UNIVERSITY OF TECHNOLOGY SYDNEY
Faculty of Engineering and Information Technology

STREAMING DATA REGRESSION

by

Hang Yu

A Thesis Submitted
in Fulfillment of the
Requirements for the Degree

Doctor of Philosophy

Sydney, Australia

November, 2020

Certificate of Authorship/Originality

I, Hang Yu declare that this thesis, is submitted in fulfillment of the requirements for the award of Doctor of Philosophy, in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Production Note:

Signature: Signature removed prior to publication.

Date: 10/11/2020

ABSTRACT

STREAMING DATA REGRESSION

Machine learning is a field of computer science that gives computers the ability to learn knowledge. Regression analysis is one of the most important tasks to address in the area of machine learning, and it is a form of predictive modeling technique that investigates the relationship between dependent and independent variables. However, most regression algorithms, whether against linear regression or nonlinear regression analysis, were designed based on batch datasets. Nowadays, technological advancements make it possible to access fast and potentially infinite data known as streaming data. In streaming data, the data is displayed in the form of sequences and can only be read once in a predetermined order, so batched regression algorithms cannot be used to process streaming data. The streaming algorithm is a new type of technique in machine learning. In streaming algorithms, data are processed sequentially as well and can be examined in only a few passes (typically just one).

However, as a novel learning technique, the streaming algorithm is still immature and imperfect for the regression problem. Firstly, most of the existing streaming regression algorithms only can address precise data; however, in many real-world applications, streaming data is generated under noisy environments. The noisy data impacts the learning process of many regression algorithms and thereby resulting in the performance of many algorithms decrease dramatically. Secondly, more studies on streaming data show that data distribution is nonstationary; it can change or evolve. Concept drift refers to this unpredictable change of data distribution in streaming data, and the performance of an algorithm becomes declines when concept drift occurs. Hence, concept drift in streaming data is also a factor that impacts the performance of streaming regression algorithms. Finally, in many real-world applications, the regression problem of streaming data becomes more complicated.

Two or more outputs instead of single output need to be predicted. However, multi-output regression, which corresponds to two or more outputs, has been discussed extensively for offline, static settings. Only a few works address how to solve this problem for streaming data. Motivated by this reasoning, our research on streaming data regression aims to conquer the aforementioned challenges.

In order to solve streaming data regression under a noisy environment, we propose a novel online regression algorithm, called online robust support vector regression (ORSVR). ORSVR is able to solve nonparallel bound functions simultaneously. Hence, the large quadratic programming problem (QPP) in classical v-SVR is decomposed into two smaller QPPs. An online learning algorithm then solves each QPP step-by-step. The results of a series of comparative experiments demonstrate that the ORSVR algorithm efficiently solves regression problems in streaming data, with or without noise, and speeds up the learning process. Furthermore, we also propose an online topology learning algorithm to filter noise data in the data preprocessing stage, called Gaussian membership-based self-organizing incremental neural network (Gm-SOINN). Gm-SOINN is an unsupervised learning algorithm and can learn a topology network to represent the data distribution accurately. The size of the topology network is much smaller than the size of the training data. In addition, Gm-SOINN utilizes the advantages of fuzzy logic, unlike other SOINN-based methods that allow only one node to be identified as a "winner" (the nearest node), Gm-SOINN allows for any node to be selected as the winner and uses a Gaussian membership to indicate the degree to which nodes are identified as winners.

In order to the streaming data regression problem under evolving environments, we propose continuous support vector regression (C-SVR) for nonstationary streaming data. Like an ensemble-based method, in C-SVR, a series of regression models are continuously learned in a series of time windows to determine the relationship between the input and output at different timestamps. Additionally, in contrast to algorithms that forget all learned knowledge, learning processes in different time windows are not independent in C-SVR. A similarity term added to the QPP carries some learned knowledge from the last model forward into the current model. The

problem of evolving streaming data regression has been a topic of consistent research in the fuzzy systems community. Hence, a novel evolving-fuzzy-neuro system, called the topology learning-based fuzzy random neural network (TLFRNN), is proposed. In TLFRNN, we revised our proposed Gm-SOINN to self-organize each layer of TLFRNN. However, different from current EFN systems, TLFRNN learns multiple fuzzy sets to reduce the impact of noises on each fuzzy set, and a randomness layer is designed, which assigning the probability of each fuzzy set. Also, TLFRNN does not utilize TSK rules; instead uses a simple inference that considering fuzzy and random information of data simultaneously. More importantly, in TLFRNN, concept drift can be detected and adapted easily and rapidly.

In order to solve the multiple-output regression problem of streaming data, we present an online multi-output regression system, called MORStreaming, for streaming data. MORStreaming uses an instance-based model to make a prediction because this model can quickly adapt to change by only storing new instances or by throwing away old instances. However, learning instances in our regression system is constrained by online demand, and need to consider the relationship between outputs. Hence, MORStreaming consists of two main algorithms: 1) an online learning instances algorithm based on topology networks was designed to make MORStreaming robust to noise and determines the number of instances. 2) an online learning structured-outputs algorithm based on adaptive rules was designed for MORStreaming to learn the correlation between outputs automatically.

In summary, our thesis describes original research into streaming data regression, a problem that is important but relatively under explored. The original contribution is made in 3 aspects: (i) dealing with noisy streaming data; (ii) dealing with evolving streaming data; (iii) dealing with streaming data with multiple outputs.

Dissertation directed by Professor Jie Lu
Australian Artificial Intelligence Institute

Dedication

To my parents Tao Yu and Li Guo, and my girlfriend, Yiqun Jiang

Acknowledgements

It has been an exciting and memorable journey at the University of Technology Sydney (UTS) in pursuit of a Ph.D. degree over the past four years. I am sincerely grateful to the people who inspired and helped me in many ways.

I would like to express my earnest thanks to my principal supervisor, Distinguished Professor Jie Lu. Without her help, I would not have been able to obtain this Ph.D. degree. She led me into a new academic research field and guided me to pursue my own research interests. In addition, her professional knowledge enlightened me and inspired me to delve further and deeper into my research, especially when I become lost in my research direction. I felt extremely honored to be guided by such a distinguished researcher as well as an enthusiastic mentor. The skills and knowledge she imparted over the past four years has benefited my Ph.D. study and will be a great treasure throughout my life. Meanwhile, I would like to express my foremost and deepest gratitude to my co-supervisor, A./Professor Guangquan Zhang. He taught me how to think and work as a professional researcher. He also unconditionally supported me in pursuing my own research interests from the beginning. Without his patience and encouragement, I would have wasted my time on trivial research ideas. My discussions with him greatly improved the scientific aspect and quality of my research. His strict academic attitude and respectful personality benefited my PhD study and will be a great memory throughout my life.

During my Ph.D. candidature, I was fortunate to join the Decision Systems & e-Service Intelligence Lab (DeSI) in the Australian Artificial Intelligence Institute (AAII). I greatly enjoyed the pleasurable and plentiful research opportunities. It was a wonderful experience to spend four years with these dedicated researchers. I especially thank Dr Junyu Xuan and Dr Anjin Liu, who helped me greatly to deeply understand my research problem during my Ph.D. candidature; Dr Feng Liu, Dr Yiliao Song and other students in DeSI lab who have shared their opinions and comments with me. I also genuinely thank Michele Mooney, who proofread certain sections of this thesis and Robyn Barden for polishing the language of my thesis and publications. They were always patient in answering all my questions about academic writing. Hence, I have learned much about academic writing from them. I also thank all my wonderful friends, classmates, and colleagues for every enjoyable moment.

Last, I would like to express my heartfelt appreciation and gratitude to my family, friends, classmates, and colleagues for their love and support.

Hang Yu
Sydney, Australia, 2020.

List of Publications

Journal Papers

J-1. **H. Yu**, J. Lu, and G. Zhang, “Online Topology Learning by a Gaussian Membership-Based Self-Organizing Incremental Neural Network,” *IEEE Transactions on Neural Networks Learning Systems*, pp. 1-15, 2019, (DOI: 10.1109/ TNNLS.2019.2947658).

J-2. **H. Yu**, J. Lu, and G. Zhang, “An Online Robust Support Vector Regression for Data Streams,” *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–14, 2020, (DOI: 10.1109/TKDE.2020.2979967).

J-3. **H. Yu**, J. Lu, and G. Zhang, “Continuous Support Vector Regression for Non-stationary Streaming Data,” *IEEE Transactions on Cybernetics*, pp. 1–14, 2020, (DOI: 10.1109/TCYB.2020.3015266)

J-4. **H. Yu**, J. Lu, and G. Zhang, “Topology Learning-based Fuzzy Random Neural Network for Streaming Data Regression,” *IEEE Transactions on Fuzzy Systems*, pp. 1–14, 2020, (Under Review)

J-5. **H. Yu**, J. Lu, and G. Zhang, “MORStreaming: A Multi-Output Regression System for Streaming Data,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–14, 2020, (Under Review)

J-6. **H. Yu**, J. Lu, and G. Zhang, “Detecting Group Concept Drift From Multiple Data Streams,” *Pattern Recognition*, pp. 1–14, 2020, (Under Review)

Conference Papers

C-1. **H. Yu**, J. Lu, and G. Zhang, ”Learning a fuzzy decision tree from uncertain data,” *Proc. Int. Conf. on Intelligent Systems and Knowledge Engineering*, pp. 1-7, Nov. 24-26, 2017.

C-2. **H. Yu**, J. Lu, and G. Zhang, "An incremental dual nu-support vector regression algorithm," *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 522-533, Jun. 3-6, 2018.

C-3. **H. Yu**, J. Lu, G. Zhang, and D. Wu, "A dual neural network based on confidence intervals for fuzzy random regression problems," *Proc. IEEE Int. Conf. on Fuzzy Systems*, pp. 1-8, Jul. 8-13, 2018.

C-4. **H. Yu**, J. Lu, and G. Zhang, "A Hybrid Incremental Regression Neural Network for Uncertain Data Streams," *Proc. IEEE Int. Joint. Conf. on Neural Networks*, pp. 1-8, Jul. 14-19, 2019.

Note: Chapeter 3 relates J-1 and J-2, Chapter 4 relates J-3 and J-4, Chapter 5 relates J-5

Contents

List of Figures

List of Tables

Abbreviation

SVR - Support Vector Regression

ORSVR - Online Robust Support Vector Regression

AONSVR - Accurate Online Support Vector Regression

INVSVR - Incremental nu-Support Vector Regression

TSVR - Twin Support Vector Regression

QPP - Quadratic Programming Problem

KKT - Karush-Kuhn-Tucker

RKHS - Reproducing Kernel Hilbert Space

SOINN - Self-Organizing Incremental Neural Network

E-SOINN - Enhanced SOINN

LB-SOINN - Load-balancing SOINN Gm-SOINN - Gaussian membership-based SOINN

KDE-SOINN - Kernel Density Estimation SOINN

LD-SOINN - Local Distribution SOINN GNG - Growing Neural Gas

TN - Topology Network

ART - Adaptive Resonance Theory

MAP - Mean Accumulated Point

eVQ - evolving Vector Quantization

eGMM - Evolving Gaussian Mixture Model

C-SVR - Continuous Support Vector Regression EFS - Evolving Fuzzy System

TSK - Takagi-Sugeno-Kang

eFN - Evolving-Fuzzy-Neuro

FCM - Fuzzy c-means

CB	Case-base
MVR	Mean-Variance-Ratio
ARMSE	Average Root Means Square Error
FLEXFIS	Flexible Fuzzy Inference System
DENFIS	Dynamic Evolving Neural-Fuzzy Inference System
RLS	Recursive Least Squares
FIMT-DD	Fast Incremental Model Trees Drift Detection
BSGD	Budgeted Stochastic Gradient Descent
AMRules	Adaptive Model Rules
IBLStreams	Instance-based Learning System for Streaming Data
ORTO	Online Option Trees For Regression
ARF-Reg	Adaptive Random Forest
Learn++.NSE	Adaptive batch-based ensembles
AddExp.C	Adaptive datum-based ensembles
MOR	Multiple Output Regression

Nomenclature and Notation

ε	is the ε-insensitive loss function and defined as $\lvert Y - f(X)\rvert\varepsilon = \max\{0, \lvert Y - f(X)\rvert - \varepsilon\}$ for a predicted value $f(X)$ and a true output Y, which does not penalize errors below some $\varepsilon > 0$, chose a prior. Thus, the region of all samples with $\{\lvert Y - f(X)\rvert \leq \varepsilon\}$ is called ε-tube.
K	is the kernel function
$\alpha_i - \alpha_i^*$	is the weight of $K(X_i, X_j)$
$*_{1i}$	is the ith * variable in the upper function
$*_{2i}$	is the ith * variable in the lower function
Δ	is the amount of the change of each variable
Q'_{ij}	is the sub-matrix of Q_{ij} after initial adjustment
$*'_{1i}$	is the ith * variable in the upper function after initial adjustment
$*'_{2i}$	is the ith * variable in the lower function after initial adjustment
$Q'_{s_sS_S}$	is the sub-matrix of Q' with the rows and columns indexed by S_S
$Q_{n_nS_S}$	is the sub-matrix of Q' with the rows and columns indexed by n_n
λ	is the number of samples during one learning period
$age_{\max}$	is the lifetime of each edge
AN	is set of all nodes
$\lVert\rVert$	is the Euclidian distance (L_2-norm)
W_i	is the n-dimensional weights vector of node i
nei_i	is the neighbour nodes of node i, i.e., the nodes that directly connect to node i

L_i	is the learning time of node i
ε	is the learning rate and usually be set to 100
p_i	is the point of node i
sp_i	is the sum of points of node i during a learning period
h_i	is the mean accumulated point (MAP)
Num_A	is the number of nodes in subclass A
$mean_A$	is the mean density of the nodes in subclass A
t	is the order in which the sample X inputted
$\mu_i(t)$	is the Gaussian membership of input sample $X(t)$ belongs to node i
G	is a Gaussian model
mv_G	is the mean vector of the G
Cov_G	is the covariance matrix of the G
num_G	is the winner times of the G
r_G	is a vigilance parameter to decide whether an input data belongs to the G

Chapter 1

Introduction

1.1 Background

Machine learning [7, 97] is a field of computer science that gives computers the ability to learn knowledge. Regression analysis [45, 118] is one of the most important tasks to address in the area of machine learning, and is a type of predictive modeling technique which investigates the relationship between the dependent and independent variables, such as stock price prediction, traffic flow forecasting, and the projections for sea temperatures, and after decades of research, regression analysis has made significant achievements. At present, regression analysis can be roughly divided into two categories [16, 144]: linear regression and nonlinear regression. However, both these regression algorithms were designed based on a batch dataset [125, 178]. Nowadays, technological advancements make it possible to access fast and potentially infinite data through a variety of sources such as social media feeds, cloud services, and sensors, known as streaming data [201], i.e. an ordered infinite sequence of data instances $\{(X_1, Y_1), (X_2, Y_2), \ldots, (X_{t-1}, Y_{t-1}), (X_t, Y_t), \ldots\}$. Each data instance (X_t, Y_t) is the sample of this data stream at timestamp t, where $X \in R^d$ is the input variable and $Y \in R$ is the output variable [148]. In streaming data, the data is displayed in the form of a sequence and can only be read once in a predetermined order [28], so traditional batched regression algorithms cannot be used to process it, and the need to discover new regression algorithms is becoming increasingly urgent.

In computer science, streaming algorithms are a new type of technique in machine learning [31]. In streaming algorithms, data are processed sequentially and can also be examined in only a few passes (typically just one) [123]. In addition, streaming algorithms only require limited memory and limited processing time per item. Hence, the streaming algorithm represents a dynamic technique of supervised learning and unsupervised learning that can be applied when training data becomes available gradually over time, or its size is out of the system memory limits [129]. The aim of streaming algorithms is to adapt the learning model to new data without forgetting its existing knowledge, so it does not retrain the model. However, as a novel learning technique, streaming algorithms are still immature and imperfect for solving the regression problem. Firstly, most of the existing streaming regression algorithms can only handle precise data, but streaming data in many real-world applications contain a lot of noisy data [167]. The reason for this is the environment in which streaming data is generated is commonly noisy environments. The noisy data impacts the learning process of many streaming regression algorithms, thereby resulting in a dramatic decline in the performance of many streaming regression algorithms. Secondly, an increasing number of studies on streaming data show that the data distribution is nonstationary [36]; it can change or evolve. Concept drift [73, 181] refers to this unpredictable change of data distribution in streaming data. For example, one concept in weather data is the season which is not explicitly specified in temperature data but may influence temperature data. Another example is customer purchasing behaviour over time which may be influenced by the strength of the economy, where the strength of the economy is not explicitly specified in the data. The performance of a regression algorithm may become a decline when the concept drifts occurs[132]. Hence, concept drifts in streaming data also impact the performance of streaming regression algorithms. Finally, in many real-world applications, the regression problem of streaming data becomes more complicated. Two or more outputs instead of a single output need to be predicted, i.e., multi-output regression [104]. However, multi-output regression only has been extensively discussed in relation to offline, static settings [186]. Few works investigate a solution to this problem for streaming data. Motivated by this, our research on streaming data regression aims to solve the aforementioned challenges.

1.2 Research Objectives

The objectives of this research are to:

- **Propose a streaming regression algorithm for noisy streaming data.**

  Most of the existing streaming regression algorithms only address precise data, but for noisy streaming data, the information of data with noise features, so we design a streaming regression algorithm to address noisy data. Hence, to deal with the noisy streaming data, the biggest challenge is that noisy data cannot be identified since we cannot get prior knowledge of training data. Thus, we will design an adjust method robust to noise to dynamically adjust our model when an instance arrives.

- **Propose a streaming regression algorithm for evolving streaming data.**

  The status of streaming data streams can evolve in real-world applications; that is, concept drift will exist in streaming data. Thus, a streaming algorithm not only needs to adjust the model instance-by-instance; it also needs to detect and react to concept drift. However, in classification/clustering cases, the definition of concept drift is represented by inequality between joint probability density functions (PDF) of features and the label at two different time points [147]. Most existing concept drift related techniques are built based on this PDF definition. However, the definition of concept drift is less applicable in regression cases as the dependent variable is continuous, and its probability of being any single value is 0. Thus, how to detect and adapt concept drift is a critical issue.

- **Propose a streaming regression algorithm for streaming data with multiple-outputs.**

  With the arrival of the big data era, streaming data regression has become more complicated when handling large volumes of streaming data. Multi-output regression, a model that predicts two or more numerical values, is commonly used in many applications. Compared with single-output regression, multi-output regression is more complicated because these outputs may have a structure to represent the relationship between outputs [88]. Hence, the biggest challenge is how to learn the structure of outputs in an online manner. However, the structure of outputs is unknown, since we cannot obtain all training data, and the structure of outputs also can change due to concept drift.

1.3 Research Contributions

The contributions of this research are to:

- **Proposed an online robust support vector regression.**

  The contributions of this algorithm are summarized as follows:

  1) to handle noisy data, we modified the classical v-SVR as the regression model of ORSVR. In ORSVR, the QPP in a classical v-SVR [29] is transformed into two QPPs. Then, through the new regression model, ORSVR seeks up- and down-bound functions by solving two related OPPs simultaneously. Each new QPP is smaller than the large QPP in classical v-SVR, which means the KKT conditions for each bound are simpler. As a result, ORSVR learns faster than standard incremental v-SVR or

incremental ε-SVR algorithms. ORSVR is also good at capturing the characteristics of data distributions due to the modification of classical v-SVR, so it is very useful for handling noise. Moreover, the correlation between the box constraints and the size of the training sample set in the classical formulation of v-SVR [89] makes it difficult to design an incremental learning algorithm. However, with ORSVR, the box constraints are independent of the sample set size, which means ORSVR can learn online.

2) to propose an incremental learning algorithm based on KKT conditions that ensures ORSVR can solve the QPPs in a sequential manner. The basic idea is similar to [180] where the weight α_c is changed according to a new sample (X_t, Y_t) in an infinite number of discrete steps until it meets the KKT conditions while ensuring that the existing samples continue to satisfy the KKT conditions at each step [48]. However, ORSVR contains one additional complex equation constraint over ε-SVR [170], so an easy initial adjustment needs to be made to ensure the KKT conditions are still met when no new sample arrives. Using the incremental learning algorithm to seek the upper function and lower functions simultaneously, ORSVR can quickly and effectively handle data streams with noise.

- **Develop a Gaussian membership-based self-organizing incremental neural network.**

  The contributions of this algorithm are summarized as follows:

  1) to integrate fuzzy logic [110] into the online learning process of topological structure for the first time. Based on fuzzy logic, the result of Gm-SOINN shows it is not sensitive to the parameters λ and $age_{\max}$.

  2) to propose an incremental density estimation method called evolving

Gaussian mixture model ($eGMM$). The $eGMM$ method is easily integrated into Gm-SOINN. Compared with the MAP method of enhanced SOINN [68], $eGMM$ does not require to set any parameters manually and is more able to avoid building wrong connections between nodes.

3) to propose several innovations in the details of topology learning [67], including Gaussian membership-based criteria, to identify whether a node needs to be inserted, a recursive method to update the nodes, and a new method to delete noise. Using these technical innovations, the topological structure obtained by Gm-SOINN is robust to noise.

- **Propose a continuous support vector regression.**

  The contributions of this algorithm are summarized as follows:

1) to propose a continuous learning strategy for SVR where $f(X)$ is learned from new data, and some outdated learned knowledge is forgotten. Hence, $f(X)$ is not fixed; it varies in different time windows tw_i to adaptively suit the drift.

2) to propose a new definition of a QPP that incorporates a similarity term to transfer learned knowledge from $f_{i-1}(X)$ into $f_i(X)$. Therefore, learning a new $f_i(X)$ is dependent on $f_{i-1}(X)$, which prevents catastrophic forgetting of learned knowledge of $f_{i-1}(X)$ [119].

3) to propose an online way of solving the QPP of C-SVR based on KKT conditions and one datum per time step, which means C-SVR can handle streaming data.

4) to propose an automated method of deciding how much learned knowledge in $f_{i-1}(X)$ needs to be transferred to $f_i(X)$. As a result, C-SVR can forget different degrees of learned knowledge according to different types of concept drift.

- **Develop a novel evolving-fuzzy-neuro system, called topology learning-based fuzzy random neural network.**

  The contributions of this algorithm are summarized as follows:

  1) to propose a fast-online clustering method that inherits the idea of our proposed Gm-SOINN. In our proposed method, neurons in the fuzzy set layer can be self-organized to a topology network after a single pass scan of the training dataset, and robust to noise in the learning process. As a result, the structure of neurons in the fuzzy set layer can accurately represent underlying data distribution.
  2) to propose a new type of structure for the evolving-fuzzy-neuro system [15, 169] based on the topology network. In this new type of structure, multiple fuzzy sets can be included, and each fuzzy set is assigned to a probability by a randomness layer. As a result, although one of the fuzzy set has not been properly determined, the prediction accuracy of the system is still guaranteed because we can utilize other fuzzy sets to make an inference.
  3) to propose a mechanism to deal with the concept drift based on the topology network. Unlike current evolving-fuzzy-neuro systems, our mechanism does not utilize error rate to detect concept drift but changes in the topology network, and thereby making TLFRNN detect and adapt to concept drift rapidly and easily.
  4) to propose a new type of fuzzy rules based on the topology network. In TLFRNN, determining fuzzy rules does not need to split data spaces of input-output variables to obtain the subspaces that can approximate to a Takagi-Sugeno-Kang (TSK) rule [183]. In contrast, fuzzy rules are determined only by one parameter, and this parameter can be automatically

optimized by a maximum likelihood process.

- **Propose an online multi-output regression system called MORStreaming.**

  The contributions of this algorithm are summarized as follows:

  1) to propose an online learning structured-outputs algorithm based on adaptive rules. The rules, based on Hoeffding's bound [17], are not only learned online but also fit the drift of the structured-outputs. Based on these types of rules, this algorithm does not fall into the local and global categories [186]. Instead, each rule specializes in related subsets of the outputs to represent different types of structures.
  2) to develop an online learning instances algorithm based on our proposed Gm-SOINN. This algorithm shows high robustness to noise and concept drift because it inherits the advantages of topology learning [67]. In addition, when the algorithm is used to obtain the instances, it does not need to set the number of instances, and obtaining and discarding instances is also in a self-starting scheme.
  3) to construct an instance-based regression model. In this model, only instances that have the same structure as the outputs of the predicted data can be used to make a prediction. Furthermore, the correlation of outputs is utilized well by a multi-output nonparametric function. Hence, the accuracy of the prediction improves without significantly increasing learning time.

1.4 Research Significance

The theoretical and practical significance of this research is summarised as follows:

- **Theoretical significance:** This research investigates the regression problem of streaming data: several effective streaming regression algorithms are proposed, which supplements the characteristics of streaming regression algorithms and thus guarantees the validity of the streaming regression algorithms in principle; a formal definition of streaming data with sequence, infinite size, noisy data, and evolving data distribution are proposed, which lists the key characteristics of streaming data. This further facilitates studies on streaming data. The concept drift problem is divided and conquered by a set of drift adaptation problems. More specifically, as the core of concept drift adaptation is to update predictors, the proposed ideas could enrich the approach for powerful and meaningful manipulations on updating a predictor. This research also contributes to the theoretical analysis of drift detection by density-based statistics by learning a topological structure of its data distribution. Meanwhile, this research explores the adaptation of concept drifts for more complex scenarios, such as adaptation under noisy data, and adaptation under temporal dependency. The extended scenarios not only improve the impact of this area, but also motivate its continuing development. Meanwhile, this also extends the application scenario of the multi-output prediction theory.

- **Practical significance:** The findings of this research contribute to the benefit of society given the increasing demand for real-time prediction in modern life. This study develops a set of streaming regression methods to improve the prediction accuracy of streaming data. The first streaming regression method can identify different patterns and the degree to which each instance belongs to each pattern, which can also be applied to pattern recognition. The second streaming regression method is to select the most relevant data for learning predictors, which can also be used as a dissimilarity measurement and multivariate two-sample test. The third regression method generates new prediction models from the previous prediction models, which can also be used for transfer learning. Meanwhile, the network structure of the fourth method makes it easy to embed in other adaptive neural networks. Meanwhile, one method is developed in more realistic scenarios of data with multiple outputs. The findings help resolve the real-world problems of online decision making. There is the potential of other online applications could benefit from this study.

1.5 Thesis Organization

This thesis is organized as follows:

- **Chapter 1**: This chapter mainly introduces the research objectives, research contributions, and research significance of this work.
- **Chapter 2**: This chapter studies the literature of streaming data, concept drift, and streaming data regression, thereby revealing the current research gap. In this chapter, the definition of streaming data and the process of streaming algorithms were introduced at first. The concept drift problem and basic procedure of concept drift detection and adaptation are then introduced, after which categorization of the existing algorithms based on their implementation details are given. At last, the limitations of the reviewed streaming regression algorithms are discussed, which inspires the following chapters and solutions.
- **Chapter 3**: This chapter proposes a novel online regression algorithm, which is called online robust support vector regression (ORSVR). ORSVR is able to solve nonparallel bound functions simultaneously. Hence, the large quadratic programming problem (QPP) in classical v-SVR is decomposed into two smaller QPPs. An online learning algorithm then solves each QPP step-by-step. The results of a series of comparative experiments demonstrate that the ORSVR

algorithm efficiently solves regression problems in streaming data, with or without noise, and speeds up the learning process. Besides, this chapter proposes an online topology learning algorithm to filter noise data in the data preprocessing stage, called Gaussian membership-based self-organizing incremental neural network (Gm-SOINN). Gm-SOINN is an unsupervised learning algorithm and can learn a topology network to accurately represent the data distribution. The size of the topology network is much smaller than the size of the training data. In addition, Gaussian membership is a common kind of fuzzy logic membership. To utilize the advantages of fuzzy logic, unlike other SOINN-based methods that allow for only one node to be identified as a "winner" (the nearest node), Gm-SOINN allows for all nodes can be selected as the winner and uses a Gaussian membership to indicate the degree to which nodes are identified as winners.

- **Chapter 4**: This chapter presents a continuous support vector regression (C-SVR) for nonstationary streaming data. Like an ensemble-based method, in C-SVR a series of the regression model are continuously learned in a series of time windows to determine the relationship between the input and output at different timestamps. However, only one regression model is saved in memory to make a prediction. A new regression model is learned in the new time-window, and the old model, which was learned in the last time-window, is discarded. Additionally, in contrast to algorithms that forget all learned knowledge, learning processes in different time windows are not independent in C-SVR. A similarity term added to the QPP carries some learned knowledge from the last model forward into the current model. How much-learned knowledge is transferred depends on the degree of the concept drift. Further, because the data in nonstationary streaming data arrive sequentially, the QPP in C-SVR is solved incrementally. The problem of evolving streaming data regression has been a topic of consistent research in the fuzzy systems community. Hence, a novel evolving-fuzzy-neuro system, called the topology learning-based fuzzy random neural network (TLFRNN), is proposed in this chapter. In TLFRNN, we revised our proposed Gm-SOINN to self-organize each layer of TLFRNN. However, different from current EFN systems, TLFRNN learns multiple fuzzy sets to reduce the impact of noises on each fuzzy set, and a randomness layer is designed, which assigning the probability of each fuzzy set. Also, TLFRNN does not utilize TSK rules; instead uses a simple inference that considering fuzzy and random information of data simultaneously. More importantly, in TLFRNN, concept drift can be detected and adapted easily and rapidly. The experiments demonstrate that TLFRNN achieves superior performance compared to other EFSs.

- **Chapter 5**: This chapter proposes an online multi-output regression system, which is called MORStreaming, for streaming data. MORStreaming uses an instance-based model to make a prediction because this model can quickly adapt to change by only storing new instances or by throwing away old instances. However, learning instances in our regression system are constrained by online demand and need to consider the relationship between outputs. Hence, MORStreaming consists of two main algorithms: 1) an online learning instances algorithm based on topology networks was designed to make MORStreaming robust to noise and determines the number of instances; 2) an online learning structured-outputs algorithm based on adaptive rules was designed for MORStreaming to learn the correlation between outputs automatically. Experiments involving both artificial and real-world datasets indicate our proposed MORStreaming can achieve superior performance compared with other methods.

- **Chapter 6**: A brief summary of the thesis contents and its contributions are given in the final chapter. Recommendation for future works is given as well.

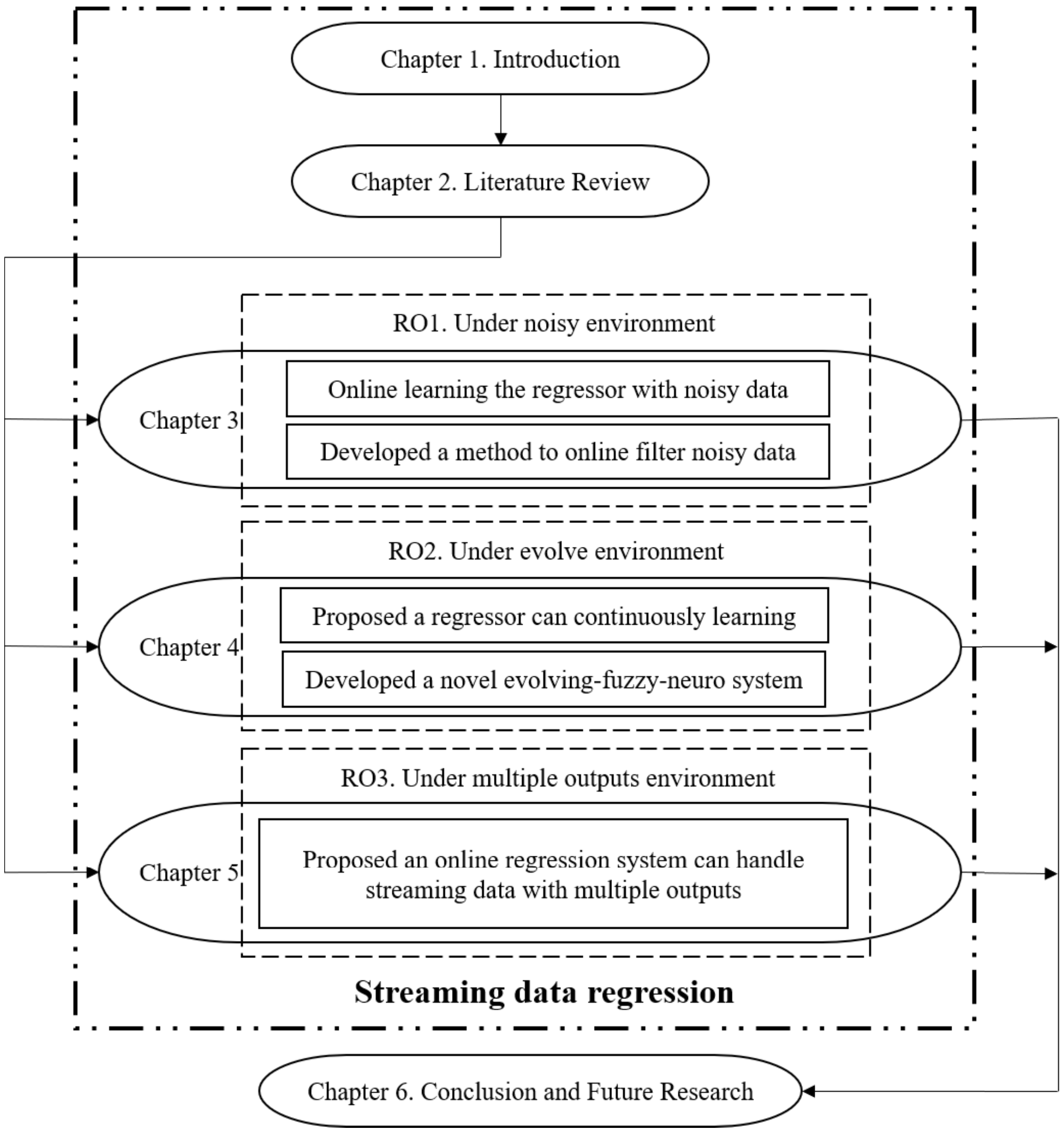

Chapter 1. Introduction
Chapter 2. Literature Review
RO1. Under noisy environment
Chapter 3
Online learning the regressor with noisy data
Developed a method to online filter noisy data
RO2. Under evolve environment
Chapter 4
Proposed a regressor can continuously learning
Developed a novel evolving-fuzzy-neuro system
RO3. Under multiple outputs environment
Chapter 5
Proposed an online regression system can handle streaming data with multiple outputs
Streaming data regression
Chapter 6. Conclusion and Future Research

Figure 1.1 : Thesis structure

Chapter 2

Literature Review

This study focuses on solving the regression problem of streaming data. This chapter reviews related research in this area that have recognized the significance of this problem or provided solutions to this problem. Section 2.1 introduces streaming data and streaming algorithms. Then, the concept drift problem is introduced, including its definition, types, and applications in Section 2.2. Regression algorithms for streaming data are comprehensively reviewed in Section 2.3.

2.1 Streaming Data Mining

This section gives a general review of data stream mining, which is the basis for understanding the problems of concept drift profoundly.

2.1.1 Streaming Data

Big data is an outcome of the current information explosion that is relevant to a diverse range of fields in the natural, life, social, and applied science, including physics, biology, medicine, economics, and management [184, 76]. Big data has been widely characterized by the three characteristics [101]: a hugely increased volume of data, a variety of data sources and quality, and the high velocity at which data is generated or obtained [32]. Big data technology holds incredible promise for improving people's lives, accelerating scientific discovery and innovation, and instigating positive societal change [201]. Meanwhile, new challenges accompanying the heterogeneity, incompleteness, scale, timeliness, privacy, and process complexity of big data, including aspects of data acquisition, data storage, information extraction, and big data analysis, need to be overcome [141, 117]. Further, three Vs are now recognized as the development of big data analysis: Veracity, which focuses on the unreliability inherent in data sources; Variability, which refers to variations in data flow rates; and Value, which refers to the issue of low-value density [74, 57, 50].

The recent development of the Internet of Things brought serious challenges to big data that data arrives in as continuous streams [130], namely the data stream, which consists of multiple infinite and fast-evolving data series [102, 44, 90]. Eight challenges in data stream mining were discussed in [115], covering the cycle of knowledge discovery from data. These challenges are summarized from three aspects [133]: 1) the development of new data mining skills for data streams; 2) the development of simpler, self-adaptive machine learning algorithms; and 3) the requirements of privacy and confidentiality for gaining the trust of the users and society in the system.

In recent years, data stream mining [160, 22] has been extensively studied in growing fields of multidisciplinary research, including databases, artificial intelligence, machine learning, automated scientific discovery, statistics, decision making, and so on [84]. The main challenges in learning the evolving data stream have been categorized into three kinds [152]: 1) concept drift refers to class boundaries change over time or the distribution of feature changes [197]. For example, in dressing recommend systems, the model is trained to recognize whether the dressing in a given photo is fashion or not. The characteristics of fashion change fast, and therefore the classification boundaries between "fashion" and "not fashion" changes; 2) feature evolution, which denotes that new features appear or feature type changes [94]. For example, the assessment model may have new features when the bank credit assessment system updates to include more details of the customers; 3) concept evolution, represents the situation that novel classes emerge [199]. For example, new topics appear, and old topics disappear or reappear in the bibliometric field, such as the novel topic of "blockchain" in 2013 and the recurrent novel class of "neural network".

