DeepStream: Autoencoder-Based Stream Temporal Clustering and Anomaly Detection

Shimon Harush, Yair Meidan, Asaf Shabtai

Department of Software and Information Systems Engineering
Ben-Gurion University of the Negev

Abstract

The increasing number of IoT devices in "smart" environments, such as homes, offices, and cities, produce seemingly endless data streams and drive many daily decisions. Consequently, there is growing interest in identifying contextual information from sensor data to facilitate the performance of various tasks, e.g., traffic management, cyber attack detection, and healthcare monitoring. The correct identification of contexts in data streams is helpful for many tasks, for example, it can assist in providing high-quality recommendations to end users and in reporting anomalous behavior based on the detection of unusual contexts. This paper presents DeepStream, a novel data stream temporal clustering algorithm that dynamically detects sequential and overlapping clusters. DeepStream is tuned to classify contextual information in real time and is capable of coping with a high-dimensional feature space. DeepStream utilizes stacked autoencoders to reduce the dimensionality of unbounded data streams and for cluster representation. This method detects contextual behavior and captures nonlinear relations of the input data, giving it an advantage over existing methods that rely on PCA. We evaluated DeepStream empirically using four sensor and IoT datasets and compared it to five state-of-the-art stream clustering algorithms. Our evaluation shows that DeepStream outperforms all of these algorithms. Our evaluation also demonstrates how DeepStream's im-

Email address: hshimo@post.bgu.ac.il, yairme@post.bgu.ac.il, shabtaia@bgu.ac.il (Shimon Harush, Yair Meidan, Asaf Shabtai)

proved clustering performance results in improved detection of anomalous data.
Keywords: Stream clustering; Autoencoder; Dimensionality reduction; Anomaly detection; Activity recognition

1. Introduction

Clustering is an important machine learning technique which has drawn much research attention in recent decades [1]. The principal task of clustering is to group items (entities) that share similar patterns into meaningful clusters within a given dataset. It is widely used in many data mining applications, such as document classification and categorization in the field of natural language processing [2, 3] and clustering images using deep learning in the field of computer vision [4, 5].

Data *stream* clustering is a special type of clustering that has recently received increased attention from researchers motivated by the challenges posed by applications involving large datasets and streams of data, and the need to learn tasks in real time [6]. Some examples of stream clustering applications include retrieving information from search engines which are constantly updated [7, 8] and monitoring IoT network traffic and sensor data [9].

Stream clustering algorithms are different from traditional clustering algorithms in several ways. First, since stream clustering algorithms are exposed to the data incrementally, they cannot iterate over all of the data or access different data locations. In addition, most of the data needs to be discarded over time because of memory limitations. Second, the data distribution may change over time and have concept drifts. For this reason, stream clustering models should adapt themselves to new data and have the ability to disregard older, no longer relevant data. Third, incoming data streams accumulate over time, and stream clustering models do not have the time or storage space to handle large amounts of data. Therefore, stream clustering algorithms should only save the minimal amount of data needed to retain their ability to continuously cluster new data efficiently.

Several stream clustering algorithms have been proposed in the past, including CluStream [10], DenStream [11], D-Stream [12], DBStream [13], and pcStream [14]. Most of these algorithms have two phases: an online phase, which summarizes the data into many micro-clusters, and an offline phase, in which these micro-clusters are re-clustered into a smaller number of final clusters [10]. The re-clustering phase is performed offline as needed and not in real time, since it is time-consuming and cannot cope with continuous streaming data. Although the abovementioned stream clustering algorithms are considered state-of-the-art, they typically have the following two limitations (see Section 2), namely their inherent inability to capture temporal relations or nonlinear relations, especially within high-dimensional input data.

To address the two limitations mentioned above, in this paper we introduce DeepStream, a novel stream clustering method which leverages deep learning techniques. As opposed to CluStream, DenStream, D-Stream (and others), DeepStream takes into account the order of the arriving data points when deciding, e.g., whether to construct or split clusters. By doing so, DeepStream is capable of capturing temporal relations. Although pcStream also has this capability, it uses principal component analysis (PCA) for dimensionality reduction, so it cannot capture nonlinear relations, and is also highly sensitive to the input's dimensionality. In contrast, DeepStream uses the embedded representation output of a pretrained stacked autoencoder (SAE) and is therefore able to better deal with high-dimensional data streams while capturing nonlinear relations among input features.

We evaluated our proposed method using four different datasets and compared its performance with that of five existing stream clustering algorithms. The results of our evaluation show significant improvement in clustering capabilities (using the adjusted Rand index metric) and better detection of point and contextual anomalies compared to the pcStream algorithm.

To conclude, the main contributions of our work are:

- We propose a stream clustering algorithm which is capable of capturing

temporal contexts. As a result of integrating SAEs, our method is able to (1) capture nonlinear relations between features, and (2) handle high-dimensional input vectors by using a smaller amount of input features.

- We empirically compare our algorithm with state-of-the-art algorithms. Our evaluation shows that our algorithm outperforms the other algorithms in clustering temporal context streams which are present in four datasets.
- We present techniques for finding two types of anomalies in the clustering process that show improvement over an existing comparative algorithm.
- Finally, we make our code and datasets publicly available.[1]

The remainder of this paper is constructed as follows: In Section 2, we review existing algorithms for stream clustering, as well as autoencoders and their use in clustering. Then, in Section 3, we describe *DeepStream*, our proposed method for stream clustering. Later, in Section 4, we describe our evaluation method, and in Section 5, we report our experimental results and provide a demonstration of how DeepStream can detect point anomalies and collective anomalies. Finally, in Section 6, we present the main conclusions of our research.

2. Background and Related Work

2.1. Stream Clustering

In the past few years there has been increasing interest in managing massive, unbounded sequences of data objects that are streamed at rapid rates (also referred to as data streams) [15, 16, 17]. Data stream applications include data generated from computer network traffic [18], smartphone sensors [19], IoT devices [9], stock markets [20], etc. The respective datasets of these applications may be too large to process efficiently or store in main or secondary memory devices. In this paper, we define a data stream S as a sequence of data $S =$

[1] This is a double-blind submission; the code will only be added upon paper publication.

$x^1; x^2; :::; x^m$, in which m may be unbounded. Each data object (x^i) is described by an N-dimensional feature vector.

Stream clustering aims to find and maintain a set of valid clusters within a perpetual stream of data records [21]. Stream clustering algorithms differ from ordinary clustering algorithms due to the need to summarize the data as it comes, in order to enable in-memory handling of vast data streams. In addition to this requirement, stream clustering algorithms are often faced with several challenges [17, 21], namely: *(i)* data objects arrive continuously, *(ii)* the data stream size may be unbounded, *(iii)* after being processed, data cannot be stored in the memory for a long period of time, and *(iv)* the data stream's probability distribution may change over time.

In this paper, we propose a novel stream clustering algorithm called DeepStream and empirically compare it with the following state-of-the-art stream clustering algorithms:

1. DenStream [11]: A density-based stream clustering algorithm, which uses the clustering feature (CF) form to determine whether a group of micro-clusters is a legitimate cluster or a collection of outliers

2. D-Stream [12]: Also performs density-based stream clustering, however across a grid

3. DBStream [13]: Captures the shared density between each two micro-clusters in order to decide whether or they belong to the same cluster

4. CluStream [10]: Takes a two-phased approach (dividing the clustering process into online and offline macro-clustering) which provides flexibility to explore the nature of the evolution of the clusters over different time periods

5. pcStream [14]: Summarizes clusters using the mean and principal components (vectors of highest variance) of the clusters' last records. Unlike the other methods evaluated, pcStream considers the temporal relation between arriving records when making clustering decisions.

