Assignment 2: Automated Machine Learning for Data Streams *

Emanuel Tomé †, Maria Ferreira ‡

DCC-FCUP, July 2021

1 Introduction

Recently, considerable research development has benefited Automated Machine Learning (AutoML) approaches. Developing machine learning models with consistent high prediction accuracy is a challenging undertaking that almost always need the help of a specialist. As the number of machine learning algorithms and hyperparameters grows, the number of configurations grows as well, so non-experts may obtain adequate outcomes with AutoML tools and approaches, while professionals can automate and optimise their work.

Despite their growing popularity and advancements, existing AutoML technologies are not suitable for data stream mining, since at the start and during the training process, all of the training data must be available and the data used for prediction is expected to have the same distribution as the training data and that it does not vary over time [9].

In this work, we start with a general review of the AutoML approaches, presenting the most important branches of AutoML: Meta-learning, Neural Architecture Search and Hyper-parameter Optimisation. Then we present a review on the the existing tools and frameworks for AutoML. The discussion is then continued with the detailed description of four papers in the scope of application of AutoML for data streams. The first two present two strategies for hyperparameter tuning of machine learning models in data streams. The third one present different approaches in order to adapt existing AutoML tools and used then for tuning in an automated way machine learning pipelines of data streams. The fourth paper presents an approach that automatically generates alarm reports with a summary of the events and features that are important to it. Finally, the open challenges in the hot research topic of AutoML are presented and discussed.

2 AutoML approaches: review

The first question that a newcomer to the field may ask is "what is automated machine learning (AutoML)?". AutoML can be defined as the process of automating the end-to-end process of applying Machine Learning (ML) to real-world problems. Although ML is very successful, its success crucially rely on human ML experts, who pre-process the data, select appropriate ML family and architectures, choose and optimise their hyper-parameters, post-process the ML model and critically analyse the obtained results (Figure 1). With the rapid growth of ML applications, there is an increase demand for off-the-shelf ML methods that can be used easily and without expert knowledge [1].

Moreover, the demand for ML experts has outpaced the supply. Indeed, "it has been acknowledge that data scientists can not scale and it is almost impossible to balance between the number of qualified data scientists and the required effort to manually analyse the increasingly growing size of available data" [5]. Therefore, AutoML software can be used for automating a large part of the ML workflow,

*This work was submitted on the framework of the course Data Stream Mining

†Emanuel Tomé is a student at the Faculty of Sciences of the University of Porto. Currently, he is enrolled in the 1^{st} year of the Master's in Data Science (e-mail: up200702634@edu.fc.up.pt).

‡Maria Ferreira is a student at the Faculty of Sciences of the University of Porto. Currently, she is enrolled in the 1^{st} year of the Master's in Data Science (e-mail: up202004113@edu.fc.up.pt).

which includes automatic training and tuning of many models within a user-specified time-limit. This need resulted in the current hot research area that is AutoML.

However, it is also important to define what is not AutoML. AutoML is not automated data science and it will not replace Data Scientists. All the methods of automated machine learning are developed to support data scientist, not to replace them. Indeed, AutoML aims to free data scientists from the burden of repetitive and time-consuming tasks (such as ML pipeline design and hyper-parameter optimisation) so they can better spend their time in tasks that are much more difficult to automate. In a ideal world, AutoML tools may be applied in several steps of a ML pipeline as depicted in Figure 1.

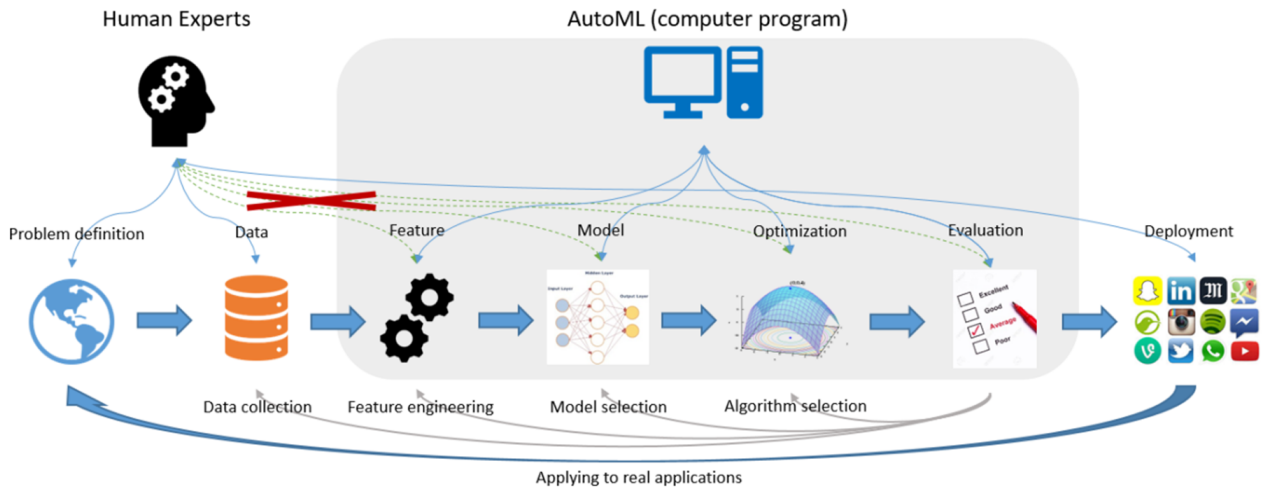


Figure 1: Machine learning pipeline [18].

The AutoML tools can be divided in three main branches: Meta-learning, Neural Architecture Search (NAS) and Hyper-Parameter Optimisation (HPO). These branches are discussed in detail in the following subsections. Other two branches with less developments as the three referred before are Automated data clean and automated feature engineering. These are mainly related with AutoML in the pipeline steps of data pre-processing, while the others are more related with the step related with the tuning of the ML model.

2.1 Meta-Learning

Meta-learning can be defined as learning how to learn [13]. It is the science of observing how different ML approaches perform on a wide range of learning tasks, and the learning from this experience to learn new tasks much faster [13]. Meta-learning is inspired by the way humans learn and by the fact they do not start from scratch every time but use their previous learning experience to learn new tasks [9]. Meta-learning tries to achieve the same for ML tasks.

The meta-learning techniques are usually categorised into three groups [5, 13]: *learning based on task properties, learning from previous model evaluations* and *learning from already pre-trained models* (Figure 2).

Learning based on model evaluation. In this branch of meta-learning, the problem can be defined as follows: "given a set of ML tasks $t_j \in T$, their corresponding learned models along their hyper parameters $\theta in \Theta$ and $P_{i,j} = P(\theta_i, t_j)$, the problem is to learn a meta-learned L that is trained on meta-data $\mathbf{P} \cup \mathbf{P}_{new}$ to predict recommended configuration Θ^*_{new} for a new task t_{new}, where T is the set of all prior machine learning tasks. Θ is the configuration space (hyper-parameter setting, pipeline components, network architecture, and network hyper-parameter), Θ_{new} is the configuration space for a new machine learning task t_{new}, $\mathbf{P}$ is the set of all prior evaluations $P_{i,j}$ of configuration θ_i on a prior task t_j, and $\mathbf{P}_{new}$ is a set of evaluations $P_{i,new}$ for a new task t_{new}" [5]. A way of getting hyper-parameter recomendationf for a new task t_{new} is to recommend based on similar prior tasks t_j. Three different ways can be considered:

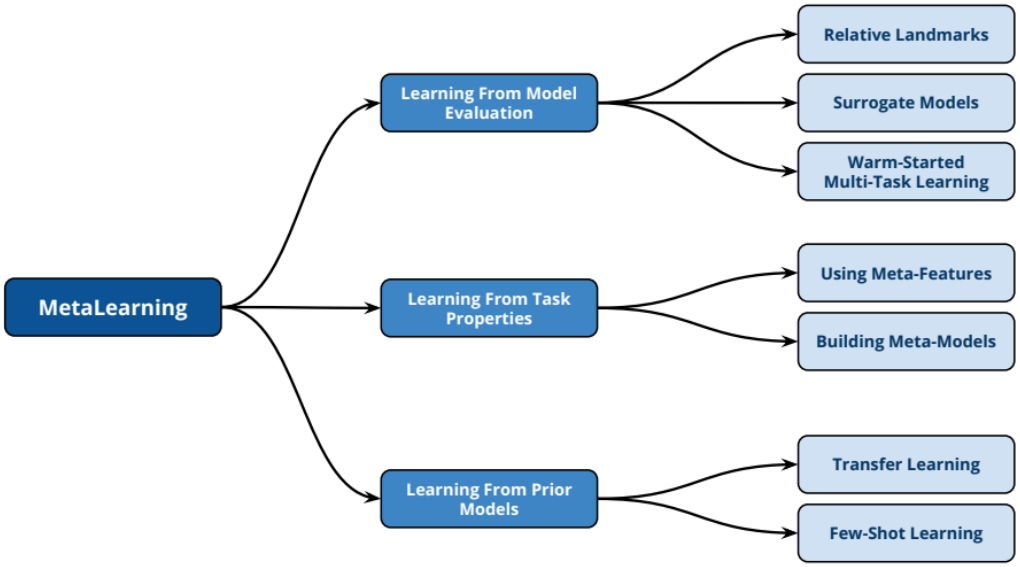


Figure 2: A Taxonomy of Meta-Learning Techniques [5].

- One way to measure the similarity between t_{new} and t_j is using the relative landmarks that measures the performance difference between two model configurations on the same task. Two tasks t_{new} and t_j are considered similar if their relative landmarks performance of the considered configurations are also similar.

- Surrogate models get trained on all prior evaluations of for all prior tasks t_j. Simply, for a particular task t_j, if the surrogate model can predict accurate configuration for a new task t_{new}, then tasks t_{new} and t_j are considered similar

- Warm-Started Multi-task Learning is another way to capture similarity between t_j and t_{new}. Warm-Started Multitask Learning uses the set of prior evaluations P to learn a joint task representation which is used to train surrogate models on prior tasks t_j and integrate them to a feed-forward neural network to learn a joint task representation that can predict accurately a set of evaluation t_{new}.

