# Classifying High-dimensional Data Streams using Adaptive Learning and t-Stochastic Neighborhood Embedding

André de M. Wlodkovski
Escola Politécnica, Pontifícia Universidade Católica do Paraná (PUCPR), Brazil

Kalebe R. Szlachta
Escola Politécnica, Pontifícia Universidade Católica do Paraná (PUCPR), Brazil

Jean Paul Barddal
Graduate Program in Informatics (PPGIa), Pontifícia Universidade Católica do Paraná (PUCPR), Brazil

## ABSTRACT

Data streams can be defined as a potentially infinite sequence of elements that are used to represent data that becomes available over time for both testing and model updates. In this paper, we focus on high-dimensional data streams that cause classifications harder due to data sparsity. Our proposal is to apply parametric t-Stochastic Neighborhood Embedding (t-SNE) on high-dimensional data streams, which is a variation of the regular SNE, but based on Student t-distribution instead of a Gaussian distribution. This change makes the t-SNE easily optimizable compared to its parent because its cost function complexity is quadratic to the number of data points, while the cost function complexity of SNE is exponential. This is relevant for streaming scenarios as the trade-off between accuracy and processing time is relevant. Furthermore, parametric t-SNE allows model updates, which is relevant for continuous learning. We have applied parametric t-SNE to multiple classifiers for streaming data and on different dimensionality settings. Results showed that the application of parametric t-SNE as a preprocessing step for handling high-dimensional data streams did not yield superior classification rates when compared to the original data representation.

## 1 INTRODUCTION

Data streams are fast-paced, potentially infinite, and ubiquitous data sequences. Examples of streaming data include, but are not limited to: banking transactions, readings from mobile sensors, posts in social media, and stock market values and trades. Data stream mining is a highly active research topic since both researchers and practitioners are consistently developing novel techniques for learning and updating predictive and descriptive models from such data flows.

Focusing on data stream classification, algorithms tailored for such task must (i) process instances sequentially according to their arrival, (ii) act within limited memory space and processing time, (iii) deal with data instability (concept drifts) [31]; and (iv) generalize well as instances' labels become available [13].

*Conference'17, July 2017, Washington, DC, USA*
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

In this paper we are particularly concerned with high-dimensional data streams. Despite the abundant and potentially ever-flowing amount of data, classifying high dimensional data presents a special challenge for streaming algorithms as (i) data becomes more sparse, and (ii) feature importance values may drift over time, a phenomenon named feature drift [5].

To overcome the challenges presented by high-dimensional data streams, we propose the continuous application of manifold learning to learn a low-dimensional data representation as a preprocessing step in a classification system. More specifically, we focus on a parametric version of the t-Stochastic Neighborhood Embedding (t-SNE) algorithm [29], which has been proposed in [22], and show how it can be continuously adapted over a data stream and coupled with existing data stream classifiers.

This paper is divided as follows. Section 2 brings forward a formalization of data stream classification, concept drift, and the problem of classifying high-dimensional data streams. Section 3 discusses related works that handle high-dimensional data streams, either via feature selection, feature weighting, or dimensionality reduction. Section 4 brings forward our proposed framework for continuously adapting a parametric t-SNE manifold in data streams. Next, Section 5 brings forward an experimental analysis of the proposed framework. Finally, Section 6 concludes this paper and states future works.

## 2 DATA STREAM CLASSIFICATION

In this paper, we formally denote the classification task as follows. Given a set of classes $Y = \{y_1, \ldots, y_c\}$ and a set of labeled instances $D = \{(\vec{x}^1, y^1), \ldots, (\vec{x}^n, y^n)\}$, a classifier builds a predictive model $f : x \longrightarrow y$ for predicting unlabeled instances. More precisely, we assume that $\vec{x}$ is a $d$-dimensional feature vector in $\mathbb{R}^d$.

Similarly to batch scenarios, classification problems are widely tackled in data streams. Data stream classification is a variant of traditional batch classification in which data becomes available over time for both testing and model updates instead of in a fully-available batch [13]. Hereafter, we denote $\mathcal{S} = [(\vec{x}^t, y^t)]_{t=0}^{\infty}$ to be a data stream providing instances in the $(x^t, y^t)$ format, each of which arriving at a timestamp $t$.

### 2.1 Concept Drift

One of the biggest traits of streaming scenarios is concept drift [30, 31], in which the data distribution changes over time. There are numerous cases related to concept drift, many of which are associated to the real world. A recent event example that caused radical changes with concept drift in various scenarios was the COVID-19 pandemic, which radically changed the world's lifestyle, including the way that people buy products and visit public places for example. In general, concept drift causes accuracy decay on machine learning models, making worse predictions, and providing wrong results to the final objective. As highlighted in seminal and recent works [13, 25], it is also noteworthy that changes and their reasons are unpredictable, and thus, predictive models must automatically detect and adapt to them.