Table 1: Capabilities of data stream clustering algorithms.

Algorithm	Data structure	Outlier detection	Cluster shape	Contextual data	Clustering measure	High-dimensional data
CluStream [10]	feature vector	statistical-based	spherical	-	distance-based	-
DenStream [11]	feature vector	density	arbitrary	-	distance-based	-
D-Stream [12]	grid	density	arbitrary	-	grid-based	-
DBStream [13]	feature vector	density	arbitrary	-	distance-based	-
pcStream [14]	feature vector	distance	arbitrary	+	distance-based	+ using lower dimension of features
DeepStream	feature vector	distance/ reconstruction error	arbitrary	+	distance-based	+ using lower dimension of features

There are other algorithms designed for stream clustering. One example is HPstream [22], which can process high-dimensional data streams. HPstream's drawbacks are related to its requirement of inputting the average clustering dimension, a value which is difficult to determine in practice. In addition, it does not solve the clustering problem of non-convex data.

Table 1 summarizes the capabilities of each of the algorithms evaluated. Of them, our algorithm works most similarly to pcStream in the way that decisions are made regarding cluster affiliation and the formation of new clusters. That is, none of the first four algorithms in Table 1 (i.e., CluStream, DenStream, D-Stream, and DBStream) are suitable for clustering *contextual* sensor streams. This is due to the fact that these four algorithms are designed to produce clusters that are agnostic to the order of arriving records; as a result, temporal relations cannot be detected. In contrast, DeepStream, like pcStream, is capable of capturing the temporal importance of the incoming data, as the arrival order of data points plays a crucial role in its clustering mechanism.

Although pcStream can detect temporal relations between data records, it uses PCA [23] and is therefore limited to the exploration of *linear* relations between features. To overcome this limitation, DeepStream leverages deep learning for feature extraction while performing dimensionality reduction at the same time, which enables the exploration of nonlinear relations. With the growing success of this approach, various clustering methods have already been proposed (see Subsection 2.3), however most of them were used for image clustering, and none are suitable for online stream clustering [5, 24, 25].

2.2. Autoencoders

An autoencoder (AE) is a type of artificial neural network used to learn efficient data encoding in an unsupervised manner. Traditionally, AEs were used for dimensionality reduction or feature learning. AEs take structural input data and try to reconstruct the input after encoding it. An AE is comprised of two fundamental components: an encoder and decoder. The *encoder* is a multi-layered neural network that outputs a smaller representation of the data it receives as input, which is also called the latent representation. This component is actually used by the AE to perform dimensionality reduction. The *decoder* reproduces the input that the encoder receives by using the latent representation received from the encoder's output. In most cases, the encoder and decoder have the same architecture although the decoder is built with reversed order of the encoder's layers. An AE generally aims at reconstructing the input in the output layer, and minimizing the reconstruction error in terms of the selected loss function. Fig. 1 schematically illustrates an AE with a hidden layer that is used by DeepStream as a compact representation of the raw data for clustering.

2.3. Autoencoder-Based Clustering

The output of the network's middle layer, also known as its latent representation, has various uses, one of which is clustering. Many (deep) AE variants for clustering have improved different aspects of the AE model. For instance, some of those variants focus on the network architecture, such as the number

of layers, latent dimension size, or even the use of CAE (convolutional AEs) for image data. Other methods are aimed at preventing overfitting and improving robustness by adding noise to the input [26] or by using dropout [27] to generalize the network. For example, DEC [5] uses a joint deep neural network (DNN) and clustering that simultaneously learns feature representations and cluster assignments with an SAE network. In [28], the authors introduced a graph clustering method based on a DNN, which uses a sparse AE as its building block. Dimensionality reduction using a deep belief network with nonparametric maximum margin clustering was proposed in [29]. In [4], the authors suggested a recurrent framework for joint unsupervised learning of deep representations and image clusters. In contrast to some of the abovementioned methods which combine the association of clusters with the DNN architecture, our method does not assume a certain number of clusters. In addition, to the best of our knowledge, the above methods have not yet been applied to temporal stream clustering.

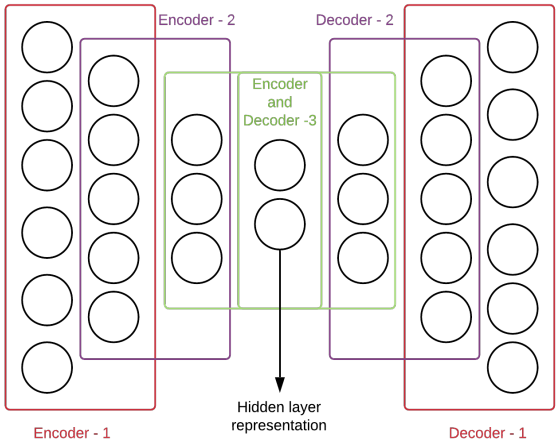


Figure 1: Example of the stacked autoencoder architecture.

3. Proposed Method

Our proposed method, called DeepStream, consists of an offline and online phase. In the offline phase a neural network learns the data in the training set in order to generate a model that produces a compact representation of each record. The online phase continuously generates clusters from incoming data streams using the model trained in the offline phase. In this section, we first introduce DeepStream's offline phase, which includes the dimensionality reduction method selected, as well as architecture and parameter selection. Then, we introduce DeepStream's online phase and describe the algorithm's streaming and clustering selection process.

3.1. DeepStream's O ine Phase

One of the key components of DeepStream consists of a SAE, which is a more advanced architecture than a basic AE. Research has shown that SAEs consistently produce semantically meaningful and well-separated representations on real-world datasets [30, 31]. Thus, the unsupervised representation learned by an SAE represents the nonlinear relationship between the input variables and naturally helps DeepStream better separate between different contexts.

A SAE network is built layer by layer, and each layer is a denoising AE which is trained separately to reconstruct the previous layer's output [31]. Each denoising AE is made up of two layers.

$$\tilde{x} \quad Dropout(x) \tag{1}$$

$$h = g_1(W_1\tilde{x} + b_1) \tag{2}$$

$$\tilde{h} \quad Dropout(h) \tag{3}$$

$$\hat{x} = g_2(W_2\tilde{h} + b_2) \tag{4}$$

In Equation (1), input x passes through $Dropout()$; dropout is a regularization technique which prevents overfitting by dropping out units during the training [27]. h is the hidden representation result of the g_1 activation function, and W and b are neural network parameters. Hidden layer h adds L2 normalization to the latent space during training; in [24], the authors showed that adding L2 normalization to the latent space improves separability of clusters in a variety of deep autoencoder models.

Equations (3) and (4) are the decoder layers that attempt to reproduce input x. In the training phase, the mean squared error ($MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i \quad \hat{x}_i)^2$) function is minimized. We use the output of h as the input for training the next autoencoder.

Each internal layer uses ReLU as an activation function, because it is simple and computationally inexpensive, and it does not suffer from the vanishing gradient problem like sigmoid does. However, ReLU suffers from a different problem known as the "dying ReLU" problem [32]. With the ReLU function, the output is zero for all of the negative inputs, and this causes many neurons to go into a 'death' state in which none of the weights is updated; this means that subsequent iterations will have the same output. In order to lower the chances of "dying ReLU," we use SELU as the internal layer activation function in some of the datasets when evaluating DeepStream's performance. After training each layer separately, we build an AE model from all of the hidden layers (h), as shown in Fig. 1, where the layers of the encoder and decoder are built the same way but in reverse order.

