confStream: Automated Algorithm Selection and Configuration of Stream Clustering Algorithms

Matthias Carnein[1(✉)], Heike Trautmann[1], Albert Bifet[2], and Bernhard Pfahringer[2]

[1] University of Münster, Münster, Germany
{carnein,trautmann}@wi.uni-muenster.de
[2] University of Waikato, Hamilton, New Zealand
{abifet,bernhard}@waikato.ac.nz

Abstract. Machine learning has become one of the most important tools in data analysis. However, selecting the most appropriate machine learning algorithm and tuning its hyperparameters to their optimal values remains a difficult task. This is even more difficult for streaming applications where automated approaches are often not available to help during algorithm selection and configuration. This paper proposes the first approach for automated algorithm selection and configuration of stream clustering algorithms. We train an ensemble of different stream clustering algorithms and configurations in parallel and use the best performing configuration to obtain a clustering solution. By drawing new configurations from better performing ones, we are able to improve the ensemble performance over time. In large experiments on real and artificial data we show how our ensemble approach can improve upon default configurations and can also compete with a-posteriori algorithm configuration. Our approach is considerably faster than a-posteriori approaches and applicable in real-time. In addition, it is not limited to stream clustering and can be generalised to all streaming applications, including stream classification and regression.

Keywords: Stream clustering · Data streams · Automated Machine Learning · Algorithm configuration · Algorithm selection

1 Introduction

Over the past decades, machine learning has revolutionised many of its application areas. However, due to the abundance of machine learning algorithms and application scenarios, it is often necessary to select an algorithm which is most suited for a given problem. In addition, machine learning algorithms tend to be very sensitive to their configuration and it is important to tune hyperparameters to their optimal values, which can be difficult even for experienced users. A first approach that tries to alleviate the user from this is Automated

I. S. Kotsireas and P. M. Pardalos (Eds.): LION 14 2020, LNCS 12096, pp. 80–95, 2020.
https://doi.org/10.1007/978-3-030-53552-0_10

Machine Learning [18]. It attempts to make design decisions such as the selection and configuration of machine learning algorithms automatically. Unfortunately, many of these automated approaches require multiple passes over the data and cannot adapt to changes over time. This makes them infeasible for any online or streaming application. However, many of today's data sources are data streams due to the widespread usage of sensors, the internet-of-things and social media. In this paper, we address the problem of automated algorithm selection and configuration of stream clustering algorithms which aim to maintain clusters over time in a stream of observations.

By using an ensemble of different algorithms and configurations, we are able to adapt the optimal algorithm and its hyperparameter settings over time. Our approach is not limited to stream clustering but can be applied to all streaming scenarios, including stream classification and regression. In our experiments, we use several state-of-the-art stream clustering algorithms. On multiple real and artificial data streams, we show that our ensemble-approach always performs better than the default configuration. Even compared to offline and a-posteriori configuration approaches it produces competitive results, while being much faster and applicable in real-time.

2 Background

2.1 Stream Clustering

Clustering is a popular tool for pattern recognition and is often used in marketing or network analysis. However, a major drawback of traditional clustering is that it requires a fixed data set. Whenever new data becomes available, the entire analysis needs to be repeated. This is time consuming, undesirable and often infeasible when working with data streams. An approach to solve this is stream clustering which is able to cluster a continuous and possibly infinite stream of observations. In stream clustering, relevant information is usually extracted into so called *micro-clusters* before discarding an observation. The micro-clusters are then "reclustered" into the final *macro-clusters* upon the user's request. A survey of stream clustering algorithms is available in [10].

Unfortunately, (stream) clustering algorithms usually require many hyperparameters to be set a-priori [10]. For example, typical implementations of `DenStream` [6] have 8 hyperparameters [5] ranging from distance and weight thresholds to window sizes. In practice, these settings are often difficult and unintuitive to choose. With streaming data, the hyperparameters also need to be adapted over time as the data changes. In this paper, we aim to automatically select the best algorithm and its optimal configuration over time.

2.2 Automated Machine Learning

Automated Machine Learning (AutoML) attempts to make the design decision in machine learning automatically [18]. For example, it tries tune the hyperparameters of algorithms automatically or select the most appropriate algorithms. Popular approaches in AutoML are `irace` [20], `SMAC` [16] or `ParamILS` [17].

`irace` for example can be used for automated algorithm configuration. It uses a racing procedure where configurations that perform statistically worse are removed after every race. New parameter configurations are drawn according to probability distributions and the sampling is biased towards better performing configurations. Unfortunately, the racing procedure makes it difficult to apply `irace` for streaming applications. However, we can draw some inspiration from the parameter sampling for our streaming case.

First ideas for Automated Algorithm Selection in the streaming scenario can be found in the stream *classification* [21–24], stream *regression* [25] and *online* learning [13,14] literature. First attempts used an ensemble approach where a meta-classifier periodically predicts the most suitable algorithm based on the stream's characteristics [23]. Similarly, the `BLAST` algorithm [22,24] also uses an ensemble of algorithms, but simply selects the best algorithm of the last window as the *active* classifier. In the following, we use the same idea as an inspiration to select and configure stream *clustering* algorithms.