Learning based on task proprieties. This branch of meta-learning can be divided in the following groups:

- Meta-features: Each prior task is characterised by a feature vector $m(t_j)$ of k features. Then, information from a prior task t_j can be transferred to a new task t_{new} based on their similarity, where their similarity can be determined using a distance between their feature vectors [5].

- meta-models: In this process, the aim is to build a meta model L that learns complete relationships between meta features of prior tasks t_j. For a new task t_{new}, given the meta features for task t_{new}, model L is used to **recommend the best configurations**. Meta-models can also be used to rank a particular set of configurations by using the K-nearest neighbour model on the meta features of prior tasks and predicting the top k tasks that are similar to new task t_{new} and then ranking the best set of configurations of these similar tasks [5].

Learning from prior models can be done using Transfer Learning or Few-Shot Learning:

- Transfer Learning: Consists in the process of utilisation of pre-trained models on prior tasks t_j to be adapted on a new task t_{new}, where tasks t_j and t_{new} are similar. Transfer learning usually works well when the new task to be learned is similar to the prior tasks, otherwise transfer learning may lead to unsatisfactory results.[5]

- Few-Shot Learning: Assumes that a model is required to be trained using a few training instances given the prior experience gained from already trained models on similar tasks. Learning a common feature representation for different tasks can be used to take the advantage of utilising pre-trained models that can initially guide the model parameters optimisation for the few instances available.

2.2 Neural Architecture Search

Neural Architecture Search (NAS) is a fundamental step in automating the machine learning process and has been successfully used to design the model architecture for image and language tasks.

First, NAS specifies the search space, then uses the search strategy to discover the candidate network structure. Additionally, evaluates it and then performs the next round of search depending on the feedback [8].

The five primary types of NAS methods are as follows: *random search, reinforcement learning, gradient-based methods, evolutionary methods and Bayesian optimisation.*

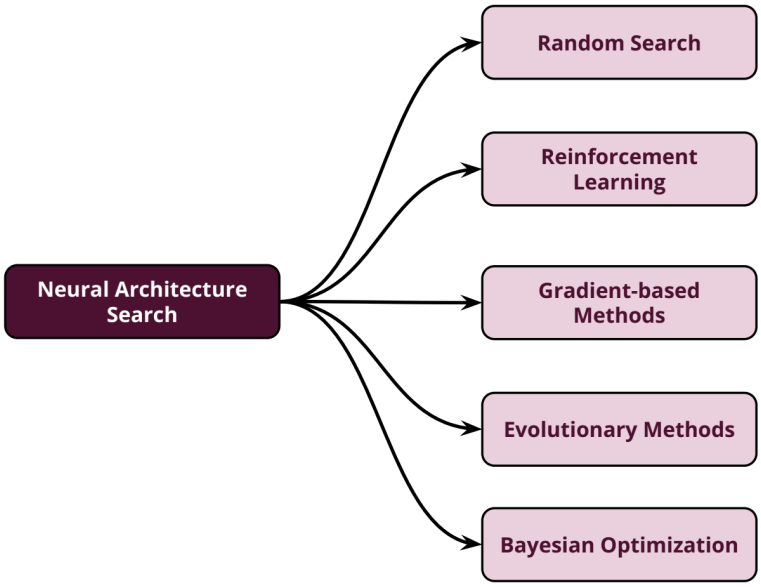


Figure 3: A Taxonomy of Neural Architecture Search [5].

- Random Search: One of the most naive and simplest approaches for network architecture search.
- Reinforcement Learning: Technique that aims to find the best network architecture.
- Gradient-Based Methods: a gradient-based neural architecture search method can avoid thetrial-and-error cost of network modeling, and is a strong migration method suitable for multi-domain and multi-task [19]. The Gradient Descent method aims to find the objective function minimisation. It uses the gradient information and iteratively adjusts the parameters to find the appropriate target value.
- Evolutionary Methods: Are well suited for optimising arbitrary structure. To improve the neural network architecture and weights, several evolutionary techniques based on genetic algorithms are utilised, while others depend on hierarchical evolution [19].
- Bayesian Optimization: Method based on Gaussian processes, may outperform evolutionary algorithms in some problems [19].

2.3 Hyper-parameter Optimisation

AutoML in data streams context strongly relies on the problem of choosing a set of optimal hyper-parameters for a learning algorithm.

Techniques for hyperparameter optimization can be divided in categories based in what kind of search space can the optimizer operate on and what kind of feedbacks it needs [18]. Thus, automated hyper-parameter tuning approaches may also be divided into two groups in principle: multi-fidelity optimisation techniques and black-box optimisation techniques.

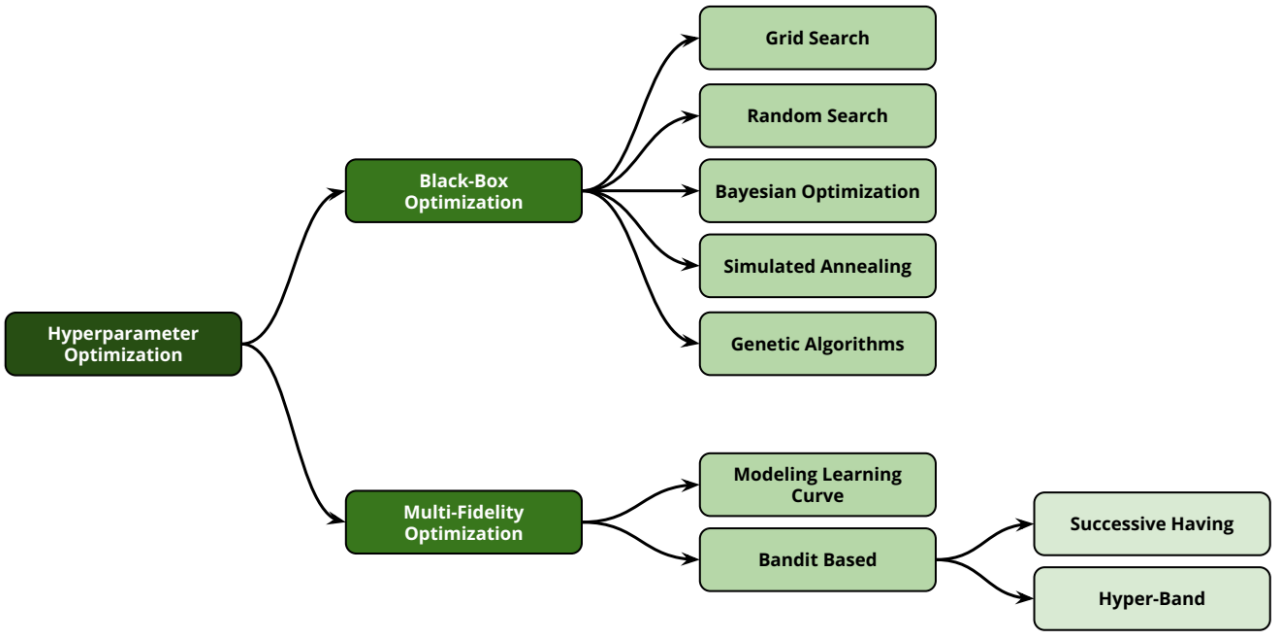


Figure 4: A Taxonomy of Hyper-parameter Optimisation [5].

Black-Box Optimisation

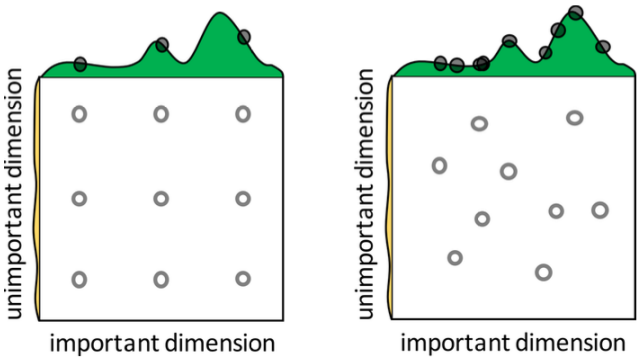
Simple search techniques do not make assumptions about the search space, so each configuration in the search space can be evaluated independently. Grid search and random search are two common approaches.

Grid search is one of the most commonly-used methods to explore hyper-parameter configuration space. To get the optimal hyper-parameter setting, grid search have to enumerate every possible configurations in the search space. This method is easily implemented and paralleled, however, it's been shown inefficient for high-dimensional hyper-parameter configuration space. If there are several parameters to tune and a large data set, it's not ideal to use grid search since it will take a lot of time and computer capacity [16].

Random search overcomes certain limitations of grid search. It randomly selects a pre-defined number of samples between the upper and lower bounds as candidate hyper-parameter values and train them [16]. If the configuration space is large enough, then the global optimums (or approximations) can be detected. Random search samples a fixed number of parameter combinations from the specified distribution, which improves system efficiency by reducing the probability of wasting much time on a small poor-performing area [16] [8].

In both cases of simple search approaches, every evaluation in their iterations is independent of previous evaluations, not exploiting the knowledge gained from the past evaluations. It means that they waste time evaluating poorly-performing areas of the search space, becoming in most cases inefficient [16].

Bayesian optimization is a type of model-based derivative-free optimization. These techniques build a model based on visited samples, using feedback from evaluator to helps it generate more optimistic samples [8]. Bayesian optimization builds a probabilistic model (Gaussian, tree-based model, deep network or other) in order to map the configurations to their performance with uncertainty. Then, to balance exploration and exploitation during search, it develops an acquisition function based



(a) Grid search. (b) Random search.

Figure 5: Illustration of grid and random search [16].

on the probabilistic model, such as predicted improvement and upper confidence boundaries. By optimizing the acquisition function, a new sample is obtained at each iteration, which is then utilized to update the probabilistic model after it has been assessed [16].