Next, the SAE undergoes additional training in its entirety to fine-tune the parameters simultaneously. This results in a trained deep AE with the capability of producing a reduced representation of the input; the output of the encoder (i.e., the SAE's hidden layer) is then used in the online clustering phase.

3.2. DeepStream's Online Clustering Phase

DeepStream's online clustering phase is based on its predecessor, the pcStream algorithm [14], except for the way in which the input is manipulated.

While pcStream relies on the soft independent modeling by class analogy method (SIMCA) [33] for similarity score calculation, DeepStream uses a SAE to encode high-dimensional input to lower dimensions in order to obtain semantically meaningful context separation by leveraging the nonlinear relationship between the input features.

DeepStream's clustering phase (Algorithm 1) has six different hyperparameters: threshold (t) represents the level of sensitivity of the algorithm; drift buffer size (d) sets the minimum size of each context; memory size (m) limits the maximum size of each context; ($encoder$) stands for a pretrained encoder that encodes each input from stream X; and context limit (lim) limits the size of the algorithm's contexts.

As part of Algorithm 1's initiation phase (lines 1-4), a list of contexts C is initialized with a context that contains the first d instances from stream X, by calling the $updateContext(X[0::d])$ function. It is assumed that the first d records belong to the same cluster to provide the algorithm a starting point. Then, each record x of the data stream beyond the first d records is processed as follows (lines 5 and on): First, x is encoded to a lower-dimensional representation by the $encode(x; encoder)$ function (line 6). Subsequently, the Mahalanobis distance [34] is measured on $\tilde{x}$ from each context in list C. We chose to use the Mahalanobis distance instead of the Euclidean distance [35], since we cannot assume that the clusters have identical co-variances. Clusters with elliptical shaped co-variances are modeled much better using the Mahalanobis distance measure. In rare cases when features have zero correlations, the Euclidean and Mahalanobis distances are the same, except that the Euclidean distance performs faster [34]. After measuring all of the distances between $\tilde{x}$ and the cluster centers, the closest cluster index is chosen (line 7). In cases in which the closest cluster distance is less than a threshold t (lines 8-11), the attempt to create a new cluster breaks down, and therefore if the drift buffer (denoted as $Buffer$) is not empty, the records are dumped into the $C[bestContext]$ cluster in function $updateContext()$; and the current instance $\tilde{x}$ joins the $C[bestContext]$ cluster. In order to save memory, function $updateContext(C[bestContext]; \tilde{x}; m)$

Algorithm 1: DeepStream online clustering algorithm.

input : threshold (t), drift buffer size (d), memory size (m), trained encoder ($encoder$), data stream (X), context limit (lim)

output: Context list (C)

```
1  C ← []
2  append(updateContext(X[0::d]); C)
3  Buffer ← []
4  X ← X[d::]
5  for x ∈ X do
6      x̃ ← encode(x; encoder)
7      bestContext ← argmin^dist_{c∈C}(x̃; c)
8      if dist(x̃; bestContext) < t then
9          updateContext(C[bestContext]; Buffer; m)
10         Buffer ← []
11         updateContext(C[bestContext]; x̃; m)
       else
12         append(x̃; Buffer)
13         if len(Buffer) = d then
14             append(updateContext(Buffer); C)
15             if len(C) = lim then
16                 C ← merge(C)
               end
           end
       end
   end
```

updates the cluster $C[bestContext]$ with the m newest incoming records. In addition, the use of the FIFO policy allows clusters to adapt themselves to changes over time (i.e., concept drift). Then, the inverse covariance matrix is calculated for the Mahalanobis distance measurement. In cases in which the closest con-

text distance is greater than threshold (t) (lines 12 and on), $\tilde{x}$ is appended to the drift buffer ($Buffer$), and if the maximum number (d) of instances in the drift buffer has been reached, a new context is created with those instances.

We note that in some cases, especially in real-time applications, stream clustering systems are designed to handle only a limited number of clusters. To this end, an overabundance of clusters (which we refer to as an 'explosion of contexts') could result from, e.g., a gradual concept drift that ends up creating redundant clusters for the same context. This phenomenon might result in *(1)* reduced clustering quality over time, and/or *(2)* increased computational costs, e.g., calculating the Mahalanobis distance between each arriving instance and every existing cluster. In accordance, as part of the DeepStream algorithm, we use the parameter lim to limit the number of possible contexts. This is done by using the $merge(C)$ function (see Algorithm 2) to merge the two closest contexts. The first part of the $merge(C)$ function includes identifying the two closest contexts from context list C and removing them from C (lines 1-2). Then, the merging operation is carried out simply by taking the newest m instances from the two merging contexts (lines 5-9). Each instance of DeepStream must be saved with its arrival timestamp in order to enable accurate merging.

4. Evaluation Method

4.1. Datasets

In order to evaluate the performance and generality of our proposed algorithm, we tested DeepStream on four real-world contextual sensor datasets, each of which captures a different context domain:

1. **HAR:** The human activity recognition (HAR) dataset is based on the recording of group of 30 volunteers ranging from 19 to 48 years old. Each person performed six activities (walking, walking upstairs, walking downstairs, sitting, standing, laying down) [36]. The dataset contains 561 features generated from the accelerometer and gyroscope sensors. Each

Algorithm 2: Merging the closest pair of contexts

input : Context list (C), memory size (m)

output: Context list (C)

```
c1; c2   argmin^C_{i;j=0}  dist(C[i]; C[j])
Remove([c1; c2]; C)
idx1; idx2   0
mergedContext   ;
while len(c1) < idx1 and len(c2) > idx2 do
    if c1[idx1] < c2[idx2] then
        append(c1[idx1 + +]; mergedContext)
    else
        append(c2[idx2 + +]; mergedContext)
    end
end
return append(mergedContext[0 : m]; C)
```

record of those features was generated from a 128 window size samples. From each window, a vector of features was obtained by calculating variables from the time and frequency domain.

2. **HearO:** The HearO dataset is a sensor-fused dataset recorded from smartphones. This dataset has explicit context labels provided by each smartphone user at various times over several months. From the HearO dataset we used five different features in our experiment and three different levels of contexts: high-level contexts, low-level contexts, and a plugged-in label which indicates if a smartphone is connected to a charger or not.

3. **IoT:** The IoT dataset consists of a set of communication requests from IoT components to their controlling units (servers) [37]. The information was recorded at Ben-Gurion University of the Negev's and Singapore University of Technology and Design. The dataset is labeled based on the type

of IoT device, with 18 different types of IoT devices, including security cameras, a smart refrigerator, baby monitors, motion sensors, smoke detectors, sockets, etc. The dataset also contains 273 different features from the communication requests made to the server. Since the IoT dataset is divided into multiple subsets, each corresponding to a specific label (i.e., IoT device type), and does not take the form of natural temporal streams with importance placed on the order of arrival, we synthetically reconstructed the dataset such that it would include sequences of varying lengths (between 12 and 80 communication sessions in each sequence), each consisting of the same device type label.

4. **PAMAP2:** The Physical Activity Monitoring (PAMAP2) dataset contains data associated with 18 different physical activities (such as walking, cycling, and playing soccer) [38]. We created a temporal dataset that represents different activities. For this purpose, we randomly chose to test 'subject 101' and selected a subset of 12,000 records with varying lengths for each activity.

In Table 2, we present a summary of the above datasets that we used to empirically evaluate and compare the stream clustering algorithms.

