Data Mining and Knowledge Discovery (2024) 38:1289–1315
https://doi.org/10.1007/s10618-023-00997-7

Improving hyper-parameter self-tuning for data streams by adapting an evolutionary approach

Antonio R. Moya[1] · **Bruno Veloso**[2,3] · **João Gama**[2,3] · **Sebastián Ventura**[1]

Received: 9 February 2023 / Accepted: 25 November 2023 / Published online: 21 December 2023

Abstract

Hyper-parameter tuning of machine learning models has become a crucial task in achieving optimal results in terms of performance. Several researchers have explored the optimisation task during the last decades to reach a state-of-the-art method. However, most of them focus on batch or offline learning, where data distributions do not change arbitrarily over time. On the other hand, dealing with data streams and online learning is a challenging problem. In fact, the higher the technology goes, the greater the importance of sophisticated techniques to process these data streams. Thus, improving hyper-parameter self-tuning during online learning of these machine learning models is crucial. To this end, in this paper, we present MESSPT, an evolutionary algorithm for self-hyper-parameter tuning for data streams. We apply Differential Evolution to dynamically-sized samples, requiring a single pass-over of data to train and evaluate models and choose the best configurations. We take care of the number of configurations to be evaluated, which necessarily has to be reduced, thus making this evolutionary approach a micro-evolutionary one. Furthermore, we control how our evolutionary algorithm deals with concept drift. Experiments on different learning tasks and over well-known datasets show that our proposed MESSPT outperforms the state-of-the-art on hyper-parameter tuning for data streams.

Keywords Data streams · Concept drift · Optimisation · Hyper-parameters · Evolutionary algorithms

Responsible editor: Johannes Fürnkranz.

✉ Antonio R. Moya
amoya@uco.es

[1] Department of Computer Science and Numerical Analysis, Andalusian Research Institute in Data Science and Computational Intelligence (DaSCI), University of Cordoba, Cordoba, Spain

[2] INESC TEC, Porto, Portugal

[3] FEP, University of Porto, Porto, Portugal

1 Introduction

The evolution of technology leads to increased data generation, enabling tasks such as predictive maintenance, healthcare monitoring or automation. Thus, learning and extracting knowledge from these data streams is highly interesting for the private and public sectors. On the other hand, the automation of machine learning (AutoML) has grown during the last two decades, focusing on automatically adapting algorithms to specific problems. In this scope, hyper-parameter self-tuning has reached high interest due to improved results. Researchers used algorithms such as grid-search (Lerman 1980), random-search (Bergstra and Bengio 2012), Bayesian Optimisation (Mockus et al. 1978), or Evolutionary Algorithms (EAs) (Bäck 1996) to optimise the hyper-parameters of their algorithms.

However, automation is usually adapted to offline processing, requiring training using offline data batches. These supervised models typically do not react to the appearance of concept drift (Gama et al. 2014). In order to keep the model adjusted to the current data distribution, restarting the tuning process is necessary every time the concept drift detector generates an alarm. Thus, online AutoML advances, particularly in hyper-parameter self-tuning, are highly interesting. Although the number of works in this scope is very scarce, some remarkable ideas exist. Some focus on analysing key aspects and challenges to be solved while working with evolving data streams (Bahri et al. 2021).

There are different options between the online tuning proposals to deal with concept drift. Some ideas use an approach based on traditional offline learning without attending to concept drift but adapting it to an incremental process (Bakhashwain and Sagheer 2020). Another option is related to restarting hyper-parameter tuning after concept drift (Veloso et al. 2021). Finally, another interesting approach uses meta-learning to update the tuning process when concept drift occurs. It employs previous knowledge to continue online training after this drift (Lacombe et al. 2021; Lobo et al. 2021).

Regarding evolution-based models for hyper-parameter optimisation, they are easy to adapt to online scenarios because they are not gradient-based algorithms. Furthermore, against other algorithms, they achieve a good trade-off between exploration and exploitation, which is crucial to this online training in which changes over data distributions could lead to high jumps into search space to achieve a local minimum. These evolution-based algorithms can change from one to another subspace by using offspring and mutations and, inside each of them, exploit to best solutions. Hence, some authors have adapted different evolutionary algorithms to online learning (Bakhashwain and Sagheer 2020; Kulbach et al. 2022).

We propose adapting Differential Evolution (DE) to online learning, allowing hyper-parameter self-tuning and taking advantage of the evolutionary benefits stated above. We focus on some critical points inside this scope: single-pass-over data (instead of using offline batches), detection and reaction to concept drift,

and proper development of our DE algorithm to achieve fast convergence without losing exploration capacity. Finally, considering the importance of reducing the computational time and cost as much as possible, Micro-Evolutionary Single-pass Self-hyper-Parameter Tuning (MESSPT) could be considered a micro-evolutionary approach, following similar ideas to a micro-genetic algorithm which proved to have good results (Coello and Pulido 2001). The main advantage of adopting evolutionary algorithms against other optimisers which allow fast convergence, as Nelder-mead, is that they can lead better with the multi-local minimum problem.

The key contributions of our proposal are the following:

1. A novel evolutionary algorithm to solve the hyper-parameter self-tuning task on online learning.
2. A solution to deal with concept drift when it occurs before the convergence of the tuning algorithm.
3. Extensive and robust evaluation of MESSPT on two different prediction tasks (regression and classification).

The document contains five sections. The related work (Sect. 2) focuses on hyper-parameter optimisation algorithms. Section 3 describes the MESSPT optimisation algorithm. Section 4 presents the experiments and results. Finally, Sect. 5 will gather the significant conclusions.

2 Related work

Regarding the related work, we focus on state-of-the-art evolutionary algorithms (especially DE), concept-drift detectors and online hyper-parameter tuning algorithms.

Evolutionary algorithms appeared in the 1960 s when some new techniques adapted the principles of natural selection to solve global optimisation problems. The number of problems they solved and the strategies included inside this scope make possible the existence of many surveys on evolutionary algorithms. For instance, Guliashki et al. (2009) review the use of EAs for multi-objective optimisation; Hruschka et al. (2009) cover the EAs employment for clustering; and a last example, Barros et al. (2012) made a survey of EAs for decision-tree induction. The first proposals (Zhan et al. 2022) inside EAs were related to Evolutionary Programming, Evolutionary Strategies, Genetic Programming and Genetic Algorithms (Koza 1995; Galletly 1998). Other algorithms could be highlighted, such as Differential Evolution (Das and Suganthan 2011), or Estimation Distribution of Algorithm (Hauschild and Pelikan 2011). Nonetheless, within the high applicability of this type of optimisation algorithms, we can found two literature gaps that we want to mention: i) the scalability and efficiency problem; in this work, we are trying to

explore the adoption of micro-evolutionary algorithms to improve the scalability; and ii) dynamic and uncertain environments: in this work, we developed a solution to optimise a stream-base model, where the optimal solution may change over time or where the problem parameters are not precisely known.

Concept drift refers to the phenomenon where the statistical properties of a target variable change over time. In other words, it occurs when the relationships between the input features and the target variable evolve, leading to a degradation in the performance or accuracy of a predictive model. There are different types of concept drifts: gradual, incremental, recurrent and abrupt concept drifts. Detecting and adapting to concept drift is crucial for maintaining the effectiveness of machine learning models in dynamic and evolving scenarios.

There are several existing solutions in the literature for drift detection. One widely known method is the Drift Detection Method (DDM) proposed by Gama et al. (2004), which monitors the learner's error rate. It assumes that the error rate should decrease as more examples are observed, indicating a stationary data distribution. An alarm is triggered if the error rate surpasses a specific threshold calculated based on the error rate and standard deviation at a particular moment. Baena-Garcıa et al. (2006) introduced the Early Drift Detection Method (EDDM) to improve detection performance under gradual concept drifts. This variant utilizes running average distance and running standard deviations instead of fixed-time calculations.

Another approach is the Hoeffding Drift Detection Method (HDDM) proposed by Frias-Blanco et al. (2014). HDDM employs Hoeffding's inequality and a moving average test to identify concept drifts. Sebastião and Fernandes (2017) introduced the Page-Hinkley method, which employs the CUSUM control chart to detect changes. It calculates observed values and their mean until the current moment and triggers an alarm if a change in mean (either increase or decrease) surpasses a predefined threshold.

Additionally, the Adaptive Windowing method (ADWIN) proposed by Bifet and Gavalda (2007) utilizes a dynamic-sized window that retains recent examples with the same data distribution. The window can be split into two sub-windows for comparison, and an alarm is raised if the data distribution deviates. Although this study focuses not on developing new drift detection methods, it aims to utilize these methods to identify data distribution changes and reinitiate the optimisation process.