In a first proof-of-concept [11], we already propose an ensemble approach for automated algorithm *configuration* of stream *clustering* algorithms, called `confStream`. For this, we use an ensemble of different configurations. Periodically, the clustering quality for every configuration is evaluated. Based on the observed performance, a regression model is trained to predict the performance of unknown configurations. Subsequently, a well performing configuration is sampled from the ensemble and used to create a new configuration from it. If its predicted performance is good enough, it replaces one of the configurations in the ensemble.

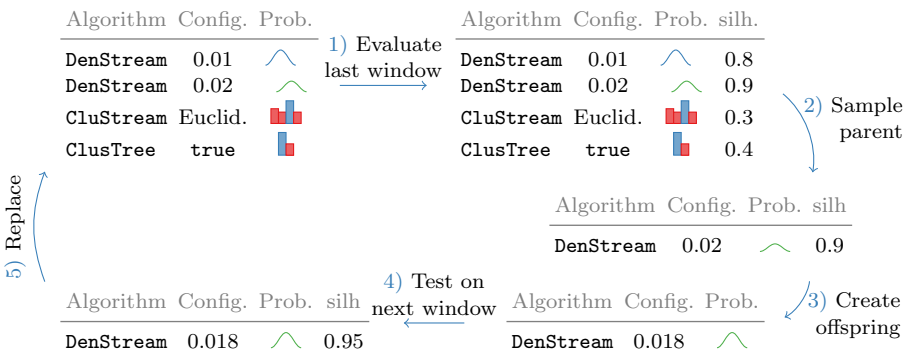


Fig. 1. First, the performance of all algorithms and configurations in the ensemble is evaluated. Afterwards, one algorithm is sampled to create an offspring and tested on the next window. If its performance is high enough, it is used to replace one of the algorithms in the ensemble. For brevity, only one hyperparameter per algorithm is shown.

3 Automated Algorithm Selection and Configuration for Stream Clustering

In this section, we extend our ensemble idea for the `confStream` algorithm. In our initial proposal, we only optimised one numeric parameter for a given algorithm. Here, we extend upon this and also include the algorithm *selection* problem. In particular, we treat the algorithm selection and algorithm configuration problem as one large optimisation task. In addition, we show how to optimise multiple parameters per algorithm which can be of different types such as numerical, categorical, integer, binary or ordinal. Finally, we also improve the selection process of new configurations and extensively test and compare the algorithm.

Our main idea is summarised in Fig. 1. Our algorithm uses a given starting configuration, i.e. a list of algorithms, their initial configurations and the corresponding parameter ranges. For example, this can be the default configuration of all available algorithms.

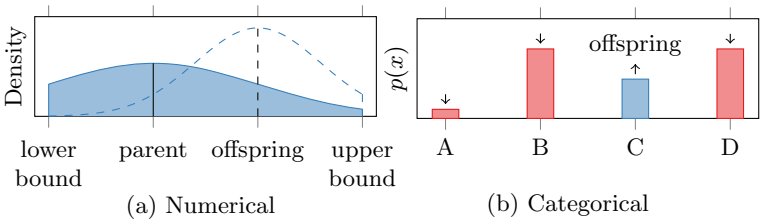
To apply our ensemble strategy, we process the stream in windows of size h. We use the observations in a window in order to train the algorithms in the current ensemble. After every window, we evaluate the clustering quality for every configuration (Step 1). In our experiments, we used the Silhouette Width as a measure of cluster quality due to its popularity but other quality metrics are equally applicable. For every observation i in the window, the Silhouette Width uses the average similarity to its own cluster $a(i)$ and compares it to the average similarity to its closest cluster $b(i)$:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}. \tag{1}$$

The Silhouette Width is usually averaged over all observations to obtain a single index. The algorithm with the highest cluster quality becomes the active clusterer or *incumbent* and provides the current solution of `confStream`.

To obtain new configurations, a configuration is sampled from the ensemble and serves as a parent (Step 2). The sampling is performed proportionally to the performance of the algorithms such that better performing configurations have a higher probability to be selected. In general, this can be implemented using a simple Roulette Wheel Selection. Note, however, that the Silhouette Width has a range of $[-1, 1]$. Since negative values cannot be used in Roulette Wheel Selection, we shift the values by $|\min \boldsymbol{x}| - \min \boldsymbol{x}$, where $\boldsymbol{x}$ are the Silhouette Widths of all configurations. This would turn a sequence $\{-0.5, 0.8, 1\}$ into $\{0.5, 1.8, 2\}$. Also note that we do not necessarily choose the best performing algorithm in order to increase the diversity in the ensemble.

