Data Mining and Knowledge Discovery (2022) 36:2006–2032
https://doi.org/10.1007/s10618-022-00858-9

SOKNL: A novel way of integrating K-nearest neighbours with adaptive random forest regression for data streams

Yibin Sun[1] · Bernhard Pfahringer[1] · Heitor Murilo Gomes[1,2] · Albert Bifet[1]

Received: 11 December 2021 / Accepted: 18 July 2022 / Published online: 13 August 2022

Abstract

Most research in machine learning for data streams has focused on classification algorithms, whereas regression methods have received a lot less attention. This paper proposes Self-Optimising K-Nearest Leaves (SOKNL), a novel forest-based algorithm for streaming regression problems. Specifically, the Adaptive Random Forest Regression, a state-of-the-art online regression algorithm is extended like this: in each leaf, a representative data point – also called centroid – is generated by compressing the information from all instances in that leaf. During the prediction step, instead of letting all trees in the forest participate, the distances between the input instance and all centroids from relevant leaves are calculated, only k trees that possess the smallest distances are utilised for the prediction. Furthermore, we simplify the algorithm by introducing a mechanism for tuning the k values, which is dynamically and automatically optimised based on historical information. This new algorithm produces promising predictive results and achieves a superior ranking according to statistical testing when compared with several standard stream regression methods over typical

Responsible editor: Albrecht Zimmermann.

Bernhard Pfahringer, Heitor Murilo Gomes and Albert Bifet have contributed equally to this work.

✉ Yibin Sun
ys388@students.waikato.ac.nz

Bernhard Pfahringer
bernhard@waikato.ac.nz

Heitor Murilo Gomes
heitor.gomes@waikato.ac.nz

Albert Bifet
abifet@waikato.ac.nz

[1] AI Institute, University of Waikato, Hamilton, New Zealand

[2] School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand

benchmark datasets. This improvement incurs only a small increase in runtime and memory consumption over the basic Adaptive Random Forest Regressor.

Keywords Data streams · Regression · KNN · ARF-Reg

1 Introduction

Vast amounts of data are generated nowadays continuously in real-time as data streams. Streaming data assumes that data examples can only be inspected once, making it unfeasible to iterate over a dataset repeatedly to obtain a better solution. Consequently, most of the algorithms for conventional batch learning can not be applied to stream learning directly.

Moreover, concept drift is another significant challenge in streaming tasks. The basic assumption of concept drift is that data streams may evolve over time; in other words, the distributional properties of the streaming instances vary in some unforeseeable ways. The concepts of concept drift for regression and classification problems are only slightly different. Commonly, there are three aspects for identifying drifts – from the feature space perspective, from the target value perspective, and from the performance perspective. The largest difference is the estimation of the distributional properties of the target values. Instead of exploiting discrete statistic model like Poisson's distribution, regression problems use continuous model like Gaussian distribution to determine the changes of the concept. Furthermore, regression tasks suspect drifts when the metrics of performance like MAE or RMSE increase. Elaborate review and survey about concept drift are sufficient in literature (Lu et al. 2018), (Choudhary et al. 2021). If concept drifts happened, the current models were no longer suitable and accurate. Therefore, the capability of detecting and adapting to the changes in the datasets is another characteristic that streaming algorithms must possess.

Regression learning is an important task of machine learning. However, regression learning for streaming data is relatively under-represented in comparison to classification algorithms. Nonetheless, some high-quality data stream regression algorithms are available, two of which are relevant to this paper: k Nearest Neighbours (KNN) (Dhanabal and Chandramathi 2011) and Adaptive Random Forest for Regression (ARF-Reg) (Gomes et al. 2018).

The predictive performance of both algorithms is remarkable (as shown by previous studies (Gomes et al. 2018, 2020)), but they still have some shortcomings. KNN requires many distance calculations during prediction, which can be slow and prohibitive. On the other hand, due to the random nature of ARF-Reg, not every tree in it may produce accurate predictions, which is adding unnecessary noise to the final prediction when all those individual trees are aggregated.

The main contributions of this paper are the following:

- a novel approach to regression analysis for data streams by combining ARF-Reg and KNN into the Self-Optimising k Nearest Leaves (SOKNL) algorithm;

- through the combination of both algorithms, we improve the predictive performance of ARF-Reg without adding too much extra pressure on computational resources.
- a dynamical parameter-choosing methodology enabling the algorithm to self-adapt the value of k;
- an extensive empirical evaluation and a statistical test show how the new method compares with other previous state-of-the-art online regression algorithms.

The rest of this paper is organised as follows. The Sect. 2 introduces the sliding window KNN and other related work. In the following section, we proceed to explain our approach. The third section depicts the experimental results and analyses. Finally, we conclude the paper by summarising our contributions and presenting future work.

2 Related work

Traditionally, the regression version of the KNN algorithm has access to the whole dataset. Consequently, it has enough information for finding the k instances with the smallest distances from the incoming instance, i.e. the k nearest neighbours. The predicted value is usually given by aggregating the target values of the k neighbours using the mean, weighted mean, or other strategies.

However, in the stream setting, it is implausible to grant the access to the dataset as a whole. Due to memory constraints, storing all the instances is infeasible. A sliding window strategy can be used to circumvent this limitation. The sliding window only contains a certain number of instances, and when a new instance arrives for training, the oldest instance in the window is removed. At the prediction phase, only the k nearest neighbours *inside* the window are utilised for providing the final result. By design, KNN with a sliding window automatically handles concept drifts, as it "forgets" older instances when they drop out of the sliding window. However, there is a trade-off regarding the window size: smaller windows respond faster to changes but may not keep sufficient data for high-quality predictions and noise resistance.

Recently, SAM-kNN (Losing et al. 2018) was proposed to improve upon the performance of streaming kNN for data stream classification in evolving scenarios. SAM-kNN maintains two types of memories during execution, a short-term and a long-term one. Instances from the short-term memory will be transferred into the long-term one when the predictive error increases. When the long-term memory reaches capacity, its instances will be compressed using kMeans++ (Arthur and Vassilvitskii 2006). Thus, SAM-kNN can keep track of both current and previous concepts in a data stream, enabling drift adaptation.

Hoeffding Trees (Domingos and Hulten 2000) became a popular incremental decision tree algorithm due to its promise of convergence to the same structure of a batch decision tree. Hoeffding trees are suitable for online classification as they update the tree structure incrementally instead of processing a batch of instances. Hoeffding trees are based on the idea of using the Hoeffding bound (Hoeffding 1994) to determine when to split without seeing too many instances. Hoeffding trees continuously check the split points for different features and their related statistics during the instances

streaming in. If the optimal splitting decision proves to exist (determined by Hoeffding bound), the splitting will be executed. These ideas inspired a lot of later incremental decision tree algorithms, including the Fast Incremental Model Trees with Drift Detection (FIMT-DD) (Ikonomovska et al. 2011).

FIMT-DD is a variant of the Hoeffding tree algorithm for regression problems, which also incorporates the capability to adapt to concept drifts. Resembling a regular Hoeffding Tree, FIMT-DD starts from an empty node - which is called the root node - that is trained along with instances arriving until it reaches the end of a grace period. When this happens, merits for each feature of specific splitting values will be calculated based on their variance. Whereafter, the tree will branch if the difference between the best and the second-best merits surpasses the Hoeffding Bound, and then the process iterates. If the variance dramatically increases, the drift detector will trigger and implement adaptions.

The Adaptive Random Forest Regressor algorithm (ARF-Reg) (Gomes et al. 2018) "ensembles" several FIMT-DDs to achieve higher predictive performance. In order to introduce diversity into the ensemble, the trees are trained on disparate subsets of the datasets as well as the feature space (Breiman 2001), which is usually named Random Patches (Louppe and Geurts 2012). In addition, each training instance is trained for multiple times based on a Poisson Distribution with the parameter $\lambda = 6$, which is an essence of the Leveraging Bagging technique (Bifet et al. 2010). With these manners, the ARF-Reg consists of multiple diverse yet powerful single trees that ensure the outstanding outcomes in most cases and tasks. ARF-Reg uses the same strategies to cope with concept drift as its classification counterpart, Adaptive Random Forest (ARF) (Gomes et al. 2017). The concept drift conquering strategy in ARF contains two levels – the Tree level and the Ensemble level. We provide detailed description in Sect. 3.3.

Moreover, for the purpose of comparison, two more algorithms are included in the experimental phase. The first one is On-line Regression/Model Tree with Options (ORTO), which is a variant of FIMT-DD. ORTO (Ikonomovska et al. 2011) introduces optional nodes instead of only having binary split children as FIMT-DD does. Examples will be passed down to every optional node and if there is any ambiguity of where the best split should be, the algorithms will split on all the competitive nodes. The second algorithm is Adaptive Model Rules (AMRules) (Almeida et al. 2013), which starts with an empty rule set (RS) and a default rule. Each instance is checked if it is covered by any rule in the RS. The Page-Hinckley based change detection mechanism takes place after the rules testify the instance. If changes are identified, the rule will be removed. If the instance is not covered by any existing rule, a default rule will be expanded and appended into the RS. AMRules employs standard deviation reduction (SDR) (Ikonomovska et al. 2011) and the Hoeffding Bound (Hoeffding 1994) to determine which split is the best choice for expansion. The expanding procedure is also applied on a regular basis to update the existing rules. Any type of expansion will only be considered after a certain number of examples are processed.

