuracy. In fact, CS-sam$k$NN maintains models for current and past concepts which makes it memory inefficient. PCA-kNN performs worse, in terms of resource usage, than RP since it incrementally stores and updates the eigenvectors and eigenvalues, confirming previous studies [37]. Our proposed approach is also faster than HT-kNN, although they have similar memory behavior, because both are based on RP and do not rely on data.

For some datasets such as Spam, CS-kNN outperforms kNN (using the whole data) simply because finding relevant combinations of existing features and presenting them in a different space can help supervised models to improve accuracy. Even if data are not sparse, CS surprisingly performs projections on suitable dimensions.

Figures 3a and 3d depict the typical trade-off for accuracy: a small feature space cannot properly represent data, therefore it can significantly degrade the accuracy; whereas a higher dimensional space (e.g., 50) increases the accuracy and makes it closer to the results with kNN. We also notice the stability of our CS-kNN, i.e., the accuracy is linearly boosted with the target space size and converges to the accuracy of kNN. On the other hand, CS-sam$k$NN, HT-kNN and PCA-kNN have different behaviors, clearly illustrated in Figure 3a; this results deduce that, in practice, it may be hard to fix a proper space size. We show that kNN, PCA-kNN and HT-kNN are outperformed in terms of processing time (Figures 3b and 3e) and that CS-kNN requires also less memory compared to these baselines. For instance, with Tweet2 and Enron in Figures 3c and 3f respectively, we observe large gains compared to kNN, PCA-kNN, and CS-sam$k$NN, albeit our proposal has the same memory usage as HT-kNN because both do not rely on data. We also observe that the behavior of memory usage is correlated to the running time trends, i.e., when the memory usage increases, the processing time also increases accordingly.

Tables 7, 8 and 9 show, using only 10 learners, CSB-kNN performs better than the reputed CS-ARF [18] using 30 learners (trees) on most of the datasets. We noticed that with CSB-kNN, when the features set is large (e.g. Spam), the memory usage is relatively high. On the other hand, for large datasets (e.g. Tweets), CS-ARF and CS-HTree^{LB} require more memory whereas our approaches use less, making them useful for the stream setting. CS-kNN^{LB} is the most memory efficient and even proving competitive with CS-HTree^{LB}