The selected algorithm and configuration is then used as a parent in order to derive a new configuration from it (Step 3). For this, we use some of the ideas of `irace` [20]. Specifically, every parameter of every configuration has an associated probability distribution. For numerical parameters, every parameter maintains a truncated normal distribution $\mathcal{N}(\mu, \sigma)$ with expectation μ and standard deviation σ. The expectation of the truncated normal distribution is placed



(a) Numerical (b) Categorical

Fig. 2. For numerical parameters, a truncated normal distribution is placed at the parent to sample a new configuration. For categorical parameters, a new configuration is drawn according to a list of probabilities. Both approaches increase the probability for the offspring to favour promising solutions over time.

at the position of the parent to sample a new configuration. Subsequently, the standard deviation of the child is reduced to increase exploitation of this area (Fig. 2a). In our case, we use an exponential decay of the standard deviation according to a fading factor λ:

$$\sigma_{t+1} = \sigma_t \cdot 2^{-\lambda} \tag{2}$$

The idea is to reduce the standard deviation in exponentially smaller steps in order to narrow down a promising parameter region. Initially, the standard deviation is set to half the parameter range. For integer parameters, the same sampling strategy can be used, where the result is rounded to the nearest integer. Similarly, for ordinal parameters (e.g. strong, medium, weak) we can use the integer sampling strategy, where the resulting integer is used as the index in the list of possible outcomes.

For categorical parameters, a list of probabilities for every outcome is maintained. Starting with equal probabilities, the new configuration is sampled according to the list. Subsequently, the probability of the winning category is increased to facilitate exploitation (Fig. 2b). In our case, we increase the probability by the same amount that we reduce the standard deviation in the numerical case:

$$p_{t+1} = p_t \cdot \left(2 - 2^{-\lambda}\right). \tag{3}$$

Note that in both cases we use a factor of one as the baseline and either increase or decrease the factor by $1 - 2^{-\lambda}$ to decrease the standard deviation or increase the probability. To obtain probabilities, all values are then scaled to sum to one again.

Since the streams' distribution can change over time, it is not necessarily beneficial to exploit promising regions further. Instead, it is also necessary to explore new regions of the search space. For this, for a fraction of the ensemble, we reset the standard deviation or the probability vector to its initial value with probability p to explore new regions.

Note that adapting parameter values to create the child is not always easy. While some parameters, such as distance thresholds, can be easily changed "on-

the-fly", i.e. while the algorithm is running, some parameters are much harder to change. For example, if the parameter influences a tree-height as in `ClusTree` [19], changing the parameter is often not possible due to the implications for the underlying data structure. For all cases where we cannot change the parameter, we instead initialise a new algorithm instance based on the new configuration. However, to keep as much information as possible, we then train the new algorithm with the micro-clusters of the parent configuration. For this, the centres of the micro-clusters are used as virtual points to train the child algorithm. While this will not reproduce the exact same micro-clusters, it passes on some information about the current clusters.

In our initial proposal [11], we trained a regression model [15] to predict the performance of the new configurations. However, we noticed that the regression model often favoured algorithms with many valid configurations (such as `BICO` [12]) whereas it disfavoured algorithms where some configurations perform exceptionally well but many fail. To prevent this, we eliminated the predictor and introduce a "test ensemble" instead where new configurations are evaluated on the actual stream before deciding whether to incorporate them into the ensemble (Step 4). After every window, we fill the test ensemble with a number of new configurations. The algorithms of the test ensemble are compared to algorithms in the active ensemble when clustering the next window. If a configuration in the test-ensemble outperforms a configuration in the active ensemble (on data both have never seen before), it is considered promising and replaces a configuration in the active ensemble. The decision to use or discard a new configuration is therefore lagged by one window.

Again, we sample the configuration that is replaced proportionally to its fitness, removing less fit solutions with higher probability. However, we never remove the best configuration of an algorithm and always keep at least one configuration per algorithm. This prevents that an algorithm is removed entirely and cannot be used for the generation of new configurations. For larger ensembles, it can also be beneficial to keep the default configurations in the ensemble as a fall-back. Note that we use an ensemble of fixed size here and initially fill the ensemble with new configurations to its maximum capacity.

4 Experiments

4.1 Experimental Setup

To evaluate our algorithm, we implemented it in Java[1] as a clustering algorithm for the `MOA` framework [4,5]. For our experiments, we aim to select the optimal algorithm and its configuration among the available stream clustering algorithms in `MOA` for a specific stream. For a fair comparison, we restrict ourselves to all clustering algorithms which expose the micro-clusters. Specifically, we optimise `DenStream` [6], `CluStree` [19], `CluStream` [2] and `BICO` [12] which are all state-of-the-art stream clustering algorithms [10]. For every algorithm,

[1] Implementation available at: https://www.matthias-carnein.de/confStream.

we optimise all parameters that influence the clustering result in their full value range (Table 1). Unbounded value ranges are artificially capped using appropriate maximum values.

Table 1. Overview of optimised algorithms and parameters. Parameter names as used in `MOA` [5].