On the other hand, the interest in online hyper-parameter tuning dealing with data streams is on the rise nowadays. Zhan et al. (2018) and Lin et al. (2019) adapted hyper-gradient-based optimisation to online learning. However, the authors do not consider concept drifts effects in both approaches. Imbrea (2021) applies state-of-the-art AutoML tools to real streamed data and concludes how necessary adaptation to concept drift is to deal with these data. Veloso et al. (2018) proposed SPT, which applies Nelder-Mead (Nelder and Mead 1965) to dynamically sized windows during online training to detect the best configurations at each step. A refined version proposed by Veloso et al. (2021), uses a single-pass algorithm (against a double-pass). In SSPT it is highlighted the availability of this algorithm to react to concept drifts in (near) real-time.

Bahri et al. (2021) analysed foundations and challenges towards AutoML applied on data streams. Celik and Vanschoren (2021) studied how some AutoML methods (Bayesian Optimisation, random search and evolutionary computation) respond to different types of concept drift (abrupt, gradual, etc.). They propose six adaptation strategies to address the concept drift problem: two strategies based on detection and retraining (warm-start and from scratch); two based on detection and re-run of the AutoML technique (warm-start and from scratch); and finally, two strategies without drift detector, which are based on periodic restarting and training once. Regarding the hyper-parameters self-tuning, SOKNL (Sun et al. 2022) introduces the extension of Adaptive Random Forest Regression for streaming regression problems. To outperform this algorithm, they consider dynamically and automatically tuning the k value of trees which will participate in the prediction step. To this end, k most relevant leaves are selected, considering the centroids stored within each leaf.

Lacombe et al. (2021) proposed a framework to recommend hyper-parameter values from previous knowledge obtained. They optimised the hyper-parameters of the Adaptive Random Forest (predictive classifier) and the drift detector. For this purpose, a multi-objective function considers the evaluation of inversely correlated metrics as performance vs resource consumption. Therefore, a meta-knowledge based model adapts the best hyper-parameters in the initialization and after each drift detected. Related to adapting hyper-parameters after drift detection, Lobo et al. (2021) made two proposals in this scope, which create a grid of possible suitable configurations from the current one (for instance, creating a neighbourhood) and evaluate them in sliding windows to select the best one to continue training.

Lastly, we will highlight two works that use evolution-based optimisation methods for online learning. Bakhashwain and Sagheer (2020) presented an online tuning approach for the hyper-parameters of a deep long short-term memory model (LSTM) in a dynamic fashion. They applied a genetic algorithm to this hyper-parameter optimisation problem. They focused on how their proposal outperformed the static approach, but they did not deal with concept drift. Kulbach et al. (2022) considered adapting the CASH problem to an online scenario. They used a genetic algorithm to adapt these hyper-parameters during training on evolving data streams. The difference in this work is that the learning model detects and adapts to the concept drift. In our proposal, we use the drift detector to adjust the hyperparameters.

Summarising, the available online optimisation solutions can be grouped into static and dynamic strategies. In the first case, the solutions are restricted to periodic or single optimisation, and in this case, all the common optimisation strategies can be applied to a batch of data. This raises several issues, like which is the proper size for this batch of data. Can this data well represent the data distribution over the data stream? Another problem is when is the right moment to re-optimise the learning model. Using a periodic re-optimisation can be problematic. What is the right time interval? The second group of online optimisation methods tried to solve at least when is the right moment to optimise. They achieved this goal using a drift detector to identify degradation in the learning process and then restart the optimisation process. However, there are limited options, and they mainly suffer from performance

issues in the exploration phase. This performance issue is due to the data stream's dynamic nature and multiple local minimum presences. Based on this gap, we face the adaptation of evolutionary algorithms to data streams to minimize the problem identified in the exploration phase. It allows better exploration of the search space and escape better from the fall to local minimums. It achieves this without losing the exploitation of the most promising areas. It usually requires high computational cost, but the way in which we will configure our MESSPT proposal will lead to faster convergence (we will show it in Sect. 4.2). Moreover, we propose a little solution to premature concept drift appearance. This solution is based on the addition of a random solution at each step and it will be explained during Sect. 3.2. It allows better behavior for premature concept drifts as will be demonstrated in 4.2.3. In short, we will follow a dynamic strategy which will adapt an evolutionary algorithm to solve some critical weaknesses of dynamic state-of-the-art proposals.

Thus, from the literature, we can conclude the following considerations:

- Hyper-parameter self-tuning for data streams is an increasing topic in which the related work is relatively scarce. Thus, a proposal to outperform its state-of-the-art is of high interest.
- Dealing with concept drifts is crucial for working on data streams. Their detection, and decisions taken after it, are one of the key points to these types of proposals.
- Evolutionary algorithms have shown good performance in many problems related to hyper-parameter optimisation. This point, joined with their capacity to keep a trade-off between exploration at each step (necessary in changing data distributions) and exploitation (necessary to access to best solutions), made them optimal to be used to hyper-parameter self-tuning for data streams.
- Time consumption is critical, too, in an online training scenario. That could difficult the use of evolutionary algorithms for this purpose. It is necessary a proper adjustment to achieve fast convergence and not to use too big populations.

3 Micro-evolutionary self-hyper-parameter tuning

The problem of hyper-parameter tuning can be formulated by: A data stream s, a machine learning algorithm A with its hyper-parameters A_{hp} (hp of the hyper-parameters extracted from a set of possible hyper-parameters H) and $L(\_, \_, \_)$ a loss metric. Considering a batch of n-dimensional elements $x_i \in \mathbb{R}^d$ with $i = 1, \ldots, n$ extracted at each step from stream s and the target value related to each element y_i, our objective is to find the best A^*_{hp} which satisfies that: $A^*_{hp} = \underset{hp \in H}{argmin} \frac{1}{n} \sum_{i=1}^{n} L(A_{hp}, x_i, y_i)$.

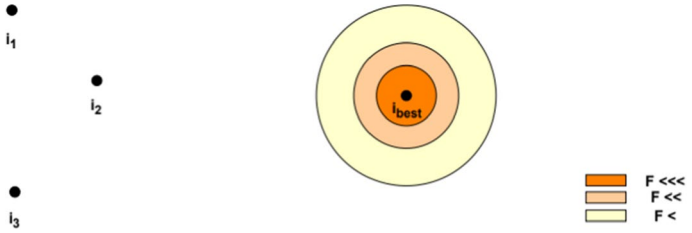


Fig. 1 Case $CR = 1$. Given the i_j configurations in the current generation, new individuals will be in colored regions depending on how lower is F. The lower F is, the closer to current best that offsprings will be

3.1 Online differential evolution optimisation algorithm

We carefully adapt DE to online learning scenarios, keeping the trade-off between exploitation and exploration of the search space. To this end, firstly, mutation and crossover operators have to be defined. The algorithm initially explores the search space and, after some steps, converges to a good solution. For mutation, the algorithm applies the rand-to-best mutation operator. Thus, the best solution will greatly influence the next generation. Therefore, given a number of individuals in our population n, an individual x_i with $i \in \{1, \ldots, n\}$ and a generation G, we define the mutation of this individual as $u_i^G = x_{best}^G + F(x_{r_1} - x_{r_2})$ with $F \in [0, 2]$,$r_1 \neq r_2$ and both not equal to i. On the other hand, classical crossover operation is used. Thus, given an individual x_i and its mutation u_i, the new individual will follow the next equation:

$$x_{i_j}^{G+1} = \begin{cases} u_{i_j}^G, & \text{if } r \leq CR \text{ or } l = j \\ x_{i_j}^G & \text{otherwise} \end{cases} \tag{1}$$

being r a random number (from an uniform distribution) between 0 and 1; $CR \in [0, 1]$; and $l \in \{1, \ldots, m\}$ (m is the total number of gens inside each individual) the gen that for sure will be changed at this step.

