Textual One-Pass Stream Clustering with Automated Distance Threshold Adaption

Dennis Assenmacher[2](✉) and Heike Trautmann[1]

[1] University of Münster, Münster, Germany
[2] GESIS - Leibniz Institute for the Social Sciences, Cologne, Germany
dennis.assenmacher@gesis.org

Abstract. Stream clustering is a technique capable of identifying homogeneous groups of observations that continuously arrive in a digital stream. In this work, we inherently refine a TF-IDF-based text stream clustering algorithm by the introduction of an automated distance threshold adaption technique for document insertion and cluster merging, improving the performance during distributional changes in the data stream. By conducting a thorough evaluation study, we show that our new fast approach outperforms state-of-the-art one-pass and batch-based stream clustering algorithms on various existing benchmarking datasets as well as a newly introduced dataset that poses additional challenges to the community. Moreover, we find that current evaluation approaches in the field of textual stream clustering are not adequate for a sound clustering performance assessment of evolving distributions. We thus demand the utilization of time-based evaluation.

Keywords: Stream clustering · Text mining · Concept drift

1 Introduction

Nowadays, researchers are confronted with large amounts of potentially unbounded data that have to be processed to extract desired information as quickly as possible in (almost) real-time. Even worse, there is so much data available that algorithms can only iterate over the data once to ensure timely processing leading to a new sub-domain of unsupervised learning: stream clustering [13]. In this work, we compare the performance of state-of-the art text-based stream clustering algorithms on various challenging datasets. Moreover, we improve and refine the initial draft of a textual stream clustering algorithm that we proposed in [5] to analyze chats on the Twitch platform. We introduce a mechanism that automatically determines the critical distance threshold parameter that controls the creation of new clusters and develop a new strategy to merge existing clusters close to each other. To validate the performance of our new approach, we conduct a thorough comparison between this new algorithm which we call `textClust` and existing state-of-the-art competitors on multiple datasets and evaluation metrics.

We show that we outperform existing one-pass stream clustering algorithms and even their computationally more complex batch-based alternatives. We identify shortcomings of current evaluation approaches that do not account for concept drift and propose time-based evaluation in upcoming studies.

Stream Clustering. Usually, stream clustering algorithms follow a widely accepted two-phase approach [1]. First, incoming data is aggregated on-the-fly, i.e., within an online phase. Here, not the complete dataset but only statistical summaries of dense areas in data-space (text-documents with high similarity) are kept for further analyses. These statistics are often referred to as *micro-clusters*. While the online phase is continuously running, micro-clusters may be re-clustered on demand and in an offline manner into so-called *macro-clusters*. In this offline step, traditional clustering approaches can be applied, and it is possible to iterate over the micro-clusters multiple times. An inherent property of data streams is that the underlying data distribution may change over time (also called concept drift). A comprehensive benchmark study between different stream clustering algorithms was conducted in [4].

1.1 Recent Work on Textual Stream Clustering

Textual stream clustering algorithms can be broadly classified according to *processing mode* and *representation*. The processing mode differentiates between one-pass and batch-based approaches. While an established assumption in the field is that streaming data must be processed and discarded afterward, some recent works relax this strict assumption and allow for batch-processing. In each batch, multiple iterations over the data are allowed [16]. Well-known representatives of this batch variant are the `MStream` algorithm presented by Yin et al. [16], and `DP-BMM` by Chen et al. [6]. One-pass clustering only allows that a new observation is processed once and has to be discarded after cluster assignment. `OSDM`, `EStream`, and the `textClust` algorithm are all representatives of this type of stream algorithms, although `MStream` and `DP-BMM` also offer a one-pass variant that can be utilized (for `MStream` this variant is called `MStreamF`).

Next to the processing mode, textual stream clustering algorithms are differentiated by their underlying representation of input documents. In *vector-space methods* each document and also the corresponding clusters are represented in a high-dimensional vector space using Term Frequency (TF) and Inverse Document Frequency (IDF) based representations [2,5,8]. The cosine similarity and the Euclidean distance are usually employed for calculating the proximity between clusters and new documents. Most recent research endeavors focus on *model-based approaches*, arguing that vector-space variants, in general, suffer from high dimensionality. Moreover, it is argued that selecting an appropriate distance threshold for deciding if a document is assigned to a cluster or not is infeasible. On the other hand, model-based approaches assume that topics (or clusters) emerge from an underlying generative model [6,9,17]. Those algorithms try to infer the distributional parameters by various means, such as Gibbs Sampling. Recently, Kumar et al. proposed a new stream clustering algorithm that explicitly targets short text content [9]. Their model-based `OSDM` algorithm follows the idea of several other approaches [6,9,17], which assume that (text)-documents are constructed by a generative Dirichlet multinomial mixture model.