Algorithm	Configuration	Type	Range	default
`DenStream`	e	Numeric	$[0, 1]$	0.02
	b	Numeric	$[0, 1]$	0.2
	m	Integer	$\{0 \dots 10000\}$	1
	o	Integer	$\{2 \dots 20\}$	2
	l	Numeric	$[0, 1]$	0.25
`ClusTree`	H	Integer	$\{1 \dots 20\}$	8
	B	Boolean	$\{1, 0\}$	0
`CluStream`	k	Integer	$\{2 \dots 20\}$	5
	m	Integer	$\{1 \dots 10000\}$	100
	t	Integer	$\{1 \dots 10\}$	2
`BICO`	k	Integer	$\{2 \dots 20\}$	5
	n	Integer	$\{1 \dots 2000\}$	1000
	p	Integer	$\{1 \dots 20\}$	10

To determine whether `confStream` is able to improve configurations over time, we first compare its performance to `MOA`'s [4,5] default configurations. For this, we initialise our ensemble with the same default configurations and compare the results. Afterwards, we use `irace` [20] and optimise the parameter's a-posteriori to find the best overall configuration. We then compare whether `confStream`'s adaptive approach is able to compete with the optimal result. For `confStream`, we set the ensemble size $e_{\text{size}} = 20$, fading factor $\lambda = 0.05$, reset probability $p = 0.01$ and evaluate the solutions every $h = 1000$ observations using the Silhouette Width. After each window, we create $e_{\text{test}} = 10$ new configurations.

We evaluate our algorithm on four data sets. Specifically, we use a `Random Radial Basis Function (RBF)` stream [4], the `sensor` stream[2], the `power-supply` stream[3] and the `covertype` data set[4]. All data sets are popular choices within the stream clustering and classification literature and for our analysis we use all numeric parameters of the streams. Note that the `covertype` data set is a static data set, which we turn into a data stream by processing observations one by one. This is a common strategy in stream clustering due to the limited

[2] http://db.csail.mit.edu/labdata/labdata.html.
[3] http://www.cse.fau.edu/~xqzhu/stream.html.
[4] http://archive.ics.uci.edu/ml/datasets/Covertype.

number of openly available data streams [1,7,8,12]. An overview of all data sets is given in Table 2. To avoid differences in scale, we standardise the data sets by subtracting the mean and dividing by the standard deviation per feature. In real world scenarios, the values for normalisation can often be updated incrementally [3,9]. Our goal was to include data streams with diverse characteristics. For this, we included both real and artificial data streams which all include different forms of concept drift, i.e. a shift of the underlying distribution. We also included a static data set which does not include any temporal changes. In addition, our data streams have between 2 and 10 dimensions and some have more than 2 million observations.

Table 2. Overview of the four data streams used in our experiments. All data streams are popular in the stream clustering literature [7].

Data set	n	d	Type	Drift
`Random RBF`	2,000,000	2	Artificial	✓
`sensor`	2,219,803	4	Real	✓
`powersupply`	29,928	2	Real	✓
`covertype`	581,012	10	Real	–

4.2 Results

Comparison to Default Configuration. For a start, we compare our `confStream` algorithm with `MOA`'s default algorithm configurations. For this, `confStream` is initialised with the same default configurations (Table 1) but optimises the parameters. Figure 3 shows the Silhouette Width for every window of our test data streams. The boxplots on the right summarise the distribution of the Silhouette Width values along the stream. It is obvious that our ensemble approach produces a considerably better result throughout the entirety of the stream. For example, for the `Random RBF` stream, `confStream` yields a median Silhouette Width of 0.86 while the other algorithms perform much worse with median Silhouette Widths between 0.54 and 0.65. Similar results can also be observed for the remaining data streams. This is particularly visible for the `covertype` and `powersupply` data streams where most algorithms produce much worse results by default.

To evaluate how `confStream` optimises the algorithms in the ensemble over time, we analyse the best configuration for every clustering algorithm within the ensemble (Fig. 4a). We can see that throughout most segments of the stream, a configuration of `ClusTree` is the incumbent, i.e. the best configuration. `BICO` on the other hand tends to have the worst performance within the ensemble. This also shows in the ensemble composition (Fig. 4b). The ensemble is quickly filled with more configurations of `ClusTree` as it shows the best performance. Even

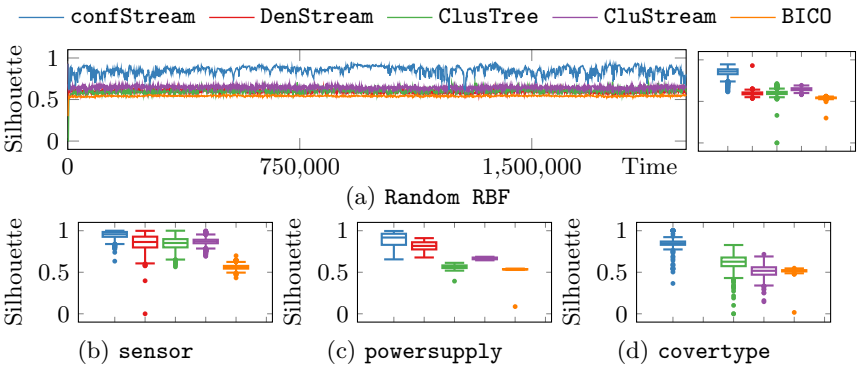


Fig. 3. Development of Silhouette Width for all data streams.

though the other algorithms have less relevance, we can see that their share in the ensemble increases whenever their performance improves. This is particularly visible for the periodic peaks of `DenStream` and also shows for `CluStream` as it improves after around 1.5 million observations.