In this paper we denote a concept $C = \bigcup_{y \in Y} \{(P[y], P[\vec{x}|y])\}$ to be a ser of prior class probabilities and class-conditional probability density functions [30]. Therefore, given a data stream $\mathcal{S}$, instances $(\vec{x}^t, y^t)$ will be generated by the current concept $C^t$. If during every instant $t_i$ of $\mathcal{S}$ we have $C^{t_i} = C^{t_{i-1}}$, the concept is said to be stable. Otherwise, if between any two timestamps $t_i$ and $t_j = t_i + \Delta$, with $\Delta > 1$, occurs that $C^{t_i} \neq C^{t_j}$, we have a concept drift.

To deal with changes, data stream learning systems often employ drift detectors to identify changes. Regarding drift detectors, the most used approaches receive as input the classification error rates, and whenever a significant change is observed, a drift is flagged. Approaches like Drift Detection Method (DDM) [14], Adaptive Sliding Window (ADWIN) [7], and Hoeffding Drift Detection Method (HDDM) [12] fall within this category. The interested reader can refer to the works of [15] for an extensive analysis on concept drift detectors.

### 2.2 Classifying High-dimensional Data Streams

Classifying high-dimensional data streams is particularly challenging as the processing time and memory consumption aspects scale up quickly, and like low-dimensional data streams, they must be processed as quickly as possible to avoid data congestion, and potential system crashes. Over the recent years, researchers have devoted significant efforts towards handling high-dimensional data, either via (i) feature selection, (ii) feature weighting, or (iii) dimensionality reduction [5]. Feature selection focuses on selecting a subset of features that best describes the concept to be learned. Next, feature weighting assigns importance scores to features so that the predictive model emphasizes them accordingly over time. Finally, dimensionality reduction compresses data by combining features so that the core aspects and data structure are maintained. The next section brings forward existing works that fall in the categories above.

## 3 RELATED WORKS

In this section we bring forward existing works that tackle high-dimensional data stream classification either via (i) feature selection, (ii) feature weighting, or (iii) dimensionality reduction.

### 3.1 Feature Selection

Data streams may contain a large number of features, many of which may be irrelevant or redundant for the learning task. Dealing with a high number of features is not only computationally expensive but it also jeopardizes inductive learning due to data sparsity and the curse of dimensionality [1]. Therefore, it is of interest to continuously select which features are relevant and non-redundant during the stream processing [5].

A seminal work to address feature selection in data streams is DynamIc SymmetriCal Uncertainty Selection for Streams (DISCUSS) [4]. DISCUSS has at its core the Symmetrical Uncertainty scoring operator that quantifies how important a feature it w.r.t. class prediction and within a sliding window that reflects recent data. During experimentation, DISCUSS showed limited computational resources consumption and accuracy improvements when applied to different classifiers. Nonetheless, authors have also showed that DISCUSS is is unable to depict higher-order interactions between relevant features and the class since it performs a "flat" and individual evaluation of the features.

One of the methods proposed to overcome DISCUSS' limitations is the Iterative Subset Selection (ISS) algorithm [33]. ISS splits the feature selection process into two stages by first ranking the features using a scoring function, e.g., Average Euclidean Distance, Information Gain, and Symmetrical Uncertainty; and then iteratively selecting feature subsets according to this ranking and also using Backward Feature Elimination [32].

Finally, authors in [3] proposed a dynamic feature selection method based on Boosting [11] called Adaptive Boosting for Feature Selection (ABFS). ABFS chains decision stumps and drift detectors to identify high order interactions between features and the target. Experimental results have showed that ABFS leverages different the recognition rates of different classifiers in feature drifting scenarios, yet, it selects too many falsely relevant features in high-dimensional data streams.

### 3.2 Feature Weighting

Feature weighting is a technique used to approximate the optimal degree of influence of features in data. When successful, relevant features are attributed with high weights, whereas irrelevant features are associated with weight values close to zero. Feature weighting is broadly used in batch learning [9], while very few feature weighting techniques to streaming environments have been proposed so far [2]. Here, we highlight the work of [6], in which authors proposed a dynamic feature weighting scheme based on Shannon's entropy [27, 28]. This work is focused on feature drifts, and thus, the rationale is to continuously update feature-independent, class-conditional entropy values, and information gain with low computational cost. These weights were also used as part of the learning processes of bayesian, decision trees, and k-nearest neighbor classifiers. The results showed major improvements of all classifiers in feature-drifting scenarios, while at the cost of bounded processing time and memory consumption usage that scaled linearly according to the number of features in the data stream.

