AutoClass: AutoML for Data Stream Classification

Maroua Bahri, Nikolaos Georgantas

AutoClass: AutoML for Data Stream Classification

Maroua Bahri
Inria Paris
Paris, France
maroua.bahri@inria.fr

Nikolaos Georgantas
Inria Paris
Paris, France
nikolaos.georgantas@inria.fr

***Abstract*—Automated Machine Learning (autoML) is a novel topic that aims to tackle the parameter configuration issue using automatic monitoring models and comprises different machine learning tasks, such as feature selection, model selection, and hyper-parameter tuning. It makes easier use of algorithms for non-ML experts as well as ML experts by automating tasks that rely on expert domain knowledge. Nevertheless, autoML is in its infancy stage and not well explored yet in the offline and stream settings. In this paper, we propose automated Classification (*autoClass*) method for automated algorithm selection and configuration for data stream classification. AutoClass consists of training an ensemble of different tuned configurations and selecting the best-performing configuration to do the prediction. We present experiments performed on a diverse set of real and artificial datasets and show how our proposed approach can outperform the performance of competitive state-of-the-art ensemble and single-based methods.**

***Index Terms*—AutoML, stream classification, data stream, algorithm selection, hyper-parameter tuning.**

I. Introduction

In recent years, we have witnessed a digital revolution that has dramatically changed the way in which we generate and consume data. This new paradigm of ubiquitous data led to several emerging applications, sensors, and Internet-of-Things (IoT) devices that generate massive amounts of data continuously and online as big data streams. In this context, data stream mining tasks have become indispensable in multiple real-world applications that demand real-time – or near real-time – analysis, such as traffic management, smart cities, and predictive maintenance for transport and logistics.

Machine Learning (ML) benefited from tremendous research progress recently in many application areas, particularly for the stream setting. Often, to unravel the full potential of ML, some human expertise and domain knowledge are required in order to select the most effective algorithm, tune its hyper-parameters, as well as to perform pre-processing including, but not limited to, data cleaning, filling missing data, feature selection, feature extraction, etc [1]. This is often a tedious and non-trivial task for researchers with all levels of expertise. To alleviate such a burden for researchers, several techniques have been proposed under the emerging automated Machine Learning (autoML) topic that is still in its infancy stage and not well explored yet in both offline and stream settings.

AutoML tackles mainly the problem of finding ML models with high predictive performance. In the traditional offline setting, existing approaches assume that all data are available at the beginning of the training process. They configure and optimize a pipeline of preprocessing, feature engineering, and model selection by choosing suitable hyper-parameters. With the creation of a vast amount of data from all kinds of sources every day, our capability to process and understand these datasets in a single batch is no longer viable. In fact, the unbounded nature of evolving data streams raises some technical and practical limitations that make traditional stream algorithms fail because of the high use of resources, such as time and memory, to process dynamic data distributions. By training ML models incrementally (i.e. online learning[1]), the flood of data can be processed sequentially within data streams. However, if one assumes an online learning scenario, where an autoML instance executes on evolving data streams, the question of the best model and its optimal configuration remains open.

Numerous frameworks for data stream processing have emerged that focus on the application of ML algorithms, which mainly pursue online ML requirements. Common frameworks such as MOA [2], and river [3], aim to foster research for algorithms on data streams. Nevertheless, on the conceptual level for the processing and orchestration of data streams, further frameworks and functionalities are needed to perform autoML. In this paper, we focus on these issues in the classification of data streams using an ensemble of different algorithms and configurations and a hyper-parameters optimization technique. The main focus of our work attempts to propose an autoML solution for classification to deal with massive data by tuning the parameters incrementally and selecting the best-performing classifier at evaluation time. Our main contributions can be summarized as follows:

- *autoClass*. We propose an automated classification approach for data streams that naturally adapts the best algorithm and its hyper-parameters over time.
- *Empirical results*. We present an experimental study that shows the abilities of our proposal to obtain good results against popular baselines.

[1]In compliance with literature terminology, we use the terms online and stream learning interchangeably.

The remainder of this work is organized as follows. In Section II, we detail related work where we present the state-of-the-art in autoML. Section III presents the description of our proposed approach for automated stream classification. In Section IV, we show and discuss the experimental results using a diverse set of datasets. We finally end this paper by drawing conclusions in Section V.

II. Related work

AutoML is a novel topic that has attracted several researchers in recent years due to the importance of its application. It supports researchers and practitioners with the tedious work of manually designing ML pipelines and can also be viewed as the process that makes ML easier by avoiding manual hyper-parameters tuning for both ML experts and non-experts. Automated approaches often model the autoML task as the Combined Algorithm Selection and Hyper-parameter optimization problem, aka CASH [4].

Most of the autoML solutions have been initially proposed for the batch setting and are discussed in recent surveys on autoML and its open challenges [5], [6]. Examples of established autoML frameworks are Auto-WEKA [7] and Auto-Sklearn [8] which are two automated systems that have been implemented on top of the well-known ML software scikit-learn and WEKA, respectively, GAMA [9], TPOT [10], and H_2O [11]. These solutions mainly differ in their search space and Hyper-Parameter Optimization (HPO) technique. For instance, Auto-WEKA uses a random forest algorithm as HPO, Auto-Sklearn exploits a Bayesian optimization technique [12], H_2O, GAMA, and TPOT employs a grid search approach. However, these tools and techniques exclusively work in the batch setting and cannot work with evolving data streams because they allow, among others, several access to data instances and therefore break the requirements of the streaming framework, notably the one-pass constraint.