Gradient-based optimization are based in gradient descent. For specific machine learning algorithms, the gradient of certain hyper-parameters can be calculated to identify the promising direction, and then the gradient descent can be used to optimize these hyper-parameters [8]. Compared with previews methods, gradients offer the most accurate information where better configurations locates, also shows a faster convergence speed reach local optimum [17]. The gradients can only be used in some cases where it is possible to obtain the gradient of hyper-parameters.

Evolutionary Algorithm is part of the heuristic search, where biologic behaviours and events are frequently used as inspiration for methods [17]. Each iteration generates a new population based on the previous one, and the fitness (performances) of the individuals are measured. The key concept of heuristic search is how to keep the population updated. The generation step of evolutionary algorithms contains crossover and mutation. Crossover involves two different individuals (ancestors) from the last generation. It combines them in some way to generate an new individual. Better individuals will be more likely to survive and have more capable offspring, while the worse individuals will gradually disappear. After several generations, the individual with the best adaptability will be identified as the global optimum [8].

Multi-fidelity optimization is an optimization technique which focuses on decreasing the evaluation cost by combining a large number of cheap low-fidelity evaluations and a small number of expensive high-fidelity evaluation. The idea behind the multi-fidelity evaluation is to use many low-fidelity evaluation to reduce the total evaluation cost.

- Modeling Learning Curve: is an optimization approach that models learning curves during hyper-parameter optimization and determines whether to provide extra resources or stop the training operation for a specific configuration. A curve, for example, may represent the performance of a certain hyper-parameter on a growing subset of the dataset.

- Bandit Based (Successive Having and Hyper-band): Optimizes traffic allocation among discrete alternatives by progressively updating traffic allocation depending on each candidate's past performance [10].

Successivehalving: There's a n hyper-parameter combinations, and that they are evaluated with uniformly-allocated budgets (b = B/n).[10] Then, according to the evaluation results for each iteration, half of the poorly-performing hyper-parameter configurations are eliminated, and the better-performing half is passed to the next iteration with double budgets. The process is repeated until the final optimal hyper-parameter combination is detected. This method is directly affected by the trade-off between the number of hyper-parameter configurations and the budgets allocated to each configuration, which proves demands experts to find how to allocate the budget and how to determine whether to test fewer configurations with a higher budget for each or to test more configurations with a lower budget for each.

Hyperband was proposed by [10] in order to choose a reasonable number of configurations for successive halving algorithms, which aims to find a fair trade-off between the number of hyper-parameter configurations and their allocated budgets by dividing the total budgets into n pieces and allocating these pieces to each configuration. The successive halving algorithm serves as a subroutine on each set of random configurations to eliminate the poorly-performing hyper-parameter configurations and improve efficiency.

3 Tools and Frameworks

There are a variety of automated hyper-parameter optimisation frameworks available, each with its own set of advantages and disadvantages when applied to different types of situations.

To problems of hyper-parameter optimisation and model selection, different types of AutoML can be used:

AutoWEKA considers the problem of simultaneously selecting a learning algorithm and defining its hyperparameters, going beyond previous methods that deal with these issues in isolation. Auto-WEKA does this using a fully automated approach, taking advantage of recent innovations in Bayesian optimization. It's used in Java and is targeted primarily at non-experienced users to more effectively identify machine learning algorithms and hyper-parameter settings appropriate for their applications, and thus obtain improved performance.

Auto-sklearn is an extension of AutoWEKA and uses scikit-learn library in Python. It also uses Bayesian optimisation to find good machine learning channels. It improves over AutoWEKA by using meta-learning to increase search efficiency and post-hoc ensemble building to combine the models generated during the hyperparameter optimisation process.

TPOT is a data-science assistant which optimizes machine learning pipelines using genetic programming. The TPOT (Tree-based Pipeline Optimisation Tool) was created to be a science-time assistant, and is a machine learning tool in Python that optimises pipeline machine learning using genetic programming. It will automate the most tedious part of machine learning by intelligently exploring thousands of possible pipelines to find the best one for the selected data. Figure 5 shows the part of the process that is automated by the TPOT approach.

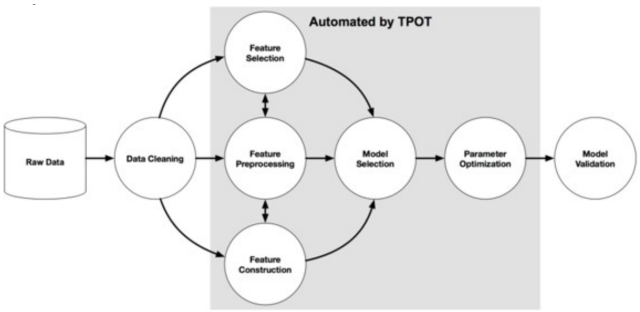


Figure 6: Automation by TPOT framework.

In addition to automating the parameter optimisation process, TPOT also automates feature selection, feature preprocessing, feature construction and model selection. After TPOT finishes searching, it provides the Python code for the best pipeline found, so it's possible to perform with the pipeline from there.

H2O AutoML provides automated model selection and assembling for the H2O machine learning and data analytics platform. It offers a platform that democratises artificial intelligence, capable of empowering every employee, customer and citizen with sophisticated artificial intelligence technology and easy-to-use artificial intelligence applications.

The *mlr* is a R package that contains several hyper-parameter optimisation techniques for machine learning problems. It also provides supervised methods such as classification, regression, and survival analysis, along with their corresponding evaluation and optimisation methods, as well as unsupervised methods. The package is written in such a way that the user can extend or deviate from the imple-

mented convenience methods and build their own experiments or complex algorithms. Figure 6 shows all the functions of the mlr package, through its categories (stable, maturing, experimental, planned).

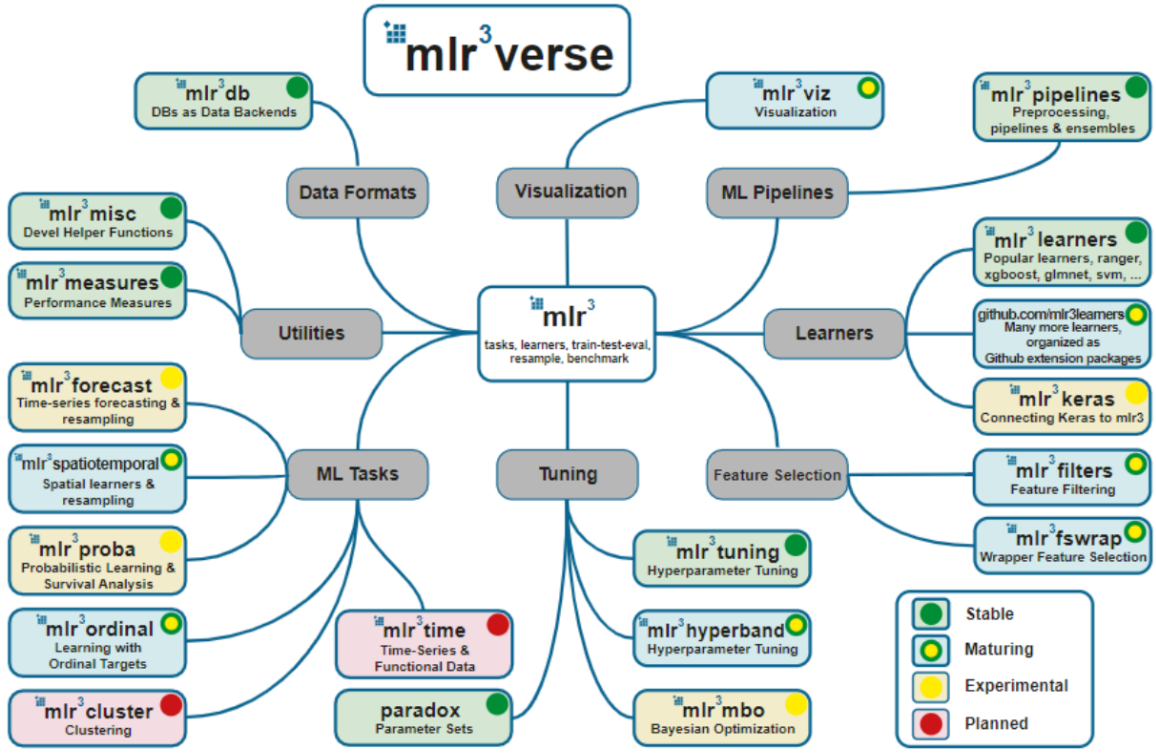


Figure 7: mlr package in R.

There's a few AutoML instances that perform with deep neural network architecture search:

The *Google Cloud AutoML* is an could-based machine learning service which so far provides the automated generation of computer vision pipelines and allow users to submit ML model training codebases along with hyper-parameters to be tuned. There are some products offered by Google regarding AutoML, like Vertex AI, to help build, deploy and scale more AI models; AutoML Vision, Derive insights from object detection and image classification, in the cloud or at the edge; AutoML Video Intelligence (beta), Enable powerful content discovery and engaging video; AutoML Tables (beta), that automatically build and deploy state-of-the-art machine learning models on structured data.

Auto Keras is an open-source python package for neural architecture search. Is an implementation of AutoML for deep learning models using the Keras API, specifically the TensorFlow 2, tf.keras API. This implementation uses a process of searching through neural network architectures to best address a modeling task, referred to more generally as Neural Architecture Search. It also provides an easy-to-use interface for different tasks (such as image classification, structured data classification or regression). It's only required to specify the location of the data and the number of models to try and is returned a model that achieves the best performance (under the configured constraints) on that dataset.

4 AutoML in Data Streams

In this section are summarised four papers in the scope of application of AutoML for data streams. The first two present two strategies for hyperparameter tuning of machine learning models in data streams. The third one present different approaches in order to adapt existing AutoML tools and used then for tuning in an automated way machine learning pipelines of data streams. The fourth paper presents an approach that automatically generates alarm reports with a summary of the events and features that are important to it. Naturally, other papers could have been considered. However, we opted for these ones since they represent different perspectives of AutoML for data streams (hyperparameter tuning of machine learning models in data streams, pipeline tuning and adaptation of

existing AutoML tools for data streams, and automatic generation of reports).