While `MStream` utilizes unigrams for text representation, DP-BMM uses bigrams to overcome the data sparsity problem that often comes with short-text processing. The authors of `OSDM` extend these works by incorporating the semantic importance of words via co-occurrence information. Thereupon, Rakib et al. [11] improved the semantic clustering approach by utilizing pretrained document embeddings for outlier detection. Xu et al. improve the original `MStream` by considering topical correlations between different time steps (batches) with the introduction of `DCSS` [15]. In very recent work, Rakib et al. introduce the `EStream` online one-pass algorithm that dynamically computes similarity thresholds by utilizing distributional properties of common feature similarities [12]. In the following, we describe `textClust` in detail and moreover elaborate on significant improvements of the original work, such as an automated distance threshold selection.

2 Distance Based Clustering with Automatic Threshold Determination (`textClust`)

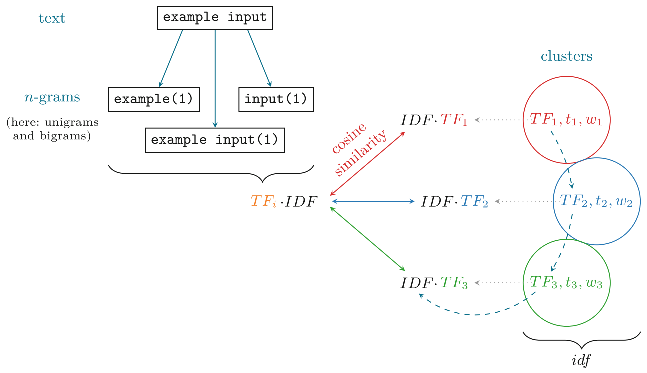

Fig. 1. Visualisation of the insertion strategy. A new incoming text is tokenized and n-grams and term frequencies created. Then, the IDF is computed from the term frequencies of all existing clusters. This allows to compute the TF-IDF vectors in order to measure the cosine similarity between the existing clusters and the new text.

`textClust` makes use of the common two-phase approach of stream clustering (see Sect. 1). First, an online component summarizes the stream in real-time, resulting in several micro-clusters that represent dense areas in the data space. In order to remove outdated clusters and merge similar clusters, a cleanup procedure is executed periodically. Finally, when the algorithm is supposed to output the final topics, these micro-clusters are reclustered into a final set of macro-clusters. `textClust` represents its micro-clusters mc as 3-tuples:

$$mc = (TF, t, w) \tag{1}$$

With $mc.TF$, the cluster holds a vector of token term frequencies as n-grams. The scalar $mc.t$ depicts the last time the micro-cluster was updated and the weight $mc.w$ how often a new observation was merged into that cluster (indirectly depicting cluster importance). Each time a new observation is merged into an existing cluster, the respective weight increases by 1. To account for distributional changes (concept drift), the cluster weights are exponentially faded each time a new observation appears in the data stream:

$$f(w) := w \cdot 2^{-\lambda(t_{now}-t)}, \lambda > 0, t_{now} > t \tag{2}$$

The parameter λ represents the fading (decay) factor, t_{now} the recent time stamp, and t the time of the last micro-cluster update. t_{gap} specifies the interval (number of new observations), after which a cleanup procedure and micro-cluster merging are triggered. This cleanup procedure removes all micro-clusters below a predefined weight threshold (i.e., clusters that were not recently updated) from the clustering result, ensuring that outdated micro-clusters are ultimately removed from the set of all existing clusters. The same fading method is applied to the token level (term fading). All tokens within a micro-cluster ($mc.TF$) have an associated weight which is updated and increased by one when a new observation with the same token is merged into the cluster. To determine the distance between two micro-clusters or a micro-cluster and a new incoming text, the inverse document frequency of a term is calculated over all micro-clusters (which are assumed to represent all documents). To calculate the distance between two of these sparse TF-IDF vectors a and b, initially, the cosine similarity is calculated by using the dot product:

$$\cos(\theta) = \frac{a \cdot b}{\|a\|_2 \cdot \|b\|_2} \quad \text{with } \theta := \angle(a, b) \tag{3}$$

As the TF-IDF vectors contain positive frequencies, $cos(\omega) \in [0, 1]$, this leads to the *cosine distance*

$$D_{\cos}(a, b) = 1 - \cos(\theta). \tag{4}$$

When a new observation arrives at the stream, a new virtual micro-cluster mc_{new} is created (with $w = 1$), and its cosine distance to all other micro-clusters is calculated. The closest micro-cluster is selected to be merged when its distance falls below a predefined threshold tr. The merging process between both clusters is realized by fading their weight according to the current time t_{now} and taking the union of their term frequencies:

$$mc_1 + mc_2 = \left(tf_1 \cup tf_2, \quad t_{now}, \quad w_1 \cdot 2^{-\lambda(t_{now}-t_1)} + w_2 \cdot 2^{-\lambda(t_{now}-t_2)}\right). \tag{5}$$

If, however, the distance threshold tr is not reached, the new virtual micro-cluster is added to the set of all clusters. The hyperparameters tr, λ, and t_{gap} have to be set in advance and especially tr highly influences the final clustering result (see Sect. 3.4). `textClust's` insertion strategy is displayed in Fig. 1.