The first attempts to improve the performance of ML methods, and make them automated, have been proposed using ensemble and meta-learning approaches. The first attempts for automated solutions in the classification task consist of using an ensemble and selecting the best-performing classifier on the last window to be used for the next window of instances as the BLAST [13]. However, this method operates as a heterogeneous ensemble and does not tune the parameters. The few existing automated solutions in classification, such as EvoAutoML [14], use an ensemble and precise a diverse set of algorithms with fixed hyper-parameter values from which a solution is randomly selected whenever the current models in the ensemble are evaluated and the worst one has to be changed. Nevertheless, in such solutions, there is no automated tuning because they operate as heterogeneous ensembles where the parameters' values are already fixed.

In the clustering task, a first solution, called ConfStream [15], for automated configuration for clustering evolving data streams has been proposed. It uses an ensemble of a diverse set of stream clustering algorithms with different configurations and the best-performing one will be used to create a new one from it that will replace one of the configurations in the ensemble (if it is good enough). The silhouette width measure has been used to evaluate the clustering quality of every configuration in ConfStream. Therefore, the configuration with the best cluster quality is selected for the next window of the stream. However, ConfStream uses periodically a small part of the stream (window) to select the model and the configuration that obtain the best performance and use them for the next window from the stream, and so on. Albeit it adopts a batch-incremental strategy, ConfStream is a very promising method for unsupervised autoML for data streams that could be extended to serve the supervised setting.

III. Automated classification for data streams

Classification algorithms require usually hyper-parameters to be set *a priori*, such as the k-nearest neighbors algorithm which has one parameter k, and the stream version of the decision tree, Hoeffding tree [16], which requires to set the grace period (the number of instances a lead should observe between split attempts) and the split confidence. In practice, it is difficult to find appropriate values for these hyper-parameters that potentially need to be changed for each dataset and whenever the data distribution changes. Whence the need for a method that naturally adapts its configurations within a small ensemble to enable the adaptation of algorithm hyper-parameter search spaces to data streams.

The question of changing and adapting the configuration of an algorithm as well as the orchestration of models while considering the requirements of the stream setting remains open. One viable solution is to propose a supervised version of ConfStream [15] where the dynamic adaptation of the hyper-parameters is mandatory to have a fully automated method and explore the HPO that could be used with classification to improve the accuracy.

As previously mentioned, an autoML method, based on algorithm configuration and selection, comprises (i) a configuration space that consists of a set of algorithms that are going to be selected and the corresponding ranges for their hyper-parameters, and (ii) an experimental setup to evaluate the candidate configurations composed of {algorithms, hyper-parameters} based on a predefined quality measure that enables the autoML method to select the best performing configuration. In this work, we propose an ensemble approach for automated data stream classification, called *autoClass*, that considers the configuration space and setup and aims at predicting class labels for potentially infinite data streams using the best-performing model. Our main idea is depicted in Fig. 1 and explained in the following sections.

A. Configuration space

The configuration space models a standard ML pipeline that comprises main high-level components which are the learning and prediction. Other components such as *preprocessing* including feature selection and extraction could be freely added to the pipeline.

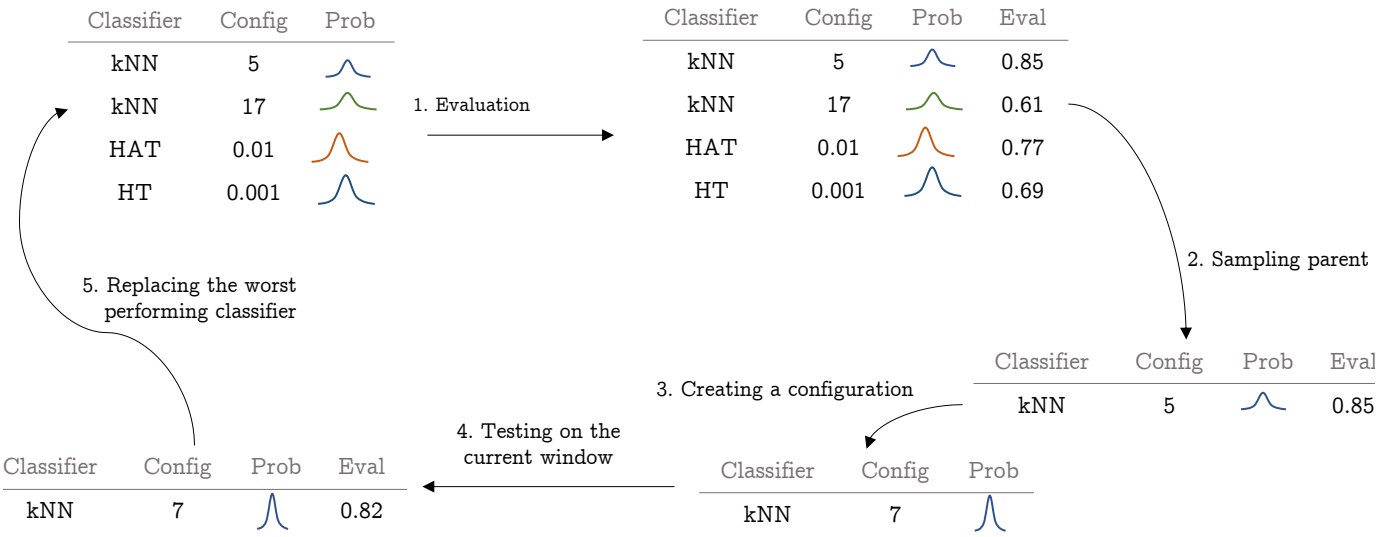


Fig. 1: AutoClass.

