Stream Clustering of Chat Messages with Applications to Twitch Streams

Matthias Carnein(✉), Dennis Assenmacher, and Heike Trautmann

University of Münster, Münster, Germany
{matthias.carnein,dennis.assenmacher,
heike.trautmann}@ercis.uni-muenster.de

Abstract. This paper proposes a new stream clustering algorithm for text streams. The algorithm combines concepts from stream clustering and text analysis in order to incrementally maintain a number of text droplets that represent topics within the stream. Our algorithm adapts to changes of topic over time and can handle noise and outliers gracefully by decaying the importance of irrelevant clusters. We demonstrate the performance of our approach by using more than one million real-world texts from the video streaming platform Twitch.tv.

Keywords: Data stream · Stream clustering · Text analysis · Text clustering · Twitch.tv

1 Introduction

Due to the increasing number of real-world applications producing data streams, the analysis of streaming data has become a key area of interest for research and practice. A core topic is the clustering of streaming data [9] which can be a valuable tool, e.g. for sensor analysis or customer segmentation [5]. However, a considerable amount of data nowadays comes in the form of text data such as e-mails, websites or chats. In order to analyse this data, it is necessary to build stream clustering algorithms that handle and analyse streams of texts [1].

In this paper, we combine ideas from text analysis and stream clustering in order to build an algorithm that can handle rapid data streams of text data. The texts can be of arbitrary length, language and content. Our approach uses a popular stream clustering approach [2] where the stream is first summarised, resulting in a number of small discussion threads. The algorithm adapts to changes in topics by forgetting outdated clusters while simultaneously identifying emerging clusters. Whenever necessary, the identified summaries can be *reclustered* by using a distance-based clustering algorithm to generate the overall topics. We demonstrate and evaluate our approach on real-world data streams by analysing more than one million chat messages from the video streaming platform Twitch.tv.

The remainder of this paper is structured as follows: Sect. 2 presents relevant approaches from text analysis and stream clustering. Next, Sect. 3 proposes our new algorithm which is able to cluster data streams of arbitrary text data. We evaluate the algorithm in Sect. 4 by clustering real-world chat messages. Finally, Sect. 5 concludes with a summary of the results and an outlook on future research.

2 Related Work

2.1 Text Analysis

A common approach when analysing text data is to vectorize the input texts and use the distance between the vectors as a measure of similarity. A simple approach is to use a vector of Term Frequencies (TFs) as the number of occurrences of a term t in a document d, i.e. $\mathrm{TF}(t,d) = |\{t \in d\}|$. However, a more popular approach is to count the number of occurrences for term-sequences in order to capture the context of words. Such term-sequences are called n-grams where n describes the size of the sequence.

Since not all terms provide the same amount of information, one can weight the TF with the Inverse Document Frequency (IDF). The IDF denotes whether a term is rare or common among all N available documents D. The underlying assumption is that rare terms provide more information. The IDF is often smoothed by taking the logarithm:

$$\mathrm{IDF}(t,D) = \log\left(\frac{N}{|\{d \in D : t \in d\}|}\right) \tag{1}$$

By multiplying both values, one obtains the so called Term Frequency – Inverse Document Frequency (TF-IDF) $= \mathrm{TF}(t,d) \cdot \mathrm{IDF}(t,D)$.

Finally, the distance between the resulting TF-IDF vectors $\boldsymbol{A}$ and $\boldsymbol{B}$ can then be computed as the cosine of the angle between them:

$$\cos(\theta) = \frac{\boldsymbol{A} \cdot \boldsymbol{B}}{\|\boldsymbol{A}\|\|\boldsymbol{B}\|}. \tag{2}$$

2.2 Stream Clustering

Due to the popularity of streaming applications, stream clustering has gained considerable attention in the past. While traditional cluster analysis assumes a fixed set of data, stream clustering works on a continuous and unbounded stream of new observations which cannot be permanently stored or ordered. Stream clustering aims to maintain a set of currently valid clusters, i.e. by removing outdated and learning emerging structures.