Another critical point is the number of configurations in each population and the number of generations or stopping criteria. In real-time training, a fast convergence becomes crucial. Therefore, the number of individuals and generations should be as low as possible. Some works show that a micro-evolutionary approach can achieve a good performance (Coello and Pulido 2001). Considering that we need a low computational cost, we will follow a micro-evolutionary approach. To this end, we will use four individuals per population. Regarding the number of generations and stopping criteria, we have to configure our algorithm to achieve a fast convergence while keeping the exploration availability. With this objective, we applied the following vital decisions:

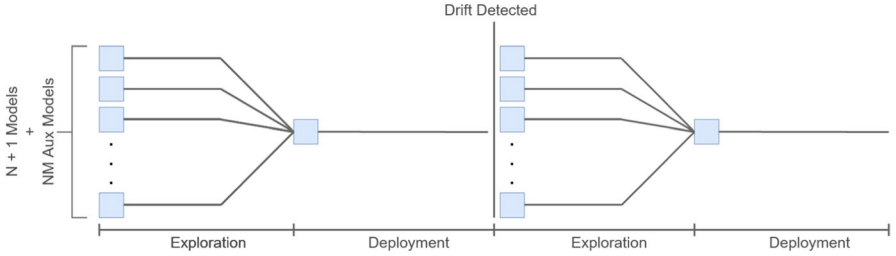


Fig. 2 Two operation phases (Veloso et al. 2021)

- *F* and *CR* values will be initialised to medium values (i.e. 0.5 both). It allows starting the process focusing on ensuring exploration.
- *F* and *CR* will be auto-adapted during learning process. We will make *F* lower and *CR* higher after each step, which makes offspring more similar to the current best solution (see Fig. 1).
- Regarding hyper-parameters, a new generation will include one individual with the current best configuration (from the old population or current offspring) and three individuals with random configurations. Thus, when *F* is close to 0, and *CR* is close to 1, we will focus offspring on the current best one (little influence from these three random options), but we will keep the availability of exploration by taking into account these random configurations in the current population.
- We consider our algorithm converges to a good solution when, given a number *k* of algorithm iterations, the differences between best solutions at each iteration are tiny.

3.2 MESSPT algorithm

Our MESSPT algorithm[1] consists of two operation phases, as it occurs in SSPT: the exploration and deployment phases. During exploration, the optimisation algorithm (DE) tunes the hyper-parameters trying to seek convergence to a local minimum; regarding the deployment phase, it involves only training the best model (found in the exploration phase) with new data until the detection of concept drift (Fig. 2).

Specifically, inside these two phases we can indicate some steps:

1. Given new data from any data stream, we follow the evaluation scheme of these data with no extra checks or operations until a grace period is reached. During this evaluation phase, two types of individuals are evaluated: those that belong to the current population; and those that belong to the offspring population generated

[1] Code available: https://gitfront.io/r/user-9515996/T1tZ5eH3W949/MESSPT/.

in the previous grace period. The following steps will clarify how we generate these population by transitioning from one generation to another (from one grace period to the next). We have to remark that we look for fast convergence and a micro-evolutionary scheme, so as we stated above, current and offspring populations have four individuals.

2. When we reach the grace period (we have evaluated enough data), a new population is created. Given the best individual at this step, we check whether the algorithm has converged. If not, a new offspring population is generated from this current population. In order to accelerate convergence but pay attention to the exploration of the search space, the creation of these two populations is based on mixing the best individual until the last grace period with random individuals from the search space. Before giving more details about how these populations are generated, it is essential to point out that each individual represents a wrapper, in which we include the ML algorithm, its configuration and its metric. Having indicated this, we can explain the creation of the new current and offspring populations in more detail.

   - To create the new current population, we keep the best individual with no modifications and generate new ones. To generate these new individuals, each wrapper consists of the ML model of the best individual, a new random configuration, and the metric of the best individual. Thus, we can keep the best model but try new configurations while saving the best one. The use of the best individual metric is just for consistency throughout the evaluations so that in each new evaluation period, all individuals start from the same metric. Algorithm 2 supports what we have explained in this paragraph.
   - We cross and mutate individuals from this new population to generate the offspring population. Mutation and cross operators have been presented in the subsection before (Sect. 3.1). To create a new child, we must clarify that its wrapper consists of the ML model of the best individual, the configuration obtained from mutation and cross operators, and the metric of the best individual. We have to clarify that F and CR values stated in subsection before (Sect. 3.1) are updated in each grace period (before creating the offspring) to control the faster or slower convergence of the algorithm. Algorithm 3 supports all the stated about how to create the offspring population.

3. We check if the algorithm has converged during each grace period before creating the offspring population. If yes, we will finish the exploration phase and move on to deployment. For this purpose, we keep the best individual (model+configuration) after convergence.
4. We continue on the deployment phase until concept drift is detected. If it occurs, we restart the algorithm. To this end, we create a new population by adding three

random individuals to this best model. Now, the wrapper of these three new individuals consists of the ML from scratch (not the fit model of the best individual as it was in previous steps), a random configuration, and the metric of the best individual to keep consistency during the evaluation. Moreover, *F* and *CR* values are restarted too. Thus, the deployment phase finishes and the exploration phase starts again. We now repeat the previous steps. Algorithm 1 summarises all the steps stated above.

Algorithm 1 Learning process of MESSPT

Input c_pop, $offsprings$, F, CR, $best$

procedure LEARN ONE(x, y) ▷ x is the input and y the label
  if `no convergence (exploration phase)` **then**
    for $ind \in c\_pop$ **do**
      $ind \leftarrow update\_metric(ind, x, y)$ ▷ testing
      $ind \leftarrow update\_model(ind, x, y)$ ▷ training
    end for
    for $ind \in offsprings$ **do**
      $ind \leftarrow update\_metric(ind, x, y)$
      $ind \leftarrow update\_model(ind, x, y)$
    end for
    `Update model and metric from offsprings`
    $best \leftarrow update\_best(c\_pop, offsprings)$
    if grace period **then**
      $F \leftarrow update\_F()$ ▷ Reducing F
      $CR \leftarrow update\_CR()$ ▷ Increasing CR
      $c\_pop \leftarrow generate\_next\_pop(best)$
      `check convergence`
      if no convergence **then**
        $offsprings \leftarrow generate\_offspring\_pop(c\_pop)$
      end if
    end if
  else ▷ (deployment phase)
    `check concept drift`
    if concept drift is detected **then**
      `Restart`
    else
      $best \leftarrow update\_metric(best, x, y)$
      $best \leftarrow update\_model(best, x, y)$
    end if
  end if
end procedure

Algorithm 2 Generate next population

Input c_pop, $best$

procedure GENERATE_NEXT_POP($best$) ▷ Generate next population
  $c\_pop \leftarrow \{\}$
  $model\_best, conf\_best, metric\_best \leftarrow extractFromInd(best)$
  $c\_pop.add(best)$ ▷ Add the best
  for $i \in 1, \ldots, 3$ **do** ▷ Add random
    $conf\_r \leftarrow randConf()$
    $new\_ind \leftarrow create\_ind(model\_best, conf\_r, metric\_best)$
    $c\_pop.add(new\_ind)$
  end for
end procedure

Algorithm 3 Generate offspring population

Input c_pop, $offsprings$, $best$

procedure GENERATE_OFFSPRING_POP(c_pop) ▷ Generate new offspring pop
  $model\_best, conf\_best, metric\_best \leftarrow extractFromInd(best)$
  for $ind \in c\_pop$ **do**
    $model\_ind, conf\_ind, \_ \leftarrow extractFromInd(ind)$
    $conf\_r1, conf\_r2 \leftarrow selectConfFromRand(c\_pop)$
    $conf\_child \leftarrow mutOperator(conf\_best, conf\_r1, conf\_r2)$
    $conf\_child \leftarrow cross(conf\_ind, conf\_child)$
    $new\_child \leftarrow create\_ind(model\_best, conf\_child, metric\_best)$
    $offsprings.add(new\_child)$
  end for
end procedure

Regarding the occurrence of concept drift, MESSPT, on the exploration phase, ignores the drift detector and continues training. It is because, during the exploration phase, there are lots of oscillations until convergence, provoking a false positive on the drift detector, which could generate difficulties in finding a good configuration. Related to the deployment phase, given new data instances, among all existing detectors indicated (see Sect. 2) we have considered the use of the ADWIN drift detector (Bifet and Gavaldà 2007) to analyse the model prediction and detect

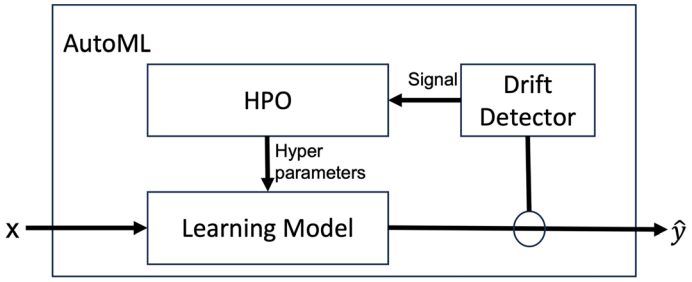


Fig. 3 MESSPT - Wrapper model

concept drifts. If concept drift occurs, MESSPT restarts the exploration phase. Figure 3 shows the MESSPT wrapper model.