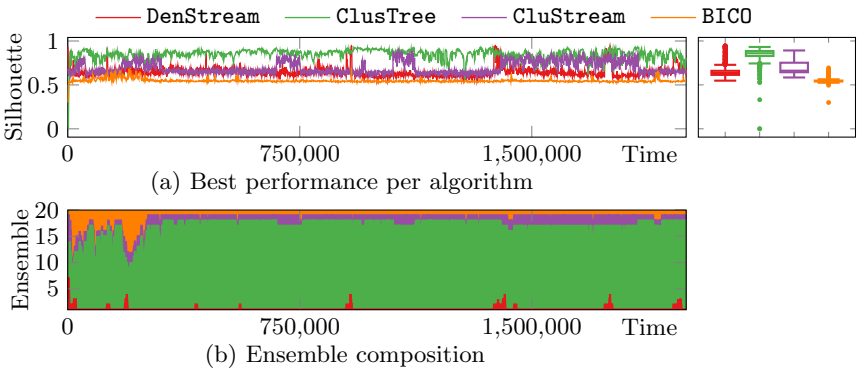


Fig. 4. Ensemble performance and composition for the `Random RBF` stream.

We can also see how `confStream` quickly adapts the parameter values depending on their performance. Figure 5 shows the current best algorithm and its parameter values for the first 100,000 observations of the `Random RBF` stream on a logarithmic scale. We can see that the algorithm initially switches between configurations of `DenStream` and `CluStream`, before eventually settling on `ClusTree` as the incumbent. After about 30.000 observations `CluStream` briefly becomes the incumbent and its parameters are improved before the algorithm goes back to `ClusTree`.

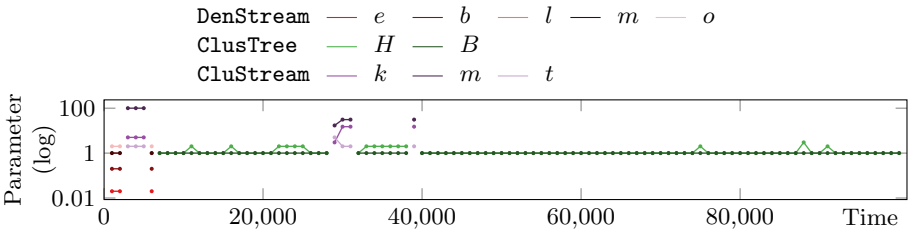


Fig. 5. Best algorithm and configuration over time for the first 100,000 observations of the `Random RBF` stream. The currently best algorithm with its best configuration is shown on a logarithmic scale. The parameters are color coded based on the algorithm and the parameter names of `MOA` [4,5] are used.

Evaluation of Combined Algorithm Selection and Hyperparameter Configuration. In our experiments, we treated both algorithm selection and configuration as one large optimisation problem. This is also known as the Combined Algorithm Selection and Hyperparameter Configuration (CASH) [18] problem. In order to evaluate how this affects `confStream`'s performance, we also compare it to individually configured stream clustering algorithms. For this, we use the default configuration per algorithm as separate starting configurations for `confStream` and optimise the algorithms separately using the same ensemble size and settings. Figure 6 compares the combined optimisation and individual optimisation. The results show that `confStream`'s solution to the CASH problem is similar to the individual optimisation. This is a surprising result, given the tremendously increased search space of the combined optimisation problem. Generally, we would expect the individual optimisation to perform better. The fact that the CASH problem yields similar results shows that the algorithm can handle the increased search space. This shows that it is possible to perform algorithm selection and configuration simultaneously without sacrificing the clustering quality.

It is also interesting to compare the individual optimisation (Fig. 6) to the algorithms' performance of the combined optimisation problem (Fig. 4a). We can see, that `ClusTree` and `BICO` are almost as well configured in the combined optimisation as in the individual optimisation. `DenStream` on the other hand yields better results when optimised individually. This is likely due to the fact that the algorithm has many parameters and large parameter ranges which causes a large search space. Its weak performance, however, leads the ensemble to remove most instances of `DenStream` which yields too few instances to explore the large search space.

Comparison to Offline Optimisation. The above results show that `confStream` can considerably improve upon the default configurations with little additional knowledge. However, we should also compare our algorithm to optimised configurations using an a-posteriori scenario, i.e. given the knowledge of

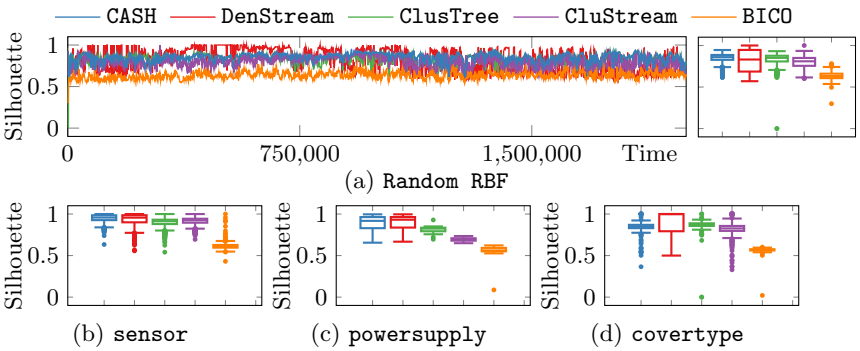


Fig. 6. Combined algorithm selection and hyperparameter optimisation (CASH) compared to only hyperparameter optimisation for all data streams.