There are algorithms that extend KNN. One example is the Instance Based Classification and Regression on Data Streams (IBLStreams) (Shaker and Hüllermeier 2012). In IBLStreams, three aspects of the usefulness of the instances are considered: *Temporal Relevance*, *Spatial Relevance*, and *Consistency*. To put it simply, *Temporal*

Relevance means that newer instances contain more information than older ones; *Spatial Relevance* means that instances in a sparse region are more relevant than those in a dense region; *Consistency* means that an instance will be determined as useless when its behaviour is evidently different from its neighbours. IBLStreams stores a certain number of instances for predictions that is called a "case base". When a new instance arrives, the system will check if the instance is significantly distinguished from the neighbours, and if true, the new instance is removed to guarantee the *Consistency*. Otherwise, the neighbourhood of the new instance will be checked for density and the new instances will be checked for redundancy. If the region is dense and the instances are redundant, the oldest instance will be removed and the new instances will be added to the case base. In this manner, *Temporal Relevance* and *Spatial Relevance* are ensured. The prediction procedure is similar to the KNN algorithm within the case base.

Weighting or eliminating the participants in an ensemble learner has also received attention. In general, they are called Dynamic Ensemble Selection (DES) or Abstaining Ensemble. Recently, (Krawczyk and Cano 2018) proposed an ensemble abstaining strategy for improving classification predictive accuracy. In their proposal, the algorithm, Online Ensemble of Abstaining Classifiers, is set with a dynamic threshold of accuracy. All the predictive accuracy of the base learners is compared against this threshold at the prediction step. If an individual learner is not more accurate than the threshold, it will be forced to abstain from the following majority voting process. Arbitrating Dynamic Ensemble (ADE), presented in (Cerqueira et al. 2017) by Cerqueira et al., utilises a meta-learning strategy. In ADE, a base-model layer $\mathcal{M}$ and a meta layer $\mathcal{Z}$ are established at the same time. $\mathcal{M}$ is trained in a regular machine learning manner and the $\mathcal{Z}$ is updated according to the predictive performance of the base-models. An evaluation of the expertise of those learners in $\mathcal{M}$ is structured based on $\mathcal{Z}$ and the final predictions will be a weighted vote. Boulegane et al. furthered the idea of ADE in (Boulegane et al. 2019) by proposing Streaming Arbitrated Dynamic Ensemble (Streaming-ADE). Streaming-ADE also maintains a two-layer system similar to ADE. The major difference is that the Streaming-ADE also introduces an abstaining course. Not confident enough models in $\mathcal{M}$ are not allowed to contribute to the final predictions.

3 Self-optimising K nearest leaves

As mentioned in Sect. 2, both KNN and ARF-Reg have their own specific inherent shortcomings.

Streaming versions of KNN rely only on a sliding window where older instances are forgotten to limit memory usage. This strategy can be beneficial in situations where older instances no longer represent the current concept (i.e. a drift has happened); however, the older instances are relevant to building a more robust model in many situations. Under the circumstances where the concept of the data streams has not drifted, which means all observed data points are valuable for establishing learning model, we may want to somehow store those information from older instances. One

of the core questions for streaming KNN is how to retain some of that older but still relevant information.

ARF-Reg randomly trains and grows ensemble trees on different subspaces of the datasets and features. Thus this approach relinquishes at random some bits of informative data for each ensemble member to introduce diversity into the ensemble. For instance, the artificial dataset "fried", which is used in our experiments, contains ten features, only five out of which are related to the target value. If some trees in ARF-Reg are trained mainly on the irrelevant features, they can potentially yield inaccurate single predictions, negatively impacting the aggregated ensemble prediction.

Consequently, the idea of *k* Nearest Leaves (KNL) – which is the integration of KNN and ARF-Reg (see in Sect. 3.1)– emerged. The intuition is to overcome the transient behaviour of KNN by using the trees in the ARF-Reg ensemble to keep information for much longer, by maintaining a centroid for each leaf in each tree; and to improve the ensemble prediction aggregation procedure using KNN over the centroids selected by each tree at prediction phase. Using centroid to summarise information is common, one of the most famous applications is the Clustering Feature Tree (CF-Tree) from BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) (Zhang et al. 1996). In our approach, each leaf is mapped to a "data point" for later use of the KNN procedure.

In general, there are two perspectives to comprehend the KNL algorithm. From the standpoint of KNN, ARF-Reg is providing the leaves – which can also be regarded as micro-clusters, condensed into one compact and robust representation: a centroid. From the ARF-Reg perspective, the prediction is more robust by excluding leaves that may be too dissimilar to the current prediction instance.

Figure 1 is a diagram visualising the idea of KNL.

Our approach has a virtue as a streaming algorithm, which is all the calculating, learning, updating, and comparing can be accomplished by only using the statistics storing in the system. As a consequence, the instances are not demanded to stay in the trees or the forests. Therefore, it is not violating the only-one-see-instance setting for data streams. For instances, the centroid in the leaf could be calculated by a counter and an array of sum value of individual features of all instances that have been through the leaves. Memory constraint is complied with via thus approach.

There is a potential downside to this combination, as KNL now adds one more hyperparameter to the set of hyperparameters of ARF-Reg: the *k* value needed for selecting the closest leaves. To simplify the application of KNL, we also introduce a technique for automatically and dynamically selecting a good value for this "k" hyperparameter. More details are given below in Sect. 3.2. This modified version of KNL is called Self-Optimising *k* Nearest Leaves, abbreviated as SOKNL.

In the rest of this section, SOKNL will be explained more specifically. In general, our contributions can be separated into two segments: a) Integrating KNN with ARF-Reg, and b) the Self-Optimising Strategy. See the pseudo-code of SOKNL in Algorithm 1 for an intuitive understanding.

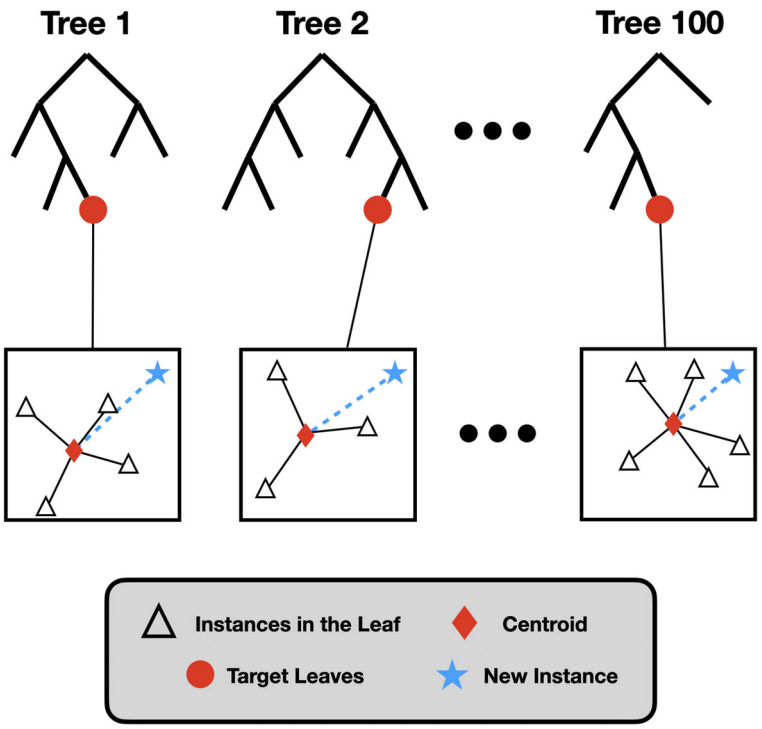

Fig. 1 Diagram of K Nearest Leaves

3.1 Integrating K-nearest procedure with ARF-reg

As aforementioned in Sect. 2, the ARF-Reg is an ensemble of multiple tree learners. Every tree will traverse an incoming instance into exactly one of its respective leaves. Thus an ensemble of n trees will return n leaves for prediction.

3.1.1 Selection of K nearest leaves

Instead of aggregating the leaf predictions of all trees in the ensemble, SOKNL only averages the target values of the k nearest leaves. Since ARF-Reg trees are growing semi-randomly, some leaves will be more informative than others, for a given instance. SOKNL is able to select the most relevant leaves, thanks to the centroids stored within each leaf.

The abstraction of centroids in our method is simple. For all the instances in a leaf, we take the mean values of each feature. For example, we average the values of Feature 1 of all the instances in the leaf and put it at the Feature 1 position of the centroid. With this method, all the information of the instances in a certain leaf is compressed into a robust, space- and memory-efficient, and incrementally calculable representative instance – the centroid.

Algorithm 1 Self-Optimising k Nearest Leaves

function PREDICT($instance$)
  ARF-specific updates
  $leaves \leftarrow \{t_i.findLeaf(x), t_i \in trees\}$
  $leaves \leftarrow sort\_by\_centroid\_dist(leaves, x)$
  $p = aggregate\_predictions(leaves)[1 : k_{best}]$
  return p
end function
function TRAIN($instance$)
  ARF-specific updates
  $leaves \leftarrow \{t_i.findLeaf(x), t_i \in trees\}$
  for $leaf \in leaves$ **do**
    $leaf.update\_centroid(x)$
  end for
  $leaves \leftarrow sort\_by\_centroid\_dist(leaves, x)$
  $p \leftarrow 0$
  for $i \in 1..k_{max}$ **do**
    $p \leftarrow aggregate(p, leaves[i].predict(x))$
    $evaluators[i].update(x, p)$
  end for
  $k_{best} \leftarrow \underset{i}{argmin}(evaluators[i].SSE)$
end function

3.1.2 Measurement of the distance

The next question is how to measure the distance from an instance to a leaf (a cluster of instances). In principle, there are at least two options available:

- Calculate all the distances from the incoming instance to the instances in the leaf, and use the average as the measurement of the distance between an instance and a leaf.
- Maintain a centroid in each leaf, which is an average of all features of all the leaf's instances. Thereupon, the distance from an instance to a leaf can be defined as the distance to the leaf's centroid.