4.1 Self Hyper-Parameter Tuning for Data Streams

The paper [14] proposes and evaluates a new parameter tuning algorithm called Self-Parameter Tuning (SPT), for data streams approaches, which consists of an adaptation of the Nelder-Mead optimization algorithm for hyper-parameter tuning. It presents a method to evaluate a given solution using a dynamic data sample and concept drift detection to tune and find good parameter configuration that aims to minimize the objective function.

The authors' proposition for Self-Parameter Tuning (STP) consists of the use of the Nelder-Mead algorithm as a direct-search algorithm to find optimal solutions on a search space, with the objective to dynamic size data stream samples, continuously searching for the optimal hyper-parameters. The major contribution of this work is adapting Nelder-Mead to data streams.

The Nelder-Mead Optimization Algorithm process dynamically, using a previously saved copy of the models, each data stream sample until the input models converge. Each model has a Nelder-Mead algorithm vertex. The algorithm is computed in parallel to minimize the time response. The initial model vertexes are selected randomly, while the Nelder-Mead operator is applied at dynamic intervals.

The algorithm uses a simplex for the search in multidimensional unconstrained optimization without derivatives. The shape of the simplex, that are defined by k vertexes (settings), are iteratively updated in order to sequentially discard the vertex associated with the largest cost function value. It means that k = 1 + number of parameters that it wants to minimize (if simplex has 3 points, it optimize 2 parameters). Then, it will evaluate each of the three settings according to the root mean square error (RMSE) value: best (B), good (G), which is the closest to the best vertex, and worst (W). M is a midvertex (auxiliary model). In addition, the algorithm relies in the following operations: reflection, shrinkage, contraction and expansion. In figure [X], it's possible to identify the operations of the Nelder-Mead algorithm, where each bullet represents a model containing a set of hyper-parameters.

Algorithm 1 Nelder-Mead - reflect (a) or expand operators (d).

```
1: M = (B + G)/2
2: R = 2M − W
3: if f(R) < f(G) then
4:     if f(B) < f(R) then
5:         W = R
6:     else
7:         E = 2R − M
8:         if f(E) < f(B) then
9:             W = E
10:        else
11:            W = R
12:        end if
13:    end if
14: end if
```

Figure 8: Pseudo-code of Nelder-Mead reflect (a) or expand operators (d). [14].

Figure 7 shows the pseudo-code for the operations reflect (a), expand (d), contract (c) and shrink (d). It's necessary to compute an additional set of vertexes (midpoint (M), reflection (R), expansion (E), contraction (C) and shrinkage (S)) for each Nelder-Mead operation and then check if the calculated vertexes belong to the search space. Algorithm 1 computes the midpoint (M) of the best face of the shape as well as the reflection point (R). Based on a set of established arguments, it chooses whether to reflect or expand.

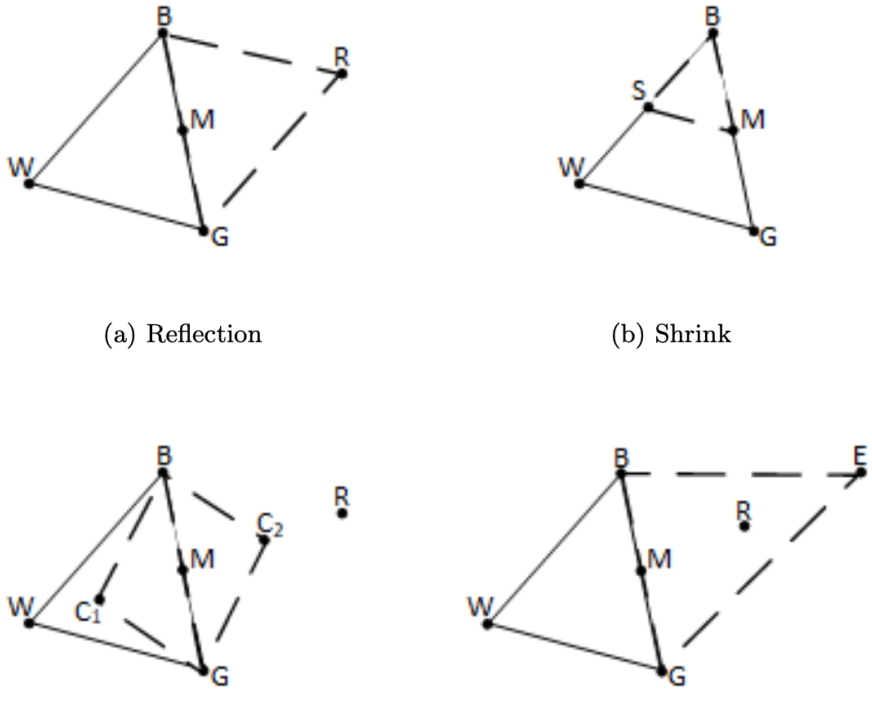
The contraction point (C) of the shape's worst face – the midpoint between the worst vertex (W) and the midpoint M – and the shrinking point (S) – the midway between the best (B) and worst (W) vertexes – are calculated using Algorithm 2. Then, using a set of established arguments, it decides whether to contract or shrink.

Parallel processing is used for adapting Nelder-Mead algorithms to data streams. The main thread launches the n + 1 model threads and starts a continuous event processing loop. This loop distributes

Algorithm 2 Nelder-Mead - contract (c) or shrink (b) operators.

```
1: M = (B + G)/2
2: R = 2M − W
3: if f(R) ≥ f(G) then
4:     if f(R) < f(W) then
5:         W = R
6:     else
7:         C = (W + M)/2
8:         if f(C) < f(W) then
9:             W = C
10:        else
11:            S = (B + W)/2
12:            if f(S) < f(W) then
13:                W = S
14:            end if
15:            if f(M) < f(G) then
16:                G = M
17:            end if
18:        end if
19:    end if
20: end if
```

Figure 9: Pseudo-code of Nelder-Mead - contract (c) or shrink (b) operators. [14].



(a) Reflection

(b) Shrink

(c) Contraction

(d) Expansion

Figure 10: Nelder-Mead Operators. [14].

events to the model threads and calculates the new sample size after it exceeds the sample size interval specified by the model threads.

The models' evaluation process consists in ordering models n+1 by RMSE and substituting the worst model by applying the Nelder-Mead algorithm. The algorithm creates a thread per Nelder-Mead operator, where each operator generates a new model and calculates the cumulative RMSE using the instances of the last sample size interval. The Nelder-Mead operator thread model with the lowest RMSE replaces the worst model.

Method: The model was implemented using MOA and rules regression technique was chosen. The algorithm tried to optimize the split confidence, learning rate, and learning rate decay, since these factors have greatest influence on the method output. Three datasets were considered.

The data is divided into two halves: 50 % for train and 50 % for test. The holdout algorithm is used to find the optimal solution for the selected parameters. The model is then built using the data collected by the train and the optimal hyper-parameters, then it is subjected to a holdout algorithm. The prequential protocol uses the whole data for training and testing. In the case of the decision model, the algorithm updates it after each prediction. To determine the average and standard deviation of the assessment metrics, the holdout and prequential tests were repeated 30 times.

The hyper-parameter optimization algorithm was added to the MOA framework through the definition of a new regression task. When the task is launched, it initializes four identical regression models with randomly selected values for the three hyper-parameters and applies our algorithm.

Evaluation: The authors stated that the evaluation process consists in holdout evaluation, to find an optimal solution for the hyper-parameters and assess the reproducibility of the algorithm with multiple experiments, and prequential evaluation. They also determined the incremental RMSE, which is calculated incrementally after each new instance. After verifying the convergence of the models, the holdout evaluation assessed the performance of the regression algorithm with both the baseline (B) and the SPT hyper-parameters. This process was repeated 30 times with randomly shuffled data variations to compute the average and standard deviation of the prediction RMSE, that shows decrease of 0.3% with the Twitter data set, drops 4.3% with the YearPredictionMSD data set and reduction of 34.4% with the SGEMM GPU data set.

Result: Figure 10 shows the convergence of the optimization of the three hyper-parameters with the data sets. Results shows that on Twitter data set, all models quickly converge and on SGEMM GPU data set, only three of the four models converge. Significance tests like Wilcoxon Test, McNemar and the critical distance (CD) measure (using Nemenyi test) were applied in order to detect the statistical differences between STP and the baseline approaches. The results showed that SPT and baseline are statistically different, therefore the null hypothesis is rejected for all data sets. The critical distance between the proposed and baseline approaches shows that they are statistically different.

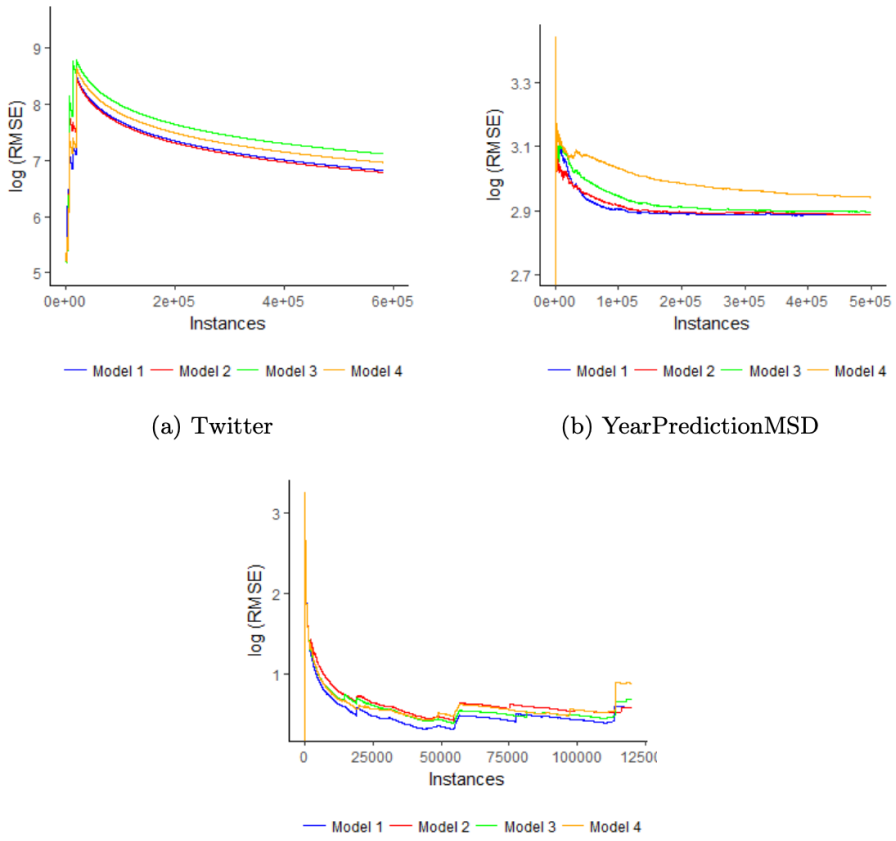
The prequential evaluation results shows that the hyper-parameters found by SPT were not the best for all stream instances. The algorithm is framed responsive to concept drift and received feedback from the Page-Hinkley test, which detects data changes. Whenever a drift occurs, the optimization algorithm reinitiates the search for new hyper-parameters. Then, the reaction of the algorithm is applied to the data changes and improve the initial results.