4.2. Benchmarking

In our experiments, we compared DeepStream with five state-of-the-art stream clustering algorithms: pcStream [14], DBstream [13], DenStream [11], D-Stream [12] and CluStream [10]. To enable a fair comparison of the algorithms, each of them was trained on exactly the same training set and was later applied to the same test set. For each algorithm evaluated, we performed a grid search for hyperparameter tuning in order to optimize the algorithm's performance.

In order to evaluate D-Stream, DenStream, DBStream, and CluStream we used an R package called streamMOA [39], which implements all of these algorithms. In streamMOA, the algorithms classify records as noise if they do not fit into any cluster. In order to perform a fair comparison with pcStream

Table 2: Description of the datasets used in our study.

Dataset	# of features	Examples of features	# of rows in test set	Context groups	# of labels	Examples of labels
HAR	561	Accl. (x,y,z), FFT...	349	Human activity recognition	6	sitting, walking, going upstairs
HearO	5	Accl cor. (xy,xz,yz), temperature, battery level	1257	High-level	4	at home, at work, on a break
				Low-level	9	hungry, interested, shopping
				Battery plugged in	2	yes, no
IoT	237	Packet size, bytes, http time..	1024	IoT security camera models	7	Withings, Foscam
PAMAP2	51	IMU hand, IMU chest, IMU ankle	2442	Human activity monitoring	12	Sitting, standing, rope jumping, descending stairs...

and DeepStream, we only took into account results with less than 5% of noisy records.

When empirically evaluating DeepStream and comparing it to the abovementioned stream clustering algorithms, we placed special emphasis on its comparison to pcStream. The reason for this is that, as noted earlier, of the evaluated algorithms, DeepStream is most closely related to pcStream. Both of these algorithms consider the temporal relation between arriving records when making clustering decisions, however they use different information extraction methods: PCA is used in pcStream, and SAEs are used in DeepStream. To align our work with the original paper presenting pcStream and provide a better comparison to that paper, we used the HAR and HearO datasets with similar configurations for partitioning the data into training and test sets. Moreover, we used the publicly available code[2] released by the author to report the performance of pcStream.

[2]https://github.com/ymirsky/pcStream

4.3. Parameter Tuning

In DeepStream, the parameter tuning process is divided into two parts: (1) the offline training phase in which the AE is trained to encode the input data, and (2) the online phase in which the DeepStream model is trained to cluster the stream data in the training set. For the offline phase, we trained SAEs in order to use them later by the encoder for dimensionality reduction of the input stream. For each of the evaluation datasets summarized in Table 2, we trained an SAE whose architecture was derived from the input size of each dataset. In practice, the DeepStream architecture (specifically, the number of nodes in each layer of the SAE) selected for the IoT dataset is [d, 260, 100, 30, 100, 260, d], where d is a 237-feature input size. For the HearO dataset the architecture selected is [d, 10, 5, 2, 5, 10, d] with an input size of five features. For the HAR dataset the architecture is [d, 220, 100, 30, 20, 30, 100, 220, d] for a 561-feature input size. For the PAMAP2 dataset the architecture selected is [d, 34, 18, 10, 18, 34, d] with an input size of 51 features.

All of the internal layers are activated by the ReLU nonlinearity function [40] except for the IoT dataset which uses SELU (scaled exponential linear unit) [41] as the internal activation function. The reason for this is that the IoT dataset suffered from the 'dying ReLU' phenomenon, which caused a large number of neurons in the network to 'die' [32]. SELU can automatically force the activation towards a zero mean and unit variance for better convergence but requires using the *lecun_normal* as the weight initialization and *AlphaDropout* if dropouts are needed. The optimizer of all of the networks was set to SGD with a learning rate of 0.1 and momentum of 0.9.

For the online phase, we used the encoder trained in the previous offline phase and executed DeepStream with grid search on two parameters: threshold (t) and drift buffer size (d). The memory size (m) was set to 1,000 as well as the context limit.

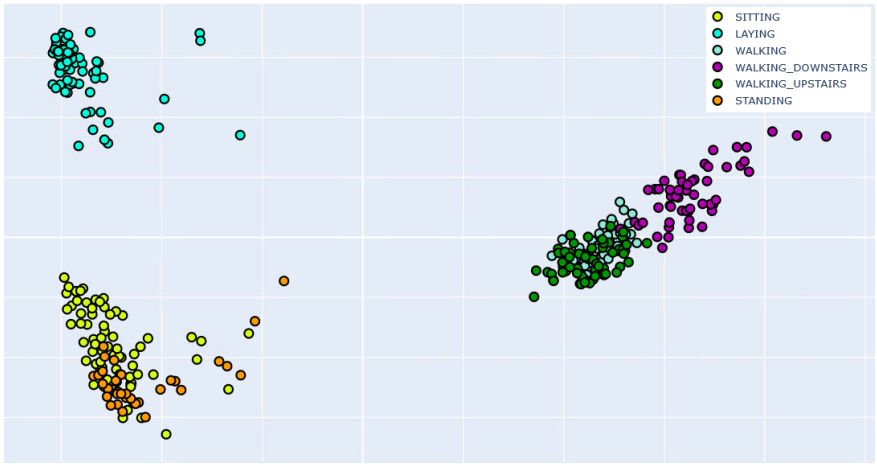
4.4. Evaluation Metric

In many stream clustering experiments the adjusted Rand index (ARI) and sum of squares (SSQ) are the most common evaluation metrics. However, the SSQ metric assumes that clusters do not overlap, and therefore it is not suitable for our clustering problem. Hence, all of the clustering methods were evaluated in our experiments using the ARI metric [42] which is widely used to compare the quality of different partitions given the ground truth. The ARI is a measure of similarity between two data clustering assignments regardless of their spatial qualities. ARI values range from -1 to 1, where the closer to 1 the index is, the better the agreement. A value of 0 indicates that the cluster distribution agreement is strictly coincidental. Negative ARI values indicate that the structure found in the cartographic variables is not aligned with the ground truth. Although DeepStream is designed to work without any labeling, we decided to use labels in our experiments in order to test the clustering quality on the test set using the ARI measure. In practice, none of the labels of the training records are necessary for model training.