As stated, using a drift detector during the exploration phase could lead to a non-optimised solution or difficulties in the convergence. However, we propose to smooth the effect of possible concept drifts during this phase by adding a random individual (from scratch) at each step in our population. It means, its wrapper consists of: the model from scratch; a random configuration; and the metric of the best individual to keep consistency. MESSPT will not consider this random individual for mutation and crossover. This individual will be restarted from scratch at each grace period (instead of keeping the current best model, as all other individuals in the population do). If this individual is the best at the end of the grace period, it could be due to two options: the first one is that our current population is full of weak solutions; the second one is due to a drastic change in the data distribution (in relation to the target), and thus, the best model is the one that has no previous knowledge. In either cases, if this random individual is the best after this period, a restart of our algorithm is necessary. We will observe the effects of this idea after premature concept drift (see Sect. 4.2.3) and how it has no drastic effects on non-premature concept drifts scenarios. Algorithm 4 clarifies how the modified learning process (Algorithm 1) works.

Algorithm 4 Learning process of MESSPT with little solution to premature concept drift

Input c_pop, $offsprings$, F, CR, $best$

procedure LEARN ONE(x, y) ▷ x is the input and y the label
  if `no convergence` **then**
    for $ind \in c\_pop$ **do** ▷ A random option is included in this pop
      $ind \leftarrow update\_metric(ind, x, y)$ ▷ testing
      $ind \leftarrow update\_model(ind, x, y)$ ▷ training
    end for
    `Update model and metric from offsprings`
    $best \leftarrow update\_best(c\_pop, offsprings)$ ▷ Here, c_pop has a random option included
    if grace period **then**
      `check if random option is the best`
      if `random is the best` **then**
        `Restart`
      else
        `Update` F `and` CR
        $c\_pop \leftarrow generate\_next\_pop(best)$ ▷ To do this, random option is not taken into account
        `check convergence`
        if no convergence **then**
          $offsprings \leftarrow cross\_pop(c\_pop)$ ▷ To do this, random option is not taken into account
        end if
      end if
    end if
  else
    `check concept drift`
    if concept drift is detected **then**
      `Restart`
    else
      $best \leftarrow update\_metric(best, x, y)$
      $best \leftarrow update\_model(best, x, y)$
    end if
  end if
end procedure

4 Experiments and discussion

All algorithms used are available/implemented in *river* (Montiel et al. 2021) framework for our experiments. The experimental setup aims to show the impact on the performance of our solution when compared with other proposals. Thus, addressing online hyper-parameter optimisation, random search (Bergstra and Bengio 2012) is adapted to online learning and added to our experiments; SSPT (Veloso et al. 2021) is included in our experiments too.

In order to get robust conclusions, we evaluated MESSPT on two prediction tasks, classification and regression. In Sect. 4.1 we will describe the learning algorithm, hyper-parameters to optimise and the metric used to evaluate for each learning task. In the appendix we describe the datasets included in our experiments (see Appendix.)

We follow the Prequential evaluation scheme for our experiments, which is typically used dealing with data streams and online learning. This scheme is based on: giving new data from a data stream, the are used first for testing the model (or models instead) and then for training it (or them). Because of not having static data, we need this scheme to measure the goodness of the models (instead of the classical train/test offline scheme). Moreover, it allows the restarting of the training process when the concept drift detector emits one alarm. We collect the results attending to each task and dataset. After discussing these results, we will compare how the different proposals react to concept drift in both cases before and after convergence (occurrence in exploration vs deployment phase).

4.1 Learning algorithms, hyper-parameters to optimise and metrics to evaluate

Our proposal is applicable to any supervised stream-based algorithm. Thus, for our experiments, we consider algorithms from different families. Firstly, for both machine learning tasks (classification and regression), we apply a pre-processing method (adaptive standard scaler) on the data streams. This pre-processing method relies on exponentially weighted moving averages and variance. After this pre-processing phase, we train on the one hand, Hoeffding Tree-based models [Hoeffding Tree Classifier (HTC) and Hoeffding Tree Regressor (HTR) (Hulten et al. 2001)] to predict the value for each new data instance. This algorithm requires the optimisation of three hyper-parameters: *delta*, which is the significance level to compute the Hoeffding bound; the *grace period*, which is related to the number of instances to observe before split attempts; and *tau*, which is a bound to control when to break ties and produce splits.

On the other hand, considering other families of algorithms, Logistic Regression (LR) (McCullagh and Nelder 1989) and AMRules (AMR) (Duarte et al. 2016) will also be used within the classification and regression task, respectively. For LR, we tune two hyper-parameters: the value of *L2* regularization; and the *learing rate* used for updating the intercept. Just in case, we have to mention that we use a generalization to multi-class of LR: Softmax Regression (SR), for these datasets in which

Table 1 Summary of learning algorithms, hyper-parameteres to optimise and metric used to evaluate for each task

Task	Algorithm	Parameter	Range	Metric
Classification	*HTC*	Delta	(0.00001, 0.0001)	Accuracy
		Grace period	(100, 500)	
		Tau	(0.01, 0.09)	
Classification	*LR/SR*	Learning rate	(0.005, 0.025)	Accuracy
		L2	(0.0, 0.01)	
Regression	*HTR*	Delta	(0.00001, 0.0001)	RMSE
		Grace period	(100, 500)	
		Tau	(0.01, 0.09)	
Regression	*AMR*	Delta	(0.00000001, 0.0001)	RMSE
		n_min	(100, 500)	
		Tau	(0.01, 0.09)	

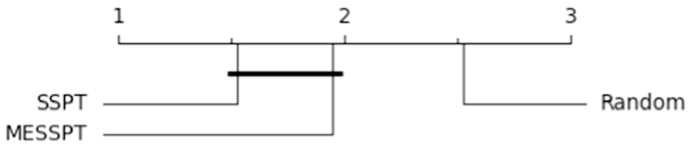


Fig. 4 Results of statistical tests for computational cost

classification is not binary. In these cases, we optimise both *L2* regularization and *learning rate* used in the sequential optimiser. In addition, for the AMR algorithm, we consider the optimisation of three hyper-parameters: *delta*, *tau* and *n_min*. The first two are similar to those stated before. The last one is related to the weight that a rule must observe between expansion attempts.

Table 1 presents the valid ranges for each hyper-parameter. Lastly, the metric to evaluate the performance is the *accuracy* for the classification task and *RMSE* for the regression task.

4.2 Results and discussion

In this subsection we present and discuss the results obtained in our experiments (see dataset description at Appendix). Firstly, we briefly focus on presenting the computational cost of our algorithm compared with the other two optimisation algorithms (SSPT and random). To this end, we carried out hypothesis testing through non-parametric statistical tests to determine significant differences in overall cost in these algorithms. Thus, Friedman's test will analyse general differences (with *p-value*= 0.05) between them, and the Nemenyi posthoc test will make pairwise comparisons (Garcia and Herrera 2008). Friedman's test considers differences with $p = 0.000042$. If we observe now the results of the Nemenyi's test (see Fig. 4):

Table 2 Table of results for the classification task

Classifier		HTC			LR/SR	
Alg/Model	Random	SSPT	MESSPT	Random	SSPT	MESSPT
Agra	64.74 ± 1.85	68.66 ± 1.16	**70.11 ± 0.34**	62.67 ± 0.18	62.60 ± 0.20	**62.91 ± 0.24**
Postures	56.18 ± 2.3	60.68 ± 3.54	**63.77 ± 0.56**	**69.45 ± 2.3**	69.18 ± 0.25	69.31 ± 0.16
Bank	88.27 ± 0.67	**89.18 ± 0.03**	**89.18 ± 0.27**	89.77 ± 0.06	89.41 ± 0.19	**89.82 ± 0.05**
Sine	63.41 ± 2.14	80.73 ± 0.85	**82.08 ± 0.08**	81.62 ± 0.13	82.23 ± 0.21	**82.39 ± 0.6**
Tweet500	81.33 ± 0.33	82.01 ± 0.31	**82.08 ± 0.33**	85.89 ± 0.04	**85.94 ± 0.27**	85.55 ± 0.2
Tweet1000	80.43 ± 0.38	**81.38 ± 0.59**	80.83 ± 0.44	**85.38 ± 0.03**	84.81 ± 0.36	85.1 ± 0.18
Cardio	59.22 ± 0.00	60.8 ± 3.51	**67.59 ± 0.00**	64.88 ± 0.10	60.54 ± 0.72	**64.93 ± 0.52**
Nomao	86.84 ± 0.41	87.9 ± 2.54	**92.88 ± 0.24**	91.25 ± 0.09	91.47 ± 0.74	**92.52 ± 0.17**
Enron	92.91 ± 1.66	91.74 ± 0.21	**94.39 ± 0.55**	94.29 ± 0.2	**94.38 ± 0.08**	93.99 ± 0.52
SEA	82.69 ± 1.51	88.31 ± 0.54	**88.72 ± 0.16**	88.91 ± 0.13	88.35 ± 0.31	**88.95 ± 0.08**

Bold values highlight the best result per dataset for each classifier

This test shows that our algorithm is worse ranked than SSPT, but no significant differences can be stated between them regarding computational cost. We have achieved a computational cost which has no significant differences with SSPT and has them with the random option. This fact is related to the design decisions, trying to obtain fast convergence and evaluating the same number of models at each step as SSPT during the exploration phase.