A recent example of a stream clustering algorithm is `DBSTREAM` [7]. In experimental results, the algorithm has shown the highest quality among popular stream clustering algorithms and fast computation time [5]. `DBSTREAM` employs a popular two-phase approach to cluster data streams [2]. Within an online phase, new data points are evaluated in real time and relevant summary statistics are captured. The result is a number of *micro-clusters* that summarise a large number of preliminary clusters in the data. During an offline phase, these micro-clusters can be 'reclustered' to derive the final clusters, called 'macro-clusters'.

In the online phase, `DBSTREAM` assigns a new observation to already existing micro-clusters if it lies within a radius threshold. When a micro-cluster absorbs the observation, its weight is increased and its centre is moved towards the new observation. If the observation lies within the distance threshold of multiple micro-clusters, it is added to all of them and a shared density between the clusters is stored as the number of points in the intersection of their radii. If the observation cannot be absorbed, it is used to initialize a new micro-cluster at its position. Since data streams may evolve over time, algorithms 'forget' outdated information [4]. `DBSTREAM` employs an exponential decay where the weights of micro-clusters are faded by $2^{-\lambda}$, in every time step. In fixed intervals, the algorithm evaluates the weight of all micro-clusters and removes those, whose weight decayed below a threshold. The offline component merges micro-clusters that have a high shared density to build the final clustering result.

2.3 Stream Text Clustering

One of the earliest algorithms to extend stream clustering to categorial and text data is `ConStream` (Condensation based Stream Clustering) [3]. The algorithm was proposed in 2006 and also follows the previously introduced two-phase clustering approach. Within the online phase, the algorithm maintains a set of summary statistics called cluster droplets which are similar to the concept of Clustering Features [10]. A cluster droplet is described as a tuple $(\overrightarrow{DF2}, \overrightarrow{DF1}, n, w(t), l)$ at a specific time t. $\overrightarrow{DF1}$ represents the sum of weighted occurrences for each word in a cluster. $\overrightarrow{DF2}$ stores, for each pair of terms, the sum of the weighted co-occurrences. This component is only used during the offline phase in order to analyse inter-cluster correlations. The remaining components of the tuple are scalar values that represent the number of observations within the cluster n, the total weight at a specific point in time $w(t)$ and a time stamp l that denotes the last time, the corresponding micro-cluster was updated.

Within the online phase, the similarity between a new data point $\overrightarrow{X}$ and all existing cluster droplets is calculated by using the cosine similarity on $\overrightarrow{DF1}$. If the similarity to the closest droplet is less than a predefined threshold, the algorithm searches for an inactive cluster which has not been updated in a while. It replaces the longest inactive cluster and initializes a new one for $\overrightarrow{X}$ instead. If no inactive cluster exists or the similarity is larger than the threshold, $\overrightarrow{X}$ is merged with the closest cluster. In the latter case, the updated droplet is faded by using the previously introduced decay function. In order to be able to compare and analyse different time horizons, a snapshot of all existing droplets is frequently persisted on secondary memory.

A review of relevant text stream algorithms can be found in [1]. However, we observed that within the popular research field of stream clustering, the aggregation of text data has received little attention and leaves room for improvement.

3 Efficient Stream Clustering of Text Data

In this section, we propose a new approach to cluster streams of text data. Our approach incrementally builds micro-clusters for text messages where each cluster maintains enough information to calculate a TF-IDF vector. The novelty our approach is as follows: (1) We combine state-of-art stream clustering concepts with text analysis using TF-IDF vectors, (2) we propose an incremental maintenance of the TF-IDF vectors, (3) we propose two fading approaches that efficiently maintain clusters. The approach is easy to implement and applicable to texts of arbitrary content, language and length.

The algorithm employs the widely accepted two-phase clustering approach where the online component updates the model whenever a new text is observed. The pseudo-code of this step is shown in Algorithm 1. We start by tokenizing the text and building the corresponding n-grams (line 2). Instead of using fixed size n-grams, we allow the user to specify a range between $n_{\min}$ and $n_{\max}$ and build all term-sequences within the given range. The result is an occurrence count for each n-gram in the text which can be considered its weight. Afterwards a temporary micro-cluster c is initialized for the new text. The micro-cluster is represented by the n-grams count, its weight is initialized to one and its time of last update is set to the current time t (line 3).