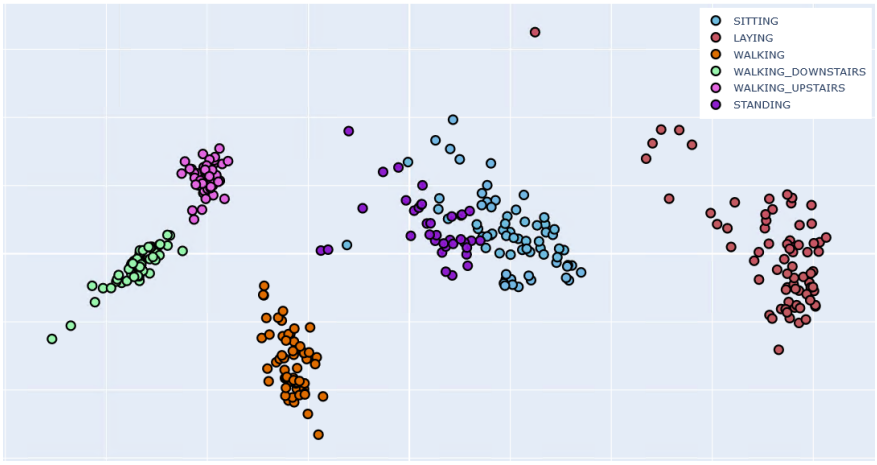
5. Experimental Results

5.1. Clustering Quality

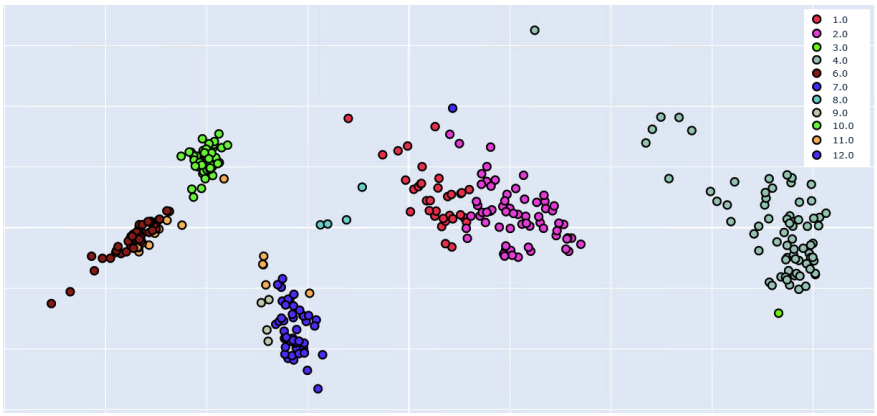
In this subsection, we report the best results (in terms of the ARI) attained by using grid search for each of the algorithms evaluated. The results are presented in Table 3, where it is evident that DeepStream outperforms all of the other methods when applied to any dataset. Since we only report the results with less than 5% of noisy records, D-Stream lacks results on the HAR, IoT, and PAMAP2 datasets. All algorithms obtained low ARI scores on the HearO-Low-High and PAMAP2 datasets (since these two datasets consist of real-life data which makes them noisy and hard to cluster). Niklas *et al.* [43] showed that avoiding some of the unclear labels from the PAMAP2 dataset significantly improves clustering performance in terms of the ARI metric.



(a) True labels of the test set data.



(b) True labels of the test set AE's embeddings.



(c) AE embeddings' DeepStream cluster assignments.

Figure 2: The division into clusters performed by the AE on the Human Activity Recognition (HAR) dataset. PCA was applied to plot this data in 2D.

Table 3: ARI values for the datasets evaluated and stream clustering algorithms; the HearO dataset was tested several times with different levels of label granularity.

	DeepStream	pcStream	CluStream	DBstream	DenStream	D-Stream
HAR	**0.867**	0.622	0.434	0.560	0.460	-
HearO – High-level	**0.283**	0.250	0.137	0.089	0.089	0.180
HearO – Low-level	**0.389**	0.138	0.174	0.089	0.089	0.180
HearO - Plugged in	**0.510**	0.115	0.104	0.039	0.184	0.003
IoT - Security camera	**0.726**	0.551	0.537	0.497	0.572	-
PAMAP2	**0.420**	0.362	0.177	0.179	0.215	-

Fig. 2 demonstrates DeepStream's effectiveness on the HAR dataset. It illustrates the distribution space (after PCA analysis was performed to enable 2D visualization) for three different configurations, as follows: (1) the original data distribution colored by the true labels, (2) the AE's embeddings colored by the true labels, and (3) clustering assignment performed on the trained model without any prior knowledge of the true labels. The three corresponding plots in Fig. 2 show clearly that the AE significantly improves the division of activities into separate groups using the HAR dataset.

On the HearO dataset, the results (see Table 3) show that the higher the granularity of the labels, the more sensitive the algorithm needs to be. Accordingly, the values for the selected drift size (d) and threshold (t) parameters were lower for the 'low-level' labels compared to the 'high-level' labels.

Finally, our evaluation supports the empirical evidence reported by Mirsky *et al.* [14], as pcStream outperforms most of the state-of-the-art stream clustering methods evaluated (except DeepStream) (Table 1) on a wide range of temporal context datasets.

5.2. Runtime Analysis

In this subsection, we describe the runtime analysis conducted, in which we compared DeepStream[3] with D-Stream, DenStream, CluStream, DBStream,

[3]DeepStream: Keras implementation with no quantization, executed in pyCharm IDE

and pcStream[4] in terms of online instance processing time. For this analysis we used two datasets: (1) PAMAP2, which is relatively low-dimensional, and (2) HAR, which is relatively high-dimensional.

Each algorithm was evaluated using the best parameter configuration found, as described in the previous section. This experiment was performed using a single logical core on an 1.6 GHz Intel i5 processor with 8GB of RAM.

Table 4 contains the evaluation results. It is evident that DeepStream processes the data during clustering significantly faster than its predecessor, pcStream. We expect this ratio to increase as the data dimentionality increases, since pcStream applies PCA which is highly sensitive to data dimensionality. Indeed we see a 6.9 runtime ratio (*pcStream=DeepStream*) on the PAMAP2 dataset versus a ratio of 11.5 on the HAR dataset.

As expected, both DeepStream and pcStream generally performed slower than the other algorithms. The main reason for this is that both methods use dimensionality reduction techniques that are computationally heavy as part of the clustering process. This time can be reduced by applying model quantization and utilizing a GPU machine. Also note that setting a higher threshold parameter (t) with DeepStream and pcStream will result in a shorter training time. This is because a higher t results in less insertions into the drift buffer (d), thereby resulting in less cluster creation, a relatively expensive operation.

5.3. Point Anomaly Detection Using DeepStream

In addition to DeepStream's ability to cluster temporal data streams (demonstrated in Table 3), it can also be used to detect point anomalies in data streams, using two different techniques:

1. **Distance measure.** With this technique, a record is considered anomalous if its distance from each of the existing clusters (created by DeepStream based on benign data) exceeds a certain threshold.

[4] All of the other algorithms: R open-source implementation, executed in R-Studio IDE

Table 4: Online runtime evaluation.

	Algorithm	Total processing time (sec)	Average instance processing time (ms)
PAMAP2 9970 instances 51 features	DeepStream	10.172	1
	pcStream	67.443	6.9
	CluStream	1.82	0.18
	DBStream	1.1	0.11
	D-Stream	-	-
	DenStream	23.142	2.35

	Algorithm	Total processing time (sec)	Average instance processing time (ms)
HAR 9949 instances 561 features	DeepStream	178	17.9
	pcStream	2052	206
	CluStream	32.132	3.21
	DBStream	19.997	2.00
	D-Stream	-	-
	DenStream	18.9	1.90

2. **AE reconstruction error.** This technique relies on the reconstruction error of the pretrained AE used for dimensionality reduction. That is, a high reconstruction error of an input vector increases the likelihood that this vector is unusual, because the AE does not recognize it and cannot reconstruct it properly. In such cases, DeepStream classifies this record as anomalous.

We tested DeepStream's anomaly detection capabilities using two datasets:

1. **KDD'99** [44]. A subset of 50,000 records from the original dataset which was introduced in the Third International Knowledge Discovery and Data Mining Tools Competition. The task in this competition was to train a network intrusion detector, which is a predictive model capable of distinguishing between 'good' (normal) connections and 'bad' connections (abnormalities/intrusions/attacks). The 'bad' connections represent a wide variety of intrusions simulated in a military network environment. Based on the KDD'99 dataset, numerous algorithms have been proposed for intrusion detection, many of which relied on anomaly-based approaches

(e.g., [45] and [46]). In our experiment, we labeled the 'bad' connections as anomalies and used the 38 features as predictors. We partitioned the dataset such that 80% was allocated for training and the remaining 20% was allocated for testing. We trained DeepStream with only benign data and tested it using 10,000 records, 22.3% of which were labeled as malicious. The architecture of DeepStream's SAE was [38, 34, 18 ,10] where '10' represents the latent dimension.