After briefly pointing out the above related to computational cost, we will focus on how our algorithm outperforms the other options regarding performance. Thus, a summary of results is presented in Tables 2 and 3.

4.2.1 Discussion related to performance in classification task

We will focus firstly on results obtained using the HRC classifier. It is possible to observe in Table 2 that MESSPT obtains the best results in 8 datasets (and shares the best position in the Bank dataset). Compared directly with SSPT, the performance of MESSPT is better or equal in most cases (only for Tweet1000 it is different). Synthetic datasets like Agra, Sine and SEA included on the benchmark ensure that we have datasets with concept drifts. MESSPT is the best option in all synthetic datasets (Sect. 4.2.3 presents further experiments).

In addition, we will analyse inferential information. Thus, once again, a Friedman's test is performed. It considers differences with $p = 0.00038$. After that, Nemenyi's posthoc tests will make pairwise comparisons between algorithms included in this subsection (see Fig. 5).

This test allows us to extract key conclusions. MESSPT (against SSPT) achieve significant differences with the random search option for this classification task. We can not consider that there are significant differences between MESSPT and SSPT. However, considering all stated above and Fig. 5, we can indicate (dealing with no significant differences) that MESSPT is the best-ranked algorithm related to performance. Thus, our algorithm is the most recommendable option in this case.

Table 3 Results for the regression task

Regressor		HTR			AMR	
Alg/model	Random	SSPT	MESSPT	Random	SSPT	MESSPT
2DPlanes	**1.97 ± 0.11**	2.12 ± 0.03	**1.97 ± 0.04**	**2.23 ± 0.06**	2.34 ± 0.07	2.25 ± 0.04
Ailerons	0.000384 ± 2.09E-6	0.0003798 ± 4.07E-6	**0.0003404 ± 2.58E-6**	0.000357 ± 8.01E-6	0.0003382 ± 6.24*E*-6	**0.0003342 ± 7.48E-6**
Friedman	3.58 ± 0.016	3.7 ± 0.02	3.52 ± 0.03	**3.67 ± 0.10**	3.80 ± 0.10	3.71 ± 0.04
Sgemm	411.48 ± 65.71	174.07 ± 10.36	**112.91 ± 9.36**	151.91 ± 20.26	135.91 ± 1.36	**130.77 ± 2.38**
Transcoding	28.67 ± 11.97	14.766 ± 1.18	**12.98 ± 0.51**	12.37 ± 2.62	11.27 ± 0.05	**11.23 ± 0.11**
Appliances	103.8 ± 1.77	98.22 ± 1.2	**94.72 ± 0.81**	98.48 ± 5.77	**94.31 ± 0.25**	94.36 ± 0.2
Bias	2.65 ± 0.02	**2.56 ± 0.04**	2.57 ± 0.05	2.85 ± 0.12	**2.57 ± 0.13**	2.59 ± 0.08
Metro	**1747.4 ± 7.63**	1755.09 ± 6.89	1750.43 ± 2.58	1877.72 ± 42.72	1894.66 ± 30.57	**1825.76 ± 2.35**
Tetuan	6596.76 ± 864.19	5732.59 ± 164.56	**5613.02 ± 230.99**	6351.27 ± 37.49	6332.48 ± 103.95	**6298.60 ± 25.31**
Garment	**0.1389 ± 0.0046**	0.1478 ± 0.0057	0.1441 ± 0.0046	0.1666 ± 0.0092	**0.1617 ± 0.0039**	0.1739 ± 0.006

Bold values highlight the best result per dataset for each regressor

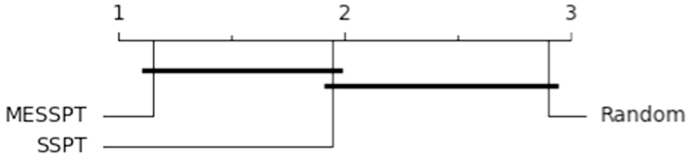


Fig. 5 Results of statistical tests for the classification task using HTC

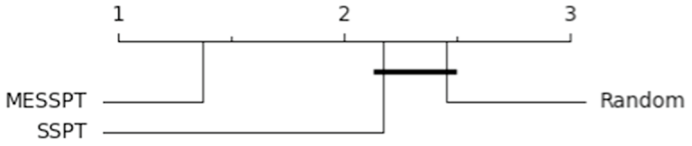


Fig. 6 Results of statistical tests for the classification task (global results)

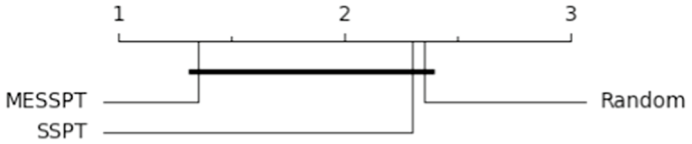


Fig. 7 Results of statistical tests for regression task using HTR

Now, we can consider the discussion for the LR (or SR) classifier (see again Table 2). MESSPT outperforms the other options in 6 of 10 cases. As it occurred previously, MESSPT is the best for Agra, SEA and Sine, reinforcing that it is pretty good for datasets in which the appearance of concept drift is sure. Once again, we analyse inferential information. After conducting Friedman's test, no significant differences are considered ($p = 0.202$). However, we can remark that our algorithm (MESSPT) is the best ranked with a rank of 1.6; following our proposal, Random obtains a value of 2.0; and finally, SSPT gets 2.4. Thus, our algorithm is once again the best ranked. Random obtains better results than SSPT, which further reinforces the improvement of our algorithm over SSPT.

Finally, we perform a Friedman's test considering all the results obtained with both algorithms to obtain general conclusions inside the classification task. It considers significant differences with $p = 0.002$. We carry out now the Nemenyi's posthoc test, and the results are shown in Fig. 6

If we consider all results together, MESSPT obtains significant differences against SSPT and Random for the classification task.

4.2.2 Discussion related to performance in regression task

Regarding regression task using HTR (see Table 3), MESSPT is the best option in 6 of 10 cases (and sharing best in 2Dplanes with random). As we did for classification, we perform a Friedman's test. Friedman's test indicates that there are differences with $p = 0.039$. After that, the Nemenyi posthoc test will make pairwise

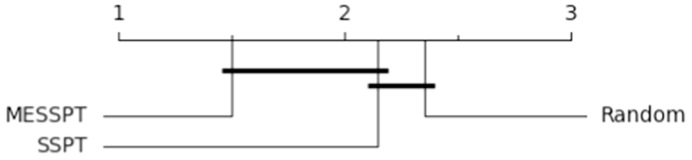


Fig. 8 Results of statistical tests for regression task

comparisons (Fig. 7). Attending to this figure, unless no significant differences can be stated between MESSPT and SSPT, MESSPT is best ranked than SSPT.

Related to the regression task using AMRules (see Table 3 again), MESSPT is the best option in 5 of 10 cases. Comparing by pairs, MESSPT is better than Random in 7 of 10 cases and better than SSPT in 7 of 10 cases.

Once again, we perform a Friedman's test to compare these algorithms. As a result of this test, no differences are considered ($p = 0.202$). However, it ranks the algorithms as follows: Random obtains a rank of 2.4; SSPT obtains a value of 2.0; MESSPT gets 1.6. Thus, MESSPT continues to be the best-ranked for AMR.

