# Hyperparameter Self-Tuning for Data Streams

Bruno Veloso[b,a,*], João Gama[c,a], Benedita Malheiro[d,a], João Vinagre[a,e]

[a] *INESC TEC, Porto, Portugal*
[b] *Universidade Portucalense, Porto, Portugal*
[c] *FEP - University of Porto, Porto, Portugal*
[d] *ISEP - Polytechnic Institute of Porto, Porto, Portugal*
[e] *FCUP - University of Porto, Porto, Portugal*

**Abstract**

The number of Internet of Things devices generating data streams is expected to grow exponentially with the support of emergent technologies such as 5G networks. The online processing of these data streams therefore requires the design and development of suitable machine learning algorithms, able to learn online, as data is generated. Like their batch-learning counterparts, stream-based learning algorithms require careful hyperparameter settings. However, this problem is exacerbated in online learning settings, especially with the occurrence of concept drifts, that frequently require the reconfiguration of hyperparameters. In this article, we present `SSPT`, an extension of the Self Parameter Tuning (`SPT`) optimisation algorithm for data streams. We apply the Nelder-Mead algorithm to dynamically-sized samples, converging to optimal settings in a single-pass over data, while using a relatively small number of hyperparameter configurations. In addition, our proposal automatically readjusts hyperparameters when concept drift occurs. To assess the effectiveness of `SSPT`, the algorithm is evaluated with three different machine learning problems: recommendation, regression, and classification. Experiments with well-known data sets show that the proposed algorithm can outperform previous hyperparameter tuning efforts by human experts. Results show that `SSPT` converges significantly faster and

*Corresponding author
*Email addresses:* `bruno.m.veloso@inesctec.pt` (Bruno Veloso), `jgama@fep.up.pt` (João Gama), `mbm@isep.ipp.pt` (Benedita Malheiro), `jnsilva@inesctec.pt` (João Vinagre)

presents at least similar accuracy when compared with the previous double-pass version of the SPT algorithm.

*Keywords:* Data Streams, Optimisation, hyperparameters

*2010 MSC:* 00-01, 99-00

## 1. Introduction

The ongoing evolution of mobile communication network technology will support the dissemination of Internet of Things devices, that generate data streams at potentially high rates, across multiple application domains, such as automation, manufacturing, healthcare or surveillance. With this continuous growth of data generation, practitioners need to apply machine learning algorithms to extract meaningful knowledge. However, the application of machine learning algorithms, in real-time, to data streams is far from trivial. Maximising the algorithm performance still mostly relies on the expertise and time-consuming efforts of data scientists, typically involving several trial-and-error rounds. The progressive automation of machine learning (AutoML) promises to alleviate this burden with automation. One of the most important – and time-consuming – steps is the tuning of hyperparameters.

Hyperparameter optimisation is studied since the 80s with the help of optimisation algorithms based on grid-search [1], random-search [2] and gradient descent [3]. Optimisation algorithms can be gradient-based, *e.g.*, gradient descent, or gradient-free, *e.g.*, Nelder-Mead [4]. The main limitation of approaches available in the literature is that they require offline processing, with train and validation stages, making them applicable to batch learning problems, but not in streaming scenarios. Moreover, in online supervised learning scenarios, the relation between the input data and the target variable frequently changes over time, giving rise to the so-called concept drift [5]. The main issue with concept drifts is that they eventually make existing hyperparameter configurations obsolete [6]. We argue that the optimisation of hyperparameters in algorithms that learn from data streams requires a specialised AutoML approach.

This is the case of the hyperparameter Self-Tuning algorithm for Data Streams (SPT) we proposed in [7, 8]. SPT performs a double-pass direct-search to find optimal solutions in a configuration space. Specifically, it applies the Nelder-Mead algorithm to dynamically-sized windows over a data stream, continuously searching for optimal hyperparameter configurations.

The main contribution of this work is the refinement of the SPT algorithm. The new version of SPT is a single-pass algorithm (SSPT), capable of optimising multiple continuous or discrete hyperparameters. This proposal is not only applicable to recommendation, regression, and classification problems, but also is, to the best of our knowledge, the only one which effectively works with data streams and reacts to concept drifts in near real-time.

The novel aspects of this work are: (*i*) the systematic literature review on AutoML; (*ii*) the single-pass data exploration mode; and (*iii*) an extended evaluation using three different machine learning tasks (recommendation, regression and classification).

The document is organised in five sections. Section 2 describes a systematic literature review on automatic machine learning. Section 3 presents the SSPT version for the identified problem. Section 4 details the experiments and discusses the results. Finally, Section 5 draws the conclusions and suggests future developments.

## 2. Related Work

While Auto Machine Learning (AutoML) has been explored for decades for algorithm selection [9], contributions in the literature related to hyperparameter tuning are much more recent. Two surveys on the topic were published regarding solutions and open challenges [10, 11]. Our literature search was focused on offline hyperparameter optimisation, Nelder-Mead-based optimisation, online hyperparameter tuning, and AutoML tools which include hyperparameter optimisation.

In terms of hyperparameter optimisation algorithms, we identified the con-

tributions of [12–17]. Kohavi and John [12] describe a method to automatically select a hyperparameter. This method relies on the minimisation of the estimated error and applies a grid search algorithm to find local minima. The problem of this solution is that it has exponential complexity, *i.e.*, with the increase of the number of parameters or the resolution of the search, the number of required function assessments grows exponentially. Shaker and Lughofer [13] rely on a concept drift detector based on the Page-Hinkley test to adjust the hyperparameters. This detector measures the magnitude and duration of the drift. With this information, the algorithm adjusts the forgetting parameters to counterbalance the change of direction on the error metric. Nonetheless, this approach requires an initial optimisation step. Finn *et al.* [14] propose a fine-tuning mechanism for the gradient descent algorithm which is applied periodically to fixed-size data samples. Li *et al.* [15] extend the Successive Halving multi-armed bandit algorithm [18] to continuously reduce the number of configuration candidates, by focusing on the most promising ones. The extended algorithm – Hyperband – regularly adjusts the trade-off between the available computational budget and the number of candidates. Nichol *et al.* [16] propose a scalable meta-learning algorithm which learns a parameter initialisation for future tasks. The proposed algorithm tunes the parameter by repeatedly using Stochastic Gradient Descent (SGD) on the training task. The above proposals are computationally expensive and/or require manual parameter tuning to some extent.

There are several works that rely on the Nelder-Mead algorithm for optimisation [19–22]. Koenigstein *et al.* [19] adopt the Nelder-Mead direct search to optimise more than twenty hyperparameters. Kar *et al.* [20] apply an exponentially decaying centrifugal force to all vertices of the Nelder-Mead algorithm to obtain better objective values. However, this process requires more iterations to converge to a local minimum. Fernandes *et al.* [21] propose a method to estimate the parameters and the initialisation of a CANDECOMP / PARAFAC tensor decomposition for link prediction. The authors adopt Nelder-Mead to identify the optimal hyperparameter initialisation. Pfaffe *et al.* [22] present

an online auto tuning algorithm for string matching algorithms. It uses the e-greedy policy, a well-known reinforcement learning technique, to select the algorithm to be used in each iteration and adopts Nelder-Mead to tune the parameters until a pre-defined user criterion is met, *e.g.*, number of tuning iterations. These optimisation solutions illustrate the applicability of Nelder-Mead to different Machine Learning Tasks. However, they all adopt batch processing, which is not applicable to the stream-based scenario.

In terms of online hyperparameter tuning algorithms, there are few works [17, 23–26] that use support vector machines together with batch processing, gradient solutions combined with brute force or genetic algorithms to optimise hyperparameters. Lawal and Abdulkarim [23] introduce an incremental learning-model selection method for data stream batches. It relies on incremental $k$-fold cross-validation to perform the incremental tuning of the support vector machine hyperparameters. Zhan *et al.* [24] adopt an online projected hyper-gradient descent algorithm. The algorithm computes the hyperparameter gradients on the fly whenever a new datum is observed and, then, updates smoothly the hyperparameters with the average of the past and current hypergradients. Minku [25] proposed an online hyperparameter tuning method that maintains a number of model instances created from different subsets. The method applies computational brute force to find the model instance with the smallest validation error. Zhou *et al.* [26] use a set of parallel configurations to adjust the hyperparameters. The authors use genetic algorithms and apply a set of four operators (elitism, preservation, recombination and mutation) to the available configurations to adapt the hyperparameters. These operators are applied periodically ($n$ steps). Lin *et al.* [17] propose a forward hyper-gradient optimisation method that uses the gradient of the response of a function to adjust the hyperparameters [27]. This optimisation method trains the hyperparameters and the network weights simultaneously.