2.1.2 Streaming Algorithms

As data evolves over time, the validity and reliability of the historical data are questionable. Streaming data mining [112, 105] has to consider these issues to perform accurate, up-to-date, real-time analysis—for example, the detection of highway flooding [159]. Real-time prediction is one of the most important applications of data stream mining that is widely demanded in the real world [69], which is to make predictions in real-time [99]. It is clearly different from the prediction in a stationary setting because each instance in a data stream is first used to test the learned predictor, and then to train/update the predictor.

Real-time prediction for a non-stationary data stream arises in many scenarios, such as online transactions in the financial market, weather forecasting, air quality prediction, and so on [157, 120]. For example, 20 years ago, TV was the main source of weather forecast information for most people. Additionally, weather forecasts would simply indicate the weather for tomorrow or the day after tomorrow at most. Today, people expect hourly weather reports and weather forecasts for a week in advance. Historically, a numerical weather prediction model was used to compute long-term weather variables. However, such models are not flexible enough to make hourly weather predictions. Nor can they be applied to a short-term, high-frequency online forecast system, as it takes a significant amount of time for the models to compute the necessary partial differential equations.

Conventional machine learning methods are not applicable to make a real-time prediction for streaming data because the prediction accuracy of the learner predictor is deteriorated due to the evolving nature of streaming data that future data may exhibit different patterns from those of the previous data used to learn the predictor. Although streaming data mining has become important research topics during the last decade, a truly autonomous, self-maintaining, adaptive data mining system is still lacking [114]. The short lifespan of data restricts us from storing and accessing all historical data during each processing cycle; however, processing accuracy has been strictly limited by the fact that the data can be accessed only once (one-pass setting) [3].

2.2 Concept Drift

Concept drift [92] is a particularly important factor in data stream mining. In this section, the concept drift problem and corresponding state-of-the-art solutions are reviewed.

2.2.1 Definitions and Types of Concept Drift

Given the *target variable* $\boldsymbol{y}$, in traditional machine learning models, the probability function or the probability distribution of y, $f(\boldsymbol{y})$, is assumed to be stationary. Under this condition, statistical theory guarantees the error(discrepancy between the predicted and real value) will decrease and thus machine learning can precisely make predictions with convergence by continuously updating itself based on data [72]. In a high-speed data stream, however, the distribution function rewritten as $f_t(\boldsymbol{y})$, varies with time which may lead to increased error. Usually, in a machine learning model, the data stream also includes corresponding *feature vectors* as $\boldsymbol{x}$. Therefore, concept drift is often defined in Eq. (2.1).

$$f_t(\boldsymbol{x}, \boldsymbol{y}) \neq f_{t+\tau}(\boldsymbol{x}, \boldsymbol{y}) \tag{2.1}$$

where τ represents the degree of time changing.

The joint probability density function $f_t(\boldsymbol{x}, \boldsymbol{y})$ can be decomposed into two parts as follows:

$$f_t(\boldsymbol{x}, \boldsymbol{y}) = f_t(\boldsymbol{x}) \times f_t(\boldsymbol{y} \mid \boldsymbol{x}) \tag{2.2}$$

Each part in Eq. (2.2) represents a kind of source of concept: $f_t(\boldsymbol{x})$ means the distribution of feature vectors has changed and $f_t(\boldsymbol{y}|\boldsymbol{x})$ is the conditional probability of the target variable given feature vectors. Concept drift is divided into two types by decomposition: *virtual drift* and *real concept drift* [73]:

(1) Virtual drift refers to changes only in the incoming data, namely $f_t(\boldsymbol{x})$. It doesn't affect $f_t(\boldsymbol{y}|\boldsymbol{x})$.

(2) Real concept drift refers to changes in $f_t(\boldsymbol{y}|\boldsymbol{x})$.

Figure 2.1 from [73] gives a clear insight into the two types of drift. It can be seen that the critical difference between real concept drift and virtual drift is whether the dotted line changes. The green dots move in both virtual and real concept drift cases, but the dotted line only shifts in the case of real concept drift.

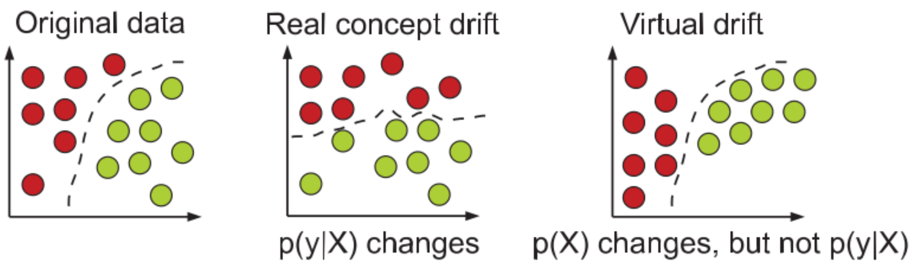


Figure 2.1 : Types of drifts: the red and green circles represent instances of different classes.

Furthermore, concept drift can be recognized as various patterns including *sudden/abrupt*, incremental, gradual, reoccurring drift as shown in Figure 2.2 [73].

2.2.2 Detecting Concept Drift

According to the above section, concept drift has many different patterns, and usually, the detection methods differ for different patterns. Most of the research in this area to date is related to sudden drift [135]. In some papers, outlier detection

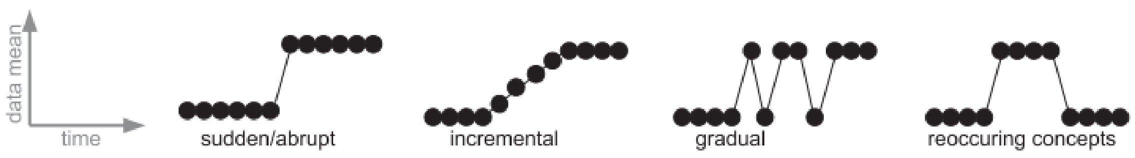


Figure 2.2 : Patterns of changes over time.

is included in the concept drift field. However, as pointed out by Gama et al. [73], outliers or noise, which refers to a once-off random deviation or anomaly, is not concept drift as we need to select outliers from input data to improve the model's accuracy rather than acquiring information from the noise. Identifying concept drift from noise is also one of the challenges in this area.

In general, current detection methods can be divided into three main types based on their mechanics [132].

Data Distribution

As the definition of concept drift is based on a probability density function, a direct detection method is to identify whether the distributions are different from the previous cases and the new stream. In statistical theory, two well-known non-parametric methods, the Wilcoxon test [75], and K-S test [58] are designed to compare two samples. K-S test constructs statistics to measure the distance between two samples, and then determines if the distance can be considered as a significant discrepancy by giving the probability distribution of K-S statistics. It has been broadly used in the statistics field but rarely directly applied in the concept drift area as it limits itself in one-dimensional distribution, and the real world is frequently of high dimension. Still, the main idea of the K-S test is convincing, and most drift detection by distribution methods are modifications of it. Kifer et al. [108] designed a KS structure concept drift detection method by introducing a novel family of a distance measure between distributions. Take the research of Dasu and Krishnan

[38] as an example.

To quantify the discrepancy between the old and new data streams, they introduced the Kullback-Leibler (KL) divergence (also called relative entropy) and refined it as Eq. (2.3).

$$D\left(W_1 \| W_2\right) = \sum\nolimits_{a \in A} P_{w_1}(a) \frac{P_{w_1}(a)}{P_{w_2}(a)} \tag{2.3}$$

where $P_{W_1}(x)$ and $P_{W_2}(x)$ are the probability mass functions of window W_1 and W_2 separately.

Note that the proposed distance has neither a known exact distribution, nor is it based on familiar data distribution, so the authors introduced a non-parameter hypothesis testing method, bootstrap. The bootstrapping procedure can be used to estimate the standard error by random sampling with replacement.

Learner Outputs

Instead of testing the discrepancy between the distributions, Gama et al. [72] suggested detecting drift by controlling the online error-rate of the algorithm. As discussed in section 3.1, in the condition of a stationary environment, the error will decrease. Therefore, drift is recognized when the error significantly increases. To determine to what extent the augment of error is considered as a drift, a threshold of a warning level or a drift level is needed.

Drift detection using learner output methods has been broadly used in this field, especially in regression cases. Gama et al. [72] treated the error of a sequence of cluster cases as random variables from a Bernoulli trial with the probability of misclassification p_i and standard deviation $s_i = \sqrt{p_i\left(1 - p_i\right)/p_i}$. They set the warning and the drift level based on the significance level and declared a new concept when $p_i + s_i$ reaches the warning level, and a new model is learned when the drift level is exceeded. An Early Drift Detection Method (EDDM) has been proposed based on Gama's model, which can be available in gradual concept drift cases [13]. Hulten et al. [96] proposed an algorithm called the Concept-adapting Very Fast Decision Tree learner (CVFDT), which can detect and respond to concept drift in the example-generating process. In the detection part, they introduced the Hoeffding bounds [17] or additive Chernoff bounds [86] to provide a statistical guarantee for the threshold.

Parameters

Compared to the former two categories, very limited research has been conducted on detecting concept drift using parameters. In the framework of Su et al. [14], they directly structure a learning model which can overcome the time-changing problem rather than recognizing the appearance of drift by optimizing the parameters in a discriminate model. The main idea of their model is to build a dynamical probabilistic model using Eq. (2.4) which includes all the drift information in the parameter vector w_t.

$$\begin{cases} w_t = g\left(w_{t-1}\right) + s \\ p\left(C_k \mid x_t\right) = f\left(w_t\right) + v \end{cases} \tag{2.4}$$

Furthermore, they assume the parameter vector w_t satisfies:

$$\begin{cases} w_t = w_{t-1} + s \\ s \sim N(0, aI) \\ v \sim N(0, r) \end{cases} \tag{2.5}$$

where s is the uncertainty caused by concept drift, aI indicates the degree of concept drift, and r is the variance of noise.

Another example of drift detection using parameters is proposed by Fromont et al. [20]. They solve the drift problem by resorting to a variational Bayes inference scheme in which the probability distribution, as well as the drift, are parameterized using latent variables. One advantage of their model is that it allows them to represent both gradual and abrupt concept drift.

2.2.3 Adapting to Concept Drift

Many learning algorithms have been proposed to deal with concept drift, including rule-based learning [162], decision trees [2], and their incremental versions [190], clustering [164], SVM [33], Bayesian networks [103], RBF-networks [64], instance-based learning [39], and so on [134]. In general, they can be divided into three categories: instance selection (window-based), instance weighting (weight-based), and ensemble learning (learning with multiple models) [134].

Instance selection

Instance selection is the most common concept drift handling technique [40]. The key idea of instance selection is to pick out the most related instances to the current concept and avoid outdated data in the input set. Generally, it will involve generalizing from a moving window over time. Delany et al. [40] proposed a case-based technique for concept drift in spam filtering. It includes two parts: feature selection and case retrieval. The feature selection part is to select the most predictive features, and case retrieval is applied to solve the feature-value redundancy problem and speed up the efficiency of the algorithm. The rules for instance selection are simple, that is, to rebuild the case-base model by updating the misclassified cases at the end of each day. Fan [56] focused on the concept drift problem in situations what kind of old data will help detect concept drift, and in their opinion, this problem can be easily solved by an algorithm that is extremely efficient in comparing all sensible choices with little extra cost. Based on a decision tree ensemble, they proposed a solution for concept drift, which can 'sift off' old data and combine new relevant data into a model.

Instance weighting

The key idea of instance weighting methods is to give each instance an appropriate weight, which is based on its ‘age’ (its relevance to the current concept). In this way, the problem of determining which instance to include or delete from the input set is transformed by comparing the importance of the instance at each time point. Koychev [113] believed the importance of examples decreased with time and proposed a gradual forgetting function based on time. The forgetting function can be defined by various functions. For example, a linear gradual forgetting function has the following form:

$$\begin{cases} w_i = -\frac{2k}{n-1}(i-1) + 1 + k, w_i > 0 \\ \sum w_i = n \end{cases} \tag{2.6}$$

where i is a counter of observations starting from the most recent, and it goes back over time, n is the length of the observed sequence, and $k \in [0,1]$ is a parameter computed as a discrepancy between the increasing percentage of the last one (or the decreasing percentage of the first one) and the average.

Another example of instance weighting is the experiments conducted on two scenarios of concept drift [195]. In this paper, the authors applied the kernel mean matching method and optimal weights adjustment method to weight samples for loose concept drifting (LCD) and rigorous concept drifting (RCD) separately. They used different functions because in LCD cases, concepts in the adjacent streaming data are close to each other while they are randomly and rapidly changed in RCD cases.

Ensemble Learning

Nowadays, ensemble learning methods have been successfully and broadly applied to deal with concept drift [176, 42, 142]. It combines the advantages of instance selection and instance weighting for the 'ensemble' characters. Minku et al. [142] explained the mechanism of why an ensemble method is helpful for drifting data using a series of experiments that validate the relationship between learning error and a model's diversity both before drift and shortly after drift. The results show that ensembles with less diversity obtain lower test errors, and highly diverse ensembles can obtain lower test errors shortly after drift [142]. Generally speaking, there are three ensemble frameworks of ensemble learning, horizontal ensemble, vertical ensemble, and a hybrid of the two. Both the horizontal and vertical ensemble combines the models built on different data chunks using the same or different learning models. The difference between them is in vertical ensemble cases; only the most recent data chunk is used [195].

2.3 Regression For Streaming Data

Regression is an important task in streaming data mining. In this section, the streaming data regression problem and corresponding state-of-the-art solutions will be reviewed.

2.3.1 Online Regression Algorithms

In the past ten years, although different learning tasks such as classification and regression for data streams have been considered, regression on data streams has gained less attention than classification, with a few notable exceptions. For instance, the Hoeffding tree-based methods [155] has attracted a lot of attention from researchers, and many modifications and improvements of the original method have been proposed. Fast Incremental Model Trees with drift detection (FIMT-DD), initially presented in [182], is the main example. Similar to standard Hoeffding Trees, features are ranked according to their variance in FIMT-DD, and if the two best-ranked features differ by at least the Hoeffding Bound [17], the tree branches and the process is repeated. In addition, Gomes proposed an adaptive random forest by assembling the Hoeffding tree [79]. Ikonomovska et al. modified the Hoeffding tree method to present option trees and ensembles of option trees for regression [98]. Adaptive Model Rules [40] is another relevant representative of streaming data regression and make a prediction by learning rules. In Adaptive Model Rules, the rules are specified by a conjunction of logical operations on the input attributes in the premise part and a linear function of the attributes in the consequent. Support vector regression (SVR) is also extended to handle streaming data regression, Ma et al. [140] proposed an accurate online SVR algorithm that follows the idea of [26]. In the accurate online SVR, three sets of training data (named remaining, error, and supporting) are built according to Karush-Kuhn-Tucker (KKT) conditions [170]. When a new sample arrives, it is assigned to one set through an infinite number of discrete steps until it meets the KKT conditions, while ensuring that the existing data continue to satisfy the KKT conditions at each step. Following the idea of accurate online SVR, Gu et al. [82] proposed an incremental v-SVR algorithm, and Hang et al. [191] proposed an incremental dual-v-SVR algorithm as an extension to the incremental v-SVR. In contrast to the above model-based algorithms, IBLStreams [168] is an instance-based learning system, where the prediction result can be estimated by the weighted mean of the k-nearest neighbor instances. However, the above algorithms only consider if there is a single output in the regression problem.

2.3.2 Forgetting Mechanism

In most of the current online regression algorithms, there are two main types of forgetting mechanisms were designed to help algorithms deal with concept drift. In the first type of forgetting mechanism, the forgetting of learned knowledge is independent of the detection result of concept drift. Hence, the forgetting strategy is triggered as long as new data arrive, thereby discarding outdated knowledge. For example, Wang et al. [19] proposed a Budgeted Stochastic Gradient Descent (BSGD) for training support vector machines. In BSGD, the weight w_i of model $f_i(X) = w_i X$ is updated when a new datum arrives, and a new support vector is added. If the number of support vector greater than the user-decided budget, one outdated support vector will be deleted, and the weight w_i is readjusted. However, setting the value of the budget is a difficult task in itself. In the above online SVR-based methods [140, 82, 191], a decremental algorithm is used to forget one datum from one of the three sets and then update $f(X)$. However, the adjustment speed of the model is too slow to keep up with sudden drift. Omitaomu et al. [149] proposed an incremental SVR with varying parameters (VPI-SVR) rather than fixed ones. VPI-SVR uses different parameters to learn $f(X)$ in a different time window tw_i. However, when $f(X)$ is stable in conjoint time windows, VPI-SVR uses the different parameters to learn $f(X)$, and the performance suffers as a result.

In the second type of forgetting mechanism, the learned knowledge is forgotten according to the detection result of concept drift. This type forms the largest category of forgetting mechanism and is used in many algorithms. For example, Fast and Incremental Model Trees (FIMT-DD) encompasses a change detection scheme that periodically flags and adapts sub-branches of the tree where significant variance increases are observed [182]. As for Adaptive Model Rules (AMRules) [6], to detect and adapt to concept drifts, each rule is associated with a Page-Hinkley drift detector [145], which prunes the rule set given changes in the incoming data. Liu and Zio [127] proposed an algorithm called feature vector selection to detect drift, and a new SVR based on the new data is learned when concept drift is detected. However, it is not always easy to detect concept drift because its underlying causes are not necessarily evident in the data. In contrast to the above model-based methods, IBLStreams [168] is an instance-based learning algorithm, so the prediction result can be estimated by the weighted mean of the k neighbor instances. IBLStreams also uses a change detection scheme to update parameters such as the k and kernel width.

Another important branch of this type of forgetting strategy is based on ensemble [109, 126, 128], i.e., a new learner will be trained after concept drift is detected. For example, Zhai et al. [192] ensemble BSGD with a different budget, Ikonomovska ensemble online option trees for regression (ORTO) [98], and Gomes proposed an adaptive random forest (ARF-Reg) [79] by assembling FIMT-DD. The overall idea of ensemble-based methods is to learn a series of $f_i(X)$ directly from a series of time windows tw_i, so $f_i(X)$ essentially represents separate streaming data at different periods. However, as more and more $f_i(X)$ are saved, their size will eventually exceed the computer's memory. Therefore, these approaches include a method to select which of $f_i(X)$ should be forgotten. However, designing a selection method is a difficult task in itself, especially since different types of concept drifts can co-exist in streaming data. Furthermore, ensemble-based methods normally rely on the same forgetting mechanism for different types of concept drift. Yet ignoring the differences between different types of concept drift will mean reduced performance. Moreover, each $f_i(X)$ for the current time window tw_i is constructed independently from the last time window tw_{i-1}. However, as for non-stationary streaming data with gradual or incremental drift, as the old concept incrementally changes into a new concept over a long period of time, so $f_i(X)$ for the current time window tw_i can be similar to the last time window tw_{i-1} and they are not independent. This results in ensemble-based methods having drawbacks when handling non-stationary streaming data with gradual or incremental drift.

In summary, many online or incremental algorithms have been proposed to discover knowledge from streaming data. However, compared with classification and clustering, regression has gained less attention, and many challenges have not been overcome well. Hence, tn this thesis, we design streaming regression algorithms from three aspects: (i) dealing with noisy streaming data; (ii) dealing with evolving streaming data; (iii) dealing with streaming data with multiple outputs.

Chapter 3

Streaming Data Regression Under Noisy Environments

3.1 Introduction

To solve the regression problem of streaming data under noisy environments, one solution is to propose a new streaming regression algorithm that is able to learn an accurate prediction model from noisy streaming data.

Most current streaming regression algorithms were designed based on a classical regression model but using online learning theory to learn data, i.e., data is processed sequentially, i.e., $(X_1, Y_1), (X_2, Y_2), \ldots, (X_t, Y_t)$ and, with each new input data, the model is updated (X_t, Y_t) as $f_t \leftarrow f_{t-1}$ [31]. As a flexible parametric regression algorithm, support vector regression (SVR) [170] is a popular regression algorithm in many fields - for example, it is used to predict electrical loads [49], stock market prices [189], and wind speeds [131]. SVR trains using a symmetrical loss function, which equally penalizes high and low mis-estimates. In addition, SVR uses Vapnik's ε-insensitive approach [170], so a flexible tube of the minimal radius is formed symmetrically around the estimated function, such that the absolute values of errors less than a certain threshold ε are ignored both above and below the estimate. In this manner, points outside the tube are penalized, but those within the tube, either above or below the function, receive no penalty [12]. One of the main advantages of SVR is that its computational complexity does not depend on the dimensionality of the input space. Additionally, it has excellent generalization capability with high prediction accuracy.

Hence, for using SVR to handle streaming data, Liu et al. [126] proposed an ensemble SVR. The proposed approach creates new sub-models directly from a basic model, and the sub-models represent the stream data separately at different periods. Chen et al. [34] proposed an improved multiple kernel SVR approach. However, this is, in fact, the learning of these approaches is still needed a small batch of data, so these methods cannot always work in a streaming data environment because only one sample can be operated in extreme conditions. Based on the reason, online learning [165], which investigates how to learn in an extreme condition, has been proposed as a possible solution. As a result, researchers proposed an online version of SVR for handling data streams. Essentially, online SVR integrates an SVR algorithm with online learning to handle regression problems with streaming data. Examples of this approach include Ma et al. [140], who introduced ε-SVR in an online algorithm called accurate online support vector regression (AONSVR). A new adjustment method which based on [26] was proposed to adjust model f_t. In this adjustment method, all samples were assigned into three sets: remaining, supporting, error set. Gu et al. [82] then improved this adjustment method to solve the extra constriction in v-SVR and proposed an incremental v-SVR algorithm (INVSVR). In INVSVR, the speed of adjusting model f_t was significantly improved. Next, for handling uncertain data streams, Hang et al. [191] decomposed the classical v-SVR into a new model called dual-v-SVR, and use the same adjustment method in INVSVR to adjust the model f_t. Omitaomu et al. [149] applied the AONSVR to handle evolving data streams. In his method, the AONSVR with varying rather than fixed parameters and proposed a method to update parameter weight automatically.

Not like other online regression algorithms such as online regression tree [98] and online multiple kernel regression [166], the above mentioned online/incremental SVR algorithms do not need any prior knowledge of whole data streams. For example, in an online regression tree, the depth of the tree need to be user-defined, and the number of kernels needs to be user-defined in online multiple kernel regression. These parameters could be set suitable if some prior knowledge of whole data streams were obtained. However, the performance of online SVR algorithms is still limited by two challenges:

1). If a data source is very noisy due to, say, electromagnetic interference, temporary failure of sensors, and so on, accuracy will suffer. Some robust SVR algorithms have been explicitly designed to handle noisy data, and these perform better than classical SVRs [124, 154], but performance is still not optimal. For example, Lin et al. [124] incorporated the concept of fuzzy set theory into SVR, while Peng [154] proposed an interval twin SVR algorithm. Other SVR algorithms handle noisy data [66, 1, 95] by replacing the constraints in classical SVR with probability constraints, such as the SVR algorithm in [95] that is robust to bounded noise. However, the quadratic programming problems (QPPs) in these above mentioned SVR algorithms remain too complex for direct translation into incremental SVR.

2). Online SVRs learn slowly. The main idea of the online learning algorithm is to adjust the weight w_{new} of a new sample in an infinite number of discrete steps until it meets the Karush-Kuhn-Tucker (KKT) conditions. Further, the existing samples must continue to satisfy the KKT conditions at each step. However, the KKT conditions in existing incremental SVR algorithms are complex, so, in practice, any adjustments to w_{new} tend to create many conflicts among the KKT conditions of the existing samples. As a result, repeated adjustments are needed for w_{new}, which drastically slows down the learning speed.

To overcome these two challenges, we proposed a novel online SVR algorithm called online robust support vector regression (ORSVR). ORSVR is a novel variant of incremental v-SVR algorithm that modified classical v-SVR to handle noisy data and proposed a new incremental learning method. ORSVR seeks two unparalleled bound functions to construct an insensitive zone that includes as many samples as possible. A specifically-designed incremental learning algorithm with KKT conditions is then used to solve the QPPs in ORSVR step-by-step based on only one sample per round.

To solve the regression problem of streaming data under noisy environment, another solution is to propose a method to filter noisy data, and thereby only precise data can be used in streaming data algorithms.

Topology learning [174, 139], which aims to learn an accurate topological structure, means the topological structure [187, 61] can closely reflect the data distribution in a dataset [67]. Usually, the topological structure consists of one or more neural networks. In each neural network, similar samples are grouped into nodes, and the nodes are connected by an edge if the corresponding nodes have a sample in common. The same procedure is also adapted to connect the neural networks, so a topology structure can be considered to be a compressed version of the original dataset [67]. Based on the topological structure, our knowledge about this dataset can be increased by measuring its topological features such as connectedness and intrinsic dimension [70]. Therefore, the incremental neural networks algorithm that learns a topological structure from data streams have been proposed. Growing neural networks [63] is another kind of incremental neural network which uses a different learning strategy. For example, Growing Neural Gas (GNG) [63] uses a combined strategy of competitive Hebbian learning and neural gas to learn neural networks. However, these two kinds of incremental neural networks cannot handle data streams directly because the size of neural networks permanently increases and eventually exceed the size of memory. Although some deleting mechanisms [65] were proposed to shrink the size of the neural networks, these two kinds of incremental neural networks have the trade-off between deleting previously learned nodes and inserting new nodes, i.e., the stability-plasticity dilemma [24]. In order to solve the stability-plasticity dilemma, adaptive resonance theory (ART) [24] has been introduced. Lim and Harrison [122] propose a hybrid network that combines the advantages of the Fuzzy ART Map [25] and probabilistic neural networks for incremental learning neural networks. Marko [175] proposes a topology learning hierarchical ART network based on two Fuzzy ART. However, these methods mainly work for supervised online learning.

Although the performance of SOINN has been improved, the stability-plasticity dilemma [158, 24] leads to inaccurate results of SOINN, even though the training is repeated in the same environment. Two main reasons account for the problem: (1). Parameters are not properly set. In SOINN, the λ and $age_{\max}$ parameters need be manually set, the parameter λ is used to define the frequency of node removal while $age_{\max}$ is defined as the lifetime of each edge. Therefore, a relatively small λ and $age_{\max}$ results in more previously learned nodes being deleted and more new nodes being inserted. This implies that some nodes representing the data distribution of some areas were deleted, and some nodes representing noisy information were inserted into a topological structure. While a relatively large λ and $age_{\max}$ results in fewer nodes being deleted, this implies that some nodes representing noisy information were not deleted and new nodes representing the data distribution of new areas cannot be inserted. Unfortunately, due to the lack of prior knowledge of data streams, we don't know how to reasonably set the values of the parameters. (2). Incorrect density estimation. In SOINN, whether to connect two nodes depends on the density of the two subclasses that two nodes belong to. An algorithm was designed to calculate the mean accumulated point (MAP) of nodes, and then the MAP is used as the density of nodes because the author assumes the density of the two subclasses can be approximated to two Gaussian distributions if using MAP as the density of node. However, this assumption is very difficult to achieve in practice. In addition, the algorithm that calculates the MAP strongly depends on the sequence of samples, i.e., the density distribution of a subclass can be different if samples follow different sequences. Different density distribution of subclasses will result in a different decision of whether to connect two nodes. However, in data streams, we do not know the sequence of samples in advance.

Therefore, to propose a topology learning algorithm without the stability-plasticity dilemma in unsupervised online learning, a Gaussian membership-based self-organizing incremental neural network (Gm-SOINN) was proposed. Gaussian membership is a common kind of fuzzy logic membership [47]. Previous works have proved that fuzzy logic integrated into learning systems contribute considerably to the modeling and processing of various forms of uncertain information, such as fuzzy rule-based systems [138] and neuro-fuzzy systems [107]. Therefore, to utilize the advantages of fuzzy logic, unlike other SOINN-based methods that allow for only one node to be identified as a "winner" (the nearest node), Gm-SOINN allows for all nodes can be selected as the winner and uses a Gaussian membership to indicate the degree to which nodes are identified as winners. As a result, the stability-plasticity dilemma has been solved in the learning process of our Gm-SOINN. For example, a topology network that can accurately represent the original data distribution can be obtained only when $\lambda = 100$ and $age_{\max} = 50$. However, in our proposed method, an accurate topology network can be obtained when λ is in the range $[50 - 200]$ and $age_{\max}$ in the range $[25 - 100]$.

3.2 Noisy Streaming Data Regression by an Online Robust Support Vector Regression

3.2.1 Preliminary

A regression algorithm learns a model $Y = f(X) =< w, X > +b$, where X means one or more independent variables and Y means a response variable. w represents the weight of X, b represents the bias, and $\langle \cdot, \cdot \rangle$ denotes inner product in reproducing kernel Hilbert space (RKHS).

v-Support Vector Regression (v-SVR) is a popular regression algorithm, which can automatically adjust the parameter ε of an ε-insensitive loss function. Given a training sample set $T = \{(X_1, Y_1), (X_2, Y_2), \ldots, (X_i, Y_i)\}$ with $X_i \in R^d$ and $Y_i \in R$, considered the following primal problem:

$$\begin{aligned} &\min_{\omega,\varepsilon,b,\xi_i^{(*)}} \frac{1}{2}\|w\|^2 + C \cdot \left(v\varepsilon + \frac{1}{l}\sum_{i=1}^{l} (\xi_i + \xi_i^*) \right) \\ &s.t. \quad (\langle w, \phi(X_i)\rangle + b) - Y_i \leqslant \varepsilon + \xi_i, \\ &\qquad Y_i - (\langle w, \phi(X_i)\rangle + b) \leqslant \varepsilon + \xi_i^*, \\ &\qquad \xi_i^{(*)} \geqslant 0, \quad \varepsilon \geqslant 0, \, i = 1, \ldots, l. \end{aligned} \tag{3.1}$$

where the training samples X_i are mapped into a high dimensional RKHS using the transformation function Φ. $\|(w)\|$ is a regularization term, which characterizes the complexity of the regression model. C is the regularization constant, and $\xi^{(*)}$ is the slack variable ('*' is shorthand for the variables both with and without asterisks). v is the introduced proportion parameter with $0 \leq v \leq 1$, which controls the number of support vectors and errors.

Then, the corresponding dual is:

$$\begin{aligned} &\min_{\alpha,\alpha^*} \frac{1}{2}\sum_{i,j=1}^{l} (\alpha_i^* - \alpha_i)\left(\alpha_j^* - \alpha_j\right) K(X_i, X_j) \\ &\qquad - \sum_{i=1}^{l} (\alpha_i^* - \alpha_i) Y_i \\ &s.t. \sum_{i=1}^{l} (\alpha_i^* - \alpha_i) = 0, \sum_{i=1}^{l} (\alpha_i^* + \alpha_i) \leqslant Cv, \\ &\qquad 0 \leqslant \alpha_i^{(*)} \leqslant \frac{C}{l}, i = 1, \ldots, l \end{aligned} \tag{3.2}$$

where $K(X_i, X_j) = \langle \Phi(X_i), \Phi(X_j) \rangle$.

In v-SVR, the parameter v is used to determine the proportion of the number of support vectors which are preferable to keep in the solution with respect to the total number of samples in the dataset, and the parameter ε in the optimization problem formulation is estimated automatically (optimally) according to v. However, in ε-SVR, there is no control over how many data vectors will become support vectors. It could be a few; it could be many. But it is possible to have total control over how much error is allowed in the model, and anything beyond the specified ε will be penalized in proportion to C, i.e., the regularization parameter.

3.2.2 Formulation of Online Robust Support Vector Regression

Although v-SVR has some advantages over ε-SVR, v-SVR introduces two complications. First, the box constraints are related to the size of the training sample set. Second, the formulation contains an additional inequality constraint, which makes them more complicated than ε-SVR. These complications lead to researchers need to design extra adjustments to solve it. For example, in [82], Gu extends the training samples and adds an extra adjustment, called an initial adjustment, as a preprocessing step. In addition, classical v-SVR is not robust to noisy data well, so the prediction performance of v-SVR will decrease sharply if the training dataset has noisy data.

In this research, we propose a novel type of SVR with a simplified formulation and design an adjustment method to adjust it online. In addition, this novel SVR also robust to noisy data. In [154], Peng et al. proposed twin SVR (TSVR), which is a regressor that determines a pair of up and down-bound functions by solving two related SVM-type problems. Each problem is smaller than that of a classical SVR. Hence, to simplify the formulation of the classical v-SVR, we design a novel SVR which called ORSVR following the spirit of TSVR. Like TSVR, ORSVR transforms the classical v-SVR into a new nonparallel plane regressor, but different from TSVR, ORSVR is robust to noisy data.

ORSVR is based on the assumption that the upper and lower bounds will build an insensitive zone. Then, according to the same concept of an insensitive zone in classical v-SVR, any deviations inside the insensitive zone are discarded, and any deviations outside the insensitive zone are rejected. Further, the insensitive zone should be of minimal size to include as many training samples as possible (X_i, Y_i), and as soon as possible, so as to ensure a better fit for the given samples. Consequently, the upper bound function $f_1(X)$ should have a minimal w, b, ε, and ξ and be moved upward. That way, more samples remain below the upper bound function $f_1(X)$. Any error above $f_1(X)$ will not be captured in the slack variable ξ_{1i}, which is penalized in the objective function via the regularization parameter $C_1 > 0$, which is chosen a priori. Similarly, the lower bound function $f_2(X)$ of the insensitive zone is moved downward by maximizing w, b, ε, and ξ in the objective function, again, to ensure that as much training data as possible (X_i, Y_i) is kept above the lower bound. The specifics of the upper and lower bound functions, $f_1(X)$ and $f_2(X)$, are introduced below.

$$\begin{aligned} &\min_{w_1,\varepsilon_1,b_1,\xi_{1i}} \frac{1}{2}\|w_1\|^2 + C_1\left(v_1\varepsilon_1 + \frac{1}{N}\sum_{i=1}^{N}\xi_{1i}\right) \\ &s.t. \quad \langle w_1 \cdot \Phi(X_i)\rangle + b_1 - Y_i \leqslant \varepsilon_1 - \xi_{1i} \\ &\qquad \xi_{1i} \geqslant 0, \quad \varepsilon_1 \geqslant 0, \quad for\ i = 1, \dots, N \end{aligned} \tag{3.3}$$

and

$$\begin{aligned} &\min_{w_2,\varepsilon_2,b_2,\xi_{2i}} \frac{1}{2}\|w_2\|^2 + C_2\left(-v_2\varepsilon_2 + \frac{1}{N}\sum_{i=1}^{N}\xi_{2i}\right) \\ &s.t. \quad \langle w_2 \cdot \Phi(X_i)\rangle + b_2 - Y_i \leqslant \varepsilon_2 + \xi_{2i} \\ &\qquad \xi_{1i} \geqslant 0, \quad \varepsilon_2 \geqslant 0, \quad for\ i = 1, \dots, N \end{aligned} \tag{3.4}$$

Each of these two equations determines the upper and lower bounds; however, they cannot be solved directly. Therefore, the problem of seeking the upper bound function $f_1(X) = \langle w_1 \cdot \Phi(X_i) \rangle + b_1$ is transformed into solving the following QPPs:

$$\begin{aligned} &\min_{w_1, b_1, \xi_{1i}} \frac{1}{2}\|w_1\|^2 + C_1 \left(v_1 (b_1 \text{ - } \varepsilon_1) + \frac{1}{N}\sum_{i=1}^{N} \xi_{1i} \right) \\ &s.t. \quad \langle w_1 \cdot \Phi(X_i) \rangle + (b_1 - \varepsilon_1) \geqslant Y_i - \xi_{1i} \text{ and } \xi_{1i} \geqslant 0 \\ &\qquad\qquad for\ i = 1, \ldots, N \end{aligned} \tag{3.5}$$

where the meaning of the parameters is the same as Eq. (3.2). If we set $B_1 = b_1 - \varepsilon_1$, the solution of minimizing w_1, B_1, and ξ_1 can be found by solving the corresponding dual:

$$\begin{aligned} &\max -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_{1i}\alpha_{1j} K \langle X_i \cdot X_j \rangle + \sum_{i=1}^{N} \alpha_{1i} Y_i \\ &s.t. \sum_{i=1}^{N} \alpha_{1i} = C_1 v_1 \\ &\alpha_{1i} \in [0, C_1/N], \quad i = 1, \ldots, N \end{aligned} \tag{3.6}$$

where $K(X_i, X_j) = \langle \Phi(X_i), \Phi(X_j) \rangle$, $\langle \cdot, \cdot \rangle$ denotes inner product in RKHS. Similarly, the problem of estimating $f_2(X_i) = \langle w_2 \cdot \Phi(X_i) \rangle + b_2$ is equivalent to the following optimization problem:

$$\begin{aligned} &\min_{w_1, b_1, \xi_{1i}} \frac{1}{2}\|w_2\|^2 + C_2 \left(-v_2 B_2 + \frac{1}{N}\sum_{i=1}^{N} \xi_{2i} \right) \\ &s.t. \quad \langle w_2 \cdot \Phi(X_i) \rangle + B_2 \leqslant Y_i + \xi_{2i} \text{ and } \xi_{2i} \geqslant 0 \\ &\qquad\qquad for\ i = 1, \ldots, N \end{aligned} \tag{3.7}$$

and the solution is found by solving this dual:

$$\begin{aligned} &\max -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_{2i}\alpha_{2j} K \langle X_i \cdot X_j \rangle - \sum_{i=1}^{N} \alpha_{1i} Y_i \\ &s.t. \sum_{i=1}^{N} \alpha_{2i} = C_2 v_2 \\ &\alpha_{2i} \in [0, C_2/N], \quad i = 1, \ldots, N \end{aligned} \tag{3.8}$$

After estimating the upper bound and lower bound functions, the final regression function is constructed as

$$f(X_i) = \frac{1}{2}[f_1(X_i) + f_2(X_i)] = \frac{1}{2}\left[\sum_{i=1}^{N} (\alpha_{1i} - \alpha_{2i}) K \langle X_i \cdot X_j \rangle + (B_1 + B_2)\right] \tag{3.9}$$

Based on Eq. (3.5) and Eq. (3.7), it is clear that the formulation of ORSVR is quite different from that of v-SVR in one fundamental way. ORSVR solves a pair of QPPs, whereas in v-SVR, only a single QPP needs to be solved. Further, in v-SVR, the QPP has two groups of constraints for all data points, but there is only one group of constraints per QPP in ORSVR for all data points. This strategy of solving two smaller sized QPPs, rather than one large QPP, means the formulation of ORSVR simpler than classical v-SVR.