Finally, to extract a general conclusion for the regression task, we will compare the results of Random, SSPT and MESSPT generally for both HTR and AMR. In this scope, we carry out again a Friedman's test. It considers differences between them with $p = 0.019$. Thus, we analyse these differences with the Nemenyi test (see Fig. 8)

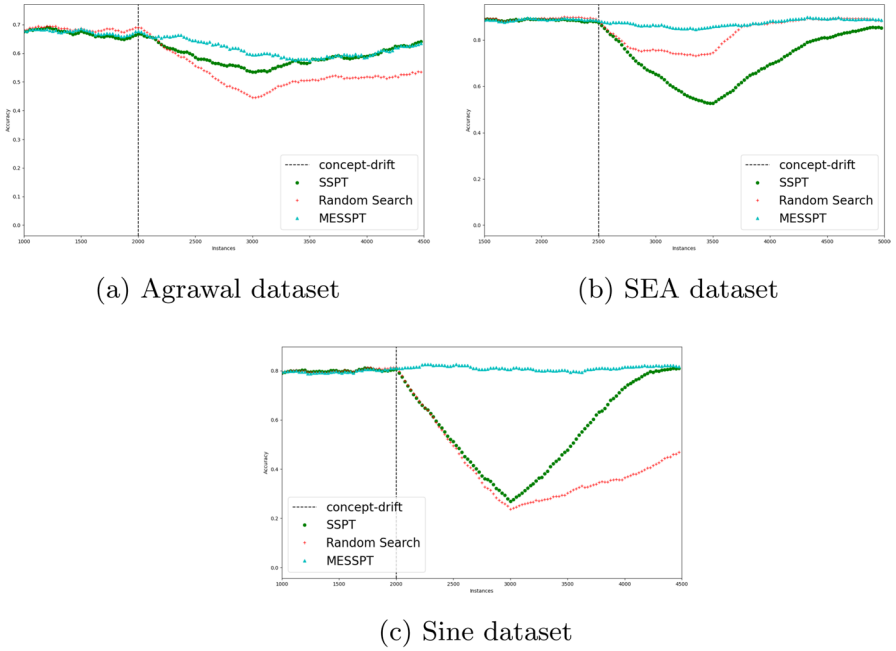
Related to this test, we could indicate that, for regression tasks, MESSPT achieves better performance than Random and better or equal to SSPT. Thus, we can reinforce that MESSPT's behaviour is better than Random's and SSPT algorithms.

To sum up, our algorithm is always the best choice regarding performance among all the self-tuning algorithms compared. Moreover, as stated above, its computational cost is better than the random's cost and similar to the SSPT one.

4.2.3 How does our algorithm react to concept drift?

An interesting point dealing with online learning is related to a good reaction of the algorithms when concept drift appears. To make this analysis, we will show a graph performance obtained immediately after concept drift. To clarify the plot, performance at each point is the mean performance of a window, equivalent to 2 grace periods. In all cases, we will show this window with 1000 points (grace period defined to 500 data instances). Finally, before showing each graphic, we will consider datasets in which we can control the concept drift appearance. For this purpose, we force drifting on all data. We focus firstly on the classification task. To this end, we consider Agrawal, SEA and Sine datasets. In all these datasets, we deal with the occurrence of abrupt concept drift, which allows to compare the behaviour of these three algorithms to this type of concept drift.

We will analyse two different situations: the behaviour of algorithms when this concept drift occurs in earlier stages (SSPT and MESSPT have not converged at this stage); and when this concept drift occurs in later stages (both have converged).



(a) Agrawal dataset (b) SEA dataset

(c) Sine dataset

Fig. 9 Behavior when concept drift occurs early

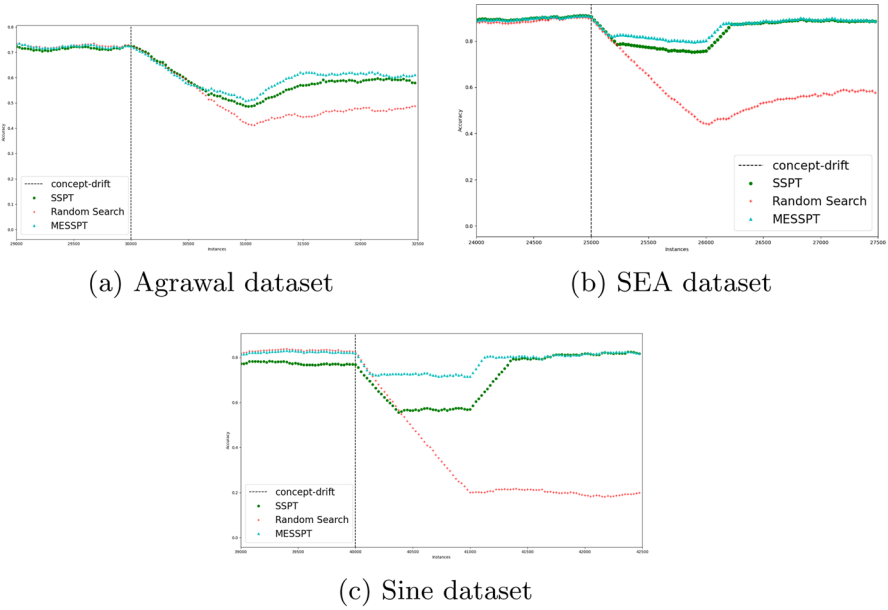
Regarding behaviour when concept drift occurs earlier, Fig. 9 shows performance since the concept drift occurrence until the equivalent of 5 grace periods.

All these figures are related to the classification task by using the HTC classifier. In all of them, we can observe how MESSPT is the one which better reacts to this abrupt concept drift. Considering that this case is an early concept drift and the algorithm has not converged, the main reason for this advantage is the proper exploring of the search space and the restart step proposed in our algorithm.

Now, we will focus on convergence in a later stage (SSPT and MESSPT have already converged). Figure 10 shows the performance close to the abrupt concept drift appearance at this stage.

We can observe how dealing with abrupt concept drift at this stage, SSPT and MESSPT are the most prominent options. They both deal properly with concept drift appearance. MESSPT showed a good performance in earlier stages, and now, it presents good behaviour again. Compared to SSPT, MESSPT reacts better in this late stage (the MESSPT accuracy curve is above SSPT one). Thus, we have proposed a more robust algorithm for the classification task to deal with abrupt concept drift appearance.

Regarding regression (by using the HTR regressor), we have included Friedman as the only dataset in which we can control the appearance of concept drift. Once again, we will analyse behaviour in both earlier and later stages. We have two critical points for the Friedman dataset in which concept drift appears. We will show the reaction to concept drift when both appear before convergence (see Fig. 11). Against



(a) Agrawal dataset (b) SEA dataset

(c) Sine dataset

Fig. 10 Behavior when concept drift occurs later

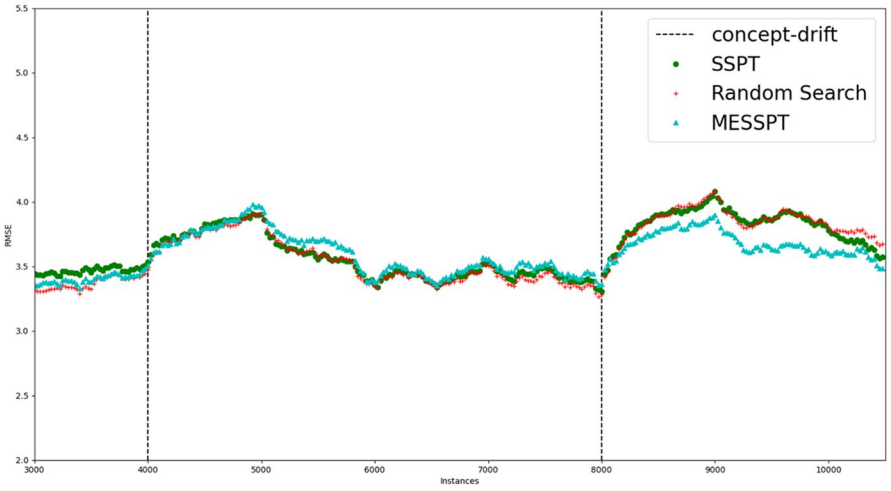
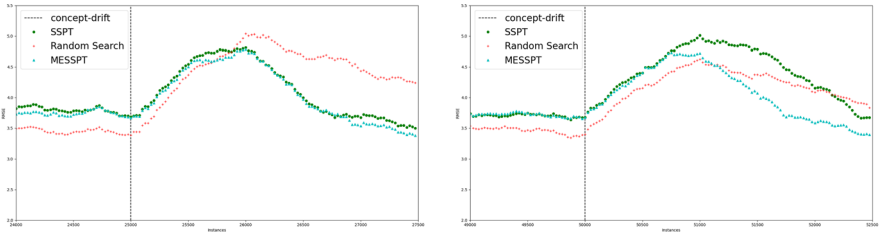


Fig. 11 Behavior when both concept drifts occurs early in Friedman

the abrupt concept drift stated before, we observe now how the different proposals react to another type of concept drift: the recurrent concept drift:

Related to recurrent concept drift appearance, after the first critical point, our proposal does not perform better than others (in fact, taking into account small



(a) First concept drift occurrence (b) Second concept drift occurrence

Fig. 12 Behavior when concept drift occurs later in Friedman dataset

differences, we can consider it the worst). However, after the second, ours is the algorithm which better reacts to concept drift.