Regarding existing AutoML tools, we focus on those that support multiple machine learning tasks, automatically select algorithms and optimise hyperparameters. According to Elshawi *et al.* [11], AutoML tools can be grouped by

the type of: (*i*) underlying machine learning library, *e.g.*, scikit-learn [28–31] or Weka [32, 33]; (*ii*) optimisation technique, *e.g.*, Bayesian optimisation [28–30], genetic algorithms [31] and neural networks [33]; and (*iii*) machine learning task, *e.g.*, classification and regression [28–33]. Hyperopt is a python library developed for hyperparameter optimisation [28]. Internally, it adopts a Bayesian optimiser and uses cross-validation to guide the search. Although it can be used together with scikit-learn to optimise models, it does not support data streams. Thornton *et al.* [29] present a framework for classification problems called Auto-Weka, which performs hyperparameter tuning using a Bayesian optimiser together with cross-validation. More recently, AutoWeka 2.0 includes both regression and classification algorithms [30]. Feurer and Hutter [32] propose an extension to Auto-Weka called Auto-SkLearn. When compared with Auto-Weka, Auto-SkLearn takes into account the performance of similar data sets, making it more data-efficient. The H2O AutoML module is part of the H2O open-source distributed machine learning platform. This module allows the automatic training and tuning of classification and regression models within a user-specified time-limit [34]. Specifically, it performs a fast random search in combination with automated stacked ensembles, using random forests, gradient boosting machines, deep neural nets, or generalised linear models [35]. Olson and Moore [31] suggest a tree-based pipeline optimisation tool called TPOT. This system combines genetic programming with Pareto optimisation to tune a classification algorithm on top of scikit-learn [36]. Mendoza *et al.* [33] present another Auto-Weka extension, called Auto-Net which uses neural networks to explore optimal hyperparameter solutions. Table 1 summarises the comparison of the referred AutoML tools based on processing mode (batch or stream processing), machine learning (ML) task (recommendation, regression or classification) and ability to use concept drifts to trigger the readjustment of hyperparameters. Contrary to our proposal, these AutoML tools use batch processing for hyperparameter optimisation, do not support recommendation nor readjust hyperparameters whenever concept drifts occur. Although existing meta-learning-based methods like V. Naumova *et al.* [37] can be used for online

Table 1: Comparison of AutoML Tools.

| **Tool** | **Processing** | | **ML Task** | | | **Concept Drift** |
|---|---|---|---|---|---|---|
| | Batch | Stream | Recom. | Regre. | Class. | |
| Auto-Net | ✓ | | | ✓ | ✓ | |
| Auto-Sklearn | ✓ | | | ✓ | ✓ | |
| Auto-Weka | ✓ | | | ✓ | ✓ | |
| H2O AutoML | ✓ | | | ✓ | ✓ | |
| Hyperopt | ✓ | | | ✓ | ✓ | |
| TPOT | ✓ | | | ✓ | ✓ | ✓ |
| **Our Requirements** | | ✓ | ✓ | ✓ | ✓ | ✓ |

data processing, they are trained offline (in batch learning), meaning that they do not perform online hyperparameter optimisation. Our approach transforms the Nelder-Mead heuristic optimisation algorithm into a stream-based optimisation algorithm. This new implementation only requires a single-pass over the data to optimise the set of parameters, making it more versatile and less computationally expensive.

## 3. Hyperparameter Self-Tuning for Data Streams

Hyperparameters are used to tune certain aspects of an algorithm that have direct impact on its performance, depending on the data it learns from [38].

The problem with the automation of the hyperparameter tuning process can be stated as follows: given a machine learning algorithm $A$ with a default hyperparameter configuration $c$, represented by $A_c$, and a data stream $S$, the goal is to find an optimal algorithm configuration $A_c^*$ where $A_c^*$ yields higher performance score than $A_c$. The optimised technique can be defined as $A_c^* \in \operatorname{argmin} L(A_c, S)$, where $L(A_c, S)$ is the loss function of algorithm $A$ applied to data stream $S$.

This article presents a Single-pass Self Parameter Tuning (`SSPT`) algorithm, which is a refined version of the Double-pass `SPT` (`DSPT`) algorithm, designed to optimise a set of hyperparameters in large search spaces. Both `SSPT` and `DSPT` versions modify the Nelder-Mead method [4] to work with data streams and

adopt a heuristics-based direct-search algorithm to select the best hyperparameters. The difference between SSPT and DSPT is that SSPT has the advantage of processing the data stream in single-pass. In both versions, the time complexity of the Nelder-Mead algorithm grows linearly with the number of hyperparameters. The overall complexity of the system is typically dominated by the underlying learning algorithm.

Figure 1 presents the two operation phases of all SPT versions: (*i*) exploration, where the goal is to seek convergence to a local minimum by exploring the multiple candidate models created by the Nelder-Mead operators; and (*ii*) deployment, where the algorithm uses the best learning model obtained in the exploration stage, continuously updating and monitoring it for the occurrence of concept drifts. The detection of a concept drift triggers a new exploration phase.

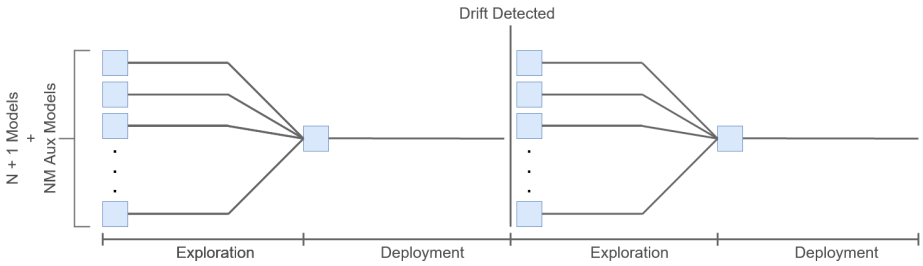



Figure 1: SPT Operation.

### *3.1. Exploration*

To find a solution for $n$ hyperparameters, Nelder-Mead creates a simplex with $n + 1$ learning models, each corresponding to a distinct hyperparameter configuration. In addition to these, we launch seven experimental models to enable comparisons and transitions in the four possible operations in the Nelder-Mead algorithm, as described in the following paragraphs. Exploration starts by randomly selecting the $n+1$ configurations of the learning models. SPT updates the learning and experimental models incrementally with each example in the data stream until the learning models converge to an optimal configuration.

Models can be trained in parallel to speed up the execution. This algorithm is a gradient-free simplex search algorithm for multidimensional unconstrained optimisation. The vertices of the simplex, that define the convex hull of the initial search space, are iteratively updated, discarding the vertex associated with the largest cost function value at each iteration. During exploration, the variance of the error of the learning models is used to dynamically define the sample size. Formally, the dynamic sample size $S_{size}$ is given by Equation (1), where $\sigma$ represents the error standard deviation and $C$ the confidence level. `SPT` uses a confidence level $C = 95\,\%$. The size of the sample increases when the variance of the error grows, and decreases otherwise, down to a lower bound of 30 samples, to avoid using small-sized samples.

$$S_{size} = \frac{16\sigma^2}{C^2} \tag{1}$$

Figure 2 illustrates the exploration stage. The vertical lines represent the boundaries of the dynamically-sized samples. Whenever the sample size changes, the

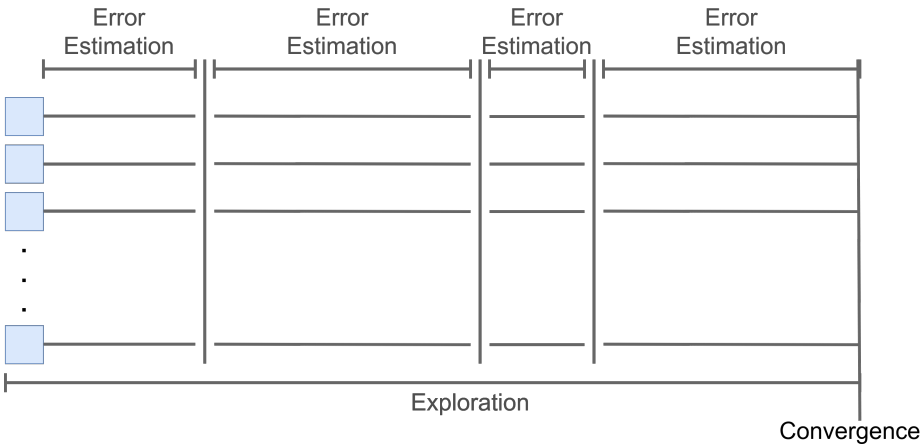
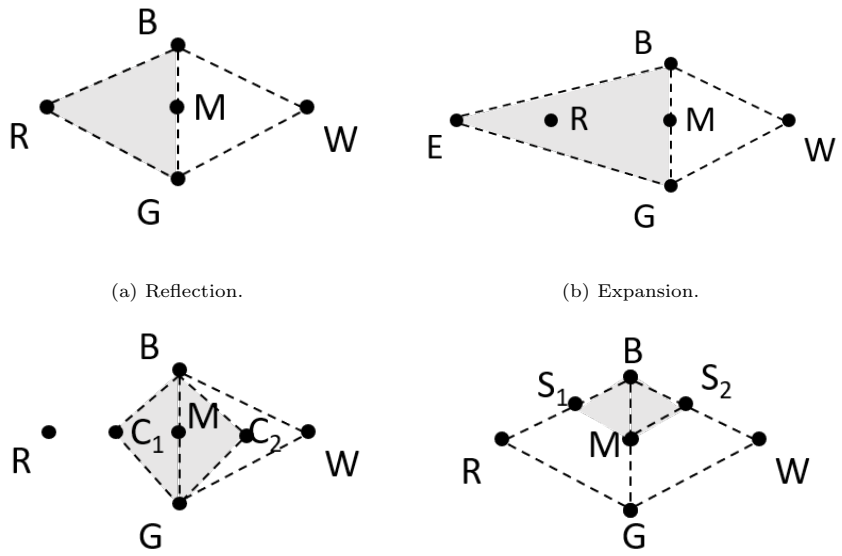



Figure 2: Exploration.

worst learning model is replaced by a copy of the best experimental model. Then, seven new experimental models are created, with hyperparameter configurations given by the Nelder-Mead algorithm, as detailed below.