2.1 Automatic Tresholding During the Online Phase

One of the main limitations of similarity or distance-based stream clustering is the adequate threshold (tr) selection for micro-cluster assignments. In the preliminary concept of our algorithm, a fixed distance threshold was used for (a) deciding whether new observations are merged into the closest existing micro-cluster and (b) for determining when existing micro-clusters should be merged during the cleanup procedure. Experiments and several real-world applications of the algorithm revealed that the threshold parameter highly influences the final clustering result [3]. Intuitively, a distance threshold that is too large will result in a small number of micro-clusters that contain diverse topics. On the contrary, a threshold that is too small will result in multiple, small mixed-topic micro-clusters. Selecting an adequate threshold is therefore imperative for achieving good clustering results. However, as we are dealing with stream data that is subject to distributional changes, we are additionally confronted with the problem that a fixed threshold may not be optimal along the entire stream.

Until now, when a new observation arrives at the stream, a new virtual micro-cluster mc_{new} is created and `textClust` calculates the distance between mc_{new} and each existing micro-cluster. If this distance is smaller than the previously specified fixed threshold (tr), the algorithm merges mc_{new} into the closest cluster (mc_{cl}). Otherwise, mc_{new} is added to the set of all micro-clusters. To automate the threshold selection, we assume that we want to merge an observation into an existing micro-cluster if it is sufficiently close to it compared to all other currently existing micro-clusters. As the distance between mc_{new} and each existing micro-cluster is already calculated during the online phase, we can easily record the mean distance as:

$$\mu = \frac{1}{|MC| - 1} \sum_{mc \in MC \setminus \{mc_{cl}\}} D_{cos}(mc_{new}, mc) \tag{6}$$

where MC denotes the set of micro-clusters. The standard deviation σ of those distances results as:

$$\sigma = \sqrt{\left(\frac{1}{|MC| - 1} \sum_{mc \in MC \setminus \{mc_{cl}\}} D_{cos}(mc_{new}, mc)^2 \right) - \mu^2}. \tag{7}$$

We can calculate both μ and σ by storing the (square) sum of all distances leading to a new definition of the threshold tr as

$$tr = \mu - c \cdot \sigma. \tag{8}$$

Intuitively, tr is set such that the observation's distance to its closest micro-cluster is smaller (within c times sd) than the average distance to all other micro-clusters. Thus, tr is always adjusted w.r.t. existing micro-cluster distances. Hence, distributional changes in the stream are also reflected in a dynamic change of the threshold, instead of using a fixed distance for the complete stream. In the calculation of μ and σ, we explicitly exclude the closest micro-cluster mc_{cl} in order to avoid bias. Our experiments (see Sect. 3.4) indicate that a constant weight c of 0.5 for σ proved to produce very satisfying results for a variety of different datasets.

A distance threshold is also required during the micro-cluster merging process for cleaning purposes. Originally, `textClust` used the same fixed threshold as for cluster assignment. For dynamic adaption, we now keep track of the minimum distances D_{cos} w.r.t. merging observations into an existing micro-cluster:

$$D_{merge} = D_{merge} + D_{cos}(mc_{new}, mc_{cl}), \quad \text{if } D_{cos}(mc_{new}, mc_{cl}) < tr \tag{9}$$

Intuitively, we collect a sum of distances for each time window that lead to merging new observations rather than creating new micro-clusters. If two micro-clusters are closer than the average observation merging distance, we assume that they belong together and should be merged. The observation merging distance may vary over time due to concept drift and our dynamic threshold adaption. Therefore, we reset D_{merge} at the end of each cleanup procedure.

2.2 Algorithm Specification

The pseudo-code of our new online-component is shown in Algorithm 1. First, the algorithm reads a new text $\boldsymbol{x}$ from the stream (Line 3). As a preprocessing step, the text is then converted to lower case and split into individual tokens or words (Line 4). In contrast to the initial proposal of this algorithm, we also remove all stop words from the input.

The remaining list of tokens is then used to construct n-grams and count their frequency with n_{min} and n_{max} specifying the gram-range (Lines 5 – Line 6). From the results we create a new virtual micro-cluster mc_{new} (Line 7). Before computing the cosine similarity/distance, we also require the IDF vector across all documents which are computed on-the-fly from the set of all existing clusters. With the IDF vector we calculate the TF-IDF representations of each existing and the new virtual micro-cluster mc_{new}, allowing us to compute the cosine distance between them (Line 12). The closest micro-cluster becomes our candidate for merging (Line 15).