Definition 1: Data stream CASH, adapted from [5]. Let $\mathcal{A} = \{A^{(1)}, \ldots, A^{(n)}\}$ be a set of n different stream classifiers, and let the hyper-parameters of each algorithm $A^{(j)}$ have a domain $\Lambda^{(j)}$. Let S be a potentially infinite stream of instances $\{x_i, y_i\}$, and $S^t = \{x_0, y_0\}, \ldots, \{x_t, y_t\}$ be an ordered sequence of past examples, where x_i is a tuple of attributes and y_i is the true class label that we aim to predict. Let $\mathcal{L}(\mathcal{M}_{\overrightarrow{A},\overrightarrow{\lambda}}(S^T), S^V)$ denote the loss that algorithm combination $M^{(j)}$ achieves on a set of validation examples $S^V \subset S^t$ when trained on $S^T \subset S^t$ with hyper-parameters $\overrightarrow{\lambda}$. Denote that the algorithm is used for testing using S^V before being used for training using S^T using the Test-Then-Train validation.

Then the stream CASH problem is to find the joint algorithm combination and hyper-parameter setting that minimize the loss:

$$\overrightarrow{A}^*, \overrightarrow{\lambda}^* \in \underset{M^{(j)} \in \mathcal{M}, \lambda \in \Lambda^{(j)}, A \in \mathcal{A}}{\arg\min} \mathcal{L}(\mathcal{M}_{\overrightarrow{A},\overrightarrow{\lambda}}(S^T), S^V) \quad (1)$$

Existing ensemble-based methods for data streams [17]–[19] do not fully cover the stream CASH problem since their hyper-parameters are usually set at the start of the stream processing and are not changed as the stream evolves. Moreover, they are mostly composed of homogeneous algorithms while our proposed automated solution addresses the online CASH and considers the configuration space for each algorithm.

In Algorithm 1, we present the training phase of the autoClass algorithm that adapts the algorithm configuration in an online manner. The first step of our approach consists of having a set of classifiers generated from $\mathcal{A}$ and their hyper-parameter ranges that constitute the main core of our automated solution. The learning phase of this component consists of learning on incoming data from the stream and is performed on an ensemble $\mathcal{M}$ of size s of candidate methods from the configuration space that are composed of different algorithms and different hyper-parameter values. Each instance received from the stream will be used to train all learners in $\mathcal{M}$. Besides, given an unlabeled instance from the stream, the prediction is made using the best-performing algorithm inside the ensemble. In this work, since data are evolving and are not all available in the memory, we use the *Test-Then-Train* evaluation method where each instance from the stream is firstly used for prediction by testing the best-performing classifier, then using it to train all the classifiers in $\mathcal{M}$.

Algorithm 1 autoClass training

Input:
Data stream S, Ensemble size s, sampling rate w, loss function $\mathcal{L}$, configuration space $\mathcal{A}, \Lambda$
Output:
Set of suited algorithms configurations:
$\mathcal{M} = \{M^{(1)}, \ldots, M^{(s)}\}$

$\mathcal{M} \leftarrow \emptyset$ ▷ Initialization
while $|\mathcal{M}| < s$ AND $\mathcal{M}$ is $\emptyset$ **do**
  $M \leftarrow$ Add($\mathcal{A}, \Lambda$) ▷ Add the algorithms in A with the default parameters
  $\mathcal{M} \leftarrow \mathcal{M} \cup M$
end while
$t \leftarrow 0$
while $HasNext(S)$ **do** ▷ Start the data stream
  $(x, y) \leftarrow Next(S)$
  if $t \mod w == 0$ **then** ▷ Each w instances
    $M^{best} \leftarrow \min_{M \in \mathcal{M}} \mathcal{L}(M(S^T), S^V)$
    $M^{worst} \leftarrow \max_{M \in \mathcal{M}} \mathcal{L}(M(S^T), S^V)$
    $M^{mut} \leftarrow$ Mutate(M^{best})
    $\mathcal{M} \leftarrow \mathcal{M} \cup M^{mut}$ ▷ Add the new generated configuration
    $\mathcal{M} \leftarrow \mathcal{M} \backslash M^{worst}$ ▷ Remove the weakest configuration
  end if
  for $M \in \mathcal{M}$ **do** ▷ Update the ensemble
    M.fit(x, y)
  end for
  $t \leftarrow t + 1$
end while

B. Configuration setup

The most important factors involved in the configuration setup concern (i) the performance metric used to evaluate the

different configurations within $\mathcal{M}$ and (ii) the generation of new configurations, i.e., {algorithms, hyper-parameters}.

Performance metric. The choice of the performance metric is related to the task one wants to address. For example, for clustering, typical measures include the Sum of Squares Error (SSE) and silhouette. For regression, one can use Mean Squared Error (MSE). In our classification case, since we deal with balanced datasets, we adopt the traditional accuracy as a quality measure computed as the number of correct predictions over the total number of predictions.

Generation of new configurations. We build an ensemble of candidate algorithms that contains the set of chosen algorithms with their default hyper-parameter values (lines 8-11, Algorithm 1). This solution is automated, i.e., the configuration (the representative algorithm and its hyper-parameter values) may change over time since data are evolving. So, the currently learned classifiers in $\mathcal{M}$ may no longer be representative of the next upcoming data, thus classifiers and their hyper-parameters need to change. For this to be achieved, we process the stream in sliding window [20] of a fixed size w by keeping the most recent instances from the stream at each time (line 15, Algorithm 1). We evaluate the predictive performance of every algorithm and configuration in the ensemble $\mathcal{M}$ and use the method with the highest accuracy for prediction.