The Nelder-Mead algorithm relies on four simple operations: reflection, ex-

pansion, contraction and shrinkage (Figure 3). The vertices are ordered by the




(a) Reflection.

(b) Expansion.

(c) Contraction.

(d) Shrinkage.

Figure 3: Nelder-Mead Operators. Reflection - We choose a test point $R$ that is obtained by reflecting the triangle $B, G, W$; Expansion - We extend the line segment through $M$ and $R$ to the point $E$. This forms an expanded triangle $B, G, E$; Contraction - The function is smaller at midpoint $M$, but to keep the triangle shape we cannot replace $W$ with $M$. Consider two midpoints $C_1$ and $C_2$ of the line segments $WM$ and $MR$; Shrinkage - The function value at $C$ is not less than the value at $W$, the points $G$ and $W$ must be shrunk toward $B$ resulting in points $S_1$ and $S_2$.

evaluation metric value: best ($B$), good ($G$), which is the second best vertex, and worst ($W$) [39]. For each Nelder-Mead operation, it is necessary to compute an additional set of vertices – midpoint ($M$), reflection ($R$), expansion ($E$), contraction ($C_1$ and $C_2$) and shrinkage ($S_1$ and $S_2$) – and verify if the calculated vertices belong to the search space. These operations are implemented by Algorithm 1.

In the case of reflection ($R$) or expansion ($E$), the algorithm computes the midpoint ($M$) of the best face of the shape as well as the reflection point ($R$). After this initial step, it determines whether to reflect or expand based on the

**Algorithm 1:** SSPT: Contraction, Expansion, Reflection and Shrinkage.

```
input: B; G; W; E; C1; C2; M; R; S1; S2
begin
    if f(R) < f(G) then
        if f(B) < f(R) then
            W = R
        else
            if f(E) < f(B) then
                W = E        /* Replace the Worst model by the Expansion model.  */
            else
                W = R        /* Replace the Worst model by the Reflection model.  */
    else
        if f(R) < f(W) then
            W = R
            /* Replace the Worst model by the Reflection model.                   */
            S = S1
            C = C1
        else
            S = S2
            C = C2
            if f(C) < f(W) then
                W = C        /* Replace the Worst model by the Contraction model.  */
            else
                if f(S) < f(W) then
                    W = S    /* Replace the Worst model by the Shrinkage model.  */
                if f(M) < f(G) then
                    G = M    /* Replace the Good model by the Midpoint model.  */
    M = (B + G)/2
    R = 2M − W
    E = 2R − M
    C1 = (R + M)/2
    C2 = (W + M)/2
    S = (B + W)/2
    /* All configurations are evaluated in parallel, using the next data sample.  */
```

set of predetermined heuristics. In the case of contraction ($C$) or shrinkage ($S$), it calculates two points. The two contraction points ($C_1$, $C_2$), located on both faces of the shape, are $C_1$, which corresponds to the midpoint between $M$ and $R$

vertices, and $C_2$, that corresponds to the midpoint between $M$ and $W$. Similarly, the two shrinkage points $(S_1, S_2)$, located on the two faces of the shape, are $S_1$, the midpoint between $B$ and $R$ vertices, and $S_2$, the midpoint between $B$ and $W$ vertices. Then, the algorithm decides whether to contract or to shrink based on the predetermined set of heuristics. This process is repeated during exploration until the best, good and worst learning models ($B$, $G$, $W$) converge to a sphere with a radius equal to the highest hyperparameter incremental step. The radius of the convergence sphere follows Jung's theorem [40], which formulates $d = \max_{p,q \in K} |\sqrt{\sum_{i=1}^{N}(p_i - q_i)^2}|$ where $p$ and $q$ are points containing $N$ euclidean dimensions and $K \subset \mathbb{R}^n$ is the largest Euclidean distance between any pair $p$ and $q$. Jung's theorem states that there exists a sphere with radius $r \leq d\sqrt{\frac{n}{2(n+1)}}$ containing $K$.

Algorithm 1 details the pseudo-code of the new single-pass `SPT` (`SSPT`). Like the previous double-pass `SPT` (`DSPT`), it creates $n+1$ learning models plus seven experimental models, using shallow copies of the best model $B$, and applies the Nelder-Mead operators. The main difference between `SSPT` and `DSPT` is that the first uses the current data window to determine the possible Nelder-Mead configurations and then assesses them using the next data window, whereas the latter uses the current data window to determine and evaluate the possible Nelder-Mead configurations with a double pass. The pseudocode of `DSPT` is available in Appendix A (Algorithm 2), for reference.

### *3.2. Deployment*

During the deployment stage, `SPT` is an event-driven algorithm that continuously updates the current learning model, monitors the error and reacts to concept drift. In this work, concept drifts are detected with the Drift Detection Method (DDM) proposed by Gama *et al.* [41]. Whenever DDM detects a concept drift, `SPT` triggers a new exploration phase to adjust the hyperparameters according to new data.

## 4. Experiments and Results

This section describes the experiments, including the learning tasks, base algorithms, data sets, and adopted evaluation metrics. The experiments were run on an Intel Xeon CPU E5-2680 2.40 GHz Central Processing Unit (CPU), 32 GB DDR3 Random Access Memory (RAM) and 1 TB hard drive, running the Linux Ubuntu 16.04 OS. All algorithms were implemented in the Massive Online Analysis (MOA) framework [42]. MOA is an open-source framework for the development and execution of machine learning and data mining experiments on evolving data streams.

The research questions we address are:

1. Which variant of `SPT` obtains better results?
2. How does `SPT` compare against other hyperparameter tuning methods?
3. How does `SPT` react to drift?

Since `SPT` is a wrapper around a learning algorithm, it can be used in different learning tasks. In this work, we evaluate `SPT` on three stream-based learning tasks: Recommendation, Regression and Classification. `SPT` is, as far as we know, the first online hyperparameter tuning algorithm for data stream learning.

### *4.1. Learning Tasks and Learning Algorithms*

`SPT` searches for the best hyperparameter configuration of the base learner, given a data set. Table 2 describes, for each task, which base learner we use and what hyperparameters `SPT` is tuning. We compare `SPT` against its natural competitors: default, random search and grid search parameter optimisation. These offline techniques rely either on handcrafting by experts (default) or brute force algorithms (random and grid search) for hyperparameter optimisation. Table 2 presents parameter initialisation values for the default and grid configurations.

In the case of the recommendation algorithm, the default hyperparameter initialisation was 0.164 for the learning rate and 0.001 for the regularisation parameters. These values are typically the result of an offline optimisation

Table 2: Algorithms – Hyperparameter Initialisation.

| Task | Algorithm | Parameter | Default | Grid | | |
|---|---|---|---|---|---|---|
| | | | | Start | End | Increment |
| Recommendation | BRISMF[1] | Learning rate | 0.164 | 0.1 | 1.0 | 0.1 |
| | | Regularisation | 0.001 | 0.1 | 1.0 | 0.1 |
| Regression | AMRules[2] | Split confidence | 0.001 | 0.001 | 0.01 | 0.001 |
| | | Tie Threshold | 0.05 | 0.01 | 0.1 | 0.01 |
| Classification | VFDT[3] | Grace Period | 200 | 50 | 450 | 40 |
| | | Tie Threshold | 0.05 | 0.01 | 0.1 | 0.01 |

[1] Biased Regularised Incremental Simultaneous Matrix Factorisation (BRISMF) [43]

[2] Adaptive Model Rules (AMRules) [44]

[3] Very Fast Decision Trees (VFDT) [45]

performed by experts. For the regression algorithm, the default initialisation was 0.001 for the split confidence, 0.05 for the tie threshold. In the classification task, the values were 200 for the grace period and 0.05 for the tie threshold.