Both options are feasible and have their own pros and cons. The former one keeps more information which gives it more flexibility. It also needs much more memory and runtime. Due to the data stream algorithms' requirements for timeliness and memory space utilisation, the latter option is more promising, and consequently suitable for SOKNL.

Another notable point is, although SOKNL also requires distance calculation, it limits the amount of the calculation to the ensemble number of the ARF-Reg which is 100 in our experiments (it is already a quite large choice for ensemble number). In the Sliding Window KNN case, the quantity of distance calculation equals to the window length, which is typically several thousands or longer. Hence, in terms of time for calculating distance, SOKNL manages to surpass the KNN algorithms.

3.2 Self-optimising strategy

Choosing good k values for kNN requires specialised knowledge and sometimes "luck". Even for experts, there is no convenient way for finding the best k. Moreover, there may even not be a "global best" k due to concept drifts or other causes. Therefore, a self-optimising regime is introduced into SOKNL to automatically determine the currently best-performing value for k.

Self-optimising, or self-tuning, is a popular way to reduce the labour of hyperparameter tuning (Huang et al. 2021; Veloso et al. 2018; Luo 2016). Here, the self-optimising mechanism measures performance using the Sum Squared Error (SSE). For every possible value of k, for k in $1..k_{max}$, an evaluator keeps track of the SSE for this k value. Updating all k evaluators can be done very efficiently: first, all leaves are sorted by distance, and then the predictions for larger and larger values of k are incrementally computed in one linear sweep over the sorted leaves, resulting in $O(k_{max} * log(k_{max}))$ runtime. Even though the possibility of two k values maintaining exact same SSE is extremely small, there is no guarantee for that not to happen, especially at the beginning of the learning progress. We simply choose the smaller k value if it occurs.

3.3 Change adaptation

How our approach adapts to the drifts is straightforward. It relies on the built-in adaption methodology in ARF-Reg as well as implicit adaptation of the centroids.

In ARF-Reg, the concept drift detection and adaptation exist in both trees and ensembles. In every node of the tree learners, there is a Page-Hinckley (PH) test (Mouss et al. 2004), which is an extension to the CUSUM test Page (1954). PH maintains two variables in the system, a cumulative value m_t and the minimum of m_t until the current moment M_t. m_t is defined as the cumulation of the difference between the current target value and the average value ($\bar{x}$) of the whole time and an indicative parameter α, which is responsible for controlling how much of the change ought to be identified as a drift. Eqs. 1 and 2 denote m_t and M_t respectively.

$$m_t = \sum_{t=1}^{N} (x_t - \bar{x} - \alpha) \tag{1}$$

$$M_t = min\{m_t, t = 1, 2, ..., N\} \tag{2}$$

Consequently, the PH test is at the moment of t is defined by Eq. 3:

$$\mathrm{PH}_t = m_t - M_t \tag{3}$$

If the PH_t is larger than a threshold parameter λ that is specified by the users to command the sensitivity of the test, a drift is confirmed. Then the associated node will be removed and a new branch will be built.

ARF-Reg has an ADaptive WINdow (ADWIN) (Bifet and Gavalda 2007) algorithm as an external change detector at the ensemble level. The operation of the ADWIN requires storing two windows that have window lengths of $\mathcal{W}_{old}$ and $\mathcal{W}_{new}$. The mean

Table 1 Datasets Overview

Datasets	$N_{Features}$	$N_{Instances}$	Source
Ailerons	40	13750	Synthetic
Elevators	18	16599	Synthetic
Fried	10	40768	Synthetic
HyperA	10	500000	Synthetic
Abalone	8	4977	Real
Bike	12	17379	Real
House8L	8	22784	Real
MetroTraffic	7	48204	Real

values of each window, μ_{old} and μ_{new} , are used to compare to the corresponding Hoeffding Bound ϵ by Inequation 4.

$$\mid \mu_{old} - \mu_{new} \mid \geq \epsilon = \sqrt{\frac{1}{2m} \cdot \ln \frac{4 \mid \mathcal{W} \mid}{\delta}} \tag{4}$$

where δ denotes a use-defined confidence level in the range of [0, 1], and m is the harmonic mean of the sub-windows' length as in Eq. 5.

$$m = \frac{2}{\frac{1}{|\mathcal{W}_{old}|} + \frac{1}{|\mathcal{W}_{new}|}} \tag{5}$$

If Inequation 4 holds, a drift is detected and the old window is dropped.

In addition, SOKNL has another manner to adapt to the drifts. The centroids in SOKNL are moving according to incoming instances. If instances on average shift, the centroids will shift as well, thus implicitly providing a certain capability for drift adaptation, on top of the explicit change detectors. Furthermore, when a leaf is split into two new leaves, the old centroid is deleted, and two new centroids will be computed from the newly arriving instances, yet again supporting adaptation to potential drifts implicitly.

4 Experimental setting

In this section, information regarding our experiments is specified for reproducibility.

4.1 Datasets

Table 1 provides an overview of the involved datasets:

Most of them are standard benchmark datasets. For example, Abalone is from a non-machine-learning research paper (Nash et al. 1994) and aims at predicting the age of abalones based on some physical measurements. Another example, the Fried (Friedman 1991) dataset, is a synthetic one, using this highly non-linear formula:

$y = 10sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \sigma(0, 1)$. Notably, the Fried dataset includes five more features that are not involved in the generation formula, which means they are irrelevant to the ground truths. By introducing irrelevant features, this dataset is able to further test the robustness of regressors. Moreover, the synthetic dataset HyperA is another one that requires specific introduction. HyperA generates an hyperplane in a d-dimensional space. The goal is to predict the distance from randonly generated data points to the hyperplane. The more crucial characteristic of HyperA dataset is, since it is artificial, it simulates drift detection every $125K$ instances, i.e., at the position of $125K$, $250K$ and $375K$ instances, which makes it applicable for assessing the drift detection ability of algorithms. It is worth mentioning that there are no nominal attributes in these datasets, which makes the centroid calculation more straightforward. However, it is easy to cope with nominal attributes by applying One-Hot-Encoding method or other techniques. The effectiveness will be assessed with following research.

4.2 Data pre-processing

Data pre-processing techniques are commonly used in regression tasks, one of which is standardisation. Amongst the many approaches to standardisation, we use Z-scores to transform the data to have zero mean and a variance of one. The z-score formula is:

$$X' = (X - \overline{X})/\sigma \tag{6}$$

where the $\overline{X}$ is the mean and the σ is the standard deviation of the original data, and the X' is the transformed X value.

However, the problem is that the distributional properties are unknown in streaming data. Hence, instead of using the "global" mean and standard deviation, only dynamically updated online estimates of mean and standard deviation are used for standardisation. Notably, categorical features would need to be transformed by one-hot encoding or a similar technique.

The results with or without pre-processing are strongly similar towards one another. Thus, due to the space constraint, only pre-processed results are exhibited in this paper.

4.3 Algorithms

For comparison, experiments on the above datasets with some other algorithms are conducted, including Traditional kNN; Self-Optimising kNN;[1] FIMT-DD; ORTO; AMRules; ARF-Reg; and k Nearest Leaves(KNL).[2]

Most of these algorithms have been introduced in Sect. 2, yet there are several things to be specified here.

- **Traditional KNN:** The classic KNN with fixed k values of 1, 5, and 10.

[1] A variant of kNN with the proposed self-optimising method.

[2] The intermediate version of SOKNL without self-optimising.

- **Self-Optimising k Nearest Neighbours(SOKNN):** A variant of KNN with the same auto-selecting k value method as used in SOKNL .
- **FIMT-DD:** The hyper-parameters of the FIMT-DD are as default in (Ikonomovska et al. 2011).
- **ORTO:** Option trees using the default parameter setting in (Ikonomovska et al. 2011).
- **AMRules:** Adaptive Model Rules, also known as AMRules, is the first rule algorithm in regression for data streams (Almeida et al. 2013). We use the same algorithm setup as in that paper, except for the split confidence parameter δ, where $1E-7$ is used instead of 0.01.
- **ARF-Reg:** See details in Sect. 2 and the algorithm hyper-parameters are set as this: *a)* **Ensemble learner:** FIMT-DD; *b)* **Ensemble Size:** 100; *c)* **Feature Numbers in Subspace:** 60%; *d)* λ **for Poisson Distribution:** 6 (for simulating the bootstrapping procedure)
- **K Nearest Leaves:** For comparison to SOKNL , a plain KNL with a fixed k value is also in included. Values used for a k are 1, 5, and 10. We choose these k values as they illustrate the improvements by increasing the number of neighbours. The differences are more significant when k values are small. After 10, the improvements start getting inconspicuous and the rise of computational cost seems unworthy.

In order to make our experiments reproducible, the hyperparameters for the algorithms are included here. Moreover, the hyperparameters for ARF-Reg are introduced particularly since our approach is based on the ARF-Reg and the basic hyperparameters are the same. Notably, the ensemble numbers of ARF-Reg and SOKNL in our experiments are arbitrarily fixed to 100. That is because the number could be too small, otherwise, the centroid selecting procedure would be too limited to be effective. Also, large numbers could cause a dramatic increase in the computational resource requirements. We made some attempts with some "appropriate" ensemble numbers and chose 100 as it is the best in terms of balancing effectiveness and efficiency.