### 3.3 Dimensionality Reduction

In opposition to feature selection, the rationale behind dimensionality reduction is to transform a high-dimensional problem into a small-dimensional one by learning manifolds that combine the original features according to some heuristic [22]. A famous approach for dimensionality reduction is Principal Component Analysis (PCA) [18]. PCA has been widely studied in batch scenarios and its goal is to conduct dimensionality reduction so that the new feature space is given by uncorrelated variables. Regarding streaming contexts, we highlight the work of [19] in which PCA was applied in data chunks for unsupervised drift detection. A memory-limited and pure online approach for PCA in streaming data is the work of [23], yet, it does not encompasses drift detection nor forgetting for drift adaptation.

More recently, authors in [21] proposed t-Stochastic Neighbor Embedding (t-SNE), which was initially tailored for reducing the dimensionality of a dataset focusing on visualization. t-SNE allows data visualization different scales on a single map, retaining local structures and revealing global structures, such as similar data groups in different scales. However, the original t-SNE does not learn a mapping between the original features and those in the lower-dimensional feature space. Therefore, it cannot be applied to supervised machine learning scenarios. There is, however, a parametric version of t-SNE proposed in [22] that comes up for this regard, while also focusing on greater efficiency when applied to larger amounts of data. Details on t-SNE and its parametric counterpart are given in the following section alongside our proposal.

## 4 PROPOSAL

In this section we bring forward our proposal of applying parametric t-SNE to tackle high-dimensional data stream classification. First, we detail t-SNE and its parametric counterpart. Next, we show t-SNE can be applied as a pre-processing step for dimensionality reduction in data stream classification.

### 4.1 t-Stochastic Neighbor Embedding

t-SNE (t-Stochastic Neighborhood Embedding) [29] is a variation of SNE (Stochastic Neighborhood Embedding), which is a data dimensionality reduction technique that focuses on generating a bi- or tri-dimensional manifold. This technique was introduced to optimize data visualization and was compared against other dimensionality reduction techniques, such as Sammon, CCA, SNE and Isomap.

t-SNE relies on core concepts as presented in [29], such as Kullback-Leibler divergence between joints, pairwise similarities in the low-dimensional and high-dimensional map and a gradient equation as part of the symmetric SNE, and a Student t-distribution that is used to convert distances into probabilities in order to mitigate the SNE crowding problem described in the research.

First, the low-dimensional space pairwise similarities for any point $q_{ij}$ is given by Equation 1.

$$q_{ij} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq l} exp(-\|y_k - y_l\|^2)} \tag{1}$$

On the other hand, the high-dimensional space pairwise similarities for any point $p_{ij}$ is given by Equation 2.

$$p_{ij} = \frac{exp(-\|x_i - x_j\|^2/2\sigma^2)}{\sum_{k \neq l} exp(-\|x_k - x_l\|^2/2\sigma^2)} \tag{2}$$

After calculating both the high-dimensional and low-dimensional probabilities, the Kullback-Leibler divergence measures the faithfulness between a joint $P$ in the high-dimensional space and a joint $Q$ in the low-dimensional space, given by Equation 3.

$$C = KL(P||Q) = \sum_i \sum_j p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{3}$$

The gradient of the symmetric SNE minimizes a single Kullback-Leibler divergence between the data space and the latent space and is determined by Equation 4.

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) \tag{4}$$

When it comes to t-SNE, there are a few changes from the symmetric SNE. The joint probability for any point $q_{ij}$ includes the Student t-distribution with one degree of freedom in the low-dimensional map, acting as a heavy-tailed distribution, and is defined by Equation 5:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2)^{-1}} \tag{5}$$

The second aspect where the t-SNE differs from the symmetric SNE is the function where the Kullback-Leibler divergence between joints $P$ and $Q$ is minimized, given by Equation .

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1} \tag{6}$$

Compared to the other techniques, t-SNE brought better results in the MNIST dataset [20], for example. In the meantime, the other databases had some mixtures and overlaps between the groups, and the t-SNE revealed in detail the local data structure, in addition to presenting a clear separation between the classes, showing only some inconsistencies, caused by distorted digits that are being mixture with other groups to which do not belong.

However, t-SNE presents some weaknesses. The first is that it is not clear about the performance of the t-SNE in cases where the data is reduced to a dimensionality greater than 3, that is, in cases with more dimensions than a three-dimensional map. The second is that t-SNE can become sensitive to the intrinsic characteristics of a database by reducing the dimensionality based on local properties, and it may not be as successful in datasets with high intrinsic dimensionality. The last among the main weaknesses is its cost function, which is not convex, making it highly dependent on the parameters chosen for its optimization.

### 4.2 Parametric t-Stochastic Neighbor Embedding

Parametric t-SNE (Parametric t-Stochastic Neighborhood Embedding) [22] is a t-SNE variant that proves to be more efficient than its counterparts for dimensionality reduction. More specifically, it is a technique that learns a parametric mapping between the high-dimensional data space and the low-dimensional data space in the form of a neural network, which is trained to preserve local data structures. Parametric t-SNE is inspired in the training of autoencoders based on Restricted Boltzmann Machines (RBMs), which is followed by backpropagation. The steps encompassed by parametric t-SNE are as follows: (i) RBM stack training, (ii) pre-trained neural network construction, and (iii) neural network calibration with backpropagation.