4.2 Hyperparameter Self-Tuning for Data Streams

The Single-pass Self Parameter Tuning (SSPT) method, proposed in the paper [15], is a modified version of the Double-pass Self Parameter Tuning (DSPT) technique that is used to optimize a number of hyperparameters in vast search spaces. To choose the optimum hyperparameters, both the SSPT and DSPT versions adapt the Nelder–Mead technique to operate with data streams and use a heuristics-based direct-search algorithm.

SPT has been updated to include a single-pass method (SSPT) that can optimize several continuous or discrete hyperparameters. This solution is not only relevant to recommendation, regression, and classification issues, but so far it is also the only one that successfully works with data streams and responds to concept drifts in near real time.

All SPT versions have two operation phases: exploration and deployment. In the exploration stage, the goal is to find a local minimum by exploring the multiple candidate models created by the



(a) Twitter

(b) YearPredictionMSD

(c) SGEMM GPU

Figure 11: Convergence of Regression Model. [14].

Nelder–Mead operators. In deployment process, the algorithm uses the best learning model obtained in the exploration stage, continuously updating and monitoring it for concept drifts. Concept drifts are detected with the Drift Detection Method (DDM) proposed by [7]. Whenever DDM detects a concept drift, SPT triggers a new exploration phase to adjust the hyperparameters according to new data, allowing the hyperparameters to be adjusted based on fresh data.

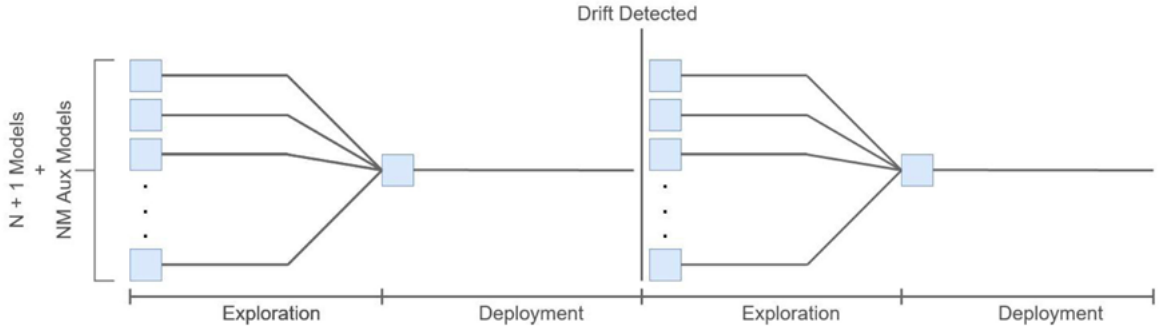


Figure 12: SPT Operation [15].

The SSPT can be summarized as follows:

1. Create $n + 1$ learning models and train them.
2. Choose the Best, the Good and the Worst models from the $n + 1$ previously trained models.
3. Create 7 experimental new models using the Nelder-Mead operators
4. Train the 10 models (the best, the good, the worst and the seven models created by the Nelder-Mead operators)

5. Compute the window size S, $S = \frac{16\sigma^2}{M^2}$, where σ represents the error standard deviation and M the confidence level.

6. Repeat steps 2-5 until the convergence criteria is met.

7. Deploy the best model.

8. Use the Drift Detection Method (DDM) to react to concept drift. Whenever DDM detects a concept drift, we should go back to step 1. Note that the SSPT is an event-driven algorithm that continuously updates the current learning model.

The paper addresses the following questions: which SPT version produces the best results, what are the advantages and disadvantages of SPT over other hyperparameter tuning methods and what is SPT's reaction to drift. SPT was evaluated on three stream-based learning tasks: Recommendation, Regression and Classification.

The goal of recommendation experiments is to optimise the learning rate and regularisation hyperparameters. The algorithm chosen was BRISMF. Classification tasks aims to optimise the grace period and tie-threshold hyperparameters and the algorithm chosen was VFDT classification algorithm. In regression experiments, the main goal was optimise the split confidence and tie-threshold hyperparameters, using the algorithm AMRules regression.

Publicly benchmark data sets that were selected for each of the three learning tasks, in total of six. The proposed method was evaluated using the predictive sequential (prequential) data stream evaluation protocol.

The evaluation metrics adopted to tune the algorithms were the Root Mean Squared Error (RMSE) for recommendation and regression tasks and the loss function (error-rate) for classification.

A comparison was made between the SPT variants in order to find which one performs better, where the results show that SSPT is generally better than DSPT. Regardless of the version, SPT uses more memory than the standalone algorithm, since it requires enough memory to hold ten models during exploration phase.

TASK: Recommendation	Default	DSPT	SSPT	WSPT
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	Default	DSPT	SSPT	WSPT
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	Default	DSPT	SSPT	WSPT
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

Figure 13: Comparison Against Competing Techniques [15].

To measure how SPT compare against other hyperparameter tuning methods, the methods chosen to be compared to were random search and grid search. Results show that SPT is faster and requires less memory than the other methods.

In recommendation task, all SPT variants rank better than the default configuration for all data sets, also improving over grid search and random search in all data sets. Regardless of the data set, it is consistently the fastest optimisation algorithm.

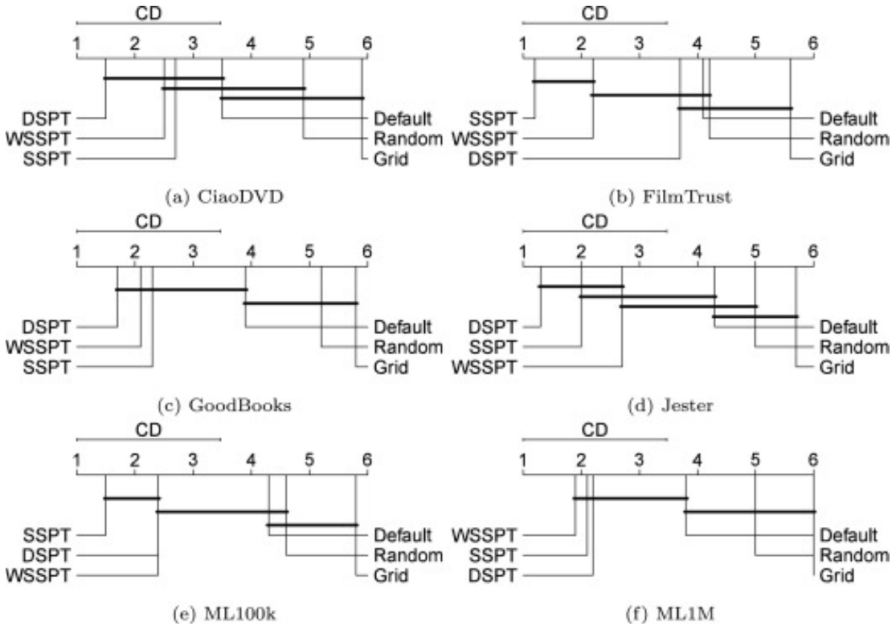


Figure 14: Recommendation – Performance Critical Distance (lower is better) of Default Parameter Initialisation, Grid Search, and Random Search, DSPT, SSPT and WSSPT. [15].

The regression task shows that DSPT present better performance, on average, than SSPT. In addition, SSPT shows better results regarding accuracy and time required to converge to a local optimal solution than DSPT. It's possible to conclude that DSPT is the variant with better rank.

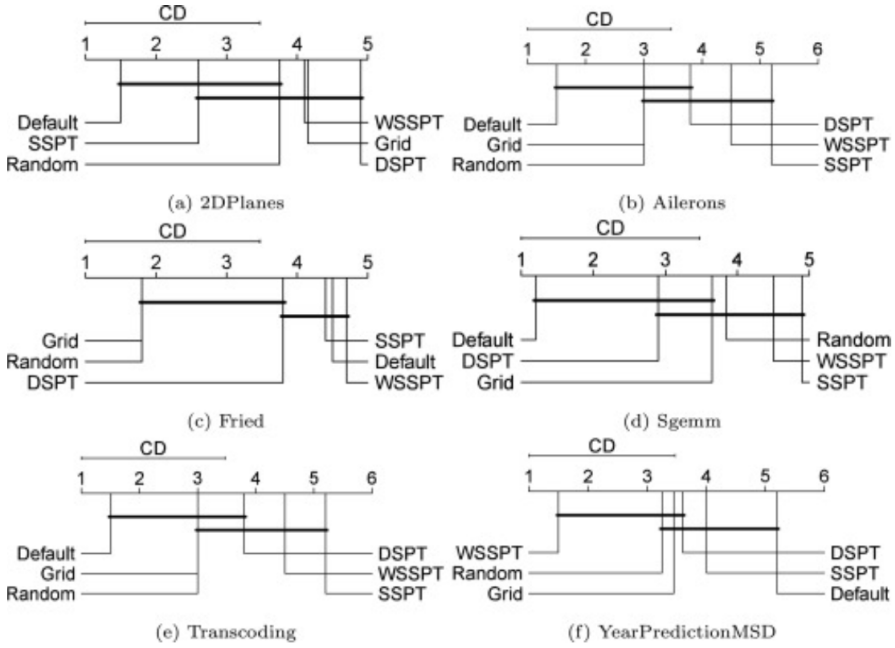


Figure 15: Regression – Performance Critical Distance (lower is better) of Default Initialisation, Grid Search, Random Search, DSPT, SSPT and WSSPT [15].