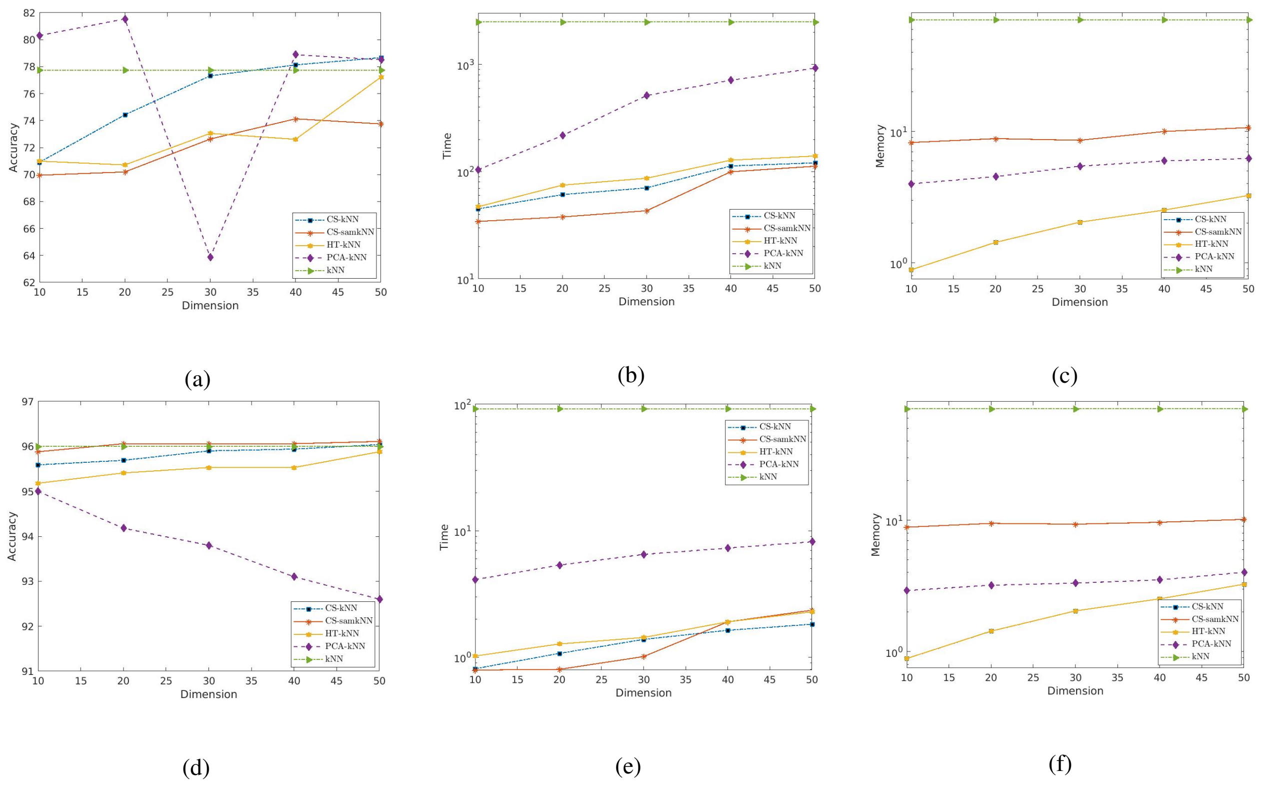

Figure 3: Sorted plots of accuracy, time and memory over different output dimensions. **a** Accuracy Tweet$_2$. **b** Time Tweet$_2$. **c** Memory Tweet$_2$. **d** Accuracy Enron. **e** Time Enron. **f** Memory Enron.

Table 7: Accuracy (%) comparison of CS-kNN^{LB} , CSB-kNN, CS-HTree^{LB}, and CS-ARF

Dataset	CS-kNN^{LB}	CSB-kNN	CS-HTree^{LB}	CS-ARF
Tweets$_1$	78.94	81.80	81.35	81.53
Tweets$_2$	78.24	81.28	80.39	80.75
Tweets$_3$	76.06	80.40	78.59	79.54
RBF	98.90	99.68	99.24	99.25
CNAE	71.64	81.48	65.70	62.55
Enron	95.94	96.00	96.17	95.88
IMDB	70.02	74.27	74.80	74.88
Spam	86.08	90.28	90.02	89.04
Covt	91.09	91.76	88.48	88.01
Overall ∅	82.99	86.33	83.86	83.49

Table 8: Time (SEC) comparison of CS-kNN^{LB}, CSB-kNN, CS-HTree^{LB}, and CS-ARF

Dataset	CS-kNN^{LB}	CSB-kNN	CS-HTree^{LB}	CS-ARF
Tweets$_1$	1130.44	1251.77	82.18	170.52
Tweets$_2$	1449.44	1526.30	105.87	212.69
Tweets$_3$	1668.26	1825.41	127.19	239.97
RBF	735.21	772.62	90.22	223.08
CNAE	8.99	11.02	1.80	4.66
Enron	20.07	21.92	2.11	3.78
IMDB	1552.81	1649.94	90.17	174.54
Spam	359.07	2194.93	218.16	270.15
Covt	612.62	694.02	41.69	115.3
Overall ∅	837.43	1105.33	84.37	108.70

Table 9: Memory (MB) comparison of CS-kNN^{LB}, CSB-kNN, CS-HTree^{LB}, and CS-ARF

Dataset	CS-kNN^{LB}	CSB-kNN	CS-HTree^{LB}	CS-ARF
Tweets$_1$	6.16	27.13	60.71	175.71
Tweets$_2$	6.16	28.97	66.75	177.32
Tweets$_3$	6.16	30.80	73.89	176.92
RBF	6.16	25.96	9.91	25.90
CNAE	6.15	28.11	0.48	1.31
Enron	6.15	28.59	1.59	4.10
IMDB	6.16	28.60	5.60	18.63
Spam	5.38	151.91	5.15	10.44
Covt	6.16	24.10	4.44	11.66
Overall ∅	6.07	41.57	25.39	66.89

and CS-ARF. However, this is at the price of being slower. Also, computational resources of CSB-kNN with different CS matrices increase considerably to allow higher accuracy and diversity (enabling the ensemble to generalize well).

In conclusion, our CSB-kNN ensemble method has good overall performance compared to other methods. We showed that our proposal can be used to classify accurately data streams with a large number of attributes using a relatively small number of base learners, in contrast with CS-ARF the number of base trees can considerably affect the classification performance.