Grid search uses different incremental steps for each ML task: recommendation – learning rate and regularisation hyperparameter have increments of 0.0333; regression – split confidence and tie threshold have increments of 0.000 333 and 0.0033, respectively; and classification – grace period has increments of 150 and tie threshold 0.003 33.

Random search, grid search and `SPT` explore ten hyperparameter configurations. Then, average performance, memory and run-time are analysed. Due to memory limitations, random and grid search were run sequentially, rather than in parallel, in our experiments.

### *4.2. Evaluation Protocol*

The evaluation protocol defines the data ordering, partitions and distribution. The proposed method was assessed with the predictive sequential (prequential) evaluation protocol for data streams [46].

The goal of the recommendation experiments is to optimise the learning rate and regularisation hyperparameters. These experiments were performed by

defining new recommendation tasks in MOA. The recommendation algorithm is the BRISMF algorithm proposed by Takacs *et al.* [43].

The goal of the regression experiments is to optimise the split confidence and tie-threshold hyperparameters. These experiments were performed by defining new regression tasks in MOA. These tasks use the AMRules regression algorithm of Duarte *et al.* [47].

The goal of the classification experiments is to optimise the grace period and tie-threshold hyperparameters. The experiments were performed by defining new classification tasks in MOA. These tasks use the VFDT classification algorithm proposed by Domingos *et al.* [45].

The evaluation metrics adopted to tune the algorithms were the Root Mean Squared Error (RMSE) for recommendation and regression tasks and the $0-1$ loss function (error-rate) for classification. Experiments were repeated 10 times and the results were used to calculate the average performance critical distance as well as the average run-time and memory consumption. The memory consumption was measured using a Java Agent – sizeofag[1] used by MOA.

### *4.3. Data Sets*

For each of the three learning tasks, we selected six publicly available benchmark data sets, large enough to be processed as a stream. Table 3 summarises the data sets used in the three tasks.

In the recommendation experiments we use CiaoDVD, FilmTrust, GoodBooks, Jester, MovieLens 100k (ML100k) and MovieLens 1M (ML1M). For the regression experiments, we chose 2DPlanes, Ailerons, Fried, Sgemm, Transcoding, and YearPredictionMSD. Finally, the data sets used in the classification task are Agrawal, Bank Marketing, Electricity, Postures, SEA and Tweet500. The contents of these data sets is detailed in Appendix B.

[1] `github.com/fracpete/sizeofag`

Table 3: Data Sets Used in the Experimental Section.

| **TASK: Recommendation** | | | | |
|---|---|---|---|---|
| **Data set** | **Users** | **Items** | **Ratings** | **Sparsity** (%) |
| CiaoDVD | 17 615 | 16 121 | 72 665 | 99.9 |
| FilmTrust | 1508 | 2071 | 35 497 | 98.9 |
| GoodBooks | 53 424 | 10 000 | 961 756 | 99.8 |
| Jester | 73 421 | 100 | 4 100 000 | 44.2 |
| ML100k | 943 | 1682 | 100 000 | 93.7 |
| ML1M | 3706 | 6040 | 1 000 209 | 95.5 |
| **TASK: Regression** | | | | |
| **Data set** | | | **Instances** | **Attributes** |
| 2DPlanes | | | 40 768 | 11 |
| Ailerons | | | 13 750 | 41 |
| Fried | | | 40 768 | 11 |
| Sgemm | | | 241 600 | 18 |
| Transcoding | | | 168 286 | 11 |
| YearPredictionMSD | | | 515 345 | 90 |
| **TASK: Classification** | | | | |
| **Data set** | | | **Instances** | **Attributes** | **Classes** |
| Agra | | | 1 000 000 | 9 | 2 |
| Bank | | | 45 211 | 17 | 2 |
| Electricity | | | 45 312 | 8 | 2 |
| Postures | | | 78 095 | 38 | 5 |
| SEA | | | 60 000 | 3 | 4 |
| Tweet500 | | | 1 000 000 | 500 | 2 |

### *4.4. Learning Goals*

We use different base learners for the different learning tasks. The algorithm used for the recommendation task, is the *biased regularised incremental simultaneous matrix factorisation* (BRISMF) algorithm [43]. The goal of the recommendation experiments is to optimise the *learning rate* and the *regularisation* hyperparameters.

The base regression algorithm is the *Adaptive Model Rules* (AMRules) [47]. The goal of the regression experiments is to optimise the *split confidence* and the *tie-threshold* hyperparameters.

The base learning algorithm for classification is the *Very Fast Decision Tree* (VFDT) algorithm [45]. The goal of the classification experiments is to optimise the *grace period* and the *tie-threshold* hyperparameters.

### *4.5. Which Variant of SPT Performs Better?*

In this subsection we compare the behavior of different variants of `SPT`. The Double-pass `SPT` (`DSPT`) was first discussed in [7]. The Single-pass `SPT` (`SSPT`), and Warm Start Single-pass `SPT` (`WSSPT`) are original contributions of this paper.

At start-up, the three `SPT` versions create three identical learning models. `DSPT` and `SSPT` initialise the hyperparameters of the three models with random values (random start), while `WSSPT` initialises the hyperparameters of two models randomly and the third model with the default initialisation values (warm start).

Table 4 presents, for the different learning tasks, performance results of the three variants of `SPT` in comparison with the default configuration of the base learner. We present the average accuracy per data set, as well as the overall average across data sets of accuracy, rank, memory usage and run-time for the different variants.

#### *4.5.1. Discussion*

Considering the time required to converge to a local optimal solution and the accuracy, our results show that `SSPT` is generally better than `DSPT`. Moreover, the influence of the warm start becomes clear when comparing the random start `SSPT` and warm start `WSSPT` results. Regardless of the version, `SPT` uses obviously more memory than the default algorithm, since it requires enough memory to hold ten (three learning and seven experimental) models during exploration[2].

Although the default parameter configuration presents higher accuracy than random start `SPT` versions in regression and classification, it is unable to react to concept drift and relies on the availability, expertise and time of data scientists for optimisation, making it inappropriate for data streams.

[2]This means that, in the case of `SPT` versions, it is possible to optimise memory usage during deployment by freeing the memory occupied by all but the best learning model.

Table 4: Comparison Between SPT Variants.

| **TASK: Recommendation** | | | | |
|---|---|---|---|---|
| **Data set** | **Default** | DSPT | SSPT | WSPT |
| CiaoDVD | 0.224±0.000 | 0.220±0.002 | 0.224±0.006 | 0.222±0.004 |
| FilmTrust | 0.233±0.001 | 0.218±0.028 | 0.186±0.011 | 0.187±0.006 |
| GoodBooks | 0.193±0.001 | 0.189±0.001 | 0.190±0.003 | 0.189±0.001 |
| Jester | 0.288±0.002 | 0.231±0.005 | 0.234±0.002 | 0.241±0.014 |
| ML100k | 0.205±0.001 | 0.178±0.007 | 0.174±0.005 | 0.179±0.004 |
| ML1M | 0.200±0.000 | 0.196±0.002 | 0.197±0.006 | 0.196±0.001 |
| Avg. Performance | 0.224 | 0.205 | 0.201 | 0.202 |
| Avg. Rank | 3.83 | 1.50 | 2.17 | 2.00 |
| Avg. Time | 1.000 | 1.515 | 1.157 | 1.439 |
| Avg. Memory | 1.000 | 2.995 | 2.995 | 2.994 |
| **TASK: Regression** | | | | |
| **Data set** | **Default** | DSPT | SSPT | WSPT |
| 2DPlanes | 1.276±0.000 | 1.429±0.093 | 1.309±0.032 | 1.366±0.103 |
| Ailerons | 0.173±0.000 | 0.194±0.019 | 0.207±0.010 | 0.200±0.037 |
| Fried | 2.316±0.000 | 2.307±0.067 | 2.330±0.101 | 2.376±0.151 |
| Sgemm | 0.842±0.000 | 2.778±3.822 | 5.664±4.569 | 4.317±3.187 |
| Transcoding | 17.452±0.000 | 18.108±8.623 | 21.722±10.978 | 12.816±0.000 |
| YearPredMSD | 15.117±0.000 | 15.900±0.383 | 15.999±0.365 | 16.051±0.574 |
| Geometric Mean | 2.201 | 2.828 | 3.279 | 2.885 |
| Avg. Performance | 6.196 | 6.786 | 7.872 | 6.188 |
| Avg. Rank | 1.33 | 2.33 | 3.33 | 3.00 |
| Avg. Time | 1.000 | 2.343 | 2.302 | 9.305 |
| Avg. Memory | 1.000 | 1.768 | 1.857 | 2.318 |
| **TASK: Classification** | | | | |
| **Data set** | **Default** | DSPT | SSPT | WSPT |
| Agra | 0.122±0.000 | 0.324±0.113 | 0.160±0.067 | 0.161±0.049 |
| Bank | 0.285±0.000 | 0.334±0.077 | 0.280±0.037 | 0.276±0.032 |
| Electricity | 0.271±0.000 | 0.186±0.041 | 0.139±0.042 | 0.125±0.027 |
| Postures | 0.252±0.000 | 0.478±0.456 | 0.694±0.366 | 0.448±0.359 |
| SEA | 0.131±0.000 | 0.217±0.046 | 0.132±0.042 | 0.125±0.005 |
| Tweet500 | 0.101±0.000 | 0.116±0.017 | 0.109±0.015 | 0.113±0.020 |
| Avg. Performance | 0.194 | 0.276 | 0.252 | 0.208 |
| Avg. Rank | 2.00 | 3.67 | 2.50 | 1.83 |
| Avg. Time | 1.000 | 226.191 | 3.214 | 3.368 |
| Avg. Memory | 1.000 | 23.901 | 5.579 | 6.588 |