Subsequently we check whether the candidate is sufficiently close to mc_{new} such that both can be merged. Suppose its distance is not larger than tr, which we compute from the mean and standard deviation (μ, σ) of all other micro-cluster distances. In that case, we assume that the text fits into the cluster (μ and σ are computed as specified in Line 16). We then merge the components of mc_{new} into the candidate cluster (Line 21). In order to update the weight correctly, we first need to ensure that the cluster's weight has been decayed to the current time (Line 20). However, if the distance to the candidate is larger than tr, mc_{new} does not fit the cluster sufficiently well. In this case, we add the temporary micro-cluster mc_{new} to the set of all clusters (Line 23). Note that the set of micro-clusters MC will be empty initially. In this case, we can skip the candidate selection and initialize a new cluster for $\boldsymbol{x}$ directly. The cleanup

Algorithm 1. `textClust`

```
Require: $n_{\min}$, $n_{\max}$, $\lambda$, $t_{gap}$, $c$
Initialize: $MC = \emptyset, S = \emptyset$
1: while stream is active do
2:     $t_{now} \leftarrow$ current timestamp
3:     Read new text $\boldsymbol{x}$ from stream at time $t_{now}$
4:     Preprocess $\boldsymbol{x}$ (lowercase, tokenization and stopword removal)
5:     Build $n$-grams from tokens $\forall n \in \{n_{\min}, \dots, n_{\max}\}$
6:     $tf \leftarrow$ Count frequency of $n$-grams
7:     $mc_{new} \leftarrow (tf, t_{now}, 1)$      ▷ cluster with weight one, Eq. (1)
8:     $\mu, \sigma, D_s, D_{ss} \leftarrow 0$
9:     if $|MC| > 2$ then      ▷ The first two observations are considered as separate clusters
10:        $idf \leftarrow$ Calculate IDF from term-frequencies of micro-clusters in $MC$
11:        for each micro-cluster $mc \in MC$ do
12:            $dist \leftarrow D_{cos}(mc_{new}, mc)$      ▷ Cosine distance between TF-IDF vectors, Eq. (4)
13:            $D_s = D_s + dist$
14:            $D_{ss} = D_{ss} + dist^2$
15:        $mc_{cl} \leftarrow$ closest micro-cluster according to cosine distance $D_{cos}$
16:        $\mu \leftarrow \frac{D_s}{|MC|-1}$, $\sigma \leftarrow \sqrt{\frac{D_{ss}}{|MC|-1} - \mu^2}$
17:        $tr \leftarrow \mu - c \cdot \sigma$
18:        if $D_{cos}(mc_{new}, mc_{cl}) \leq tr$ then
19:            $D_{merge} \leftarrow D_{merge} + D_{cos}(mc_{new}, mc_{cl})$
20:            $mc_{cl}.w \leftarrow mc_{cl}.w \cdot 2^{-\lambda(t_{now} - mc_{cl}.t)}$      ▷ fade weights (last update $mc_{cl}.t$) Eq. (2)
21:            Merge $mc_{new}$ into micro-cluster $mc_{cl}$      ▷ Eq. (5)
22:        else
23:            Add $mc_{new}$ to set of micro-clusters $MC$
24:    else
25:        Add $mc_{new}$ to set of micro-clusters $MC$
26:    if $t_{now} \mod t_{gap} = 0$ then
27:        CLEANUP( · )
28:        MERGE($D_{merge}$)
```

procedure which is responsible for cluster- and token fading, as well as micro-cluster merging is triggered each t_{gap} timesteps.

3 Experiments

3.1 Benchmarking Datasets

The most frequently used and accepted text-based stream datasets are `News-T` and `Tweets-T` [16]. Recently, Rakib et al. combined both datasets to `NT`. Also, they proposed a new dataset containing question titles from Stackoverflow (`SO-T`), and combined it with `NT` into `NTS`. All datasets (except for `SO-T`) are sorted by topics since it is argued that in a real-world application, topics usually emerge and disappear after some time [16]. However, we see that there is still potential for additional evaluation sets that focus on different problems not captured yet by research. In this work, we generate a new Twitter-based dataset that is (a) significantly larger than the existing benchmarking data and (b) captures a low number of popular (trending) and intertwined topics on Twitter over time. In contrast to existing benchmarking datasets, the captured data is close to a real-world scenario, where raw social streams (including irrelevant posts) are monitored, and cluster algorithms should reveal groups of similar topics (conversations) in real-time. This allows us to investigate whether an algorithm works comparably well in a realistic setting and not only when, unrealistically, the text is already preprocessed and sorted. Additionally, we include timestamp information to enable clustering based on real-world throughput. We use the Twitter API to collect labelled data from the new dedicated Academic `API` and assume that a filtered search result for one hashtag/trend represents a single topic distribution (one class). We query Tweets of different trending hashtags/topics on the day they occur (we capture 24h of the day) and assign each tweet belonging to the same hashtag to the same class. Data was collected from 10 different topics/trends. Our new `Trends-T` dataset consists of 200.000 Tweets.