Implementations of the algorithms mentioned in this section, including the standardisation filter, can be found and utilised in the Massive Online Analysis (MOA) (Bifet et al. 2010) – a well-known free open-source framework software for machine learning and data stream mining.

4.4 Experimental evaluation

In this section, we briefly introduce the corresponding evaluation methods and metrics used in our experiments.

4.4.1 Metrics

There are lots of ways to evaluate the basic performance of machine learning algorithms. In this paper, the Root Mean Squarer Error (RMSE) is used for evaluating the overall performance of the algorithms. In addition, for comparing ARF-Reg with SOKNL, the Root Relative Squared Error (RRSE) is also presented. The formulas for

RMSE and RRSE are :

$$RMSE = \sqrt{\frac{\sum_{j=1}^{n} \left(P_j - T_j\right)^2}{n}}$$

$$RRSE = \sqrt{\frac{\sum_{j=1}^{n} \left(P_j - T_j\right)^2}{\sum_{j=1}^{n} \left(T_j - \overline{T}\right)^2}}$$

where P_j is the prediction; T_j is the target value; and $\overline{T} = \frac{1}{n} \sum_{j=1}^{n} T_j$ is the mean of all target values.

The main advantage of the RRSE is that it can convert the RMSE from different datasets to a similar scale so that the "horizontal" comparisons become meaningful. For error estimation, the lower the RMSE or RRSE, the better.

4.4.2 Processing time

Processing time is an important aspect that should be paid attention to in stream mining. The streaming data may be so enormous that if the algorithms are not efficient enough, the incoming rate will exceed the processing speed, violating the principle of data stream mining. Therefore, we include a table showing the running time of all algorithms in Sect. 5.

4.4.3 Coefficient of determination

Coefficient of Determination (Wright 1921), also known as R-squared or R^2 in literature, is formulated as follow:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (f(x_i) - y_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \tag{7}$$

where $f(x_i)$ is the prediction given x_i, y_i denotes the true value, and the $\bar{y}$ represents the mean of the true value.

The range of R^2 is from $-\infty$ to 1.0, again allowing for comparison between different algorithms over multiple datasets. The recently published (Chicco et al. 2021) presents arguments in favour of R^2 over the use of MSE, MAE, or similar.

4.4.4 Quade test

We also take a ranking methodology to evaluate the performance of our proposed algorithm. The Quade test (Quade 1979), which is a weighted nonparametric testing procedure, provides a manner for ranking multiple algorithms on multiple datasets. Instead of generating the ranks under the assumption that every dataset deserves the same significance (as in Friedman Aligned Ranks (Friedman 1937)), the Quade test takes into consideration the "difficulties" of the datasets and then weighs the ranks accordingly.

Assume there are $i \times j$ metric values (e.g. R^2) to rank (i datasets and j algorithms), the Quade test works like this:

1. Compute d_i, which is the difference between the best and worst case for each dataset;
2. Rank d_i as Q_i and assign Q_i to the associated dataset, using averages for tie situations;
3. Rank the performance of the algorithms for each dataset and denote them by r_i^j;
4. Calculate the weighted rank W_{ij} [3] for every element in the table using the formula: $W_{ij} = Q_i \times r_i^j$.
5. Average the W_{ij} for each algorithm to gain the final rank.

Self-evidently, the lower the rank is, the better the algorithm is. (García et al. 2010) provides more detail, explanation, and examples for the Quade test.

5 Experimental results and discussion

Tables and figures containing the outcomes from our experiments are presented in this section.

Table 2 illustrates the RMSE results after standardising the data. The best results for each dataset are displayed in bold font. We run each algorithm on each dataset ten times to get fair results, during which the order of the examples inside of the datasets is not modified or changed. The cells are in the form of MEAN(SD). Different random seeds are set for non-deterministic algorithms (e.g. AMRules, ARF-Reg) in different runs. We did not manually modify the order of the instances in the datasets during each running. Similar to the RMSE illustration, the running time of the experiments is shown in Table 3. To measure the experimental memory use, we include Table 4 which exhibits evaluations of RAM. RAM metric is simply the amount of memory consumption for maintaining the predictive model during the whole experiment.

The most apparent conclusion from Table 2 is that SOKNL provides the best results on five out of eight datasets in terms of RMSE. Furthermore, SOKNL takes the first place in all the real datasets (Abalone, Bike, House8L, and MetroTraffic), which makes more practical significance since the synthetic datasets, no matter how complicated and randomised their forming formulae are, would still struggle in simulating the real-world problems. Especially for the MetroTraffic dataset, which possesses information regarding the traffic volume of a metro station in the US, all the algorithms have difficulties in predicting accurately. There are three datasets in that SOKNL loses the first place. In the cases of Ailerons and Elevators, SOKNL loses it to the family of KNN algorithm. The reason, in our assumption, is that these two synthetic datasets are generated by some distance-relevant or distance-sensitive formulae. Therefore, the KNN family is evidently more suitable for them while SOKNL drop certain information after the micro-clustering procedure. Nevertheless, the gaps are tiny, thus acceptable,

[3] There is a different way to calculate the weighted rank using this formula: $W'_{ij} = Q_i \times [r_i^j - \frac{1+j}{2}]$, where $\frac{1+j}{2}$ is the average ranking within single dataset. This is called the average adjusting and does not impact the final conclusion of the ranking procedure. The current approach is chosen to avoid negative ranks.

Table 2 Experimental Root Mean Squared Error Results

	Abalone		Ailerons		Bike		Elevators		Fried		House8L		HyperA		MetroTraffic	
FIMTDD	3.05		0.089		113.33		5.64E-3		2.99		37965		4.62		1952	
ORTO	8.40		0.093		157.81		1.61E-2		14.93		38105		10.99		1982	
KNN 1	2.75		0.071		123.93		4.47E-3		3.71		46400		4.38		2083	
KNN 5	2.35		**0.059**		136.61		4.17E-3		2.73		39894		3.14		1949	
KNN 10	2.33		0.060		137.08		4.33E-3		2.68		39881		3.04		1944	
SOKNN	2.52		0.060		123.80		**4.16E-3**		2.72		40384		3.05		1922	
AMRules	2.28	(0.23)	0.333	(2E-6)	224.66	(25)	4.83E-3	(2E-3)	2.48	(0)	40705	(97)	**2.06**	(0.01)	3259	(2985)
ARF-Reg	2.45	(0.02)	0.074	(6E-4)	104.92	(1.70)	5.09E-3	(1E-4)	2.93	(0.04)	36118	(364)	4.70	(0.01)	1691	(2)
KNL 1	3.39	(0.01)	0.102	(3E-3)	139.49	(4.32)	6.07E-3	(8E-5)	2.98	(0.07)	39256	(760)	3.89	(0.02)	2249	(21)
KNL 5	2.73	(0.08)	0.096	(1E-3)	120.67	(2.62)	5.94E-3	(9E-5)	2.39	(0.06)	35219	(544)	3.04	(0.03)	1827	(8)
KNL 10	2.56	(0.04)	0.094	(1E-3)	116.41	(2.45)	5.88E-3	(1E-4)	2.29	(0.06)	35629	(423)	2.91	(0.03)	1781	(3)
SOKNL	**2.22**	(0.03)	0.065	(1E-3)	**96.69**	(2.74)	4.30E-3	(2E-4)	**2.23**	(0.05)	**35135**	(298)	2.97	(0.03)	**1557**	(5)

Table 3 Experimental Running Time (Seconds)

	Abalone		Ailerons		Bike		Elevators		Fried		House8L		HyperA		MetroTraffic	
FIMTDD	0.12	(0.03)	1.27	(0.04)	0.42	(0.07)	0.54	(0.05)	0.52	(0.06)	0.31	(0.04)	7.02	(0.15)	0.65	(0.06)
ORTO	0.27	(0.05)	1.53	(0.06)	0.39	(0.08)	1.99	(0.08)	1.20	(0.06)	0.28	(0.05)	9.26	(0.47)	0.49	(0.08)
KNN 1	0.44	(0.07)	6.49	(0.07)	2.76	(0.07)	3.66	(0.13)	4.72	(0.07)	10.37	(2.06)	56.64	(0.40)	5.08	(0.31)
KNN 5	0.49	(0.09)	7.14	(0.08)	2.94	(0.10)	4.08	(0.06)	5.53	(0.15)	10.50	(2.12)	66.87	(0.51)	5.54	(0.12)
KNN 10	0.49	(0.07)	7.50	(0.08)	3.09	(0.09)	4.24	(0.10)	5.84	(0.08)	10.75	(2.02)	70.39	(0.48)	5.84	(0.09)
SOKNN	1.52	(0.06)	13.02	(0.24)	6.47	(0.47)	8.15	(0.11)	13.83	(0.53)	6.98	(0.33)	167.05	(1.36)	13.20	(0.64)
AMRules	0.50	(0.14)	1.64	(0.53)	1.16	(0.17)	1.41	(0.64)	2.15	(0.36)	1.41	(0.33)	16.68	(0.93)	2022	(0.64)
ARF-Reg	6.72	(0.69)	82.88	(5.14)	30.23	(1.87)	20.58	(1.19)	28.56	(2.55)	25.93	(3.19)	537.50	(44)	59.37	(2.60)
KNL 1	6.33	(0.71)	84.11	(6.52)	32.35	(1.25)	21.04	(1.12)	27.88	(1.83)	25.63	(1.57)	738.29	(28)	58.50	(2.19)
KNL 5	6.42	(0.45)	84.20	(8.59)	32.13	(1.32)	20.86	(1.20)	27.82	(2.08)	25.62	(2.26)	739.41	(35)	58.62	(2.76)
KNL 10	6.46	(0.70)	83.98	(5.53)	32.27	(1.60)	20.95	(0.79)	28.13	(1.28)	25.71	(2.67)	741.50	(29)	59.01	(2.12)
SOKNL	7.65	(0.52)	90.21	(3.93)	36.26	(2.02)	24.64	(1.19)	36.85	(1.57)	30.79	(2.12)	845.25	(28)	71.49	(3.03)

in those cases. The most strange dataset is the HyperA where AMRules are better than SOKNL by almost 30%. Combining the information from Fig. 7, the RMSE is always better for AMRules than SOKNL before and after concept drifts. We currently do not have an explanation for this phenomenon. However, results from a single dataset can not shake the status of SOKNL on an overall view.