Periodically, at the end of each window, a regression model is applied to the observed performance to predict the performance for unknown configurations. Besides, the best-performing configuration in the ensemble is used to create a new configuration (child) from it using the truncated normal distribution [21]. If the predicted performance is good enough, it replaces the worst configuration in the ensemble (lines 16-20, Algorithm 1). For each w processed instances, we evaluate the predictive performance of the algorithms, as depicted in Fig. 1 (step 1). To obtain new configurations, a configuration is sampled from the ensemble to serve as a parent (step 2) to derive a new configuration from it. The sampling is performed proportionally to the predictive performance of the ensemble members where better-performing algorithms have a higher probability to be selected. For this, we used *roulette wheel selection*, also called *fitness proportionate selection*, commonly used in genetic algorithms for selecting potential configurations in the ensemble. In our case, the fitness function f consists of obtaining the predictive performance of each configuration in order to associate a probability of selection with each configuration. Let us suppose that f_i is the fitness of configuration i in our ensemble $\mathcal{M}$, its probability of being selected is:

$$p_i = \frac{f_i}{\sum_{j=1}^{s} f_j}, \tag{2}$$

where s is the size of the ensemble and j represents the jth configuration. Moreover, we calculate the Cumulative Probability (CP), q_i, for each learner within the ensemble as follows:

$$q_i = \sum_{j=1}^{i} p_j. \tag{3}$$

Then, we generate a uniform random number $r \in [0, 1]$ and compare it with the CP. Based on this comparison, we, therefore, make a decision about which of the configuration c_i is going to be selected as a parent as follows:

1) if $r < q_1$ then select the first configuration c_1,
2) else, $q_{i-1} < r \leq q_i$, select c_i.

We do not necessarily choose the best-performing configuration as in *rank selection* in order to increase the ensemble diversity and avoid having the same algorithm with different hyper-parameter values in our ensemble. The sampled algorithm and configuration are therefore used as a parent to create an offspring and allow for mutations, as shown in Fig. 1, step 3. Similarly to confStream [15], [22], we consider that every parameter of every configuration maintains an associated probability distribution. Every numerical parameter has a truncated normal numerical distribution $\mathcal{N}(\mu, \sigma)$, where μ is the expectation and σ is the standard deviation that is derived from a normal distribution by bounding the random variable within a limited range of values. This truncated distribution is derived from a normal distribution by bounding the random variable within a limited range of values.

For a new configuration, we place μ at the position of the parent. Thereafter, the standard deviation σ of the child is reduced to increase the exploitation of this area using a fading factor λ through a decay function as follows:

$$\sigma_{t+1} = \sigma_t \times 2^{-\lambda}, \tag{4}$$

where σ_t and σ_{t+1} are the standard deviation of the parent and child, respectively.

For integer parameters, the same process is performed to do the sampling but the result is rounded to the nearest integer. Ordinal parameters are a kind of categorical data with a set order that is processed similarly to integer parameters using an integer sampling strategy where the sampled integer is used as the index in the list of possible ordinal values. An instance example of ordinal parameters from decision trees is the split criterion parameter such as *Gini*, *Information gain*, *Standard deviation reduction*, and *Variance reduction* to which is assigned a numerical value from 1-4 as the index. For categorical parameters, we maintain a list of probabilities for each outcome and start with equal probabilities. The new configuration is sampled based on the latter list and the probability of its corresponding category is increased to serve exploitation. Similarly to confStream, the probability is increased by the same quantity that we reduce σ with numerical parameters as follows:

$$p_{t+1} = p_t \times (2 - 2^{-\lambda}). \tag{5}$$

To obtain the final probabilities, the values are therefore normalized to sum one.

TABLE I: Overview of the datasets.

Dataset	#Inst	#Att	#Classes	Type
AGR_a	1,000,000	9	2	Synthetic
AGR_g	1,000,000	9	2	Synthetic
RBF_f	1,000,000	10	5	Synthetic
RBF_m	1,000,000	10	5	Synthetic
Hyper	1,000,000	10	2	Synthetic
CNAE	1,080	856	9	Real
Poker	829,201	10	10	Real
Covtype	581,012	54	7	Real
KDD	4,898,431	41	23	Real
Airlines	539,383	7	2	Real

IV. Experimental study

We present experiments to show the abilities of our proposal in the three axes: (i) the accuracy obtained as the final percentage (%) of instances correctly classified; (ii) the memory in megabytes (MB) used to maintain the learned models or/and statistical information about the models; and (iii) the running time in seconds (s) required to learn and predict from data and also for tuning.

A. Datasets

In our experiments, we used a diverse set of both synthetic and real datasets in different scenarios. We used data generators and real data, most of which have been thoroughly used in the literature [18], [19], [23]–[25] to assess the performance of data stream classification algorithms used notably in the original papers introducing the baseline algorithms [14], [16], [18], [19], [26]. Table I shows an overview of the datasets used for experiments and further details are provided in what follows.

AGR. The AGR_a and AGR_g datasets are based on the AGRAWAL generator [27] which creates data streams with 9 attributes and 2 classes. A perturbation factor is used to change the original value of the data by adding a deviation value to it. AGR is used to simulate 3 gradual drifts in the generated stream. AGR_g simulates 3 gradual drifts, while AGR_a simulates 3 abrupt drifts.