Algorithm 1. Update procedure

```
Require: $t$, $n_{\min}$, $n_{\max}$, message, $\lambda$, $t_{gap}$, $r$
 1: function INSERT(message, $n_{\min}$, $n_{\max}$)
 2:     ngrams ← N-GRAMTOKENIZER(message, $n_{\min}$, $n_{\max}$)
 3:     $c \leftarrow (\text{ngrams}, t, 1)$                      ▷ temporary micro-cluster
 4:     for $i \leftarrow 1, ..., |MC|$ do
 5:         $d_i \leftarrow$ COSINESIMILIARITY($mc_i$, $c$)
 6:     $j \leftarrow \arg\min_i(d_i)$                           ▷ find closest micro-cluster
 7:     if $d_j \leq r$ then
 8:         $mc_j \leftarrow$ MERGE($c$, $mc_j$, $2^{-\lambda \Delta t}$)
 9:     else
10:         add $c$ to $MC$
11:     if $t \mod t_{gap} = 0$ then
12:         CLEANUP( )
13:     $t \leftarrow t + 1$
```

Next, we search for the closest existing micro-cluster to c by calculating the cosine similarity between the TF-IDF vector of c and all existing micro-clusters $mc_i \in MC$ (lines 4 – 6). While the TF-vector is directly available, the IDF vector can be calculated by summing the n-gram occurrences over all available micro-clusters.

If the closest micro-cluster is within a distance threshold r, we merge c into it (line 8). Micro-clusters can be easily merged by summing the number of occurrences per n-gram, summing the weight and setting the time of last update to the current time. If the closest micro-cluster is not within the distance-threshold, the temporary-cluster c is added as a new micro-cluster into the model (line 10).

In order to forget outdated clusters, we exponentially decay clusters using the fading function $2^{-\lambda \Delta t}$, where λ is a user chosen parameter and Δt is the time since the cluster was last updated. Whenever we update a cluster, e.g. by merging, we also update its weight. In addition, we also employ the same strategy to decay the frequency of each n-gram within the clusters.

Finally, we adjust the clustering every t_{gap} time-units to account for changes in weight and similarity of clusters (line 12). Algorithm 2 outlines its pseudo-code. First, we update the cluster-weight by applying the fading function to each micro-cluster (line 3). If a weight has decayed below $2^{-\lambda t_{gap}}$ we delete the micro-cluster (line 5). We choose this threshold, since it takes a new micro cluster at least t_{gap} time to decay to this weight [7]. Similarly, we update the weight of all n-grams within the micro-clusters by decaying their frequency and removing those that decayed below the same threshold (line 9). Finally, we evaluate the similarity between all pairs of micro-clusters. If clusters have moved into the distance threshold r, it is likely that they belong to the same cluster and we merge them as outlined above (line 10).

Algorithm 2. Cleanup procedure

```
 1: function CLEANUP( )
Require: $t_{gap}$, $MC$, $\lambda$, $r$
 2:     for each $mc \in MC$ do                                    ▷ fade micro-cluster
 3:         WEIGHT($mc$) ← WEIGHT($mc$) $\cdot 2^{-\lambda \Delta t}$
 4:         if WEIGHT($mc$) $\leq 2^{-\lambda t_{gap}}$ then
 5:             REMOVE($mc$)
 6:         for each $n$-gram $\in mc$ do                          ▷ fade $n$-grams
 7:             WEIGHT($n$-gram) ← WEIGHT($n$-gram) $\cdot 2^{-\lambda \Delta t}$
 8:             if WEIGHT($n$-gram) $\leq 2^{-\lambda t_{gap}}$ then
 9:                 REMOVE($n$-gram) from $mc$
10:     Merge all $mc_i, mc_j$ where COSINESIMILIARITY($mc_i$, $mc_j$) $\leq r$
```

The result of the online component is a number of small micro-clusters, their weight and relevant n-grams that describe the cluster. This can be thought of as subtopics or small discussion threads in the stream. In order to derive the final topics from the micro-clusters we recluster them using a traditional clustering approach. To do so, we calculate a pair-wise distance matrix between all micro-clusters based on their cosine-similarity. We use the resulting matrix to apply hierarchical agglomerative clustering with complete linkage but any distance-based clustering approach is applicable.