However, the box constraints from Eq. (3.6) and Eq. (3.8) are still correlated to the size of the training samples, which makes it difficult to design an incremental learning algorithm for the upper and down-bound functions. Therefore, to obtain an equivalent formulation with box constraints that are independent of the sample size, the objective function of Eq. (3.3) is multiplied by the size of the training sample set, resulting in the following primal problem:

$$\begin{gathered}\min_{w_1,b_1,\xi_{1i}} \frac{N}{2}\|w_1\|^2 + C_1\left(v_1 B_1 N + \sum_{i=1}^{N}\xi_{1i}\right)\\ s.t. \quad \langle w_1 \cdot \Phi(X_i)\rangle + B_1 \geqslant Y_i - \xi_{1i} \text{ and } \xi_{1i} \geqslant 0\\ \text{for } i = 1, \ldots, N\end{gathered} \tag{3.10}$$

It is easy to verify that Eq. (3.10) is equivalent to the primal problem Eq. (3.5), and the dual problem for Eq. (3.10) is

$$\begin{gathered}\max -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_{1i}\alpha_{1j}Q_{ij} + \sum_{i=1}^{N}\alpha_{1i}Y_i\\ s.t. \sum_{i=1}^{N}\alpha_{1i} = C_1 v_1 N\\ \alpha_{1i} \in [0, C_1], \quad i = 1, \ldots, N\end{gathered} \tag{3.11}$$

where Q is a positive semidefinite matrix with $Q_{ij} = (1/N) \cdot K(X_i, X_j)$. The lower bound function $f_2(X) = \langle w_2 \Phi(X_i)\rangle + b_2$ can be modified in the same way.

After modification, the formulation of the classical v-SVR is transformed into the ORSVR where two smaller sized QPPs need be solved, and the box constraints are independent of the size of the training sample set for each QPP. In addition, the upper bound and lower bound functions of the regression model, as estimated by ORSVR, well capture the characteristics of the data distributions. This allows the conditional mean and the predictive variance to be estimated both automatically and simultaneously. Such a feature could be useful in many cases, especially when the noise is heteroscedastic and depends strongly on the input values. Due to these advantages, ORSVR makes a good choice as the regression model for handling noisy data.

3.2.3 Learning Process of Online Robust Support Vector Regression

The previous section introduced the formulation of ORSVR. This section presents the details of the online learning process.

Karush-Kuhn-Tucker Conditions

According to convex optimization theory, a solution for the minimization problem Eq. (3.11) could be obtained by minimizing the following convex quadratic objective function under constraints:

$$\begin{aligned}\min W =& \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_{1i}\alpha_{1j}Q_{ij} - \sum_{i=1}^{N}\alpha_{1i}Y_i + \sum_{i=1}^{N}\delta_i\left(\alpha_{1i} - C_1 v_1 N\right)\\ &+\sum_{i=1}^{N}\eta_i\alpha_{1i} - \sum_{i=1}^{N}\left[\mu_i\left(\alpha_{1i} - C_1\right)\right]\\ &s.t.\ \eta_i, \mu_i \geqslant 0, \quad i = 1, \dots N\end{aligned} \tag{3.12}$$

Then, according to the KKT theorem [170], the first-order derivative of W leads to the following KKT conditions:

$$\frac{\partial W}{\partial \alpha_{1i}} = \frac{1}{2}\sum_{i=1}^{N}\alpha_{1j}Q_{ij} - Y_i + \delta + \eta_i - \mu_i = 0 \tag{3.13}$$

$$\frac{\partial W}{\partial \delta_{\mathrm{i}}} = \sum_{i=1}^{N}\alpha_{1i} - C_1 v_1 N = 0 \tag{3.14}$$

To increase readability, we can replace δ with B_1 then, as explained above, this becomes an optimization problem with a convex domain. Following the Kuhn-Tucker Theorem, the sufficient conditions for a point to be an optimum are

$$\alpha_{1i} \in [0,\ C_1], \quad \alpha_{1i}\left(\sum_{j=1}^{N} \alpha_{1j} Q_{ij} + B_1 - Y_i + \eta_i - \mu_i\right) = 0 \tag{3.15}$$

$$\eta_i \geqslant 0, \quad \eta_i \alpha_{1i} = 0 \tag{3.16}$$

$$\mu_i \geqslant 0, \quad \mu_i (\alpha_{1i} - C_1) = 0 \tag{3.17}$$

The regression function $f(X_i)$ estimation can be written as

$$f(X_i) = \sum_{i=1}^{N} \alpha_{1j} Q_{ij} + B_1 \tag{3.18}$$

and margin function is defined as

$$h(X_i) = f(X_i) - Y_i \tag{3.19}$$

Replacing $\sum_{j=1}^{N} \alpha_{1j} Q_{ij} + B_1$ with $h(X_i)$ in Eq. (3.15), then the relation of $h(X_i)$ and 0 at the changing of α_{1i} can be found:

$$\begin{cases} h(X_i) \geq 0 & \alpha_{1i} = 0 \\ h(X_i) = 0 & \alpha_{1i} \in [0, C_1] \\ h(X_i) \leq 0 & \alpha_{1i} = C_1 \end{cases} \tag{3.20}$$

Equation (3.20) is defined as a system of conditions, called Karush-Kuhn-Tucker (KKT) conditions. The training sample set S is then partitioned into three independent sets according to the value of $h(X_i)$ (see Figure 3.1):

i. $S_s = \{i : h(X_i) = 0, 0 < \alpha_{1i} < C_1\}$ is the supporting set SS, which includes the training samples strictly on the tube;

ii. $S_E = \{i : h(X_i) \leqslant 0, \alpha_{1i} = C_1\}$includes the training samples exceeding the tube; and

iii. $S_R = \{i : h(X_i) \geqslant 0, \alpha_{1i} = 0\}$ is the remaining set SR, which includes the training samples covered by the tube.

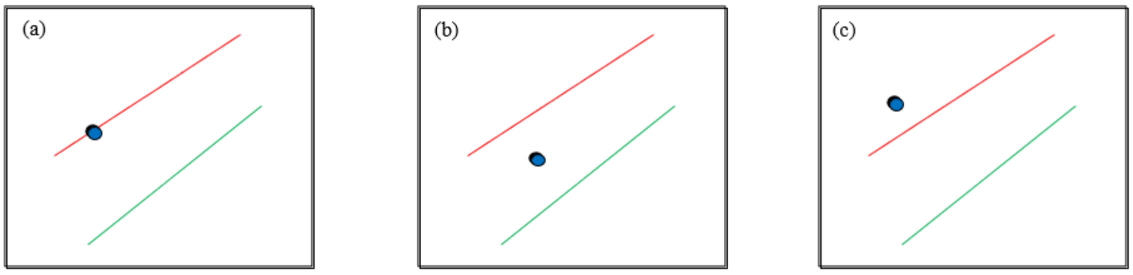


Figure 3.1 : The partitioning of the training samples S into three independent sets by KKT conditions. (a) S_S. (b) S_E. (c) S_R.

The same procedure is used for finding the lower function $f_2(X_i)$, and the KKT condition for the analyses is

$$\begin{cases} h(X_i) \leq 0 & \alpha_{2i} = 0 \\ h(X_i) = 0 & \alpha_{2i} \in [0, C_2] \\ h(X_i) \geq 0 & \alpha_{2i} = C_2 \end{cases} \tag{3.21}$$

Similarly, the training sample set S for the lower function $f_2(X_i)$ is also be partitioned into three independent sets according to the value of $h(X_i)$.

Adjustment Methods

In this section, we focus on the step-by-step process for obtaining the optimum solution of the ORSVR model. The main idea of this adjustment method follows the same procedure as used in AONSVR and INVSVR. That is, the weight α_c is changed according to a new sample X_t in an infinite number of discrete steps until it meets the KKT conditions, while ensuring that the existing samples continue to satisfy the KKT conditions at each step.

However, in AONSVR, when a new sample X_t arrives, the weights α_c of the new sample X_t are initially set to 0 because the sum of α_{1i} equals 0. Two adjustment steps are then needed to ensure all existing samples meet the KKT conditions. In INVSVR, the sum of α_{1i} equals CvN. But because the training samples in INVSVR extended to (X_i, Y_i, z_i), where $z_i = \{-1, 1\}$ is the label of the training sample (X_i, Y_i), and the constraint $\sum_{i=1}^{2N} \alpha_i = CvN$ has a conflict with $\sum_{i=1}^{2N} z_i \alpha_i = 0$. Thus, the incremental learning algorithm in [82] must include an extra adjustment step, called the initial adjustment, to preprocess the training samples. Note that two further adjustment steps are still needed to resolve any conflicts.

Hence, we designed a simple adjustment method for upper function to seek the minimization problem Eq. (3.10). Adding a new sample X_c, according to Eq. (3.11), the equation:

$$\sum_{i=1}^{N} \alpha_{1i} + \alpha_{1c} = C_1 v_1 (N+1) \tag{3.22}$$

need to be met. However, if we let the weights α_{1c} be the equal of 0, the sum of α_{1i} should equals $C_1 v_1 (N+1)$, i.e., $\sum_{i=1}^{N} \alpha_{1i} = C_1 v_1 (N+1)$.

To ensure all the existing samples continue to satisfy the KKT conditions, weight α_{1i} of each sample $\{s_1, s_2, \cdots, s_s\}$ is adjusted through following equation (see Appendix A.1 for the proof):

$$\begin{bmatrix} B_1' \\ \alpha_{1s_1}' \\ \vdots \\ \alpha_{1s_s}' \end{bmatrix} = \begin{bmatrix} B_1 \\ \alpha_{1s_1} \\ \vdots \\ \alpha_{1s_s} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & Q_{s_1 s_1}' & \dots & Q_{s_1 s_s}' \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q_{s_s s_1}' & \dots & Q_{s_s s_s}' \end{bmatrix}}_{R}^{-1} \begin{bmatrix} C_1 v_1 \\ \vdots \\ -h' \\ -h' \end{bmatrix} \tag{3.23}$$

where α_{1i}' represents the new weight of each sample in the supporting set, and B_1' represents the new bias. Thus, $Q_{ij}' = (1/(N+1)) \cdot K(X_i, X_j)$. Then according to Eq. (19), we get

$$h'(X_i) = f'(X_i) - Y_i = \sum_{i=1}^{N} \alpha_{1j} Q_{ij}' + B_1 - Y_i \tag{3.24}$$

Because $\sum_{i=1}^{N} \alpha_{1i}' = C_1 v_1 (N+1)$, when a new sample X_c arrives, the weight α_{1c}'

of the new sample X_c can be set to 0, but it needs to be assigned to a set to satisfy the KKT conditions. If the assignment violates the KKT conditions, α'_{1c} will be adjusted.

The process for making these adjustments is as follows. When adding a new sample (X_c, Y_c), the margin function changes to

$$h'(X_i) = \sum_{j=1}^{N} Q'_{ij}\alpha^*_{1j} + Q'_{ij}\alpha^*_{1c} + B^*_1 - Y_i \tag{3.25}$$

The variation of the margin can then be easily computed with

$$\Delta\alpha_{1i} = \alpha^*_{1i} - \alpha'_{1i}, \quad i = 1, \ldots N, c \tag{3.26}$$

$$\Delta B_1 = B^*_1 - B'_1 \tag{3.27}$$

$$\Delta h'(X_i) = \sum_{j=1}^{N} Q'_{ij}\Delta\alpha_{1j} + Q'_{ic}\Delta\alpha_{1c} + \Delta b_1 \tag{3.28}$$

Because $\sum_{j=1}^{N}\alpha'_{1j} = C_1 v_1(N+1)$ and $\alpha'_{1c} + \sum_{j=1}^{N}\alpha'_{1j} = C_1 v_1(N+1)$ the following equation can be constructed:

$$\sum_{j=1}^{N}\Delta\alpha_{1j} + \Delta\alpha_{1c} = 0 \Rightarrow \sum_{j=1}^{N}\Delta\alpha_{1j} = -\Delta\alpha_{1c} \tag{3.29}$$

Given the specific properties of each of the three sets, only the samples in supporting set $\{s_1, s_2, \cdots, s_s\}$ are able to change $\Delta\alpha_{1j}$, so only these samples can contribute to the new equation

$$\begin{aligned} &\sum_{j\in S} Q'_{ij}\Delta\alpha_{1j} + \Delta B_1 = -Q'_{ic}\Delta\alpha_{1c} \quad \text{where } i \in S_S \\ &\sum_{j\in S}\Delta\alpha_{1j} = -\Delta\alpha_{1c} \end{aligned} \tag{3.30}$$

Then, the equations above can be rewritten in an equivalent matrix form:

$$\begin{bmatrix} \Delta B_1 \\ \Delta \alpha_{1s_1} \\ \vdots \\ \Delta \alpha_{1s_s} \end{bmatrix} = -\underbrace{\begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & Q'_{s_1 s_1} & \dots & Q'_{s_1 s_s} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q'_{s_s s_1} & \dots & Q'_{s_s s_s} \end{bmatrix}^{-1}}_{R} \begin{bmatrix} 1 \\ Q'_{s_1 c} \\ \vdots \\ Q'_{s_s c} \end{bmatrix} \Delta \alpha_{1c} \tag{3.31}$$

Equation (31) can also be rewritten as follows:

$$\begin{bmatrix} \Delta B_1 \\ \Delta \alpha_{1s_1} \\ \vdots \\ \Delta \alpha_{1s_s} \end{bmatrix} == \beta \Delta \alpha_{1c} = \begin{bmatrix} \beta_{1b} \\ \beta_{1s_1} \\ \vdots \\ \beta_{1s_s} \end{bmatrix} \Delta \alpha_{1c} = -R \begin{bmatrix} 1 \\ Q'_{s_1 c} \\ \vdots \\ Q'_{s_s c} \end{bmatrix} \Delta \alpha_{1c} \tag{3.32}$$

According to Eq. (3.32), the values of $\Delta \alpha_{1i}$ and Δb_1 values are updated to compute β.

In terms of the error and the remaining samples, a set $N = S_E \cup S_R = \{n_1, n_2, \cdots, n_n\}$ is defined. These do not change $\Delta \alpha_{1i}$, but they do change $h'(X_i)$. The variations in $h'(X_i)$ are rewritten in matrix notation:

$$\begin{bmatrix} \Delta h'(X_{n_1}) \\ \vdots \\ \Delta h'(X_{n_n}) \end{bmatrix} = \begin{bmatrix} \Delta Q'_{n_1 c} \\ \vdots \\ \Delta Q'_{n_n c} \end{bmatrix} \Delta \alpha_{1c} + \begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & Q'_{n_1 s_1} & \cdots & Q'_{n_1 s_s} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q'_{n_n s_1} & \cdots & Q'_{n_n s_s} \end{bmatrix} \begin{bmatrix} \Delta B_1 \\ \Delta \alpha_{1s_1} \\ \vdots \\ \Delta \alpha_{1s_s} \end{bmatrix} \tag{3.33}$$

Replacing the variations of $\Delta \alpha_{1i}$ and ΔB_1 with the results obtained in Eq. (3.32) gives

$$\begin{bmatrix} \Delta h'(X_{n_1}) \\ \vdots \\ \Delta h'(X_{n_n}) \end{bmatrix} = \begin{bmatrix} \Delta Q'_{n_1 c} \\ \vdots \\ \Delta Q'_{n_n c} \end{bmatrix} \Delta \alpha_{1c} + \begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & Q'_{n_1 s_1} & \dots & Q'_{n_1 s_s} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q'_{n_n s_1} & \dots & Q'_{n_n s_s} \end{bmatrix} \beta \Delta \alpha_{1c} \tag{3.34}$$

Then, γ can be defined as

$$\gamma = \begin{bmatrix} \Delta Q'_{n_1 c} \\ \vdots \\ \Delta Q'_{n_n c} \end{bmatrix} + \begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & Q'_{n_1 s_1} & \cdots & Q'_{n_1 s_s} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q'_{n_n s_1} & \cdots & Q'_{n_n s_s} \end{bmatrix} \beta \tag{3.35}$$

And Eq. (3.34) is rewritten as

$$\begin{bmatrix} \Delta h'\left(X_{n_1}\right) \\ \vdots \\ \Delta h'\left(X_{n_n}\right) \end{bmatrix} = \gamma \Delta \alpha_{1c} \tag{3.36}$$

According to Eq. (3.36), the values of $\Delta h'(X_i)$ can be updated by computing γ.

In summary, if the weight α'_{1c} of the new sample X_c be set to 0 initially, adjusting the value of α_{1i}, b_1, and $h(X_i)$ can be followed Eq. (3.32) and Eq. (3.36).

Algorithm 1 summarizes ORSVR in pseudo code. Lines 2-6 perform the first step of the adjustment, after which the sum of α'_{1j} equals $C_1 v_1 (N+1)$. Lines 7-18 assign the new samples X_c into one of the three sets S_S, S_E, S_R. Among lines 7-18, line 14 derives the minimal increment $\Delta\alpha_{1c}^{\min}$ using the method in [140], then line 15 adjusts the weight α'_{1c} of the new sample X_c until the assignment meets the KKT conditions.

However, it may happen that $\alpha'_{1i} < 0$ after the update, which means $\alpha'_{1i} < 0$ conflicts with $\alpha'_{1i} \in (0, C_1)$. Therefore, we designed an additional method to overcome this possible conflict. If $\alpha'_{1i} < 0$, we will transform one supporting sample with a minimal $\Delta\alpha'_{1i}$ into the remaining set S_R , and $\Delta\alpha'_{1i}$ is obtained with the following equation:

$$\begin{cases} \Delta\alpha'_{1i} = \alpha'_{1i}/\beta_i, & \text{when } D \cdot \beta_i > 0 \\ \Delta\alpha'_{1i} = -\alpha'_{1i}/\beta_i, & \text{when } D \cdot \beta_i < 0 \end{cases} \tag{3.37}$$

where D represents the direction and is decided by the following equation:

$$D = \operatorname{sign}(-h'_c) \tag{3.38}$$

Calculating all $\Delta\alpha'_{1i}$ yields the minimal $\Delta\alpha'_{1i}$, and the other α'_{1i} and b' are up-

Algorithm 3.1. Online Robust Support Vector Regression (ORSVR)

```
1: Read a new sample $(X_c, Y_c)$,
2: Set $\alpha_{1c} = 0$
3: Compute $h'(X_i)$, $i$ in $\{s_1, s_2, \cdots, s_s, c\}$.
4: **if** existing $h'(X_i) \neq 0$:
5:   Adjust $\alpha_{1i}$ through (23) to get the new value $\alpha'_{1i}$, $i$ in $\{s_1, s_2, \cdots, s_s, c\}$
6: **end if**
7: Compute $h'(X_c)$
8: **if** $h'(X_c) > 0$:
9:   Add $X_c$ to the remaining set and **Exit**
10: **else:**
11:   Compute $\Delta h'(X_i)$. $i$ in $\{s_1, s_2, \cdots, s_s\}$
12:   **while** $X_c$ is not added into a set:
13:       Compute $\beta$ and $\gamma$ according to (36) and (39)
14:       Compute the minimal increment $\Delta\alpha_{1c}^{\min}$.
15:       Update $\alpha'_{1i}$, $\Delta h'(X_i)$, $S_S$, $S_E$ and $S_R$.
16:       Update the inverse matrix $R$.
17:   **end while**
18: **end if**
```

dated as follows:

$$\begin{bmatrix} b_1' \\ \alpha_{1s_1}' \\ \vdots \\ \alpha_{1s_s}' \end{bmatrix} = \begin{bmatrix} b_1' \\ \alpha_{1s_1}' \\ \vdots \\ \alpha_{1s_s}' \end{bmatrix} + \begin{bmatrix} \Delta b_1 \\ \Delta\alpha_{1s_1} \\ \vdots \\ \Delta\alpha_{1s_s} \end{bmatrix} = \begin{bmatrix} b_1' \\ \alpha_{1s_1}' \\ \vdots \\ \alpha_{1s_s}' \end{bmatrix} + \beta\Delta\alpha_{1c} \tag{3.39}$$

Once complete, the α_{1i}' of the rest supporting samples will increase, so this operation is repeated until all $\alpha_{1i}' \in (0, C_1)$.

Algorithm 3.1 shows that rebuilding the matrix R is very inefficient at each iteration due to the high complexity of the matrix inversion (about $O\left(n^2 \log(n)\right)$). To avoid this problem, a further method is needed that is specific to this type of matrix R and reduced the complexity to about $O(s^2)$, where s is the number of samples in the supporting set.

The first sample is updated with

$$R = \begin{bmatrix} -\frac{N'}{N'+1}Q_{11}' & 1 \\ 1 & 0 \end{bmatrix} \tag{3.40}$$

where $N' = N + 1$.

The other samples are then added and updated with

$$R_{\text{new}} = \begin{bmatrix} & & & 0 \\ & -\frac{N'}{N'+1}\text{R} & & \vdots \\ & & & 0 \\ 0 & \dots & 0 & 0 \end{bmatrix} + \frac{1}{\gamma_i}\begin{bmatrix} \beta \\ 1 \end{bmatrix}\begin{bmatrix} \beta^T & 1 \end{bmatrix} \tag{3.41}$$

If a new sample is added to the supporting set S_s, γ_i is defined as:

$$\gamma_i = \frac{N'}{N'+1}Q_{cc}' + \begin{bmatrix} 1 & \frac{N'}{N'+1}Q_{cs_1}' & \cdots & \frac{N'}{N'+1}Q_{cs_{l_s}}' \end{bmatrix}\beta \tag{3.42}$$

If the new sample is moved from the error set S_E or the remaining set S_R into the supporting set, β and γ_i also need to be recomputed as follows:

$$\beta = -\frac{N'}{N'+1} R \begin{bmatrix} 1 \\ Q'_{is_1} \\ \vdots \\ Q'_{is_{l_s}} \end{bmatrix} \tag{3.43}$$

$$\gamma_i = \frac{N'}{N'+1} Q'_{ii} + \begin{bmatrix} 1 & \frac{N'}{N'+1} Q'_{is_1} & \cdots \frac{N'}{N'+1} Q'_{is_{l_s}} \end{bmatrix} \beta \tag{3.44}$$

If the sample is moved from the supporting set S_S to the error S_E or remaining set S_R, the matrix R is updated as follows:

$$\begin{aligned} I &= [1 \quad \cdots \quad (i-1) \quad (i+1) \quad \cdots \quad (l_s+1)] \\ R_{new} &= R_{I,I} - \frac{R_{I,i} R_{i,I}}{R_{i,i}} \end{aligned} \tag{3.45}$$

which deletes the row and column of the removed sample and updates the others.

Complexity

Like every good complexity analysis, we compute the value of the complexity in the worst case (O($f(x)$)).

We firstly analyze the time complexity. In Algorithm 1, the time complexity is caused by three operations: 1). Step 3, i.e., calculating the margin distance $h'(X_i)$ of each sample. The time complexity of this operation is O(n)O(kernel); 2). Step 5, i.e., initially adjusting the weight of each support vectors. The time complexity of this operation is O(n); 3). From Step 12 to Step 17, i.e., adjusting the weights of each sample O(5n). Therefore, the total time complexity is $O(n^3)$*O(kernel). As for space complexity, it is very easy to compute in Algorithm 3.1, because it is mainly caused by saving kernel matrix. Therefore, the total space complexity is $O(n^2)$.

The complexity of the kernel operations can easily be avoided by saving all the kernel values in a matrix. This reduces time complexity, yet adds to the space complexity the factor $O(n^2)$. Although the complexity $O(n^3)$ can seem same compared to other online SVR algorithms, but, in practice, this does not happen. In the average case, the algorithm has almost half complexity of other online SVR algorithms, as shown in the experimental section. The speed of learning depends mostly on the number of support vectors, which can significantly influence performances.

3.2.4 Experiments

In this section, we illustrate the effectiveness of our proposed ORSVR through comparing its performance to other online SVR algorithms. The evaluations involved different scenarios using both artificial and real-world datasets. The artificial datasets allowed us to control the relevant parameters and to evaluate the algorithms empirically with specific types of changes. The real-world datasets enabled us to evaluate the merit of the proposed approach in practical scenarios. All experiments were conducted in Python 3.5 on a PC running Windows 7 with an Intel Core i5 processor (2.40 GHz) and 8-GB RAM.

Artificial Datasets

Table 3.1 : Parameters for each compared incremental SVR

Parameters	*Method*		
	AONSVR	*INVSVR*	*ORSVR*
C	100	100	100
ε	0.1	0.1	*
v	*	0.01	*
Kernel	200	200	200

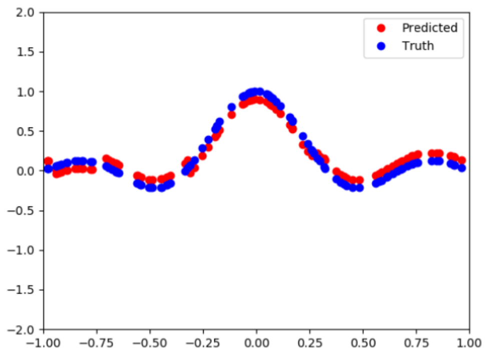
* means the parameter does not need to be set in this algorithm

In our first set of experiments, we compared ORSVR with AONSVR [140] and INVSVR [82] on several artificial datasets. For simplicity, we used the radial basis function (RBF) kernel for all algorithms and set the model parameters $C_1 = C_2 = C$ and $v_1 = v_2 = v$ for each algorithm to the values listed in Table 3.1.

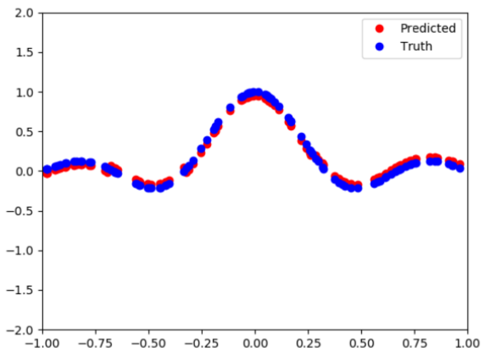
The first test was to estimate a *sinc* function, given N examples (X_i, Y_i), with X_i drawn uniformly from $[-3\pi, 3\pi]$ and $Y_i = \text{sinc}\,(X_i) + e_i$.

$$\text{sinc}(X) = \begin{cases} \sin(X)/X & \text{if} \quad X \neq 0 \\ 1 & \text{if} \quad X = 0 \end{cases} \tag{3.46}$$

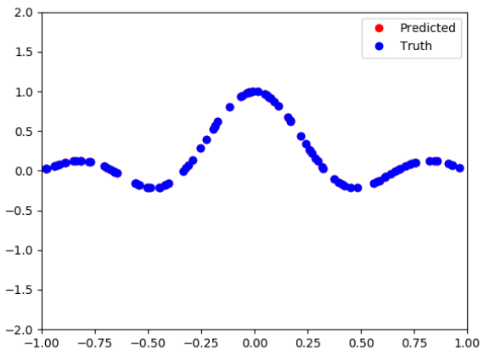
where e_i is the noise drawn from a uniform distribution on $U(-k, k)$. Here, $U(-k, k)$ represents the uniformly random variable in $[-k, k]$.



(a) AONSVR



(b) INVSVR

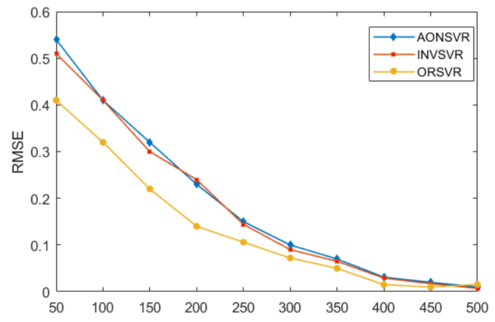


(c) ORSVR

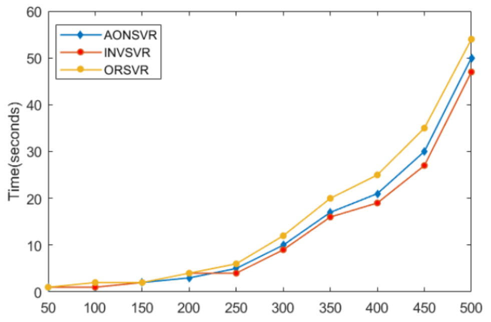
Figure 3.2 : The regression models obtained built by applying different incremental SVR algorithms on the *sinc* data.

Figure 3.2 shows the results for the *sinc* evaluation. The red dots represent the predicted value, and the blue dots represent the true value. When the predicted value is very close to the true value, the blue dot will almost cover the red dot. Therefore, from Figure 3.2, we can see the regression model built by AONSVR does not perfectly fit the data, because most of the red dots are not covered. The results with INVSVR were better, but the model still does not perfectly fit the data. By contrast, the regression model built by ORSVR does fit the data well, showing that ORSVR built the model with the highest accuracy.

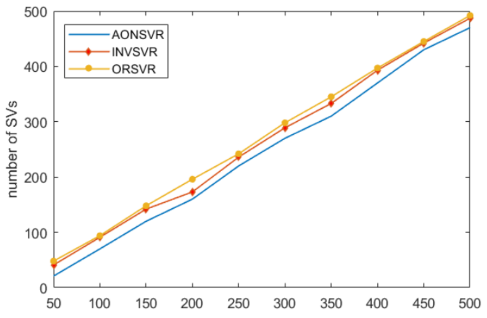
Figure 3.3 shows the results of the same experiment in terms of root-mean-square error (RMSE) [27], training time (in seconds), and the number of support vectors needed to estimate the final regression function for different sample sizes N.



(a) RMSE



(b) Training Time

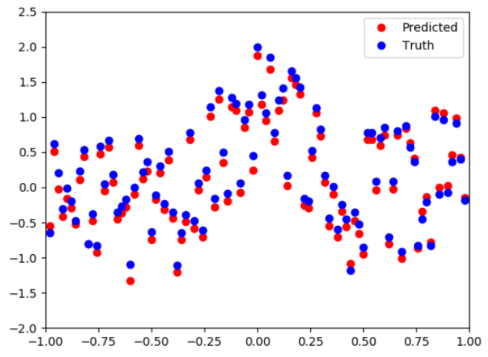


(c) Support Vectors

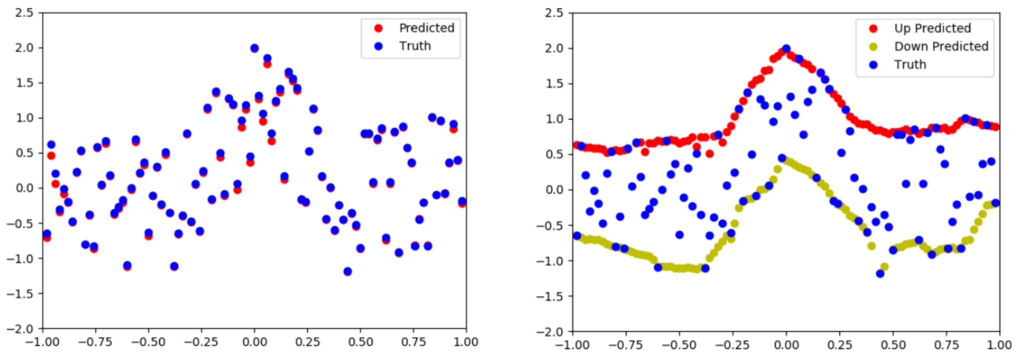
Figure 3.3 : Comparative results in terms of RMSE, training time, support vectors in *sinc* dataset (the x-axis represents the size of instances).

As shown in Figure 3.3(a), ORSVR had a smaller RMSE than AONSVR and

INVSVR, which shows that ORSVR has better generalization ability. Figure 3.3(b) shows that ORSVR's learning speed was not significantly lower than either AONSVR or INVSVR, but ORSVR was able to include more support vectors (as shown in Figure 3.3(c)). These results show that although ORSVR has more support vectors, the learning speed of ORSVR not much slower than AONSVR and INVSVR. The reason may be the strategy of converting a single large QPP into two smaller QPPs speeds up the learning process.



(a) AONSVR



(b) INVSVR

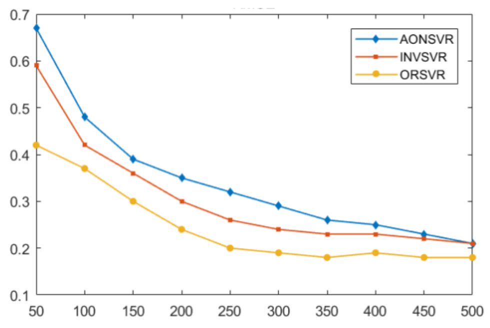
(c) ORSVR

Figure 3.4 : The regression model obtained by applying different incremental SVR algorithms on the *sinc* dataset with noise.

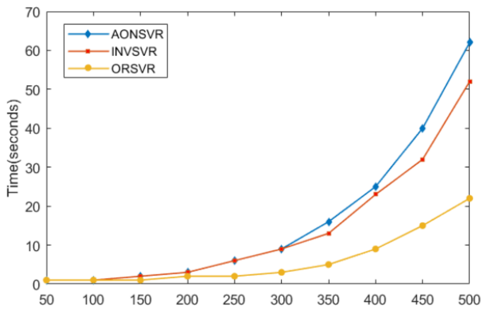
To further explore the potential advantages of ORSVR over AONSVR and INVSVR, we compared the regression performance trends on a noisy version of the *sinc* data and an increased sample size N. The noisy data are randomly sampled from a

Uniform distribution Uni(-0.015,0.015). The noisy *sinc* data is shown in Figure 3.4.

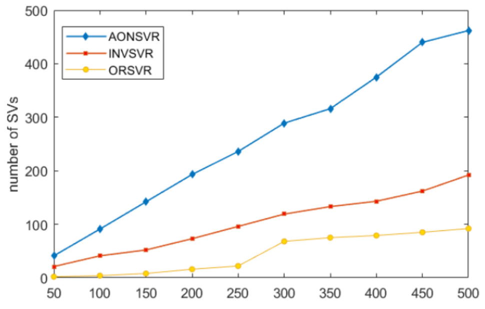
From Figure 3.4, we can see that AONSVR and INVSVR suffered serious over-fitting problems, and many noisy samples were selected as support vectors, whereas ORSVR was still able to represent the data distribution accurately despite the noise. Further, ORSVR only selected a few samples as support vectors, ignoring a large proportion of the noisy ones. The results demonstrate that ORSVR is not sensitive to the variances brought about by noise. We attribute this to ORSVR's ability to automatically increase the width of the insensitive zone as the amount of noise increases. As such, these results also illustrate one of ORSVR's advantages.



(a) RMSE



(b) Training time



(c) Support vectors

Figure 3.5 : Comparative results in terms of RMSE, training time, support vectors in *sinc* dataset with noise (the x-axis represents the size of instances).

Figure 3.5 shows results of this experiment in terms of root-mean-square error (RMSE), training time (in seconds), and the number of support vectors required to estimate the final regression function for different data sizes N.