5 Conclusions

In this work, we presented a scheme to enable the k-nearest neighbors algorithm to be efficient with evolving high-dimensional data streams in terms of classification performance and computational resources (memory and time) after space transformations provided by compressed sensing given its ability to ensure theoretical lower and upper bounds on pairwise data transformations. Our first contribution is the mix of two main ingredients: compressed sensing and kNN, thus resulting in the CS-kNN algorithm designed to work on evolving data streams while operating on a reduced feature space.

We proposed also an ensemble method, CSB-kNN, that uses CS-kNN as base learner under the Leveraging Bagging, where each ensemble member has a different CS matrix to help increasing the overall accuracy. We showed theoretically that CS-kNN using Gaussian matrices, the neighborhood distance used in kNN is preserved up to some $1 \pm \epsilon$-factor. The key idea is to show that squared kNN distances, in the original data, are too within the same factor. Consequently, our CS-kNN algorithm also conserves such distances.

We evaluated the proposed algorithms via extensive experiments using synthetic and real-world datasets with different parameters. Results show the potential of the CS-kNN and CSB-kNN algorithms to obtain close approximations to what it would be obtained using the input instances from data streams. We compared our proposals against well-known approaches from the literature, showing improvements along 3 dimensions: accuracy, memory usage, and time.

In future work, we intend to investigate how to optimize the processing time of CSB-kNN algorithm and provide guarantees along the number of output dimensions. Once we fix the latter, we could efficiently project the data streams knowing the input dimensions.

Acknowledgements

This work was done in the context of IoTA AAP Emergence DigiCosme Project and was funded by Labex DigiCosme.

References

[1] Dimitris Achlioptas, ‘Database-friendly random projections: Johnson-lindenstrauss with binary coins’, *JCSS*, **66**(4), 671–687, (2003).

[2] Youness Arjoune, Naima Kaabouch, Hassan El Ghazi, and Ahmed Tamtaoui, ‘A performance comparison of measurement matrices in compressive sensing’, *International Journal of Communication Systems*, **31**(10), e3576, (2018).

[3] Maroua Bahri, Silviu Maniu, and Albert Bifet, ‘A sketch-based naive bayes algorithms for evolving data streams’, in *International Conference on Big Data*, pp. 604–613. IEEE, (2018).

[4] Richard Baraniuk, Mark Davenport, Ronald DeVore, and Michael Wakin, ‘A simple proof of the restricted isometry property for random matrices’, *Constructive Approximation*, **28**(3), 253–263, (2008).

[5] Jean Paul Barddal, Heitor Murilo Gomes, Fabrício Enembreck, and Bernhard Pfahringer, ‘A survey on feature drift adaptation: Definition, benchmark, challenges and future directions’, *Journal of Systems and Software*, **127**, 278–294, (2017).

[6] Albert Bifet, Ricard Gavaldà, Geoff Holmes, and Bernhard Pfahringer, *Machine learning for data streams: with practical examples in MOA*, MIT Press, 2018.

[7] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer, ‘Moa: Massive online analysis’, *JMLR*, **11**(May), 1601–1604, (2010).

[8] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer, ‘Leveraging bagging for evolving data streams’, in *Joint European conference on machine learning and knowledge discovery in databases*, pp. 135–150. Springer, (2010).

[9] Albert Bifet, Bernhard Pfahringer, Jesse Read, and Geoff Holmes, ‘Efficient data stream classification via probabilistic adaptive windows’, in *SIGAPP*, pp. 801–806. ACM, (2013).

[10] Alfred M Bruckstein, David L Donoho, and Michael Elad, ‘From sparse solutions of systems of equations to sparse modeling of signals and images’, *SIAM review*, **51**(1), 34–81, (2009).

[11] Avishy Y Carmi, Lyudmila Mihaylova, and Simon J Godsill, *Compressed sensing and sparse filtering*, Springer, 2014.

[12] Patrick Marques Ciarelli and Elias Oliveira, ‘Agglomeration and elimination of terms for dimensionality reduction’, in *ISDA*, pp. 547–552. IEEE, (2009).

[13] John P Cunningham and Zoubin Ghahramani, ‘Linear dimensionality reduction: Survey, insights, and generalizations’, *JMLR*, **16**(1), 2859–2900, (2015).

[14] Pedro Domingos and Geoff Hulten, ‘Mining high-speed data streams’, in *SIGKDD*, pp. 71–80. ACM, (2000).

[15] David L Donoho, ‘Compressed sensing’, *IEEE Transactions on Information Theory*, **52**(4), 1289–1306, (2006).