2. **CICIDS'2017** [47]. Like the KDD'99 dataset, the CICIDS'2017 dataset is used for intrusion detection, however it was collected much more recently. The CICIDS'2017 dataset consists of eight different files containing five days of normal and attack traffic data captured by the Canadian Institute of Cybersecurity. In this dataset there are several types of attacks, including brute-force, DoS, Heartbleed, infiltration, botnet, web, and DDoS attacks. This dataset comprises 80 network flow features from the captured network traffic. The dataset is labeled based on the timestamp, source and destination ports and IPs, and attacks. To evaluate point anomaly detection we used 78 traffic features from the CICIDS'2017 dataset. The traffic data was selected from a specific day that includes both benign and anomalous data. For training we only used benign data, while the test data consists of both attack and benign data. The architecture of DeepStream's SAE was [78, 60, 40 ,30], where '30' represents the latent dimension.

Unlike the other algorithms we used in our evaluation, pcStream can detect several types of anomalies, hence we used it to compare with DeepStream's performance. pcStream was tested in exactly the same way it was presented in its GitHub repository [14]. The metrics we used to compare pcStream and DeepStream are the area under the receiver operating characteristic curve (AUC-ROC) and the area under the precision/recall curve (AUPRC), which further emphasizes [48] the correctness of the top ranked instances (most anomalous traffic records in our evaluation).

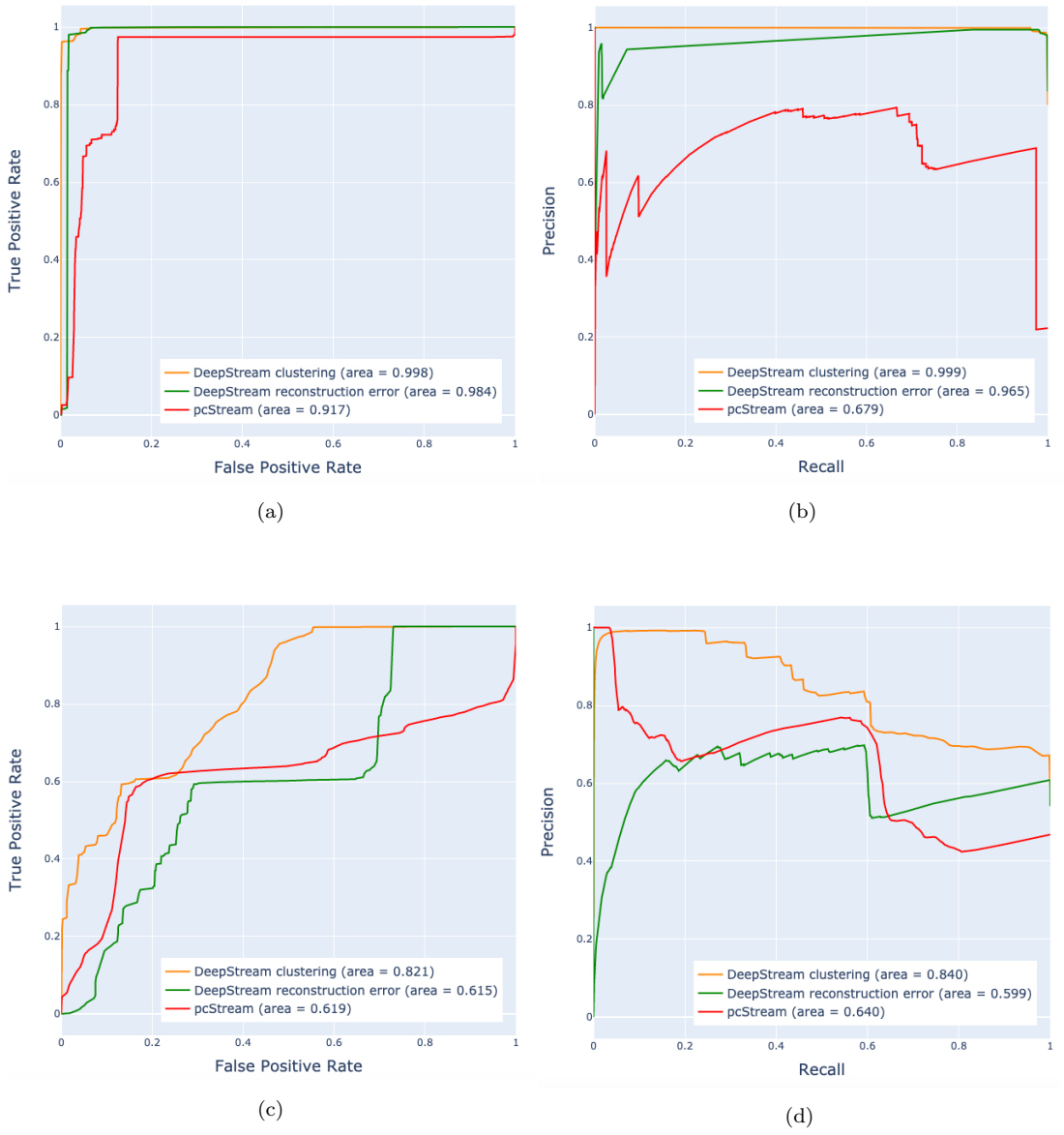


Figure 3: Comparison of the ability to detect point anomalies between DeepStream and pcStream, using the KDD'99 dataset (a and b) and the CICIDS'2017 dataset (c and d).

The results of our comparison of the methods' ability to detect point anomalies are portrayed in Fig. 3. With the KDD'99 dataset, pcStream obtained a

lower ROC-AUC (0.917) than DeepStream for each of the anomaly detection techniques used, as DeepStream obtained an ROC-AUC of 0.998 when using the first anomaly detection technique, i.e., when records are considered anomalous on the basis of their distance from clusters, and an ROC-AUC of 0.984 when using the second technique, which is based on the AE's reconstruction error (see Fig. 3a). pcStream also under-performed compared to DeepStream in terms of the AUPRC (see Fig. 3b): Area of 0.640 under the Precision-Recall curve for pcStream, and areas of 0.840 and 0.599 for DeepStream using clustering and the reconstruction error, respectively. With the CICIDS'2017 dataset, both the methods' results for the ROC-AUC (Fig. 3c) and AUPRC (Fig. 3d) metrics demonstrated the advantage of DeepStream, when using clustering, over pcStream, which in turn performed better than DeepStream, when using the reconstruction error.

5.4. Collective Anomaly Detection Using DeepStream

In addition to the ability to detect point anomalies, DeepStream is capable of detecting collective anomalous behavior. Collective anomalies are a set of related observations which together are anomalous with respect to the dataset but are not necessarily anomalous individually. In order to create such anomalies, we used the HAR dataset and injected collective anomalies into the test set. Injected collective anomalies are sequences of activities that do not make sense and hence do not frequently appear in the data, i.e., the activity 'walking_downstairs' or 'walking_upstairs' cannot appear right after 'sitting.' Each of these activities are not anomalous individually, but together they are considered a collective anomaly. To detect the collective anomalies, an LSTM AE was trained on the clustering results of DeepStream; the input consists of a long sequence of numbered clusters that DeepStream clustered. This sequence was converted to one-hot encoding input in order to nullify the numerical value of each cluster number. The LSTM AE model was built to reconstruct cluster sequences. The model consists of five layers of LSTM units [d, 32, 16, 13, 16, 32, d] with ReLU as the activation function and the Adam optimizer. The input data

was converted to the one-hot encoding format, and we found that an input data window of 13 cluster assignments worked best in our evaluation. The model trained for 400 epochs with early stopping on the validation set; a batch size of 512 was used to avoid overfitting. In order to evaluate collective anomalies, the HAR dataset was split into training and test sets exactly the same as it was split for clustering performance evaluation, as mentioned in Table 2, except the size of the test set which was increased due to the anomalous injections. The comparison was performed against pcStream which is currently the only stream clustering algorithm with the ability to detect collective anomalies. The pcStream algorithm detects collective anomalies by using a first order Markov chain (a detailed explanation can be found in [19]). In order to assess the effect of the LSTM AE, DeepStream was evaluated using three different variations: (1) using a first order Markov chain on DeepStream results, (2) using an LSTM AE model, and (3) by smoothing the LSTM model errors using a moving average.