In opposition to the regular t-SNE, the parametric variant may use multiple degrees of freedom, instead of the latter, which only uses one. The definition of the mapping from the data space to the latent space, modified from the regular t-SNE, leads to Equation 7:

$$q_{ij} = \frac{(1 + ||f(x_i|W) - f(x_j|W)||^2/\alpha)^{-\frac{\alpha+2}{2}}}{\sum_{k \neq l}(1 + ||f(x_k|W) - f(x_l|W)||^2/\alpha)^{-\frac{\alpha+2}{2}}} \quad (7)$$

where $q_{ij}$ represents the pairwise similarities in the latent space.

Next, an RBM is a fully connected, bipartite, graphical probabilistic model in which one group of nodes models the data and the other group models the hidden (latent) structure of the data. RBMs are trained to minimize the Kullback-Leibler divergence (see Equation 3), which measures the difference between two probability distributions.

Finally, calibration is done by converting the distances between pairs in both spaces into probabilities that measure the similarity of the two data points. The $\alpha$ parameter of the Student's t-distribution is also considered, which characterizes its degree of freedom. The higher the value of $\alpha$, the lighter the tails of the distribution, leading to a correction for the exponential growth of the hidden space volume. This degree can be defined in three different ways:

(1) Fixed value, usually $\alpha = 1$, but this approximation does not correct for exponential growth.
(2) Linear relationship with the dimensionality of latent space.
(3) By learning, in which, unlike the previous one, more attributes are considered in addition to the latent space, since there is no 100% between both variables. Therefore, this approach also considers the agglomerations problem.

## 4.3 Proposed framework

In this paper our proposal is to apply parametric t-SNE to convert a high-dimensional data stream into a low-dimensional one. Since parametric t-SNE is built on neural networks, we constantly update it as new data becomes available.

In order to join the parametric t-SNE and the data streams together, the TSNEClassifier class was created, wrapping up both components with its own t-SNE parameters and data stream classifier methods, such as "fit", "predict" and "learn one". When it comes to learning new data, the parametric t-SNE is always trained before the data streams, so the data can be transformed into a low dimensional dataset. Similar steps are executed during the prediction phase, however, since the parametric t-SNE has already been trained during the learning phase, it just transforms high dimensional data into low dimensional data, and then the data stream classifier performs the prediction.

Algorithm 1 illustrates how the parametric t-SNE communicates with the classifier in order to reduce the dimensional space and learn a stream of $x$ points and $y$ labels. In practice, the parametric t-SNE algorithm learns the new data before transforming it into a low-dimensional set of *new_x* points that is given to the classifier to learn.

Next, Algorithm 2 illustrates how a given high-dimensional stream of $x$ points is predicted after fitting data. Differently than the fit algorithm, the parametric t-SNE does not learn the given stream. Instead, it only transforms the high-dimensional $x$ set of points into a low-dimensional *new_x* set of points, which will be forwarded to the classifier in order to do the prediction.

For the sake of research reproducibility, the source code for the proposed framework that combines River [24] and Luke Lee's t-SNE implementation[1] is publicly available at linkomitted[2]

## 5 ANALYSIS

In this section we analyzed the proposed framework for classifying data streams using parametric t-SNE. First, we delineate the experimental protocol, which is followed by a discussion on the results obtained.

### 5.1 Experimental Protocol

To assess the proposed framework we have developed a testbed containing multiple classifiers and datasets. Our implementation was made in Python and extended the River [24] framework. First, the validation protocol used was interleaved test-then-train (Prequential) [13]. In practice, $n$ instances were retrieved from the stream for testing and later for training, thus allowing the assessment of predictive models over time.

The experiment requires a few parameters in order to be run, and it is divided in three different parts: 2-component t-SNE classifiers, 3-component t-SNE classifiers and the same classifiers without the parametric t-SNE. All experiments have the same number of stream instances, burnout window size and block size, which are, respectively, 100,000, 1,000 and 1,000. The number of features is different for each experiment, yet, each parametric t-SNE experiment has its equivalent non-t-SNE experiment. For each experiment part, 6 experiments are run, with 50, 100, 200, 500, 1000 and 2000 features, respectively, and features are split in half between categorical and numerical data. In total, the resulting number of experiments is 18.