In terms of running time, as shown in Table 3, KNL and SOKNL are taking longer than ARF, which is expected, given the additional processing of leaf information in KNL and SOKNL. The results indicate that the additional time required by SOKNL is worthwhile considering the consequent improvement in prediction performance.

SOKNL is taking almost the same amount of memory as the fixed k values KNL as showing in Table 4, which means that the Self-optimising procedure is not occupying too much memory. For most datasets, the SOKNL maintains similar order of magnitude. Furthermore, the model only grows to about 1 Gigabytes with the largest dataset (HyperA). It proves that the memory utilisation of SOKNL is evidently acceptable for dealing with data streams.

Table 5 is a combined result illustration for both R^2 metric and the Quade test. Except for the last column, all the cells contain two numbers formed as $R^2(W_{ij})$ (see in Sect. 4.4.4). The last column exhibits the final ranks for all algorithms. The best result in each column is emphasised in bold font.

The ranking results for individual datasets are the same as RMSE. Nonetheless, comparison on various datasets is possible due to the R^2 metric. It is a simple conclusion that SOKNL achieves an R^2 score higher than 0.5 in almost all cases while other algorithms are struggling. Moreover, the best rank lies in the SOKNL, which indicates that SOKNL's competitiveness is eye-catching and evident even amongst these state-of-art online regression algorithms. One of the advantages of Quade test is that it provides the capability of ranking the datasets in terms of "difficulty". As a consequence, algorithms that gain better results on difficult tasks receive better total ranks. In Table 5, SOKNL achieves a very good ranking in the difficult datasets such as Fired and Abalone. In other words, SOKNL has more ability to conquer "hard" problems. The $p - value$ from this Quade test is $3.771e^{-5}$. Therefore, the H_0 is rejected.

Figure 2 and Fig. 3 demonstrate the RRSE of all KNN and KNL related algorithms. The error bars in Fig. 3 indicate standard deviations, as (SO)KNL is randomisable.

These two figures illustrate the comparisons inside the KNN and KNL families. SOKNL almost always outperforms KNL with fixed k values. This is a clear indication that the proposed self-optimising procedure works well most of the time. In terms of the SOKNN, which mimics the parameter self-tuning mechanism of SOKNL , the self-optimisation seems not to be working so well since only in two out of eight cases it outperforms the fixed-value KNN results. Our assumption is that for KNN, the historical information has not much influence on the latter predictions since KNN only considers the distances. On the other hand, due to its robustness, SOKNL not only considers the distances between instances but also is affected by the integrated decision trees.

Figure 4 reveals the relation of the results between ARF-Reg and the proposed SOKNL. The result values are computed by dividing the SOKNL values by the ARF-Reg values. For instance, the error ratio is $\frac{\text{RMSE}_{\text{SOKNL}}}{\text{RMSE}_{\text{ARF-Reg}}}$. Hence, the error ratio being

Table 4 Experimental Memory Use Estimation (GBs)

	Abalone		Ailerons		Bike		Elevators		Fried		House8L		HyperA		MetroTraffic	
FIMTDD	3.2E-03	2.2E-06	5.8E-03	2.7E-06	5.9E-03	2.6E-06	9.5E-03	2.2E-06	2.3E-02	2.7E-06	1.1E-02	2.5E-06	6.7E-02	2.7E-06	8.1E-03	2.6E-06
ORTO	9.8E-04	2.6E-06	5.8E-03	2.2E-06	2.2E-03	2.6E-06	9.3E-03	2.5E-06	3.8E-03	2.6E-06	3.3E-04	2.1E-06	1.2E-03	2.6E-06	8.4E-04	2.7E-06
KNN1	1.1E-03	1.2E-19	3.5E-03	1.4E-19	1.5E-03	1.4E-19	1.9E-03	2.1E-19	1.2E-03	1.0E-19	1.1E-03	1.4E-19	1.2E-03	1.1E-19	1.0E-03	0.0E+00
KNN5	1.1E-03	1.0E-19	3.5E-03	0.0E+00	1.5E-03	1.2E-19	1.9E-03	1.8E-19	1.2E-03	1.1E-19	1.1E-03	1.1E-19	1.2E-03	1.5E-19	1.0E-03	6.9E-20
KNN10	1.1E-03	1.1E-19	3.5E-03	1.8E-19	1.5E-03	1.4E-19	1.9E-03	0.0E+00	1.2E-03	9.1E-20	1.1E-03	1.1E-19	1.2E-03	9.1E-20	1.0E-03	9.1E-20
SOKNN	1.2E-03	1.0E-19	3.6E-03	2.7E-19	1.5E-03	1.5E-19	1.9E-03	1.0E-19	1.3E-03	1.5E-19	1.2E-03	6.9E-20	1.3E-03	6.9E-20	1.1E-03	1.1E-19
AMRules	8.0E-04	7.2E-20	1.1E-03	1.3E-07	1.4E-03	2.0E-07	1.1E-03	0.0E+00	2.3E-03	1.4E-19	9.8E-04	1.3E-07	2.5E-03	1.3E-07	1.5E-03	1.4E-04
ARF-Reg	5.4E-02	3.0E-03	7.7E-02	3.3E-03	1.8E-01	1.0E-02	1.2E-01	4.2E-03	1.6E+00	4.4E-02	6.7E-01	3.5E-02	8.9E-01	1.0E+00	1.1E-02	9.6E-04
KNL1	4.1E-02	4.1E-03	8.9E-02	2.1E-03	1.8E-01	1.1E-02	1.2E+00	5.1E-03	1.5E+00	2.1E-02	7.6E-01	4.5E-02	1.4E+00	1.8E-01	1.5E-02	1.3E-03
KNL5	4.1E-02	4.1E-03	8.9E-02	2.1E-03	1.8E-01	1.1E-02	1.2E+00	5.1E-03	1.5E+00	2.1E-02	7.6E-01	4.5E-02	1.4E+00	1.8E-01	1.5E-02	1.3E-03
KNL10	4.1E-02	4.1E-03	8.9E-02	2.1E-03	1.8E-01	1.1E-02	1.2E+00	5.1E-03	1.5E+00	2.1E-02	7.6E-01	4.5E-02	1.4E+00	1.8E-01	1.5E-02	1.3E-03
SOKNL	4.1E-02	4.1E-03	8.9E-02	2.1E-03	1.8E-01	1.1E-02	1.2E+00	5.1E-03	1.5E+00	2.1E-02	7.6E-01	4.5E-02	1.4E+00	1.8E-01	1.5E-02	1.3E-03

Table 5 Coefficient of Determination and the Quade Test Results

Q_i for Quade	Abalone 6		Ailerons 7		Bike 2		Elevators 5		Fried 8		House8L 1		HyperA 3.5		MetroTraffic 3.5		Ranks
FIMTDD	0.16	(60)	0.38	(49)	0.61	(6)	0.29	(40)	0.64	(80)	0.48	(5)	0.62	(35)	0.03	(28)	8.42
ORTO	−5.38	(72)	0.33	(56)	0.24	(22)	−4.77	(60)	−7.93	(96)	0.48	(6)	−1.16	(42)	0.01	(31.5)	10.71
KNN 1	0.32	(54)	0.61	(35)	0.53	(14)	0.56	(25)	0.45	(88)	0.23	(12)	0.66	(31.5)	−0.10	(35)	8.18
KNN 5	0.50	(24)	**0.73**	(7)	0.43	(16)	0.61	(10)	0.70	(56)	0.44	(8)	0.82	(24.5)	0.04	(24.5)	4.72
KNN 10	0.51	(18)	0.72	(21)	0.43	(18)	0.58	(20)	0.71	(40)	0.43	(9)	0.83	(17.5)	0.04	(21)	4.57
SOKNN	0.43	(36)	0.72	(14)	0.53	(12)	**0.6**	(5)	0.70	(48)	0.42	(10)	0.83	(21)	0.06	(3.5)	4.54
AMRules	0.53	(12)	−7.64	(84)	−0.53	(24)	0.48	(30)	0.75	(32)	0.41	(11)	**0.92**	(3.5)	−1.69	(56)	6.63
ARF−Reg	0.46	(30)	0.57	(42)	0.67	(4)	0.43	(35)	0.66	(64)	0.53	(4)	0.61	(38.5)	0.28	(7)	6.24
KNL 1	−0.04	(66)	0.19	(77)	0.41	(20)	0.18	(55)	0.64	(72)	0.45	(7)	0.73	(28)	−0.28	(38.5)	10.10
KNL 5	0.33	(48)	0.28	(70)	0.56	(10)	0.22	(50)	0.77	(24)	0.56	(2)	0.83	(14)	0.15	(14)	6.44
KNL 10	0.41	(42)	0.31	(63)	0.59	(8)	0.23	(45)	0.79	(16)	0.55	(3)	0.85	(7)	0.20	(10.5)	5.40
SOKNL	**0.55**	(6)	0.67	(28)	**0.72**	(2)	0.59	(15)	**0.80**	(8)	**0. 56**	(1)	0.84	(10.5)	**0.39**	(3.5)	**2.06**