Table 3. Overview of optimal configurations as identified by `irace` (rounded).

Algorithm	Configuration	RBF	sensor	power	cover
`DenStream`	e	0.08	0.80	0.35	0.55
	b	0.32	0.26	0.02	0.61
	m	2913.1	9085.1	4027.1	282.4
	o	16.49	7.08	7.54	3.37
	l	0.10	0.07	0.88	0.11
`ClusTree`	H	8	3	1	1
	B	`false`	`true`	`true`	`false`
`CluStream`	k	5	8	5	3
	m	100	98	200	4
	t	2	2	2	2
`BICO`	k	2	6	14	16
	n	36	1880	53	637
	p	7	9	3	2

the total stream. In [7], we already used `irace` [20] to find configurations for some of the algorithms and data streams by optimising for the optimal adjusted Rand index and in [8] when optimising for the Sum of Squares (SSQ). In addition, we now also use `irace` and optimise for the Silhouette Width directly. This gives us three configurations for most algorithms, which have been optimised a-posteriori. We initialise `irace` with the default algorithm configuration and allow up to 150 evaluations for every data stream. Note that this would be equivalent to running `confStream` with an ensemble size of 150. In contrast, we only use 20 in our experiments, which gives `irace` a clear advantage. The configurations which produced the highest median Silhouette Width across the entire stream are listed in Table 3.

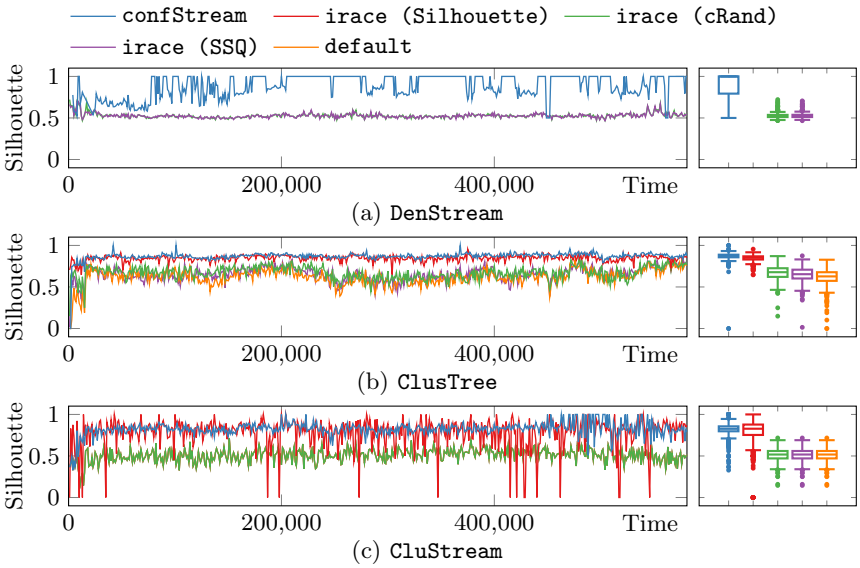


Fig. 7. Comparing performance of `confStream`, the optimal configuration found by `irace` optimised for the adjusted Rand Index (cRand) [7], the `SSQ` [8] and Silhouette Width as well as the default configuration for the `covertype` data stream.

Figure 7 compares the performance of `confStream` (without algorithm selection) with its default and the three optimised configurations for the `covertype` data set. For example, for the `ClusTree` algorithm (Fig. 7b), the default configuration yields the worst quality with a median Silhouette Widh of 0.63. As expected, all `irace` configurations perform better and the one optimised for the Silhouette Width also yields a higher Silhouette of 0.85. However, throughout the vast majority of the stream, `confStream` yields even higher quality than the a-posteriori solutions with a median Silhouette Width of 0.87. This is because `confStream` can adapt the configuration over time which allows it to adapt to changes in the stream. The results for `CluStream` are overall similar (Fig. 7c) and for `DenStream`, the default configuration and one of the optimised configurations did not produce a valid solution at all (Fig. 7a). `confstream` on the other hand produced good solutions throughout most of the stream.

Note that we highlight the results for the `covertype` data set here because we have the most complete set of configurations for this scenario. The same analysis for the other data streams is summarised in Fig. 8. For brevity, we only report the boxplots of the performance. Note that configurations of `irace` optimised for the adjusted Rand Index and SSQ are only available for some combinations. For all cases where they are available, `confStream` yields considerably better Silhouette Width. In comparison to `irace` optimised for the Silhouette Width, the quality of `confStream` is often similar. For example, for the `powersupply` data stream, the results of `confStream` is very slightly better than the a-posteriori