Regarding concept drift occurrence after convergence of SSPT and MESSPT (late stage), we show performance in Fig. 12.

In both cases, MESSPT shows better behaviour. Thus, MESSPT reacts better for this dataset than all other proposals for late recurrent concept drift. MESSPT reacts well to recurrent concept drift after MESSPT convergence.

We can conclude from this discussion related to the regression task (and recurrent concept drift) that, generally, MESSPT reacts well to recurrent concept drift, being this reaction remarkable when recurrent concept drift appears after convergence.

To sum up this block, we have proved the better reaction of our algorithm to two different types of concept drift (in both cases of study, early and late concept drifts) than other approaches included in our experimental section (random and SSPT), which is crucial in online learning.

5 Conclusions and future research directions

We have presented MESSPT, an online solution based on DE for the automated optimisation of hyper-parameters. We have properly adapted DE to online learning, proposing a micro-evolutionary option which allows feasible computational cost during this online learning process. Strong conclusions related to the Prequential evaluation are obtained from our experimental section, pointing out our proposal as a better option than others such as SSPT or random search: is the best option regarding performance, and no significant differences exist with SSPT related to computational cost. Moreover, firm conclusions are obtained regarding how our algorithm reacts to concept drift, proving (empirically) that it offers a good reaction to its occurrence. Thus, we have proposed a robust online solution for self-hyper-parameter tuning for data streams.

Our evolutionary algorithm counts with the strength of allowing better search space exploration than other algorithms (on account of how evolutionary algorithms work). However, there is a limitation since our proposal is a micro-evolutionary algorithm. It has a faster convergence, but the exploration is lower than in non-micro-evolutionary approaches. Moreover, our algorithm, by definition, may have

difficulties treating categorical hyperparameters. We can adapt it to do it properly, but other evolutionary algorithms could deal better with these parameters. Taking both limitations into account, the immediate future research direction (considering as the basis that we have proved the good behaviour of an evolutionary algorithm to this end) is the adaptation of other evolutionary algorithms that solve these problems (as non-micro genetic algorithms) to allow fast convergence and not too high computational cost as we have achieved. There are not too many proposals for this hyper-parameter self-tuning for data streams task. Therefore, adapting another optimisation algorithm for this problem may also be interesting.

Appendix: Datasets

For the classification task, we used the following benchmark datasets:

- **Agra** (Agrawal et al. 1993): Offers a stream generator in which its possible to change the concept drift by changing a classification function with ten possible values. It contains nine features, and 60000 inputs are generated, with a concept drift in position 30000 and a binary target variable.[2]
- **Bank** (Moro et al. 2014): It includes data from marketing campaigns of a Portuguese banking institution. They try to discover if a term deposit will be subscribed to by the clients or not (binary target variable). It contains 17 attributes and 45211 instances.[3]
- **Sine** (Gama et al. 2004): It is a sine generator. It generates four relevant features (between 0-1), with two relevant for classification and two optionally added as noise. It offers four classification functions to label the output. Changing these classification functions allows the creation of abrupt concept drift. 50000 instances are generated, with abrupt concept drift positioned since position 25000. Again, it is a dataset with two labels (binary target variable).
- **Tweet 500** (Bahri et al. 2020a): It consists of 100000 tweets of 500 attributes and two possible classes.[4]
- **Tweet 1000** (Bahri et al. 2020a): It consists of 100000 tweets of 1000 attributes and two possible classes.[5]
- **Postures** (Gardner et al. 2014): Decision-related to 5 hand postures (multiclass target variable). To this end, a motion capture camera system recollects information from each posture.[6] A '0' value replaces missing values.

[2] https://riverml.xyz/0.14.0/api/datasets/synth/Agrawal/.
[3] http://archive.ics.uci.edu/ml/datasets/Bank+Marketing.
[4] https://github.com/marouabahri/CS-ARF.
[5] https://github.com/marouabahri/CS-ARF.
[6] https://archive.ics.uci.edu/ml/datasets/MoCap+Hand+Postures.

- **Nomao** (Candillier and Lemaire 2012): It collects data about places (name, phone, location) from many sources. The classification consists of predicting if data belong to 2 spots.[7]
- **Enron** (Bahri et al. 2020b): A cleaned version of a large set of emails aimed to detect fraud. This version has 1702 instances and 1000 attributes.[8]
- **SEA** (Street and Kim 2001): Each instance consists of two integer attributes randomly chosen between 0 and 10. The result of adding them will suppose each instance to belong to one or another class. It allows abrupt concept drift by changing the conditions related to the threshold that separates both classes. We include 50000 instances and the appearance of concept drift after 25000 of them.[9]
- **Cardio** (Dua and Graff 2017): Fetal heart rate (FHR) and uterine contraction (UC) features measured on cardiotocograph classified by expert obstetricians. We try to predict the FHR pattern class code (1 to 10). It counts with 2126 instances and 23 real attributes.[10]

For the regression task, we adopted the following benchmark datasets:

- **2DPlanes** (Breiman et al. 1984): It is a synthetic dataset that contains 10 attributes. For our experiments, we generated 50000 synthetic data instances.
- **Ailerons**[11]: It contains 13750 instances and 41 attributes.
- **Friedman** (Ikonomovska et al. 2011): It is a synthetic dataset composed of 10 features between 0 and 1 uniformly sampled. We use a river option to trigger two global Recurring Abrupt drift points. We generate 70000 instances and choose 25000 and 50000 as these drift positions.[12]
- **Sgemm** (Ballester-Ripoll et al. 2019): It measures the running time of a matrix product, measuring the future output to predict new products. It contains 14 attributes and 241600 instances.[13]
- **Transcoding** (Deneke et al. 2014): Transcoding time prediction (related to online video). It contains 20 attributes regarding online video characteristics to this end.[14]
- **Appliances** (Candanedo et al. 2017): It includes measurements from a low-energy building obtained with 10 min periods and related to 4.5 months. Appliances energy prediction is the target. It includes 19735 and 29 attributes.[15]

[7] https://github.com/marouabahri/CS-ARF/tree/master/datasets.
[8] https://github.com/marouabahri/CS-kNN/tree/master/datasets.
[9] https://riverml.xyz/0.14.0/api/datasets/synth/SEA/.
[10] https://archive.ics.uci.edu/ml/datasets/Cardiotocography.
[11] https://github.com/renatopp/arff-datasets/blob/master/regression/2dplanes.arff.
[12] https://riverml.xyz/0.14.0/api/datasets/synth/FriedmanDrift/.
[13] http://archive.ics.uci.edu/ml/datasets/SGEMM+GPU+kernel+performance.
[14] https://archive.ics.uci.edu/ml/datasets/Online+Video+Characteristics+and+Transcoding+Time+Dataset.
[15] https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction.

- **Bias** (Cho et al. 2020): It contains meteorological forecast data and auxiliary geographical variables to make a temperature prediction. It presents 7750 instances with 25 attributes collected in Seoul, South Korea, in the summer.[16]
- **Metro**[17]: It measures Metro Interstate Traffic Volume. It contains variables that impact the volume, like hourly weather features. It has 48204 and 9 numerical attributes.
- **Tetuan** (Salam and El Hibaoui 2018): It includes nine numerical attributes and 52,417 instances in order to predict the power consumption of Tetouan city.[18]
- **Garment** (Rahim et al. 2021): It is focused on productivity prediction of garment employees. It includes 1197 and 15 numerical attributes.[19]

Acknowledgements This work has been funded by the Ministry of Science and Innovation and the European Regional Development Fund, project PID2020-115832GB-I00, and by the Ministry of Universities, predoctoral grant FPU18/06307.This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project UIDB/50014/2020.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

Agrawal R, Imielinski T, Swami A (1993) Database mining: a performance perspective. IEEE Trans Knowl Data Eng 5(6):914–925

Bäck T (1996) Evolutionary algorithms in theory and practice - evolution strategies, evolutionary programming, genetic algorithms. Oxford University Press, Oxford

Baena-Garcıa M, del Campo-Ávila J, Fidalgo R, et al (2006) Early drift detection method. In: Fourth international workshop on knowledge discovery from data streams, Citeseer, pp 77–86

Bahri M, Gomes HM, Bifet A, et al (2020a) Cs-arf: compressed adaptive random forests for evolving data stream classification. In: 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, pp 1–8

Bahri M, Maniu S, Bifet A, et al (2020b) Compressed k-nearest neighbors ensembles for evolving data streams. In: ECAI 2020-24th European conference on artificial intelligence

Bahri M, Bifet A, Gama J et al (2021) Data stream analysis: foundations, major tasks and tools. Wiley Interdiscipl Rev: Data Min Knowl Discov 11(3):e1405

Bakhashwain N, Sagheer A (2020) Online tuning of hyperparameters in deep LSTM for time series applications. Int J Intell Eng Syst 14(1):212–220

Ballester-Ripoll R, Paredes EG, Pajarola R (2019) Sobol tensor trains for global sensitivity analysis. Reliab Eng Syst Safety 183:311–322

Barros RC, Basgalupp MP, De Carvalho ACPLF et al (2012) A survey of evolutionary algorithms for decision-tree induction. IEEE Trans Syst Man Cybern Part C Appl Rev 42(3):291–312

Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. J Mach Learn Res 13(2):281–305

[16] https://archive.ics.uci.edu/ml/datasets/Bias+correction+of+numerical+prediction+model+temperature+forecast.