3.2 Experimental Setup

We evaluate `textClust` with $c = 0.5$ for automated threshold adaption on five different datasets frequently used in related literature for benchmarking purposes. Additionally, we conduct a comparison of the algorithms on our new `Trends-T` dataset. We compare the performance of the previously described state-of-the-art stream clustering one-pass algorithms (`EStream`, `OSDM`), as well as four batch-based alternatives (`MStreamF`, `DP-BMM`, `MS-Rakib` and `DCSS`) [6,10,11,16]. To establish a fair comparison, we use the deterministic one-pass variants of these algorithms. We did not shuffle the datasets randomly, as most of them are specifically constructed in a way such that topics only appear in specific time windows (similar to real-world conversation scenarios). Upon inspection, the previously introduced `SO-T` dataset exhibits unique characteristics. Topics are globally shuffled and do not reveal true time-dependency. Secondly, there are systematically more ground-truth clusters (10% of the total data stream vs. less than 1% for the other datasets). Choosing the correct evaluation procedure for a textual stream clustering scenario is not an easy task. We find that there does not seem to exist consent in the literature on the procedure to be applied. In practice, we observe the utilization of two different evaluation strategies. The first evaluation method we will refer to as *global evaluation*. Each time a new observation is assigned to a new cluster, the cluster's ID is stored, and then the clustering is updated. After the complete data-input stream has been clustered, the different metrics are calculated on the stored clustering IDs and compared to the ground-truth. This technique is quite accepted in literature, as it was used in several recent works [6,11]. Second, there is an *interval-based evaluation*. Here, the cluster results are split into equally sized batches (horizons), and all evaluation metrics are calculated for each horizon individually. This allows for tracking the performance of the algorithms over time. When applied, this approach is usually conducted in addition to the global evaluation [9,16]. We argue that all evaluation approaches should conduct such a time- or interval-based evaluation procedure as a sole global evaluation is insufficient for determining how algorithms can adapt to upcoming distributional changes (concept drift) and often overestimate the actual performance. If, however, prediction quality and robustness of the stream is focused, prequential [7] evaluation is recommended. To establish a fair comparison, we use the reference parameter settings, specified by the authors and do not change them for all other datasets. For both batch-based algorithms (DP-BMM and MStreamF) the standard settings are often optimized on one of the well-established baseline datasets (News-T or Tweets-T). For OSDM, we set $\alpha = 2e^{-3}$, $\beta = 4e^{-5}$ and $\lambda = 6e^{-6}$. MStreamF uses $\alpha = 3e^{-2}$, $\beta = 3e^{-2}$, $iter = 10$ (number of iterations) and the number of stored batches to 1. The EStream algorithm only consists of a single parameter which determines the delete interval (DI). We set DI to 500, as proposed in the original work [12]. For textClust we only need two parameters, as the threshold is now dynamically determined, i.e. we fix standard parameters $\lambda = 1e^{-2}$ and $t_{gap} = 200$.

3.3 Evaluation Metrics

In this work, we focus on external evaluation metrics, i.e., metrics that use ground-truth (*a-priori*) information. Moreover, we only evaluate micro-clusters and leave the macro-cluster perspective to a future endeavor as all of the competing algorithms are developed as pure online methods, only focusing on micro-level. While there are internal cluster evaluation metrics such as the Silhouette coefficient, the Dunn Index and other density-measures, we omit these in our experimental evaluation since those evaluation metrics are intrinsic measures that are usually biased towards certain cluster shapes (and thus favor those algorithms which optimize towards them). For micro-cluster evaluation, we use the homogeneity, completeness and Normalized Mutual Information (NMI) [14]. We employ the widely accepted normalized variant of Mutual Information (NMI) for evaluation which scales the results to a range between 0 and 1 [14].

3.4 Experimental Results

In Table 1 the performances of the algorithms for both *global* and *interval-based* evaluation are displayed for all datasets[1]. The horizon for the interval-based approach is heuristically set to 1000 and the mean result of all horizons is calculated. As textClust and EStream are deterministic, no repeated runs are necessary. For the other algorithms we report the mean of 5 runs. Clearly, the NMI results show that textClust outperforms existing algorithms on most of the examined datasets (News-T, Tweets-T, NT, NTS and Trends-T). An exception is the new SO-T dataset, where EStream produces slightly better results. Nonetheless, textClust can compete in contrast to other approaches which seem to deteriorate over time (see Fig. 2). Interestingly and in accordance with previous publications, we find that completeness scores tend to be lower than the homogeneity scores. This indicates that online clusters themselves are quite pure with low entropy, but the topics themselves are distributed over multiple clusters instead of a single one. We argue that this behavior does not necessarily exhibit wrong clustering behavior or a bad clustering at all. To achieve a perfect completeness score, each observation of a specific class must be assigned to the same cluster. However, because of the dynamic nature of stream clustering algorithms,

[1] All evaluation scripts and results are made publicly available on Github: https://github.com/Dennis1989/textClust-experiments.

