Sampling in Dirichlet Process Mixture Models for Clustering Streaming Data

Or Dinari
Ben-Gurion University

Oren Freifeld
Ben-Gurion University

Abstract

Practical tools for clustering streaming data must be fast enough to handle the arrival rate of the observations. Typically, they also must adapt on the fly to possible lack of stationarity; *i.e.*, the data statistics may be time-dependent due to various forms of drifts, changes in the number of clusters, *etc*. The Dirichlet Process Mixture Model (DPMM), whose Bayesian nonparametric nature allows it to adapt its complexity to the data, seems a natural choice for the streaming-data case. In its classical formulation, however, the DPMM cannot capture common types of drifts in the data statistics. Moreover, and regardless of that limitation, existing methods for online DPMM inference are too slow to handle rapid data streams. In this work we propose adapting both the DPMM and a known DPMM sampling-based non-streaming inference method for streaming-data clustering. We demonstrate the utility of the proposed method on several challenging settings, where it obtains state-of-the-art results while being on par with other methods in terms of speed.

1 INTRODUCTION

In today's Age of Data, the need for data-analysis methods that are both fast and effective is more important than ever. In this context, this work focuses on the challenging problem of streaming-data clustering; namely, the unsupervised-learning task at hand is to cluster observations where the latter arrive constantly at some rate, forming an ever-growing, possibly infinite, data stream. In this prevalent setting, traditional clustering methods are mostly inapplicable due to the following reasons. 1) Storing the entire stream in memory is often impractical or even impossible. 2) Even when storing the entire stream is possible, users usually cannot wait until the stream ends; rather, they often need, at any given moment, a reliable estimate of the model based on the data seen so far, and keep updating that estimate as more data arrives. 3) The statistical properties of the data might be non-stationary (Ramírez-Gallego et al., 2017) as, *e.g.*, clusters may appear or disappear, the distribution underlying a specific cluster may be time-dependent due to various forms of drift, *etc*. Thus, as new observations arrive, we must be able to update the model and adapt it to the newer data without having to revisit previously-seen observations.

Here we propose a new streaming-data clustering method based on the Dirichlet Process Mixture Model (DPMM) and, in particular, a specific non-streaming DPMM sampler (Chang and Fisher III, 2013). The DPMM (Ferguson, 1973; Antoniak, 1974) is a classical Bayesian Nonparametric (BNP) model (Hjort et al., 2010) which is often used in clustering problems where K, the number of clusters, is unknown (Müller et al., 2015). The DPMM's flexibility and adaptiveness to the data complexity support the inference of K (which is also the number of *instantiated* mixture components) together with the rest of the model. When clustering streaming data, the ability to modify K is an important property, sought after by most methods in this field. As the DPMM possesses that property, to some extent it can be used as is for streaming-data clustering. However, one problem that arises is that the DPMM cannot "forget" previously-seen observations, even those from the distant past. This is a limitation even with DPMM methods that do no need to keep all the data in memory (such as the Stochastic Online

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151. Copyright 2022 by the author(s).

Acknowledgements. This work was supported by the Lynn and William Frankel Center at BGU CS, by the Israeli Council for Higher Education via the BGU Data Science Research Center, and by Israel Science Foundation Personal Grant #360/21. O.D. was also funded by the Jabotinsky Scholarship from Israel's Ministry of Technology and Science, and by BGU's Hi-Tech Scholarship.

Variational Bayes (SoVB) (Hoffman et al., 2013) or the Memoized Variational Bayes (Hughes and Sudderth, 2013)); *e.g.*, when the number of observations grows larger and larger, new points have almost no effect on the estimated model. Another problem caused by the lack of forgetting is the inability to handle incremental concept drifts (Ramírez-Gallego et al., 2017): when the components change over time, a DPMM method will often opt to instantiate a new component, instead of modifying existing ones. Lastly, as DPMM inference (at least in most formulations) requires multiple passes over the data, it cannot scale (in terms of speed and memory) to long data streams and is inapplicable in the case of infinitely-long streams.

To adapt the DPMM for streaming data, we utilize an idea widely used in streaming-data clustering, the Damped Window (Zubaroğlu and Atalay, 2021) (*i.e.*, using finitely-many time-decaying weights). When applying this idea in the calculation of the DPMM's posterior distribution, we are able to not only better adapt to new data (thereby accommodating various concept drifts) but also obviate the need to reiterate over previously-processed observations. Concretely, we incorporate the Damped Window within a fast and recent implementation (Dinari et al., 2019) of Chang and Fisher's DPMM sampler (Chang and Fisher III, 2013) and eliminate the need to revisit, during inference, previously-seen data batches. Consequently, the sampler's speed improves to the point that it becomes on par with other streaming-data clustering methods. Additionally, we also introduce a deterministic subroutine (mostly based on predictive posterior distributions) that, when added to the sampler, significantly improves the performance in the streaming-data clustering task. A thorough experimental study (§ 5), shows that compared with several key methods, our method almost uniformly dominates in several common metrics.

To summarize, our two main contributions are: 1) a novel and fast streaming-data clustering method, called *ScStream*, that achieves state-of-the-art (SOTA) results and that was obtained by adapting a DPMM method to the streaming-data setting; 2) we created and/or adapted several datasets for evaluating clustering methods for streaming data. Finally, our Julia code and is available at `github.com/BGU-CS-VIL/DPMMSubClustersStreaming.jl` while its (optional) Python wrapper is available at `github.com/BGU-CS-VIL/dpmmpythonStreaming`.

2 RELATED WORK

Streaming-data clustering: Most methods for clustering streaming data use a two-phase process. The first phase is online during which new data is obtained, processed, and summarized. The second is offline, is typically called only upon request, and generates the clusters. An early example of such methods is BIRCH (Zhang et al., 1996) which introduced *micro-clusters* and *macro-clusters*. By maintaining a clustering features (CF) tree structure, whose nodes are called a micro-clusters, new points are assigned to micro-clusters based on feature similarity. A new leaf is created whenever the best similarity is insufficient. In the offline step the micro-clusters are clustered into so-called macro-clusters whose number is predefined. The macro-clusters then serve as a predicting model for labeling new data. CluStream (Aggarwal et al., 2003) improves BIRCH by allowing the clustering over different time horizons, storing not just the point summary but also time-dependent snapshots of the micro clusters. DenStream (Cao et al., 2006) extends BIRCH via a time-decaying CF (reducing the weight of older micro-clusters). BIRCH, together with CluStream and DenStream, inspired other works (*e.g.*: A-BIRCH (Lorbeer et al., 2017); ScaleKM (Bradley et al., 1998); ACSC (Fahy et al., 2018); HCluStream (Yang and Zhou, 2006); SDStream (Ren and Ma, 2009); C-DenStream (Ruiz et al., 2009); HDenStream (Lin and Lin, 2009); HCDD (Zgraja and Woźniak, 2018); LeaDen-Stream (Amini and Wah, 2013)) that utilize its key idea.

Another approach is to store cluster medoids (instead of the CF). An example is StreamKM++ (Ackermann et al., 2012), where a weighted subset (a coreset), of the data is stored in a tree. BICO (Fichtenberger et al., 2013) combines the StreamKM++ coreset approach with BIRCH's CF by storing the coresets in a tree structure, where each node is a CF. As an alternative for storing the medoids, some methods, such as STREAM (O'Callaghan et al., 2002), only store the clusters' centers. In competitive-learning stream algorithms, the centroids of the clusters evolve over time. Examples for such methods are SOStream (Isaksson et al., 2012), DBSTREAM (Hahsler et al., 2017), evoStream (Carnein and Trautmann, 2018) and G-Stream (Ghesmoune et al., 2014).

While the algorithms described so far represent different approaches, what they all have in common is that they are density-based. In another approach, based on partitioning the space using a grid, the macro-clusters are usually found by grouping together adjacent grid cells. In D-Stream (Chen and Tu, 2007), a fixed grid is used. ExCC (Bhatnagar and Kaur, 2007) lets the user set the grid boundaries and number of cells. Stats-Grid (Park and Lee, 2004) recursively splits the grid until the cells are sufficiently small. Some methods (*e.g.*, Amini et al. (2014)) combine the density- and grid-based approaches. For a thorough study of the methods above and more, see a recent review by Carnein and Trautmann (2019).

While most of the algorithms use the two-phase approach, there are several ones which are fully online. A prime example is Mini-Batch K-Means (Sculley, 2010), a very fast algorithm which, having adapted the classical K-Means (Lloyd, 1982; MacQueen et al., 1967) to a batched version, updates centroids using a step in a gradient-based direction (as opposed to a full recalculation as in K-Means). Adaptive Streaming K-Means (Puschmann et al., 2016) is another online approach that aims at handling concept drifts. Another online algorithm is pcStream (Mirsky et al., 2015) which uses Principal Component Analysis (PCA) to capture contexts in the data.

Online DPMM: While DPMM seems a natural candidate model for streaming-data clustering due to its innate ability of adapting its complexity to the data, efficient and scalable DPMM inference remains a great challenge. Moreover, its applicability to streaming data is not trivial. There have been a few works which adapted the DPMM to an online setting. To the best of our knowledge, all of them rely on variational inference. This is a key difference from our sampling-based method. SoVB (Hoffman et al., 2013) provides a framework which can be applied to BNP models, the DPMM included. In that version the data is processed in batches, and the model is updated according to a chosen learning rate. While the process in SoVB does not need to revisit previous batches, that method makes the strong assumption that the statistics are similar across the batches; our method does not suffer from this limitation. MemoizedVB (Hughes and Sudderth, 2013) is an online variational inference method intended for large datasets where the entire data does not fit in the memory. It processes the data is processed in batches and store the sufficient statistics of each cluster in each batch separately. MemoizedVB, however, must revisit each batch multiple times, so it is less relevant for streaming data. MVCL (Yang et al., 2019) is a similar method which extends MemoizedVB to continual learning with multiple datasets. That method too, however, is inapplicable in a stream setting as it requires revisiting batches. Lin (2013) proposed a learning algorithm for DPMM that requires only a single pass on the data and thus, theoretically, can operate in a streaming setting, at least on stationary data; however, it cannot adapt to concept drifts. This is also a limitation of the distributed method from (Campbell et al., 2015).

Campbell et al. (2013) propose a method that uses the Dependent Dirichlet Process (MacEachern, 1999) (an extension of the Dirichlet Process that supports evolving mixture models) for handling batch sequential data of an unknown number of evolving clusters. D-Means (Campbell et al., 2019) is a related BNP clustering algorithm for evolving linearly-separable spherical clusters based on small-variance asymptotic analysis. Unlike those two works above which are restricted to spherical Gaussian components, our implementation supports full-covariance Gaussians as well as multinomial components. More generally, our method supports any exponential family. Moreover, we are unaware of Dependent DPMM methods (let alone implementations) that scale to streaming data.