### *4.6. How Does SPT Compare Against Other Hyperparameter Tuning Methods?*

Table 5: Comparison Against Competing Techniques.

| **TASK: Recommendation** | | | | |
|---|---|---|---|---|
| **Data set** | **Default** | **Grid** | **Random** | `WSPT` |
| CiaoDVD | 0.224±0.000 | 0.378±0.069 | 0.305±0.056 | 0.222±0.004 |
| FilmTrust | 0.233±0.001 | 0.334±0.068 | 0.249±0.074 | 0.187±0.006 |
| GoodBooks | 0.193±0.001 | 0.425±0.086 | 0.330±0.096 | 0.189±0.001 |
| Jester | 0.288±0.002 | 0.391±0.076 | 0.320±0.056 | 0.241±0.014 |
| ML100k | 0.205±0.001 | 0.405±0.084 | 0.280±0.113 | 0.179±0.004 |
| ML1M | 0.200±0.000 | 0.403±0.079 | 0.196±0.074 | 0.196±0.001 |
| Avg. Performance | 0.224 | 0.389 | 0.297 | 0.202 |
| Avg. Rank | 2.00 | 4.00 | 3.00 | 1.00 |
| Avg. Time | 1.000 | 12.498 | 12.639 | 1.439 |
| Avg. Memory | 1.000 | 10.000 | 10.000 | 2.994 |
| **TASK: Regression** | | | | |
| **Data set** | **Default** | **Grid** | **Random** | `WSPT` |
| 2DPlanes | 1.276±0.000 | 1.390±0.000 | 1.372±0.057 | 1.366±0.103 |
| Ailerons | 0.173±0.000 | 0.175±0.000 | 0.175±0.000 | 0.200±0.037 |
| Fried | 2.316±0.000 | 2.231±0.000 | 2.231±0.000 | 2.376±0.151 |
| Sgemm | 0.842±0.000 | 1.219±0.000 | 1.668±1.419 | 4.317±3.187 |
| Transcoding | 17.452±0.000 | 15.195±0.000 | 15.053±0.449 | 12.816±0.000 |
| YearPredMSD | 15.117±0.000 | 16.438±0.000 | 16.438±0.000 | 16.051±0.574 |
| Geometric Mean | 2.201 | 2.343 | 2.459 | 2.885 |
| Avg. Performance | 6.196 | 6.108 | 6.156 | 6.188 |
| Avg. Rank | 1.83 | 2.50 | 2.33 | 2.83 |
| Avg. Time | 1.000 | 12.217 | 11.704 | 9.305 |
| Avg. Memory | 1.000 | 15.809 | 15.752 | 2.318 |
| **TASK: Classification** | | | | |
| **Data set** | **Default** | **Grid** | **Random** | `WSPT` |
| Agra | 0.122±0.000 | 0.111±0.077 | 0.105±0.046 | 0.161±0.049 |
| Bank | 0.285±0.000 | 0.252±0.019 | 0.306±0.056 | 0.276±0.032 |
| Electricity | 0.271±0.000 | 0.145±0.032 | 0.133±0.021 | 0.125±0.027 |
| Postures | 0.252±0.000 | 0.016±0.009 | 0.037±0.028 | 0.448±0.359 |
| SEA | 0.131±0.000 | 0.117±0.002 | 0.120±0.005 | 0.125±0.005 |
| Tweet500 | 0.101±0.000 | 0.111±0.015 | 0.103±0.008 | 0.113±0.020 |
| Avg. Performance | 0.194 | 0.125 | 0.134 | 0.208 |
| Avg. Rank | 3.00 | 1.84 | 2.17 | 3.00 |
| Avg. Time | 1.000 | 12.388 | 14.918 | 3.368 |
| Avg. Memory | 1.000 | 15.864 | 38.943 | 6.588 |

In this section, we compare the best SPT strategy against *grid search* and *random search* optimisation. Both are offline brute force techniques that require longer exploration stages, making them inappropriate for data stream problems. Table 5 also includes the default configuration for reference.

Results show that SPT is faster and requires less memory than its competitors. In the next section, we compare the different models using critical distance performance plots.

#### *4.6.1. Discussion on Recommendation Task*

In the recommendation task all SPT variants rank better that the default configuration – Table 4 – for all data sets. The best strategy is SSPT. All SPT variants require three times more memory than the default configuration, but the overhead in actual run-time is reasonably small. Moreover, the critical distance diagrams (Figure 4) show that all SPT variants improve over grid search and random search in all data sets. This is the task that most clearly illustrates the advantage of SPT.

Regarding time, the main advantage of SSPT naturally occurs in the exploration phase. Regardless of the data set, it is consistently the fastest optimisation algorithm. The time spent on the exploration (the most demanding phase) is, for the CiaoDVD, FilmTrust, GoodBooks, Jester, ML100k and ML1M data sets, 61.2 %, 53.7 %, 45.9 %, 66.0 %, 88.4 % and 89.5 % of the total time presented on Table 5.

Taking both the convergence time and the accuracy of the deployed model, results show that, on average, SSPT outperforms DSPT. Considering the recommendation accuracy of models configured by the analysed hyperparameter optimisation techniques, grid and random search are the only SPT competitors. However, not only do they take considerably longer to explore the search space (exploration phase), but also they are offline algorithms, unsuitable for stream-based recommendation.

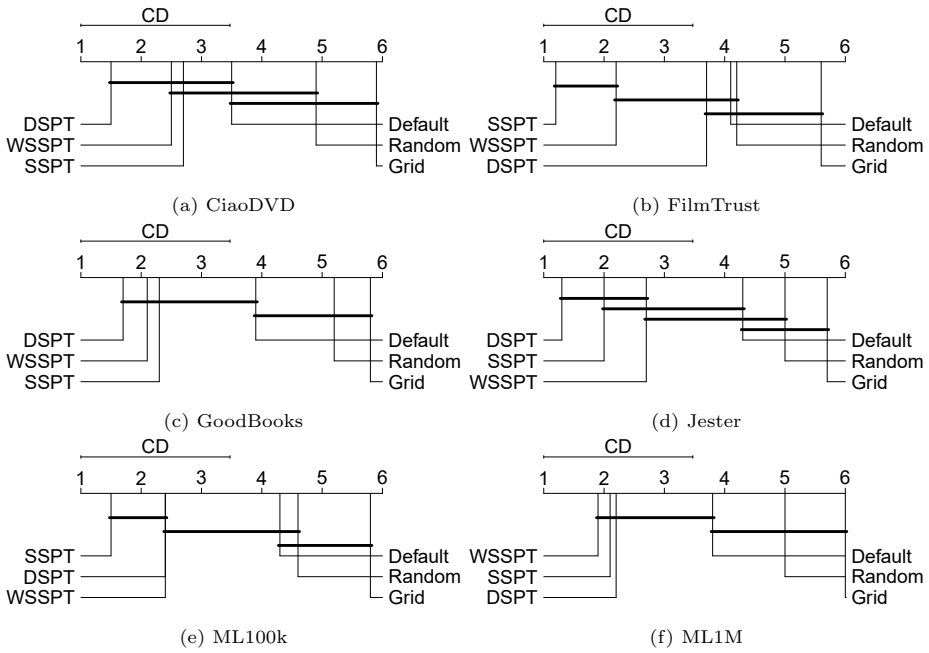



Figure 4: Recommendation – Performance Critical Distance (lower is better) of Default Parameter Initialisation, Grid Search, and Random Search, `DSPT`, `SSPT` and `WSSPT`.

*4.6.2. Discussion on Regression Task*

Figure 5 displays the critical distance accuracy plots of the six data sets and optimisation techniques for the regression task. Results show that, for all data sets and with a confidence level of 95 %, there are no significant differences between the three variants of `SPT`. The variant with better rank is the double-pass `SPT`. In terms of ranking, the default parameter configuration yields the lowest (best) rank. Grid search, random search and `WSSPT` have somewhat similar ranks.