As can be seen in the reconstruction error graph presented in Fig. 4, the LSTM AE model was able to detect all of the anomalous transitions but not necessarily the entire length of the anomalous sequences. In Fig. 5, DeepStream with the LSTM model shows significant improvement over both pcStream and DeepStream with the Markov chain module, obtaining an ROC-AUC of 0.819 compared with pcStream's AUC score of 0.635.

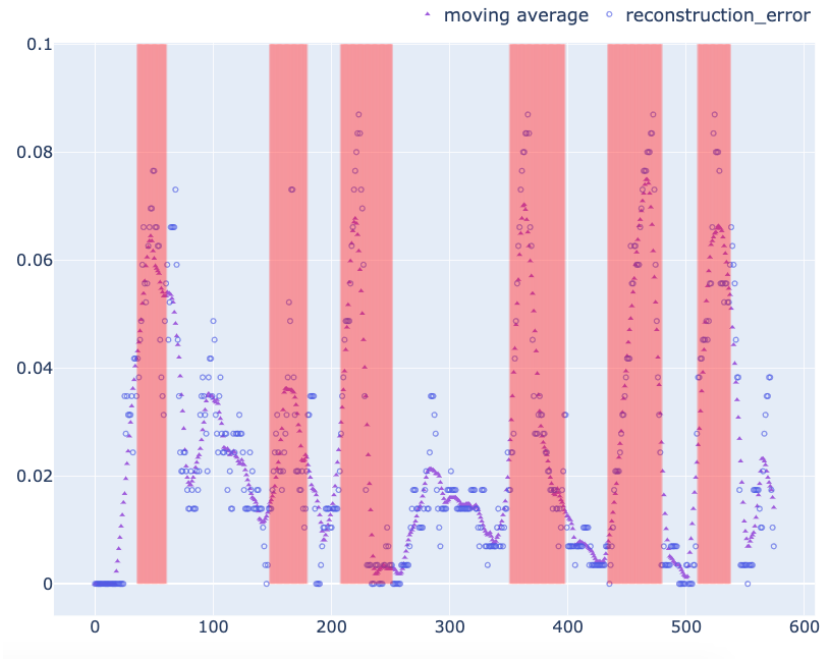


Figure 4: Reconstruction error graph produced by the LSTM AE. Vertical red lines represent anomalous transitions of activities.

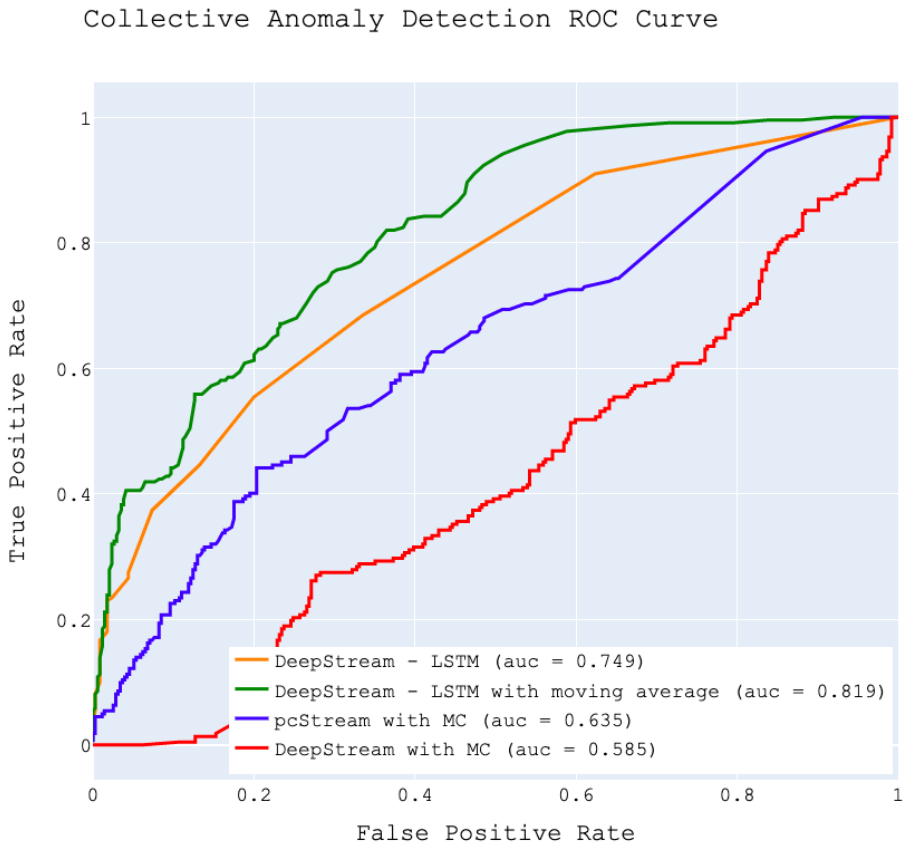


Figure 5: Collective anomaly detection using the Human Activity Recognition dataset - ROC curves and AUC results.

6. Conclusion

In this paper, we presented DeepStream, a stream clustering algorithm that uses deep learning to detect temporal contexts on unbounded temporal data streams. Among its strengths, we note that DeepStream (and particularly the AEs it relies on) does not require any labeled data for training. In addition, DeepStream leverages the capability of AEs to model complex nonlinear functions, compress a high-dimensional feature space into a low-dimensional latent space, and improve stream clustering performance on high-dimensional datasets.

Moreover, we showed that DeepStream can detect point anomalies by using the AE's reconstruction error and an LSTM AE for collective anomaly detection. Both methods outperforms the other stream clustering algorithms evaluated. Our empirical evaluation showed that DeepStream performs better than state-of-the-art stream clustering algorithms, particularly on high-dimensional sensor and IoT temporal datasets.

In future work, DeepStream can be adapted to contextual anomaly detection. Furthermore, DeepStream's AE can be continuously trained with online learning methods, thus addressing potential concept drifts.

7. Acknowledgments

This project received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 830927.

References

[1] S. Ding, F. Wu, J. Qian, H. Jia, F. Jin, Research on data stream clustering algorithms, Artificial Intelligence Review 43 (4) (2015) 593–600.

[2] L. D. Baker, A. K. McCallum, Distributional clustering of words for text classification, in: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, 1998, pp. 96–103.

[3] R. Bekkerman, R. El-Yaniv, N. Tishby, Y. Winter, On feature distributional clustering for text categorization, in: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, 2001, pp. 146–153.

[4] J. Yang, D. Parikh, D. Batra, Joint unsupervised learning of deep representations and image clusters, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 5147–5156.

[5] J. Xie, R. Girshick, A. Farhadi, Unsupervised deep embedding for clustering analysis, in: International conference on machine learning, 2016, pp. 478–487.