4 Evaluation

4.1 Experimental Setup

In order to evaluate our approach, we implemented it as an extension to the `stream` package [6], a plugin for the statistical programming language `R`. In addition, we made use of real-world chat messages from the video streaming platform Twitch.tv. The platform is mostly hosting video-game related streams that regularly attract thousand of viewers which use a chat in order to communicate. Technologically, the chat is based on the Internet Relay Chat (IRC) protocol which allows real-time access to the chat.

For our analysis, we identified the ten most popular streams based on viewership and number of followers and collected the chat data from April, 12 to April, 14 2017. These channels produced more than one million text messages and on average we received 6.71 messages per second. Figure 1 indicates some seasonality in the data since streamers and viewers are less active around noon, as highlighted in red.

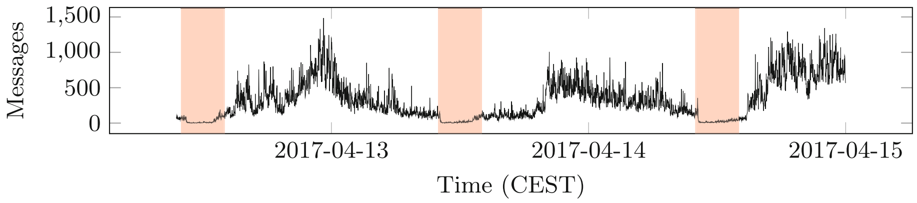


Fig. 1. Distribution of messages over the recorded period (Color figure online)

The goal of our analysis is to see whether we are able to identify reasonable discussion topics in the chat stream. Additionally, we try to find out whether the discussion contents of the streams differ, i.e. whether we are able to identify clusters which coincide with the different channels.

A core concept of our approach which differs from previous works is the capability to fade n-grams within a micro-cluster individually. This ensures that only relevant n-grams remain which greatly reduces the memory usage while maintaining comparable quality. In order to evaluate the performance of 'token fading', we executed our whole evaluation procedure on two different setups of the algorithm: the first setup explicitly utilizes the individual fading of tokens (cf. Algorithm 2 lines $6-9$), whereas the second setup only fades the cluster-weights.

4.2 Evaluation Criteria

Evaluating cluster quality can be a challenging task since clustering aims at the discovery of unknown patterns. For this reason, quality is often evaluated using intrinsic information of the clusters, such as shape, size, and distance to other clusters. This approach is called internal evaluation and typically assumes that compact, spherical shapes indicate a higher quality. An example for this is the Silhouette Width as a measure of how similar an observations is to its own cluster, compared to other clusters. It is defined as $(b(i) - a(i))/\max\{a(i), b(i)\}$, where $a(i)$ is the average distance of i to other points in its cluster and $b(i)$ is the lowest average distance of i to points in another cluster. The silhouette width is typically averaged over all observations to derive a single index.

An additional approach to assess cluster quality is external evaluation. If a true partition of the data is known a priori, one can compare the clustering to the known groups. As an example, the purity of a cluster describes the proportion of points that belong to the majority class in the cluster. In our scenario it is difficult to apply external evaluation since the chat is not labelled by default. However, we can infer labels based on the channel or game that is played. As an example, we can label each message by the channel name to see whether we can evaluate whether topics differ across channels.

4.3 Tuning

In order to find appropriate parameter configurations for the proposed algorithm we apply iterated racing (`irace`) [8]. `irace` samples new parameter configurations and iteratively biases the sampling towards better solutions. We evaluate the performance of a configuration by using prequential evaluation [4,7] with a horizon of length $h = 100$. The idea is to evaluate the current model with the next h examples in the stream. Afterwards, the same examples are incorporated into the model before repeating the same process with the next h points. In our scenario, we use `irace` to find the configurations that yield the highest average purity over all horizons. We search $r \in [0, 1]$, $\lambda \in [0, 1]$, $t_{gap} \in \{50, ..., 1000\}$ and utilize unigrams to reduce the complexity. Since the purity can be arbitrarily improved by increasing the number of micro-clusters, we enforce an upper limit by discarding configurations that produce more than 500 micro-clusters. Due to the high computational complexity of this step, we restrict the parameter optimization to the first 100.000 observations in the stream. However, we observed that the results are stable for the remaining part of the stream. For our setup with token fading, the best solution was found for $r = 0.3593$, $\lambda = 0.7682$, $t_{gap} = 392$. Without token fading the best solution was $r = 0.1902$, $\lambda = 0.7105$, $t_{gap} = 329$.