As shown in Figure 3.5(a), ORSVR had a smaller RMSE than AONSVR and INVSVR in this experiment, too, which again shows that ORSVR is more robust to noisy samples. By handling noisy samples effectively with a 'dynamic' insensitive zone, the resulting model was less influenced by the noisy samples. Thus, ORSVR has better generalization ability (i.e., a smaller RMSE). This is because the ORSVR is good at characterizing data distributions, whereas AONSVR and INVSVR had a tendency to overfit the training samples. These algorithms cannot prevent the influence of noisy samples because ε-SVR and v-SVR consider the deviations overall training samples to minimize the cost function. Comparing ORSVR with AONSVR and INVSVR with many different settings of data size N, the generalization ability of ORSVR was the best. Figure 3.5(b) highlights that ORSVR learned significantly faster than AONSVR and INVSVR. The strategy of dividing the large QPP into two smaller QPPs is partly responsible, but the sparsity is another contributing factor. ORSVR had the lowest sparsity of the three algorithms. In addition, as shown in Figure 3.5(c), the number of supporting vectors required to estimate the regression function with AONSVR and INVSVR was almost equal to the number of noisy samples, while the proportion was significantly lower with ORSVR. Hence, ORSVR also showed the best sparsity among all approaches and, as mentioned earlier, the number of support vectors is the main determinant of prediction speed. In summary, we find from the results of this analysis that the ORSVR algorithm provides superior results in the face of noisy data compared while preserving the advantage of faster learning speeds.

Real-world Datasets

Turning to the real world, we tested ORSVR on nine publicly-available regression datasets. Each dataset represents a different application, with a wide range of data sizes and dimensionalities. We divided the datasets into two groups. The first five being small datasets, and the last four being larger datasets. Table 3.2 lists the details for each.

Table 3.2 : List of datasets

ID	Name	Instances	Attributes
	Small Datasets		
D1	Triazines	186	60
D2	Housing	506	13
D3	Strength	1030	8
D4	Abalone	4177	8
D5	Parkinson's	5875	20
	Large Datasets		
D6	CPUsmall	8192	12
D7	Laser	10073	10
D8	Friedman	15000	10
D9	Cadata	20640	8

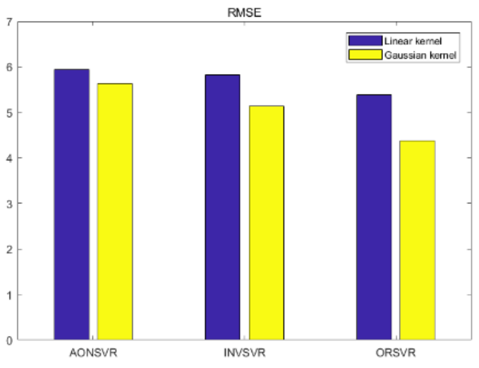
* represents the parameter is not be set in this algorithm

Datasets D1-D5 were sourced from the UCI repository (http://archive.ics.uci.edu/ml/). D6 is available at http:/ /www.csie.ntu.edu.tw/ ∼cjlin/libsvmtools/datasets/regression.html. D7 comes from the Santa Fe Time Series Competition Datasets (http://www.psych.stanford.edu /∼andreas/Time-Series /SantaFe.html). D8 is a noisy dataset (Friedman, 1991), where the input attributes $(x_1, \ldots, x_10)$ are generated independently, each of which is uniformly distributed over $[0, 1]$. The dataset is produced by

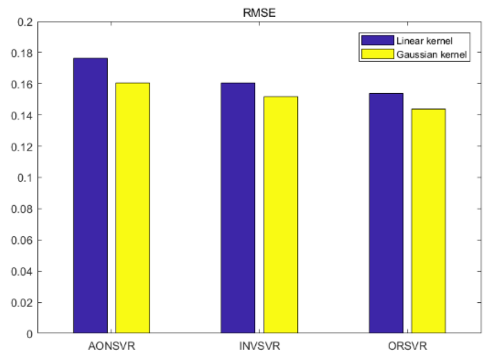
$$y = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \delta(0, 1) \tag{3.47}$$

where $\sigma(0, 1)$ is the noise term, which is normally distributed with a mean of 0 and a variance of 1. Note that $x_1, \ldots, x_5$ only are used in (34), while $x_6, \ldots, x_{10}$ are noisy irrelevant input attributes. D9 is are available from StaLib.

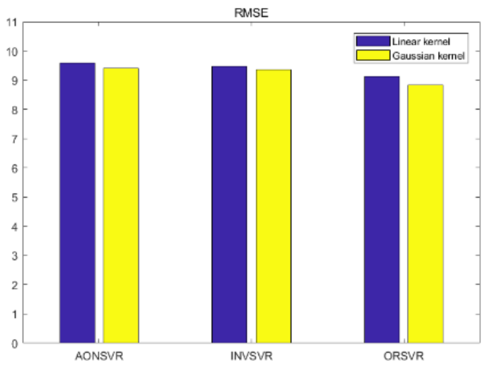
In next experiments, we used a linear kernel $K\left(x_1, x_2\right) = \exp\left(-\|x_1 - x_2\|^2/2\sigma^2\right)$, and a Gaussian kernel $K(x_1, x_2) = \exp\left(-\|x_1 - x_2\|^2/2\sigma^2\right)$ with $\sigma = 0.5$ for all experiments. The parameter ε , representing the strictness of the restoration adjustments, was fixed at -1. The values for v and C were fixed at 0.1 and 100, respectively, in all experiments.



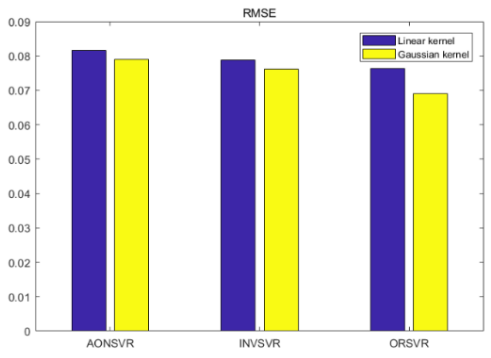
(a) D1



(b) D2

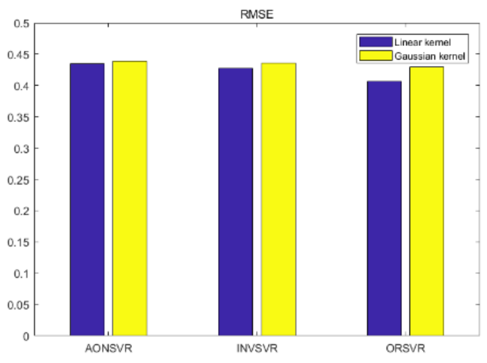


(c) D3

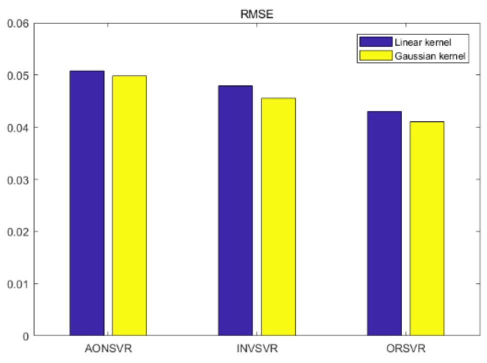


(d) D4

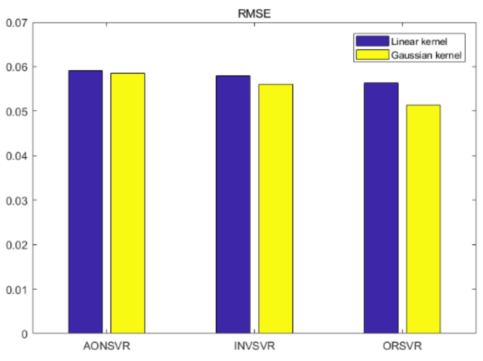
Figure 3.6 : RMSE obtained by different algorithms in D1 - D4 datasets.



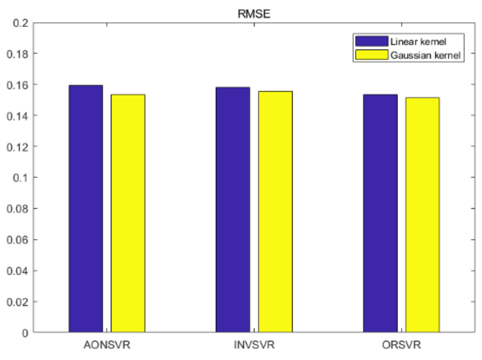
(a) D5



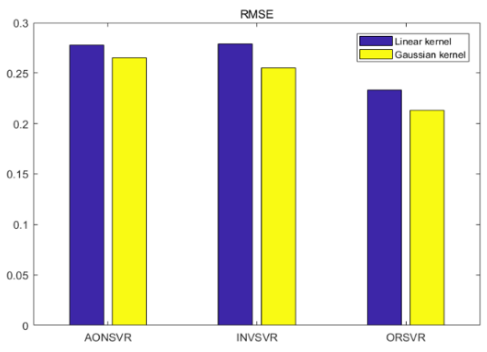
(b) D6



(c) D7



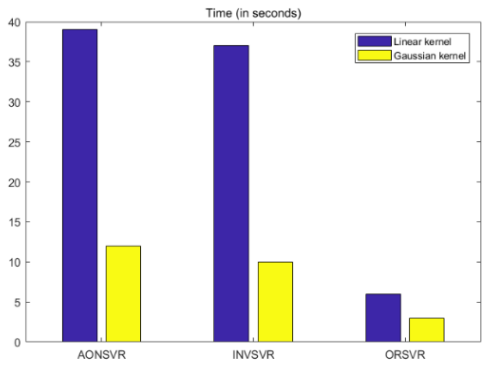
(d) D8



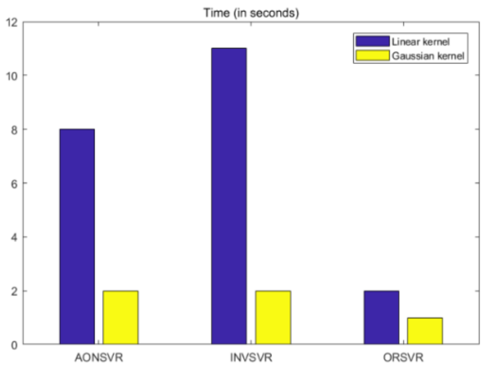
(e) D9

Figure 3.7 : RMSE obtained by different algorithms in D5 - D9 datasets.

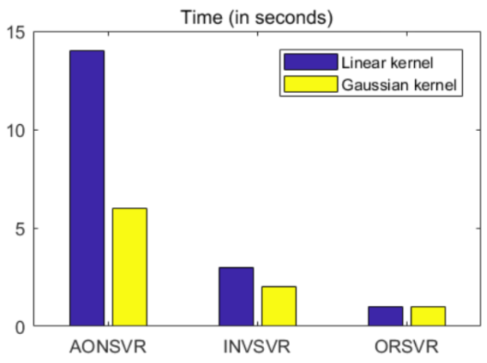
Figure 3.6 and 3.7 compare the regression performance of ORSVR, AONSVR, and INVSVR on the different experimental datasets and different kernels. Over all nine datasets, the Gaussian kernel produced the lowest RMSE with the exception of the Parkinson's dataset (D5). ORSVR and INVSVR performed better than AONSVR on all datasets because both use the parameter v to control the bounds of the proportion of support vectors and errors. The ORSVR algorithm showed markedly better on the Friedman dataset (D8) because it includes noisy data and a heteroscedastic error structure, which offers further support for the advantages of ORSVR over INVSVR. This result also highlights the benefits of allowing an arbitrarily-shaped insensitive zone in ORSVR, rather than the tube shape in v-SVR.



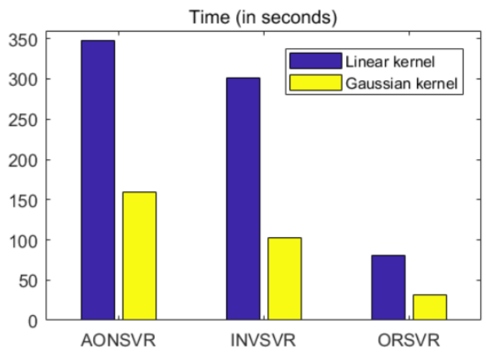
(a) D1



(b) D2

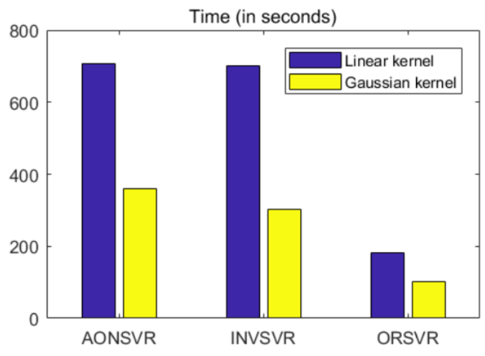


(c) D3

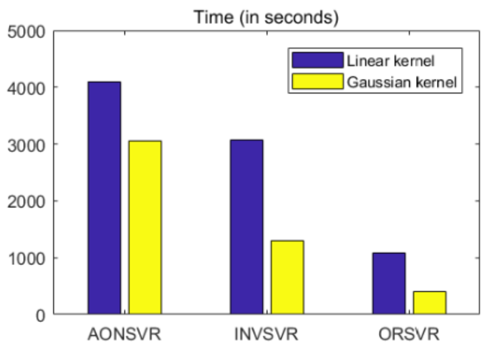


(d) D4

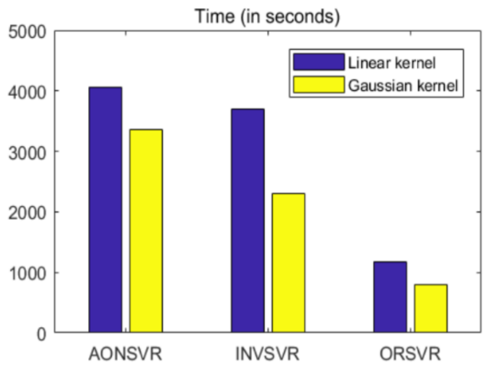
Figure 3.8 : Training time obtained by different algorithms in D1 - D4 datasets.



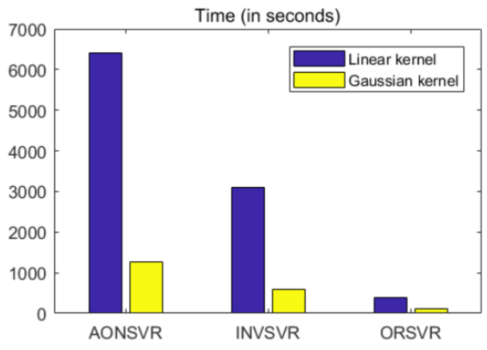
(a) D5



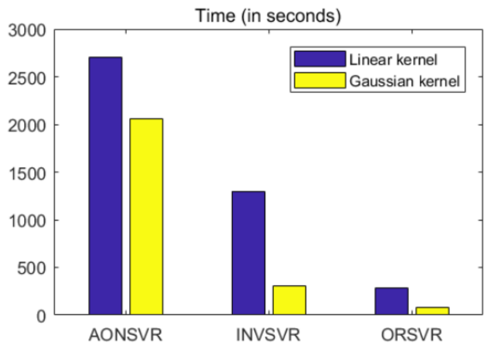
(b) D6



(c) D7



(d) D8



(e) D9

Figure 3.9 : Training time obtained by different algorithms in D5 - D9 datasets.

Figure 3.8 and 3.9 compare the run-time of the three algorithms. In general, as the size of the training samples size N increased, the run-time increased for all three. Further, the Gaussian kernel was faster than the linear kernel and, moreover, the ORSVR's had the fastest training speed over all data sets. These results support the notion that solving two smaller-sized QPP instead of a single larger QPP speeds up the learning process. Like the synthetic experiments, ORSVR required fewer support vectors in proportion to the number of training samples, which further contributed to the faster run-time.

In many real-world applications, the ability to generalize a model and the speed of learning are important considerations. By successfully combining the advantages of SVR and incremental learning, ORSVR shows promise as an alternative to these situations. It not only has the advantage of faster learning speeds but is also sparser and has a greater capacity for generalization than other incremental SVRs in terms of high prediction speeds and satisfactory test accuracy. In addition, ORSVR has the advantage of handling noisy data, which makes it suitable for real-world regression problems with data streams.

3.3 A Gaussian Membership-based Self-Organizing Incremental Neural Network to Filter Noisy Streaming Data

3.3.1 Overview of Enhanced SOINN

E-SOINN was developed for online unsupervised clustering tasks [68]. When a new sample X is presented to E-SOINN, it calculates the nearest node (winner), denoted as s_1, and the second-nearest node (second-winner), denoted as s_2, by the following equations:

$$s_1 = \arg\min_{c \in AN} \|X - W_c\| \tag{3.48}$$

$$s_2 = \arg\min_{c \in AN \setminus \{s_1\}} \|X - W_c\| \tag{3.49}$$

If the distance between the new sample X and the s_1 or s_2 is less than the threshold T_i, the data is assigned to the s_1 or s_2. Otherwise, E-SOINN determines that the new sample X is very different from the current nodes and a between-class insertion should be done. The similarity threshold T_i is calculated using the maximum distance between node i and its neighbor nodes

$$T_i = \max_{j \in \mathrm{nei}_i} \|W_i - W_j\| \tag{3.50}$$

If node i does not have nodes that connect to it, the threshold is defined as the minimum distance between node i and other nodes in the network

$$T_i = \min_{j \in AN \setminus \{i\}} \|W_i - W_j\| \tag{3.51}$$

Consider the case in which a new sample X is assigned to the nearest node s_1. The weights vector W_{s_1} of the winning node and its neighbors are updated as follows:

$$\Delta W_{s_1} = \frac{1}{L_{s_1}} (X - W_{s_1}) \tag{3.52}$$

$$\Delta W_j = \frac{1}{\varepsilon L_{s_1}} (X - W_j) \tag{3.53}$$

where j is the neighbors of the winning node s_1. In addition, E-SOINN uses the MAP of a node to describe the density of the node. The accumulated points sp_i are calculated as the sum of points for node i during a learning period

$$sp_i = \sum_{j=1}^{n} \left(\sum_{k=1}^{\lambda} p_i \right) \tag{3.54}$$

where n indicates the number of learning cycles. The point of node i is calculated as the follow:

$$p_i = \begin{cases} \frac{1}{\left(1 + \frac{1}{m_i} \sum_{j=1}^{m_i} \|W_i - W_j\|\right)^2} & \text{if node } i \text{ is winner} \\ 0 & \text{else} \end{cases} \tag{3.55}$$

where m is the number of neighbors of node i. Then the MAP of node i is calculated as follows:

$$h_i = \overline{sp_i} = \frac{1}{M} sp_i = \frac{1}{M} \sum_{j=1}^{n} \left(\sum_{k=1}^{\lambda} p_i \right) \tag{3.56}$$

where $M = n \times \lambda$.

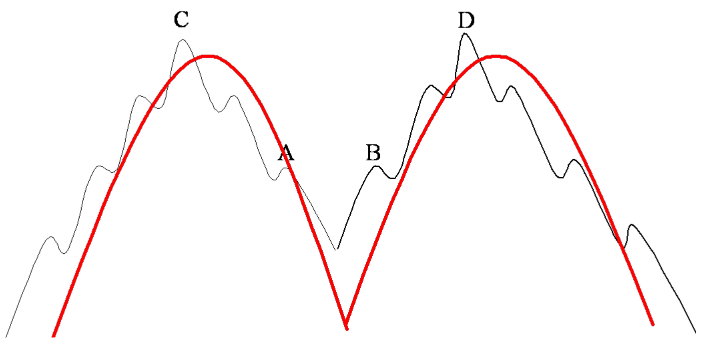


Figure 3.10 : Fluctuating distribution with overlapped area.

Figure 3.10 is an example of a density distribution with an overlapping area, but the density distribution fluctuates. E-SOINN approximates the density distribution into Gaussian distributions. Algorithm 3.2 and Algorithm 3.3 were proposed respectively, to separate the composite class into subclasses and build connections between nodes.

Equations (3.57) and (3.58) are described below

$$\min\left(h_{winner}, h_{\sec ondwinner}\right) = \theta_A A_{\max} \tag{3.57}$$

$$\min\left(h_{winner}, h_{\sec ondwinner}\right) > \theta_B B_{\max} \tag{3.58}$$

In both equations, the winner and second winner lie in the overlapping area between subclasses A and B. The parameter θ with three possible values within the range [0, 1] can be automatically calculated using the threshold function below

$$\theta = \begin{cases} 0.0, & \text{if } 2\times \text{ mean }_A \geq A_{\max} \\ 0.5, & \text{if } 3\times \text{ mean }_A \geq A_{\max} > 2\times \text{ mean} \\ 1.0, & \text{if } A_{\max} > 3\times \text{ mean }_A \end{cases} \tag{3.59}$$

where $A_{\max}$ is the apex density of subclass A, $B_{\max}$ is the apex density of subclass B, and the $mean_A$ is mean density of the nodes in subclass A

$$mean_A = \frac{1}{Num_A}\sum_{i\in A} h_i \tag{3.60}$$

Algorithm 3.2 Separate a composite class into subclasses

```
Input: the set of all nodes $AN$
Output: overlapped area
1: for $node_i$ in $\{AN\}$ do
2:   if $node_i$ has a local maximum density do
3:     $\{N\} \leftarrow$ node$_i$
4:   end if
5: end for
6: for $N_i$ in $\{N\}$ do
7:   $\{C\} \leftarrow findConnectedNodes\,(N_i)$
8:   for $C_i$ in $\{C\}$ do
9:     $C_i$.lable $\leftarrow N_i$.label
10:  end for
11: end for
12: return $C_i$ with two labels
```

For deleting noisy nodes, the following strategy is used: (1) For all nodes in AN, if node i has two neighbors, and $h_i < c_1 \sum_{j=1}^{N_A} h_j/N_{AN}$, then remove node i. (2) For all nodes in AN, if node i has one neighbor, and $h_i < c_2 \sum_{j=1}^{N_A} h_j/N_{AN}$, then remove node i. (3) For all nodes in AN, if node i has no neighbor, the control parameters c_1 and c_2 need to be predefined by the user, as it is difficult to give these parameters a standard for every task.

3.3.2 Gaussian Membership-based Self-Organizing Incremental Neural Network

Gm-SOINN inherits and develops the single-layer SOINN, therefore the patterns that neighbor the feature space, i.e., "similar patterns", are mapped to the same

Algorithm 3.3 Build connections between nodes

```
Input: winner $s_1$ and second winner $s_2$
Output: connections between $s_1$ and $s_2$
1: if $s_1$ with no label or $s_2$ with no label
2:   connect the two nodes with an edge
3: else if the label of $s_1$ and $s_2$ are same
4:     connect the two nodes with an edge
5: else if $s_1$ belongs to subclass $A$ and $s_2$ belongs to subclass $B$
6:     if Equation (3.57) or (3.58) is satisfied
7:         connect the two nodes with an edge
8:     else
9:         do not connect the two nodes
10:     end if
11: end if
```

node, or to nodes that are adjacent in the network. There is only one layer of Gm-SOINN, and this layer is used to process the received data and record the learning result. The structure of the layer (such as the node number and the topology structure) is not predefined and is automatically obtained with the incremental learning of external data. In Gm-SOINN, Gaussian membership is used to represent the degree of input sample $X(t)$ belonging to a node and updates the winning node and the neighbors of the winning node based on Gaussian membership. In addition, Gm-SOINN not only records the weights vector of one node but also the density distribution of the subclass that it belongs to, and the density distribution is learned by $eGMM$. $eGMM$ is implemented, based on evolving Vector Quantization (eVQ) [138, 87] algorithm, so it can accommodate data online and refine the models generated using merge-operations.

The flowchart of Gm-SOINN is the same process as other single-layered SOINN-based methods, such as E-SOINN [68], LD-SOINN [185], and KDE-SOINN [193]. When a new sample $X(t)$ is given to Gm-SOINN, it finds the winner s_1 and the second-winner s_2 of the sample by a similarity metric. If the metric between the new sample $X(t)$ and the s_1 or s_2 is more than the similarity threshold T_i, the sample $X(t)$ is assigned to s_1. If no edge connects s_1 and s_2, GM-SOINN should determine whether to connect s_1 and s_2 with an edge. If s_1 and s_2 connects, then the "age" of the edge is set as "0", and the age of all edges linked to s_1 is subsequently increased by "1". In addition, the weights vector of s_1 and its neighbors is updated. The density distribution of the subclass of s_1 is also updated. Otherwise, Gm-SOINN determines that the sample $X(t)$ is very different from the current nodes and should be inserted as a new pattern. When the age of an edge exceeds $age_{\max}$, the edge is deleted. After λ learning iterations, Gm-SOINN classifies nodes to different classes, then deletes noisy nodes. When the learning process has finished, Gm-SOINN outputs the final topological structure. Below, we introduce the details of Gm-SOINN.

Insert New Pattern

Assume a data stream with samples $X(1), \ldots, X(t), \ldots, X(Q) \in R^d$. The learning task of Gm-SOINN is to represent the dataset by one or more neural networks. In the neural networks, similar samples are grouped into ith nodes with a weights vector $W_i(t) \in R^d$. These nodes are connected by an edge, if the nodes are close to a sample in common after a single pass scan of the training dataset. The learning task is formally defined in the literature as a minimization of the reconstruction error

$$\sum_{\mathrm{t}=1}^{Q} \sum_{i \in AN} \mu_i(t)\, d_i^2(t) \tag{3.61}$$

where $d_i(t)$ is the Euclidian distance (L_2-norm) between the inputs $X(t)$ and $W_i(t)$, so:

$$d_i^2(t) = \|(X(t) - W_i(t))\|^2, \; 1 \leqslant t \leqslant Q, \quad 1 \leqslant i \leqslant AN \tag{3.62}$$

However, $\mu_i(t)$ is not represented by "0" or "1"; rather, it is represented by Gaussian membership [91]. The greater the value of $\mu_i(t)$ represents the closer input of $X(t)$ is to $W_i(t)$, and

$$0 \leqslant \mu_i(t) \leqslant 1, \qquad \sum_{i \in AN} \mu_i(t) = 1 \tag{3.63}$$

The Gaussian membership $\mu_i(t)$ should be determined by the following equations:

$$\mu_i(t) = \left[d_i^2(t) \sum_{j=1}^{AN} \left(\frac{1}{d_j^2(t)} \right) \right]^{-1} \tag{3.64}$$

from Equation (3.64), we can see that the Gaussian membership $\mu_i(t)$ is determined based on all distances $d_i(t)$, i.e., based on all node's weights vector $W_i(t)$.

The weights vector $W_i(t)$ should be defined as the weighted mean of the sample $X(t)$ belonging to the ith node and the Gausssian membership $\mu_i(t)$ represents the weights, so $W_i(t)$ is calculated as follows:

$$W_{\mathrm{i}}(t) = \frac{\sum_i^{AN} \mu_i^2(t) X(t)}{\sum_i^{AN} \mu_i^2(t)} \tag{3.65}$$

However, if Eq. (3.65) is used to calculate $W_i(t)$, we need to know the exact value of AN, and the value of AN will not change during each operation. Unfortunately, Gm-SOINN is an incremental learning algorithm, thus the value of N cannot be known because it may be increased. Based on this reason, we propose an incremental method to calculate the Gaussian membership $\mu_i(t)$.

According to Eq. (3.65), we know the weights vector $W_i(t+1)$ is denoted as:

$$W_{\mathrm{i}}(t+1) = \frac{\sum_{k=1}^{t+1} \mu_i^2(k) X(k)}{\sum_{k=1}^{t+1} \mu_i^2(k)} = \frac{\sum_{k=1}^{t} \mu_i^2(k) X(k) + \mu_i^2(k+1) X(k+1)}{\sum_{k=1}^{t} \mu_i^2(k) + \mu_i^2(k+1)} \tag{3.66}$$

where $k = 1, 2, \ldots, t+1$.

The relationship between the old weights vector $W_i(t)$ and the new vector $W_i(t+1)$ is then introduced as follows:

$$W_i\left(t+1\right) = W_i\left(t\right) + \Delta W_i\left(t+1\right) \tag{3.67}$$

Therefore, in order to obtain the $\Delta W_i(t+1)$, we first transfer the definition of $W_i(t+1)$ from Eq. (3.66) to Eq. (3.68):

$$W_{\mathrm{i}}\left(t+1\right) = W_i\left(k\right) - \frac{W_i\left(k\right)\mu_i^2\left(t+1\right)}{\sum\limits_{k=1}^{t+1}\mu_i^2\left(k\right)} + \frac{\mu_i^2\left(t+1\right)X\left(t+1\right)}{\sum\limits_{k=1}^{t}\mu_i^2\left(k\right)+\mu_i^2\left(t+1\right)} \tag{3.68}$$

Then, $\Delta W_i(t+1)$ can be denoted as:

$$\Delta W_{\mathrm{i}}\left(t+1\right) = \frac{\mu_i^2\left(t+1\right)\left(X\left(t+1\right)-W_i\left(t\right)\right)}{\sum\limits_{k=1}^{t}\mu_i^2\left(k\right)+\mu_i^2\left(t+1\right)} \tag{3.69}$$

The weights vector $\Delta W_i(t+1)$ updated in Eq. (3.69) cannot be calculated in incremental form, since the denominator in Eq. (3.69) cannot be calculated directly, and the calculation of Gaussian membership $\mu_i(k)$ requires all the past input sample $X(k)$. Therefore, an approximate calculation of this term is proposed by introducing the exponential weighting of the past Gaussian membership $\mu_i(k)$, calculated at each time instant, so that the weights vector $W_i(t)$ of the past input sample $X(k)$ decreases exponentially.

The term in the denominator is denoted as $U_i(k) \in R^d$ and calculated as

$$U_i\left(t+1\right) = f_v U_i\left(t\right) + \mu_i^2\left(t+1\right) \tag{3.70}$$

where $U_i(t)$ is defined as follows:

$$U_i\left(t\right) = \sum_{k=1}^{t}\mu_i^2\left(k\right) \tag{3.71}$$

The parameter $f_v, (0 \leq f_v \leq 1)$ denotes the forgetting factor of past input data, i.e., the forgetting factor of the past Gaussian membership $U_i(t)$. The $\Delta W_i(t+1)$ can now be written as:

$$\Delta W_i (t+1) = \frac{\mu_i^2 (t+1) (X (t+1) - W_i (t))}{U_i (t+1)} \tag{3.72}$$

where the current Gaussian membership $\mu_i(t+1)$ is calculated by Eq. (3.64).

Build Connections Between Nodes

When s_1 and s_2 are found, it is necessary to judge whether a connection between them should be built. However, in Gm-SOINN, it is difficult to use Algorithm 3.3 to judge whether to build this connection when the two nodes belong to different

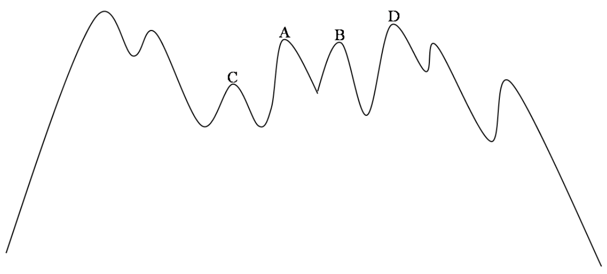


Figure 3.11 : Density distribution in Gm-SOINN.

classes. The reason is if the MAP is still used for the density of nodes in Gm-SOINN, the density distribution of a subclass that the node belongs to will be as shown in Figure 3.11, not as in Figure 3.10. It can be seen from Figure 3.11 that the density distribution cannot be approximated to two Gaussian distributions. Therefore, a new method is needed to ascertain whether to build a connection between s_1 and a s_2.

In Gm-SOINN, each node will store a Gaussian model that it belongs to. Hence, we propose an *eGMM* method based on *eVQ* algorithm [138] to directly learn the density distribution of each Gaussian model. Whether to build a connection between a s_1 and s_2 can then be judged, based on the Gaussian models belonging to the two nodes.

Algorithm 3.4 Build connections between nodes

```
Input: node $s_1$ and node $s_2$
Output: connections between $s_1$ and $s_2$
1: if $s_1$ with no label or $s_2$ with no label
2:   connect the two nodes with an edge
3: else if the label of $s_1$ and second winner $s_2$ are same
4:     connect the two nodes with an edge
5: else if $s_1$ belongs to $G_{s_1}$ and $s_2$ belongs to subclass $G_{s_2}$
6:     if $G_{s_1}$ overlap $G_{s_2}$ and $G_{s_1}$ and $G_{s_2}$ is the homogeneity
7:         connect the two nodes with an edge
8:     else
9:         do not connect the two nodes
10:    end if
11: end if
```

In *eGMM*, each Gaussian model G is associated with a 4-tuple $G =< mv_Q, Cov_Q, num_Q, r_Q >$ to represent its density distribution. According to the theory of bivariate normal contours, we set $r_S = x_d^2(\alpha)$. Here, $x_d^2(\alpha)$ is a value of x^2 distribution with d degrees of freedom and α confidence; α is usually equal to 0.90 or 0.95. Based on the Gaussian model G, a layered tree-structured Gaussian mixture model can be generated to model the structure of *eGMM*. Using a down-top hierarchical clustering approach, each node in the model's layers represents a cluster of the Gaussian components in *eGMM*, and is modeled by a single Gaussian model G. Each leaf node of the tree-structured model corresponds to a node in the topological structure.

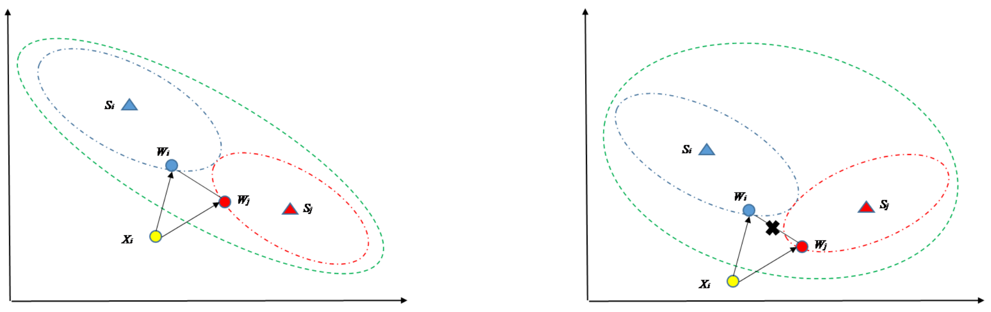
Following this idea, we propose Algorithm 3.4 to judge whether to build connections between s_1 and s_2.

From Algorithm 3.4, the judgment (step 6) is split into two steps as follows: (1a) measures the degree of overlap of the two Gaussian models, and (1b) assesses the homogeneity of the two Gaussian models.

The Bhattacharyya distance [106] is applied to calculate the degree of overlap, which is defined for multivariate Gaussian distributions as:

$$\begin{aligned} olap\left(G_{s_1}, G_{s_2}\right) =& \frac{1}{8}\left(mv_{s_1} - mv_{s_2}\right)^T \times Cov^{-1}\left(mv_{s_1} - mv_{s_2}\right) \\ &+ \frac{1}{2}\ln\left(\frac{\left|Cov^{-1}\right|}{\sqrt{\left|Cov_{s_1}^{-1}\right| \times \left|Cov_{s_2}^{-1}\right|}}\right) \end{aligned} \tag{3.73}$$

where $Cov^{-1} = \left(Cov_{s_1}^{-1} + Cov_{s_2}^{-1}\right)/2$. The useful property of Bhattacharyya is that the distance delivers exactly 0 if two ellipsoids are touching, ">0" when they are overlapping (the closer the value to 1, the bigger the overlap) and "<0" when they are disjoint. Thus, a feasible threshold for Gaussian models overlap candidates is 0.



(a) Gaussian models form a homogenous combination

(b) Gaussian models do not form a homogenous combination

Figure 3.12 : Touching Gaussian models.

The final decision as to whether s_1 and s_2 should be connected depends on the homogeneity of Gaussian models that two nodes belong to. Two Gaussian models are homogenous when their joint distribution follows the "behavior" of their single distributions. To illustrate, we provide two examples of touching Gaussian models in Figure 3.12. In Figure 3.12(a), the two Gaussian models are homogenous. In Figure 3.12(b), the models follow different trends in two-dimensional space, and therefore it is not homogeneity. Our homogeneity condition expresses the change in the volume of the merged Gaussian models; an undetermined high-volume blow-up indicates an inconsistency in the joint subclass homogeneity. Therefore, we hypothetically merge the two Gaussian models, G_{s_1} and G_{s_2}, into one Gaussian model (for formulas see below) G_{md} and compare the volume V_{md} of the new Gaussian model G_{md} with the volume of the old two Gaussian models: V_{s_1} and V_{s_2}.