RBF. The RBF_f and RBF_m datasets were generated using the Radial Basis Function (RBF) generator. The latter creates centroids at random positions, and each one has a standard deviation, a weight and a class label. This dataset simulates drift by moving the centroids at a constant speed. Both RBF_m and RBF_f were parametrized with 50 centroids and all of them drift. RBF_m simulates a "moderate" incremental drift (speed of change set to 0.0001) while RBF_f simulates a more challenging "fast" incremental drift (speed of change set to 0.001).

Hyper. The Hyperplane generator [28] is used to generate streams with gradual concept drift by changing the values of its weights. We parameterize HYP with 10 attributes and a magnitude of change equal to 0.001.

CNAE. CNAE is the national classification of economic activities dataset, initially used in [29]. It contains 1,080 instances, each of 856 attributes, representing descriptions of Brazilian companies categorized into 9 classes. The original texts were preprocessed to obtain the current highly sparse dataset.

Poker. The Poker hand dataset consists of 829,201 instances and 10 attributes. Each instance of the Poker-Hand dataset is an example of a hand consisting of five playing cards.

Covertype. The forest covertype dataset was obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. It contains 54 attributes and 7 classes.

KDD99. KDD cup'99[2] for network intrusion detection. This dataset contains 41 attributes and 23 classes. It has been often used to evaluate big data stream algorithms' performance.

Airlines. The Airlines dataset was inspired by the regression dataset from Ikonomovska[3]. The task is to predict whether a given flight will be delayed given information on the scheduled departure. Thus, it has 2 possible classes: delayed or not delayed. This data set contains $539,383$ records with 7 attributes.

TABLE II: The algorithms and their hyper-parameter ranges.

Algorithm	Parameter	Type	default	range
HT	g	Integer	200	[10,200]
	c	Numeric	0.01	[0,1]
HAT	g	Integer	200	[10,200]
	c	Numeric	0.01	[0,1]
kNN	k	Integer	10	[2,30]
kNN^a	k	Integer	10	[2,30]

B. Search space

We aim to select the optimal algorithm and its hyper-parameterization among the available stream classification algorithms in MOA [2] for a specific stream. To do so, we consider a representative subset of the baseline classifiers and particularly use state-of-the-art classification algorithms [20]: Hoeffding Tree (HT) [16], Hoeffding Adaptive Tree (HAT) [30], kNN, and kNN with ADWIN (kNN^a) [26] to handle concept drifts. For each algorithm, we aim to optimize the parameters that influence the classification prediction. For example, the kNN algorithm can be parameterized by the number of neighbors k. The details on the hyper-parameters and their ranges for every algorithm used are explained in Table II. Besides, more algorithms and hyper-parameters can be tuned and considered in this approach.

C. Experimental setup

To evaluate our approach, we implemented it in Java as a classification algorithm under the MOA software. We evaluated our approach using a population size $s = |\mathcal{M}| = 10$ and a window of size $w = 1000$, where we evaluate our s learners for each w instances. All algorithms are trained in an ensemble, so a high number of learners in the ensemble or

[2]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[3]https://www.openml.org/search?type=data&sort=runs&id=42493&status=active

big window (w) would lead to expensive computation resource usage. The parameters s and w have been fixed based on an evaluation that has been done during the implementation of this approach and supported by previous empirical studies [18], [26]. We show an extensive evaluation against well-known state-of-the-art single algorithms that are widely used in comparison for the data stream classification task and also ensemble algorithms, such as Adaptive Random Forest [18], Stream Random Patches (SRP) [19], and Leveraging Bagging (LB) [17]. While current ensemble-based methods are formed of homogeneous algorithms, autoClass is able to handle diverse algorithms and pipelines. We also compare against the unique, as far as we know, automated solution for stream classification, EvoAutoML [14].

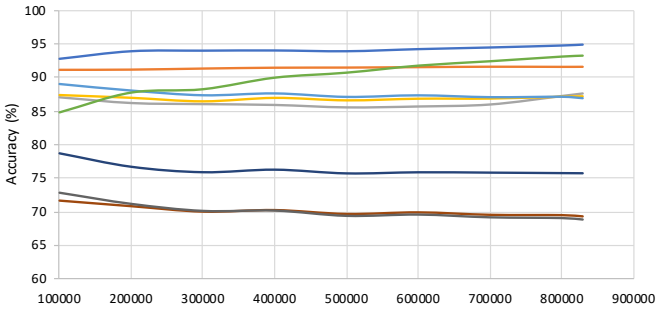
For a fair comparison, we used 10 learners for the ensemble-based competitors as the population size s of our approach with their default configuration as presented in Table II.