Table 1. Algorithm performance for global and interval-based mean evaluation. Top performances are highlighted in bold.

	Metric	Ns-T	Ts-T	NT	NTS	SO-T	Tr-T	Ns-T	Ts-T	NT	NTS	SO-T	Tr-T
		Global results						Mean results					
DP-BMM		0.548	0.395	0.506	0.453	0.118	0.039	0.875	0.913	0.948	0.895	0.402	0.229
MSTREAM		0.872	0.872	0.927	0.875	0.120	0.352	0.884	0.910	0.956	0.885	0.359	0.524
MS-RAKIB		0.866	0.948	**0.982**	0.965	0.632	0.342	0.839	0.956	**0.982**	0.931	0.652	0.509
DCSS		0.916	0.928	0.949	0.947	0.314	0.231	0.874	0.872	0.917	0.832	0.121	0.027
OSDM	Hom.	0.900	0.939	0.961	0.903	0.146	0.373	0.891	0.957	**0.982**	0.900	0.168	0.387
EStream		**0.956**	**0.963**	0.931	0.930	**0.720**	0.381	**0.949**	0.966	0.975	**0.967**	**0.687**	0.244
textClust-bigram		0.912	0.941	0.972	**0.973**	0.628	0.927	0.901	**0.959**	0.960	0.940	0.611	0.918
textClust-unigram		0.900	0.937	0.968	0.960	0.595	**0.929**	0.893	0.953	0.954	0.905	0.614	**0.904**
DP-BMM		0.718	0.561	0.769	0.641	0.351	0.070	0.791	0.765	0.691	0.537	0.953	0.375
MSTREAM		0.870	0.879	0.873	0.798	0.387	0.343	0.793	0.798	0.691	0.535	0.947	0.475
MS-RAKIB		0.785	0.689	0.741	0.674	0.705	0.397	0.684	0.570	0.492	0.370	0.963	0.540
DCSS		0.820	0.77	0.887	0.872	0.556	0.097	0.750	0.717	0.713	0.605	0.928	0.097
OSDM	Comp.	0.818	0.764	0.825	0.732	0.726	0.232	0.726	0.635	0.604	0.458	0.956	0.298
EStream		0.757	0.750	0.798	0.686	**0.730**	0.119	0.656	0.660	0.579	0.434	0.964	0.176
textClust-bigram		**0.871**	0.914	0.908	**0.843**	0.706	0.350	0.798	0.853	0.751	**0.588**	0.966	0.572
textClust-unigram		0.866	**0.920**	**0.912**	0.826	0.697	**0.412**	**0.800**	**0.864**	**0.758**	0.583	**0.967**	**0.585**
DP-BMM		0.622	0.463	0.610	0.531	0.177	0.050	0.829	0.828	0.781	0.640	0.565	0.281
MSTREAM		0.871	0.875	0.899	0.834	0.183	0.347	0.838	0.846	0.783	0.632	0.520	0.496
MS-RAKIB		0.823	0.798	0.845	0.794	0.666	0.368	0.750	0.709	0.638	0.501	0.777	0.521
DCSS		0.860	0.842	0.916	0.902	0.401	0.137	0.787	0.790	0.787	0.682	0.211	0.038
OSDM	NMI	0.857	0.842	0.888	0.809	0.243	0.286	0.796	0.759	0.730	0.573	0.283	0.335
EStream		0.845	0.843	0.859	0.789	**0.725**	0.182	0.773	0.780	0.709	0.557	**0.802**	0.203
textClust-bigram		**0.891**	0.927	0.939	**0.903**	0.665	0.509	**0.845**	0.898	0.823	**0.687**	0.748	0.703
textClust-unigram		0.883	**0.929**	**0.939**	0.887	0.601	**0.582**	0.842	**0.903**	**0.825**	0.668	0.751	**0.709**

they tend to produce more micro-clusters than classes, decreasing overall completeness. This specifically holds for more complex datasets with a labelling that captures broader topics like our new `Trend-T` dataset. An explanation for this is that the labeling for these datasets is not perfect and only captures "global" topics.

Figure 2 displays the algorithm's NMI performance in batches containing 1000 observations for all evaluation datasets. With this, we want to emphasize the necessity to capture the performance over time, as we can identify potential shortcomings of current techniques that lead to considerable performance dips (especially in the context of `NT` and `NTS`). Manual inspection revealed that these dips, which occur for each tested algorithm, are caused by batches that indicate a radical decrease in the number of contained topics. In extreme cases, a batch only consisted of one single topic resulting in a dip to 0 NMI. Interestingly, despite capturing more context and thus being more computationally demanding, the utilization of bigrams proved to be less efficient or only marginally better than unigrams. This indicates that good clustering can already be achieved on a single word level. Figure 2 again shows that `textClust` is always superior or en par with the top-performing competitors when evaluated for different horizons.