4.4 Results

The results of our analysis shows that both setups of our algorithm produce very pure clusters (Fig. 2). For micro-clusters, the median purity of both setups is above 96% and both reach perfect purity at times while never dropping below 84%. When reclustering the results to macro clusters, the purity naturally drops but its median remains above 80%. This result shows that chat messages from the same channel are grouped into the same cluster and channels are rarely mixed.

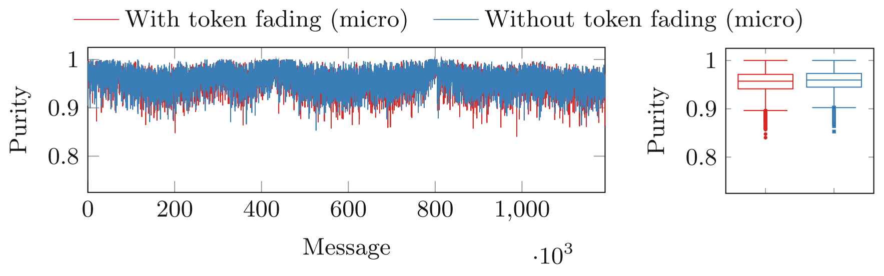


Fig. 2. Purity of clusters over time

Since the purity is very much influenced by the number of micro-clusters, we can observe that our parameter optimization yielded configuration that utilize close to 500 micro-clusters, i.e. our upper limit. Despite this strong compression of the original stream with 1.2 million messages, our algorithm retained enough information to group observations accurately.

When looking at the Average Silhouette Width (Fig. 3), we can observe a median width of around 0.3 indicating structures of medium quality. In peaks, the silhouette width can reach 0.8, indicating very strong structures but it can also drop to poor quality at times. As an example, we can observe severe decreases after 800.000 messages where the silhouette width decreases by almost 60%. This is likely explained by a swift change in topic, e.g. because channels stopped broadcasting.

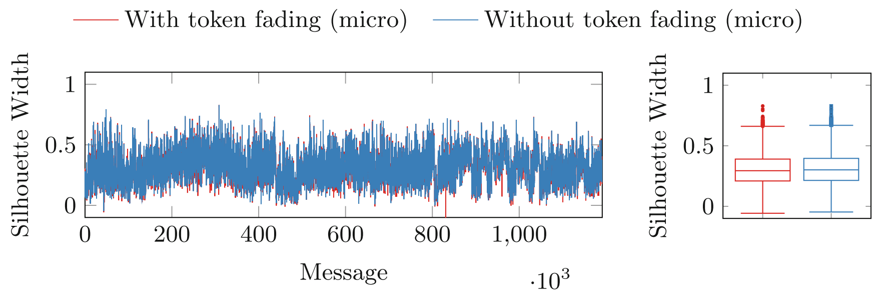


Fig. 3. Avg. silhouette width of clusters over time

All of the evaluation measures indicate that there is almost no difference in quality, when clustering was executed with or without n-gram fading. This observation supports the assumption that all the n-grams that were individually faded within a micro-cluster do not contribute to the clustering quality. In addition, our fading strategy requires less memory and reduces the median number of maintained n-grams by 78. However, we also observe that the λ parameter influences to what extent the TF vectors are reduced. For a high fading factor, the entire cluster will be removed before the effect of the n-gram fading can be observed.