[6] C. C. Aggarwal, Data streams: models and algorithms, Vol. 31, Springer Science & Business Media, 2007.

[7] S. Chien, N. Immorlica, Semantic similarity between search engine queries using temporal correlation, in: Proceedings of the 14th international conference on World Wide Web, 2005, pp. 2–11.

[8] H.-J. Zeng, Q.-C. He, Z. Chen, W.-Y. Ma, J. Ma, Learning to cluster web search results, in: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, 2004, pp. 210–217.

[9] D. Puschmann, P. Barnaghi, R. Tafazolli, Adaptive clustering for dynamic iot data streams, IEEE Internet of Things Journal 4 (1) (2016) 64–74.

[10] C. C. Aggarwal, J. Han, J. Wang, P. S. Yu, A framework for clustering evolving data streams, in: Proceedings of the 29th international conference on Very large data bases-Volume 29, VLDB Endowment, 2003, pp. 81–92.

[11] F. Cao, M. Estert, W. Qian, A. Zhou, Density-based clustering over an evolving data stream with noise, in: Proceedings of the 2006 SIAM international conference on data mining, SIAM, 2006, pp. 328–339.

[12] Y. Chen, L. Tu, Density-based clustering for real-time stream data, in: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2007, pp. 133–142.

[13] M. Hahsler, M. Bolaños, Clustering data streams based on shared density between micro-clusters, IEEE Transactions on Knowledge and Data Engineering 28 (6) (2016) 1449–1461. doi:10.1109/TKDE.2016.2522412.

[14] Y. Mirsky, B. Shapira, L. Rokach, Y. Elovici, pcstream: A stream clustering algorithm for dynamically detecting and managing temporal contexts, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2015, pp. 119–133.

[15] J. Gama, M. M. Gaber, Learning from data streams: processing techniques in sensor networks, Springer, 2007.

[16] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. De Carvalho, J. Gama, Data stream clustering: A survey, ACM Computing Surveys (CSUR) 46 (1) (2013) 13.

[17] C. C. Aggarwal, A survey of stream clustering algorithms, in: Data Clustering, Chapman and Hall/CRC, 2018, pp. 231–258.

[18] H.-L. Nguyen, Y.-K. Woon, W.-K. Ng, A survey on data stream clustering and classification, Knowledge and information systems 45 (3) (2015) 535–569.

[19] Y. Mirsky, A. Shabtai, B. Shapira, Y. Elovici, L. Rokach, Anomaly detection for smartphone data streams, Pervasive and Mobile Computing 35 (2017) 83–107.

[20] M. Kontaki, A. N. Papadopoulos, Y. Manolopoulos, Continuous trend-based clustering in data streams, in: International Conference on Data Warehousing and Knowledge Discovery, Springer, 2008, pp. 251–262.

[21] M. R. Henzinger, P. Raghavan, S. Rajagopalan, Computing on data streams., External memory algorithms 50 (1998) 107–118.

[22] C. C. Aggarwal, J. Han, J. Wang, S. Y. Philip, On high dimensional projected clustering of data streams, Data Mining and Knowledge Discovery 10 (3) (2005) 251–273.

[23] K. Pearson, Liii. on lines and planes of closest fit to systems of points in space, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 2 (11) (1901) 559–572.

[24] C. Aytekin, X. Ni, F. Cricri, E. Aksu, Clustering and unsupervised anomaly detection with l 2 normalized deep auto-encoder representations, in: 2018 International Joint Conference on Neural Networks (IJCNN), IEEE, 2018, pp. 1–6.

[25] P. Huang, Y. Huang, W. Wang, L. Wang, Deep embedding network for clustering, in: 2014 22nd International conference on pattern recognition, IEEE, 2014, pp. 1532–1537.

[26] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: Proceedings of the 25th international conference on Machine learning, 2008, pp. 1096–1103.

[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, The journal of machine learning research 15 (1) (2014) 1929–1958.

[28] F. Tian, B. Gao, Q. Cui, E. Chen, T.-Y. Liu, Learning deep representations for graph clustering, in: Twenty-Eighth AAAI Conference on Artificial Intelligence, 2014, pp. 1293–1299.

[29] G. Chen, Deep learning with nonparametric clustering, arXiv preprint arXiv:1501.03084 (2015).

[30] G. E. Hinton, R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, science 313 (5786) (2006) 504–507.

[31] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, Journal of machine learning research 11 (Dec) (2010) 3371–3408.

[32] A. L. Maas, A. Y. Hannun, A. Y. Ng, Rectifier nonlinearities improve neural network acoustic models, in: Proc. icml, Vol. 30, 2013, p. 3.

[33] S. Wold, M. Sjostrom, Simca: a method for analyzing chemical data in terms of similarity and analogy, Chemometrics: theory and application 52 (1977) 243–282.

[34] R. De Maesschalck, D. Jouan-Rimbaud, D. L. Massart, The mahalanobis distance, Chemometrics and intelligent laboratory systems 50 (1) (2000) 1–18.

[35] J. C. Gower, Properties of euclidean and non-euclidean distance matrices, Linear Algebra and its Applications 67 (1985) 81–97.

[36] D. Anguita, A. Ghio, L. Oneto, X. Parra, J. L. Reyes-Ortiz, A public domain dataset for human activity recognition using smartphones., in: Esann, 2013, p. 3.

[37] Y. Meidan, M. Bohadana, A. Shabtai, M. Ochoa, N. O. Tippenhauer, J. D. Guarnizo, Y. Elovici, Detection of unauthorized iot devices using machine learning techniques, arXiv preprint arXiv:1709.04647 (2017).

[38] A. Reiss, D. Stricker, Introducing a new benchmarked dataset for activity monitoring, in: 2012 16th International Symposium on Wearable Computers, IEEE, 2012, pp. 108–109.

[39] M. Hahsler, M. Bolanos, J. Forrest, et al., Introduction to stream: An extensible framework for data stream clustering research with r, Journal of Statistical Software 76 (14) (2017) 1–50.

[40] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: Proceedings of the fourteenth international conference on artificial intelligence and statistics, 2011, pp. 315–323.

[41] G. Klambauer, T. Unterthiner, A. Mayr, S. Hochreiter, Self-normalizing neural networks, in: Advances in neural information processing systems, 2017, pp. 971–980.

[42] L. Hubert, P. Arabie, Comparing partitions, Journal of classification 2 (1) (1985) 193–218.

[43] N. Weber, A. Härmä, E. P. D. T. Heskes, Unsupervised learning in human activity recognition: A first foray into clustering data gathered from wearable sensors, Ph.D. thesis, Ph. D. Thesis, Radboud University, Nijmegen, The Netherlands, 2017.[Google ... (2017).

[44] Kdd cup 1999 data, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, accessed: 2020-06-01 (1999).

[45] P. Kushwaha, H. Buckchash, B. Raman, Anomaly based intrusion detection using filter based feature selection on kdd-cup 99, in: TENCON 2017 - 2017 IEEE Region 10 Conference, 2017, pp. 839–844.

[46] S. A. Aljawarneh, R. Vangipuram, Garuda: Gaussian dissimilarity measure for feature representation and anomaly detection in internet of things, The Journal of Supercomputing (2018) 1–38.

[47] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization., in: ICISSp, 2018, pp. 108–116.

[48] W. Su, Y. Yuan, M. Zhu, A relationship between the average precision and the area under the roc curve, in: Proceedings of the 2015 International Conference on The Theory of Information Retrieval, 2015, pp. 349–352.