Scalable DPMM Samplers: While all the online DPMM methods above are variational, we leverage a fast sampling-based DPMM method and adapt it to streaming data. Concretely, Chang and Fisher III (2013) proposed a DPMM sampler (for non-streaming data), which, by using an augmented space and parallel sampling, allows for fast inference with convergence guarantees and that can perform large moves, escaping many (though not all) poor local maxima. A summary of their sampler is presented in § 3. More recently, Dinari et al. (2019) proposed a more efficient and even faster implementation of Chang and Fisher's sampler. Our work is largely based on those two works, which we have adapted into a streaming setting. Particularly, we are unaware of implementations of either variational or sampling-based DPMM inference that are (in the non-streaming case) as fast as Dinari et al. (2019), let alone ones that support full-covariance Gaussians and/or multinomials. Thus, that is the implementation we chose to modify. Importantly, while Dinari et al. (2019) is still not fast enough for streaming data, our proposed algorithmic and implementation changes eliminate this problem.

3 BACKGROUND

For simplicity, our presentation below assumes that all the random vectors involved have either a probability density function (pdf) or a probability mass function (pmf). One known DPMM construction is as follows:

$$\boldsymbol{\pi}|\alpha \sim \mathrm{GEM}(\alpha), \tag{1}$$

$$\theta_k|H \overset{i.i.d.}{\sim} f_\theta(H), \qquad \forall k \in \{1,2,\ldots\}, \tag{2}$$

$$z_i|\boldsymbol{\pi} \overset{i.i.d.}{\sim} \mathrm{Cat}(\boldsymbol{\pi}), \qquad \forall i \in \{1,2,\ldots,N\}, \tag{3}$$

$$\boldsymbol{x}_i|z_i,\theta_{z_i} \sim f_{\boldsymbol{x}}(\boldsymbol{x}_i;\theta_{z_i}), \qquad \forall i \in \{1,2,\ldots,N\}. \tag{4}$$

Here *i.i.d.* stands for independent and identically distributed, H is the *base measure*, f_θ is the pdf or pmf associated with H, the infinite-length vector $\boldsymbol{\pi} = (\pi_k)_{k=1}^{\infty}$ is drawn from the Griffiths-Engen-McCloskey stickbreaking process (GEM) (Pitman, 2002) with a concentration parameter $\alpha > 0$ (particularly, $\pi_k > 0$ for every k and $\sum_{k=1}^{\infty} \pi_k = 1$) while θ_k is drawn from f_θ. Each of the N *i.i.d.* observations $(\boldsymbol{x}_i)_{i=1}^N$ is generated by first drawing a label, $z_i \in \mathbb{Z}^+$, from $\boldsymbol{\pi}$ (*i.e.*, Cat is the categorical distribution), and then $\boldsymbol{x}_i$ is drawn from (a pdf or a pmf) $f_{\boldsymbol{x}}$ parameterized by θ_{z_i}. Loosely speaking, the DPMM entertains the notion of a mixture model of infinitely many components:

$$\boldsymbol{x}_i \overset{i.i.d.}{\sim} \sum\nolimits_{k=1}^{\infty} \pi_k f_{\boldsymbol{x}}(\boldsymbol{x}_i; \theta_k) \,. \tag{5}$$

Each $f_{\boldsymbol{x}}(\cdot;\theta_k)$ is called a *component* and we make no distinction between a component, $f_{\boldsymbol{x}}(\cdot,\theta_k)$, and its parameter, θ_k. The so-called labels $(z_i)_{i=1}^N$ encode the observation-to-component assignments. A cluster is a collection of points sharing a label; *i.e.*, $\boldsymbol{x}_i$ is in cluster k, denoted by C_k, if and only if $z_i = k$. Let (the random variable) K be the number of unique labels: $K = |\{k : z_i = k \text{ for some } i \in \{1, \dots, N\}\}|$; *i.e.*, K is also the number of clusters and is bounded above by N. Typically, and as assumed in this paper, H is chosen such that f_θ will be a conjugate prior (Gelman et al., 2013) to $f_{\boldsymbol{x}}$. The latent variables here are K, $(\theta_k)_{k=1}^{\infty}$, $\boldsymbol{\pi}$, and $(z_i)_{k=1}^N$. For more details (and other constructions), see Sudderth (2006).

We now briefly review a DPMM sampler proposed by Chang and Fisher III (2013). It consists of a restricted Gibbs sampler (Robert and Casella, 2013) and a split/merge framework which together form an ergodic Markov chain. The operations in each step of that sampler are highly parallelizable. Of note, the splits and merges let the sampler make *large moves* along the (posterior) probability surface as in such operations multiple labels change their label together to the same different label; this is in contrast to what happens, *e.g.*, in methods that change each label separately from the others. We now describe the essential details.

The augmented space. The latent variables, $(\theta_k)_{k=1}^{\infty}$, $\boldsymbol{\pi}$, and $(z_i)_{k=1}^N$, are augmented with auxiliary variables. For each component θ_k two subcomponents are added, $\bar{\theta}_{k,1}, \bar{\theta}_{k,2}$, with subcomponent weights $\bar{\boldsymbol{\pi}}_k = (\bar{\pi}_{k,1}, \bar{\pi}_{k,2})$. Implicitly, this means that every cluster C_k is augmented with two subclusters, $\bar{C}_{k,1}$ and $\bar{C}_{k,2}$. For each cluster label z_i, an additional *subcluster label*, $\bar{z}_i \in \{1,2\}$, is added; *i.e.*, subcluster $\bar{C}_{k,1} \subset C_k$ consists of all the points in C_k whose subcluster label is 1 ($\bar{C}_{k,2}$, is defined similarly). The goal of this auxiliary two-component mixture is to facilitate useful cluster splitting proposals (see below).

The restricted Gibbs sampler. This restricted sampler is not allowed to change (the current estimate of) K; rather, it can change only the parameters of the existing clusters and subclusters, and when sampling the labels, it can assign an observation only to an existing cluster. Note that for each instantiated component k, changing θ_k, $\bar{\theta}_{k,1}$, and $\bar{\theta}_{k,2}$ is done using $p(\theta_k|C_k;H)$, $p(\bar{\theta}_{k,1}|\bar{C}_{k,1};H)$, and $p(\bar{\theta}_{k,2}|\bar{C}_{k,2};H)$, respectively, where the latter three are the conditional distributions of the cluster or subcluster parameters given the cluster or subclusters. For more details about the restricted Gibbs sampler, see our **appendix**.

The split/merge framework. Splits and merges allow the sampler to change K using the Metropolis-Hastings framework (Hastings, 1970). Particularly, the auxiliary variables are used to propose splitting an existing cluster or merging two exiting ones. When a split is accepted, each of the newly-born clusters is augmented with two new subclusters. The Hastings ratio of a split is (Chang and Fisher III, 2013)

$$\mathrm{H}_{\text{split}} = \frac{\alpha \Gamma(N_{k,1}) f_{\boldsymbol{x}}(\bar{C}_{k,1};H) \Gamma(N_{k,2}) f_{\boldsymbol{x}}(\bar{C}_{k,2};H)}{\Gamma(N_k) f_{\boldsymbol{x}}(C_k;H)} \tag{6}$$

where Γ is the Gamma function, N_k, $N_{k,1}$ and $N_{k,2}$ are the number of points in C_k, $\bar{C}_{k,1}$ and $\bar{C}_{k,2}$, respectively, while $f_{\boldsymbol{x}}(C_k;H)$, $f_{\boldsymbol{x}}(\bar{C}_{k,1};H)$, and $f_{\boldsymbol{x}}(\bar{C}_{k,2};H)$ represent the *marginal* likelihood of C_k, $\bar{C}_{k,1}$ and $\bar{C}_{k,2}$ respectively. Concrete expressions for the marginal likelihood, in the case of Gaussian or Multinomial components (the component types considered in our experiments) appear in our **appendix**. Finally, a *merge* proposal is based on taking two existing clusters and proposing merging them into one. The corresponding Hastings ratio is $\mathrm{H}_{\text{merge}} = 1/\mathrm{H}_{\text{split}}$ where $\bar{C}_{k,1}$ and $\bar{C}_{k,2}$ are replaced with the two clusters, and C_k is replaced with the result of the merge. For a derivation of these ratios, see Chang and Fisher III (2013).

4 METHOD

Henceforth we will refer to the sampler from § 3 as the DPMM sampler. We base our method on that sampler but introduce important modifications of the latter in order to: 1) make it compatible with streaming data that arrives (possibly rapidly and/or indefinitely) batch by batch; 2) improve the clustering and the label consistency across batches.

4.1 Batches and Time-based Weighting

For each instantiated component k in the DPMM sampler, finding $p(\theta_k|C_k;H)$ requires computing sufficient statistics based on *all* of the points in C_k (similar logic applies to $(p(\bar{\theta}_{k,j}|\bar{C}_{k,j};H))_{j\in\{1,2\}}$). As the sufficient statistics are based on summation, it is tempting to try to use online updates by incrementally increasing the sums whenever a new batch arrives. However, the sufficient statistics often change with each iteration of the DPMM sampler (due to label changes). Thus, the

Table 1: Comparing our method (ScStream) with BIRCH Zhang et al. (1996), CluStream Aggarwal et al. (2003), D-Stream Chen and Tu (2007), DBSTREAM (Hahsler and Bolaños, 2016), StreamKM++ (Ackermann et al., 2012), Mini Batch K-Means (Sculley, 2010), pcStream (Mirsky et al., 2015), SoVB (Hoffman et al., 2013). Also included is DPMM sampler (Dinari et al., 2019). N/A indicates that a method did not scale enough or lacks support for the data type.