Regarding time, `SSPT` is again faster in the exploration phase. The exploration time for the 2DPlanes, Ailerons, Fried, Sgemm, Transcoding and YearPredictionMSD data sets is 2.6 %, 37.9 %, 4.7 %, 4.2 %, 27.7 % and 0.6 % of the total time presented on Table 5. Considering the time required to converge to a local optimal solution and the accuracy, the results show that the `SSPT` is better than the previous `DSSPT`. In terms of performance, `DSPT` ranks better, on average,

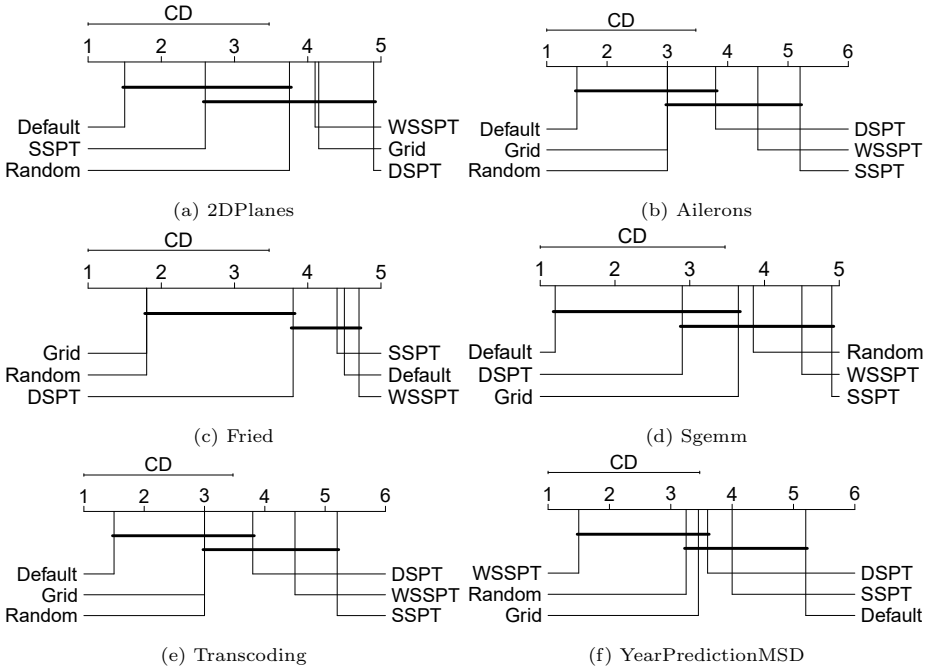



Figure 5: Regression – Performance Critical Distance (lower is better) of Default Initialisation, Grid Search, Random Search, `DSPT`, `SSPT` and `WSSPT`.

than `SSPT`.

Although the offline optimisation techniques ranked slightly better, they are unsuited for the problem at hand.

### *4.6.3. Discussion on Classification Task*

Figure 6 depicts, for the classification task, the critical distance performance plots with all six data sets and hyperparameter optimisation techniques. The accuracy rank results show that, with five data sets – all but SEA – and a confidence level of 95 %, the proposed `SSPT` is not significantly different from `DSPT`. In accuracy ranking, `SSPT` is only outperformed by the grid and random search offline techniques, in the case of the Postures data set.

In the exploration phase, the `SSPT` is faster with all data sets. The exploration time of `SSPT` with the Agra, Bank, Electricity, Postures, SEA and Tweet500 data sets is, respectively, 44.7 %, 55.5 %, 53.8 %, 54.6 %, 69.8 % and

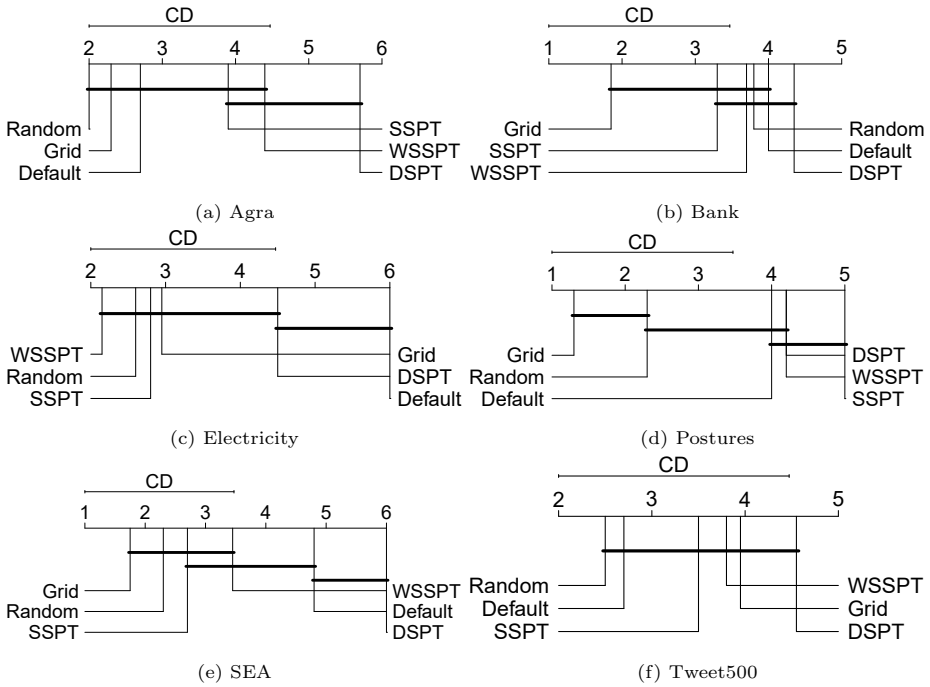



Figure 6: Classification – Performance Critical Distance (lower is better) of Default Initialisation, Grid Search, Random Search, `DSPT`, `SSPT`, and `WSSPT`.

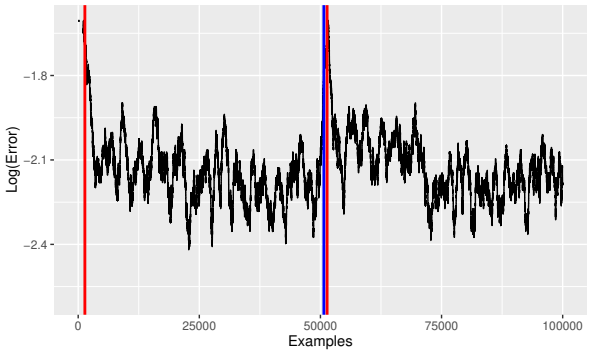
17.4 % of the total time presented on Table 5. Moreover, `SSPT` provides, on average, similar accuracy to `DSPT`.

Taking both the time to converge to a local optimal solution and the accuracy, the results show that the new `SSPT` is better than `DSPT`. Grid and random search are more accurate, but require longer exploration and are offline techniques not applicable to data streams.

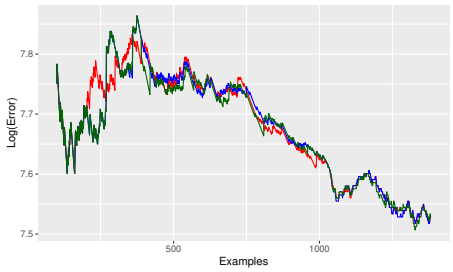
### *4.7. How Does SPT React to Drift?*

An essential feature of `SPT` is that it uses the information produced by the Drift Detector Method (DDM) [41] to restart the optimisation process. To assess the effectiveness of `SPT` when drift occurs, we conducted a dedicated experiment. We generated a synthetic data stream using the SEA generator, provided in the MOA framework. The stream contains 100 000 examples with a drift at position

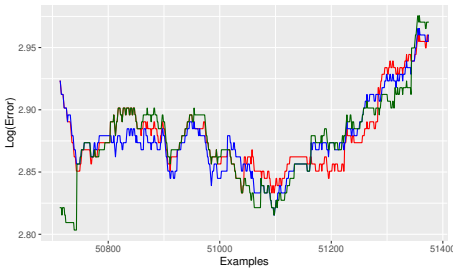
50 000. The learning algorithm is VFDT and the drift detector is DDM. Figure 7 (a) plots the logarithm of the prequential error. The two red vertical lines indicate the moment that `SPT` converges to one solution, and the blue vertical line indicates the moment where DDM detects the drift. Figure 7 (b) displays the initial exploration phase, where `SPT` required 1380 examples to converge. DDM detected a drift at example 50 714 and `SPT` restarted the optimisation process plotted in Figure 7 (c). After 660 examples, the algorithm converged to a new optimal solution.




(a) Logarithmic Error.




(b) Initial Exploration†.




(c) Drift-triggered Exploration†.

†Colours represent different exploration models.

Figure 7: Task: Classification. `SPT` reaction to drift.