[17] https://archive.ics.uci.edu/ml/datasets/Metro+Interstate+Traffic+Volume.

[18] https://archive.ics.uci.edu/ml/datasets/Power+consumption+of+Tetouan+city.

[19] https://archive.ics.uci.edu/ml/datasets/Productivity+Prediction+of+Garment+Employees.

Bifet A, Gavalda R (2007) Learning from time-changing data with adaptive windowing. In: Proceedings of the 2007 SIAM international conference on data mining, SIAM, pp 443–448
Bifet A, Gavaldà R (2007) Learning from time-changing data with adaptive windowing. In: Proceedings of the Seventh SIAM International Conference on Data Mining, April 26-28, 2007, Minneapolis, Minnesota, USA, pp 443–448
Breiman L, Friedman JH, Olshen RA et al (1984) Classification and regression trees. Wadsworth, Belmont
Candanedo LM, Feldheim V, Deramaix D (2017) Data driven prediction models of energy use of appliances in a low-energy house. Energy Build 140:81–97
Candillier L, Lemaire V (2012) Design and analysis of the nomao challenge - active learning in the real-world. In: Proceedings of the ALRA : active Learning in Real-world Applications, Workshop ECML-PKDD 2012, Friday, September 28, 2012, Bristol, UK
Celik B, Vanschoren J (2021) Adaptation strategies for automated machine learning on evolving data. IEEE Trans Pattern Anal Mach Intell 43(9):3067–3078
Cho D, Yoo C, Im J et al (2020) Comparative assessment of various machine learning-based bias correction methods for numerical weather prediction model forecasts of extreme air temperatures in urban areas. Earth Space Sci 7(4):2019000740
Coello CAC, Pulido GT (2001) A micro-genetic algorithm for multiobjective optimization. In: Evolutionary multi-criterion optimization, first international conference, EMO 2001, Zurich, Switzerland, March 7-9, 2001, Proceedings, Lecture Notes in Computer Science, vol 1993. Springer, pp 126–140
Das S, Suganthan PN (2011) Differential evolution: a survey of the state-of-the-art. IEEE Trans Evol Comput 15(1):4–31
Deneke T, Haile H, Lafond S, et al (2014) Video transcoding time prediction for proactive load balancing. In: Multimedia and expo (ICME), 2014 IEEE International Conference on, pp 1–6
Dua D, Graff C (2017) UCI machine learning repository
Duarte J, Gama J, Bifet A (2016) Adaptive model rules from high-speed data streams. ACM Trans Knowl Discov Data 10(3):30:1-30:22
Frias-Blanco I, del Campo-Ávila J, Ramos-Jimenez G et al (2014) Online and non-parametric drift detection methods based on hoeffding's bounds. IEEE Trans Knowl Data Eng 27(3):810–823
Galletly J (1998) Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Kybernetes 27(8):979–980
Gama J, Medas P, Castillo G, et al (2004) Learning with drift detection. In: Brazilian symposium on artificial intelligence, Springer, pp 286–295
Gama J, Žliobaitė I, Bifet A et al (2014) A survey on concept drift adaptation. ACM Comput Surv (CSUR) 46(4):1–37
Garcia S, Herrera F (2008) An extension on" statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. J Mach Learn Res 9(12):2677–2694
Gardner A, Duncan CA, Kanno J, et al (2014) 3d hand posture recognition from small unlabeled point sets. In: 2014 IEEE international conference on systems, man, and cybernetics (SMC), IEEE, pp 164–169
Guliashki V, Toshev H, Korsemov C (2009) Survey of evolutionary algorithms used in multiobjective optimization. Probl Eng Cybernet Robot 60(1):42–54
Hauschild M, Pelikan M (2011) An introduction and survey of estimation of distribution algorithms. Swarm Evol Comput 1(3):111–128
Hruschka ER, Campello RJ, Freitas AA et al (2009) A survey of evolutionary algorithms for clustering. IEEE Trans Syst Man Cybern Part C Appl Rev 39(2):133–155
Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, pp 97–106
Ikonomovska E, Gama J, Džeroski S (2011) Learning model trees from evolving data streams. Data Min Knowl Disc 23(1):128–168
Imbrea A (2021) Automated machine learning techniques for data streams. CoRR arXiv:abs/2106.07317
Koza JR (1995) Survey of genetic algorithms and genetic programming. pp. 589–594
Kulbach C, Montiel J, Bahri M, et al (2022) Evolution-based online automated machine learning. Lecture notes in computer science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 13280 LNAI:472 – 484
Lacombe T, Koh YS, Dobbie G, et al (2021) A meta-learning approach for automated hyperparameter tuning in evolving data streams. In: International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China, July 18–22, 2021. IEEE, pp 1–8

Lerman P (1980) Fitting segmented regression models by grid search. J Roy Stat Soc: Ser C (Appl Stat) 29(1):77–84
Lin C, Guo M, Li C, et al (2019) Online hyper-parameter learning for auto-augmentation strategy. In: 2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27—November 2, 2019. IEEE, pp 6578–6587
Lobo JL, Ser JD, Osaba E (2021) Lightweight alternatives for hyper-parameter tuning in drifting data streams. In: 2021 International Conference on Data Mining, ICDM 2021 - Workshops, Auckland, New Zealand, December 7–10, 2021. IEEE, pp 304–311
McCullagh P, Nelder JA (1989) Generalized linear models. Springer, Berlin
Mockus J, Tiesis V, Zilinskas A (1978) The application of Bayesian methods for seeking the extremum. Towards Global Optimiz 2(117–129):2
Montiel J, Halford M, Mastelini SM et al (2021) River: machine learning for streaming data in python. J Mach Learn Res 22:110:1-110:8
Moro S, Cortez P, Rita P (2014) A data-driven approach to predict the success of bank telemarketing. Decis Support Syst 62:22–31
Nelder JA, Mead R (1965) A simplex method for function minimization. Comput J 7(4):308–313
Rahim MS, Imran AA, Ahmed T (2021) Mining the productivity data of garment industry. Int J Bus Intell Data Min 1(1):1
Salam A, El Hibaoui A (2018) Comparison of machine learning algorithms for the power consumption prediction:-case study of Tetouan city. In: 2018 6th International renewable and sustainable energy conference (IRSEC), IEEE, pp 1–5
Sebastião R, Fernandes JM (2017) Supporting the page-hinkley test with empirical mode decomposition for change detection. In: Foundations of Intelligent Systems: 23rd International Symposium, ISMIS 2017, Warsaw, Poland, June 26-29, 2017, Proceedings 23, Springer, pp 492–498
Street WN, Kim Y (2001) A streaming ensemble algorithm (sea) for large-scale classification. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, pp 377–382
Sun Y, Pfahringer B, Gomes HM et al (2022) Soknl: a novel way of integrating k-nearest neighbours with adaptive random forest regression for data streams. Data Min Knowl Disc 36(5):2006–2032
Veloso B, Gama J, Malheiro B (2018) Self hyper-parameter tuning for data streams. Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics) 11198 LNAI:241—255
Veloso B, Gama J, Malheiro B et al (2021) Hyperparameter self-tuning for data streams. Inform Fusion 76:75–86
Zhan H, Gomes G, Li XS, et al (2018) Efficient online hyperparameter optimization for kernel ridge regression with applications to traffic time series prediction. CoRR arXiv:abs/1811.00620
Zhan ZH, Shi L, Tan KC et al (2022) A survey on evolutionary computation for complex continuous optimization. Artif Intell Rev 55(1):59–110