		BIRCH	CluStream†	D-Stream	DBSTREAM	StreamKM++†	Mini Batch K-Means†	pcStream	SoVB	ScStream (Ours)	DPMM Sampler
2D Gaussians	ARI:	.81 ± .12	.86 ± .11	.88 ± .16	.90 ± .11	.53 ± .11	.82 ± .09	.60 ± .12	.58 ± .11	**.93 ± .08**	.92 ± .14
	NMI:	.89 ± .04	.94 ± .03	.94 ± .05	.94 ± .04	.71 ± .05	.89 ± .03	.76 ± .07	.75 ± .05	**.95 ± .03**	.94 ± .10
	Purity:	.83 ± .06	**.94 ± .03**	.91 ± .09	.91 ± .06	.57 ± .05	.83 ± .05	.70 ± .08	.68 ± .06	.92 ± .05	.91 ± .10
	F-Measure:	.84 ± .10	.88 ± .09	.90 ± .13	.91 ± .09	.61 ± .08	.85 ± .08	.66 ± .10	.65 ± .09	**.94 ± .07**	.93 ± .11
	Full-NMI:	N/A	N/A	N/A	N/A	N/A	.48 ± .00	.37	.52	**.68 ± .01**	N/A
CoverType	ARI:	.07 ± .08	.10 ± .07	.07 ± .11	.10 ± .13	.09 ± .09	.07 ± .06	.03 ± .02	.10 ± .09	**.15 ± .11**	.10 ± .11
	NMI:	.14 ± .09	.19 ± .09	.19 ± .11	.18 ± .15	.15 ± .08	.13 ± .06	.20 ± .07	.13 ± .10	**.21 ± .14**	.16 ± .12
	Purity:	.66 ± .10	.71 ± .11	.70 ± .10	.68 ± .11	.68 ± .11	.66 ± .12	**.79 ± .08**	.66 ± .13	.71 ± .11	.67 ± .12
	F-Measure:	.44 ± .10	.33 ± .05	.58 ± .14	**.60 ± .13**	.42 ± .10	.37 ± .06	.11 ± .05	.48 ± .08	.47 ± .08	.48 ± .09
	Full-NMI:	N/A	N/A	N/A	N/A	N/A	.06 ± .01	.08	.01	**.13 ± .01**	N/A
ImageNet100	ARI:	.21 ± .11	.30 ± .13	N/A	.13 ± .15	.55 ± .15	.49 ± .17	.20 ± .09	.31 ± .18	**.63 ± .19**	.64 ± .28
	NMI:	.35 ± .11	.45 ± .09	N/A	.22 ± .17	.62 ± .09	.58 ± .12	.33 ± .08	.45 ± .20	**.69 ± .15**	.72 ± .24
	Purity:	.64 ± .12	.75 ± .12	N/A	.43 ± .13	**.91 ± .06**	.87 ± .09	.66 ± .10	.49 ± .13	.78 ± .14	.74 ± .22
	F-Measure:	.39 ± .08	.44 ± .10	N/A	.43 ± .09	.62 ± .14	.57 ± .15	.33 ± .09	.55 ± .11	**.73 ± .12**	.76 ± .17
	Full-NMI:	N/A	N/A	N/A	N/A	N/A	**.57 ± .02**	.26	.23	.48 ± .01	N/A
ImageNet1K	ARI:	N/A	.30 ± .14	N/A	.30 ± .16	N/A	.45 ± .12	.19 ± .07	.00 ± .02	**.62 ± .17**	N/A
	NMI:	N/A	.45 ± .10	N/A	.40 ± .14	N/A	.59 ± .07	.38 ± .06	.00 ± .02	**.68 ± .13**	N/A
	Purity:	N/A	.74 ± .14	N/A	.62 ± .13	N/A	**.97 ± .03**	.76 ± .08	.25 ± .04	.78 ± .13	N/A
	F-Measure:	N/A	.44 ± .09	N/A	.48 ± .11	N/A	.51 ± .12	.28 ± .09	.38 ± .04	**.72 ± .12**	N/A
	Full-NMI:	N/A	N/A	N/A	N/A	N/A	**.63 ± .01**	.30	.00	.41 ± .02	N/A
100D Multinomials	ARI:	N/A	.00 ± .01	N/A	.00 ± .00	.34 ± .24	.41 ± .24	N/A	.21 ± .14	**.78 ± .24**	.45 ± .22
	NMI:	N/A	.11 ± .05	N/A	.00 ± .00	.65 ± .16	.69 ± .16	N/A	.52 ± .14	**.89 ± .12**	.62 ± .30
	Purity:	N/A	.09 ± .03	N/A	.03 ± .00	.53 ± .25	.61 ± .25	N/A	.31 ± .15	**.84 ± .20**	.53 ± .25
	F-Measure:	N/A	.04 ± .01	N/A	.04 ± .01	.35 ± .24	.42 ± .24	N/A	.23 ± .13	**.78 ± .24**	.46 ± .22
	Full-NMI:	N/A	N/A	N/A	N/A	N/A	.54 ± .01	N/A	.27	**.72 ± .01**	N/A
20NewsGroup	ARI:	N/A	.00 ± .00	N/A	N/A	.01 ± .00	.01 ± .00	N/A	.06 ± .01	**.13 ± .01**	.12 ± .01
	NMI:	N/A	.12 ± .02	N/A	N/A	.07 ± .01	.09 ± .01	N/A	.20 ± .02	**.36 ± .03**	.33 ± .02
	Purity:	N/A	.13 ± .02	N/A	N/A	.11 ± .01	.12 ± .01	N/A	.13 ± .01	**.28 ± .02**	.24 ± .02
	F-Measure:	N/A	.10 ± .00	N/A	N/A	.10 ± .00	.09 ± .00	N/A	.14 ± .01	**.20 ± .01**	.19 ± .01
	Full-NMI:	N/A	N/A	N/A	N/A	N/A	.05 ± .01	N/A	.17	**.32 ± 0.03**	N/A

† Parametric methods given the true K.

online updates cannot be done without having all the batches that arrived so far stored in memory – but that becomes infeasible quickly as the stream grows. More generally, once a batch is processed it cannot be revisited again. Moreover, even if somehow, via an expensive and tedious bookkeeping, one could keep track of the ever-changing sufficient statistics, this would usually be a bad idea: once enough batches arrive new ones will hardly influence the sufficient statistics despite the fact that they, in the common case of non-stationarity, are more relevant than much older batches.

We address the no-revisits constraint and the non-stationarity as follows. Let $X_B = (\boldsymbol{x}_i)_{i=1}^{n_B}$ be a data batch at time B where n_B is the number of points in it. For $b \in \{1, \dots, B\}$, let n_k^b be the number of points in batch b assigned to cluster k, and let s_k^b be the sufficient statistics computed, for cluster k, based solely on those n_k^b points. Let $h_k^{1:B}$ denote the history record of sufficient statistics and counts for cluster k: $h_k^{1:B} = ((s_k^b, n_k^b))_{b=1}^{B}$. The subcluster history is similarly defined; *i.e.*, $\bar{h}_{k,j}^{1:B} = ((\bar{s}_{k,j}^b, \bar{n}_{k,j}^b))_{b=1}^{B}$ where $j \in \{1, 2\}$ and $\bar{}$ indicates subcluster-related quantities. Note that while per-batch sufficient statistics were also used in Hughes and Sudderth (2013) their method is inapplicable for streaming data as they must revisit batches. We now define the time-weighted sufficient statistics and count for cluster k:

$$S_k^B = \sum\nolimits_{b=q}^{B} \mathcal{K}(B,b) s_k^b \, , \; N_k^B = \sum\nolimits_{b=q}^{B} \mathcal{K}(B,b) n_k^b \quad (7)$$

where $\mathcal{K}(\cdot,\cdot) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}_{\geq 0}$ is a weighting function, $q = \min \{b : b \in \{1, \dots, B\}, \mathcal{K}(B,b) > \epsilon\}$, and $\epsilon > 0$ is a user-defined threshold (we used $\epsilon = 1e-08$ in all our experiments). The analogous subcluster quantities, (*i.e.*, $\bar{S}_{k,1}^b, \bar{S}_{k,2}^b$, $\bar{N}_{k,1}^b$, and $\bar{N}_{k,2}^b$) are defined similarly. Note that, in the summations above, usually $q > 1$. This limits the space/memory/time complexity, and implies we need to maintain the history records only up to a fixed maximal length; *i.e.*, for each batch B we update these records such that we keep the information from a previous batch b only if $\mathcal{K}(B,b) > \epsilon$ and discard it otherwise. As in many damped-window methods, we use $\mathcal{K}(B,b) = 2^{-\lambda(B-b)}$ where $\lambda > 0$ is user-defined (that said, other kernels may also be used).

Example 1 *Consider D-dimensional Gaussian components where the Normal Inverse Wishart (NIW) distribution serves as the base measure. Let* $(\kappa, \boldsymbol{m}, \nu, \boldsymbol{\Psi})$

denote the hyperparameters of the NIW prior. Let $X_b = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_{n_b})$ denote the data points (in $\mathbb{R}^D$) in batch b. Using a classical result (Gelman et al., 2013), the sufficient statistics here are

$$s_k^b = \left(\sum\nolimits_{\boldsymbol{x}_i \in X_b} \boldsymbol{x}_i \mathbb{1}_{z_i=k}, \sum\nolimits_{\boldsymbol{x}_i \in X_b} \boldsymbol{x}_i \boldsymbol{x}_i^T \mathbb{1}_{z_i=k}\right) , \quad (8)$$

where the indicator function $\mathbb{1}_{z_i=k}$ is 1 if $z_i = k$ and 0 otherwise. Using conjugacy (Gelman et al., 2013) as well as the replacement of the standard sufficient statistics and counts with their weighted versions, the hyperparameters of the NIW posterior for cluster k are:

$$\kappa_k^* = \kappa + N_k^B \ , \quad \nu_k^* = \nu + N_k^B \ ,$$
$$\boldsymbol{m}_k^* = \frac{1}{\kappa_k^*}\left(\kappa \boldsymbol{m} + \sum\nolimits_{b=q}^{B}\left[\mathcal{K}(B,b)\sum\nolimits_{\boldsymbol{x}_i \in X_b} \boldsymbol{x}_i \mathbb{1}_{z_i=k}\right]\right) ,$$
$$\boldsymbol{\Psi}_k^* = \frac{1}{\nu_k^*}\left(\nu \boldsymbol{\Psi} + \sum\nolimits_{b=q}^{B}\left[\mathcal{K}(B,b)\sum\nolimits_{\boldsymbol{x}_i \in X_b} \boldsymbol{x}_i \boldsymbol{x}_i^T \mathbb{1}_{z_i=k}\right]\right) \quad (9)$$

(note that together, the two nested sums in these equations constitute S_k^B). This fully defines the posterior distribution over the parameters of Gaussian k.

See the **appendix** for an analogous example for the case of multinomial components. More generally, out method applies to components from any exponential family when used with its conjugate prior. As we will show in § 4.2, to determine z_i, our method uses the predictive posterior distribution, $p(\boldsymbol{x}_i|H, S_k^B, N_k^B, z_i = k)$ (where we replaced the standard sufficient statistics and number of points with their weighted versions) as the latter induces, via proportionality, the predicted probability of observation $\boldsymbol{x}_i$ to belong to cluster k:

$$p(z_i{=}k|\boldsymbol{x}_i, H, S_k^B, N_k^B) \propto p(\boldsymbol{x}_i|H, S_k^B, N_k^B, z_i{=}k) \, . \quad (10)$$

Example 2 *Continuing Example 1, in the Gaussian case with an NIW prior, Eq. (10) becomes (Chang, 2014)*

$$p(z_i{=}k|\boldsymbol{x}_i, H, S_k^B, N_k^B) \propto \pi_k \overbrace{p(\boldsymbol{x}_i|\kappa_k^*, \boldsymbol{m}_k^*, \nu_k^*, \boldsymbol{\Psi}_k^*, z_i{=}k)}^{p(\boldsymbol{x}_i|H,S_k^B,N_k^B,z_i=k)}$$
$$= t_{\nu_k^*-D+1}\left(\boldsymbol{x}_i; \boldsymbol{m}_k^*, \frac{\kappa_k^*+1}{\kappa_k^*(\nu_k^*-D+1)}\nu_k^*\boldsymbol{\Psi}_k^*\right) \quad (11)$$

*where $t(\cdot)$ is Student's t-distribution (see the **appendix** for the multinomial case).*

4.2 The Proposed Algorithm

Before diving into the proposed algorithm, which is a novel extension of the DPMM sampler, let us consider the underlying model behind it. In the stationary case where all the data points are drawn from the same DPMM, the algorithm almost coincides with the DPMM sampler (especially if λ is high). A more interesting insight is the following. Hu et al. (2015) showed how to incorporate prior knowledge into a DPMM. When our method processes batch B, it can be viewed as DPMM inference with such an incorporated prior knowledge. Specifically, the prior knowledge consists of the K instantiated clusters where the prior knowledge of each θ_k ($k \in \{1, \dots, K\}$) is captured via the posterior distribution over θ_k implied by H and $h_k^{1:B-1}$.