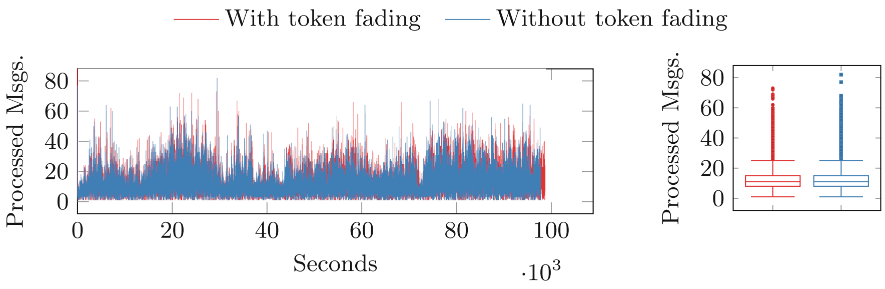
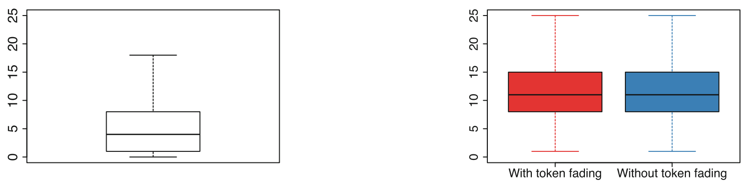


Fig. 4. Number of processed messages per second

Finally, we also evaluate number of messages that we were able to process every second on an Intel E5-2630 CPU with 2.2 Ghz (Fig. 4). On average, the algorithm was able to process 12 messages every second, however we observe many spikes in performance where almost 90 messages could be processed per second. Albeit fewer, we also observe the contrary, where performance slows down dramatically and only a single message is processed per second. The changes in processing speed can be explained by the varying complexity and length of chat messages. In order to evaluate whether our algorithm is able to process the data stream in real time, we compare the processing speed with the actual messages per second (Fig. 5). Even though there are times where the stream is faster than the processing speed, we observed that the average processing capability considerably surpasses the number of new messages in our data stream.



(a) Actual messages per second (b) Processed messages per second

Fig. 5. Comparison of arrival and processing speed

5 Conclusion and Outlook

The analysis of text streams poses many challenges due to its rapid and unstructured nature. However, finding patterns and segments in such unstructured data streams can yield valuable insights and findings. In this paper we proposed a new stream clustering algorithm for text data. It utilizes the common concept of a weighted Term Frequency (TF-IDF) in order to extract information from the text data. The algorithm can be useful to identify topics in an unbounded stream of text messages. Possible application scenarios include social media analysis or customer service, where the identification of common topics yields insights what people are discussing, e.g. a brand.

Our algorithm design is similar to traditional stream clustering algorithms where new observations are either added to their closest cluster or used to initialize new clusters. The algorithm incrementally maintains an occurrence count for a number of n-grams. This allows to calculate a TF-IDF vector from clusters in order to calculate the similarity between them. Due to our design, this calculation comes with little computational overhead or memory consumption. In addition, clusters are exponentially decayed over time in order to forget outdated data.

Future work should compare our algorithm to alternative stream clustering approaches and evaluate our algorithm on longer texts, e.g. email data. In addition, other reclustering approaches than hierarchical clustering could be investigated. Finally, means to automatically and adaptively choose the required parameters should be pursued in order to make the application of the algorithm easier.

References

1. Aggarwal, C.C.: Mining text and social streams. ACM SIGKDD Explor. Newsl. **15**(2), 9–19 (2014)
2. 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, VLDB 2003, vol. 29, pp. 81–92. VLDB Endowment, Berlin, Germany (2003)
3. Aggarwal, C.C., Yu, P.S.: On clustering massive text and categorical data streams. Knowl. Inf. Syst. **24**(2), 171–196 (2010)
4. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: Conference on Data Mining (SIAM 2006), pp. 328–339 (2006)
5. 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 (2017)
6. Hahsler, M., Bolanos, M., Forrest, J.: stream: Infrastructure for Data Stream Mining (2015). https://cran.r-project.org/web/packages/stream/index.html
7. Hahsler, M., Bolaños, M.: Clustering data streams based on shared density between micro-clusters. IEEE Trans. Knowl. Data Eng. **28**(6), 1449–1461 (2016)
8. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., Birattari, M.: The irace package: Iterated racing for automatic algorithm configuration. Oper. Res. Perspect. **3**, 43–58 (2016)
9. Silva, J.A., Faria, E.R., Barros, R.C., Hruschka, E.R., de Carvalho, A.C.O.L.F., Gama, J.: Data stream clustering: A survey. ACM Comput. Surv. **46**(1), 131–1331 (2013)
10. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: An efficient data clustering databases method for very large. In: ACM SIGMOD International Conference on Management of Data, vol. 1, pp. 103–114 (1996)