If the following condition is fulfilled:

$$V_{md} \leqslant V_{s_1} + V_{s_2} \tag{3.74}$$

the two Gaussian models must be homogenous. The parameters V denotes the volume of an arbitrary ellipsoid in high-dimensional space:

$$V_G = \sqrt{2}^{(P+1)} \left(\prod_{i=0}^{(P/2)-1} \frac{1}{P-2i} \right) \sqrt{|Cov_G|} r_G \tag{3.75}$$

where P is the dimensionality of the feature space.

However, Gm-SOINN is an incremental neural network, so the Gaussian models will change when new samples are inputted. As a result, the density distribution also is changed, so the $eGMM$ should adaptively learn the density distribution of each Gaussian model. In order to allow $eGMM$ to easily adapt to Gm-SOINN, the core components in $eGMM$ can therefore be formulated as follows:

(1) If the Gaussian membership $\mu_i(t)$ of a new input sample $X(t)$ belonging to s_1 is lower than T_{s_1} or the Gaussian membership $\mu_i(t)$ of a new input data $X(t)$

belonging to s_2 is lower than T_{s_2}, a new Gaussian model G is created for $X(t)$ as follows:

$$G : \left\langle mv_G = X(t), Cov_G = (\delta I)^{-1}, num_G = 1, r_G = x^2_{num_G}(\alpha) \right\rangle \tag{3.76}$$

(2) Otherwise, if s_1 and s_2 have a connection, the 4-tuple of G is updated as follows:

$$num_G(t+1) = num_G(t) + 1 \tag{3.77}$$

$$mv_G(t+1) = mv_G(t) + \frac{1}{num_G(t+1)}\left(X(t) - mv_G(t)\right) \tag{3.78}$$

$$\begin{aligned} Cov_G(t+1) = &\left(1 - \frac{1}{num_G(t+1)}\right) Cov_G(t) \\ &+ \frac{1}{num_G(t+1)}\left(X(t) - mv_G(t)\right)\left(X(t) - mv_G(t)\right)^T \end{aligned} \tag{3.79}$$

$$r_G(t+1) = x^2_{num_G(t+1)}(\alpha) \tag{3.80}$$

While s_1 and s_2 have no connection, if we need to connect the s_1 and s_2, the 4-tuple of G is updated as follows:

$$num_G = num_{G_{s_1}} + num_{G_{s_2}} \tag{3.81}$$

$$mv_G = \frac{num_{G_{s_1}} mv_{G_{s_1}} + num_{G_{s_2}} mv_{G_{s_2}}}{num_{G_{s_1}} + num_{G_{s_2}}} \tag{3.82}$$

$$\begin{aligned} Cov_G = &\frac{num_{G_{s_1}}}{num_G} \times \left(Cov_{Q_{s_1}} + \left(mv_G - mv_{G_{s_1}}\right)\left(mv_G - mv_{G_{s_1}}\right)^T\right) \\ &+ \frac{num_{G_{s_2}}}{num_G} \times \left(Cov_{G_{s_2}} + \left(mv_G - mv_{G_{s_2}}\right)\left(mv_G - mv_{G_{s_2}}\right)^T\right) \end{aligned} \tag{3.83}$$

$$r_G = x^2_{num_G}(\alpha) \tag{3.84}$$

If we do not need to connect s_1 and s_2, we separately update G_{s_1} and G_{s_2} follow Eq. (3.77)-Eq. (3.80).

In summary, in our proposed $eGMM$, no parameters are required be set manually and Gaussian mixture model will be updated then the new samples are inputted.

In addition, $eGMM$ is equipped with a dynamic (single-pass) class merging strategy which not only resolves significant overlap occurring over time as classes move together, but also compensates for a slight overlap or touching classes.

Classify Nodes to Different Classes

Algorithm 3.5 Classify nodes to different classes

```
Input: the set of all nodes $AN$
Output: final classes
1: for $node_i$ in $\{AN\}$ do
2:   if the $G$ of $node_i$ different with the $G$ of $node_i$ that in set $\{N\}$ do
3:       $\{N\} \leftarrow$ node $_i$
4:   end if
5: end for
6: for $N_i$ in $\{N\}$ do
7:   $\{C\} \leftarrow findConnectedNodes(N_i)$
8:   for $C_i$ in $\{C\}$ do
9:       $C_i$.lable $\leftarrow$ $N_i$.label
10:  end for
11: end for
12: return final classes
```

If the number of input samples generated so far is an integer multiple of parameter λ, we need to update the subclass label of every node. In Gm-SOINN, if two nodes can be linked with a series of edges, we say that a path exists between the two nodes, i.e., given a series of nodes $x_i \in A, i = 1, 2, \ldots, n$, makes $(i, x_1), (x_1, x_2), \ldots, (x_{(n-1)}, x_n), (x_n, j) \in C$, we say that a "path" exists between node i and node j. Martinetz [49] proposed if two nodes are connected with one path in topology representing networks , the two nodes should belong to one class. Algorithm 3.5 shows how to classify nodes to different classes.

Delete Noisy Nodes

Algorithm 3.6 Deleting noise nodes

```
Input: a set of nodes $AN$
Output: noise nodes $\{O\}$
1: for $node_i$ in $\{AN\}$ do
2:   if $node_i$ has only two or fewer two edges:
3:     $G \leftarrow$ get the Gaussian model of $node_i$
4:     calculate the $Mahalanobis$ distance $MS_i$ between $G$ and $node_i$
5:     if $MS_i > r_G$
6:       $\{O\} \leftarrow node_i$
7:     end if
8:   end if
9: end for
10: for $node_i$ in $\{O\}$ do
11:   delete $node_i$ and its edges
12: end for
```

Finally, we need to delete noisy nodes. Because the nodes in Gm-SOINN are associated with a Gaussian model G, the simple Algorithm 3.6 can be adopted in Gm-SOINN to delete noisy nodes and edges. Unlike E-SOINN, it is not necessary to predefine the control parameters c_1 and c_2 in Algorithm 3.6, because $eGMM$ handles data from a global perspective. Thus, the Gaussian models and density distribution are relative stable. In addition, different recognition results for the nodes will not be returned when the training is repeated in the same environment using the same dataset presented in a different sequence.

Complete Algorithm of Gm-SOINN

The learning process is stopped when the final classes and the output of the prototype vectors of every class have been obtained. The complete algorithm of Gm-SOINN is shown in Algorithm 3.7.

In Algorithm 3.7, the similarity threshold T_i is calculated using the maximum Gaussian membership between node i and its neighbor nodes

$$T_i = \max_{j \in \mathrm{nei}_i} \mu_{ij} \tag{3.85}$$

If node i does not have neighbors, the threshold is defined as the minimum Gaussian membership between node i and other nodes in the network

$$T_i = \min_{j \in AN \setminus \{i\}} \mu_{ij} \tag{3.86}$$

It can be seen from Algorithm 3.7 that Gm-SOINN only sets two parameters manually. In the learning process, one difference between Gm-SOINN and other SOINN-based methods is that Gm-SOINN uses the Gaussian membership μ_i as a similarity metric, to identify whether a new pattern needs to be inserted when a new input data $X(t)$ arrives. The Gaussian membership μ_i is updated at each step. Another difference is that Gm-SOINN stores Gaussian mixture models to represent the density distribution of the current observations, so building connections between two nodes and deleting noisy nodes will be based on the Gaussian mixture models.

3.3.3 Experiments

In this section, we introduce the experiments on artificial and real-world datasets. Since Gm-SOINN aims to combine the advantages of topology learning and Gaus-

Algorithm 3.7. Gaussian membership-based self-organizing incremental neural network (Gm-SOINN)

```
Input: Sequence $\{X\}$, $\lambda$, $age_{\max}$
Output: the number of classes and the topology of every class
1: Initialize set $AN$ with the first 2 samples drawn from $\{X\}$.
2: while $\{X\}$ is not empty do
3:   Find winner $s_1$ and second winner $s_2$ from neuron set $AN$, as
       $s_1 = \arg\max_{i \in AN} \mu_i$, $s_2 = \arg\max_{i \in AN \backslash \{s_1\}} \mu_i$
4:   if $s_1 < T_{s_1}$ or $s_2 < T_{s_2}$ then
5:     insert $X$ with weight $W_X(t)$ to $AN$, and set winning times $L_X = 0$.
6:     Add a new Gaussian model $G_X$
7:   else
8:     judge whether to build a connection between $s_1$ and $s_2$ and connect $s_1$
        and $s_2$ by Algorithm 3.4. 9:   update the Gaussian mixture models
10:  Increase the winning times as $L_X(t) = L_X(t-1) + 1$. Then adapt $U$ as
       $U_{n_1}(t) = \left(1 - 0.1^{C_{n_1}}\right) U_{n_1}(t-1) + \mu_i^2(t)$
       $U_{n_{1i}}(t) = \left(1 - 0.1^{C_{n_{1i}}+2}\right) U_{n_1}(t-1) + \mu_i^2(t)$ for all direct neighbors i of $s_1$
11:  Update $s_1$ and its neighbor according Eq. (3.67).
12:  Increase the age of all edges linked with $s_1$ by 1.
13:  for all edge $age_{i,j} > age_{\max}$ do
14:        Remove the edge connecting $i$ and $j$.
15:    end for
16:  end if
17:  if number of input samples divides $\lambda$ then
18:        classify nodes to different classes using Algorithm 3.5.
19:        delete noisy nodes using Algorithm 3.6
20:  end if
21: end while
```

sian membership, with a self-organizing incremental neural network, other SOINN-based methods were implemented for comparison. All experiments were conducted in Python 3.5 on a PC running Windows 7 with Intel Core i5 processor system configuration (2.40 GHz) and 8-GB RAM.

There are normally four parameters in E-SOINN [68]: λ, $age_{\max}$, c_1 and c_2. The definition of λ and $age_{\max}$ has been introduced in Section I and these two parameters significantly influence the shape of the topological structure. The two parameters c_1 and c_2 are defined for controlling the deletion behavior.

A relatively large c_1 or c_2 value will contribute to a higher noise tolerance, but more useful nodes will be deleted at the same time. Therefore, if these four parameters are not properly set, a topology structure that can closely represent the data distribution of the dataset cannot be obtained. In our proposed Gm-SOINN, the topological structure only depends on the two parameters, λ and $age_{\max}$. The definition of λ and $age_{\max}$ is the same as in the E-SOINN, i.e., λ is defined as the frequency of node removal and $age_{\max}$ is defined as the lifetime of each edge.

As for the strategy of selecting parameters, due to in the other SOINN-based methods, λ is set to a multiple φ of 100 and $age_{\max}$ is set to a multiple φ of 50, where $\varphi = 1/2, 1, 2, 3, \ldots$ [68]. Thus, for comparing with other SOINN-based methods, we build a candidate set of $\lambda_S = \{\{\varphi 100 \mid \varphi = 1/2, 1, 2 \ldots\}\}$ and a candidate set of $age_{\max_S} = \{\{\varphi 50 \mid \varphi = 1/2, 1, 2 \ldots\}\}$ in each experiment, then use a grid search method [50] to select the best parameters of every SOINN-based method.

Artificial Datasets and Experiment Results

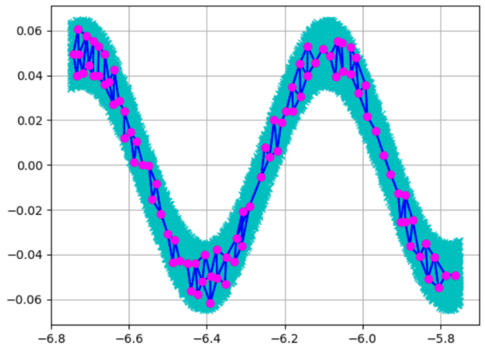
Our proposed Gm-SOINN has the advantage that the stability-plasticity dilemma will be relieved. To test this advantage, we compared Gm-SOINN with three SOINN-based methods (E-SOINN, LB-SOINN, and KDE-SOINN) on the artificial dataset II that include one class, because Gm-SOINN and these SOINN-based methods use a similar strategy to handle data in the same class. The significant difference between Gm-SOINN and these SOINN-based methods is that the Gaussian membership μ_i of nodes will be considered when the nodes are updated.

The artificial dataset II is generated by

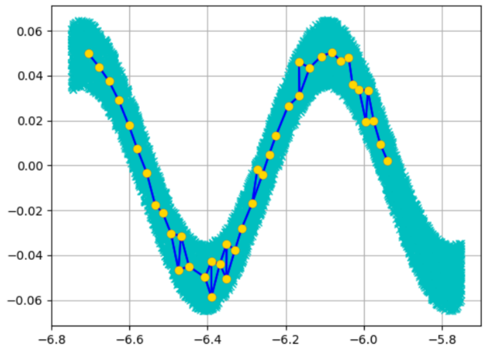
$$\begin{cases} y(t) = \sin(o(t))/20 + \varepsilon_i \\ X(t) = o(t)/10 - 6.25 \end{cases} \tag{3.87}$$

where $o(t)$ is randomly sampled from (0,5), and ε_i is randomly sampled from a Uniform distribution $Uni(-0.015, 0.015)$. For parameters, we chose the $\lambda_S = \{50, 200\}$ and $age_{\max_S} = \{50, 100\}$ in this experiment.

Figure 3.13 and 3.14 shows the results of our comparison, based on some combinations of λ and $age_{\max}$. It can be seen that the number of nodes decreases with λ and $age_{\max}$ decreases for all SOINN-based methods. In addition, when λ is set over 100 and $age_{\max} = 100$, almost every SOINN-based method learns a topological structure to represent dataset II accurately. However, when the parameters are set as $\lambda = 100$ and $age_{\max} = 50$, the results illustrate that the E-SOINN and LB-SOINN methods are unable to accurately represent the data distribution of the artificial dataset II, because some nodes that represent some area of data distribution are deleted. The result of KDE-SOINN method is also unable to accurately represent the data distribution, because the nodes are wrongly classified to three classes. The reason is KDE-SOINN methods save more nodes than the other SOINN-based methods even when the same λ and $age_{\max}$ parameters are used. In contrast, Gm-SOINN achieves a topology structure that accurately represents the data distribution and classifies the nodes into one class. The results prove that Gm-SOINN is less sensitive to parameters than other SOINN-based methods.

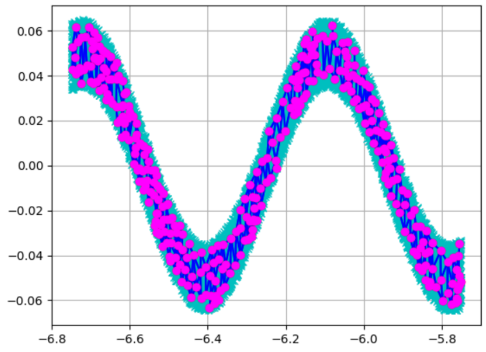


$\lambda = 200, age_{\max} = 100$

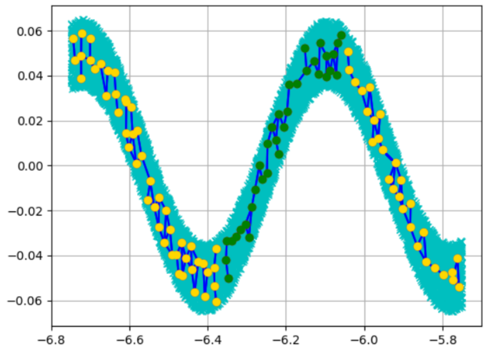


$\lambda = 100, age_{\max} = 50$

(a). E-SOINN

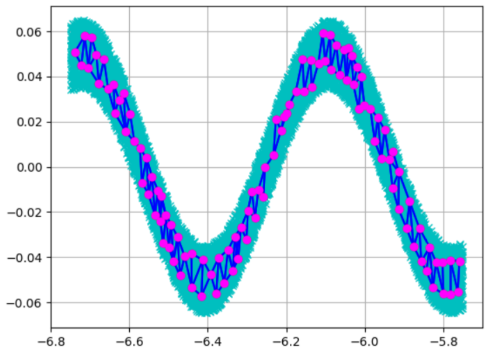


$\lambda = 200, age_{\max} = 100$

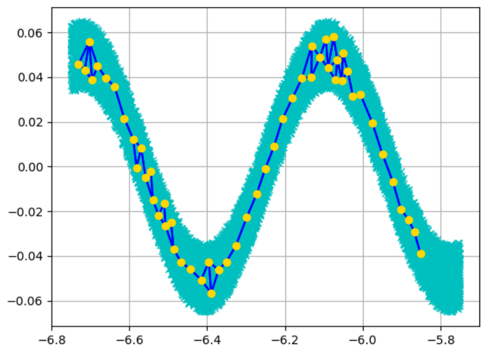


$\lambda = 100, age_{\max} = 50$

(b). KDE-SOINN



$\lambda = 200, age_{\max} = 100$



$\lambda = 100, age_{\max} = 50$

(c). LB-SOINN

Figure 3.13 : Topological structure results on dataset II

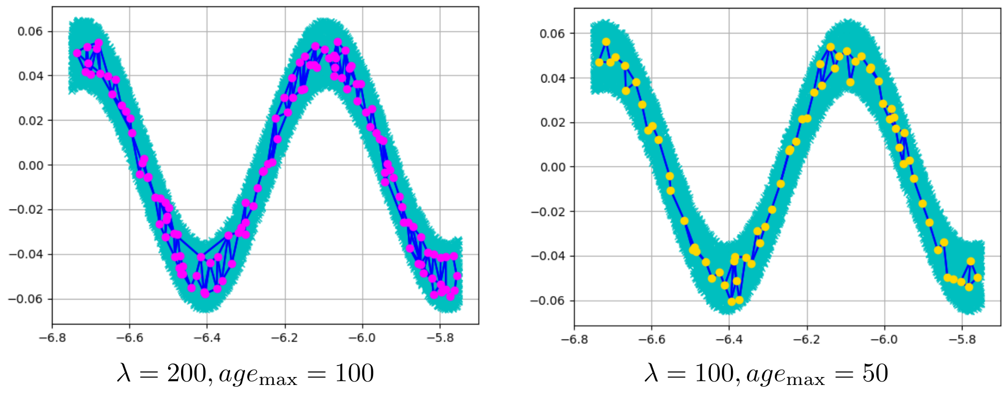
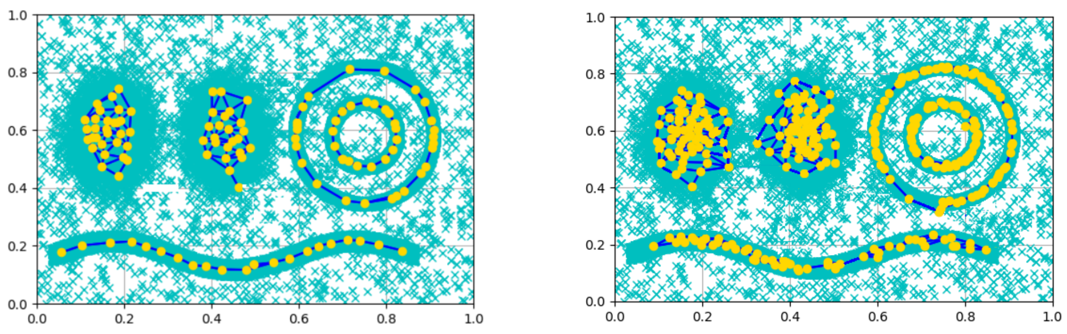


Figure 3.14 : Topological structure results (Gm-SOINN) on dataset II.

Other SOINN-based methods [67, 68] can also learn the topological structures of every class when a dataset includes multiple classes. Therefore, to validate the performance of our proposed Gm-SOINN in processing multi-class data, we adopted artificial dataset III, which has been used in other SOINN-based methods [67, 68]. Dataset III comprises two overlapped Gaussian distributions, two concentric rings, and a sinusoidal curve. In addition, for illustrating our proposed Gm-SOINN is also robust to noise. Like other SOINN-based methods [67, 68], we add 10% noise to test data and noise is distributed over the whole data.

Moreover, to compare with other SOINN-methods, the definition of stationary and non-stationary environments that are introduced in other SOINN-based methods [67] are introduced in our experiment. A stationary environment dictates that patterns are chosen randomly from the whole dataset to train the network online. A non-stationary environment dictates that patterns are chosen sequentially from five areas of the original artificial dataset III to train the network online. We tested Gm-SOINN using this dataset in both stationary and non-stationary environments. When selecting parameters, E-SOINN obtains the correct topological structure when the parameters are set as $\lambda = 100$ and $age_{\max} = 100$, so in this experiment, $\lambda_S = \{50, 100, 200\}$ and $age_{\max_S} = \{25, 50, 100\}$ is used when comparing with the other SOINN-based method.

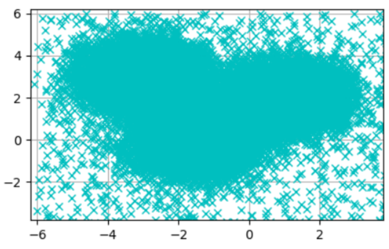


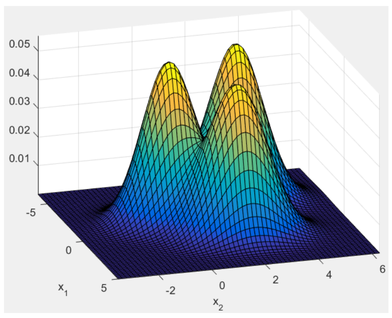
(a) Stationary environment (b) Non-stationary environment

Figure 3.15 : Topological structure of artificial dataset III.

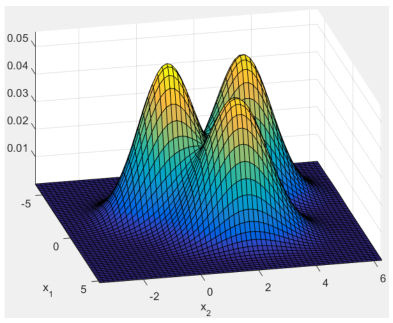
In general, with a decrease of λ and $age_{\max}$, the number of nodes and edges also decreased. Figure 3.15(a) shows the result in a stationary environment, while Figure 3.15(b) depicts the result in a non-stationary environment when $\lambda = 100$, $age_{\max} = 100$. In both stationary and non-stationary environments, the Gm-SOINN system reports that five classes exist and gives the topological structure of each class, illustrating that the Gm-SOINN algorithm performs the same functions as other SOINN-based methods for the same artificial dataset.

In Gm-SOINN, an $eGMM$ was devised to determine whether a connection between s_1 and s_2 should be built. The $eGMM$ can make a correct decision on building a connection between s_1 and s_2 even when the two nodes appear in highly overlapped classes. To validate this advantage, we used artificial dataset IV, illustrated in Figure 3.16(a), which comprises three overlapping Gaussian distributions. In addition, to compare with other SOINN-based methods, we also used the same methods, i.e., noise is distributed over the whole data, and 10% noise is added to this dataset. Figure 3.16(a) shows that the density of the overlapping area in the dataset is high, but the dataset can still be separated into three classes based solely on its appearance when plotted. To compare Gm-SOINN with other SOINN-based methods, we also test our algorithm in stationary and non-stationary environments. When selecting parameters, E-SOINN obtains the correct topological structure when the parameters as $\lambda = 200$ and $age_{\max} = 50$, so in this experiment, $\lambda_S = \{50, 100, 200\}$ and $age_{\max_S} = \{25, 50, 100\}$ is used when comparing with other SOINN-based method.

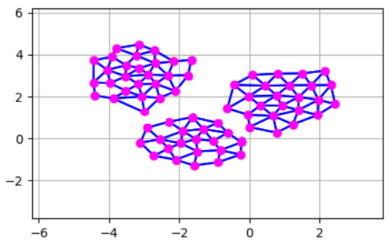

(a) Dataset IV



(b) density distribution in stationary



(c) density distribution in non-stationary


(d) topological structure in stationary

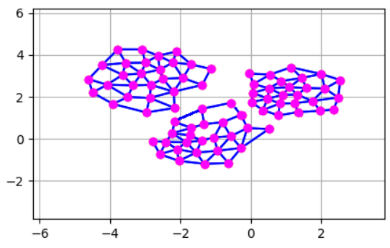

(e) topological strucutre in non-stationary

Figure 3.16 : The result of artificial dataset IV.

The conclusion, how the parameters λ and $age_{\max}$ affect the result is the same as the artificial dataset III experiment. The density distribution obtained by the $eGMM$ method in stationary and non-stationary environments is depicted in Figure 3.16(b) and (c), and the topological structure obtained by Gm-SOINN in stationary and non-stationary environments is shown in Figure 3.16(d) and (e) when $\lambda = 100$, $age_{\max} = 100$. As Figure 3.16(b) and 3.16(c) show, *eGMM* estimates the density distribution accurately and separates the three high-density overlapping subclasses in both environments. The density distribution estimated by *eGMM* is very similar in both environments. Figure 3.16(d) and 3.16(e) show that Gm-SOINN reports the existence of three classes in dataset IV and gives the topological structure of every class. Therefore, the results shown in Figure 3.16 prove that Gm-SOINN satisfactorily obtains the topological structure for a multi-class dataset where the distribution area of every class overlaps significantly. Moreover, we find that the nodes in each topological structure obtained by Gm-SOINN are close to the center of their Gaussian distribution.

Real-world Datasets and Experiment Results

In the first real-world experiments, we used the dataset called fashion-mnist (https://github.com/zalandoresearch/ fashion-mnist) to test Gm-SOINN. The database consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28×28 grayscale image, associated with a label from 10 classes.

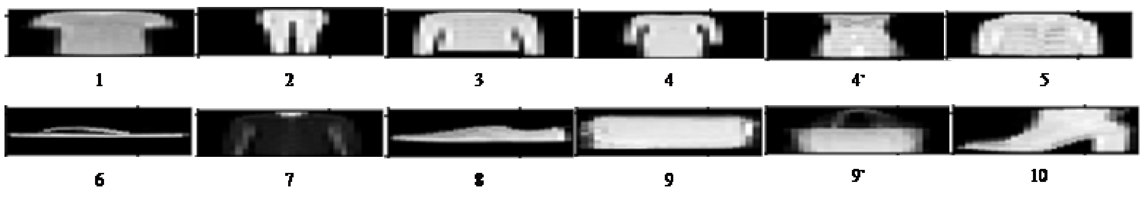


Figure 3.17 : Results of fashion-mnist dataset.

For the stationary environment, 10,000 samples are chosen randomly from the original dataset, and the number of samples in each class is 1000. For the non-stationary environment, samples from class 1 are input to the system in the first stage. After being trained 1000 times, samples from class 2 are input to the system, and so on. When the parameters are set as $\lambda = 800$, $age_{\max} = 100$, for both stationary and non-stationary environments, Gm-SOINN reports that 12 classes exist in the original dataset, which is more than 10 classes in the original datasets. Figure 3.17 gives the prototype vectors (network nodes) of every class, and the results show that Gm-SOINN separates classes 4 and 9 into two classes, respectively, because of the differences between the original image samples in classes 4 and 4', and in classes 9 and 9'.

We then compared Gm-SOINN with three batch methods, three SOINN-based methods (E-SOINN, LB-SOINN, and KDE-SOINN) and one incremental fuzzy method in terms of recognition ratio. Three batch methods used were LinearSVC [55], RandomForestClassifier [121] and KNeighborsClassifier [37]. The incremental fuzzy method is the fuzzy board learning system (FBL) [59]. Note that batch methods cannot learn in a non-stationary environment. The parameters of batch methods were selected by [153]. For SOINN-based methods, the algorithm that classifies nodes into different classes is operated only when the number of input samples is a multiple of λ. In this experiment, we hope the classify operation will operate when the last sample is inputted, so we set λ to a value that can be divided by 60000, i.e., $\lambda_S = \{200, 400, 600, 800\}$ and $age_{\max_S} = \{100, 200, 300, 400\}$.

However, in the non-stationary environment, the recognition ratio obtained by Gm-SOINN and other SOINN-based methods is improved; Gm-SOINN obtains the highest recognition ratio, which is higher than the ratio of the batch methods. From this experiment, we know that Gm-SOINN separates overlapping classes more successfully than other SOINN-based methods, giving Gm-SOINN a higher recognition ratio.

A comparison of the results is shown in Table 3.3. Although Gm-SOINN outperforms other SOINN-based methods and FBL in the stationary environment, the recognition ratio it achieves is less than the ratio obtained by the batch methods.

Table 3.3 : Comparison of results based on fashion-mnist dataset

Methods	Parameter	Recognition ratio	
		S	N
LinearSVC	$C = 100$, "kernel":"linear"	0.943	*
RandomForest Classifier	"criterion":"gini", max_depth = 100, n_estimators = 100	**0.951**	*
KNeighbors Classifier	n_neighbors = 5, p = 1, "weights":"distance"	0.939	*
E-SOINN	$\lambda = 800$, $age_{\max} = 100$, $C_1 = 0.001$, $C_2 = 1.0$	0.881	0.890
LB-SOINN	$\lambda = 800$, $age_{\max} = 100$, $C_1 = 0.001$, $C_2 = 1.0$	0.904	0.912
KDE-SOINN	$\lambda = 800$, $age_{\max} = 100$, $C = 1.0$	0.921	0.933
FBL	$N_r = 7$,$N_t = 1$, $N_e = 53$	0.872	0.861
Gm-SOINN	$\lambda = 800$, $age_{\max} = 100$	0.930	**0.943**

S represents stationary, N represents non-stationary, * represents no value

Next, we show Gm-SOINN to be less sensitive to parameters than other SOINN-based methods. The parameter λ was sampled from (100,1000), and $age_{\max}$ was only set as 100, because the parameter does not influence the number of classes in this experiment. The number of nodes and classes obtained by each SOINN-based method is shown in Table 3.4. Table 3.4 shows that the number of classes obtained by Gm-SOINN by setting different λ values in both the stationary and non-stationary environments is between 8 and 10. The difference between the number of classes in the stationary environment and the non-stationary environment is not significant. KDE-SOINN saves the largest number of nodes. When λ is set to a value smaller than 300, the number of classes obtained by KDE-SOINN is approximately 10, but when λ is set to a value over 400, the number of classes obtained by KDE-SOINN is significantly more than 10. By contrast, when λ is set to a value smaller than 500, E-SOINN and LB-SOINN obtain significantly less than 10 classes as a result of saving fewer nodes, but LB-SOINN obtains a more stable result than E-SOINN. Gm-SOINN saves the second highest number of nodes, so when λ is set to a value smaller than 300, the number of classes obtained by Gm-SOINN is also

Table 3.4 : Comparison of stability results based on fashion-mnist dataset.

		$\lambda = 100$		$\lambda = 200$		$\lambda = 300$		$\lambda = 400$		$\lambda = 500$		$\lambda = 600$		$\lambda = 700$		$\lambda = 800$		$\lambda = 900$		$\lambda = 1000$	
methods	*number*	S	N	S	N	S	N	S	N	S	N	S	N	S	N	S	N	S	N	S	N
E-SOINN	nodes	18	20	32	35	50	53	49	57	69	77	116	123	114	124	138	141	121	127	109	111
	classes	4	4	2	3	4	4	8	8	7	8	15	13	9	12	12	14	7	9	9	9
LB-SOINN	nodes	18	21	34	37	49	51	49	56	71	79	119	120	113	125	137	142	123	133	112	109
	classes	4	4	5	5	5	6	7	8	9	9	13	12	11	12	10	12	8	8	9	8
KDE-SOINN	nodes	40	42	52	55	79	91	98	102	120	131	143	154	176	189	209	221	187	199	168	175
	classes	5	7	4	5	6	8	9	10	13	11	16	15	12	13	14	14	12	14	11	11
Gm-SOINN	nodes	35	36	49	53	79	81	87	90	109	110	125	126	131	132	139	144	140	142	136	139
	classes	4	4	5	6	7	8	8	8	9	9	10	10	10	11	12	12	12	11	11	12

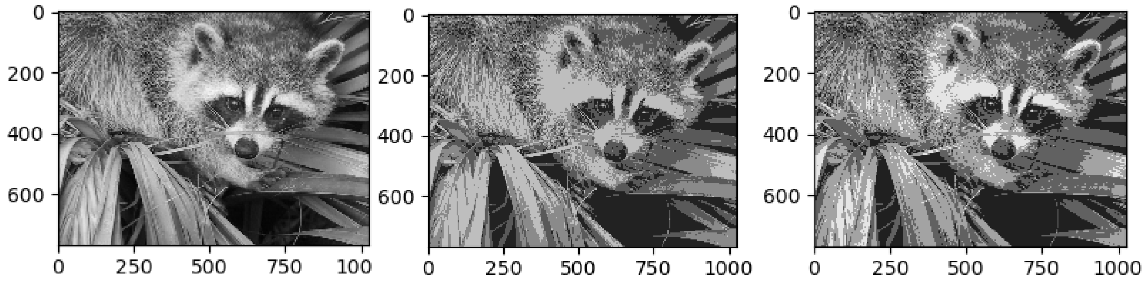
S represents stationary, N represents non-stationary

approximately 10, but when λ is set to a value over than 400, the number of classes obtained by KDE-SOINN is not significantly more than 10. These results illustrate that Gm-SOINN is more stable than other SOINN-based methods, i.e., under more candidates of sets of parameters than in other SOINN-based method, Gm-SOINN can divide the dataset into about 10 classes.

An AT&T face dataset was chosen for the second experiment, which includes 40 distinct subjects and 10 different images per subject. The dataset is commonly used to test the performance of SOINN-based methods. The method in [67] is used to preprocess the dataset for obtaining prototype vectors from the network. Then we used these prototype vectors to classify the original face images and calculate the recognition ratio. The same parameters of each comparative approaches were used, as shown in Table 3.3. Compared with other SOINN-based methods and the FBL algorithm, the recognition ratio obtained by Gm-SOINN is higher in the stationary (96.1%) and in the non-stationary environment (95.8%), but lower than batch methods in the stationary environment.

One goal of topology learning systems is to minimize distortion error. Vector Quantization (VQ) [81] is a typical application in which error minimization is important. For this reason, we experimented using VQ to code a 1024 $\times$ 768 size image of a raccoon face (Figure 3.18(a)). To illustrate the advantage of our proposed method, we compared Gm-SOINN with K-means, GNG and the SOINN-based methods E-SOINN, LB-SOINN, and KDE-SOINN. In the K-means method, we set a set of $k = [5, 7, 9]$, then use a grid search method to select the best k values. In GNG, Gm-SOINN and the SOINN-based methods, we set $\lambda_S = \{25, 50, 100\}$, and $age_{\max_S} = \{25, 50, 100\}$. Since K-means and GNG are unsuitable for sequential input vectors, this experiment only compares the Gm-SOINN method with other SOINN-based methods in the non-stationary environment. Compression ratio and peak signal to noise ratio (PSNR) are often used to evaluate VQ algorithm performance in image compression applications. Here, we use bit per pixel (BPP) to measure the compression ratio, and PSNR is defined as [146]. For VQ, given the same bpp, higher PSNR means less information loss, and the corresponding codebook is consequently better.

Figure 3.18 depicts the reconstructed images are obtained by the Gm-SOINN in stationary and non-stationary environments. The reconstructed results show that our proposed GM-SOINN learns the codebook of raccoon face well in both stationary and non-stationary environments. Table 3.5 summarizes the results of Gm-SOINN and other methods. Table 3.5 shows that in the stationary environment,



(a) original image (b) stationary environment (c) non-stationary environment

Figure 3.18 : Results of VQ in different environments.

K-means induces the lowest information loss but has the least compression ratio. The compression ratios obtained by GNG, Gm-SOINN and other SOINN-based methods are very similar, and are all less than the ratio obtained by K-means. Gm-SOINN obtains the highest PSNR, which is very close to the K-means result. In the non-stationary environment, the compression ratios obtained by Gm-SOINN and the other SOINN-based methods are also very close, and none are significantly different from the results in the stationary environment. However, Gm-SOINN obtains the highest PSNR, which exceeds the K-means result. Therefore, this experiment also shows that for topology learning, the proposed method works well for VQ in both stationary and non-stationary environments.

3.3.4 Computational Complexity

The computational requirements for each stage of the Gm-SOINN algorithm are computed separately. Suppose N is the size of the nodes in the topological structure. In the Gm-SOINN algorithm, the main computation arises from calculating the Gaussian membership between nodes. When a new sample $X(t)$ is inputted we need to find s_1 and s_2 at the first stage, so the computational complexity is $O(N)$ because we only need to calculate the Gaussian membership is $X(t)$ close to every node. When getting the T_{s_1} and T_{s_2}, we need to scan all nodes again in the worst-

Table 3.5 : Comparison of BPP and PSNR with other algorithms

Methods	BPP		PSNR	
	S	N	S	N
K-means	0.146	*	**40.76**	*
GNG	0.142	*	39.73	*
E-SOINN	**0.140**	0.140	40.03	40.61
LB-SOINN	0.140	0.140	40.20	40.97
KDE-SOINN	0.142	0.143	40.57	41.44
Gm-SOINN	0.141	0.141	40.73	**41.59**

S represents stationary, N represents non-stationary, * represents no value

case scenario, i.e., the s_1 and s_2 have no neighbors, so the computational complexity increases to $O(N^2)$. In the second stage, we need to determine whether to connect the s_1 and s_2, so we need to calculate the Eq.(3.73) and Eq.(3.74), but all nodes do not need to be scanned. Therefore, the computational complexity of the second stage is $O(1)$. Finally, we need to update the Gaussian mixture models and the weights vector $W_i(t)$ of all nodes that have a path with s_1. The computational complexity of updating the Gaussian model in the worst case is $O(N)$, i.e., s_1 and s_2 belongs different Gaussian model and cannot be connected, so we need to update two Gaussian models. The computational complexity of update the weights vector $W_i(t)$ of nodes in the worst case is $O(N)$, i.e., all nodes have a path with s_1. If N is the multiple of λ, we need to classify all nodes to different classes and delete noisy nodes. These two operations also need to scan all nodes, so the computational complexity of these two operations is $O(N)$. Therefore, the total computational complexity of Gm-SOINN processing an input sample is $O(N^2 + 1 + N) = O(N^2)$. If there are N samples in the dataset, the computational complexity of Gm-SOINN is $O(N^2)$, which is same as other single-layer SOINN-based methods.