## 5. Conclusions

The goal of this research was to explore and present an online solution for the automated optimisation of hyperparameters. This article describes the Single-

pass Self Parameter Tuning algorithm for data streams. SPT explores the adoption of a simplex search mechanism, combined with dynamically-sized samples and concept drift detection, to find hyperparameter configurations that minimise the objective function. Experiments conducted with several data sets show that the automatic selection of hyperparameters has a substantial impact on the outcomes of stream-based recommendation, regression and classification tasks. On what concerns automatic hyperparameter tuning, the novel aspects of this work include: (*i*) the single-pass data exploration mode; and (*ii*) the extended evaluation using three different machine learning tasks (recommendation, regression and classification). The advantages of the proposed algorithm are that it: (*i*) works with data streams; (*ii*) is applicable to different machine learning tasks (recommendation, regression and classification); and (*iii*) reacts to the variability of the data. Finally, the results show that: (*i*) SSPT is faster to converge, which is relevant when processing data streams; and (*ii*) concept drift automatically triggers the search for a new optimal solution. SSPT is, as far as we know, the first online single-pass hyperparameter tuning algorithm for data stream learning, *i.e.*, it is able to tune hyperparameters on the fly, without any initialisation stage, based exclusively on the incoming data stream. Nonetheless, since SPT finds local optimal solutions, they may not correspond to the global optimal solution as was shown when compared with offline techniques like default or grid search. Future work will address: (*i*) the ability to select not only hyperparameters but also models; (*ii*) the comparison of SPT with other emergent AutoML optimisation algorithms; and (*iii*) the analysis of the robustness of SPT with different types of drifts. Further research will be required to assess the behaviour of SPT in face of these challenges.

## 6. Acknowledgements

This work was partially supported by: (*i*) National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) as part of project UIDB/50014/2020; and (*ii*) the European

Commission funded project "Humane AI: Toward AI Systems That Augment and Empower Humans by Understanding Us, our Society and the World Around Us" (grant #820437). The support is gratefully acknowledged.

## 7. Credit

Bruno Veloso: Software, Validation, Investigation, Writing - Original draft preparation. João Gama: Conceptualization, Methodology, Writing. Benedita Malheiro: Resources, Writing - Reviewing and Editing. João Vinagre: Writing - Reviewing and Editing.

## References

[1] P. M. Lerman, Fitting segmented regression models by grid search, Journal of the Royal Statistical Society: Series C (Applied Statistics) 29 (1) (1980) 77–84. doi:10.2307/2346413.

[2] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, Journal of Machine Learning Research 13 (1) (2012) 281–305. doi:10.5555/2188385.2188395.

[3] D. Maclaurin, D. Duvenaud, R. P. Adams, Gradient-based hyperparameter optimization through reversible learning, in: Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, ICML'15, JMLR.org, 2015, pp. 2113–2122. doi:10.5555/3045118.3045343.

[4] J. A. Nelder, R. Mead, A Simplex Method for Function Minimization, The Computer Journal 7 (4) (1965) 308–313. doi:10.1093/comjnl/7.4.308.

[5] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM computing surveys (CSUR) 46 (4) (2014) 1–37.

[6] B. Veloso, J. Gama, B. Malheiro, Classification and recommendation with data streams, in: M. Khosrow-Pour (Ed.), Encyclopedia of Information Science and Technology, Fifth Edition, 5th Edition, Vol. 2, The name of the publisher, IGI Global, Hershey, PA, USA, 2021, Ch. 47, pp. 675–684. doi:10.4018/978-1-7998-3479-3.ch047.

[7] B. Veloso, J. Gama, B. Malheiro, Self hyper-parameter tuning for data streams, in: L. Soldatova, J. Vanschoren, G. Papadopoulos, M. Ceci (Eds.), Discovery Science, Springer International Publishing, Cham, 2018, pp. 241–255. doi:10.1007/978-3-030-01771-2_16.

[8] B. Veloso, J. Gama, B. Malheiro, J. Vinagre, Self hyper-parameter tuning for stream recommendation algorithms, in: A. Monreale, C. Alzate, M. Kamp, Y. Krishnamurthy, D. Paurat, M. Sayed-Mouchaweh, A. Bifet, J. Gama, R. P. Ribeiro (Eds.), ECML PKDD 2018 Workshops, Springer International Publishing, Cham, 2019, pp. 91–102. doi:10.1007/978-3-030-14880-5_8.

[9] P. B. Brazdil, C. Soares, J. P. da Costa, Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results, Machine Learning 50 (3) (2003) 251–277. doi:10.1023/A:1021713901879.

[10] M. Feurer, F. Hutter, Hyperparameter Optimization, Springer International Publishing, Cham, 2019, Ch. 1, pp. 3–33. doi:10.1007/978-3-030-05318-5_1.

[11] R. Elshawi, M. Maher, S. Sakr, Automated machine learning: State-of-the-art and open challenges (2019). arXiv:1906.02287.

[12] R. Kohavi, G. H. John, Automatic parameter selection by minimizing estimated error, in: A. Prieditis, S. Russell (Eds.), Machine Learning Proceedings 1995, Morgan Kaufmann, San Francisco (CA), 1995, pp. 304 – 312. doi:10.1016/B978-1-55860-377-6.50045-1.

[13] A. Shaker, E. Lughofer, Self-adaptive and local strategies for a smooth treatment of drifts in data streams, Evolving Systems 5 (4) (2014) 239–257. doi:10.1007/s12530-014-9108-y.

[14] C. Finn, P. Abbeel, S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, in: D. Precup, Y. W. Teh (Eds.), Proceedings of the 34th International Conference on Machine Learning, Vol. 70 of Proceedings of Machine Learning Research, PMLR, International Convention Centre, Sydney, Australia, 2017, pp. 1126–1135. doi:10.5555/3305381.3305498.

[15] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, Hyperband: A novel bandit-based approach to hyperparameter optimization, J. Mach. Learn. Res. 18 (2017) 185:1–185:52.
URL http://jmlr.org/papers/v18/16-558.html

[16] A. Nichol, J. Achiam, J. Schulman, On first-order meta-learning algorithms, CoRR abs/1803.02999. arXiv:1803.02999.

[17] C. Lin, M. Guo, C. Li, X. Yuan, W. Wu, J. Yan, D. Lin, W. Ouyang, Online hyper-parameter learning for auto-augmentation strategy, in: Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 6579–6588. doi:10.1109/ICCV.2019.00668.

[18] K. G. Jamieson, A. Talwalkar, Non-stochastic best arm identification and hyperparameter optimization, in: A. Gretton, C. C. Robert (Eds.), Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016, Vol. 51 of JMLR Workshop and Conference Proceedings, JMLR.org, 2016, pp. 240–248.
URL http://proceedings.mlr.press/v51/jamieson16.html

[19] N. Koenigstein, G. Dror, Y. Koren, Yahoo! music recommendations: Modeling music ratings with temporal dynamics and item taxonomy, in: Proceedings of the Fifth ACM Conference on Recommender Systems, Rec-

Sys '11, ACM, New York, NY, USA, 2011, pp. 165–172. doi:10.1145/2043932.2043964.

[20] R. Kar, A. Konar, A. Chakraborty, A. L. Ralescu, A. K. Nagar, Extending the nelder-mead algorithm for feature selection from brain networks, in: 2016 IEEE Congress on Evolutionary Computation (CEC), 2016, pp. 4528–4534. doi:10.1109/CEC.2016.7744366.

[21] S. da Silva Fernandes, H. F. Tork, J. M. P. da Gama, The initialization and parameter setting problem in tensor decomposition-based link prediction, in: 2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2017, pp. 99–108. doi:10.1109/DSAA.2017.83.

[22] P. Pfaffe, M. Tillmann, S. Walter, W. F. Tichy, Online-autotuning in the presence of algorithmic choice, in: 2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2017, pp. 1379–1388. doi:10.1109/IPDPSW.2017.28.

[23] I. A. Lawal, S. A. Abdulkarim, Adaptive SVM for Data Stream Classification, South African Computer Journal 29 (2017) 27 – 42. doi:10.18489/sacj.v29i1.414.

[24] H. Zhan, G. Gomes, X. S. Li, K. Madduri, K. Wu, Efficient online hyperparameter optimization for kernel ridge regression with applications to traffic time series prediction (2018). arXiv:1811.00620.

[25] L. L. Minku, A novel online supervised hyperparameter tuning procedure applied to cross-company software effort estimation, Empirical Software Engineering 24 (5) (2019) 3153–3204. doi:10.1007/s10664-019-09686-w.

[26] Y. Zhou, W. Liu, B. Li, Efficient online hyperparameter adaptation for deep reinforcement learning, in: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), Springer, 2019, pp. 141–155. doi:10.1007/978-3-030-16692-2_10.

[27] L. Franceschi, M. Donini, P. Frasconi, M. Pontil, Forward and reverse gradient-based hyperparameter optimization, in: Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, 2017, pp. 1165–1173. doi:10.5555/3305381.3305502.

[28] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, D. D. Cox, Hyperopt: a python library for model selection and hyperparameter optimization, Computational Science & Discovery 8 (1) (2015) 014008. doi:10.1088/1749-4699/8/1/014008.

[29] C. Thornton, F. Hutter, H. H. Hoos, K. Leyton-Brown, Auto-weka: Combined selection and hyperparameter optimization of classification algorithms, in: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13, ACM, New York, NY, USA, 2013, pp. 847–855. doi:10.1145/2487575.2487629.