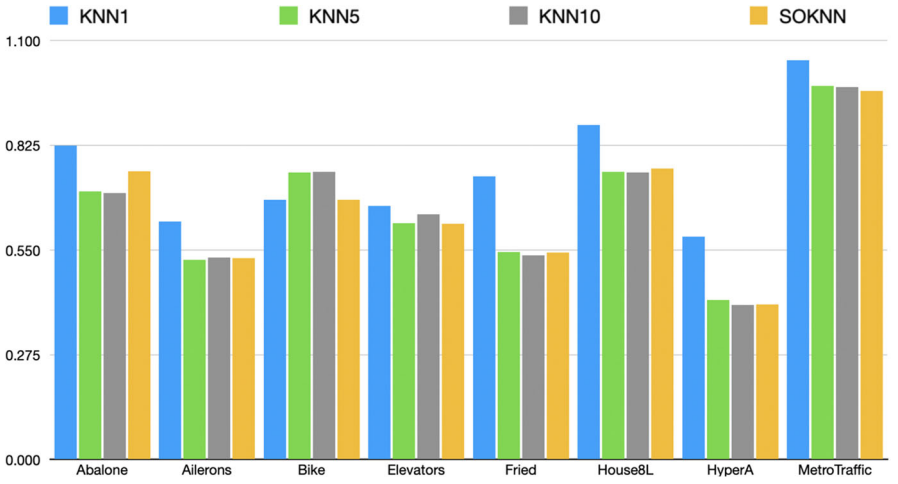

Fig. 2 KNN Related Root Relative Squared Error

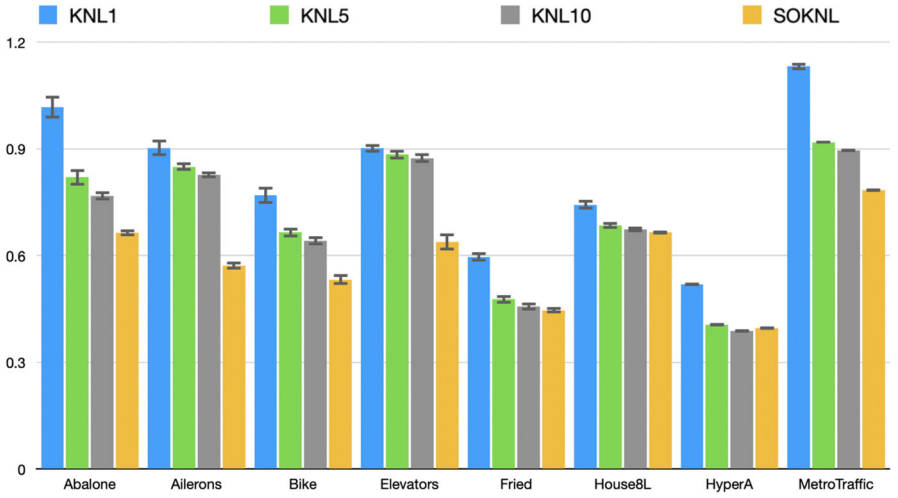

Fig. 3 KNL Related Root Relative Squared Error

smaller than 1 implies that SOKNL out-performs ARF-Reg. Time ratio, by the same logic, is $\frac{\text{Time}_{\text{SOKNL}}}{\text{Time}_{\text{ARF-Reg}}}$.

Comparing SOKNL to ARF-Reg shows a very clear picture: SOKNL outperforms ARF-Reg on all eight datasets but is also consistently slower. The additional computation is bounded, though, always less than a factor of two. The maximum performance improvement, on the other hand, can be up to a reduction in error by 50%, as seen in Fig. 4. One of our goals is achieved based on these results. Since SOKNL is an extension of the ARF-Reg, SOKNL is supposed to be better than ARF-Reg in most cases to make our modifications meaningful.

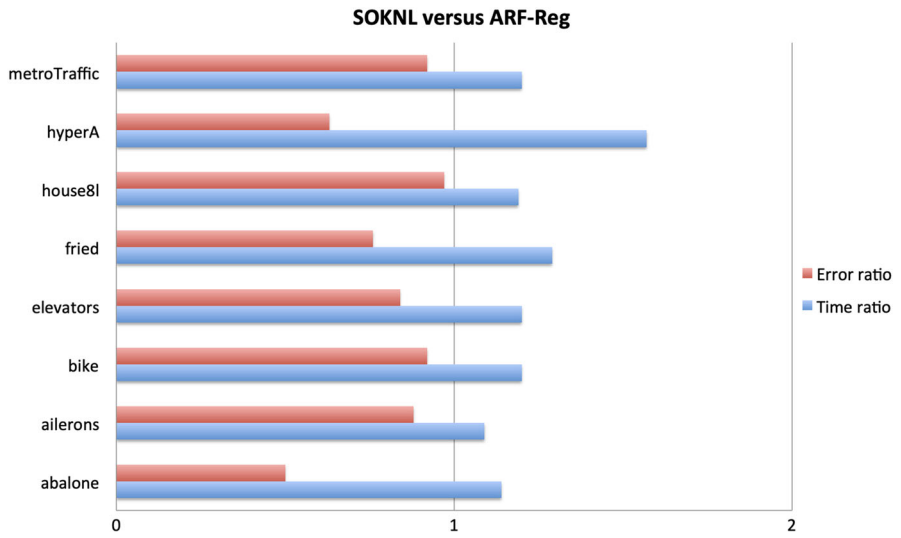

Fig. 4 SOKNL versus ARF-Reg

Figs 5 and 6 are the actual k values as chosen by the self-selecting mechanism over time. Since the amount of the instances in the datasets are practically large and in all cases the k values are having the tendency to converge to a practically small value, we use logarithmic scale on x-axis instead of an ordinary one to amplify the beginning portion. Moreover, since in almost all cases, the k values converge to a value around 10, we provide a red dashed horizontal line at $y = 10$ in each sub-figure for clearness.

In Fig. 5 it can be seen that after some fluctuation at the beginning, the k values converge to either one specific number or vary on a small subset of possible values. Interestingly, most values converge around 10, which motivated the experiments with fixed k in KNL. We can observe a very apparent unstable period of the k values in SOKNL algorithm, which we call a "learning period". These periods, in our eight experiments, end at around 10 instances, which is quite fast to "find a good" k value. Uniting with the performance of our algorithm, it is safe to say that the Self-Optimising procedure in SOKNL is working functionally. Figure 6 depicts a more erratic behaviour. It is evident that in each experiment's early stage, the k value has an increasing tendency. This stage ends around the 1000th instance, and then the value promptly converges in a way similar to SOKNL above. The strange stage coincides with the "window growing period" in KNN. In our assumption, the reason for this situation may be that while the window length increases, the prior errors of the new possible k value are voided. With the growth of the window, new evaluations of the new k numbers are added. However, when the sample quantity is small, there is a high possibility that the RMSE is tiny as well, which can result in a good change that newly added k values are selected. For instance, when the window length grows to 100 at a certain moment, the RMSE for $k = 100$ in the system is 0 since it is impossible to find 100 nearest neighbours in the window before that moment. Apparently, 0 is the smallest value the RMSE could get to. Thus, our self-optimising procedure tends