In a practical experiment, for improving the training efficiency, we save a table that records the Euclidian distance between each node, and update it with the following formula when the processing of an input sample is finished:

$$d_{ij}^2\left(t\right) = \left\|\Delta W_i\left(t\right) - \Delta W_j\left(t\right)\right\|^2 + 2d_{ij}\left(t\right)\left\|\Delta W_i\left(t\right) - \Delta W_j\left(t\right)\right\| \tag{3.88}$$

Therefore, we avoid recalculating the Euclidian distance between each node when a new sample is inputted.

3.4 Summary

To solve the regression problem of streaming data under a noisy environment, we proposed an online robust support vector regression which called ORSVR. ORSVR is an exact incremental regression algorithm for handling data streams that transforms the classical v-SVR into a dual regression model. ORSVR captures the characteristics of data distributions very well, which makes ORSVR robust to noise. Additionally, ORSVR determines the up and down-bound functions by breaking the large QPP associated with SVR into two smaller QPPs and solving each simultaneously. The KKT conditions are met for each new sample and maintained for existing samples, but each bound of the divided QPP is simpler than in classical v-SVR, which results in a faster incremental learning speed. However, the ORSVR requires an additional constraint to be compatible with incremental learning. Therefore, the ORSVR approach incorporates several new methods that constitute the incremental learning algorithm for ORSVR. One method introduces a procedure for preparing the initial solution prior to incremental learning. The other is an adjustment step to ensure all the weights of the support vectors are greater than 0. The experimental results demonstrate that ORSVR successfully handles noisy data and is faster than other incremental SVR algorithms.

Besides, we proposed an online topology learning algorithm which called GM-SOINN to filter noisy data from a data stream. Gm-SOINN method is able to learn a topological structure in an online manner from an unlabeled data set. Due to introducing fuzzy logic, unlike other SOINN-based methods, the Gm-SOINN uses a Gaussian membership to indicate the degree to which nodes are identified as a winner (closest node). In addition, based on the Gaussian membership, some theoretical and technical innovations, such as a recursive method of updating nodes and a nonparameter density estimation method which is called eGMM, were proposed. When comparing with other SOINN-based methods, these innovations make that the Gm-SOINN does not have the stability-plasticity dilemma and need fewer user-decided parameters. When comparing with the fuzzy logic system, the Gm-SOINN is not only robust to noise but also obtains better performance in stationary and nonstationary environments. Besides, although fuzzy logic is introduced, the Gm-SOINN is not required to design a method to learn the fuzzy rules. More important, topology learning makes Gm-SOINN can handle some problems, such as VQ, that cannot be directly handled in fuzzy learning systems.

Chapter 4

Streaming Data Regression Under Evolving Environments

4.1 Introduction

Since the generation of streaming data is always in a noisy environment, we have proposed some algorithms to reduce the impact of noisy data in chapter 3. However, more and more studies on streaming data show that the data distribution is not only noisy but also nonstationary [36, 116], i.e., it can evolve. Concept drift [73, 181] refers to this unpredictable change of data distribution in streaming data. For example, one concept in weather data may be the season that is not explicitly specified in temperature data but may influence temperature data [23]. Another example may be customer purchasing behavior over time that may be influenced by the strength of the economy [30], where the strength of the economy is not explicitly specified in the data. The performance of a regression algorithm may become worse when a concept drifts occurs[132]. Hence, concept drifts in streaming data also impact the performance of streaming regression algorithms. To deal with concept drifts, there are two main types of forgetting mechanisms were designed for streaming regression algorithms. In the first type of forgetting mechanism, the forgetting of learned knowledge is independent of the detection result of concept drift. Hence, the forgetting strategy is triggered as long as new data arrive, thereby discarding outdated knowledge. In the second type of forgetting mechanism, the learned knowledge is forgotten according to the detection result of concept drift. However, detect and adapt to the concept drift is a difficult task itself due to concept drift takes various types. Normally, concept drift can be classified into four types [132] according to the degree of drift: 1) sudden drift, which means a concept abruptly changes over a short period of time; 2) gradual drift, a new concept gradually replaces an old one over a period of time; 3) incremental drift, which means the old concept incrementally changes into a new concept over a long period of time; and 4) recurring drift [80, 9], an old concept may reoccur after some time. Different learning strategies should be designed for different degrees of drift, but the learning strategies of most streaming regression algorithms lack diversity.

To solve the streaming data regression problem under evolving environments, we propose continuous support vector regression (C-SVR) for nonstationary streaming data. Like an ensemble-based method [143, 78], in C-SVR a series of $f_i(X)$ are continuously learned in a series of time windows tw_i to determine the relationship between the input and output at different timestamps. However, only one $f_i(X)$ is saved in memory to make a prediction. A new $f_i(X)$ is learned in the new tw_i, and the old $f_{i-1}(X)$, which was learned in the last tw_{i-1}, is discarded. Additionally, in contrast to algorithms [127, 149] that forget all learned knowledge, $f_i(X)$ learning is not independent in C-SVR. A similarity term added to the QPP carries some learned knowledge from $f_{i-1}(X)$ forward into $f_i(X)$. How much-learned knowledge is transferred depends on the degree of the concept drift. Further, because the data in nonstationary streaming data arrive sequentially, the QPP in C-SVR is solved incrementally.

The problem of streaming data mining has been a topic of consistent research in the fuzzy systems community [41] because the generation of streaming data commonly occurs in an uncertain environment. However, classical fuzzy systems cannot deal with streaming data because it is not only uncertain, but infinite, sequential and, more importantly, evolving. It follows directly that, in order to obtain knowledge from streaming data, a fuzzy system needs to meet four learning requirements:

1) recursive and fast learning: data processing is non-iterative; 2) incremental or online learning: data are presented and learned one at a time; 3) memory-efficient learning: no need to save previously processed data; 4) adaptive learning: the structure and parameters of a model can be changed to adapt to concept drift. To address these issues, evolving fuzzy systems (EFSs) [15] have been proposed. EFSs can learn incrementally, possibly even in real-time, on streaming data, and they are also able to adapt to changes in environmental conditions or properties of the data generation process. Normally, EFSs are designed based on fuzzy rules [10, 11] or fuzzy-neuro systems [156, 171]. For example, a flexible fuzzy inference system (FLEXFIS) [138] is designed based on a fuzzy rule. In contrast, a dynamic evolving neural-fuzzy inference system [107] (DENFIS) is designed on fuzzy-neuro systems. Compared to evolving-fuzzy-rule systems, evolving-fuzzy-neuro systems use structure inspired by neural networks to determine their parameters (fuzzy sets and fuzzy rules), so it inherits the advantages of neural networks. In most evolving-fuzzy-neuro systems, an online clustering method is firstly used to obtain the system structure. Then, the parameters (fuzzy sets and fuzzy rules) based on the structure are learned. However, for streaming data regression, evolving-fuzzy-neuro systems still have some drawbacks: 1) determining fuzzy sets is not robust to data sequence, so an algorithm returns different results for one dataset when the data is presented in a different sequence; 2) determining fuzzy rules is complex due to subspaces that can approximate to a Takagi-Sugeno-Kang (TSK) [183] rule need to be obtained, and many parameters need to be optimized; 3) it is difficult to detect and adapt to changes in the data distribution, i.e., concept drift, if the output is a continuous variable.

In summary, the performance of current EFSs for streaming data regression is still limited. Hence, a novel evolving-fuzzy-neuro system, called the topology learning-based fuzzy random neural network (TLFRNN), is proposed. In TLFRNN, similar to current evolving-fuzzy-neuro systems, we still learn neurons to build a neural network as the system structure and utilize fuzzy logic to make an inference, i.e., given an input data at timestamp t, firstly compare input data with the premise part of each neuron to obtain the membership values of each consequent, then aggregate the qualified consequents to produce output data. However, TLFRNN uses a novel method to learn the structure and to determine parameters.

4.2 Continuous Support Vector Regression for Evolving Streaming Data

4.2.1 Continuous Learning Strategy

The learning strategy of our proposed C-SVR is unique. It is not like the learning strategies of other online SVR-based methods such as [140], which use a constant state, i.e., the same parameters, to learn the predictors, as the learning state of C-SVR is adaptive and dependent on the degree of concept drift. It also is not like the learning strategies of other ensemble SVR-based methods such as [128], which need to save multiple learners. In C-SVR, only one learner is saved.

Considering evolving streaming data of $\{(X_1, y_1), \ldots, (X_t, y_t), \ldots\}$ where t is the order the data arrives in, X_t is the input, and y_t is the output. Then, assume that a time window tw_i has size l so that a total of l data that can be captured in order by tw_i. For example, $tw_1 = \{(X_1, y_1), \ldots, (X_t, y_t), \ldots, (X_l, y_l)\}$, where $1 \leq t \leq l$. Our continuous learning strategy is based on this assumption and is shown in Figure 4.1. From Figure 4.1, we can see that a series of $f_i(X)$ are learned in consecutive time windows tw_i to make predictions about the near future. Note that non-stationary streaming data in consecutive tw_i do not overlap. C-SVR's learning process in each tw_i is further divided into the following steps.

Step 1. The data distribution of tw_{i-2} is compared to tw_{i-1} to determine the degree of concept drift, then the learned knowledge in $f_{i-1}(X)$ is transferred to

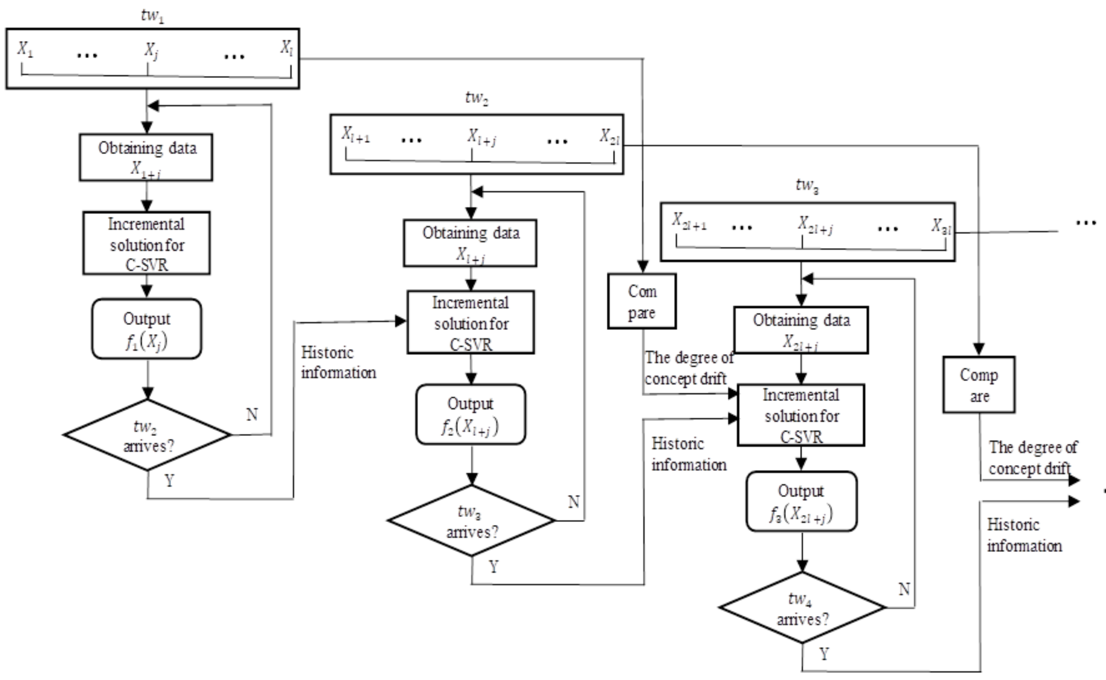


Figure 4.1 : The continuous learning strategy.

$f_i(X)$.

Step 2. One new datum X_t is taken.

Step 3. The incremental solution is used to update $f_i(X_{t-1})$ to $f_i(X_t)$ based on: the new datum X_t; the degree of concept drift; and the learned knowledge of $f_{i-1}(X)$.

Step 4. If a new time window tw_{i+1} does not arrive, go to Step 2. Otherwise, go to Step 1.

Each time a new tw_i arrives, this learning process repeats iteratively to learn the current $f_i(X)$ and continuously adapt to the current concept. However, in the first- and second-time windows, tw_1 and tw_2, the learning process is slightly different. In the first tw_1, we do not need to consider the learned knowledge because we do not have any learned knowledge. In addition, in the first tw_1 and second tw_2, we do not need to consider the degree of concept drift, the reason being that the degree of concept drift is obtained by comparing the $f_i(X)$ and $f_{i+1}(X)$ in our proposed C-SVR, so it is only when we have learned at least two functions that we can determine the degree of drift.

4.2.2 Continuous Support Vector Regression

This section demonstrates how our proposed C-SVR integrates new information from streaming data and the process for forgetting outdated information to adapt to concept drift. First, we provide a general overview of the continuous learning strategy, then each of the components is described in more detail in the individual subsections.

Quadratic Programming Problem

Based on the above section, we know that C-SVR should be able to continuously learn a series of $f_i(X)$ for consecutive time windows tw_i, and also that learning $f_i(X)$ is dependent on $f_{i-1}(X)$ and the degree of concept drift. Thus, when the data distribution of tw_{i-1} drifts slightly relative to the data distribution of tw_{i-2}, $f_i(X)$ should be more similar to $f_{i-1}(X)$ and less similar otherwise. Therefore, the best solution for our expectation is a compromise between (individual) optimality and (neighbor) similarity. Accordingly, we have defined a simple distance in C-SVR to measure d$(f_{i-1}(X), f_i(X))$ so as to quantify the similarity between two neighboring input-output functions $f_{i-1}(X)$ and $f_i(X)$ and, then, to minimize the two-term cost function:

$$\min Err_i^2 + \gamma \cdot d\left(f_i, f_{i-1}\right) \tag{4.1}$$

where the first term is the usual cost function for C-SVR and the second evaluates the total difference between the two consecutive discriminant functions. Here, the free parameter γ regulates the compromise between both terms as with any regularized fitting.

Similar to the classical ε-SVR method [29], we look for a pair of (w_i, b_i) that define an input-output function given by $f_i(X) = \langle w_i \cdot \Phi(X)\rangle + b_i$, where b_i can be computed as follows:

$$b_i = y_i - \langle w_i \cdot \Phi(X)\rangle \tag{4.2}$$

Hence, a simple quadratic distance measure is used to quantify the diversity between functions:

$$d(f_i, f_{i-1}) = \|w_i - w_{i-1}\|^2 \tag{4.3}$$

Applying this measure to (4.1) results in a new QPP for C-SVR, introduced as follows:

$$\begin{aligned}
\min_{\omega_i,\varepsilon,b_i,\xi(*)} & \frac{1}{2}\|w_i\|^2 + C \cdot \sum_{t=1}^{l} (\xi_{it} + \xi_{it}^*) + \frac{\gamma}{2}\left(\|w_i - w_{i-1}\|^2\right) \\
s.t. \quad & - y_{it} + (\langle w_i, \Phi(X_{it})\rangle + b_i) + \varepsilon_i + \xi_{it} \geqslant 0, \\
& y_{it} - (\langle w_i, \Phi(X_{it})\rangle + b_i) + \varepsilon_i + \xi_{it}^* \geqslant 0, \\
& \xi_{it}^{(*)} \geqslant 0, \varepsilon_i \geqslant 0, t = 1, \ldots, l.
\end{aligned} \tag{4.4}$$

The first two terms in Eq. (4.4) correspond to the usual margin and absolute penalty terms in an SVR. The final term in Eq. (4.4) corresponds to the new diversity penalty. Including this term means $f_i(X)$ is dependent on $f_{i-1}(X)$. Hence, the solution to this optimization problem with dual variables is to find the saddle point of the Lagrangian [161]:

$$\begin{aligned}
L_D = & \frac{1}{2}\|w_i\|^2 + C \cdot \sum_{t=1}^{l} (\xi_{it} + \xi_{it}^*) + \frac{\gamma}{2}\left(\|w_i - w_{i-1}\|^2\right) \\
& - \sum_{t=1}^{l} \alpha_{it} (y_{it} - \langle w_i, \Phi(X_{it})\rangle - b_i + \varepsilon_i + \xi_{it}) \\
& - \sum_{t=1}^{l} \alpha_{it}^* (\langle w_i, \Phi(X_{it})\rangle + b_i - y_{it} + \varepsilon_i + \xi_{it}^*) \\
& - \sum_{t=1}^{l} (\eta_{it}\xi_{it} + \eta_{it}^*\xi_{it}^*)
\end{aligned} \tag{4.5}$$

where $\alpha_{it}^{(*)}$ and $\eta_{it}^{(*)}$ are nonnegative Lagrange multipliers. Differentiating L with respect to w_i, b_i, and $\xi_{it}^{(*)}$ and setting the result to zero yields:

$$\begin{aligned}
&\frac{\partial L_D}{\partial w_i} = w_i + \gamma\left(w_i - w_{i-1}\right) - \sum_{t=1}^{l} \Phi\left(X_{it}\right)\left(\alpha_{it} - \alpha_{it}^*\right) = 0 \\
&\qquad \Rightarrow w_i = \frac{\gamma}{1+\gamma} w_{i-1} + \frac{1}{1+\gamma} \sum_{t=1}^{l} \Phi\left(X_{it}\right)\left(\alpha_{it} - \alpha_{it}^*\right) \\
&\frac{\partial L_D}{\partial b_i} = -\sum_{t=1}^{l}\left(\alpha_{it} - \alpha_{it}^*\right) = 0 \\
&\frac{\partial L_D}{\partial \zeta_{it}} = C - \eta_{it} - \alpha_{it} = 0 \Rightarrow \alpha_{it} \in [0, C] \\
&\frac{\partial L_D}{\partial \zeta_{it}^*} = C - \eta_{it}^* - \alpha_{it}^* = 0 \Rightarrow \alpha_{it}^* \in [0, C]
\end{aligned} \tag{4.6}$$

Substituting Eq. (4.6) into L leads to the following dual problem:

$$\begin{aligned}
\min_{\alpha^{(*)}} & \frac{1}{2\left(1+\gamma\right)} \sum_{t=1}^{l} \sum_{j=1}^{l} Q_{tj}\left(\alpha_{it} - \alpha_{it}^*\right)\left(\alpha_{ij} - \alpha_{ij}^*\right) \\
& + \frac{\gamma}{1+\gamma} \sum_{t=1}^{l} (\alpha_{it} - \alpha_{it}^*) y_{(i-1)t} \\
& - \sum_{t=1}^{l} y_{it}\left(\alpha_{it} - \alpha_{it}^*\right) + \sum_{t=1}^{l} \varepsilon_i\left(\alpha_{it} + \alpha_{it}^*\right) \\
s.t.\ & \alpha_{it}, \alpha_{it}^* \in [0, C], t = 1, \ldots, l. \\
& \sum_{t=1}^{l}\left(\alpha_{it} - \alpha_{it}^*\right) = 0
\end{aligned} \tag{4.7}$$

Here, Q_{ij} is a matrix with kernel properties:

$$Q_{tj} = \left\langle \Phi\left(X_{it}\right) \cdot \Phi\left(X_{ij}\right) \right\rangle = K\left(X_{it}, X_{ij}\right) \tag{4.8}$$

where K is a kernel function.

Incremental Solution

With non-stationary streaming data, one typically operates with only one datum at a time. Therefore, we propose an incremental solution to solve the QPP of C-SVR on a data-by-data basis. As shown in the previous section, minimizing α, α^* values create a dual optimization problem Eq. (4.8) that must be solved. Similarly, we can compute another Lagrange function that includes all constraints and introduces other Lagrange multiples:

$$\begin{aligned}
L_D = & \frac{1}{2(1+\gamma)} \sum_{t=1}^{l} \sum_{j=1}^{l} Q_{tj} \left(\alpha_{it} - \alpha_{it}^*\right) \left(\alpha_{ij} - \alpha_{ij}^*\right) \\
& + \frac{\gamma}{1+\gamma} \sum_{t=1}^{l} (\alpha_{it} - \alpha_{it}^*) y_{(i-1)t} \\
& - \sum_{t=1}^{l} y_{it} \left(\alpha_{it} - \alpha_{it}^*\right) + \sum_{t=1}^{l} \varepsilon_{it} \left(\alpha_{it} + \alpha_{it}^*\right) \\
& + \sum_{t=1}^{l} \left[\mu_{it} \left(\alpha_{it} - C\right) + \mu_{it}^* \left(\alpha_{it}^* - C\right)\right] \\
& + \zeta_i \sum_{t=1}^{l} \left(\alpha_{it} - \alpha_{it}^*\right) - \sum_{t=1}^{l} \left(\delta_{it}\alpha_{it} + \delta_{it}^*\alpha_{it}^*\right) \\
s.t.\ & \delta_{it}^{(*)}, \mu_{it}^{(*)}, \zeta_i \geqslant 0, \quad t = 1, \ldots, l
\end{aligned} \tag{4.9}$$

Computing partial derivatives of the Lagrangian:

$$\begin{aligned}
\frac{\partial L_D}{\partial \alpha_{it}} = & \frac{1}{2(1+\gamma)} \sum_{j=1}^{l} Q_{tj} \left(\alpha_{ij} - \alpha_{ij}^*\right) + \frac{\gamma}{1+\gamma} y_{(i-1)t} \\
& - y_{it} + \varepsilon_i - \delta_{it} + \mu_{it} + \zeta_i = 0 \\
\frac{\partial L_D}{\partial \alpha_{it}} = & - \frac{1}{2(1+\gamma)} \sum_{j=1}^{l} Q_{tj} \left(\alpha_{ij} - \alpha_{ij}^*\right) - \frac{\gamma}{1+\gamma} y_{(i-1)t} \\
& + y_{it} + \varepsilon_i - \delta_{it}^* + \mu_{it}^* - \zeta_i = 0 \\
\frac{\partial L_D}{\partial \zeta_i} = & \sum_{j=1}^{l} \left(\alpha_{ij} - \alpha_{ij}^*\right) = 0
\end{aligned} \tag{4.10}$$

Then to increase readability, we can replace ξ_i with b: $\xi_i = b$. This is still an optimization problem with a convex domain, so the sufficient conditions for a point to be optimum are (Kuhn-Tucker Theorem) [48]:

$$\begin{aligned}
&\alpha_{it} \in [0, C], \\
&\alpha_{it}\left(\sum_{j=1}^{l} Q'_{tj}\left(\alpha_{ij}-\alpha^*_{ij}\right)+b_i+y'_{(i-1)t}+\varepsilon_i-y_{it}-\delta_{it}+u_{it}\right)=0 \\
&\alpha^*_{it} \in [0, C], \\
&\alpha^*_{it}\left(-\sum_{j=1}^{l} Q'_{tj}\left(\alpha_{ij}-\alpha^*_{ij}\right)-b_i-y'_{(i-1)t}+\varepsilon_i+y_{it}-\delta^*_{it}+u^*_{it}\right)=0 \\
&\delta_{it} \geqslant 0, \quad \delta_{it}\alpha_{it}=0 \\
&\delta^*_{it} \geqslant 0, \quad \delta^*_{it}\alpha^*_{it}=0 \\
&\mu_{it} \geqslant 0, \quad \mu_{it}\left(\alpha_{it}-C\right)=0 \\
&\mu^*_{it} \geqslant 0, \quad \mu^*_{it}\left(\alpha^*_{it}-C\right)=0
\end{aligned} \tag{4.11}$$

where

$$Q'_{tj}=\frac{1}{2\left(1+\gamma\right)}Q_{tj} \tag{4.12}$$

and

$$y'_{(i-1)t}=\frac{\gamma}{(1+\gamma)}y_{(i-1)t} \tag{4.13}$$

Further, if we assume the estimated $f_i(X_t)$ in the tw_i can be written as $f'_i\left(X_t\right)=\sum_{j=1}^{l} Q'_{tj}\left(\alpha_{ij}-\alpha^*_{ij}\right)+b_i$, then a marginal function can be obtained:

$$h_i\left(X_t\right)=\left(f'_i\left(X_t\right)+y'_{(i-1)t}\right)-y_{it} \tag{4.14}$$

For simplicity, θ_{ij} can be defined as the difference between α_{ij} and α^*_{ij}, and the Kuhn-Tucker Theorem (KKT) conditions can be obtained by replacing θ_{ij} in Eq. (4.13):

$$\begin{cases}
h_i\left(X_t\right) \geq +\varepsilon & \theta_{ij}=-C \\
h_i\left(X_t\right) = +\varepsilon & \theta_{ij}\in[-C,0] \\
h_i\left(X_t\right) \in [-\varepsilon,+\varepsilon] & \theta_{ij}=0 \\
h_i\left(X_t\right) = -\varepsilon & \theta_{ij}\in[0,C] \\
h_i\left(X_t\right) \leq -\varepsilon & \theta_{ij}=+C
\end{cases} \tag{4.15}$$

Using these KKT conditions, the data can be divided into three sets:

1) Support set:

$$S_S = \{t| (\theta_{ij} \in [0, +C] \wedge h_i (X_t) = -\varepsilon_i) \vee (\theta_{ij} \in [-C, 0] \wedge h_i (X_t) = +\varepsilon_i)\}$$

2) Error set:

$$E_S = \{t| (\theta_{ij} = -C \wedge h_i (X_t) \geqslant +\varepsilon_i) \vee (\theta_{ij} = +C \wedge h_i (X_t) \leqslant -\varepsilon_i)\}$$

3) Remaining set:

$$R_S = \{t|\theta_{it} = 0 \wedge |h_i (X_t)| \leqslant \varepsilon_i\}$$

Hence, our proposed incremental solution aims to find a way to maintain consistent KKT conditions Eq. (4.15) when new data is added to one of the three sets.

The support set is defined as $S_S = \{s_1, s_2, \ldots, s_{l_s}\}$ and, to ensure all data satisfy the KKT conditions Eq. (4.15) during the learning procedure, the following linear system occurs when S_S is changed:

$$\begin{aligned} &\sum_{j \in S_S} Q'_{tj} \Delta\theta_{ij} + \Delta b_i = -Q'_{tc} \Delta\theta_{ic} \\ &\sum_{j \in S_S} \Delta\theta_{ij} = -\theta_{ic} \end{aligned} \tag{4.16}$$

These equations can be rewritten in an equivalent matrix form:

$$\begin{bmatrix} \Delta b_i \\ \Delta\theta_{s_1} \\ \vdots \\ \Delta\theta_{s_{ls}} \end{bmatrix} = \beta \Delta\theta_{ic} = \begin{bmatrix} \beta_b \\ \beta_{s_1} \\ \vdots \\ \beta_{s_{ls}} \end{bmatrix} \Delta\theta_{ic} = -R \begin{bmatrix} 1 \\ Q'_{s_1 c} \\ \vdots \\ Q'_{s_{ls} c} \end{bmatrix} \Delta\theta_{ic} \tag{4.17}$$

where matrix R can be defined as:

$$R = \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & Q'_{s_1 s_1} & \cdots & Q'_{s_1 s_{ls}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q'_{s_{ls} s_1} & \cdots & Q'_{s_{ls} s_{ls}} \end{bmatrix}^{-1} \tag{4.18}$$

For the error and remaining sets, we have defined a "not support set" as $N_S = E_S \cup R_S = \{n_1, n_2, \ldots, n_{l_n}\}$. Again, to ensure all data satisfy the KKT conditions Eq. (4.9) during the learning procedure, the following linear system occurs when N_S is changed:

$$\Delta h_i (X_t) = \sum_{j=1}^{l} Q'_{tj} \Delta\theta_{ij} + Q'_{tc} \Delta\theta_{ic} + \Delta b_i \tag{4.19}$$

Rewriting the variation of the h formula in matrix notation gives:

$$\begin{bmatrix} \Delta h_i (X_{n_1}) \\ \vdots \\ \Delta h_i (X_{n_{\mathrm{ln}}}) \end{bmatrix} = \omega \Delta\theta_{tc} \tag{4.20}$$

where matrix ω can be defined as:

$$\omega = \begin{bmatrix} \Delta Q'_{n_1 c} \\ \vdots \\ \Delta Q'_{n_{\mathrm{ln}} c} \end{bmatrix} + \begin{bmatrix} 0 & 1 & \cdots & 1 \\ 1 & Q'_{n_1 s_1} & \cdots & Q'_{n_1 s_{ls}} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q'_{n_{\mathrm{ln}} s_1} & \cdots & Q'_{n_{\mathrm{ln}} s_{ls}} \end{bmatrix} \beta \tag{4.21}$$

From Eq.(4.19) and Eq. (4.22), we can see that the values of $\Delta\theta_i$ and Δb_i can be updated by computing β, and the value of Δh_i can be updated by computing ω. Therefore, the $f_i(X)$ can be learned with Algorithm 4.1:

Algorithm 4.1 shows that the minimal increment $\Delta\theta_i$ can be obtained by the same method used in [140]. The inverse matrix R is updated in two situations: adding the first data or adding other data.

Algorithm 4.1 Incremental solution

```
Input: a new data $(X_c, y_c)$
Parameter: initial $\theta_{ic} = 0$, $b_i = 0$
Output: $f_i(X)$
1:  Compute $h_i(X_c)$
2:  if $|h_i(X_c)| < \varepsilon_i$:
3:      Add $X_c$ to the remaining set
4:      Exit
5: else:
6:  while $X_c$ is not added into a set:
7:          Compute $\beta$ and $\omega$ according to (19) and (23)
8:          Compute the minimal increment $\Delta\theta_i$.
9:          Update $\Delta\theta_i$, $\Delta b_i$, $\Delta h_i(X_c)$, $S_S$, $E_S$ and $R_S$.
10:         Update the inverse matrix $R$.
11:     end while
12: end if
13: output $f_i'(X) + y_{i-1}t'$
```

When adding the first data in R, the update follows

$$R = \begin{bmatrix} -Q'_{11} & 1 \\ 1 & 0 \end{bmatrix} \tag{4.22}$$

When adding other data, R is updated with

$$R_{\text{new}} = \begin{bmatrix} & & & 0 \\ & \text{R} & & \vdots \\ & & & 0 \\ 0 & \dots & 0 & 0 \end{bmatrix} + \frac{1}{\omega} \begin{bmatrix} \beta \\ 1 \end{bmatrix} \begin{bmatrix} \beta^T & 1 \end{bmatrix} \tag{4.23}$$

where β and ω are recomputed to follow each situation. If one datum is added to the S_S, ω is defined as:

$$\omega_i = Q'_{cc} + \begin{bmatrix} 1 & Q'_{cS_1} & \cdots & Q'_{cS_{l_s}} \end{bmatrix} \beta \tag{4.24}$$

As outlined, this incremental solution means C-SVR can handle streaming data because $f(X)$ is updated data-by-data.

4.2.3 Concept Drift Degree

From Eq. (4.7), we know that the free parameter γ regulates the diversity of $f_i(X)$ and $f_{i-1}(X)$. Low values of γ will almost decouple $f_i(X)$ and $f_{i-1}(X)$, which allows for increased flexibility. Conversely, high γ values will produce a $f_i(X)$ that is almost identical to $f_{i-1}(X)$. Therefore, parameter γ can be seen as the degree of concept drift. Parameter γ should increase to increase the similarity of $f_i(X)$ to $f_{i-1}(X)$ because $f_{i-1}(X)$ should gradually approximate the real input-output function, which does not drift. In contrast, parameter γ should decrease to differentiate $f_i(X)$ from $f_{i-1}(X)$ because the real input-output function has drifted and $f_{i-1}(X)$ no longer represents it.

To determine the value of parameter γ, we have incorporated our previous work, i.e., a competence-based drift detection method [135], into C-SVR. The reason for choosing this method is when using an SVR to learn $f(X)$, and only the support vectors are saved because the $f(X)$ is constructed only by support vectors. Hence, only support vectors can be analyzed to identify concept drift. However, the number of support vectors is small. Our proposed competence-based drift detection method is able to achieve stable and good results, especially for small samples [135]. Therefore, incorporating our proposed competence-based drift detection method results in our proposed C-SVR achieving more stable and better results than other concept drift detection methods [132].

The key idea of the competence-based drift detection method [43] is that changes in the probability distribution of data should reflect its competence. Hence, our proposed competence-based drift detection method compares two data distributions in the competence space instead of the original data feature space. Assume two datasets $S_1, S_2 \subseteq CB$(case-base), the competence-based empirical distance between S_1 and S_2 is defined as [135]:

$$d^{CB}\left(S_1, S_2\right) = 2 \times \sup_{A \in \widehat{\mathrm{A}}} \left|S_1^{CB}\left(A\right) - S_2^{CB}\left(A\right)\right| \tag{4.25}$$

where $0 \leq d^{CB}\left(S_1, S_2\right) \leq 1$. Here, the more different the distribution of the two groups of data, the larger the competence-based empirical distance.

Therefore, to get the value of parameter γ, a group of support vectors S_{i-2} in tw_{i-2} and a group of support vectors S_{i-1} in tw_{i-1} will be joined together to form one CB. After modeling all the support vectors in CB in the competence model, two related closures $RC^{CB}(S_1)$ and $RC^{CB}(S_2)$ are separately obtained. Using the density of $RC_i^{CB}(S) \in RC^{CB}(S)$, each group of support vectors can be represented by their distribution in the competence space. The difference between the support vectors of the two groups can be measured by the competence-based empirical distance $d^{CB}(S_{i-1}, S_{i-2})$. However, the distance $d^{CB}(S_{i-1}, S_{i-2})$ cannot directly be used to set the γ_i because of the range of γ_i is $[0, +\infty]$, and if the degree of concept drift increases, the value of γ_i should decrease. Therefore, we use the following formulation to calculate γ_i:

$$\gamma_i = \begin{cases} +\infty & \text{where } G(d^{CB}) = 0 \\ \frac{1}{G(d^{CB}(S_{i-1}, S_{i-2}))} & \text{where } 0 < G\left(d^{CB}\right) < +\infty \\ 0 & \text{where } G(d^{CB}) = +\infty \end{cases} \tag{4.26}$$

where $G(d^{CB}(S_{i-1}, S_{i-2}))$ can be calculated by:

$$G = \begin{cases} 0 & \text{where } d^{CB} = 1/2 \\ \frac{1}{1-1/(n-1)} - 1 & \text{where } d^{CB} = 1 - 1/n, n > 2 \\ 1/d^{CB} & \text{other} \end{cases} \tag{4.27}$$

Although $d^{CB}(S_{i-1}, S_{i-2})$ is a time-consuming task, the calculation of $d^{CB}(S_{i-1}, S_{i-2})$ is fast in C-SVR because the number of support vectors in each group is small.

4.2.4 Experiments

This section details the experiment results to compare C-SVR to the state-of-the-art approaches. The evaluations involve different scenarios using both artificial datasets and real-world datasets (see Table 4.1). The artificial datasets allowed us to control the relevant parameters and to empirically evaluate the algorithms with specific types of concept drift. The real-world datasets enabled us to evaluate the merit of the proposed approach in practical scenarios. However, note that in these latter experiments, it was not always possible to precisely state when a drift occurred or even whether there was drift at a specific timestamp. All experiments were conducted in Python 3.5 on a PC running Windows 7 with an Intel Core i5 processor (2.40 GHz) and 8-GB RAM.

Experimental Setup

To validate the effectiveness of C-SVR, we compared its performance to AON-SVR [140], VPI-SVR [149], incremental SVR based on time windows (TW-SVR), AddExp.C-SVR [111], and Learn++.NSE-SVR [51].