With this in mind, we proceed to describe the proposed algorithm, summarized in Algorithm 1 (which uses Algorithm 2 as its main subroutine). Let $\boldsymbol{X} = (X_B)_{B\in\{1,2,\dots\}}$ be a possibly-infinite data stream, where each X_B is a batch of n_B data points. One of our proposed modifications is that throughout the entire run of the algorithm, instead of using the standard sufficient statistics and point number, we use Eq. 7 which in turn is based on the history records from § 4.1.

Upon the arrival of X_B, we run the restricted Gibbs sampler on it for T iterations, where each iteration is followed by allowing splits/merges as in the DPMM sampler. If $B = 1$, we set $T = \infty$, meaning we run it till convergence (which is *very* fast since batches are small (*e.g.*, $n_B = 10^3$). If $B > 1$ then we use $T = 1$ (*i.e.*, a single iteration). Either way, after those T iterations, we perform an additional iteration (again followed by proposing and accepting/rejecting splits and merges stochastically), but this time replace the restricted Gibbs sampler with a deterministic routine (lines 6–11 in Algorithm 2) based on 1) modes (namely, the argmax of the relevant distributions) instead of sampling, and, more importantly, 2) the predictive posterior distributions of the labels and subcluster labels.

Concretely, Chang and Fisher III's restricted sampler (see § 3) determines the labels as follows: it draws $\boldsymbol{\pi}$ and $(\bar{\boldsymbol{\pi}}_k)_{k=1}^K$ from their respective conditional Dirichlet distributions, and, for each (instantiated) component k, draws $\theta_k \sim p(\theta_k|C_k; H)$, $\bar{\theta}_{k,1} \sim p(\bar{\theta}_{k,1}|\bar{C}_{k,1}; H)$, and $\bar{\theta}_{k,2} \sim p(\bar{\theta}_{k,2}|\bar{C}_{k,2}; H)$. Next, it uses these drawn parameters and weights to construct, for each $\boldsymbol{x}_i$, a likelihood-based pmf and then draws z_i from it:

$$z_i \overset{\sim}{\propto} \pi_k f_{\boldsymbol{x}}(\boldsymbol{x}_i; \theta_k, z_i = k) \quad (12)$$

(where $\overset{\sim}{\propto}$ denotes sampling proportional to the right-hand side of the equation). In contrast, our deterministic subroutine: 1) updates $\boldsymbol{\pi}$ and $(\bar{\boldsymbol{\pi}}_k)_{k=1}^K$ using the mode of each of their respective conditional Dirichlet distributions; 2) avoids sampling $(\theta_k, \bar{\theta}_{k,1}, \bar{\theta}_{k,2})$ and, instead of computing a likelihood-based pmf, computes a pmf based on the predictive posterior distribution (Eq. (10)) and uses its argmax to determine z_i:

$$z_i = \underset{k\in\{1,\dots,K\}}{\arg\max}\ \pi_k p(\boldsymbol{x}_i|H, S_k^B, N_k^B, z_i = k) \, . \quad (13)$$

While the transition from sampling to argmax only

Algorithm 1: ScStream

```
    Input: $H$, $\alpha$, $\mathcal{K}$, $\epsilon$,$T$
    Data: Stream $\boldsymbol{X}$
1   $X_1 \leftarrow \boldsymbol{X}.next$
2   $C_1 \leftarrow X_1$
3   $K \leftarrow 1$
4   Randomly partition $C_1$ into subclusters $C_{1,1}$ and
      $C_{1,2}$
5   $q \leftarrow 1$
6   Extract $h_1^{1:1} = (s_1^1, n_1^1)$, $\bar{h}_{1,1}^{1:1} = (\bar{s}_{1,1}^1, \bar{n}_{1,1}^1)$ and
      $\bar{h}_{1,2}^{1:1} = (\bar{s}_{1,2}^1, \bar{n}_{1,2}^1)$ from $(C_1, C_{1,1}, C_{1,2})$
7   $\mathcal{M} \leftarrow (h_1^{1:1}, \bar{h}_{1,1}^{1:1}, \bar{h}_{1,2}^{1:1})$
8   while Not Converged do
9       $K, \mathcal{M} \leftarrow$ Algorithm 2$(X_1; H, \alpha, K, \mathcal{K}, \infty, \_, q, B, \mathcal{M})$
10  while $X_B \leftarrow \boldsymbol{X}.next$ do
11      $(h_k^{q:(B-1)}, \bar{h}_{k,1}^{q:(B-1)}, \bar{h}_{k,2}^{q:(B-1)})_{k=1}^K \leftarrow \mathcal{M}$
12      $q \leftarrow \min\{b : b \in \{1, \ldots, B\}, \mathcal{K}(B, b) > \epsilon\}$
13      $\mathcal{M} \leftarrow (h_k^{q:B-1}, \bar{h}_{k,1}^{q:B-1}, \bar{h}_{k,2}^{q:B-1})_{k=1}^K$
14      for $t = 1 : T + 1$ do
15          $K, \mathcal{M} \leftarrow$ Algorithm 2$(X_B; H, \alpha, \ldots, t, q, B, \mathcal{M})$
16      Yield $\mathcal{M}$
```

slightly improves the stability of the results, the use of the predictive posterior leads to usually-drastic improvements in the quality and the inter-batch consistency of the predicted labels (see § 5). The high label consistency of our method is in sharp contrast to many streaming-data clustering methods (especially those with an offline reclustering step) which usually suffer from significant label switching.

Unsurprisingly, the running time grows linearly with T. Thus, the choice of T seemingly suggests a trade-off between performance and speed. However, we empirically found (see the **appendix**) that *the improvement in performance is only sublinear* and that, in practice, a *very* low value of T usually suffices. Thus, we suggest using $T = 1$. That said, if the stream is slow enough, one may benefit from using a larger T.

Remark. It is, in fact, possible to use $T = 1$ even on X_1 since even in this case, as the stream progresses the method will eventually reach good performance. However, this will hurt the results in (only) the first few batches of the stream.

Summarizing the Main Differences from the DPMM Sampler. First and foremost, our use of (a finite-length history of) weighted batched sufficient statistics is key here. Apart from the fact that it allows us to handle concept drifts gracefully (unlike the original sampler, which assumes that the data statistics are stationary) it also allows us to visit each batch only once. The second main difference is our deterministic subroutine: although it plays a less important role than

Algorithm 2: Iteration of the Modified DPMM Sampler

```
    Input: $H$, $\alpha$, $K$, $\mathcal{K}$,$T$,$t$, $q$,$B$,
           $\mathcal{M} = (h_k^{q:B}, (\bar{h}_{k,j}^{q:B})_{j \in \{1,2\}})_{k=1}^K$
    Output: $K'$,$\mathcal{M}'$
    Data: $X_B$
1   if $t < T + 1$ then
2       $(h_1^{q:B}, \bar{h}_{1,1}^{q:B}, \bar{h}_{1,2}^{q:B}) \leftarrow \mathcal{M}$
3       Compute $(S_k^B)_{k=1}^K$ and $(N_k^B)_{k=1}^K$ using Eq. (7)
4       1 iteration of the restricted sampler from § 3
          using $(S_k^B)_{k=1}^K$ and $(N_k^B)_{k=1}^K$ (see appendix
          for details)
5   else
6       $\boldsymbol{\pi} \leftarrow \left( \frac{N_1^B}{\sum_{k=1}^K N_k^B + \alpha}, \ldots, \frac{N_K^B}{\sum_{k=1}^K N_k^B + \alpha}, \frac{\alpha}{\sum_{k=1}^K N_k^B + \alpha} \right)$
7       for $k \in \{1, \ldots, K\}$ do
8           $\bar{\boldsymbol{\pi}}_k \leftarrow \left( \frac{\frac{\alpha}{2} + \bar{N}_{k,1}^B}{\alpha + \sum_{s=\{1,2\}} \bar{N}_{k,s}^B}, \frac{\frac{\alpha}{2} + \bar{N}_{k,2}^B}{\alpha + \sum_{s=\{1,2\}} \bar{N}_{k,s}^B} \right)$
9       for $x_i \in X_B$ do
10          $z_i \leftarrow \arg\max_{k \in \{1,\ldots,K\}} \pi_k p(z_i = k | \boldsymbol{x}_i, H, S_k^B, N_k^B)$
11          $\bar{z}_i \leftarrow \arg\max_{j \in \{1,2\}} \bar{\pi}_{z_i} p(\bar{z}_i = j | \boldsymbol{x}_i, H, \bar{S}_{z_i,j}^B, \bar{N}_{z_i,j}^B)$
12  for $k \in \{1, \ldots, K\}$ do
13      Extract $(s_k^B, n_k^B)$, $(\bar{s}_{k,1}^B, \bar{n}_{k,1}^B)$ and $(\bar{s}_{k,2}^B, \bar{n}_{k,2}^B)$
          (from $C_k$, $\bar{C}_{k,1}$ and $\bar{C}_{k,2}$, respectively) and
          update $(h_k^{q:B}, \bar{h}_{k,1}^{q:B}, \bar{h}_{k,2}^{q:B})$ accordingly
14  for $k \in \{1, \ldots, K\}$ do
15      Propose splitting $C_k$ to its subsclusters and
          accept the split with probability $\min(1, H_{\text{split}})$
          (Eq. (6))
16  for $k, k' \in \{1, \ldots, K\}$ do
17      Propose merging $C_k$ and $C_{k'}$ and accept the
          merge with probability $\min(1, H_{\text{merge}})$
18  $\mathcal{M}' \leftarrow (h_k^{q:B}, (\bar{h}_{k,j}^{q:B})_{j \in \{1,2\}})_{k=1}^{K'}$ where $K'$ is the
      new number of clusters
```

the first change (as we show in § 5, even without that subroutine our method already achieves SOTA results) it allows our method to have consistent labels across batches, a feat most methods are incapable of.

5 RESULTS

For evaluation, we have chosen several datasets of varying difficulty levels and compared with multiple methods. Some of these datasets are known while the ones we created/modified are accessible from our code repository. We chose the component type in our method according to the data type: Gaussian components for $\mathbb{R}^D$-valued data and multinomial components for discrete count data. As in Dinari et al. (2019), our code can run on either a single multithreaded process, or be

Table 2: Running time (in seconds)

	BIRCH	CluStream	D-Stream	DBSTREAM	StreamKM++	Mini Batch K-Means	pcStream	SoVB	ScStream (Ours)	DPMM Sampler
2D Gaussians	112.5	31.3	24.7	17.0	15.4	1.4	1020.7	53.3	22.9	589.5
CoverType	95.8	45.9	1723.8	12.1	25.5	0.8	1610.3	115.6	6.1	254.8
ImageNet100	57.9	66.7	N/A	65.2	242.7	12.0	15.7	100.5	23.1	1039.9
ImageNet1K	N/A	1454	N/A	814	N/A	148	195	9219	1005	N/A
100D Multinomials	N/A	44.7	N/A	12.9	25.5	0.8	N/A	115.6	23.5	254.8
20NewsGroup	N/A	71.9	N/A	N/A	61.1	0.2	N/A	3.1	12.7	122.6

distributed across processes and/or machines. Here, in our experiments, due to the small batch size we have chosen the former configuration.