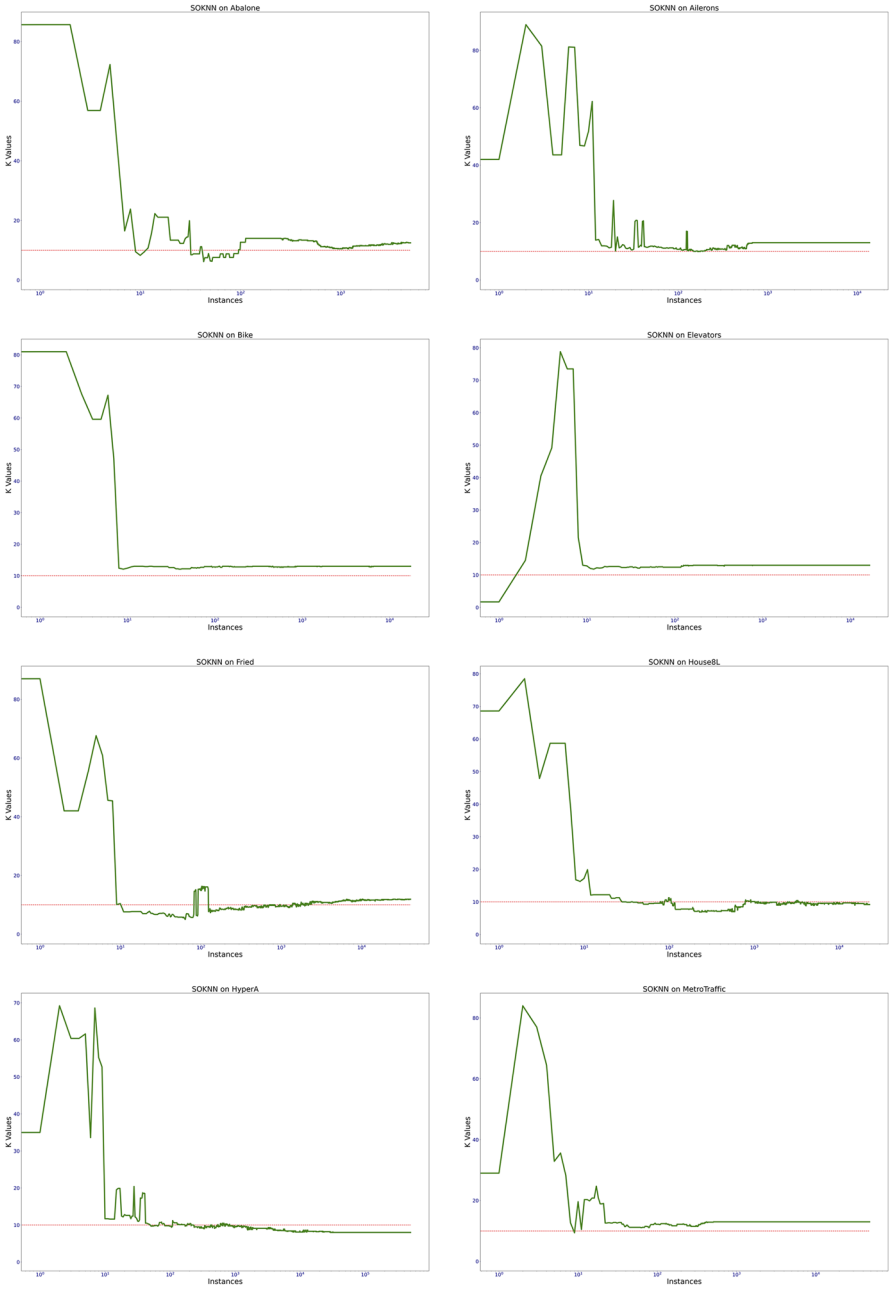

Fig. 5 K Values for SOKNL

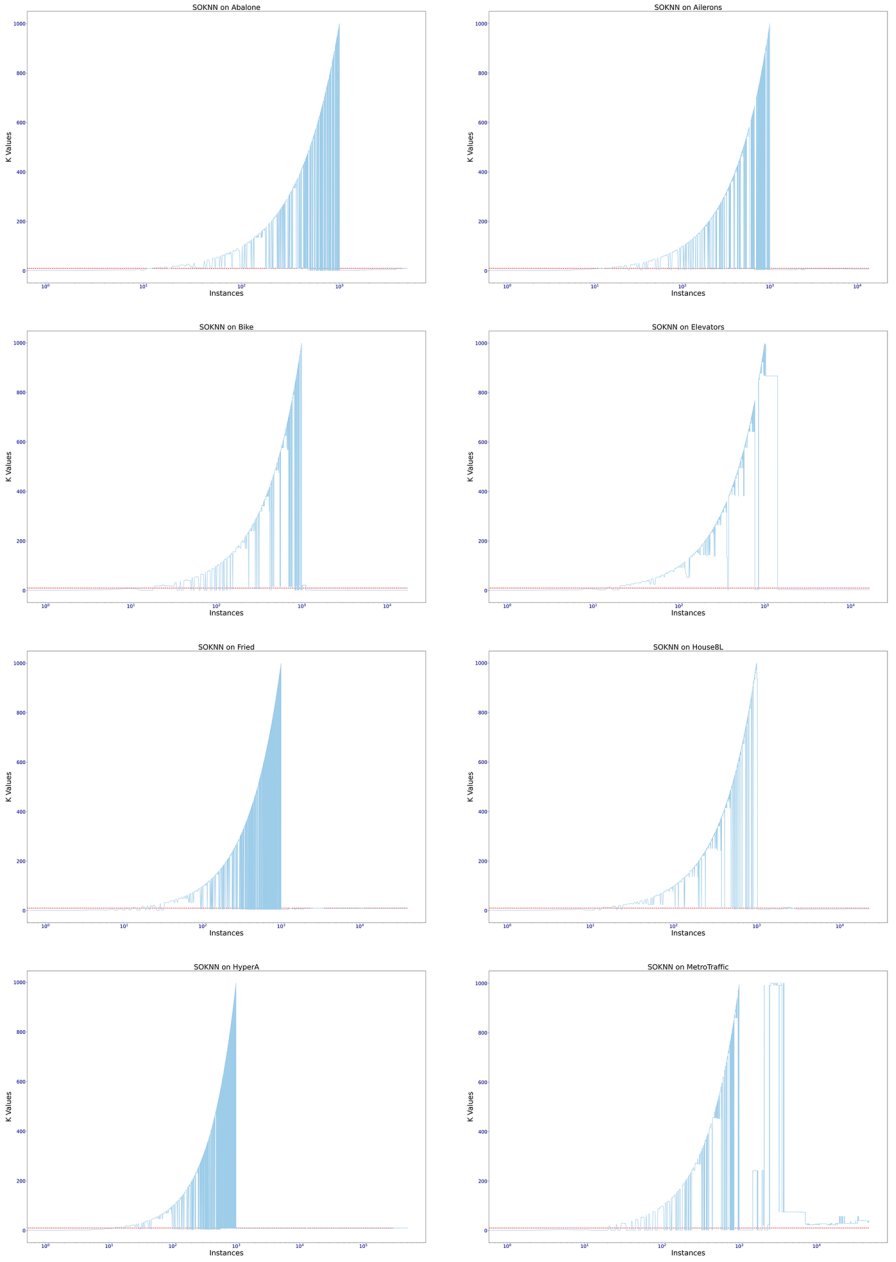

Fig. 6 K Values for SOKNN

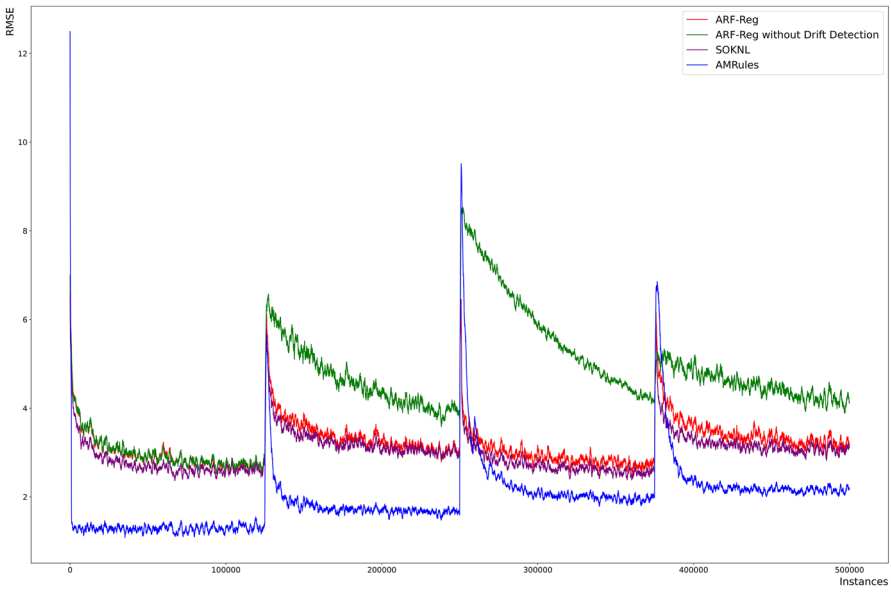

Fig. 7 RMSE Over Time on HyperA Dataset

to select the newly added possible k value. Whereas, if we look carefully into those figures, there are some fine structures in them, which indicates that not always the new values are selected. Therefore, it is normal and reasonable for our algorithm to behave in such way when applying to a sliding window approach. Admittedly, some amending measures, such as ignore a first period of the newly added k values to avoid the unstable phase, can be implemented, but our original intention of including SOKNN is to provide comparison to SOKNL. On account of this reason, we choose to present the authentic figures of this version of SOKNN.

Figure 7 illustrates the RMSE results over time (instances) for four algorithms: regular ARF-Reg (Red), ARF-Reg without Drift Detection (Green), SOKNL (Purple), and the AMRules (Blue). The associated dataset is HyperA. We include the AMRules algorithm because it achieves the best result on HyperA amongst all algorithms we experimented. Moreover, for the purpose of comparison, similar procedure has been conducted with the Fried dataset, whose results are exhibited in Fig. 8.