In classification task, SSPT does not show significantly difference from DSPT. SSPT also shows

better results regarding accuracy and time required to converge to a local optimal solution than DSPT. In addition, SSPT is only outperformed by the grid and random search.

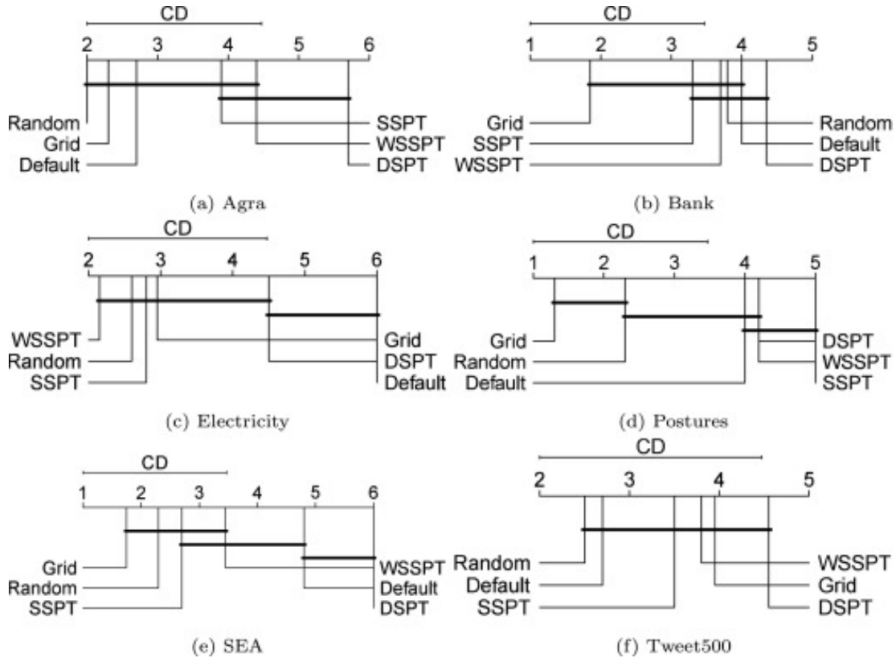


Figure 16: Classification – Performance Critical Distance (lower is better) of Default Initialisation, Grid Search, Random Search, DSPT, SSPT, and WSSPT [15].

In conclusion, 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. The proposed SSPT has the advantage of working with data streams, applicable to different machine learning tasks and reacts to the variability of the data.

4.3 Adaptation Strategies for Automated Machine Learning on Evolving Data

This paper [4] proposes six different adaptation strategies to cope with concept drift and implement them in open-source AutoML libraries. This paper assumes that data can be buffered on batches using a moving window and that there is limited memory and limited time for making new predictions, being the performance of different adaptation strategies paired with different AutoML methods evaluated accordingly. However, it should be noticed that single-pass strategies are beyond the scope of the paper. The goal of the paper is to redesign the entire ML pipeline and not only adapt a previously chosen model.

The paper proposes and evaluates six different adaptation techniques (see Figure 17). In each of the proposed strategies the AutoML algorithm is run at least once at the beginning with the initial batch and there is a forgetting mechanism, that is, a fixed length sliding window. The evaluation is made by means of data chunk evaluation [3], which is a middle-way approach that combines holdout and prequential evaluation, using data chunks of size s instead of individual instances to apply the test-then-train paradigm. For drift detection, the authors uses the Early Drift Detection Method (EDDM) algorithm [2]. The six adaptation techniques proposed are the following:

- **Detect & Increment (D&I)** - This strategy includes a drift detector that observes changes in pipeline performance and uses this to detect concept drift. AutoML is first run on the initial batch of data to build a first model (Model-A). This strategy limit the AutoML search space to pipelines with incremental learning algorithms, i.e., learners that can continue to train on ew data, such as random forest and gradient boosting. The Model-A is updated only if drift is detected. The configuration of the pipeline remains the same. Therefore, this strategy assumes that the initial pipeline configuration will remain useful and only the models need to be updated in case of concept drift.

- **Detect & Retrain (D&RT)** - This strategy is similar to the previous one, but runs AutoML with restrincti the search space. The original pipeline configurations and ensemble weights are kept. Therefore, this strategy also assumes that the initial pipeline configuration will remain useful and only the models need to be retrained in case of concept drift [4]. This approach eliminates the need of re-running expensive AutoML techniques with every drift, but may be less effective if the drift is so large that a pipeline redesign is need.

- **Detect & Warm-start (D&WS)** - In this strategy when a drift is detected, the AutoML technique is re-run to find a better pipeline configuration. However, instead of starting from scratch, the AutoML is re-run with a "warm start", that is, starting the search from the best earlier evaluated pipeline configurations. This strategy assumes that the initial pipeline configurations are no longer optimal after a concept drift. However, the use of warm start mechanism can lead to faster convergence to good configurations, hence working better under small time budgets. The drawback of this strategy is in case of a sudden drift, the previously best pipelines may be misleading the search.

- **Detect & Restart (D&RS)** - In this strategy after a drift is detected, the AutoML technique is re-run from scratch with a fixed time budget. This strategy also assumes that the pipelines need to be re-tuned after drift. Although re-running the AutoML from scratch is more expensive, it could result in significant performance improvements in the presence of a significant drift.

- **Periodic Restart (PRS)** - This approach is similar to the previous one. However, instead of using a drift detector, AutoML is restarted in a regular interval. This strategy tests whether a drift detector is useful and whether it is worth to retrain at certain intrevals in spite of the significant computational costs.

- **Train once (T1)** - This is a baseline strategy used only for comparison purposes. In this strategy AutoML is run once at the beginning and the resulting model is used to test each upcoming batch.

In order to evaluate the adaptation mechanisms, the authors use four well-known classification data streams and 15 artificial ones with different drift characteristics. The data streams are divided in batches of size 1000 and given to the algorithms in arriving order. For training, a sliding window with a fixed length of three batches is chosen for each dataset. The authors also evaluate adaptations of Autosklearn, Gama and H2O along with baselines Oza Bagging, Blast and Gradient Boosting.

The main conclusions of the paper are that the proposed strategies effectively allow AutoML techniques to recover from concept drift, and rival or outperform popular learning techniques such as Oza Bagging and BLAST. Moreover, the authors also conclude that different drift characteristics affect learning algorithms in different ways, and different adaptation strategies may be needed. They also state that on large datasets, re-optimising pipelines after drift ins detected works generally well. Indeed, if the concept drift is not too large and the pipelines are retrained frequently enough, simple retraining pipelines on the most recent data after drift is detected, without re-optimising the pipelines themselves, also works well.

4.4 Automatic Model Monitoring for Data Streams

This paper [11] proposes an automatic model monitoring system of machine learning models for data streams (SAMM - Streaming system for Automatic Model Monitoring). SAMM detects concept drift using a time and space efficient unsupervised streaming algorithm and it generates alarm reports with a summary of the events and features that are important to explain it. SAMM is able to detects the concept drift in a unsupervised way and it suggests an explanation for the drift in the form of an alarm report. Although there are some methods in the literature that already proposed unsupervised concept drift detection before, the authors claim that, for the best of their knowledge, their system is the first proposal attempting to explain why a drift occurred.

The proposed strategy can be split in three parts:

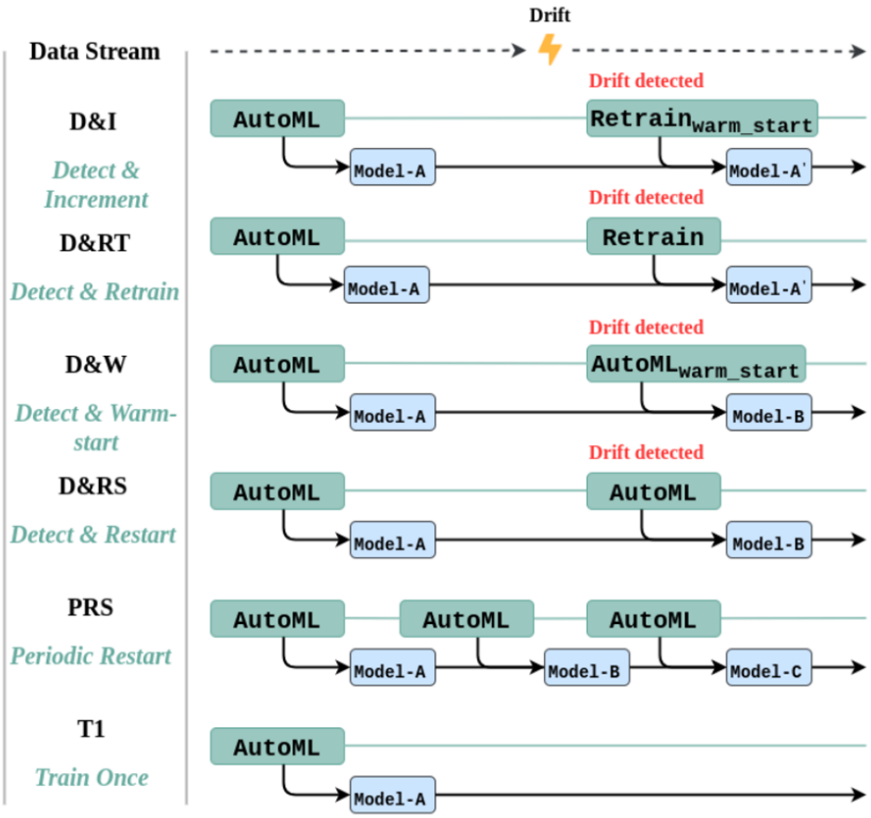


Figure 17: Adaptation strategies [4].