Methods. Due to the abundance of existing algorithms, we focused on several popular methods for clustering streaming data: BIRCH (Zhang et al., 1996), CluStream (Aggarwal et al., 2003), D-Stream (Chen and Tu, 2007), DBSTREAM (Hahsler and Bolaños, 2016), StreamKM++ (Ackermann et al., 2012), Mini-Batch K-Means (Sculley, 2010) and pcStream (Mirsky et al., 2015). That choice was based on their problem settings, popularity, and available software (taken from Hahsler et al. (2017); Bifet et al. (2011); Pedregosa et al. (2011)). In addition, we have compared with SoVB (Hoffman et al., 2013), as implemented by Hughes and Sudderth (2014). We also compare our method with the DPMM sampler (Chang and Fisher III, 2013), using its faster reimplementation (Dinari et al., 2019). For fairness of comparison, we have tuned each of the methods on each of the datasets using available black-box optimizers. The parameters of each method were tuned on the first 10 batches of each stream, setting the mean adjusted rand index (ARI) as the objective function. The R-based methods were tuned using the *irace* R package (López-Ibáñez et al., 2016). The tuning for pcStream (Mirsky et al., 2015) was done using the *black-box* (Knysh and Korkolis, 2016) Python package. For the DPMM-related methods we tuned the hyperparameters using the *BlackBoxOptim* Julia package. Note that some of the methods (BIRCH; CluStream; Mini-Batch Kmeans; StreamKM) are parametric, *thus they were always provided with the true number of clusters*, while others (the DPMM sampler; DBSTREAM; D-Stream; pcStream; ours) infer the number of clusters; this gives an unfair advantage to the former over the latter.

Datasets with points in $\mathbb{R}^D$. We created a synthetic dataset with 10^7 points in $\mathbb{R}^2$ divided into 20 clusters, where the points of each cluster were drawn from a different Gaussian. Next, we have inserted an incremental concept drift (Ramírez-Gallego et al., 2017) to that dataset, meaning that the clusters moved, independently of each other, as the stream progressed. In addition, we have used two real datasets: CoverType (Blackard and Dean, 1999) (which is often used for evaluating streaming-data clustering methods), in which each observation provides a point in $\mathbb{R}^{10}$ (describing 30 squared meters of forest). The second real dataset we used is ImageNet's (Deng et al., 2009) train set, where we initially extracted features using SWAV (Caron et al., 2020), and then used them in two different settings. In the first one we have used a subset of 100 classes (out of 1000) and used PCA to reduce the dimension to 64. In the second setting we have used the full dataset, and used PCA to reduce the dimension to 128. In both of the ImageNet settings we have added a recurring concept drift (Ramírez-Gallego et al., 2017). In all of the three real dataset settings we have normalized the data by subtracting the mean of each feature and dividing by its standard deviation.

Datasets with count data. We have created a synthetic dataset of 10^7 100-dimensional points divided into 100 clusters, where the points in each cluster were drawn from a multinomial distribution. Next, we have inserted gradual concept drift (Ramírez-Gallego et al., 2017) to that dataset. For a real dataset we have used the 20newsgroup (Lang, 1995), where each sample is a news article in one out of twenty possible subjects. We have used only the 1000 most commons words for classification in each sample.

Note that not all the methods were evaluated on all of the datasets. The reason is that some methods do not support count data and/or do not scale (*e.g.* did not finish in comparable time or ran out of memory) to high dimensions.

Evaluation. In all the experiments we used a fixed batch size of 1000 points. In order to evaluate the performance of each model we have used several popular metrics. Before processing each batch we have used the current model to predict the labels of the batch, and only then updated the model according to the new data. The only exception for this was the (unmodified) DPMM sampler, where we have clustered each batch separately and used the results as the labels. The metrics we used on the predicted labels were ARI, Normalized Mutual Information (NMI), Purity and Pairwise F-Measure, where in all cases we report

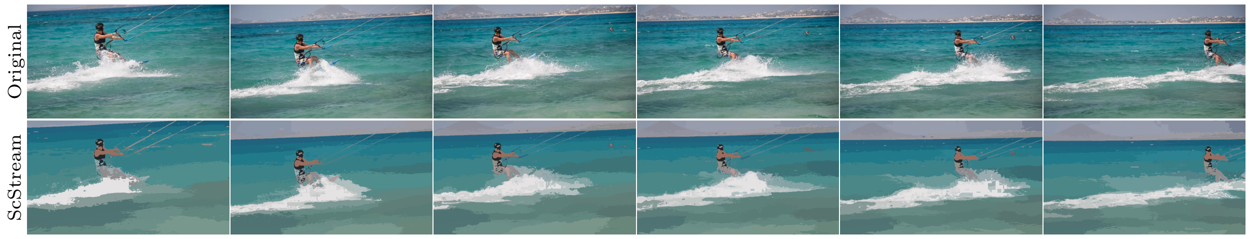


Figure 1: Select frames from a video-segmentation task. Results shown for ScStream (which inferred 80 clusters). For comparison, see also additional results of a competing method (MiniBacth K-Means) in the **appendix**.

the mean result across all of the batches. In addition, for the methods that support it, we have checked the consistency of the labels between batches, comparing the predictions for the entire dataset and the true labels. For this we used Full NMI; *i.e.*, concatenating all the predictions (across the batches) and comparing the result, via NMI, to the true labels. Table 1 summarizes the results. Our method almost uniformly outperforms the other streaming-data clustering methods. The exceptional metrics are Purity and Full NMI in the high-K cases, as there the unfair advantage we gave the parametric methods is especially significant. See the **appendix** for the box plots of the different metrics, revealing additional information (beyond the mean+std. dev.). Table 2 shows that our running time is on-par with most methods, except Mini-Batch K-means which is the fastest.

The deterministic subroutine. We have repeated the ImageNet100 experiment, but this time without using that subroutine (and instead used an additional iteration of the unmodified restricted Gibbs sampler). The results, in Table 3, show clear benefits from using the proposed subroutine.

Video Temporal Segmentation. Image (non-semantic) segmentation is a popular computer-vision task, where an image is partitioned into several different segments. Here we consider a simple case where the segmentation is based on each pixel's color and location. However, to demonstrate the utility of our streaming-data method, we apply it to video segmentation (as opposed to a single-image segmentation). In this experiment, each batch is a video frame. The resolution of each RGB frame in the 'kite-surf' video (taken from the DAVIS dataset (Perazzi et al., 2016)) was 480×854. This implies that each batch was of size $N = 409920$ and $D = 5$ (3 colors channels plus the 2D location of each pixel). The video contains 50 frames, so in total we had $20.4M$ 5D samples. Each frame took ScStream 0.6 [sec] to process (including I/O). In this quantitative experiment, the pros of using our method stand out. For example. unlike many other methods (such as DBSTREAM, Birch, *etc.*), we do not have label switching, thus it is possible to have temporally-consistent clusters. In contrast, methods that suffer from label switching cannot guarantee inter-frame consistency. Figure 1 shows example results of our method. Results of another method that does not suffer from label switching, MiniBatch K-means (Sculley, 2010), appear in the **appendix**. However, even if we ignore that fact that MiniBatch K-means is parametric (so it must be given the value of K), it cannot handle concept drifts; *e.g.*, as the figure in the appendix shows, each cluster remains in a similar spatial location between frames, and the surfer is barely distinguishable from the background. However, ScStream solves these problems as is evident by the fact that the surfer is clearly distinguishable from the background, and his label is consistent across the frames, despite the fact that his associated clusters change their statistics over time.

	Another sampling iter.	Proposed subroutine
ARI	$0.57 \pm .18$	$\mathbf{.63 \pm .19}$
NMI	$0.65 \pm .20$	$\mathbf{.69 \pm .15}$
Full NMI	$0.43 \pm .02$	$\mathbf{.48 \pm .01}$

Table 3: Using the deterministic subroutine as opposed to another sampling iteration. Data: ImageNet100 experiment.

6 CONCLUSION

We have proposed a new BNP method, called ScStream, for clustering streaming data. ScStream, a streaming-data extension of a recent fast implementation of a (non-streaming) DPMM sampler, is fast and achieves SOTA results. While ScStream supports any exponential family for the component type, it cannot handle arbitrary-shaped clusters; this is arguably its key limitation. For simplicity, our presentation here assumed that the batches arrive at times $\{1, 2, 3, \ldots\}$; however, our method is general enough to support arrivals at any monotonically-increasing time sequence (*i.e.*, $t_1 < t_2 < \ldots$) and our code already supports this generality. The only difference in such a case is that $\mathcal{K}(t_B, t_b)$ is used instead of $\mathcal{K}(B, b)$. Our publicly-available code is easy to use and offers the user an interface in either Julia or Python.

References

M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler. Streamkm++ a clustering algorithm for data streams. *Journal of Experimental Algorithmics*, 2012. 2, 1, 5

C. C. Aggarwal, S. Y. Philip, J. Han, and J. Wang. A framework for clustering evolving data streams. In *Proceedings 2003 the Very Large Data Bases conference*. Elsevier, 2003. 2, 1, 5

A. Amini and T. Y. Wah. Leaden-stream: A leader density-based clustering algorithm over evolving data stream. *Journal of Computer and communications*, 2013. 2

A. Amini, H. Saboohi, T. Y. Wah, and T. Herawan. A fast density-based clustering algorithm for real-time internet of things stream. *The Scientific World Journal*, 2014. 2

C. E. Antoniak. Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *The annals of statistics*, 1974. 1

V. Bhatnagar and S. Kaur. *Exclusive and Complete Clustering of Streams*. Springer Berlin Heidelberg, 2007. 2

A. Bifet, G. Holmes, B. Pfahringer, J. Read, P. Kranen, H. Kremer, T. Jansen, and T. Seidl. Moa: a real-time analytics open source framework. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2011. 5

J. A. Blackard and D. J. Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 1999. 5

P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *KDD*. AAAI Press, 1998. 2

T. Campbell, M. Liu, B. Kulis, J. P. How, and L. Carin. Dynamic clustering via asymptotics of the dependent Dirichlet process mixture. In *NeurIPS*, 2013. 2

T. Campbell, J. Straub, J. F. III, and J. How. Streaming, distributed variational inference for Bayesian nonparametrics. In *NeurIPS*, 2015. 2

T. Campbell, B. Kulis, and J. How. Dynamic clustering algorithms via small-variance analysis of Markov chain mixture models. *IEEE TPAMI*, 2019. 2

F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SIAM international conference on data mining*, 2006. 2

M. Carnein and H. Trautmann. evostream — evolutionary stream clustering utilizing idle times. *Big Data Research*, 2018. 2

M. Carnein and H. Trautmann. Optimizing data stream representation: An extensive survey on stream clustering algorithms. *Business and Information Systems Engineering (BISE)*, 2019. 2

M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *NeurIPS*, 2020. 5

J. Chang. *Sampling in computer vision and Bayesian nonparametric mixtures*. PhD thesis, Massachusetts Institute of Technology, 2014. 2, E

J. Chang and J. W. Fisher III. Parallel sampling of DP mixture models using sub-cluster splits. In *NeurIPS*, 2013. 1, 2, 3, 3, 5, 3, 10

Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007. 2, 1, 5

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 5

O. Dinari, A. Yu, O. Freifeld, and J. Fisher III. Distributed MCMC inference in Dirichlet process mixture models using Julia. In *IEEE CCGRID Workshop on High Performance Machine Learning*, 2019. 1, 2, 1, 5

C. Fahy, S. Yang, and M. Gongora. Ant colony stream clustering: A fast density clustering algorithm for dynamic data streams. *IEEE Transactions on Cybernetics*, 2018. 2

T. S. Ferguson. A Bayesian analysis of some nonparametric problems. *The Annals of Statistics*, 1973. 1

H. Fichtenberger, M. Gillé, M. Schmidt, C. Schwiegelshohn, and C. Sohler. Bico: Birch meets coresets for k-means clustering. In *European symposium on Algorithms*. Springer, 2013. 2

A. Gelman, H. S. Stern, J. B. Carlin, D. B. Dunson, A. Vehtari, and D. B. Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 2013. 3, 1, 1, F, F

M. Ghesmoune, H. Azzag, and M. Lebbah. G-stream: Growing neural gas over data stream. In C. K. Loo, K. S. Yap, K. W. Wong, A. Teoh, and K. Huang, editors, *International Conference on Neural Information Processing*. Springer International Publishing, 2014. 2

M. Hahsler and M. Bolaños. Clustering data streams based on shared density between micro-clusters. *IEEE Transactions on Knowledge and Data Engineering*, 2016. 1, 5

M. Hahsler, M. Bolanos, and J. Forrest. Introduction to stream: An extensible framework for data stream clustering research with R. *Journal of Statistical Software*, 2017. 2, 5

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. 1970. 3

N. L. Hjort, C. Holmes, P. Müller, and S. G. Walker. *Bayesian nonparametrics*. Cambridge University Press, 2010. 1

M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 2013. 1, 2, 1, 5

L. Hu, J. Li, X. Li, C. Shao, and X. Wang. Tsdpmm: Incorporating prior topic knowledge into dirichlet process mixture models for text clustering. In *Conference on Empirical Methods in Natural Language Processing*, 2015. 4.2

M. C. Hughes and E. B. Sudderth. Memoized online variational inference for Dirichlet process mixture models. In *NeurIPS*, 2013. 1, 2, 4.1

M. C. Hughes and E. B. Sudderth. Bnpy: Reliable and scalable variational inference for bayesian nonparametric models. In *NeurIPS Workshop on Probabilistic Programming*, 2014. 5

C. Isaksson, M. H. Dunham, and M. Hahsler. *SOStream: Self Organizing Density-Based Clustering over Data Stream*. Springer Berlin Heidelberg, 2012. 2

P. Knysh and Y. Korkolis. Blackbox: A procedure for parallel optimization of expensive black-box functions. *arXiv:1605.00998*, 2016. 5

K. Lang. Newsweeder: Learning to filter netnews. In *ICML*, 1995. 5

D. Lin. Online learning of nonparametric mixture models via sequential variational approximation. *NeurIPS*, 2013. 2

J. Lin and H. Lin. A density-based clustering over evolving heterogeneous data stream. In *ISECS International Colloquium on Computing, Communication, Control, and Management*, 2009. 2

S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 1982. 2

M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 2016. 5

B. Lorbeer, A. Kosareva, B. Deva, D. Softić, P. Ruppel, and A. Küpper. *A-BIRCH: Automatic Threshold Estimation for the BIRCH Clustering Algorithm*. Springer International Publishing, 2017. 2

S. N. MacEachern. Dependent nonparametric processes. In *ASA proceedings of the section on Bayesian statistical science*, 1999. 2

J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, 1967. 2

Y. Mirsky, B. Shapira, L. Rokach, and 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. 2, 1, 5

P. Müller, F. A. Quintana, A. Jara, and T. Hanson. *Bayesian nonparametric data analysis*. Springer, 2015. 1

L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *The International Conference on Data Engineering*, 2002. 2

N. H. Park and W. S. Lee. Statistical grid-based clustering over data streams. *SIGMOD Rec.*, 2004. 2

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2011. 5

F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *CVPR*, 2016. 5

J. Pitman. Combinatorial stochastic processes. Technical report, Technical Report 621, Dept. Statistics, UC Berkeley. Lecture notes, 2002. 3

D. Puschmann, P. Barnaghi, and R. Tafazolli. Adaptive clustering for dynamic iot data streams. *IEEE Internet of Things Journal*, 2016. 2

S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, and F. Herrera. A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing*, 2017. 1, 5

J. Ren and R. Ma. Density-based data streams clustering over sliding windows. In *International Conference on Fuzzy Systems and Knowledge Discovery*, 2009. 2

C. Robert and G. Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013. 3

C. Ruiz, E. Menasalvas, and M. Spiliopoulou. *C-DenStream: Using Domain Knowledge on a Data Stream*. Springer Berlin Heidelberg, 2009. 2

D. Sculley. Web-scale k-means clustering. In *WWW*, 2010. 2, 1, 5, 5

E. B. Sudderth. *Graphical models for visual object recognition and tracking*. PhD thesis, Massachusetts Institute of Technology, 2006. 3

C. Yang and J. Zhou. HClustream: A novel approach for clustering evolving heterogeneous data stream. In *ICDM Workshops*, 2006. 2

Y. Yang, B. Chen, and H. Liu. Memorized variational continual learning for Dirichlet process mixtures. *IEEE Access*, 2019. 2

J. Zgraja and M. Woźniak. Drifted data stream clustering based on clustree algorithm. In *International Conference on Hybrid Artificial Intelligence Systems*. Springer, 2018. 2

T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. *ACM sigmod record*, 1996. 2, 1, 5

A. Zubaroğlu and V. Atalay. Data stream clustering: a review. *Artificial Intelligence Review*, 2021. 1

Supplementary Material: Sampling in Dirichlet Process Mixture Models for Clustering Streaming Data

Abstract

This documents contains the following:

1. an additional figure for the video segmentation experiment;
2. additional Boxplots for our main experiment, omitted from the paper due to space limits.
3. an empirical verification that our runtime grows linearly with T (*i.e.*, the number of iterations of the restricted Gibbs sampler) and that a *very low* T suffices for good results;
4. the details of a single iteration in the original restricted Gibbs sampler (Chang and Fisher III, 2013);
5. the expressions for the marginal likelihood for Gaussian and multinomials components;
6. the posterior calculations in the multinomial case(the Gaussian case was already included in our paper);
7. the predictive posterior in the multinomal case (the Gaussian case was already included in our paper);
8. the full details of our experiments hyper-params and machine specification.

Proceedings of the 25th International Conference on Artificial Intelligence and Statistics (AISTATS) 2022, Valencia, Spain. PMLR: Volume 151. Copyright 2022 by the author(s).

A Additional Video Segmentation Results

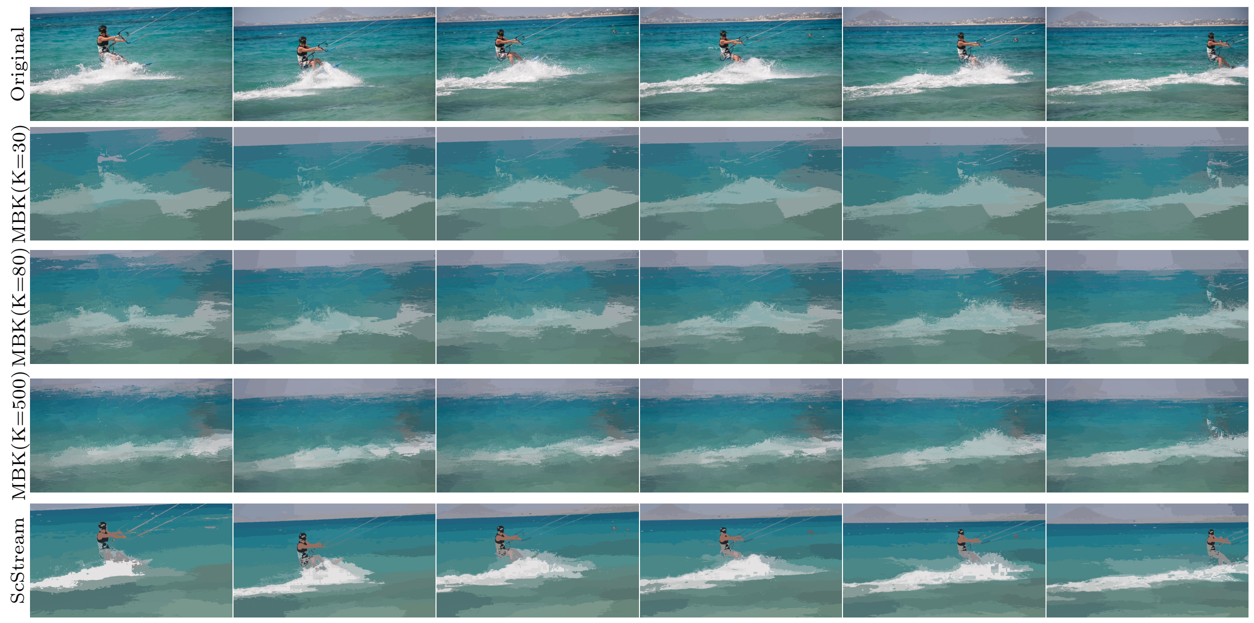
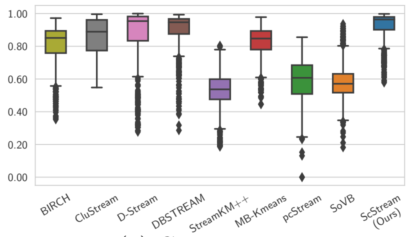


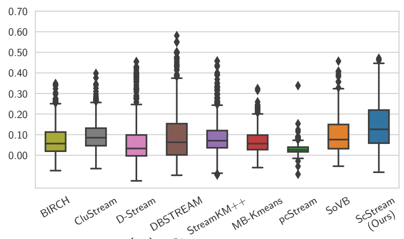
Figure 2: Video segmentation (example frames). Results shown for MiniBatch-Kmeans (denoted as MBK) with several different K values, as well as for ScStream (which inferred 80 clusters).

B Boxplots

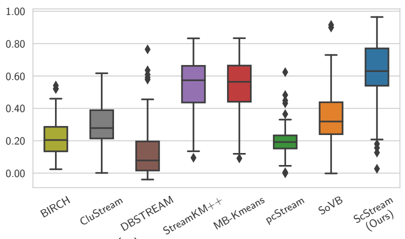
Here we provide boxplots for the ARI, NMI, Purity and F-measure metrics (for the same experiments as in the paper). Recall that for some of these metrics, the parametric methods (namely, methods that need to know K) enjoy an unfair advantage as they were given the true value of K. The parametric methods are: BIRCH; CluStream; StreamKM++; MB-Kmeans.