[1]https://github.com/luke0201/parametric-tsne-keras
[2]The actual link will be provided upon paper acceptance.

```
def fit(classifier: Classifier, tsne: ParametricTSNE, x: list[float],
  y: list[Discrete], num_instances: int):
    tsne.fit(x, y)
    new_x = tsne.transform(x)
    i = 0
    while $i < num\_instances$ do
        classifier.learn_one(new_x[i], y[i])
    end
    return (classifier, tsne)
```

**Algorithm 1:** t-SNE Classifier fit

```
def predict(classifier: Classifier, tsne: ParametricTSNE, x:
  list[float], num_instances: int):
    new_x = tsne.transform(x)
    predictions = []
    i = 0
    while $i < num\_instances$ do
        prediction = classifier.predict_one(new_x[i])
        predictions.append(prediction)
    end
    return predictions
```

**Algorithm 2:** t-SNE Classifier predict

The following classifiers from the River[24] framework were chosen in this experiment: ADWIN Bagging [26], Oza's AdaBoost [11], Adaptive Random Forest [16], Oza's Online Bagging [26], Leveraging Bagging [8].

The chosen synthetic data streams from the River library were the ones able to generate an unlimited amount of data, as well as having the number of features as a parameter, so the t-SNE Classifier can be experimented with different amounts of dimensions. The following synthetic databases have been used:

(1) Hyperplane: Generates a problem of prediction class of a rotation hyperplane. A hyperplane of $d$ dimensions is the set of points $x$ that satisfy the following equation: [17]

$$\sum_{i=1}^{d} w_i x_i = w_0 = \sum_{i=1}^{d} w_i$$

where $x_i$ and $w_i$ represent the $i$-th coordinate of the set of points and their weights, respectively.

(2) Random Radial Basis Function: Produces a radial basis function stream with an $n$ number of centroids, each of them having a random central position, standard deviation, class label and weights, generating a normally distributed hypersphere of samples on the surrounds of each centroid [24].

(3) Random Radial Basis Function with concept drift: An extension of the Random Radial Basis Function, with the addition of concept drifts by adding a "speed" to randomly selected centroids [24].

(4) Random Tree: River's [24] implementation is based on Hoeffding Randomized Trees[10], where a random tree is created by splitting features at random and each leaf becomes a label.

As the class distribution in the experiments was even, the only evaluation method chosen is accuracy. Finally, the experiments were accelerated with GPU computing as there is a large number of instances to be processed and the parametric t-SNE transformations may take advantage of it, making possible more experiments with larger data. The source code for our implementation and experiments is available at linkomitted.

## 5.2 Discussion

Tables 1 to 6 show the resulting accuracy value for each database with each classifier, along with whether the database had no transformations, 2-component t-SNE transformations or 3-component t-SNE transformations. Accuracy values are values between 0 and 100, where 0 represents null accuracy and 100 represents perfect accuracy.

After obtaining the results, each classifier, along with the accuracy values from each experiment type, was submitted to the Friedman test, followed by the Nemenyi test, in order to determine whether there was a significant difference from one model to the other. Pairs of orange-colored columns belonging to the same classifier represent models that are significantly different from each other, such as in Table 1, where the Adaptive Random Forest Classifier without t-SNE is significantly different than the same with two-dimensional t-SNE.

As shown in Tables 1 to 3, the accuracy values for classifiers without the t-SNE classifier are significantly higher for most databases, with the exception of AdaBoost Classifier and Leveraging Bagging Classifier with the Random Tree database in Tables 1 and 2, and ADWIN Bagging Classifier and Bagging Classifier along with the same latter database in Table 3. Although in these cases the t-SNE classifiers achieved better accuracy, they only slightly overcame the standard classifiers.

Tables 4 and 5 show accuracy values for experiments with higher dimensionality and, while still heavily overcome by the standard classifiers, the t-SNE Classifiers yielded better results in a few more scenarios compared to tables 1 to 3: three cases in table 4 and four cases in table 5, all of those in Random Tree databases. In fact, 4 of the 5 classifiers with 1000 features Random Tree performed better with 2 or 3 dimensional t-SNE, with the exception of the Bagging Classifier.

Therefore, although providing better accuracy for a few cases, the general outcome is that classifiers achieve higher accuracy without parametric t-SNE transformations, nullifying the H1 hypothesis, where the application of the parametric t-SNE on high dimensional data streams implies in higher classifier accuracy. Consequently, the null hypothesis is accepted, where the apliccation of the parametric t-SNE on high dimensional data streams does not imply in higher classifier accuracy.

## 6 CONCLUSION

Data Streams can be defined as an infinite sequence of elements that are used to represent data that becomes available over time for both testing and model updates [13], as presented in section 2.

Challenges experienced in High Dimensional Data Streams are data that becomes sparser, processing time, memory consumption and features that may drift over time, in a phenomenon called concept drift, in which data distribution changes causing data loss and accuracy decay, as given in Concept Drift section (2.1).

The purpose of this paper is to apply parametric t-SNE to transform a high dimensional data stream to a low dimensional one, using a neural network, that updates when new data becomes available. To merge parametric t-SNE with data streams, a new class was created (TSNE Classifier), wrapping up with its own t-SNE parameters and data stream classifier methods. To learn new data, the parametric t-SNE is trained before the data streams, to be able to transform high dimensional data sets to a low dimensional one. Section Proposed Framework 4.3 presents with more details.