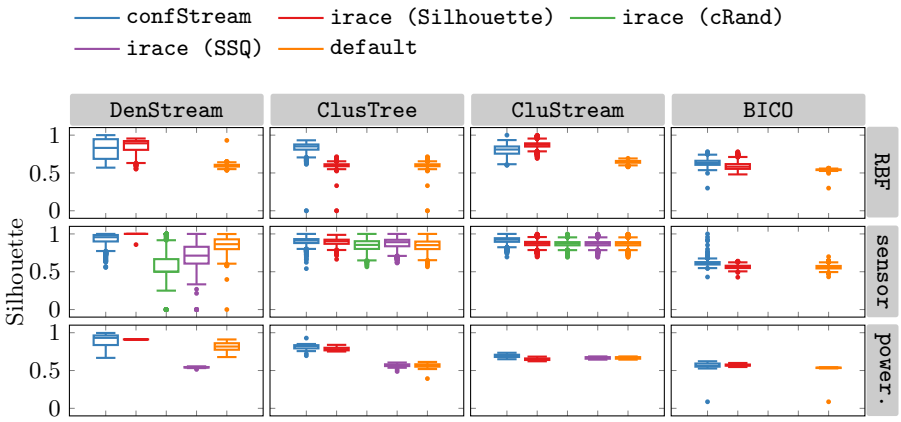


Fig. 8. Comparing performance of `confStream`, the optimal configuration found by `irace` optimised for the adjusted Rand Index (cRand) [7], the `SSQ` [8] and the Silhouette Width as well as the default configuration for the `Random RBF`, `sensor` and `powersupply` data streams.

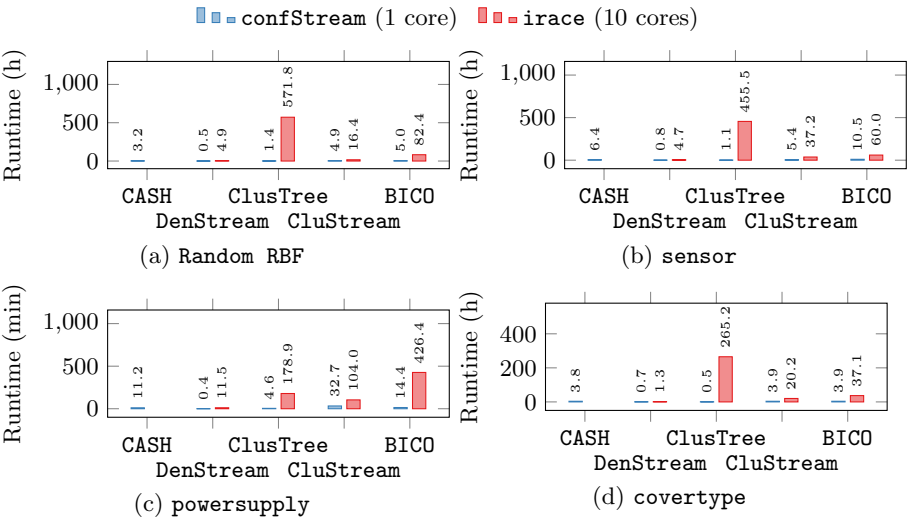


Fig. 9. Runtime of `confStream` and `irace` in hours or minutes. `irace` was paralellised with ten cores while `confStream` only used a single core. Experiments performed on an Intel Xeon CPU E5-2630 v4 with 2.20 GHz.

approach. Overall this shows, that `confStream` produces similar results in quality than a-posteriori optimisation with vastly less computational resources.

Throughout our experiments, we observed that the (online) `confStream` algorithm only requires a fraction of the number of evaluations compared to the (offline) `irace` approach. This also shows in the runtime of the algorithms as

highlighted in Fig. 9. In our experiments, we parallelised the racing procedure of `irace` on ten cores. `confStream`, on the other hand, was run on a single core. Despite this considerable disadvantage, `confStream` is much faster for every single algorithm and data stream. This particularly shows for the `ClusTree` algorithm. For example, on the `Random RBF` data stream, `irace` required more than 23 days to optimise the parameters while `confStream` finished within less than one and a half hours. These results would further improve when parallelising the training of the ensemble for `confStream`. In addition, `confStream` is also able to solve the CASH problem. The runtime of the CASH problem mostly depends on the ensemble composition. As such, the runtime usually lies between the runtime of the fastest and slowest approach. In our experiments, the runtime of the CASH problem was often similar to the fastest algorithm in `irace`.

5 Conclusion

In this paper we proposed the first approach for automated algorithm selection and hyperparameter configuration of stream clustering algorithms. Our approach allows to apply stream clustering without expert knowledge and significantly facilitates the application of stream clustering in practice. In our approach, we train an ensemble of different algorithms and configurations in parallel to identify superior solutions. By drawing new configurations from the ensemble, we are able to improve solutions along the stream. Over time, sampling is biased towards more promising solutions. Our experiments on multiple state-of-the art algorithms, their hyperparameters as well as popular and diverse data streams have shown consistently good performances. The algorithm was able to quickly improve upon its initial configuration and to find valid configurations where the default configurations failed. We also compared the performance to `irace`, where we optimised the configurations a-posteriori. Even in this comparison, `confStream` found competitive solutions, despite working online and with far fewer iterations. While training multiple algorithms in parallel is slower, `confStream` was fast enough to work in real-time since the algorithms can be trained in parallel. In particular, it is much faster than a-posteriori approaches which are usually infeasible for data streams.

In future work we will evaluate our approach on a larger number of data streams and algorithms. In addition, we plan to apply our approach to other streaming applications such as stream classification or stream regression. Furthermore, we would like to revisit the idea of predicting the performance of new configurations. In a-posteriori approaches this has shown to produce good results [16] and we believe that this is also possible in the streaming scenario.