Within an additional sensitivity analysis, we tested the influence on the weight put on σ and how well our threshold adaption performed compared to

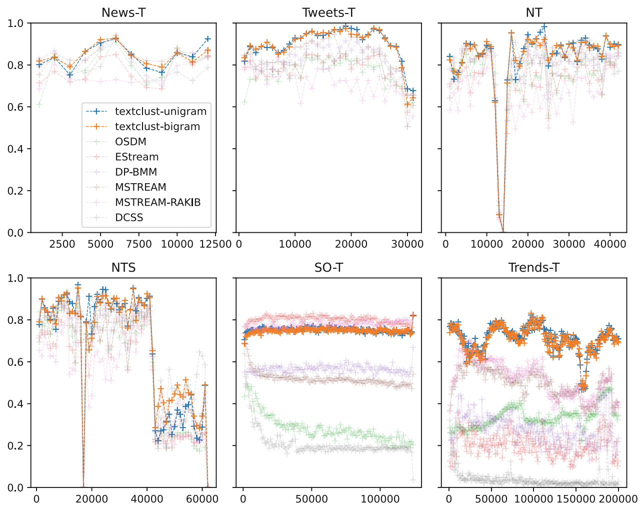

Fig. 2. Algorithm's NMI performance over time with a horizon of 1000.

the previously used fixed threshold selection. The left subplot indicates that calculating the new threshold by using 0.5 times the standard deviation, leads to good results. While 0.5 is not optimal for each of the tested datasets, we consider it as a suitable standard configuration and an acceptable trade-off for avoiding the manual selection of a hard-to-determine parameter. For the experiments shown in the right subplot we executed `textClust` on all datasets with a set of different fixed thresholds: $tr \in \{0.1, .., 1\}$. The resulting NMI score distributions are displayed as boxplots and the NMI performance with our automated threshold adaption $(0.5 \cdot \sigma)$ is added in blue. It is evident that the performance variance when searching for the weighting factor for σ is significantly smaller than the performance variance during the threshold search, indicating that our new parameter is more stable. In general, our fully automated threshold approach leads to excellent clustering solutions that are clearly above the average score achieved with this threshold search. However, we also observe that the `SO-T` dataset behaves differently than the other evaluation datasets. While with increasing σ weight, the performance slowly decreases for all other data streams, it increases for `SO-T`. Further inspection of this dataset revealed that the number of clusters in `SO-T` is unusually high, compared to the other ones (10% of the number of observations). This leads to a situation, where good performance can be simply achieved by creating a lot of micro-clusters. High weight on σ leads to low tr. Consequently, with higher weights, more micro-clusters are created. However, it remains unclear whether the scenario of the `SO-T` dataset is representative for the respective domain (Fig. 3).

Last, we inspected the runtime of all algorithms on each individual dataset (Table 2). As described in the original paper, `EStream` was specifically designed for rapid clustering. It is not surprising that this approach outperforms all other

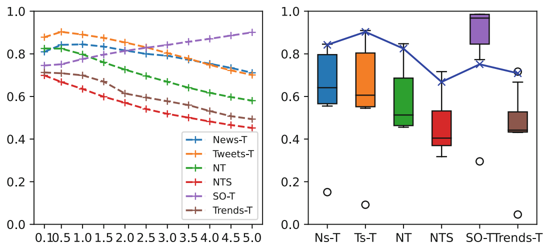

Fig. 3. Interval-based sensitivity evaluation. Left: NMI for different σ weighting factors. Right: NMI influence for different *tr* without automatic adjustments. The blue line indicates the performance of our new approach with automatic adjustments enabled. (Color figure online)

Table 2. Average algorithm runtime (in seconds)

Algorithm	Ns-T	Ts-T	NT	NTS	SO-T	Tr-T
DP-BMM	13.21	79.3	60.18	86.32	420.00	1901.20
MSTREAM	21.82	124.36	185.34	347.21	1588.14	4780.54
MS-RAKIB	32.37	102.29	143.00	210.60	711.34	10606.94
OSDM	20.09	122.77	94.38	138.96	754.41	5770.57
EStream	**2.22**	**10.05**	**14.57**	**25.77**	**147.13**	**601.62**
textClust-unigram	**9.06**	**30.45**	**37.14**	**60.75**	**165.18**	**605.01**
textClust-bigram	17.76	61.88	76.88	114.24	344.44	1625.06

algorithms, including `textClust`. However, for the larger datasets such as `SO-T` and `Trends-T`, `textClust` (especially the unigram variant) performs comparably fast and almost achieves the same speed as `EStream`. All other algorithms are at least two times slower than `textClust`. Since `textClust` outperforms `EStream` on all datasets except one, we argue that we are confronted with a trade-off between speed and clustering accuracy.