Yet, in our results, on most databases, classifiers without the parametric t-SNE achieved a significantly higher accuracy, besides some classifiers, as presented in Discussion section 5.2. Hence, it was concluded that the null hypothesis was true, where the application of the parametric t-SNE on high dimensional data streams does not imply in higher classifier accuracy.

Therefore, parametric t-SNE shows potential for better results in High Dimensional Data situations. For this regard, some studies can be applied to verify the neural network and its weights to avoid bias, more tests with different classifiers and real data sets, that represents real world data problems.

**Table 1:** Accuracy values (%) for datasets with 50 features.

| | ADWINBaggingClassifier | | | AdaBoostClassifier | | | AdaptiveRandomForestClassifier | | | BaggingClassifier | | | LeveragingBaggingClassifier | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE |
| Hyperplane | 65.03 | 54.63 | 57.31 | 84.88 | 54.70 | 56.10 | 76.46 | 51.89 | 55.37 | 93.34 | 55.96 | 57.08 | 90.37 | 54.64 | 57.07 |
| RandomRBF | 76.17 | 57.52 | 54.43 | 99.08 | 73.76 | 80.13 | 94.53 | 92.16 | 89.52 | 82.53 | 60.11 | 60.51 | 88.21 | 65.76 | 58.52 |
| RandomRBFDrift | 71.90 | 53.31 | 54.98 | 99.17 | 73.14 | 79.33 | 95.66 | 85.60 | 90.90 | 83.08 | 62.94 | 61.99 | 90.82 | 58.30 | 68.65 |
| RandomTree | 58.38 | 53.61 | 51.98 | 46.32 | 49.92 | 52.04 | 57.69 | 50.26 | 52.37 | 53.93 | 51.37 | 50.58 | 55.80 | 55.43 | 59.51 |

**Table 2:** Accuracy values (%) for datasets with 100 features.

| | ADWINBaggingClassifier | | | AdaBoostClassifier | | | AdaptiveRandomForestClassifier | | | BaggingClassifier | | | LeveragingBaggingClassifier | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE |
| Hyperplane | 64.60 | 52.56 | 54.38 | 81.92 | 52.85 | 54.30 | 68.92 | 51.31 | 52.84 | 92.13 | 53.07 | 54.01 | 89.06 | 52.75 | 53.95 |
| RandomRBF | 73.93 | 55.86 | 57.85 | 99.87 | 77.47 | 82.41 | 97.85 | 93.39 | 92.65 | 98.02 | 65.64 | 60.38 | 98.07 | 63.47 | 61.56 |
| RandomRBFDrift | 81.15 | 58.55 | 58.17 | 99.69 | 77.19 | 84.94 | 98.09 | 93.13 | 90.86 | 97.36 | 56.35 | 60.78 | 98.69 | 61.14 | 61.72 |
| RandomTree | 57.13 | 51.37 | 56.66 | 46.03 | 56.79 | 50.41 | 51.60 | 50.31 | 50.31 | 67.89 | 56.34 | 52.52 | 52.58 | 50.37 | 54.50 |

**Table 3:** Accuracy values (%) for datasets with 200 features.

| | ADWINBaggingClassifier | | | AdaBoostClassifier | | | AdaptiveRandomForestClassifier | | | BaggingClassifier | | | LeveragingBaggingClassifier | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE |
| Hyperplane | 64.27 | 51.55 | 51.52 | 79.43 | 51.53 | 52.20 | 61.05 | 50.68 | 51.72 | 91.18 | 52.36 | 52.77 | 86.65 | 51.81 | 52.63 |
| RandomRBF | 85.15 | 53.52 | 56.68 | 99.93 | 78.54 | 83.44 | 99.33 | 93.72 | 94.12 | 99.88 | 57.42 | 59.24 | 99.93 | 69.93 | 67.63 |
| RandomRBFDrift | 81.51 | 53.83 | 56.33 | 99.93 | 77.47 | 84.85 | 99.23 | 93.49 | 93.90 | 99.75 | 62.85 | 61.63 | 99.96 | 63.57 | 62.50 |
| RandomTree | 51.49 | 52.82 | 50.74 | 63.56 | 59.30 | 50.27 | 58.50 | 57.79 | 50.14 | 54.04 | 50.05 | 61.22 | 70.35 | 69.31 | 52.82 |