AON-SVR is a popular online SVR algorithm for handling streaming data because its incremental solution of the QPP was widely used in other online SVR algorithms, including those mentioned above. For example, VPI-SVR and AON-SVR use the same incremental solution. The only difference between the two is that VPI-SVR uses different parameters in different time windows to learn $f_i(X)$. The incremental solution for TW-SVR is also the same as AON-SVR, but when a new time window arrives, the information of the previous $f_{i-1}(X)$ is completely discarded and a new $f_i(X)$ is learned. AddExp.C and Learn++.NSE are two well-known adaptive ensembles for dealing with concept drift. They are adaptive datum-based ensembles (AddExp.C) and adaptive batch-based ensembles (Learn++.NSE), respectively. In our experiments, we made a small change to these two ensemble algorithms. In AddExp.C-SVR, new ε-SVR was learned in the new tw_i through the incremental solution from AON-SVR. Then the AddExp.C algorithm was used to update the weights of each expert prediction $f_i(X)$ to ultimately make the predictions. In contrast, each time step of the Learn++.NSE consists of a batch of data (these can be seen as the size of the time windows). Hence, the batches are considered to have a size of m. The weak learner is the ε-SVR. However, Learn++.NSE-SVR does not learn a new ensemble $f_i(X)$ in a new tw_i. Rather the Learn++.NSE algorithm is used to decide whether to learn a new ensemble, but the $f_i(X)$ is also updated in a tw_i by the incremental solution, which is used in AON-SVR for handling streaming data. Learn++.NSE is also used to update the weights of each ensemble $f_i(X)$ to make a prediction.

All the above mentioned online SVR algorithms are designed with a forgetting strategy. AON-SVR includes a decremental learning algorithm. When the number of processed data exceeds a certain number, the information of one previous data is ignored to update $f_i(X)$. The "certain number" can be seen as the size m of a time window. VPI-SVR and TW-SVR uses a simple forgetting strategy, i.e., the previous $f_{i-1}(X)$ is completely forgotten and a new $f_i(X)$ is learned when a new tw_i arrives. With the ensemble SVR algorithms, the forgetting strategy used follows the best result indicated by its authors. With AddExp.C-SVR, the $f_i(X)$ with the lowest weight was excluded. With Learn++.NSE-SVR, the model j with the largest current error was excluded. In all ensemble-based SVR algorithms, the maximum number of models in the ensemble was set to 20. This choice was considered the most suitable for all the ensembles since the maximum number of models usually varies between 15 and 30 [111, 51], and the use of more models linearly increases the processing time of the experiments.

Turning to the parameters of the above algorithms, two parameters C and ε needed to be set in all algorithms, each of which has been shown to affect performance [29]. However, the automatic method [149] of setting parameters C and ε in VPI-SVR means that only two predefined parameters C and ε needed to be set for the first-time window tw_1. In subsequent time windows, these parameters were set automatically. Some of the other parameters in the ensemble SVRs needed to be set for ensemble learning. For example, in AddExp.C, the parameters were set based on the pilot studies from [111]: factor for decreasing weights $\beta = 0.5$, factor for new expert weight $\gamma = 0.1$, and loss required to add a new expert $\tau = 0.5$. The parameters for Learn++.NSE were set according to the authors' suggestions [168]: slope $a = 0.5$ and infliction point $b = 10$.

Artificial Datasets

To illustrate that our method C-SVR has better performance to deal with non-stationary streaming data, this section presents some experimental results for the artificial datasets with concept drift. However, in this research, we only consider datasets with incremental, sudden and gradual drift in the following experiments (see Table 4.1), the reason being that research into these three types of drift detection focuses on how to detect the concept transformation process rapidly. In contrast, the study of recurring drift emphasizes the use of historical concepts - that is, how to find the best matched historical concepts in the shortest time.

The first is the drifting hyperplane dataset [62]. This is a well-known drift dataset for evaluating algorithms that deal with concept drift. It contains noise and incremental drifts and non-recurring drifts and is similar to the one proposed in [111] (AddExp). The whole dataset consists of 10 inputs with a uniform distribution over the interval of $[0, 1]$ and 1 output data $y_i \in [0, 1]$; and there are 2000 data ($M = 2000$) in the dataset. The dataset contains four concepts, with each concept holding 500 data. A random variate noise uniformly distributed in the interval of $[-0.1, 0.1]$ is injected into each output y_i (for $i = 1, \ldots, M$). The value of y_i is set to 0 or 1 if its value is less than 0 or greater than 1, respectively.

In this experiment, the size of the time window is 200. Then, we set $C = 100$ and $\varepsilon = 0.1$ for each of the above-mentioned algorithms. Each method is evaluated using the root mean square error (RMSE), which is calculated using the predicted outputs and the real outputs from the online data. RMSE is a widely used metric to evaluate models. It provides a loss function that penalizes larger errors. A lower RMSE means the algorithm performed better. Figure 4.2 shows the results of our comparison based on the drifting hyperplane dataset.

From Figure 4.2, in general, we can see that, in most time windows that have no incremental drift, such as tw_1 ($[0-200]$), tw_2 ($[200-400]$), tw_7 ($[1200-1400]$), and tw_{10} ($[1800-200]$), the RMSE obtained by each online SVR-based algorithm shows a convergent trend as the training data increases. However, when incremental drift occurs, i.e., $M = 500$ and $M = 1500$, the RMSE obtained by every online SVR-based algorithm shows an upward tendency as the training data increases. The trend of the RMSE is identical because each online SVR-based algorithm learns $f_1(x)$ with the same parameters in the tw_1 ($[0-200]$). However, because TW-SVR and VPI-SVR relearn a new function $f_i(x)$ in each new tw_i, such as tw_2 ($[200-400]$),

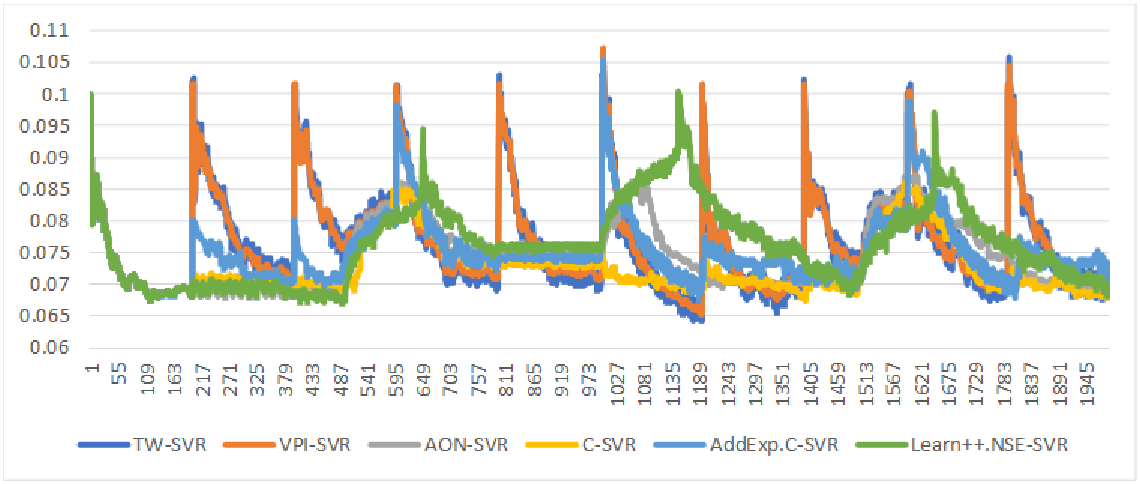


Figure 4.2 : Results on the drifting hyperplane dataset (X-axis represents the size of instances, and Y-axis represents RMSE).

the RMSE fluctuates sharply when each tw_i arrives, i.e., RMSE instantly surges and then gradually declines to final convergence. Although AddExp.C-SVR also relearns a new function $f_i(x)$ in each new tw_i, the old function $f_{i-1}(x)$ is not discarded, so the RMSE does not sharply increase. In contrast, AON-SVR and Learn+.NSE-SVR do not relearn a new function $f_i(x)$ when a new tw_i arrives, so the RMSE remains stable. However, when the first time window tw_i arrives after an incremental drift, such as tw_4 ($[600 - 800]$) and tw_9 ($[1600 - 1800]$), the RMSE convergence speed for AON-SVR and Learn++.NSE-SVR is slower than that of TW-SVR, VPI-SVR, and AddExp.C-SVR. However, each has a different reason for converging more slowly. For AON-SVR, the reason is that the information of the last $f_{i-1}(x)$ is not completely discarded. However, for Learn++.NSE-SVR, the reason is that the incremental drift cannot be detected in time, so the RMSE will continue to grow until Learn+.NSE-SVR decides to learn a new function. Also, the RMSE instantaneously surges at the moment of learning a new function. In addition, at time $M = 1000$, although incremental drift occurs, TW-SVR, VPI-SVR, and AddExp.C-SVR learn two distinct functions in tw_5 ($[800 - 1000]$) and tw_6 ($[1000 - 1200]$), and the RMSE obtained by TW-SVR, VPI-SVR, and AddExp.C-SVR is not influenced by concept drift, but the RMSE obtained by AON-SVR and Learn+.NSE-SVR is influenced by incremental drift, so it increases at that time. C-SVR also relearns a new function $f_i(x)$ in the new tw_i but, unlike the above algorithms, the information of the last function $f_{i-1}(x)$ is not completely discarded so the RMSE does not instantaneously surge when a new tw_i arrives. The RMSE only slightly trends up, such as in the early stages of tw_2 ($[200-400]$). However, when the first time window arrives after an incremental drift occurs, such as tw_4 ($[600-800]$) and tw_9 ($[1600-1800]$), C-SVR's RMSE convergence speed is slower than TW-SVR, VPI-SVR, but faster than AON-SVR, AddExp.C-SVR and Learn+.NSE-SVR. In addition, although incremental drift occurs at time $M = 1000$, C-SVR's RMSE is not affected by the drift because it learns two distinct functions in tw_5 ($[800-1000]$) and tw_6 ($[1000-1200]$).

To further illustrate C-SVR's advantage, we experimented with a drifting Friedman's function as the sudden drift dataset. Friedman's function has linear and non-linear relations between the input variables and the output variable. The function contains five input variables, $x^1, \ldots, x^5$, and 1 output variable, y_i. The input variables are uniformly distributed over the interval of $[0,1]$. To create drifting scenarios, one drifting dataset using the original Friedman's function was produced according to [98]. The dataset also has 2000 data and sudden concept drifts. The concept drift is localized in two distinct regions of the instance space. The first region with drift is defined with the conjunction of inequalities $R_1 = \{x_2 < 0.3 \wedge x_3 < 0.3 \wedge x_4 > 0.7 \wedge x_5 < 0.3\}$, and the second region is defined with the conjunction of inequalities $R_1 = \{x_2 > 0.7 \wedge x_3 > 0.7 \wedge x_4 < 0.3 \wedge x_5 < 0.7\}$. Let M denote the size of the dataset. There are three points of abrupt change in the training dataset, the first one at $\frac{1}{4}M$ data, the second one at $\frac{1}{2}M$ data and the third at $\frac{3}{4}M$ data. The complete description of these drifting data can be found in [98].

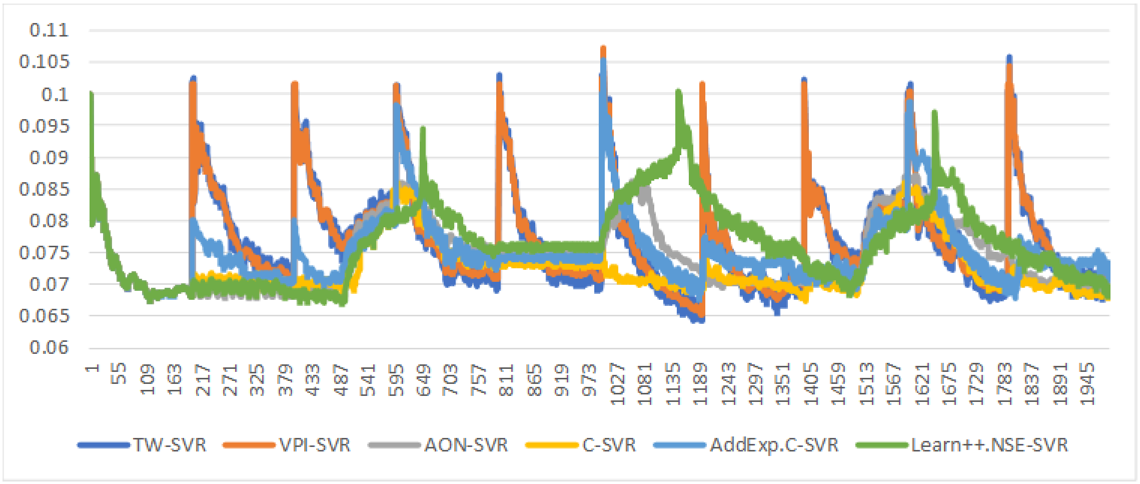


Figure 4.3 : Results on the drifting Friedman's dataset (X-axis represents the size of instances, and Y-axis represents RMSE).

Figure 4.3 shows the results of our comparison based on the drifting hyperplane dataset. From Figure 4.3, we reach a similar conclusion to the first experiment, in general. For example, in most time windows that have no sudden drift, such as tw_1 ($[0-200]$), tw_2 ($[200-400]$), tw_7 ($[1200-1400]$), and tw_{10} ($[1800-200]$), the RMSE obtained by each online SVR algorithm shows a convergent trend as the training data increases. However, unlike the previous experiment, when sudden drift occurs, i.e., $M = 500$ and $M = 1500$, the RMSE for every online SVR-based algorithm shows a more dramatic and rapid upward trend. Therefore, this also illustrates that sudden drift has a more serious impact on performance than incremental drift.

The last is a dataset with gradual drift. In this dataset, there are also four concepts with a function. The function contains 3 input variables, $x^1, \ldots, x^3 \in [0,1]$, and 1 output variable, y_i. We also simulate a window (which includes 200 instances) of change where the probability of a given instance belonging to the current concept of the new concept is governed by a sigmoid function. Basically, at the start of the window, the probability that one instance is drawn from concept 1 is higher, while at the end of the window, the probability of it being drawn from concept 2

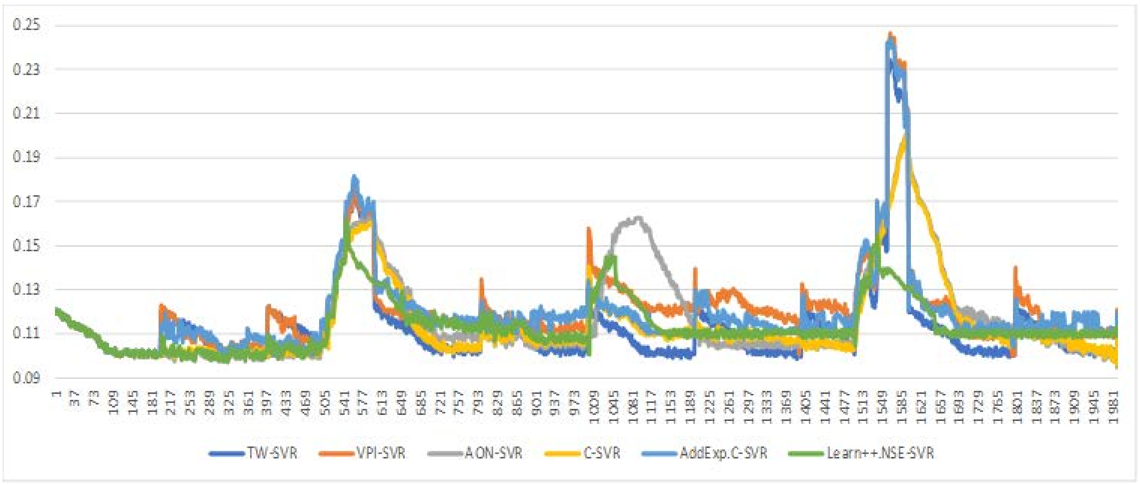


Figure 4.4 : Results on the gradual drifting dataset (X-axis represents the size of instances, and Y-axis represents RMSE).

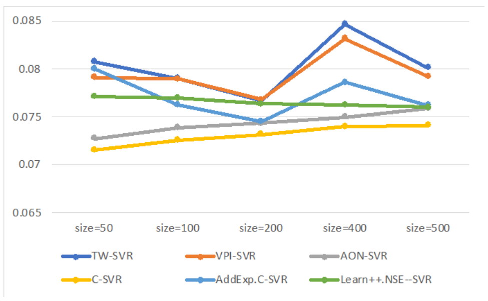
increases, after the window is overall instances will be drawn from concept 2 (the "new concept" is now stable). In this example, there are three gradual drifts centered around instances 500, 1000, and 1500 respectively.

Figure 4.4 shows the results of our comparison based on the gradual drifting dataset. From Fig. 4.4, we still reach a similar conclusion to the first and second experiment, in general. However, unlike the previous two experiments, when the gradual drift window arrives, i.e., [400-600] and [1400-1600], the RMSE for every online SVR-based algorithm shows a more rapid upward trend in [500-600] and [1500-1600] than in [400-500] and [1500-1600].

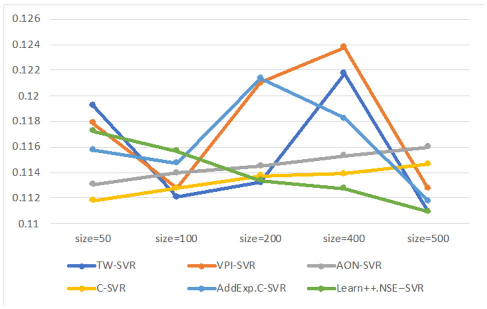
In summary, these results from the aforementioned three experiments show that our proposed C-SVR achieves more stable and better results than the other online SVR-based algorithms for non-stationary streaming data regression. The reason is C-SVR learns $f_i(X)$ which is dependent on $f_{i-1}(X)$. Hence, C-SVR is not like AON-SVR, VPI-SVR, TW-SVR, i.e., the learned knowledge is discarded in a new tw_{i+1} with or without concept drift in tw_i. In contrast, when there are no concept drifts in tw_i, the learned knowledge can be saved in C-SVR, thereby the learning of $f_i(X)$ does not need to start from an initial state and can make an accurate prediction as soon as possible. On the other hand, C-SVR is not like AddEXP.C-SVR, Learn++.NSE-SVR, i.e., outdated $f_i(X)$ cannot be discarded in time. In C-SVR, if there are concept drifts in tw_i, the learned knowledge will be discarded, therefore the learning of $f_i(X)$ is not affected by outdated knowledge, thereby reducing the impact of outdated models on the prediction results.

Next, we discuss the impact of parameters on the performance of each online SVR algorithm. Two parameters C and ε need to be set in each online SVR algorithm. The impact of these two parameters on performance is discussed in former publications [140, 82]. Therefore, there is no need to discuss the impact of these two parameters on each online SVR-based algorithm here. Each online SVR-based algorithm uses the same parameters C and ε. However, from the above two experiments' results, we can see that the performance of each algorithm in different time windows is different. Therefore, to further illustrate the advantage of our proposed C-SVR, we designed the third experiment to discuss the impact of the time windows' size on the performance of each algorithm. In this experiment, we set the size of the time window to 50, 100, 200, 400, and 500. We then compared the RMSE and the learning time of each algorithm based on time windows of different sizes. Figure 4.5 shows the comparative results.

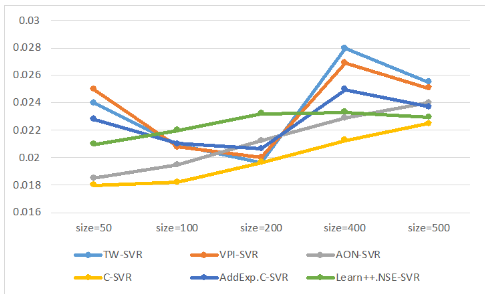
Figures 4.5 (a), 5 (b) and 5 (c) show that the average RMSE obtained by TW-SVR, VPI-SVR, and AddExp.C-SVR, in three datasets with concept drift, are significantly different if using time windows of different sizes. However, when the new time window arrives at exactly the time that each concept drift occurs, the average RMSEs obtained by TW-SVR, VPI-SVR, and AddExp.C-SVR are smaller than when the concept drift does not occur when that time window arrives. In contrast, there is little difference between the average RMSE obtained by AON-SVR and Learn+.NSE-SVR and our proposed C-SVR when using time windows of different



(a) Hyperplane dataset



(b) Friedman's dataset

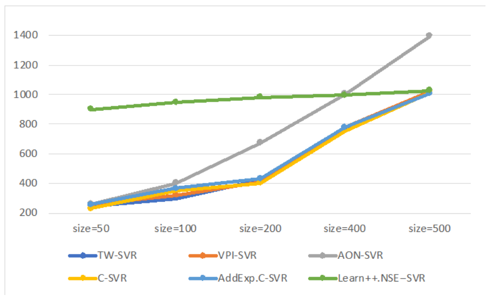


(c) Gradual drift dataset

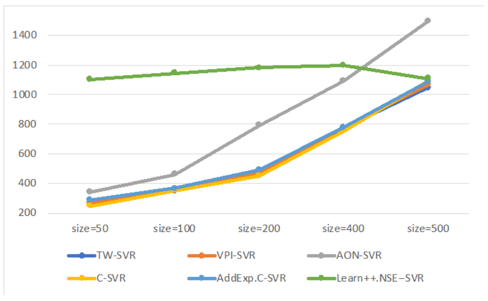
Figure 4.5 : RMSE according to the size of time-window.

sizes. Of these, the RMSE obtained by AON-SVR and C-SVR increases slightly with the increase of the time window's size, while Learn+.NSE-SVR decreases slightly with the increase of the time window's size.

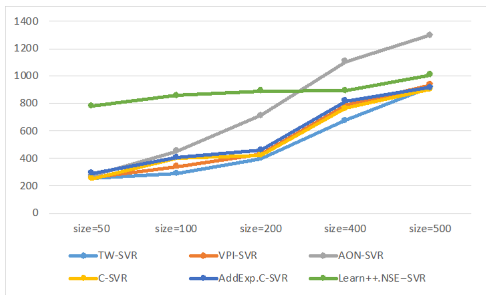
Figures 4.6 (a), 4.6 (b) and 4.6 (c) show that the learning time of TW-SVR, VPI-SVR, AddExp.C-SVR, AON-SVR, and C-SVR increases with the increase of the time window's size. The reason for this is that the training data increases with the increase in the size of the time windows, which results in the increase of data that belongs to the three sets $-S_S$ (Supporting set), E_S (Error set), and R_S (Remaining set). Therefore, when learning new data, the data in the three sets that need to be also adjusted increases, thereby increasing learning time. AON-SVR needs to choose a datum to discard, so the learning time is longer than other online SVR algorithms. Learn++. NSE-SVR only relearns a new function when concept drift occurs, so its



(a) Hyperplane dataset



(b) Friedman's dataset



(c) Gradual drift dataset

Figure 4.6 : Time according the size of time-window.

learning time is less affected by the size of the time window, but when the size of the time window is less than 500, the learning time of Learn++. NSE-SVR is much longer than the other online SVR-based algorithms.

We also carry out comparative experiments using online SVR-based algorithms and other types of online regression algorithms. An incremental SVR algorithm, named I-SVR, is included to demonstrate the advantages of a forgetting mechanism. I-SVR is without a forgetting mechanism and uses an incremental solution as in [140] to solve the QPP of a classical ε-SVR. Online SVR-based algorithms with a forgetting mechanism include TW-SVR, VPI-SVR, AON-SVR, AddExp.C-SVR, Learn++.NSE-SVR, and our proposed algorithm C-SVR. For each dataset, we set time windows of different sizes; the details are given in Table 4.1. Based on the size of the time window, we used a parameter optimization method [18] to find the approximate values of the parameters C and ε of all online SVR-based algorithms. For some parameters that need be set to AddExp.C and Learn++.NSE, we used the same parameters as those used in the artificial datasets. Other types of online regression algorithms include FIMT-DD [182], ORTO [98], ARF-Reg [79], and AMRules [6] which are implemented by MOA software [19].

Table 4.1 : Datasets details.

	Name	Instances	Attributes	Time-window size
	Hyperplane (incremental drift)	2000	10	200
Artificial	Friedman (sudden drift)	2000	5	200
	Gradual drift dataset	2000	3	200
	CCPP	9568	5	300
Real-world	Bikes	17389	16	500
	Cpusmall	8192	12	400
	House	20640	8	1000
	Solar	32686	6	1000

Table 4.2 shows the comparison results for every algorithm in terms of RMSE for different datasets. The results demonstrate that our proposed C-SVR obtains the best performance.

Real-world Datasets

In this section, we present the experiment results from six real-world regression datasets. The datasets were selected from different applications with a wide range of data sizes and dimensionality. The details for each dataset are also shown in Table 4.1.

Table 4.2 : Comparison results on synthetic datasets (RMSE)

	Hyperplane	Friedman	Gradual drifting datasets	Average-Rank
$FMITDD$	4.42E-01 (9)	2.63E-01 (8)	9.82E-02 (9)	8.67
$ORTO$	10.27E-01 (11)	8.12E-01 (11)	3.08E-01 (11)	11
$ARF-Reg$	3.88E-01 (8)	4.67E-01 (10)	8.41E-02 (8)	8.67
$AMRules$	4.48E-01 (10)	3.12E-01 (9)	1.10E-01 (10)	9.67
$I-SVR$	2.31E-01 (7)	14.82E-01 (7)	3.74E-02 (7)	7.00
$TW-SVR$	8.01E-02 (6)	11.49E-02 (5)	2.36E-02 (6)	5.67
$VPI-SVR$	7.98E-02 (5)	11.78E-02 (6)	2.35E-02 (5)	5.33
$AON-SVR$	7.49E-02 (2)	11.47E-02 (3)	2.12E-02 (2)	2.33
$AddExp.C-SVR$	7.54E-02 (3)	11.67E-02 (4)	2.26E-02 (4)	3.67
$Learn++.NSE-SVR$	7.61E-02 (4)	11.32-02 (2)	2.25E-02 (3)	3.00
$C-SVR$	7.38E-02 (1)	11.29E-02 (1)	1.99E-02 (1)	1.00

The datasets are described as follows: the CCPP and Bikes datasets are taken from the UCI repository (http://archive. ics.uci.edu/ml/). The CCPP dataset is collected from a combined cycle power plant over six years (2006-2011), and the work in [179] proved that it does not have any drift. The Bikes dataset contains daily counts of rented bicycles from the bicycle rental company $Capital-Bikeshare$ in Washington D.C., with the weather and seasonal information. The Cpusmall dataset is taken from the Delve datasets (http://www.cs.toronto.edu/ ̃delve/data/datasets.html), and the goal is to predict a computer system activity from system performance measures. The House dataset is a collection of all the housing block groups in California from the 1990 Census. A block group on average includes 1425.5 individuals living in a geographically compact area. The Solar dataset is provided by NASA and is taken from the HI-SEAS weather station in the period between Mission IV and Mission V. The data interval is roughly 5 minutes.

Table 4.3 shows the comparison results for every algorithm in terms of RMSE for different real-world datasets. The results demonstrate that although I-SVR obtains the best performance on the CCPP datasets, i.e., there is no clear concept drift in the datasets, the performance of other online regression algorithms with a forgetting mechanism is not much worse than that of I-SVR. In contrast, in the last four datasets, i.e., there may be concept drift in the datasets, online regression algorithms with the forgetting mechanism significantly outperform I-SVR. Therefore, the forgetting mechanism is an important key in solving the online non-stationary regression problem. Next, to compare the results obtained by the online SVR-based algorithms and our proposed C-SVR, we can see that C-SVR achieved a better per-

Table 4.3 : Comparison results on real-world datasets (RMSE)

$FMITDD$	3.57E+00 (9)	2.57E+00 (2)	**1.03E+00 (1)**	5.54E+04 (3)	1.14E+02 (9)	4.8
$ORTO$	4.53E+02 (11)	5.93E+01 (11)	9.31E+01 (11)	9.42E+04 (10)	2.25E+02 (11)	10.8
$ARF - Reg$	3.72E+00 (10)	**7.29E-01 (1)**	1.12E+00 (2)	10.07E+04 (11)	9.47E+01 (4)	5.6
$AMRules$	3.42E+00 (8)	2.92E+00 (3)	1.31E+00 (3)	**4.72E+04** (1)	9.52E+01 (5)	4
$I - SVR$	**3.04E+00 (1)**	6.30E+00 (10)	1.92E+00 (9)	7.19E+04 (9)	1.11E+02 (10)	7.8
$TW - SVR$	3.36E+00 (6)	5.17E+00 (9)	1.97E+00 (10)	6.01E+04 (6)	1.08E+01 (6)	7.4
$VPI - SVR$	3.32E+00 (7)	5.02E+00 (8)	1.83E+00(8)	6.06E+04 (7)	1.04E+01 (7)	7.4
AON-SVR	3.12E+00 (2)	4.29E+00 (6)	1.74E+00 (7)	6.28E+04 (8)	1.18E+02 (8)	6.2
$AddExp.C - SVR$	3.29E+00 (5)	4.30E+00 (7)	1.59E+00 (6)	5.87E+04 (5)	9.46E+01 (3)	5.2
$Learn++.NSE - SVR$	3.24E+00 (4)	3.22E+00 (4)	1.51E+00 (5)	5.82E+04 (4)	9.42E+01 (2)	3.8
$C - SVR$	3.18E+00 (3)	3.65E+00 (5)	1.46E+00 (4)	4.83E+04 (2)	**9.40E+01 (1)**	3

formance than the other online SVR-based algorithms on the last four algorithms, except for Learn++.NSE-SVR. The reason for this may be because the performance of Learn++.NSE-SVR is not impacted by the size of the time window, so Learn++.NSE-SVR achieves a better performance than C-SVR when the size of the time window has not been suitably set. However, for most of the datasets, C-SVR achieves the best performance when the size of the time window has been set to a multiple of 100. Additionally, with an increasing amount of data being assigned to one of the three sets (S_S, E_S, R_S) and no data to discard, the learning speed will become increasingly slower, until it is much slower than that of C-SVR. Next, to compare the results obtained by FIMT-DD, ORTO, ARF-Reg, AMRules, and our proposed C-SVR, we can see that C-SVR can also achieve comparative performance. However, the main advantage of the C-SVR is stability (has minimum average-rank). The reason for this is that the difference between incremental drift, gradual drift and sudden drift was omitted in ORTO, FIMT-DD, ARF-Reg, AMRules, but was considered in our proposed method.

4.3 Topology Learning-based Fuzzy Random Neural Network for Evolving Streaming Data

4.3.1 Preliminary

As a type of evolving fuzzy systems [15], to handle regression, the rules of evolving-fuzzy-neuro (EFN) systems are mainly the first-order TSK type [169]:

$$\begin{aligned} &Rule_i : IF\ \left(x_1\ is\ A_1^i\right)\ \text{AND} \ldots \text{AND}\ \left(x_n\ is\ A_n^i\right) \\ &THEN\ \left(y_i = a_0^i + a_1^i x_1 + \ldots + a_n^i x_n\right) \end{aligned} \tag{4.28}$$

where $Rule_i (i = 1, 2, \ldots, R)$ is the ith TSK rule [138], $x_j (j = 1, 2, \ldots, n)$ is the jth input variable, and A_j^i is the jth antecedent of the ith rule. $y_i (i = 1, 2, \ldots, R)$ is the output variable of the ith rule, and $a_j^i (j = 0, 1, 2, \ldots, n)$ is the jth coefficients in the consequent part of the ith rule.

When a TSK system receives input data X, each $Rule_i$ in the TSK system will be fired with a firing strength μ^i. Correspondingly, the final estimated output of the system $(\hat{y})$ is defined in terms of a weighted average of the output produced by each $Rule_i$:

$$\mu^i = \prod_{j=1}^{n} \mu_j^i \left(x_j\right) \text{ and } \hat{y} = \frac{\sum_{i=1}^{R} \mu^i y_i}{\sum_{i=1}^{R} \mu^i} \tag{4.29}$$

where the firing strength μ^i is the weight of the ith TSK rule, μ^i_j is the membership degree of the jth input variable, and $\hat{y}$ is the overall TSK system output.

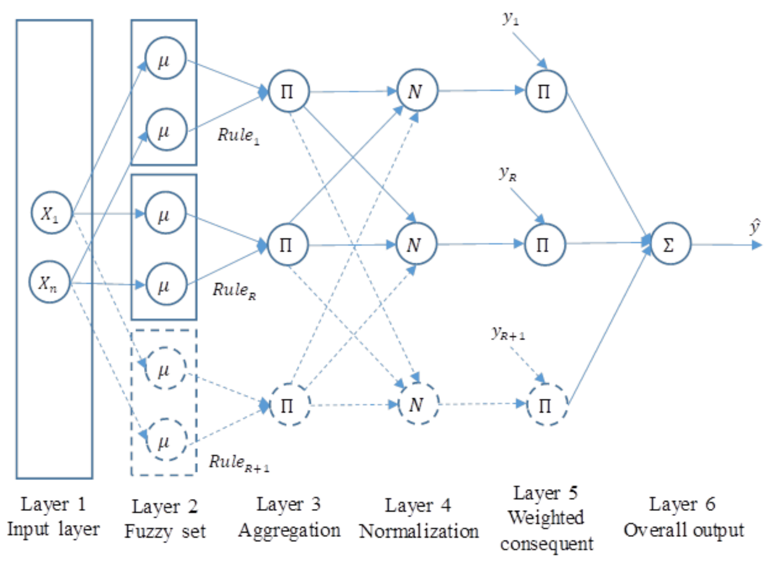


Figure 4.7 : Evolving-fuzzy-neuro systems.

An eFN system is essentially a neural network [169] that has the ability to adapt its structure and parameters to new input data through incremental/online learning. An illustration of an eFN system is shown in Figure 4.7. The neural network learns the TSK rules which are in common. The dotted neurons in the second layer represent new fuzzy neurons that were added through learning. Each layer of the neural network is described as follows:

- Layer 1: the input data $X = (x_1, x_2, \ldots, x_n)$ is received in this layer and passed to the next layer.

- Layer 2: each input variable x_j of the input data X is fuzzified into the membership degree u^i_j in this layer because each neuron in this layer is the jth antecedent of one TSK $Rule_i$ and is attached to a fuzzy membership function. For example, a neuron in this layer can represent a linguistic variable “old” while another can be used to represent "young".

- Layer 3: the firing strength μ_i of a rule is determined in this layer, because the neurons in this layer perform the product operation over the membership degrees μ_j^i which are received from layer 2.
- Layer 4: each neuron in this layer normalizes the firing strength μ_i of each $Rule_i$ received from layer 3.
- Layer 5: each neuron in this layer has two inputs, one is the normalized firing strength μ_i from the corresponding neuron in layer 4, while the second input is output y_i in the consequent parts of a rule. The output of each neuron in this layer is a weighted output.
- Layer 6: in this layer, the weighted outputs from layer 5 are tallied to generate the final estimated output $\hat{y}$ of the eFN system.

When designing an eFN system, a common approach to determine the antecedents of the TSK rule is to use clustering algorithms to obtain fuzzy neurons. Clustering algorithms [53, 83] can not only be applied to input variables, but can also be applied to output variables. To obtain TSK rules, clustering [151] is commonly applied in the data space of input-output variables and the resulting clusters are projected onto the input variables' coordinates to determine the antecedents of the TSK rule. Normally, the number of fuzzy neurons determines the number of rules because each of the fuzzy neurons delineates a fuzzy region in the data space, meaning each of the fuzzy neurons defines a TSK rule. The coefficients in the consequent part of the TSK rule are learned separately using linear-optimized methods such as RLS [52] and online gradient descent.

4.3.2 TLFRNN: Learning Structure

This section describes how to learn the structure of TLFRNN. Besides, the mechanism of TLFRNN handling concept drift is detailed in this section.

Learning Topology Network

In evolving-fuzzy-neuro systems, uncertain data need to be mapped onto the universe of discourse X in the fuzzy set according to a neuron. The X is defined as $\mu\text{A} : X \rightarrow [0, 1]$, where each element of X is mapped to a value between 0 and 1. This value, called the membership value, quantifies the grade of the membership of the element in X to the fuzzy set A. This membership value μ is later brought into the inference process and significantly affects the accuracy of the prediction. In data streams, clustering is implemented in a random environment, so an algorithm returns different results for one dataset when the data is presented in a different sequence. The reason is current online clustering methods which obtain neurons are sensitive to the data sequence. Even in some methods, inappropriate selection of initial clusters, the clustering results will be poor regardless of how the algorithm is adjusted later, and thereby impacting the accuracy of the fuzzy set and membership value. Unfortunately, in streaming data, we are not able to determine which samples can be the initial clusters since samples are randomly selected from streaming data.

To overcome the randomness problem, we integrate the idea of fuzzy random variables [163, 163] into the TLFRNN. Since the membership value is dependent on the initial cluster and the initial cluster is random if the training data are streaming data, our proposed TLFRNN will build multiple fuzzy sets, and the input data will be transferred into fuzzy random variables. For example, the inputs of one sample can be transferred into Table 4.4, where μ is the membership value, and P is the probability of this membership value.