1. **Signal Computation** - The signal value S is computed for each incoming event. The signal consists of a measure of similarity between the model scores histogram in a reference window R and the model scores histograms in a fixed-size target window T. The T window contains the last n_T events collected. R contains events in a reference period, which is prior to the target period and it consists of the n_R events immediately before T (fixed-size window), or of the events in a window with a fixed time duration ending at the oldest event in T (fixed-time window). The authors prefer to used fixed-size windows, though they can apply it to fixed-time windows as well.

   The R window size is chosen to be some multiple of the T window size (e.g. 5 times larger). In the experiments conduced by the authors, they state that their studies showed that 3 and 0.5 times the average number of daily events are good default values for the sizes of R and T.

   The signal S is defined using a measure M of similarity between histograms such that $S = M(R, T)$. The authors present results using the Jensen-Shannon Divergence (JSD), although other measures are possible.

2. **Threshold** - The threshold is defined usinf SPEAR (Streaming Percentiles EstimAtoR), a stochastic approximation method to estimate the cumulative distribution function that can be used to obtain any percentile with a single pass over the data. This approach uses a simple percentiles update strategy anchored on the principle of restoring the invariant (on each new event) that the average count per bin is the same for all bins. The pseudo-code of SPEAR is present in Figure 18. The first 10 lines correspond to the sequential consumption of the stream values as they arrive. The first $n+1$ values that arrive are inserted into the global list P in sorted order, to initialise an estimate of the $n+1$ percentile positions. The percentile position estimates are later updated when more events stream in. The subsequent values are processed on line 10. For each incoming event, the percentiles are updated taking into account the incoming value X and the current total count C. The function UpdatePercentiles is graphically represented in Figure 19. SPEAR works by maintaining the invariant that the estimated number of counts

in each bin is the same for all bins. The new value X will belong to some bin j for each the count will increase by one, which breaks the invariant. To restore the invariant, the function UpdatePercentiles loops over all bins from left to right. Bins that have a deficit count relative to the target variable are expanded to the right (steps 2 to 7 in figure 19). Bins that have an excess count are contracted to the left (equivalent to expand the next bin to the left- steps 8 to 10 in Figure 19). However, this algorithm is asymmetric (operates from left to right), which will create a directional bias in the estimate. To correct it, it should be also applied from left to right and average out the two.

Algorithm 1 SPEAR – Consumes a stream of values in real time.

stream = newStream( )
$\mathbf{P} \leftarrow []$ ▷ Initialise global empty list of approximate percentiles
$C = 0$ ▷ Initialise count variable
while stream.notClosed( ) **do**
  X = stream.getValue( ) ▷ Get last value received
  $C \leftarrow C + 1$ ▷ Update count
  if $C \leq n$ **then** ▷ Initialise $\mathbf{P}$ using first $n + 1$ values
    $\mathbf{P} \leftarrow \mathbf{P}$.InsertSorted($X$)
  else ▷ Update $\mathbf{P}$ otherwise
    $\mathbf{P} \leftarrow$ UpdatePercentiles($\mathbf{P}, X, C$)

function UpdatePercentiles($\mathbf{P}, X, C$)
  $c_{\text{per\_bin}} \leftarrow C/n$ ▷ Counts per bin before update
  $c_{\text{target}} \leftarrow (C + 1)/n$ ▷ Target counts per bin after update

  $c_{\text{this}} \leftarrow c_{\text{per\_bin}}$ ▷ Set count at current bin (bin_1)
  if $X < P_0$ **then** ▷ X to the left of bin_1
    $P_0 \leftarrow X$ ▷ Left-expand bin_1 (move P_0 left)
  if $X < P_1$ **then** ▷ X in bin_1 or to the left
    $c_{\text{this}} \leftarrow c_{\text{this}} + 1$ ▷ Increase count to add X to bin_1

  for $i \leftarrow 1, \ldots, n - 1$ **do** ▷ Move internal bin walls
    $\delta c \leftarrow c_{\text{target}} - c_{\text{this}}$ ▷ Deficit count at bin_i
    if $\delta c > 0$ **then** ▷ bin_i smaller than target
      if $X < P_{i+1}$ **then** ▷ X in bin_{i+1}
        $\rho_{\text{next}} = \frac{1+c_{\text{per\_bin}}}{P_{i+1}-P_i}$ ▷ Computes bin_{i+1}'s density
      else
        $\rho_{\text{next}} = \frac{c_{\text{per\_bin}}}{P_{i+1}-P_i}$ ▷ Computes bin_{i+1}'s density
      $P_i \leftarrow P_i + \delta c/\rho_{\text{next}}$ ▷ Right-expand bin_i
      $c_{\text{this}} \leftarrow \rho_{\text{next}}(P_{i+1} - P_i)$ ▷ Saves bin_{i+1} new count
    else
      $\rho_{\text{this}} = \frac{c_{\text{this}}}{P_i-P_{i-1}}$ ▷ Density at bin_i
      $P_i \leftarrow P_i + \delta c/\rho_{\text{this}}$ ▷ Left-expand bin_{i+1}
      $c_{\text{this}} \leftarrow c_{\text{per\_bin}} - \delta c$ ▷ Saves bin_{i+1} new count

  if $X > P_n$ **then** $P_n \leftarrow X$ ▷ Right-expand bin_n
  return $\mathbf{P}$

Figure 18: Pseudo-code of the SPEAR algorithm [11].

3. **Alarm Report** - The knowledge of the signal, alone, does not provide any information on the characteristics of the subset of events in the T window that caused the alarm [11]. Therefore, an automatic report is produced by comparing the events in the T and R windows through a measure of dissimilarity. The goal is to rank the T window events according to how likely they are to explain the alarm. For that propose, an auxiliary machine learning model is trained: for each alarm, a new target binary label with value 1 for events in T and value 0 for events in R, and then an auxiliary machine learning model is trained to learn how to separate events is the two windows. The authors use a Gradient Boosted Decision Trees (GBDT) model, which allows obtain an alarm score that can be used to rank events in T and directly obtain a measure of feature importance that deals well with correlated features. Since the goal of the ranking is to

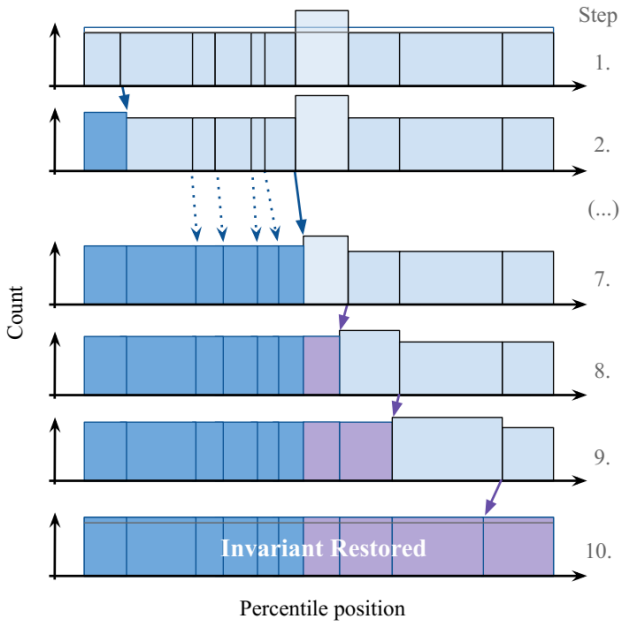


Figure 19: One call of UpdatePercentiles. 1) Adds event to bin 7 ($c_t arget$: blue horizontal line). 2) to 7) Right-expand Bin walls on the left. 8) to 10): Left-expand other bin walls [11].

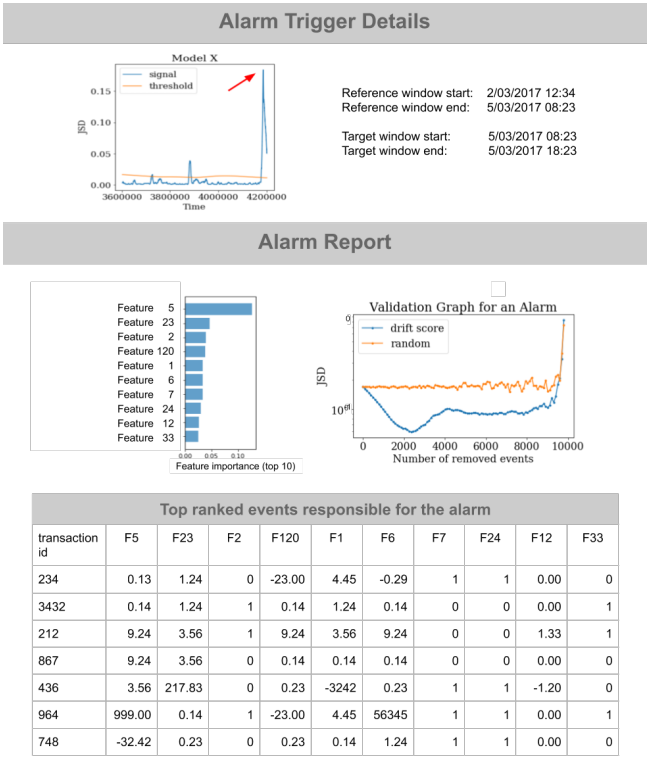
push to the top the events that are responsible for distorting the distribution of model scores in the target window, it is expected that if the events from the top are removed the signal will be suppressed. Having that in mind, the authors define a validation curve where each point is the value of the signal using R as reference, but T with the top k events removed. For comparison, they define a curve where, for each point, k events are randomly removed from T. If the signal is a false positive, the former should not be able to lower the signal value. In summary, the alarm report contains the following (a dummy example can be found in Figure 20):

- Window information with start and end timestamps
- Truncated feature importance ranking list
- Validation curve to observe how well the ranking can lower the signal
- A table of the top N events that explain the alarm.