**Table 4:** Accuracy values (%) for datasets with 500 features.

| | ADWINBaggingClassifier | | | AdaBoostClassifier | | | AdaptiveRandomForestClassifier | | | BaggingClassifier | | | LeveragingBaggingClassifier | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE |
| Hyperplane | 62.53 | 50.38 | 50.42 | 75.70 | 50.39 | 51.02 | 54.87 | 50.76 | 50.67 | 88.69 | 51.32 | 51.16 | 83.76 | 50.99 | 50.91 |
| RandomRBF | 89.44 | 55.13 | 53.28 | 100.00 | 79.48 | 85.03 | 99.85 | 95.40 | 93.15 | 99.99 | 57.93 | 61.09 | 99.99 | 59.03 | 59.96 |
| RandomRBFDrift | 85.89 | 53.71 | 56.73 | 100.00 | 81.04 | 86.46 | 99.62 | 95.98 | 92.84 | 99.98 | 60.04 | 59.76 | 99.99 | 61.54 | 58.02 |
| RandomTree | 62.18 | 50.56 | 49.99 | 51.55 | 50.14 | 50.03 | 55.80 | 85.09 | 50.66 | 51.23 | 54.35 | 52.80 | 51.85 | 64.24 | 49.76 |

**Table 5:** Accuracy values (%) for datasets with 1000 features.

| | ADWINBaggingClassifier | | | AdaBoostClassifier | | | AdaptiveRandomForestClassifier | | | BaggingClassifier | | | LeveragingBaggingClassifier | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE | No t-SNE | 2D t-SNE | 3D t-SNE |
| Hyperplane | 61.87 | 49.96 | 50.08 | 71.97 | 50.46 | 50.37 | 52.83 | 50.24 | 50.29 | 86.95 | 50.31 | 50.82 | 82.32 | 50.60 | 50.44 |
| RandomRBF | 95.71 | 53.12 | 55.63 | 100.00 | 79.89 | 84.83 | 99.79 | 96.35 | 94.16 | 99.99 | 60.08 | 60.91 | 100.00 | 59.45 | 58.23 |
| RandomRBFDrift | 94.15 | 54.10 | 57.30 | 100.00 | 82.28 | 88.96 | 99.87 | 96.13 | 94.36 | 100.00 | 60.71 | 66.09 | 100.00 | 60.59 | 56.36 |
| RandomTree | 52.42 | 56.83 | 57.49 | 50.79 | 50.24 | 75.77 | 50.44 | 50.88 | 54.61 | 59.09 | 55.21 | 58.93 | 51.40 | 50.30 | 57.00 |

## REFERENCES

[1] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. 2001. On the Surprising Behavior of Distance Metrics in High Dimensional Space. In *Database Theory — ICDT 2001*, Jan Van den Bussche and Victor Vianu (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 420–434.

[2] C. Alippi, G. Boracchi, and M. Roveri. 2009. Just in time classifiers: Managing the slow drift case. In *International Joint Conference on Neural Networks, 2009. IJCNN 2009*. 114–120. https://doi.org/10.1109/IJCNN.2009.5178799

[3] Jean Paul Barddal, Fabrício Enembreck, Heitor Murilo Gomes, Albert Bifet, and Bernhard Pfahringer. 2019. Boosting decision stumps for dynamic feature selection on data streams. *Inf. Syst.* 83 (2019), 13–29. https://doi.org/10.1016/j.is.2019.02.003

[4] Jean Paul Barddal, Fabrício Enembreck, Heitor Murilo Gomes, Albert Bifet, and Bernhard Pfahringer. 2019. Merit-guided dynamic feature selection filter for data streams. *Expert Systems with Applications* 116 (2019), 227–242. https://doi.org/10.1016/j.eswa.2018.09.031

[5] Jean Paul Barddal, Heitor Murilo Gomes, Fabrício Enembreck, and Bernhard Pfahringer. 2017. A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *Journal of Systems and Software* 127 (2017), 278–294. https://doi.org/10.1016/j.jss.2016.07.005

[6] Jean Paul Barddal, Heitor Murilo Gomes, Fabrício Enembreck, Bernhard Pfahringer, and Albert Bifet. 2016. On Dynamic Feature Weighting for Feature Drifting Data Streams, Vol. 9852. https://doi.org/10.1007/978-3-319-46227-1_9

[7] Albert Bifet and Ricard Gavaldà. 2007. Learning from Time-Changing Data with Adaptive Windowing. In *Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA*. SIAM, 443–448. https://doi.org/10.1137/1.9781611972771.42

[8] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. 2010. Leveraging Bagging for Evolving Data Streams. In *Machine Learning and Knowledge Discovery in Databases*, José Luis Balcázar, Francesco Bonchi, Aristides Gionis, and Michèle Sebag (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 135–150.