D. Results

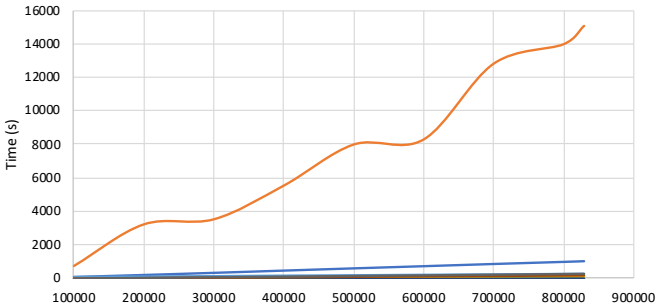
In this section, we present the results of our approach and compare them against those of state-of-the-art classifiers. To do so, autoClass is initialized with the same default hyper-parameterization of the baselines, as shown in Table II, and optimizes the parameters ahead. In Fig. 2, we depict the performance of autoClass and the baselines for the Poker dataset and show its ability to adapt and react to temporal changes along the stream. One can see that the accuracy of the competitors decreases at $\sim$ 200000 evaluated instances while autoClass remains steady. AutoClass is more accurate and more efficient in terms of memory than EvoAutoML, LB, ARF, SRP, and HT. However, in terms of running time, single-based algorithms (e.g., HT and kNN) obtain better performance (lower values). Moreover, autoClass takes more time for learning and prediction than ensemble-based learners because, in addition, it does hyper-parameters optimization and adapts them while the data stream evolves. We also point out that the only automated solution for stream classification so far, EvoAutoML, is more costly in terms of resources than our autoClass and other baselines.

resources.

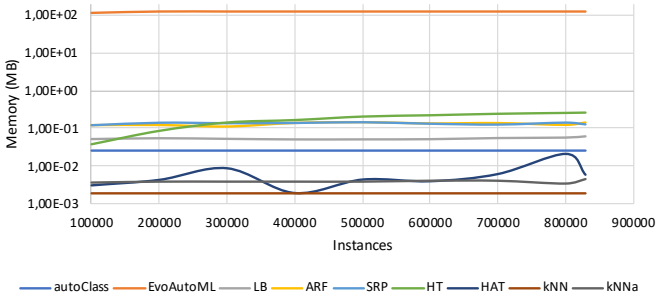
Table III shows that autoClass is very competitive in terms of predictive performance in comparison with competitors. We can see that on the overall average accuracy of 87.43% autoClass outperforms the baselines. In comparison with the top single classifiers, autoClass performs, at least, 10% better on average. It obtains the best accurate performance on almost all the datasets except for Hyper, CNAE, and Covertype data it achieves the second-best result where the difference with the highest accuracy is not huge. It is very competitive and obtains better results than well-known ensemble-based methods such as LB, ARF, and SRP because our approach is as diverse as these aforementioned ensemble methods. AutoClass is effective with datasets that contain different types of concept drift as AGR and RBF thanks to the dynamic and adaptive configuration strategy. The latter changes the best-performing method used for prediction when its accuracy



(a) Accuracy.



(b) Time.



(c) Memory.

Fig. 2: Accuracy, time, and memory performance for autoClass and baselines.

decreases because of the drift. Moreover, through its additional hyper-parameter optimization component, autoClass achieves better predictive performance than its rival EvoAutoML [14] on all datasets taking into account that the latter is based on an evolutionary strategy where methods and configurations are fixed at the beginning without tuning.

In Fig. 3, we show the time needed to predict and learn from different data streams. It shows that the better predictive performance of ensemble-based approaches is accompanied by higher time consumption than with single algorithms. However comparing the consumption of ensemble classifiers, we can observe that EvoAutoML has undeniably the highest time consumption of all the datasets. AutoClass consumes more time than LB, ARF, and SRP methods on almost all the

TABLE III: Accuracy comparison.

Dataset	autoClass	EvoAutoML	LB	ARF	SRP	HT	HAT	kNN	kNN^a
AGR$_a$	**92.21**	86.17	89.65	85.51	90.50	88.80	91.06	84.71	85.83
AGR$_g$	**88.46**	85.91	83.86	80.48	86.71	81.06	87.74	82.43	83.48
RBF$_f$	**86.37**	85.34	55.43	71.09	70.83	33.79	48.68	84.03	80.70
RBF$_m$	**90.39**	85.59	83.88	84.69	83.26	53.53	73.12	89.35	89.95
Hyper	<u>86.59</u>	85.19	**87.56**	84.54	81.85	76.93	84.97	83.32	83.87
CNAE	<u>73.33</u>	39.62	55.92	55.46	**76.20**	55.92	55.83	71.75	69.07
Poker	**95.21**	91.49	87.59	87.11	86.89	94.91	77.85	76.57	76.48
Covertype	<u>94.46</u>	94.29	91.69	94.39	**95.04**	83.62	86.99	92.59	92.10
KDD	**99.98**	99.97	99.96	99.96	99.97	99.91	99.91	99.97	99.97
Airlines	**67.34**	61.80	63.14	61.34	61.29	61.02	61.56	59.54	60.16
Overall ∅	**87.43**	81.54	79.87	80.46	83.25	72.95	76.77	82.43	82.16

Bold and underlined values indicate the best and the second best results per dataset, respectively.

TABLE IV: Memory comparison.

Dataset	autoClass	EvoAutoML	LB	ARF	SRP	HT	HAT	kNN	kNN^a
AGRa	0.025	248.63	0.066	0.129	0.127	0.637	0.703	0.002	0.005
AGRg	0.024	289.80	0.065	0.130	0.124	0.679	0.661	0.002	0.005
RBFf	0.025	126.35	0.052	0.118	0.106	0.103	0.015	0.002	0.005
RBFm	0.025	127.17	0.059	0.116	0.122	0.200	0.127	0.002	0.006
Hyper	0.025	296.52	0.064	0.110	0.119	0.612	0.004	0.002	0.005
CNAE	0.096	242.42	0.105	0.177	0.239	0.001	0.002	0.002	0.054
Poker	0.024	125.27	0.060	0.142	0.123	0.248	0.006	0.002	0.004
Covertype	0.025	129.22	0.062	0.125	0.109	0.246	0.008	0.008	0.011
KDD	0.027	129.06	0.067	0.195	0.176	0.035	0.003	0.003	0.006
Airlines	0.025	150.96	0.053	0.148	0.107	0.623	0.112	0.002	0.004
Overall ∅	0.032	186.54	0.065	0.139	0.135	0.338	0.364	0.002	0.010

datasets due to the hyper-parameter tuning component where we need to optimize the configurations to obtain the optimal method with the current concept.