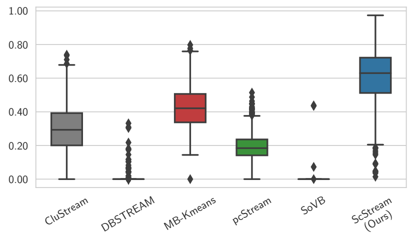
(a) Gaussian 2D



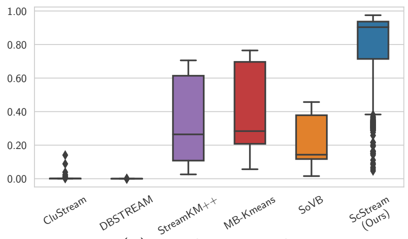
(b) CoverType



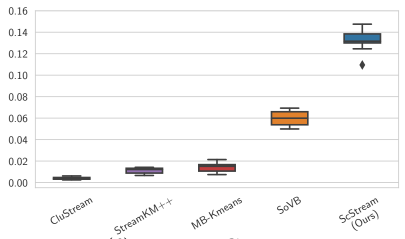
(c) ImageNet100



(d) ImageNet1K

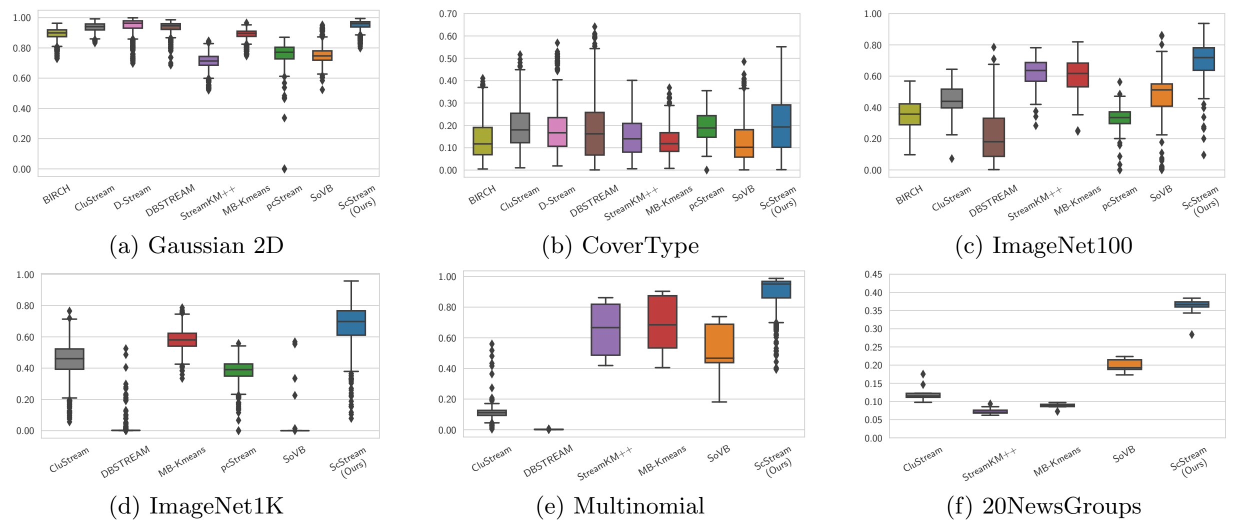


(e) Multinomial



(f) 20NewsGroups

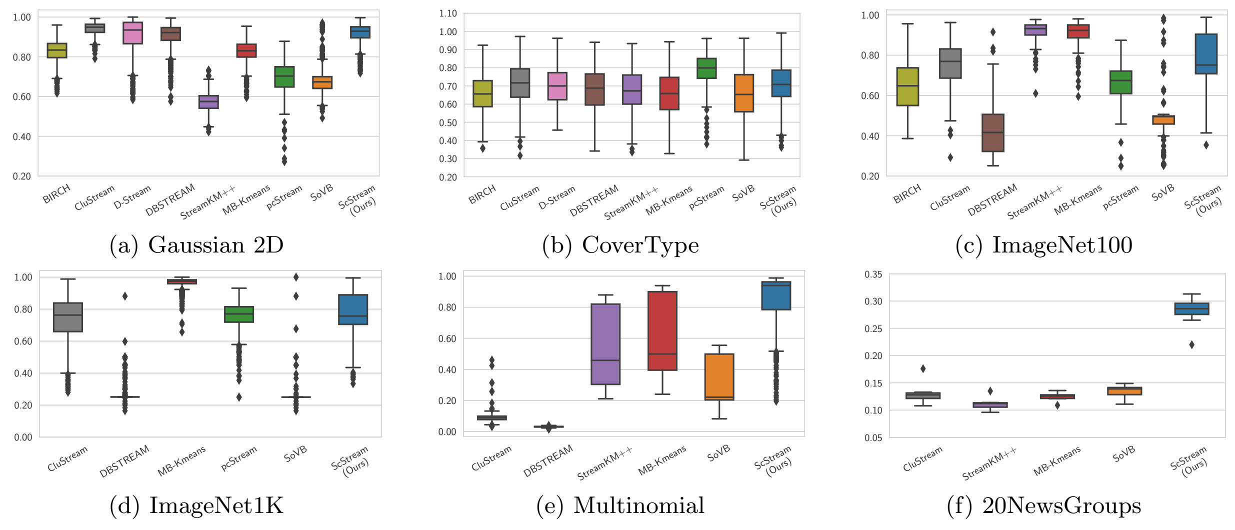
Figure 3: Box plots of the ARI metric for each of the experiments.



(a) Gaussian 2D (b) CoverType (c) ImageNet100

(d) ImageNet1K (e) Multinomial (f) 20NewsGroups

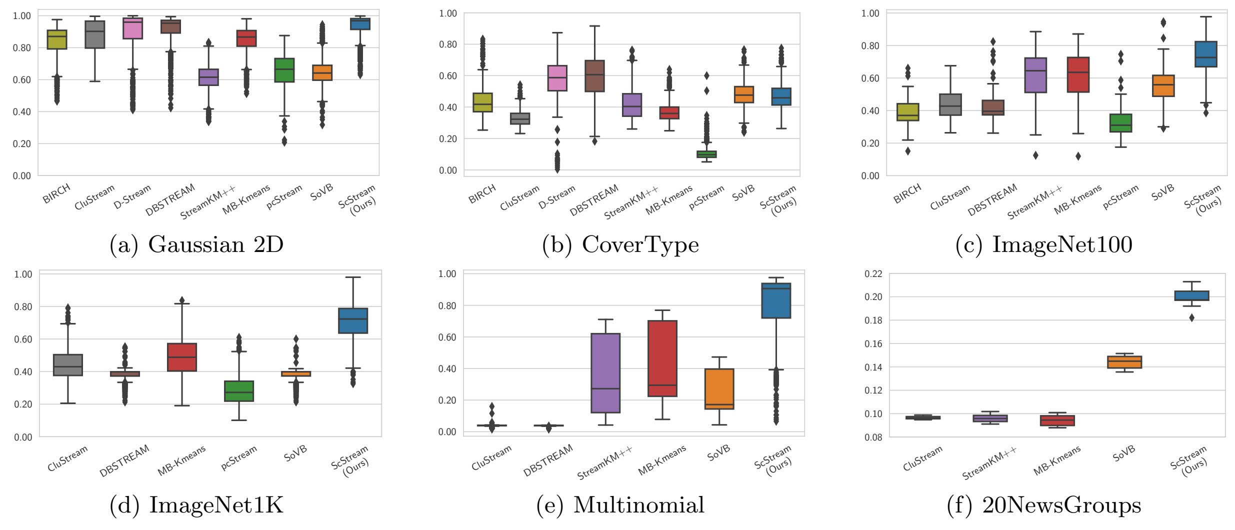
Figure 4: Box plots of the NMI metric for each of the experiments.



(a) Gaussian 2D (b) CoverType (c) ImageNet100

(d) ImageNet1K (e) Multinomial (f) 20NewsGroups

Figure 5: Box plots of the Purity metric for each of the experiments.



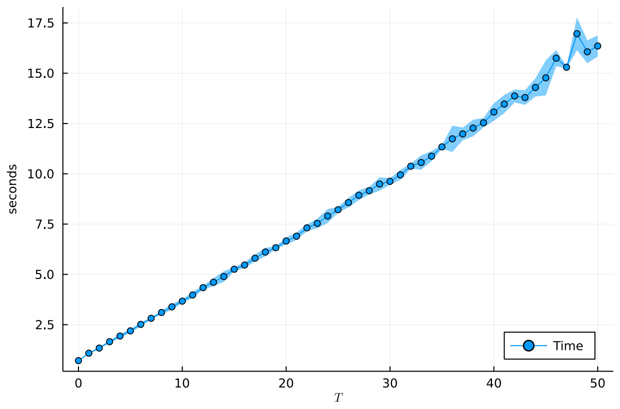
(a) Gaussian 2D (b) CoverType (c) ImageNet100

(d) ImageNet1K (e) Multinomial (f) 20NewsGroups

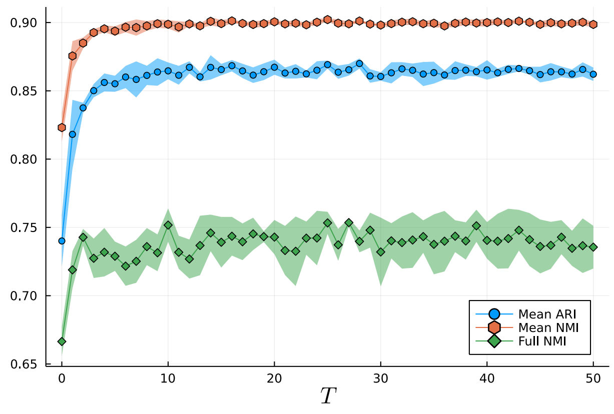
Figure 6: Box plots of the Pairwise F-measure metric for each of the experiments.

C An Empirical Verification that the Runtime Grows Linearly with T (*i.e.*, the Number of Iterations of the Restricted Gibbs Sampler) and that a Very Low T Suffices for Good Results

Recall that T is the number of iterations for which we run the restricted Gibbs sampler on each batch. Note that T should not be confused with the arrival time of the batch (we used b, not T, to denote the the batch index). To empirically validate that runtime grows linearly with T, we ran our method on a synthetic Gaussian dataset with 10^5 observations sampled from 20 overlapping components where, in addition, we have inserted both incremental and gradual concept drifts to the data. The results, in terms of running time and performance as functions of T, appear in Figure 7. It is



(a) Running time as a function of T.



(b) Performance as a function of T.

Figure 7: Running time and performance as functions to T

observable that while the performance gain initially increases with T, a plateau is quickly reached. Thus, further increasing T will result in smaller and smaller gains. The runtime, however, is linear with T, thus using a low T is usually preferred. Particularly, note that $T = 1$ already achieves good results. Therefore, all the experiments in the paper were done with $T = 1$.