In other to validate the proposed approach, the authors use two datasets with more than 20 million online transactions. Each dataset comprises several regions, existing a ML model responsible for scoring the transitions for each region. For their experiment, the authors had two data scientists for each dataset-region. 100 alarm reports were generated (20 per dataset-region) and each data scientist received a list of 1' alarm reports to analyse. For each alarm, the data scientists were asked to score, from 1 to 5, the following questions [11, 12]:

1. Please provide a score from 1 to 5 reflecting how confident you are that this report corresponds to a true alarm (1: totally sure it is false, 5: totally sure it is true).
2. Please provide a score from 1 to 5 reflecting how clearly you can see a pattern in the transactions of the report provided (1: no pattern at all, 5: very clear pattern).
3. After looking at the validation plot, please provide a new answer to Q1, regarding this new information (the validation plot was hidden from the user for Q1 and Q2).

In summary, the results revealed high scores for alarm reports for questions 1 and 3, where the results are statistically significant. These results validate the ability of the SAMM algorithm to detect abnormal events successfully. Regarding question 2, the differences were statistically significant only for one of the datasets. This may due to the fact that it is a more subjective question.



Top ranked events responsible for the alarm

transaction id	F5	F23	F2	F120	F1	F6	F7	F24	F12	F33
234	0.13	1.24	0	-23.00	4.45	-0.29	1	1	0.00	0
3432	0.14	1.24	1	0.14	1.24	0.14	0	0	0.00	1
212	9.24	3.56	1	9.24	3.56	9.24	0	0	1.33	1
867	9.24	3.56	0	0.14	0.14	0.14	0	0	0.00	0
436	3.56	217.83	0	0.23	-3242	0.23	1	1	-1.20	0
964	999.00	0.14	1	-23.00	4.45	56345	1	1	0.00	1
748	-32.42	0.23	0	0.23	0.14	1.24	1	1	0.00	0

Figure 20: Dummy report example [12].

5 Open Challenges

Although the fast growth of research in AutoML techniques over the last few years, being AutoML a vibrant and active research field in the present moment, there are still several open challenges to be solved. A deep, interesting and detailed discussion on the open challenges can be found in references [6, 8, 18]. Here, we identify and discuss the ones that we consider more relevant and that are referred by several authors:

- **Interpretability** - In general, AutoML solutions are black boxes that aim to explore the space of models that be build with a set of primitives [6]. However, ideally AutoML should be equipped with explainability and interpretability mechanisms. This enhancement could bring important contribution for making AutoML accessible to everyone, making it more user friendliness. Moreover, although AutoML algorithms can find promising configuration settings more efficiently than humans, there is a lack of scientific evidence to illustrate why the found settings perform better [8].

- **Scalability** - An important limitation of several AutoML techniques is scalability. Many of the existing frameworks for AutoML can only work on a single node which makes them not applicable in the case of large data volumes [5]. Although several distributed machine learning platforms have been recently introduced, they are still simple and limited in their capabilities. For those reasons, more research efforts and novel solutions are required to answer to the challenges of automatically build and tune machine learning models in massive datasets.

- **Reproducibility** - As machine learning, AutoML also has as major challenge reproducibility, especially for NAS, since most of the existing NAS algorithms still have many parameters that need to be set manually at the implementation level [8].

- **Robustness** - Although NAS has been proven effective in searching for promising architectures on many problems, in real-world situation the data inevitable contains noise. Even worse, data might be modified to be adversarial data with carefully designed noise. Deep learning models,

as well as NAS, are easily fooled by adversarial data. Therefore, more research in this field is still necessary.

- **Complete AutoML Pipeline** - Although there are several AutoML tools, as deteiled discussed in section 3 of this work, most of them only focus on some parts of the AutoML pipeline. Although there are works, as the third paper presented in section 4 of this work, that apply AutoML to the entire pipeline, more research and developments in this area are needed and expected in a near future.

- **Lifelong learning** - Most AutoML algorithms focus only on solving a specific task in some fixed datasets (e.g., image classification). However, a high-quality AutoMl system should have the capability of lifelong learning, that is, it should have the capability of learning new data while remember and use old knowledge [8]. In other words, an AutoML system should be able to learn from new data without forgetting the knowledge obtained from old data. However, if we use a new data set to train a pretrained model, the performance of the model in the previous data sets will be gratly reduced [8].

6 Conclusions

It is possible to observe how machine learning has emerged as one of the most important engines of the modern era and the need for this process to be automated. After intense research, it was also found that the scope of AutoML applied to data streams is still not very explored and few algorithms have been implemented so far.

Highlights were given to meta learning processes, the process of learning from previous experience gained during applying various learning algorithms on different kinds of data; also to neural architecture search, based on artificial neural networks (ANN) and process of hyper-parameter optimisation, where tuning the hyper-parameters of the model is crucial to further optimise the model performance.

The papers mentioned bring some approaches on implementing Auto-ML algorithms for data streams, with emphasis on the implementation of versions of the Single-pass Self Parameter Tuning (SSPT) algorithm, which is a refined version of the Double-pass SPT (DSPT)) algorithm, designed to optimize a set of hyperparameters in large search spaces.

The results show that Double-Pass SPT (DSPT), in general, performs better. This is also observed when comparing grid search and random search optimisation. Both are offline brute force techniques that require longer exploration stages, making them inappropriate for data stream problems.

In addition, paper [4] discuss six distinct adaption techniques using existing open-source AutoML packages for automatic tuning of ML pipelines of data streams. The main conclusion of the authors is that the proposed strategies are effective in allowing AutoML techniques to recover from concept drift and that the more suited approach may depend of the characteristics of the drift.

The paper [11] shows an approach that is able to detect the concept drift in an unsupervised way and it suggests an explanation for the drift in the form of an alarm report. The results obtained by the authors validate the proposed approach.

Finally, a discussion on the open challenges and research paths of AutoML that need to be addressed was also presented.

References

[1] *AutoML 2016 @ ICML*. URL: https://sites.google.com/site/automl2016/.

[2] Manuel Baena-Garcıa et al. "Early drift detection method". In: *Fourth international workshop on knowledge discovery from data streams*. Vol. 6. 2006, pp. 77–86.

[3] A. Bifet and Richard Kirkby. "DATA STREAM MINING A Practical Approach". In: 2009.

[4] Bilge Celik and Joaquin Vanschoren. "Adaptation Strategies for Automated Machine Learning on Evolving Data". In: *CoRR* abs/2006.06480 (2020). arXiv: 2006.06480. URL: https://arxiv.org/abs/2006.06480.

[5] Radwa Elshawi, Mohamed Maher, and Sherif Sakr. *Automated Machine Learning: State-of-The-Art and Open Challenges*. 2019. arXiv: 1906.02287 [cs.LG].

[6] Hugo Jair Escalante. *Automated Machine Learning – a brief review at the end of the early years*. 2020. arXiv: 2008.08516 [cs.LG].

[7] João Gama et al. "Learning with Drift Detection". In: vol. 8. Sept. 2004, pp. 286–295. ISBN: 978-3-540-23237-7. DOI: 10.1007/978-3-540-28645-5_29.

[8] Xin He, Kaiyong Zhao, and Xiaowen Chu. "AutoML: A survey of the state-of-the-art". In: *Knowledge-Based Systems* 212 (Jan. 2021), p. 106622. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2020.106622. URL: http://dx.doi.org/10.1016/j.knosys.2020.106622.

[9] Alexandru-Ionut Imbrea. *Automated Machine Learning Techniques for Data Streams*. 2021. arXiv: 2106.07317 [cs.LG].

[10] L. Li et al. "Hyperband: A novel bandit-based approach to hyperparameter optimization". In: *Journal of Machine Learning Research* 18 (Apr. 2018), pp. 1–52.

[11] Fábio Pinto, Marco O. P. Sampaio, and Pedro Bizarro. *Automatic Model Monitoring for Data Streams*. 2019. arXiv: 1908.04240 [cs.LG].

[12] Marco O. P. Sampaio. *ML-Powered Automatic Model Monitoring*. Nov. 2019. URL: https://medium.com/feedzaitech/ml-powered-automatic-model-monitoring-d1841efa0ba8.

[13] Joaquin Vanschoren. *Meta-Learning: A Survey*. 2018. arXiv: 1810.03548 [cs.LG].

[14] Bruno Veloso, Joao Gama, and Benedita Malheiro. "Self Hyper-Parameter Tuning for Data Streams: 21st International Conference, DS 2018, Limassol, Cyprus, October 29–31, 2018, Proceedings". In: Jan. 2018, pp. 241–255. ISBN: 978-3-030-01770-5. DOI: 10.1007/978-3-030-01771-2_16.

[15] Bruno Veloso et al. "Hyperparameter self-tuning for data streams". In: *Information Fusion* 76 (Apr. 2021). DOI: 10.1016/j.inffus.2021.04.011.

[16] Li Yang and Abdallah Shami. "On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice". In: *CoRR* abs/2007.15745 (2020). arXiv: 2007.15745. URL: https://arxiv.org/abs/2007.15745.

[17] Li Yang and Abdallah Shami. "On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice". In: *CoRR* abs/2007.15745 (2020). arXiv: 2007.15745. URL: https://arxiv.org/abs/2007.15745.

[18] Quanming Yao et al. "Taking Human out of Learning Applications: A Survey on Automated Machine Learning". In: *CoRR* abs/1810.13306 (2018). arXiv: 1810.13306. URL: http://arxiv.org/abs/1810.13306.

[19] Jiakun Zhao et al. "A Neural Architecture Search Method Based on Gradient Descent for Remaining Useful Life Estimation". In: *Neurocomputing* 438 (Jan. 2021). DOI: 10.1016/j.neucom.2021.01.072.