In terms of memory consumption, as highlighted in Table IV, single classifiers, mainly kNN and kNN^a, are more efficient because they only maintain a fixed size sliding window over the whole stream. Similar to the time behavior, autoClass is more efficient than EvoAutoML, LB, ARF, and SRP for almost all the datasets because it uses neighbor-based approaches in its ensemble $\mathcal{M}$ which turns out to be memory efficient since the size of the window limits them. It performs even better than the single HT and HAT which may be the result of (i) the existence of drifts as revealed AGR and RBF datasets which implies that the trees continue to grow up while with autoClass, a new learn starts empty; and/or (ii) our ensemble of size s is composed of multiple kNN-based methods which consume few

AutoClass meets the data stream resource requirements by providing high predictive performance and consuming less memory and time than EvoAutoML while consuming relatively more in comparison with the non-automated single and ensemble state-of-the-art algorithms. Nevertheless, autoClass outperforms these baselines, in terms of predictive performance, in a common test-then-train evaluation which shows its ability to adapt to changes in data distributions thanks to the employed hyper-parameters tuning mechanism.

ACKNOWLEDGEMENTS

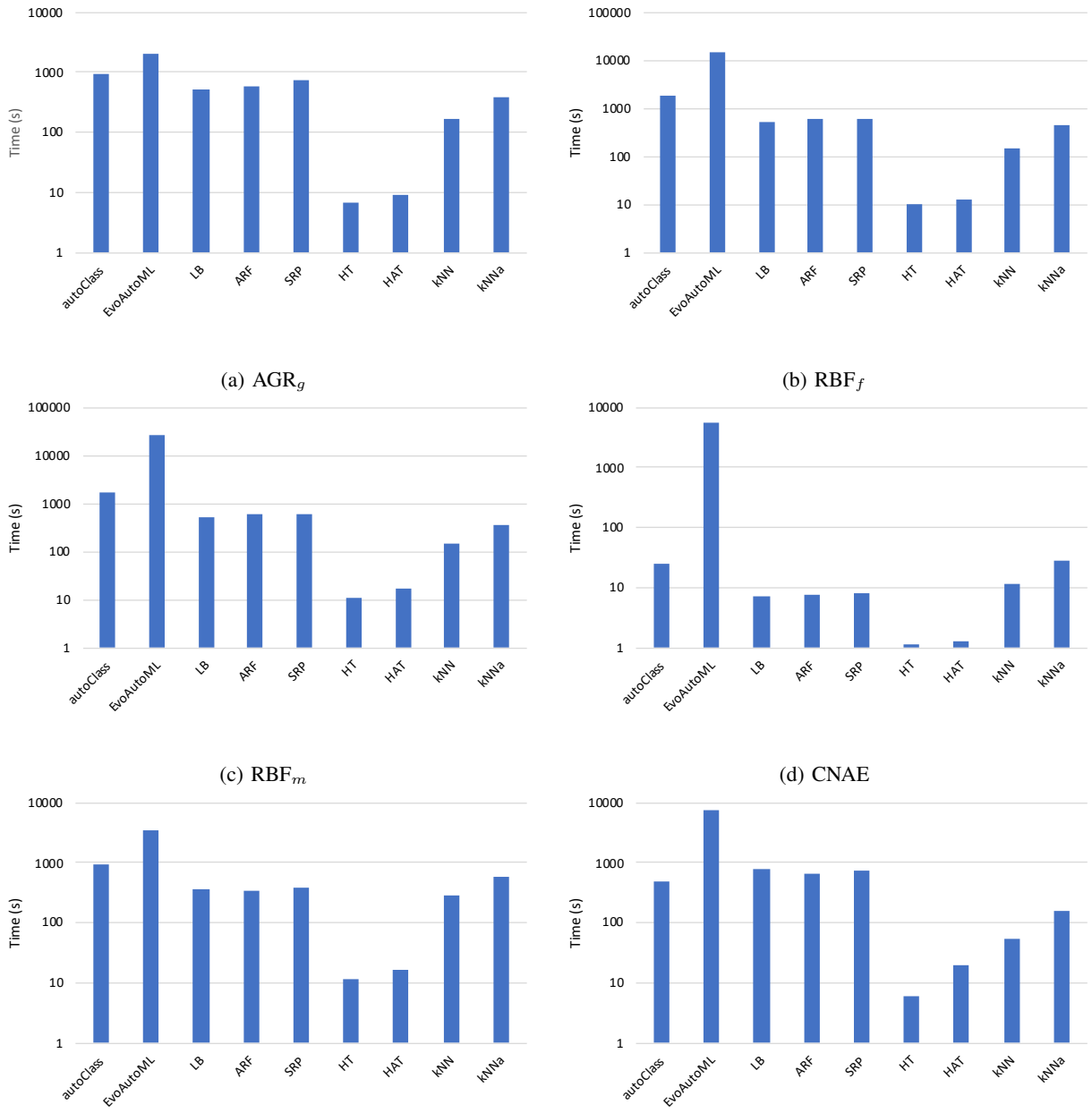
This work has been partially supported and funded by the EU Horizon Europe SEDIMARK (SEcure Decentralised Intelligent Data MARKetplace) project (Grant no. 101070074).