[16] Herbert Edelsbrunner and John Harer, ‘Persistent homology-a survey’, *Contemporary Mathematics*, **453**, 257–282, (2008).

[17] Nikolaos M Freris, Orhan Oçal, and Martin Vetterli, ‘Compressed sensing of streaming data’, in *51st Annual Allerton Conference on Communication, Control, and Computing*, pp. 1242–1249. IEEE, (2013).

[18] Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem, ‘Adaptive random forests for evolving data stream classification’, *Machine Learning*, 1–27, (2017).

[19] Simon Günter, Nicol N Schraudolph, and SVN Vishwanathan, ‘Fast iterative kernel principal component analysis’, *JMLR*, **8**(8), 1893–1918, (2007).

[20] Harold Hotelling, ‘Analysis of a complex of statistical variables into principal components.’, *Journal of Educational Psychology*, **24**(6), 417, (1933).

[21] William B Johnson, Joram Lindenstrauss, and Gideon Schechtman, ‘Extensions of lipschitz maps into banach spaces’, *Israel Journal of Mathematics*, **54**(2), 129–138, (1986).

[22] Ioannis Katakis, Grigorios Tsoumakas, Evangelos Banos, Nick Bassiliades, and Ioannis Vlahavas, ‘An adaptive personalized news dissemination system’, *Journal of Intelligent Information Systems*, **32**(2), 191–212, (2009).

[23] Bryan Klimt and Yiming Yang, ‘The enron corpus: A new dataset for email classification research’, in *ECML*, pp. 217–226. Springer, (2004).

[24] Felix Krahmer and Rachel Ward, ‘New and improved johnson–lindenstrauss embeddings via the restricted isometry property’, *Mathematical Analysis*, **43**(3), 1269–1281, (2011).

[25] Hanxi Li, Chunhua Shen, and Qinfeng Shi, ‘Real-time visual tracking using compressive sensing’, in *CVPR*, pp. 1305–1312. IEEE, (2011).

[26] Huan Liu and Hiroshi Motoda, *Feature extraction, construction and selection: A data mining perspective*, volume 453, Springer Science & Business Media, 1998.

[27] Huan Liu and Hiroshi Motoda, *Computational methods of feature selection*, CRC Press, 2007.

[28] Viktor Losing, Barbara Hammer, and Heiko Wersing, ‘Knn classifier with self adjusting memory for heterogeneous concept drift’, in *ICDM*, pp. 291–300. IEEE, (2016).

[29] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts, ‘Learning word vectors for sentiment analysis’, in *ACL-HLT*, pp. 142–150. Association for Computational Linguistics, (2011).

[30] Thu LN Nguyen and Yoan Shin, ‘Deterministic sensing matrices in compressive sensing: a survey’, *The Scientific World Journal*, (2013).

[31] Chenlu Qiu, Wei Lu, and Namrata Vaswani, ‘Real-time dynamic mr image reconstruction using kalman filtered compressed sensing’, in *ICASSP*, pp. 393–396. IEEE, (2009).

[32] Jesse Read, Albert Bifet, Bernhard Pfahringer, and Geoff Holmes, ‘Batch-incremental versus instance-incremental learning in dynamic and evolving data’, in *IDA*, pp. 313–323, (2012).

[33] Donald R Sheehy, ‘The persistent homology of distance functions under random projection’, in *SoCG*, p. 328. ACM, (2014).

[34] Jean-Luc Starck, Fionn Murtagh, and Jalal Fadili, *Sparse image and signal processing: Wavelets and related geometric multiscale analysis*, Cambridge university press, 2015.

[35] M Uttarakumari, Ashray V Achary, Sujata D Badiger, DS Avinash, Anisha Mukherjee, and Nancy Kothari, ‘Vehicle classification using compressive sensing’, in *RTEICT*, pp. 692–696. IEEE, (2017).

[36] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg, ‘Feature hashing for large scale multitask learning’, in *ICML*, pp. 1113–1120. ACM, (2009).

[37] Yair Weiss, Hyun Sung Chang, and William T Freeman, ‘Learning compressed sensing’, in *Snowbird Learning Workshop*. Citeseer, (2007).

[38] Juyang Weng, Yilu Zhang, and Wey-Shiuan Hwang, ‘Candid covariance-free incremental principal component analysis’, *TPAMI*, **25**(8), 1034–1040, (2003).