References

1. Ackermann, M.R., Märtens, M., Raupach, C., Swierkot, K., Lammersen, C., Sohler, C.: StreamKM++: a clustering algorithm for data streams. J. Exp. Algorithmics **17**, 2.4:2.1–2.4:2.30 (2012)

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 (2003)
3. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for projected clustering of high dimensional data streams. In: Proceedings of the 30th International Conference on Very Large Data Bases (VLDB 2004), vol. 30, pp. 852–863 (2004)
4. Bifet, A., Gavalda, R., Holmes, G., Pfahringer, B.: Machine Learning for Data Streams with Practical Examples in MOA. MIT Press, Cambridge (2018)
5. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: massive online analysis. J. Mach. Learn. Res. **11**, 1601–1604 (2010)
6. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: Proceedings of the Conference on Data Mining (SIAM 2006), pp. 328–339 (2006)
7. Carnein, M., Assenmacher, D., Trautmann, H.: An empirical comparison of stream clustering algorithms. In: Proceedings of the ACM International Conference on Computing Frontiers (CF 2017), pp. 361–365. ACM (2017)
8. Carnein, M., Trautmann, H.: Evostream – evolutionary stream clustering utilizing idle times. Big Data Res. **14**, 101–111 (2018)
9. Carnein, M., Trautmann, H.: Customer segmentation based on transactional data using stream clustering. In: Yang, Q., Zhou, Z.-H., Gong, Z., Zhang, M.-L., Huang, S.-J. (eds.) PAKDD 2019. LNCS (LNAI), vol. 11439, pp. 280–292. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-16148-4_22
10. Carnein, M., Trautmann, H.: Optimizing data stream representation: an extensive survey on stream clustering algorithms. Bus. Inf. Syst. Eng. (BISE) **61**, 277–297 (2019). https://doi.org/10.1007/s12599-019-00576-5
11. Carnein, M., Trautmann, H., Bifet, A., Pfahringer, B.: Towards automated configuration of stream clustering algorithms. In: Cellier, P., Driessens, K. (eds.) ECML PKDD 2019. CCIS, vol. 1167, pp. 137–143. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-43823-4_12
12. Fichtenberger, H., Gillé, M., Schmidt, M., Schwiegelshohn, C., Sohler, C.: BICO: BIRCH meets coresets for k-means clustering. In: Bodlaender, H.L., Italiano, G.F. (eds.) ESA 2013. LNCS, vol. 8125, pp. 481–492. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40450-4_41
13. Fitzgerald, T., Malitsky, Y., O'Sullivan, B., Tierney, K.: ReACT: real-time algorithm configuration through tournaments. In: Edelkamp, S., Barták, R. (eds.) Proceedings of the Seventh Annual Symposium on Combinatorial Search (SOCS 2014) (2014)
14. Fitzgerald, T., O'Sullivan, B., Malitsky, Y., Tierney, K.: Online search algorithm configuration. In: Brodley, C.E., Stone, P. (eds.) Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, pp. 3104–3105. AAAI Press (2014)
15. Gomes, H.M., Barddal, J.P., Ferreira, L.E.B., Bifet, A.: Adaptive random forests for data stream regression. In: Proceedings of the 26th European Symposium on Artificial Neural Networks (ESANN 2018) (2018)
16. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 507–523. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25566-3_40
17. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. J. Artif. Intell. Res. **36**, 267–306 (2009)

18. Hutter, F., Kotthoff, L., Vanschoren, J. (eds.): Automated Machine Learning: Methods, Systems, Challenges. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05318-5
19. Kranen, P., Assent, I., Baldauf, C., Seidl, T.: Self-adaptive anytime stream clustering. In: Proceedings of the 9th IEEE International Conference on Data Mining (ICDM 2009), pp. 249–258, December 2009
20. 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)
21. Minku, L.L.: A novel online supervised hyperparameter tuning procedure applied to cross-company software effort estimation. Empir. Softw. Eng. **24**(5), 3153–3204 (2019). https://doi.org/10.1007/s10664-019-09686-w
22. van Rijn, J.N., Holmes, G., Pfahringer, B., Vanschoren, J.: Having a blast: meta-learning and heterogeneous ensembles for data streams. In: Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM 2015), pp. 1003–1008, November 2015
23. van Rijn, J.N., Holmes, G., Pfahringer, B., Vanschoren, J.: Algorithm selection on data streams. In: Džeroski, S., Panov, P., Kocev, D., Todorovski, L. (eds.) DS 2014. LNCS (LNAI), vol. 8777, pp. 325–336. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11812-3_28
24. van Rijn, J.N., Holmes, G., Pfahringer, B., Vanschoren, J.: The online performance estimation framework: heterogeneous ensemble learning for data streams. Mach. Learn. **107**(1), 149–176 (2018). https://doi.org/10.1007/s10994-017-5686-9
25. Veloso, B., Gama, J., Malheiro, B.: Self hyper-parameter tuning for data streams. In: Soldatova, L., Vanschoren, J., Papadopoulos, G., Ceci, M. (eds.) DS 2018. LNCS (LNAI), vol. 11198, pp. 241–255. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01771-2_16