D Iteration of the Restricted Gibbs Sampler

Algorithm 3: Iteration of the Original Restricted Gibbs Sampler (Chang and Fisher III, 2013)

```
Input: $H$, $\alpha$, $K$, $\boldsymbol{N}$, $(S_k, \bar{S}_{k,1}, \bar{S}_{k,2}, N_k, \bar{N}_{k,1}, \bar{N}_{k,2})_{k=1}^{K}$
Output: $(z_i, \bar{z}_i)_{i=1}^{N}$
Data: $X$
1  Draw weights by $\boldsymbol{\pi} \sim \mathrm{Dir}(N_1, \dots, N_k, \alpha)$    // a $K+1$ dimensional Dirichlet Distribution
2  for $k \in \{1, \dots, K\}$ do
3      Draw subcluster weights by $\bar{\boldsymbol{\pi}}_k \sim \mathrm{Dir}(\bar{N}_{k,1} + \frac{\alpha}{2}, \bar{N}_{k,2} + \frac{\alpha}{2})$    // a 2-dimensional Dirichlet
        Distribution
4  for $k \in \{1, \dots, K\}$ do
5      Draw cluster $\theta_k$ parameters by $\theta_k \sim f_\theta(\theta_k; S_k, N_k, H)$
6      Draw subcluster $\bar{\theta}_{k,1}$ parameters by $\bar{\theta}_{k,1} \sim f_\theta(\bar{\theta}_{k,1}; \bar{S}_{k,1}, \bar{N}_{k,1}, H)$
7      Draw subcluster $\bar{\theta}_{k,2}$ parameters by $\bar{\theta}_{k,2} \sim f_\theta(\bar{\theta}_{k,2}; \bar{S}_{k,2}, \bar{N}_{k,2}, H)$
8  for $i \in \{1, \dots, N\}$ do
9      Draw cluster label for $\boldsymbol{x}_i$ with $p(z_i = k | x_i, \pi_k, \theta_k) \propto \pi_k f_{\boldsymbol{x}}(\boldsymbol{x}_i; \theta_k, z_i = k)$
10     Draw subcluster label for $\boldsymbol{x}_i$ with $p(\bar{z}_i = j | x_i, \bar{\pi}_{k,j}, \bar{\theta}_{k,j}) \propto \bar{\pi}_{k,j} f_{\boldsymbol{x}}(\boldsymbol{x}_i; \bar{\theta}_{k,j}, \bar{z}_i = j)$
```

Algorithm 3 (using a notation that is consistent with the one in our paper) is a single iteration of the restricted Gibbs sampler from Chang and Fisher III (2013).

E Marginal Likelihoods

For full derivation of the well-known results below, see Chang (2014).

E.1 Gaussian

Let X be the points in the cluster, let $N = |X|$ be the number of points, and let D be the dimension of each point. Let $(\kappa, \boldsymbol{m}, \nu, \boldsymbol{\Psi})$ denote the NIW prior hyperparameters, and let $(\kappa^*, \boldsymbol{m}^*, \nu^*, \boldsymbol{\Psi}^*)$ denote the posterior hyperparameters, as calculated in Example 1 in the main paper. The marginal likelihood of X under the above parameters is:

$$p(X) = \frac{\Gamma_D(\nu^*)|\nu\psi|^{\frac{\nu}{2}}}{\pi^{\frac{ND}{2}}\Gamma_D(\frac{\nu}{2})|\nu^*\psi^*|^{\frac{\nu^*}{2}}} \left(\frac{\kappa}{\kappa^*}\right)^{\frac{D}{2}} \tag{14}$$

where $\Gamma_D(\cdot)$ is the multivariate Gamma function of dimension D and $|\cdot|$ is the determinant.

E.2 Multinomial

Let X be the points in the cluster, and let $N = |X|$ be the number of points. Each point in

$$\boldsymbol{x}_i = \begin{bmatrix} x_{i1} & \dots & x_{iD} \end{bmatrix} \in X \tag{15}$$

is a D-length histogram, where x_{ij} entry in it corresponds to how many times outcome j was observed in the i-ith experiment. Let $\mathrm{Dir}(d_1, \dots, d_D)$ be the Dirichlet-distribution prior and let $A = \sum_{j=1}^{D} d_j$. The marginal likelihood of X is:

$$p(X) = \frac{N!\Gamma(A)}{\prod_{j=1}^{D}(x_{ij}!)\Gamma(A+N)} \prod_{j=1}^{D} \frac{d_j + x_{ij}}{\Gamma(d_j)} . \tag{16}$$

F Multinomial Posterior Calculation

Let $X_b = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_{n_b})$ denote the data points (in $\mathbb{Z}_{\geq 0}^{D}$) in batch b. Using a classical result (Gelman et al., 2013), the sufficient statistics here are

$$s_k^b = \left(\sum\nolimits_{\boldsymbol{x}_i \in X_b} \boldsymbol{x}_i \mathbb{1}_{z_i = k}\right) . \tag{17}$$

Let $(d_1, \ldots, d_D)$ be the hyperparameters for the Dirichlet distribution prior and its hyperparameters. By conjugacy (Gelman et al., 2013), and when using our time-weighted sufficient statistics, the hyperparameters of the posterior are:

$$(d_1^*, \ldots, d_D^*) = (d_1, \ldots, d_D) + \sum\nolimits_{b=q}^{B} \left[\mathcal{K}(B, b) \sum\nolimits_{\boldsymbol{x}_i \in X_b} \boldsymbol{x} \mathbb{1}_{z_i = k} \right] . \tag{18}$$

G Predictive Posterior of the Multinomial Distribution

Following the notation in § E.2, and letting $\text{Dir}(d_1^*, \ldots, d_D^*)$ denote the Dirichlet-distribution *posterior* and $A^* = \sum_{j=1}^{D} d_j^*$ denote the sum of the posterior hyperparameters, the predictive posterior for the multionmial distribution is the following DirMult distribution:

$$\text{DirMult}(\boldsymbol{x}_i; d_1^*, \ldots, d_D^*) = \frac{(\sum_{j=1}^{D} x_{ij})!}{\prod_{j=1}^{D} x_{ij}!} \frac{\Gamma(A^*)}{\Gamma(A^* + \sum_{j=1}^{D} x_{ij})} \prod_{j=1}^{D} \frac{\Gamma(d_j^* + x_{ij})}{\Gamma(d_j^*)} \tag{19}$$

where $\boldsymbol{x}_i$ is a single sample $\boldsymbol{x}_i = (x_{i_1}, \ldots, x_{i_D})$. Conveniently, when conditioning on k (and adding the component weight) we can drop multiplicative constants from the RHS of the equation above, simplifying the expression:

$$p(z_i{=}k | \boldsymbol{x}_i, H, S_k^B, N_k^B) \propto \pi_k p(\boldsymbol{x}_i | \boldsymbol{d}_k^* = (d_1^*, \ldots, d_D^*), z_i = k) = \pi_k \frac{\Gamma(A^*)}{\Gamma(A^* + \sum_{j=1}^{D} x_{ij})} \prod_{j=1}^{D} \frac{\Gamma(d_j^* + x_{ij})}{\Gamma(d_j^*)} \tag{20}$$

H Experiments Details

Machine Spec: All the experiments were done on an Ubuntu 20.4 machine with an Intel® Core™ i9-11900K Processor. For Julia we used version 1.6, for Python version 3.8 and for R version 4.1.1.

H.1 Experiment Parameters

All the hyperparameters, for each of the competing methods and each of the datasets, were tuned by black-box optimizers; see the paper for details. All the parametric models were given the true value of K. Here we provide the values of optimized parameters (found by the black-box optimizers) which were used in each of the experiments.

Gausian 2D:
DBSTREAM: $\lambda = 0.01, r = 0.5768, Cm = 0.39244, \alpha = 0.2709, gaptime = 2971$.
D-Stream: $\lambda = 0.01, gridsize = 0.9561, gaptime = 4931, Cm = 2.92, Cl = 2.6721$.
CluStream: $\lambda = 0.01, t = 4$.
BIRCH: $\lambda = 0.01, threshold = 1.9339, branching = 2, maxLeaf = 26$.
StreamKM: $\lambda = 0.01, sizeCoreset = 1000$.
PcStream: $driftThreshold = 0.74308542, percentVarience = 4.66021295, maxDriftSize = 250.13468314 \cdot 2$.
SoVB:$\kappa = 1, \boldsymbol{m} = zeros(2), \nu = 4, \Psi = \mathbb{I} \cdot 1.02, rhoexp = 0.55, \alpha = 1.0, rhodelay = 1$.
ScStream: $\lambda = 1, \kappa = 1, \boldsymbol{m} = zeros(2), \nu = 4, \Psi = \mathbb{I} \cdot 1.02, \alpha = 1.0, \epsilon = 1e-08$.

CoverType:
DBSTREAM: $\lambda = 0.01, r = 0.7715, Cm = 2.748, alpha = 0.173, gaptime = 597$.
D-Stream: $\lambda = 0.01, gridsize = 0.6142, gaptime = 4490, Cm = 2.143, Cl = 1.533$.
CluStream: $\lambda = 0.01, t = 3$.
BIRCH: $\lambda = 0.01, treshold = 1.4011, branching = 1, maxLeaf = 32$.
StreamKM: $\lambda = 0.01, sizeCoreset = 1000$.
PcStream: $driftThreshold = 0.80786776, percentVarience = 3.5408475, maxDriftSize = 15.0128437 \cdot 10$.
SoVB:$\kappa = 1, \boldsymbol{m} = zeros(10), \nu = 25, \Psi = \mathbb{I} \cdot 0.78, rhoexp = 0.55, \alpha = 1.0, rhodelay = 1$.
ScStream: $\lambda = 0.2, \kappa = 1, \boldsymbol{m} = zeros(10), \nu = 25, \Psi = \mathbb{I} \cdot 0.78, \alpha = 1.0, \epsilon = 1e-08$.

ImageNet100:
DBSTREAM: $\lambda = 0.01, r = 2.5092, Cm = 1.2568, alpha = 0.118, gaptime = 3528$.
D-Stream: $\lambda = 0.01, gridsize = 0.6216, gaptime = 2924, Cm = 2.7936, Cl = 1.8295$.
CluStream: $\lambda = 0.01, t = 3$.
BIRCH: $\lambda = 0.01, treshold = 1.9445, branching = 3, maxLeaf = 5$.
StreamKM: $\lambda = 0.01, sizeCoreset = 1000$.
PcStream: $driftThreshold = 1.23666252, percentVarience = 1.50457986, maxDriftSize = 9.50255966 \cdot 64$.
SoVB:$\kappa = 1, \boldsymbol{m} = zeros(64), \nu = 562, \Psi = \mathbb{I} \cdot 0.51, rhoexp = 0.55, \alpha = 1.0, rhodelay = 1$.

ScStream: $\lambda = 0.2, \kappa = 1, \boldsymbol{m} = zeros(64), \nu = 562, \Psi = \mathbb{I} \cdot 0.51, \alpha = 1.0, \epsilon = 1e - 08$.

ImageNet1000:
DBSTREAM: $\lambda = 0.01, r = 1.8247, Cm = 1.675, alpha = 0.1798, gaptime = 454$.
CluStream: $\lambda = 0.01, t = 5$.
PcStream: $driftThreshold = 1.39396418, percentVarience = 1.91681159, maxDriftSize = 4.78852301 \cdot 128$.
SoVB:$\kappa = 1, \boldsymbol{m} = zeros(128), \nu = 834, \Psi = \mathbb{I} \cdot 0.177, rhoexp = 0.55, \alpha = 1.0, rhodelay = 1$.
ScStream: $\lambda = 0.2, \kappa = 1, \boldsymbol{m} = zeros(128), \nu = 834, \Psi = \mathbb{I} \cdot 0.177, \alpha = 1.0, \epsilon = 1e - 08$.