What demonstrates in Fig. 7 is the SOKNL's capability of detecting concept drifts. As aforementioned in Sect. 4.1, the HyperA dataset is structured with drifts at the position of $125K$, $250K$, and $375K$ instances. To show to results with clarity, the windowing RMSE is chosen, which is the RMSE for the recent certain number of predictions (we choose 1000 for the diagram). Also, the first 100 instances are ignored to skip the initial learning course where the information is insufficient hence the RSME tends to be meaninglessly high. In the figure, it is very distinct that the RMSEs of all four approaches have a growing tendency at the drift points. That is because the old models are not suitable for the new concept of the dataset. After the dramatic increases of RMSE at the drifting points, regular ARF, SOKNL, and AMRules are

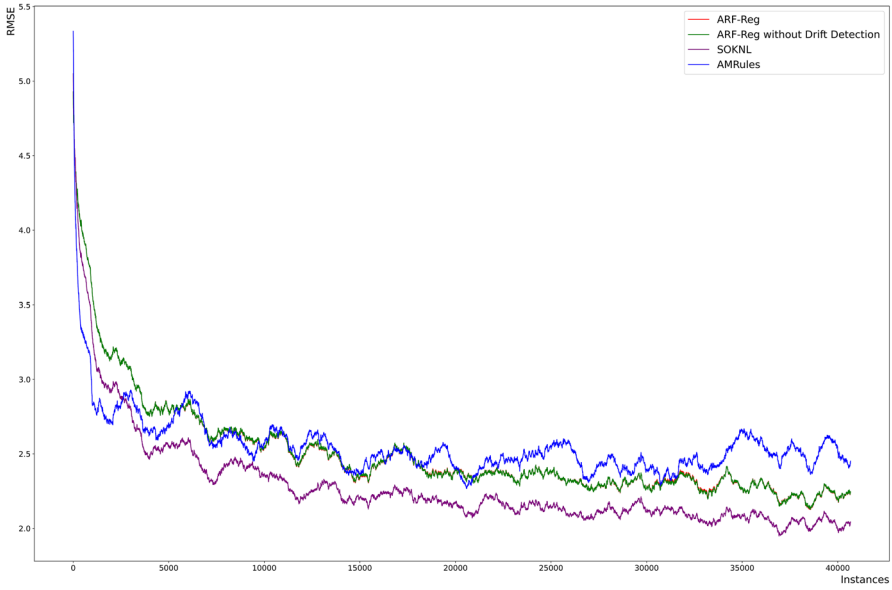

Fig. 8 RMSE Over Time on Fried Dataset

able to promptly diminish the performance back to the excellent level while the ARF-Reg without Drift Detection (which is the regular ARF-Reg with all drift detection methods disabled) struggling. It is worthy to point out that, although the ARF-Reg and SOKNL behave a little worse than AMRules in terms of RMSE, their speed of detecting and adapting to the drifts stays at the same level, sometimes even outperforming the AMRules. Whereupon, the capability of drift detection and adaptation of SOKNL are proven and evidenced. The SOKNL and the regular ARF-Reg act in a similar way due to their resembling internal drift detection method. Note that Fig. 8 is for comparison to Fig. 7. We can see in Fig. 8 that when the datasets have no drift, although the RMSEs fluctuate over time, there are no sudden growths in the diagram, which means all the trained models are capable of producing reasonable predictions. This fact testifies that in Fig. 7, the drift detection and adaptation procedures in SOKNL (and other algorithms) are appropriately contributing. What's more, the reason that the regular ARF-Reg line (red) in Fig. 8 is almost invisible is when no drift is happening, the ARF-Reg and ARF-Reg without drift detection (green) are supposed to have identical behaviours. Ergo, the green line and the red line in Fig. 8 are overlapping for nearly the whole time. In summary, SOKNL is capable of producing more accurate predictions compared to its original – ARF-Reg – and other online regression algorithms, with only limited more consumption on computing resources. SOKNL also maintains the capability of detecting and adapting to concept drifts. Furthermore, the Self-Optimising procedure for SOKNL is considerably effective in selecting more promising k values for the proposed algorithm.

6 Conclusions

This paper proposed a novel algorithm for data stream regression called SOKNL, which is a combination of k Nearest Neighbours and the Adaptive Random Forest. It integrates the merits of both KNN and ARF-Reg into one system, resulting in robust regression performance. Empirical results show that this approach can achieve more accurate predictions for a limited amount of additional computational resources.

Along with the new method, a hyperparameter self-tuning technique based on ongoing performance evaluation is implemented to make the algorithm more user-friendly as well as more robust to concept drift. The results prove that in many circumstances the self-selecting approach is capable of selecting well-performing k values.

Although the current self-optimising mechanism is effectively working for SOKNL, the outcomes of SOKNN are relatively unsatisfactory. One of the future works is the development of a different self-optimising approach for KNN. Other interesting directions include investigating similar KNN integration into other stream learners and explore how the density (measured by standard deviation) would affect the information gaining.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions

References

Almeida E, Ferreira C, Gama J (2013) Adaptive model rules from data streams. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp 480–492. Springer

Arthur D, Vassilvitskii S (2006) k-means++: The advantages of careful seeding. Technical report, Stanford

Bifet A, Gavalda R (2007) Learning from time-changing data with adaptive windowing. In: Proceedings of the 2007 SIAM International Conference on Data Mining, pp 443–448. SIAM

Bifet A, Holmes G, Kirkby R, Pfahringer B (2010) MOA: massive online analysis. J Mach Learn Res 11:1601–1604

Bifet A, Holmes G, Pfahringer B (2010) Leveraging bagging for evolving data streams. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp 135–150. Springer

Boulegane D, Bifet A, Madhusudan G (2019) Arbitrated dynamic ensemble with abstaining for time-series forecasting on data streams. In: 2019 IEEE International Conference on Big Data (Big Data), pp 1040–1045. IEEE

Breiman L (2001) Random forests. Mach Learn 45(1):5–32

Cerqueira V, Torgo L, Pinto F, Soares C (2017) Arbitrated ensemble for time series forecasting. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp 478–494. Springer

Chicco D, Warrens MJ, Jurman G (2021) The coefficient of determination r-squared is more informative than smape, mae, mape, mse and rmse in regression analysis evaluation. PeerJ Comput Sci 7:e623

Choudhary A, Jha P, Tiwari A, Bharill N (2021) A brief survey on concept drifted data stream regression. In: Tiwari A, Ahuja K, Yadav A, Bansal JC, Deep K, Nagar AK (eds) Soft Computing for Problem Solving. Singapore, Springer Singapore, pp 733–744
Dhanabal S, Chandramathi S (2011) A review of various k-nearest neighbor query processing techniques. International Journal of Computer Applications
Domingos P, Hulten G (2000) Mining high-speed data streams. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp 71–80
Friedman JH (1991) Multivariate adaptive regression splines. The Annals of Statistics, pp 1–67
Friedman M (1937) The use of ranks to avoid the assumption of normality implicit in the analysis of variance. J Am Stat Assoc 32(200):675–701
García S, Fernández A, Luengo J, Herrera F (2010) Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. Inf Sci 180(10):2044–2064
Gomes HM, Bifet A, Read J, Barddal JP, Enembreck F, Pfahringer B, Holmes G, Abdessalem T (2017) Adaptive random forests for evolving data stream classification. Mach Learn 106(9–10):1469–1495
Gomes HM, Barddal JP, Ferreira LEB, Bifet A (2018) Adaptive random forests for data stream regression. In: ESANN
Gomes HM, Montiel J, Mastelini SM, Pfahringer B, Bifet A (2020) On ensemble techniques for data stream regression. In: IJCNN. IEEE
Hoeffding W (1994) Probability inequalities for sums of bounded random variables. In: The Collected Works of Wassily Hoeffding, pp 409–426. Springer
Hoeffding W (1994) Probability inequalities for sums of bounded random variables. In: The Collected Works of Wassily Hoeffding, pp 409–426. Springer
Huang J, Rojas J, Zimmer M, Wu H, Guan Y, Weng P (2021) Hyperparameter auto-tuning in self-supervised robotic learning. IEEE Robot Autom Lett 6(2):3537–3544
Ikonomovska E, Gama J, Džeroski S (2011) Learning model trees from evolving data streams. Data Min Knowl Disc 23(1):128–168
Ikonomovska E, Gama J, Zenko B, Dzeroski S (2011) Speeding-up hoeffding-based regression trees with options. In: ICML
Krawczyk B, Cano A (2018) Online ensemble learning with abstaining classifiers for drifting and noisy data streams. Appl Soft Comput 68:677–692
Losing V, Hammer B, Wersing H (2018) Tackling heterogeneous concept drift with the self-adjusting memory (sam). Knowl Inf Syst 54(1):171–201
Louppe G, Geurts P (2012) Ensembles on random patches. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases pp 346–361. Springer
Lu J, Liu A, Dong F, Gu F, Gama J, Zhang G (2018) Learning under concept drift: A review. IEEE TKDE
Luo G (2016) A review of automatic selection methods for machine learning algorithms and hyper-parameter values. Network Modeling Analysis in Health Informatics and Bioinformatics 5(1):1–16
Mouss H, Mouss D, Mouss N, Sefouhi L (2004) Test of page-hinckley, an approach for fault detection in an agro-alimentary production system. In: 2004 5th Asian Control Conference (IEEE Cat. No. 04EX904) 2: 815–818. IEEE
Nash WJ, Sellers TL, Talbot SR, Cawthorn AJ, Ford WB (1994) The population biology of abalone (haliotis species) in tasmania. i. blacklip abalone (h. rubra) from the north coast and islands of bass strait. Sea Fisheries Division, Technical Report, 48:p411
Page ES (1954) Continuous inspection schemes. Biometrika 41(1/2):100–115
Quade D (1979) Using weighted rankings in the analysis of complete blocks with additive block effects. J Am Stat Assoc 74(367):680–683
Shaker A, Hüllermeier E (2012) Iblstreams: A system for instance-based classification and regression on data streams. Evol Syst 3(4):235–249
Veloso B, Gama J, Malheiro B (2018) Self hyper-parameter tuning for data streams. In: International Conference on Discovery Science, pp 241–255. Springer
Wright S (1921) Correlation and causation
Zhang T, Ramakrishnan R, Livny M (1996) Birch: An efficient data clustering method for very large databases. ACM SIGMOD Rec 25(2):103–114