4 Discussion and Future Work

In this work, we significantly improved our draft of a textual stream clustering algorithm by tackling various shortcomings such as fixed distance threshold settings during the online clustering. We showed that our improved approach could not only compete but also outperform state-of-the-art textual clustering algorithms in the field. Moreover, we proposed a new benchmarking dataset that poses additional challenges and moves evaluation closer towards real-world settings. Our evaluation results indicate that only evaluating the global clustering result is not sufficient and does not properly show algorithm adoption of concept drift nor possible improvements. Moreover, we showed that global evaluation often overestimates the actual performance of the different algorithms. We stress that the textual stream clustering community needs a standardized evaluation procedure to enable fair comparison and an objective performance assessment of different algorithms. Future work will focus on finding suitable and robust parameter settings for `textClust` across a large variety of textual benchmark datasets based on sophisticated algorithm configuration procedures. Ideally, the configuration framework should also allow for dynamic adjustments.

References

1. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: Proceedings of the 29th International Conference on Very Large Data Bases, Berlin, Germany, vol. 29, pp. 81–92 (2003)
2. Aggarwal, C.C., Yu, P.S.: On clustering massive text and categorical data streams. Knowl. Inf. Syst. **24**(2), 171–196 (2010)
3. Assenmacher, D., Adam, L., Trautmann, H., Grimme, C.: Towards real-time and unsupervised campaign detection in social media. In: Barták, R., Bell, E. (eds.) FLAIRS 2020 - Proceedings of the 33rd International Florida Artificial Intelligence Research Society Conference, pp. 303–306. AAAI Press (2020)
4. Carnein, M., Assenmacher, D., Trautmann, H.: An empirical comparison of stream clustering algorithms. In: Proceedings of the ACM International Conference on Computing Frontiers (CF 2017), pp. 361–365. ACM (2017)
5. Carnein, M., Assenmacher, D., Trautmann, H.: Stream clustering of chat messages with applications to twitch streams. In: de Cesare, S., Frank, U. (eds.) ER 2017. LNCS, vol. 10651, pp. 79–88. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70625-2_8
6. Chen, J., Gong, Z., Liu, W.: A dirichlet process biterm-based mixture model for short text stream clustering. Appl. Intell. **50**, 1609–1619 (2020)
7. Gama, J.A., Rodrigues, P.P., Sebastião, R.: Evaluating algorithms that learn from data streams. In: Proceedings of the 2009 ACM Symposium on Applied Computing. SAC 2009, New York, pp. 1496–1500. Association for Computing Machinery (2009)
8. Khalilian, M., Sulaiman, N.: Data stream clustering by divide and conquer approach based on vector model. J. Big Data **3**, 1–21 (2016)
9. Kumar, J., Shao, J., Uddin, S., Ali, W.: An online semantic-enhanced dirichlet model for short text stream clustering. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pp. 766–776 (2020)
10. Rakib, M.R.H., Asaduzzaman, M.: Fast clustering of short text streams using efficient cluster indexing and dynamic similarity thresholds (2021)
11. Rakib, M.R.H., Zeh, N., Milios, E.: Short text stream clustering via frequent word pairs and reassignment of outliers to clusters. In: Proceedings of the ACM Symposium on Document Engineering 2020. DocEng 2020, New York, NY, USA. Association for Computing Machinery (2020)
12. Rakib, M.R.H., Zeh, N., Milios, E.: Efficient clustering of short text streams using online-offline clustering. In: Proceedings of the 21st ACM Symposium on Document Engineering. DocEng 2021, New York, NY, USA. Association for Computing Machinery (2021). https://doi.org/10.1145/3469096.3469866
13. Silva, J.A., Faria, E.R., Barros, R.C., Hruschka, E.R., Carvalho, A.C.D., Gama, J.: Data stream clustering: a survey. ACM Comput. Surv. **46**(1), 1–31 (2013)
14. Vinh, N.X., Epps, J., Bailey, J.: Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. J. Mach. Learn. Res. **11**(95), 2837–2854 (2010)

15. Xu, W., Li, Y., Qiang, J.: Dynamic clustering for short text stream based on dirichlet process. Appl. Intell. **52**, 4651–4662 (2021)
16. Yin, J., Chao, D., Liu, Z., Zhang, W., Yu, X., Wang, J.: Model-based clustering of short text streams. In: KDD 2018, New York, USA, pp. 2634–2642. Association for Computing Machinery (2018)
17. Yin, J., Wang, J.: A text clustering algorithm using an online clustering scheme for initialization. In: KDD 2016, New York, USA, pp. 1995–2004. Association for Computing Machinery (2016)