V. CONCLUDING REMARKS AND FUTURE DIRECTIONS

In this paper, we proposed an automated approach for data stream classification that automatically tunes the hyper-parameters of stream classifiers and dynamically adapts to the eventual changes in data distributions. AutoClass outperforms the only existing automated solution to the best of our knowledge for the classification of data streams, EvoAutoML, in terms of accuracy and resources. Moreover, it obtains better predictive performance than state-of-the-art algorithms on almost all datasets while providing a good tradeoff accuracy-resources. Other stream classifiers can be easily added to this automated solution.

REFERENCES

[1] M. Bahri, F. Salutari, A. Putina, and M. Sozio, "Automl: state of the art with a focus on anomaly detection, challenges, and research directions," *IJDSA*, pp. 1–14, 2022.



(a) AGR$_g$

(b) RBF$_f$

(c) RBF$_m$

(d) CNAE

(e) Covertype

(f) Airlines

Fig. 3: Running time performance for autoClass and baseline approaches.

[2] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: Massive online analysis," *JMLR*, vol. 11, no. May, pp. 1601–1604, 2010.

[3] J. Montiel, M. Halford, S. M. Mastelini, G. Bolmier, R. Sourty, R. Vaysse, A. Zouitine, H. M. Gomes, J. Read, T. Abdessalem *et al.*, "River: machine learning for streaming data in python," 2021.

[4] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Combined selection and hpo of classification algorithms," in *ACM SIGKDD*, 2013.

[5] M. Feurer and F. Hutter, "Hyperparameter optimization," in *Automated Machine Learning*, 2019, pp. 3–33.

[6] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, 2021.

[7] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-weka 2.0: Automatic model selection and hpo in weka," *JMLR*, 2017.

[8] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter, "Auto-sklearn 2.0: The next generation," *arXiv preprint arXiv:2007.04074*, 2020.

[9] P. Gijsbers and J. Vanschoren, "Gama: genetic automated machine learning assistant," *Journal of Open Source Software*, vol. 4, no. 33, p. 1132, 2019.

[10] T. T. Le, W. Fu, and J. H. Moore, "Scaling tree-based automated machine learning to biomedical big data with a feature set selector," *Bioinformatics*, vol. 36, no. 1, 2020.

[11] E. LeDell and S. Poirier, "H2O AutoML: Scalable automatic machine learning," *ICML Workshop on Automated Machine Learning (AutoML)*, July 2020.

[12] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning*,

2019.
[13] J. N. van Rijn, G. Holmes, B. Pfahringer, and J. Vanschoren, "The online performance estimation framework: heterogeneous ensemble learning for data streams," *ML*, 2018.
[14] C. Kulbach, J. Montiel, M. Bahri, M. Heyden, and A. Bifet, "Evolution-based online automated machine learning," in *PAKDD*. Springer, 2022, pp. 472–484.
[15] M. Carnein, H. Trautmann, A. Bifet, and B. Pfahringer, "confstream: Automated algorithm selection and configuration of stream clustering," in *LION*, 2020.
[16] P. Domingos and G. Hulten, "Mining high-speed data streams," in *SIGKDD*, 2000.
[17] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *ECML PKDD*. Springer, 2010, pp. 135–150.
[18] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem, "Adaptive random forests for evolving data stream classification," *Machine Learning*, vol. 106, no. 9, pp. 1469–1495, 2017.
[19] H. M. Gomes, J. Read, and A. Bifet, "Streaming random patches for evolving data stream classification," in *ICDM*. IEEE, 2019, pp. 240–249.
[20] M. Bahri, A. Bifet, J. Gama, H. M. Gomes, and S. Maniu, "Data stream analysis: Foundations, major tasks and tools," *WIREs DMKD*, vol. 11, no. 3, p. e1405, 2021.
[21] C. P. Robert, "Simulation of truncated normal variables," *Statistics and Computing*, vol. 5, no. 2, pp. 121–125, 1995.
[22] 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.
[23] M. Bahri, S. Maniu, and A. Bifet, "Sketch-based naive bayes algorithms for evolving data streams," in *IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 604–613.
[24] M. Bahri, S. Maniu, A. Bifet, R. F. de Mello, and N. Tziortziotis, "Compressed k-nearest neighbors ensembles for evolving data streams," in *ECAI*, 2020.
[25] M. Bahri, B. Pfahringer, A. Bifet, and S. Maniu, "Efficient batch-incremental classification using umap for evolving data streams," in *Advances in Intelligent Data Analysis XVIII: 18th International Symposium on Intelligent Data Analysis, IDA 2020, Konstanz, Germany, April 27–29, 2020, Proceedings 18*. Springer, 2020, pp. 40–53.
[26] A. Bifet, B. Pfahringer, J. Read, and G. Holmes, "Efficient data stream classification via probabilistic adaptive windows," in *SAC SIGAPP*. ACM, 2013, pp. 801–806.
[27] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: A performance perspective," *IEEE transactions on knowledge and data engineering*, vol. 5, no. 6, pp. 914–925, 1993.
[28] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2001, pp. 97–106.
[29] P. M. Ciarelli and E. Oliveira, "Agglomeration and elimination of terms for dimensionality reduction," in *The ninth International Conference on Intelligent Systems Design and Applications*. IEEE, 2009, pp. 547–552.
[30] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *Intelligent Data Analysis (IDA)*. Springer, 2009, pp. 249–260.