[30] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, K. Leyton-Brown, Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka, Journal of Machine Learning Research 18 (1) (2017) 826–830. doi:10.5555/3122009.3122034.

[31] R. S. Olson, J. H. Moore, TPOT: A Tree-Based Pipeline Optimization Tool for Automating Machine Learning, Springer International Publishing, Cham, 2019, Ch. 8, pp. 151–160. doi:10.1007/978-3-030-05318-5_8.

[32] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, in: C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, R. Garnett (Eds.), Advances in Neural Information Processing Systems 28, Curran Associates, Inc., 2015, pp. 2962–2970. doi:10.5555/2969442.2969547.

[33] H. Mendoza, A. Klein, M. Feurer, J. T. Springenberg, M. Urban, M. Burkart, M. Dippel, M. Lindauer, F. Hutter, Towards Automatically-Tuned Deep Neural Networks, Springer International Publishing, Cham, 2019, Ch. 7, pp. 135–149. doi:10.1007/978-3-030-05318-5_7.

[34] E. LeDell, S. Poirier, H2O AutoML: Scalable automatic machine learning, 7th ICML Workshop on Automated Machine Learning (AutoML).
URL https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf

[35] B. Celik, J. Vanschoren, Adaptation strategies for automated machine learning on evolving data (2021). arXiv:2006.06480.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, Journal of machine learning research 12 (Oct) (2011) 2825–2830. doi:10.5555/1953048.2078195.

[37] V. Naumova, S. V. Pereverzyev, S. Sivananthan, A meta-learning approach to the regularized learning—case study: Blood glucose prediction, Neural Networks 33 (2012) 181–193.

[38] M. Claesen, B. D. Moor, Hyperparameter search in machine learning (2015). arXiv:1502.02127.

[39] J. H. Mathews, K. D. Fink, et al., Numerical methods using MATLAB, 4th Edition, Pearson prentice hall Upper Saddle River, NJ, 2004.

[40] H. Jung, Ueber die kleinste kugel, die eine räumliche figur einschliesst, Journal für die reine und angewandte Mathematik 123 (1901) 241–257. doi:10.1515/crll.1901.123.241.

[41] J. Gama, P. Medas, G. Castillo, P. P. Rodrigues, Learning with drift detection, in: A. L. C. Bazzan, S. Labidi (Eds.), Advances in Artificial Intelligence - SBIA 2004, 17th Brazilian, Vol. 3171 of Lecture Notes in Computer Science, Springer, 2004, pp. 286–295. doi:10.1007/978-3-540-28645-5_29.

[42] A. Bifet, R. Gavaldà, G. Holmes, B. Pfahringer, Machine Learning for Data Streams with Practical Examples in MOA, MIT Press, 2018. doi:10.7551/mitpress/10654.001.0001.

[43] G. Takács, I. Pilászy, B. Németh, D. Tikk, Scalable collaborative filtering approaches for large recommender systems, Journal of Machine Learning Research 10 (2009) 623–656. doi:10.5555/1577069.1577091.

[44] E. Almeida, C. Ferreira, J. Gama, Adaptive model rules from data streams, in: H. Blockeel, K. Kersting, S. Nijssen, F. Železný (Eds.), Machine Learning and Knowledge Discovery in Databases, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 480–492. doi:10.1007/978-3-642-40988-2_31.

[45] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '00, ACM, New York, NY, USA, 2000, pp. 71–80. doi:10.1145/347090.347107.

[46] J. Gama, R. Sebastião, P. P. Rodrigues, On evaluating stream learning algorithms, Machine Learning 90 (3) (2013) 317–346. doi:10.1007/s10994-012-5320-9.

[47] J. Duarte, J. Gama, A. Bifet, Adaptive model rules from high-speed data streams, ACM Transactions on Knowledge Discovery from Data 10 (3) (2016) 30:1–30:22. doi:10.1145/2829955.

## Appendix A. Double-Pass Self Parameter Tuning

**Algorithm 2:** Double-Pass Self Parameter Tuning (`DSPT`) [7]: Reflection, expansion, contraction and shrinkage

```
input: B; G; W; C, E; M; R; S
begin
    M = (B + G)/2
    R = 2M − W
    if f(R) < f(G) then
        if f(B) < f(R) then
            W = R
        else
            E = 2R − M
            if f(E) < f(B) then
                W = E        /* Replace the Worst model by the Expansion model.  */
            else
                W = R        /* Replace the Worst model by the Reflection model. */
    else
        if f(R) < f(W) then
            W = R
            /* Replace the Worst model by the Reflection model.                  */
        else
            C = (W + M)/2
            if f(C) < f(W) then
                W = C        /* Replace the Worst model by the Contraction model. */
            else
                S = (B + W)/2
                if f(S) < f(W) then
                    W = S    /* Replace the Worst model by the Shrinkage model.  */
                if f(M) < f(G) then
                    G = M    /* Replace the Good model by the Midpoint model.    */
    /* All configurations are evaluated with an additional pass over the last data
       sample.                                                                    */
```

## Appendix B. Description of Data Sets

For each of the three learning tasks, we chose six benchmark data sets. Table 3 lists all data sets employed. The selection criteria included temporal information (examples with time stamps), size (large enough to be processed as a stream) and availability (publicly available). This section describes their content.

### *Appendix B.1. Recommendation*

The recommendation experiments use CiaoDVD, FilmTrust, GoodBooks, Jester, MovieLens 100k (ML100k) and MovieLens 1M (ML1M).

CiaoDVD[3] holds information about 17 615 users and 16 121 movies, including 72 665 user ratings together with time stamps. These user ratings were collected between the 1st November 2013 and the 31st December 2013. FilmTrust[4] stores information about 1508 users and 2071 movies, including 35 497 user ratings. These user ratings were collected during July 2011. The GoodBooks[5] holds information about 53 424 users and 10 000 books, including 961 756 ratings. The Jester dataset[6] contains 100 jokes from 73 421 users, collected between 1st April 1999 and 1st May 2003. ML100k[7] contains information about 943 users and 1682 movies, including 100 000 user ratings together with time stamps. These user ratings were collected between the 19th September 1997 and the 22nd April 1998. ML1M[8]www.overleaf.com/project/5bdc3069e169aa5e1c518908 contains information about 3706 users and 6040 movies, including 1 000 209 user ratings together with time stamps. These user ratings were collected between the 26th April 2000 and the 28nd February 2003.

[3] www.librec.net/datasets/CiaoDVD.zip
[4] www.librec.net/datasets/filmtrust.zip
[5] github.com/zygmuntz/goodbooks-10k
[6] eigentaste.berkeley.edu/dataset/
[7] files.grouplens.org/datasets/movielens/ml-100k.zip
[8] files.grouplens.org/datasets/movielens/ml-1m.zip

*Appendix B.2. Regression*

The regression experiments were performed with 2DPlanes, Ailerons, Fried, Transcoding, Sgemm and YearPredictionMSD. The 2DPlanes[9] data set holds 40 768 instances and 11 features; Ailerons[10] contains 13 750 instances and 41 features; Fried[11] contains 40 768 instances and 11 features; Transcoding[12] has 168 286 instances and also 11 features; Sgemm[13] contains 241 600 instances and 18 features; and YearPredictionMSD[14] has 515 345 instances and 90 features.

*Appendix B.3. Classification*

The classification experiments used Agrawal, Bank Marketing, Electricity, Postures, SEA and Tweet500. The Agrawal data set[15] contains 1 000 000 instances and 9 attributes; the Electricity[16] contains 45 312 instances and 8 attributes; the Postures[17] contains 78 095 instances and 38 attributes; Bank Marketing[18] holds 45 211 instances and 17 attributes; the SEA[19] contains 60 000 instances, 3 attributes and four concept drifts separated by 15 000 examples; and Tweet500[20] holds 1 000 000 instances and 500 attributes.

[9] github.com/renatopp/arff-datasets/blob/master/regression/2dplanes.arff
[10] github.com/renatopp/arff-datasets/blob/master/regression/ailerons.arff
[11] github.com/renatopp/arff-datasets/blob/master/regression/fried.arff
[12] archive.ics.uci.edu/ml/datasets/Online+Video+Characteristics+and+Transcoding+Time+Dataset
[13] archive.ics.uci.edu/ml/datasets/SGEMM+GPU+kernel+performance
[14] archive.ics.uci.edu/ml/datasets/YearPredictionMSD
[15] github.com/marouabahri/CS-ARF
[16] datahub.io/machine-learning/electricity#resource-electricity_arff
[17] archive.ics.uci.edu/ml/datasets/MoCap+Hand+Postures
[18] archive.ics.uci.edu/ml/datasets/Bank+Marketing
[19] www.liaad.up.pt/kdus/products/datasets-for-concept-drift
[20] github.com/marouabahri/CS-ARF