[9] Lifei Chen and Shengrui Wang. 2012. Automated Feature Weighting in Naive Bayes for High-dimensional Data Classification. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM '12)*. ACM, New York, NY, USA, 1243–1252. https://doi.org/10.1145/2396761.2398426

[10] Pedro Domingos and Geoff Hulten. 2000. Mining High-Speed Data Streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)*. Association for Computing Machinery, New York, NY, USA, 71–80. https://doi.org/10.1145/347090.347107

[11] Yoav Freund and Robert E Schapire. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Comput. System Sci.* 55, 1 (1997), 119–139. https://doi.org/10.1006/jcss.1997.1504

[12] Isvani Frías-Blanco, José del Campo-Ávila, Gonzalo Ramos-Jiménez, Rafael Morales-Bueno, Agustín Ortiz-Díaz, and Yailé Caballero-Mota. 2015. Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds. *IEEE Transactions on Knowledge and Data Engineering* 27, 3 (2015), 810–823. https://doi.org/10.1109/TKDE.2014.2345382

[13] Joao Gama. 2010. *Knowledge Discovery from Data Streams* (1st ed.). Chapman & Hall/CRC.

[14] João Gama, Pedro Medas, Gladys Castillo, and Pedro Rodrigues. 2004. Learning with Drift Detection. In *Advances in Artificial Intelligence – SBIA 2004*, Ana L. C. Bazzan and Sofiane Labidi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 286–295.

[15] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. 2013. On Evaluating Stream Learning Algorithms. *Mach. Learn.* 90, 3 (March 2013), 317–346. https://doi.org/10.1007/s10994-012-5320-9

[16] Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. 2017. Adaptive random forests for evolving data stream classification. *Machine Learning* 106, 9 (2017), 1469–1495.

[17] Geoff Hulten, Laurie Spencer, and Pedro Domingos. 2001. Mining Time-Changing Data Streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)*. Association for Computing Machinery, New York, NY, USA, 97–106. https://doi.org/10.1145/502512.502529

[18] Ian Jolliffe. 2005. Principal component analysis. *Encyclopedia of statistics in behavioral science* (2005).

[19] Ludmila I. Kuncheva and William J. Faithfull. 2014. PCA Feature Extraction for Change Detection in Multidimensional Unlabeled Data. *IEEE Transactions on Neural Networks and Learning Systems* 25, 1 (2014), 69–80. https://doi.org/10.1109/TNNLS.2013.2248094

[20] Yann LeCun, Corinna Cortes, and CJ Burges. 2010. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[21] G Maaten, L.; Hinton. 2008. Visualizing data using t-SNE. Journal of Machine Learning Research. Journal of Machine Learning Research, 2579–2605.

[22] L. Maaten. 2009. Learning a Parametric Embedding by Preserving Local Structure. International Conference on Artificial Intelligence and Statistics.

[23] Ioannis Mitliagkas, Constantine Caramanis, and Prateek Jain. 2013. Memory Limited, Streaming PCA. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'13)*. Curran Associates Inc., Red Hook, NY, USA, 2886–2894.

[24] Jacob Montiel, Max Halford, Saulo Martiello Mastelini, Geoffrey Bolmier, Raphael Sourty, Robin Vaysse, Adil Zouitine, Heitor Murilo Gomes, Jesse Read, Talel Abdessalem, and Albert Bifet. 2020. River: machine learning for streaming data in Python. (2020). arXiv:2012.04740

[25] Hai-Long Nguyen, Yew-Kwong Woon, and Wee-Keong Ng. 2014. A survey on data stream clustering and classification. *Knowledge and Information Systems* (2014), 1–35. https://doi.org/10.1007/s10115-014-0808-1

[26] Nikunj C. Oza and Stuart Russell. 2001. Experimental Comparisons of Online and Batch Versions of Bagging and Boosting. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '01)*. Association for Computing Machinery, New York, NY, USA, 359–364. https://doi.org/10.1145/502512.502565

[27] C. E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal* 27, 3 (1948), 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

[28] Blaz Sovdat. 2014. Updating formulas and algorithms for computing entropy and Gini index on time-changing data streams. *CoRR* abs/1403.6348 (2014). arXiv:1403.6348 http://arxiv.org/abs/1403.6348

[29] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9, 86 (2008), 2579–2605. http://jmlr.org/papers/v9/vandermaaten08a.html

[30] Geoffrey I. Webb, Loong Kuan Lee, Bart Goethals, and François Petitjean. 2018. Analyzing Concept Drift and Shift from Sample Data. *Data Min. Knowl. Discov.* 32, 5 (Sept. 2018), 1179–1199. https://doi.org/10.1007/s10618-018-0554-1

[31] Gerhard Widmer and Miroslav Kubat. 1996. Learning in the Presence of Concept Drift and Hidden Contexts. *Mach. Learn.* 23, 1 (April 1996), 69–101. https://doi.org/10.1023/A:1018046501280

[32] Ian H. Witten, Eibe Frank, Mark A. Hall, and Christopher J. Pal. 2016. *Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques* (4th ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[33] Lanqin Yuan, Bernhard Pfahringer, and Jean Paul Barddal. 2018. Iterative Subset Selection for Feature Drifting Data Streams. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC '18)*. Association for Computing Machinery, New York, NY, USA, 510–517. https://doi.org/10.1145/3167132.3167188