Since the input data in our proposed TLFRNN will be converted into fuzzy

Table 4.4 : Fuzzy random variables

Inputs		
X		
$\mu_1^{P_1}, \mu_2^{P_1}, \mu_3^{P_1}$	$\mu_1^{P_2}, \mu_2^{P_2}, \mu_3^{P_2}$	$\mu_1^{P_3}, \mu_2^{P_3}, \mu_3^{P_3}$
P_1	P_2	P_3

random variables, TLFRNN has an extra layer called the randomness layer compared with the traditional structure of evolving-fuzzy-neuro systems (see Figure 4.7). Assume there are two different fuzzy sets in TLFRNN, the relation between the randomness layer and the fuzzy set layer in TLFRNN are shown in Figure 4.8.

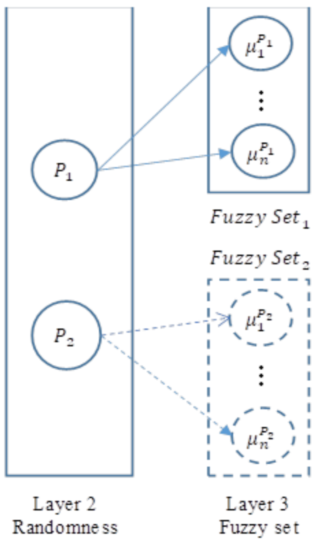


Figure 4.8 : The layers of randomness and fuzzy set in TLFRNN.

From Figure 4.8, we can see a fuzzy set connects to a neuron in the randomness layer. When a sample $Z(t) = (X(t), y(t))$, where $X(t) \in R^d$, and $y \in R^1$, arrives, according to the neurons in the randomness layer, we first get probability P_1 and probability P_2 respectively. Then, according to the neurons in the fuzzy set layer, we can get two fuzzy memberships of this sample, first fuzzy membership with probability P_1, and the second fuzzy membership with probability P_2.

Following the idea of learning neurons in the randomness and fuzzy set layer, we propose an online clustering algorithm which inherits the idea of our proposed self-organized topology learning method [55]. Hence, in our proposed clustering algorithm, two parameters need to be set manually: 1) λ is used to define the frequency of neuron removal. A relatively large λ value contributes to the deletion of fewer neurons. This implies that more neuros will remain. 2) $age_{\max}$ is defined as the lifetime of each edge between neurons. A relatively large $age_{\max}$ value will result in a neuron having more neighbors. However, noise becomes hard to delete.

Our proposed Gm-SOINN (see Section 3.2) can online learn a topology network. Based on the topology network, the learning process is robust to noise and is a better fit to the real data distribution. However, GM-SOINN was proposed for unsupervised task, hence we simplified the Gm-SOINN to a regression method.

Assume we need to learn the structure of TLFRNN as shown in Figure 4.8, the neurons in the fuzzy set layer will sustain two distinct topology networks, and each topology network needs to connect to a neuron in the randomness layer (see Figure 4.9).

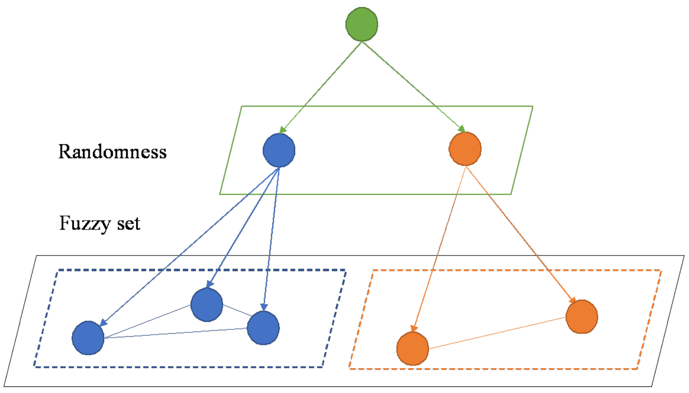


Figure 4.9 : Topology networks.

Hence, assume streaming data $\{Z\}$ with samples $\{Z(1),\dots,Z(t)\} \in R^{d+1}$, the process of learning neurons in the randomness layer and the fuzzy set layer is as follows:

In TLFRNN, each neuron in the randomness layer has a probability $P_i(X(t))$ and is calculated by

$$P_i\left(X\left(t\right)\right) = \sum_{i=1}^{K} \omega_i\left(t\right) f\left(X\left(t\right), W_i\left(t\right), \Sigma_i\left(t\right)\right) \tag{4.30}$$

where K is the number of neurons in the fuzzy set layer that connect this neuro, $\omega_i(t)$ is the weight of the ith component, such that $\sum_{i=1}^{K}\omega_i(t) = 1$, and $f(X(t), W_i(t), \sum_i(t))$ is the Gaussian density distribution with mean $W_i(t)$ and covariance matrix $\sum_i(t)$. In this case, $\sum_i(t)$ is a diagonal matrix for d dimensional, as it is shown in:

$$\Sigma_i\left(t\right) = diag\left(\sigma_{i,1}^2, \sigma_{i,2}^2, \dots, \sigma_{i,d}^2\right) \tag{4.31}$$

However, to reduce the interference of the parameter values of a neuron over the parameter values of other neurons. For this, we propose to normalize the means after multiplying them by the covariance matrix. Let $\sum_i^{-1}(t)$ be a positive semi-definite matrix and $\sum_i^{-1}(t) = AA^T$, furthermore, in our case $A = A^T$. Then, we can write $f(X(t), W_i(t), \sum_i(t))$ as:

$$f\left(X\left(t\right), W_i\left(t\right), \Sigma_i\left(t\right)\right) = \frac{1}{\sqrt{2\pi}\sigma_i^2\left(t\right)} \times \exp\left(\frac{X^{*^T}\left(t\right) W_i^*\left(t\right) - 1}{2\sigma_i^2\left(t\right)}\right) \tag{4.32}$$

where $X_i^*(t) = AX(t), W_i^*(t) = AW_i(t)$.c

Assume there are two neurons in the randomness layer, in the initial step, the value W_i of each neuron in the randomness layer is set to sample $Z(1)$, and sample $Z(2)$, respectively. If sample $Z(3)$ is inserted into the topology network that connects the first neuron in the randomness layer, the $W_i(t)$ is updated according to:

$$W_{1,j}^*\left(t+1\right) = W_{1,j}^*\left(t\right) + \left(K\omega_1\right)^{-1} p\left(1|X\left(t\right)\right)\left(X_j^*\left(t\right) - W_{1,j}^*\left(t\right)\right) \tag{4.33}$$

$$\sigma_{1,j}^2\left(t+1\right) = \sigma_{1,j}^2\left(t\right) + \left(K\omega_1\right)^{-1} p\left(1|X\left(t\right)\right)\left[\left(X_j^*\left(t\right) - W_{1,j}^*\left(t\right)\right)^2 - \sigma_{1,j}^2\left(t\right)\right] \tag{4.34}$$

$$\omega_1(t+1) = \omega_1(t) + K^{-1}\left(p\left(1|X(t)\right) - \omega_1(t)\right) \tag{4.35}$$

where:

$$p\left(i|X(t)\right) = \frac{\omega_i f\left(X(t), W_i(t), \Sigma_i(t)\right)}{P\left(X(t)\right)} \tag{4.36}$$

As for learning the fuzzy set layer, we use all the neurons $F_i(t)$ in the fuzzy set layer that are connected to the first neuron $W_1(t)$ in the randomness layer as an example.

Algorithm 4.2 lists the details of the learning topology network. From Algorithm 4.2, we can see the first step is to determine whether or not to insert a new sample $Z(t)$ in the fuzzy set layer, i.e., as a new neuron $F_i(t)$. Assume we already have N neurons $\{F_1(t), \ldots, F_N(t)\}$ existing in topology network at timestamp t. When a new sample $Z(t)$ is given, it finds the nearest neuron (winner) $F_1'(t)$ and the second-nearest neuron $F_2'(t)$ (s-winner) of the sample $Z(t)$ by the fuzzy membership $\mu_i(t)$. If the fuzzy membership $\mu_i(t)$ between the new sample $Z(t)$ and the winner $F_1'(t)$ or s-winner $F_2'(t)$ is more than the threshold T_i, and if no edge connects $F_1'(t)$ and $F_2'(t)$, $F_1'(t)$ and $F_2'(t)$ should be connected with an edge. The "age" of the edge is set as "0", and the age of all the edges linked to the $F_1'(t)$ is subsequently increased by "1". Then an edge is deleted if its age exceeds $age_{\max}$. Otherwise, the new sample $Z(t)$ is inserted into the topology network as new neuron $F_{N+1}(t)$. When the number of input samples equals λ, the deletion of neurons will be triggered.

According to the idea of Gm-SOINN, the winner $F_1'(t)$ is updated as follows:

$$F_1'(t+1) = F_1'(t) + \gamma_i(t+1)\,\Delta F_1'(t+1) \tag{4.37}$$

where the parameter $\gamma_i(t), (0 \leq \gamma_v \leq 1)$ is the reciprocal of the winning times W_i of winner $F_1'(t+1)$.

As for the other neuros $F_i(t+1)$ that connect to the winner $F_1'(t+1)$, these are

Algorithm 4.2. Learning topology network

```
Input: Sequence $\{Z\}$, $age_{\max}$, $\lambda$
Output: The topology network
1: Initialize set $AN$ with the first 2 samples drawn from $\{Z\}$
2: while $\{Z\}$ is not empty do
3:   Find winner $F_1'$ and second winner $F_2'$ from neuro set $AN$, as
     ${F_1}' = F_{\rm i}$ with $\arg\max_{i \in AN} \mu_i$, ${F_2}' = F_{\rm i}$ with $\arg\max_{i \in AN \backslash \{{F_1}'\}} \mu_i$
4:     if $F_1' < T_1$ and $F_1' < T_2$ then
5:         Add a neuro $F_i$ with weight $Z(t)$ to $AN$, and set active times $C_i = 0$.
6:   else
7:     Increase the active times as $C_{F_1'}(t) = C_{F_1'}(t-1) + 1$. Then update neuros
       in the fuzzy set layer as
       $F_1'(t+1) = F_1'(t) + \gamma_i(t+1)\Delta F_1'(t+1)$
       $F_i(t+1) = F_i(t) + \lambda \cdot \gamma_i(t+1) \cdot \Delta F_i(t+1)$
       for all neuros $F_i$ that connect to $F_1'$
8:     Increase the active time $C_i$ of all neuros linked with $F_1'$ by 1
9:     Increase the age of all edges linked with $F_1'$ by 1.
10:      for all edge $age_{i,j} > age_{\max}$ do
11:      Remove the edge connecting $F_i$ and $F_j$.
12:    end for
13:  end if
14:  if number of input samples divides $\lambda$ then
15:    Remove neuros $F_i$ that only one neuro connects to it.
16:    Remove neuros $F_i$ that the active time $C_{F_i}$ does not increase.
17:  end if
18: end while
```

updated as follows:

$$F_i\,(t+1) = F_i\,(t) + \lambda \cdot \gamma_i\,(t+1) \cdot \Delta F_i\,(t+1) \tag{4.38}$$

where λ is the learning rate which was chosen to decrease the number of samples by $\lambda = 1/k_i$, with k_i the number of samples which connect to the winner. Based on Eq. (4.49) and Eq. (4.50), all neurons F_i can be updated in an online manner.

Due to introducing a randomness layer, we can learn multiple fuzzy sets, and one fuzzy set will be assigned a probability. So our proposed TLFRNN is not sensitive to the randomness of streaming data. Besides, our topology network learning method is robust to λ and $age_{\max}$ parameters. Hence, this reduces the need for accuracy of user-set parameters.

Concept Drift Learning

Detecting and adapting concept drift is an important part of EFSs. Most current EFSs detect concept drift by detecting a change in the error rate [10, 157], since directly detecting a change in the data distribution $D(X, y)$ is a time-consuming task. However, in our proposed TLFRNN, the learned neurons will build a topology network that is able to represent the data distribution $D_{0,t}(X, y)$ accurately. Hence, by utilizing the topology network, we can easily handle the concept drift problem. In TLFRNN, adapting to concept drift does not mean the drift has to be detected. When a new sample is given, TLFRNN updates itself from a global and local viewpoint. From a global viewpoint, there is a variable C in our proposed Algorithm 1 that can be used to record the number of neurons to be activated. If a neuron is active, this means it is selected as the winner or s-winner, or it can connect to the winner by edges in this learning iteration. Hence, variable C of the neurons in each topology network does not increase in this iteration. We label them as outdated neurons and delete them from each topology network. From a local viewpoint (i.e., at each topology network), as the position of neurons will be updated according to

Algorithm 4.2, the topology network will finally adapt to the new data distribution. Furthermore, the neurons in the randomness layer are also updated according to the updating of the topology network to which it connects.

4.3.3 TLFRNN: Determining Parameters

In this section, we describe how to determine the parameters of our proposed TLFRNN. In addition, we also demonstrate how to use the maximum likelihood process to optimize the parameters of TLFRNN.

Fuzzy Set And Fuzzy Rules

For streaming data regression, the inference of current EFSs is commonly based on first-order Takagi-Sugeno fuzzy rules [169], that is, the consequent $y_i(t)$ of each fuzzy rule is modeled by a linear function (see Eq. (4.28)). Hence, determining fuzzy rules need to split data spaces of input-output variables to obtain the subspaces that can approximate to a TSK rule [200, 188]. However, in practical applications, the prediction problem in each subspace is still a nonlinear model and cannot approximate to a TSK rule. In addition, due to the first-order TSK rule is a linear function, the prediction model of current EFSs is actually a combination of multiple linear models. As a result, current EFSs are difficult to model the nonlinear relationship between input and output. Furthermore, to obtain the consequent $y_i(t)$, several parameters such as the parameters a_j^i in Eq. (4.28) need to be learned.

In our proposed TLFRNN, each neuron $F_i(t)$ in the fuzzy set layer is a fuzzy rule and has a consequent $y_i(t)$, and different with the first-order Takagi-Sugeno fuzzy rule, the consequent $y_i(t)$ of each fuzzy rule is calculated by:

$$y_i\left(t\right) = f\left(X\left(t\right) \mid \left\{F_i\left(t\right)\right\}\right) = \frac{K_{h(t)}\left(X\left(t\right) - F_i^X\left(t\right)\right) F_i^y\left(t\right)}{\sum_{j=1}^{n} K_{h(t)}\left(X\left(t\right) - F_i^X\left(t\right)\right)} \tag{4.39}$$

and the $K_{h(t)}\left(X(t)-F_i^X(t)\right)$ is defined as:

$$\begin{aligned}&K_{h(t)}\left(X\left(t\right)-F_i^X\left(t\right)\right)\\=&\frac{1}{\sqrt{\left(2\pi\right)^d\left|H\left(t\right)\right|}}\cdot\exp\left(-\frac{1}{2}\left(X\left(t\right)-F_i^X\left(t\right)\right)^TH(t)^{-1}\left(X\left(t\right)-F_i^X\left(t\right)\right)\right)\end{aligned}\tag{4.40}$$

where d is the dimension of the input data, $H(t)$ is a diagonal matrix with squares of smooth parameter $h_k(t)$, and $|H(t)|=\Pi_k h_k(t)$.

Due to the consequent $y_i(t)$ of each fuzzy rule is represented by a kernel function, so the data spaces of input-output variables do not need to be split to obtain the subspaces that can approximate to a linear model. Hence, based on this type of fuzzy rules, the inference of an input data $X(t)$ is as follows:

- **Step 1.** We first calculate the probability P_i that the input data $X(t)$ attaches each neuron in the randomness layer by Eq. (4.31).
- **Step 2.** We then obtain the fuzzy membership μ_i where by input data $X(t)$ attaches to each neuro F_i in the fuzzy set layer and normalizes the fuzzy membership μ_i to obtain the weights:

$$\theta_i=\sum_i\frac{\mu_i}{\sum_j\mu_j}\tag{4.41}$$

- **Step 3.** Next, given the input data $X(t)$, we further obtain the consequent $y_i(t)$ of each neuro F_i using Eq. (4.39).
- **Step 4.** Like other fuzzy systems, we also weight the $y_i(t)$ to obtain a weighted value $y^*(t)$ by:

$$y^*\left(t\right)=\sum_i\theta_iy_i\left(t\right)\tag{4.42}$$

- **Step 5.** Finally, we obtain the prediction value $\hat{y}(t)$:

$$\hat{y}(t) = \sum_i \frac{P_i(t)\, y_i^*(t)}{\sum_j P_j(t)} \tag{4.43}$$

Therefore, different from current EFSs, in the inference of our proposed TLFRNN, the random and fuzzy information of samples are used at the same time and the consequent $y_i(t)$ of each fuzzy rule is modeled as a nonlinear function.

Selecting Parameter

According to Eq. (4.40), only the smooth parameter $H(t)$ influences forecasting. Some methods, such as RLS and online gradient descent [136], are used to select the smooth parameter automatically. However, these methods require that the neuros F_i in the fuzzy set layer cannot be changed in the next learning iteration. However, in our proposed TLFRNN, the neuros F_i may be $F_i(t) \neq F_i(t+1)$. Hence, we develop a new method for automatically setting the appropriate smooth parameter.

In our proposed method, at first, the smooth parameter $H(t)$ is redefined as follows:

$$H(t) = diag\left(s(t)\,\delta_1^2(t), \ldots, s(t)\,\delta_k^2(t), \ldots s(t)\,\delta_d^2(t)\right) \tag{4.44}$$

so Eq. (4.40) can also be rewritten as:

$$\begin{aligned} &K_{h(t)}\left(X(t) - F_i(t)\right) \\ =&\frac{1}{\sqrt{(2\pi)^d (s(t))^d \prod_k \delta_k^2(t)}} \cdot \exp\left(-\frac{1}{2s(t)} (X(t) - F_i(t))^T \Delta(t)\,(X(t) - F_i(t))\right) \end{aligned} \tag{4.45}$$

where $\Delta(t) = \mathrm{diag}\left(\delta_1^{-2}(t), \ldots, \delta_k^{-2}(t), \ldots, \delta_d^{-2}(t)\right)$.

However, as there is a neuron $F_i(t)$ in the fuzzy set layer may change as new input data $X(t)$ arrives, $s(t)$ needs to be recalculated when new $X(t)$ arrives. Given input data $X(t)$, we denote the kth dimension of neuron F_i as F_i^k and we can obtain the possibility $\hat{P}(X(t))$ of input data $X(t)$ under moment $s(t)$ as:

$$\begin{aligned} \widehat{P}(X(t)|s^*(t)) &= \sum_i K_{h(t)}(X(t) - F_i(t)|s^*(t))\mu_i(t) \\ &= \sum_i \lambda_i(t)(s^*(t))^d \exp\left(-0.5(s^*(t))^2\chi_i(t)\right)\mu_i(t) \end{aligned} \tag{4.46}$$

where

$$\lambda_i(t) = \frac{1}{\sqrt{(2\pi)^d \prod_k \delta_k^2(t)}} \tag{4.47}$$

$$s^*(t) = s(t)^{-0.5} \tag{4.48}$$

$$\chi_i(t) = \sum_k \left(X_k(t) - F_i^k(t)\right)^2 \delta_k^{-2}(t) \tag{4.49}$$

We assume the $\hat{P}(X(t))$ of this moment input data $X(t)$ is the maximum under this moment smooth parameter$s^*(t)$, i.e. $\widehat{P}_{\max}(X(t) \mid s^*(t))$. Thus, we can obtain $\widehat{P}_{\max}(X(t) \mid s^*(t))$ by maximizing the likelihood function $L(s^*(t) \mid X(t))$. Then, the likelihood function $L(s^*(t) \mid X(t))$ can be transformed into

$$L(s^*(t)|X(t)) = \sum_i \mu_i(t) L(s^*(t)|X(t), F_i(t)) \tag{4.50}$$

Hence, maximizing $L(s^*(t) \mid X(t))$ equates to maximizing $L(s^*(t) \mid X(t), F_i(t))$ for each neuron $F_i(t)$. Next, we transform the likelihood function $L(s^*(t) \mid X(t), F_i(t))$ into the log-likelihood function, so

$$L(s^*(t)|X(t)) = \sum_i \mu_i(t)\left(\ln\lambda_i(t) + d\ln s^*(t) - 0.5(s^*(t))^2\chi_i(t)\right) \tag{4.51}$$

Let $\frac{\partial L(s^*(t)|X(t))}{\partial s^*(t)} = 0$, we have

$$s^*(t) = \sqrt{\frac{d\sum_i \mu_i(t)}{\sum_i \mu_i(t)\chi_i(t)}} \tag{4.52}$$

then according to $\sum_i \mu_i(t) = 1$, we finally obtain

$$s^*(t) = \sqrt{\frac{d}{\sum_i \mu_i(t)\chi_i(t)}} \tag{4.53}$$

where $\sum_i \mu_i(t)\chi_i(t) \propto \sum_i \mu_i(t) \left\|X(t) - F_i(t)\right\|^2$.

According to the learning objection of the topology network, the number of neurons $F_i(t)$ in the fuzzy set layer increases gradually with the continuous arrival of the streaming data, and $\sum_i \mu_i(t) \left\|X(t) - F_i(t)\right\|^2$ decreases. Then due to $\sum_i \mu_i(t)\chi_i(t) \propto \sum_i \mu_i(t) \left\|X(t) - F_i(t)\right\|^2$, the decreasing of $\sum_i \mu_i(t)\|X(t) - F_i(t)\|^2$ will result in $s^*(t)$ decreasing. Next, according to Eq. (4.43) and Eq. (4.47), we can conclude that the smooth parameter $H(t)$ decreases as the number of neurons F_i increases in our proposed method.

4.3.4 Experiments

In this section, we illustrate the advantages of our proposed TLFRNN by comparing its algorithm with other evolving fuzzy systems. As TLFRNN is an evolving-fuzzy-neuro system, we first validate our proposed algorithm for learning neural networks. Then, we validate the regression performance of TLFRNN. The evaluations involve different scenarios using both artificial and real-world datasets. All experiments are conducted in Python 3.5 version on Windows 7 running on a PC with a system configuration Intel Core i5 processor (2.40 GHz) with 8-GB RAM.

In the experiment, we employed a prequential strategy which was introduced in [71]. In this strategy, a sample is first evaluated by a learning model and is then used to update the learning model. By considering all the metrics, we calculate their mean value for all data streams. In addition, as suggested by [150], a windowed measurement method is also considered, and a non-overlapping sliding window of size 200 is also used. Aiming at reducing the effects of randomness in our evaluation, all multi-output regression methods are performed at least thirty times for each dataset, and the performance is taken as the average between the repetitions.

Learning Topology Network

To validate our proposed algorithm for learning topology network (for convenience, we call it TN in the next sections), we compare TN to fuzzy c-means (FCM) [35] and SOINN [67] - selecting FCM to illustrate the advantage of topology learning and selecting SOINN to illustrate the advantage of fuzzy learning. The specific comparison scheme is: we firstly use FCM, SOINN, and TN to separately obtain the neural networks based on the following dataset:

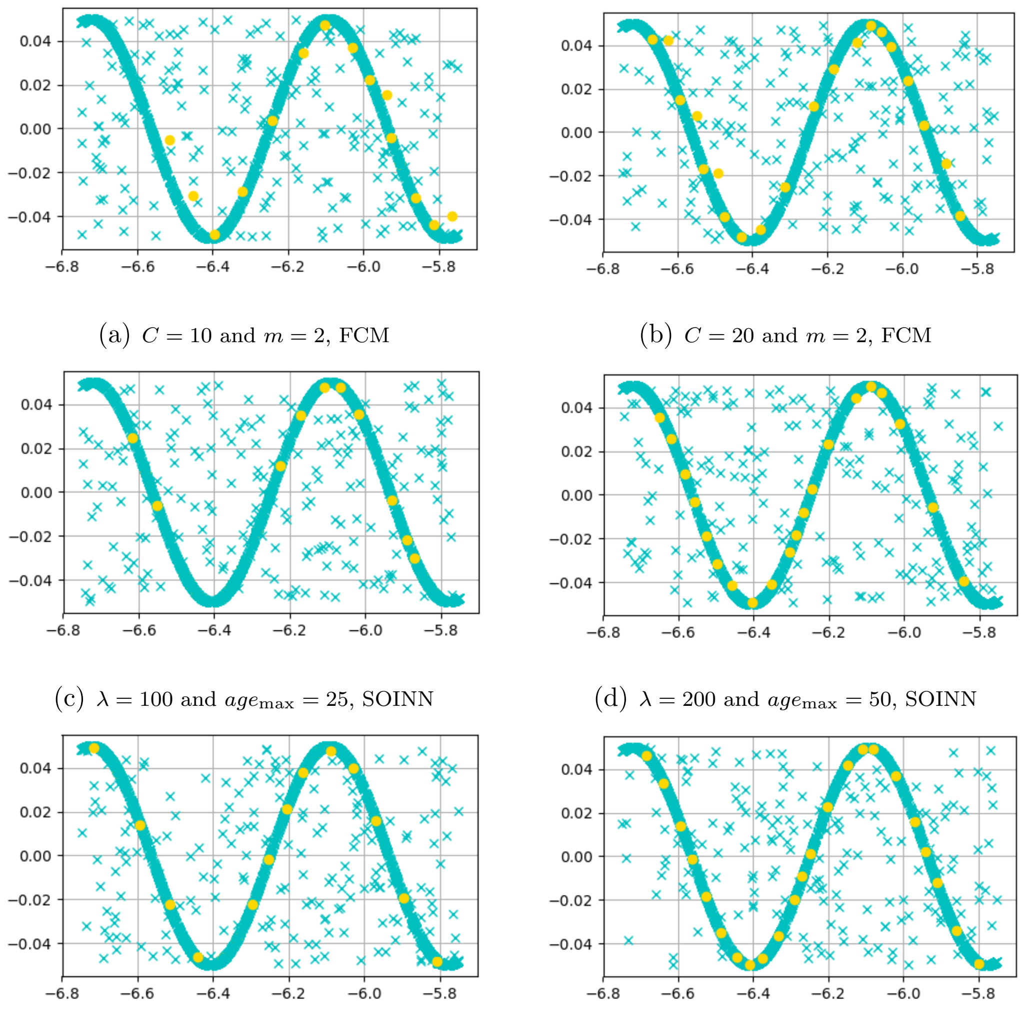
$$\begin{cases} y(t) = \sin(o(t))/20 \\ X(t) = o(t)/10 - 6.25 \end{cases} \tag{4.54}$$

where $o(t)$ is randomly sampled from (0,5). To illustrate that our proposed method is also robust to noise, we add 10% noise to the dataset and the noise is distributed over the whole.

We then make a prediction of the given dataset based on the neural network (represented by neurons) obtained by each algorithm. By comparing the prediction results in terms of the average root mean square error (ARMSE), we can validate the performance of each algorithm.

Next, we discuss the parameters of each algorithm. When using FCM to obtain the neurons, it is necessary to select two parameters: 1) C is used to define the number of neurons; 2) m is the fuzzifier m and determines the level of cluster fuzziness. A large m results in smaller membership values and, hence, fuzzier clusters. In the limit $m = 1$, the memberships converge to 0 or 1, which implies a crisp partitioning. In the absence of experimentation or domain knowledge, m is commonly set to 2. TN does not need to define the number of neurons, but it needs to set λ and $age_{\max}$. λ is used to define the frequency of neuron removal. $age_{\max}$ is defined as the lifetime of each edge. SOINN also does not need to define the number of neurons, but it needs more parameters. Except for the need to manually set λ and $age_{\max}$, C_1 and

C_2 also need to be set manually. The definition of λ and $age_{\max}$ is the same as in our proposed algorithm. C_1 and C_2 are defined to control deletion behavior. A relatively large C_1 or C_2 value will contribute to higher noise tolerance; however, more useful neurons will be deleted at the same time.


(a) $C = 10$ and $m = 2$, FCM

(b) $C = 20$ and $m = 2$, FCM

(c) $\lambda = 100$ and $age_{\max} = 25$, SOINN

(d) $\lambda = 200$ and $age_{\max} = 50$, SOINN

(e) $\lambda = 100$ and $age_{\max} = 25$, TL

(f) $\lambda = 200$ and $age_{\max} = 50$, TL

Figure 4.10 : Neurons obtained by each algorithm.

Figure 4.10 shows the topology network obtained by each algorithm using different parameters. Then we set the smooth parameter to be equal to 0.5 to make a prediction. Table 4.5 lists the prediction results. From the first line of Table 4.5, we can see that FCM achieves a good prediction result when the number of neurons is

Table 4.5 : Different smooth parameter settings

Clustering	Parameters	Neurons	ARMSE
FCM	$C = 10$ and $m = 2$,	10	0.004071
	$C = 20$ and $m = 2$	20	0.03652
SOINN	$\lambda = 100$ and $age_{\max} = 25$	13	0.003823
	$\lambda = 200$ and $age_{\max} = 50$	19	0.003712
TN	$\lambda = 100$ and $age_{\max} = 25$	13	0.003623
	$\lambda = 200$ and $age_{\max} = 50$	23	**0.003492**

set to 20. However, when the number of neurons is set to 10, the prediction result declines, as shown in the second line of Table 4.5. The reason is shown in Figure 4.10(a) and Figure 4.10(b). From Figure 4.10(b), we can see the data distribution is well represented by neurons when their number is set to 20; but from Figure 4.10(a), the data distribution cannot be well represented by neurons when the number is set to 10. Therefore, if we want to use FCM to obtain the neurons of an EFS with good performance, we must set C with an appropriate value. However, under evolving streaming data, it is very difficult to decide which value of neurons is enough to represent the data distribution. In contrast, SOINN does not need a predefined number of neurons. From Figure 4.10(c) and Figure 4.10(d), we can see SOINN automatically learns the neurons by setting the value of λ and $age_{\max}$, and, as the value of λ and $age_{\max}$ becomes smaller, the number of obtained neurons decreases. The prediction result, shown in the third line of Table 4.5, indicates that SOINN can learn neurons with a good prediction ability when $\lambda = 200$ and $age_{\max} = 50$. However, the next line of the table shows that the EFS obtained by SOINN has a poor prediction performance when $\lambda = 100$ and $age_{\max} = 25$. The reason is shown in Figure 4.10(c) and Figure 4.10(d). From Figure 4.10(a), we can see the data distribution is well represented by neurons. However, because the λ and $age_{\max}$ are set with smaller values, some neurons that represent the data distribution of various areas are deleted, resulting in the data distribution not being well represented by neurons. Similar to SOINN, TN also does not need predefined numbers of neurons. From 4.10(e) and 4.10(f), we can see the neurons decrease with the value of λ and, as $age_{\max}$ becomes smaller, the number of obtained neurons decreases. However, the last two lines of Table 4.5 show the EFS obtained by TN has a good prediction performance. The reason is shown in Figure 4.10(e) and Figure 4.10(f). From these, we can see the data distribution is well represented by neurons, when $\lambda = 200$ and $age_{\max} = 50$.

Therefore, the result proves that the robustness of these two parameters is improved using our proposed algorithm. Figure 4.10(a) and Figure 4.10(b) show that some neurons are not located in the true data distribution because of noisy neurons. However, this phenomenon does not exist in Figure 4.10(c), 4.19(d), 4.10(e), and 4.10(f), demonstrating that the neurons obtained by SOINN and TN are more robust to noisy neurons than FCM due to the integration of topology learning.

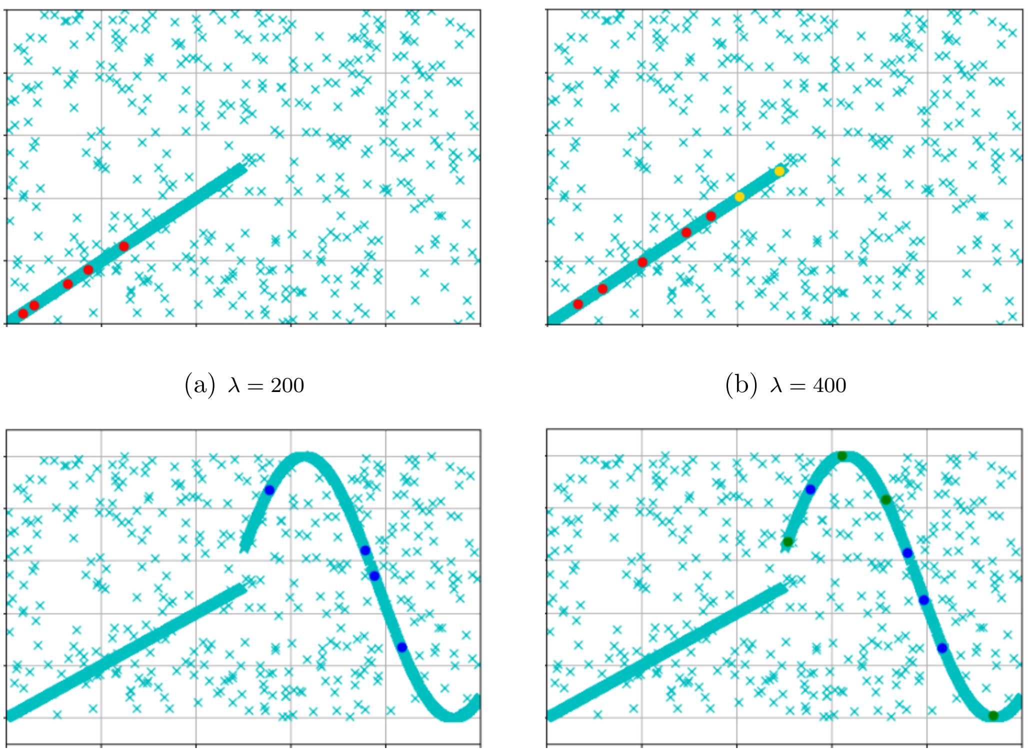
Concept Drift

Another advantage of TLFRNN is that concept drift can be solved by our proposed TN algorithm. Hence, to illustrate the advantage of this, we use the following datasets:

$$\begin{cases} y(t) = x(t) & t \text{ in } [0 - 400] \\ y(t) = 0.5\cos(10x(t)) + 0.5 & t \text{ in } (400 - 800] \end{cases} \tag{4.55}$$

where $x(t) \sim U(0, 0.5)$. The artificial dataset was generated by two models. As with the last experiment, we also add 10% noise to the dataset, and the noise is distributed over the whole dataset. Figure 4.11 shows the neurons in a different time-window. In this experiment, the length of the time-window is 200, i.e. $\lambda = 200$.

The parameter $age_{\max} = 50$.



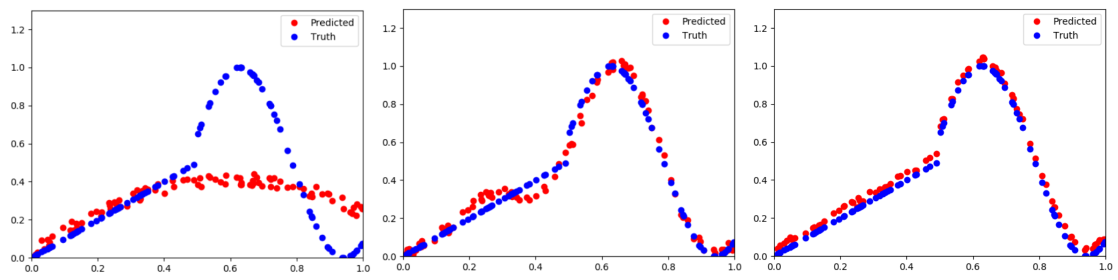
(a) $\lambda = 200$ (b) $\lambda = 400$

(c) $\lambda = 600$ (d) $\lambda = 800$

Figure 4.11 : Neurons obtained by TN in nonstationary environment.

From Figure 4.11, we can see the neurons obtained in the first time-window were all saved and updated after finishing the process of the second time-window: concept drift does not occur. However, when the learning process of the third time-window finishes, the neurons obtained in the first and second time-window are deleted. The deletion neurons indicates concept drift was detected by our method, and it adapts to the new data distribution by deleting the outdated neurons. In the last time-window, because the data distribution matches that in the third time-window, the neurons obtained in the later data were also saved and updated.

To further illustrate the advantages, we compare TN with FCM and SOINN. As in the last experiment, we use FCM, SOINN and TN to obtain the neurons based on the dataset Eq. (4.55), transform these neurons into EFS by setting the smooth parameter to 0.2, and finally compare the prediction results of the EFS that have been learned through the different algorithms. As for the parameters of FCM and SOINN, we set $C = 20$ and $m = 2$ in FCM. The reason for setting $C = 20$ is that TN obtains about 40 neurons when the learning process finishes. To compare this with TN, the values of $\lambda = 200$ and $age_{\max} = 50$ in SOINN match the values in TN.



(a) FCM (b) SOINN (c) FSOINN

Figure 4.12 : Regression learning results on the artificial dataset (two models).

Figure 4.12 compares the results. From Figure 4.12(a), we can see it is difficult for FCM to learn an EFS with good prediction results. The reason for this is it is difficult for FCM to learn suitable neurons to represent the data distribution from a dataset with concept drift. From 4.12(b), we can see SOINN obtains better results than FCM. The reason for this is SOINN has an incremental learning mechanism, so the neurons learned by SOINN will be updated to fit the new data distribution by sequentially processing the data. However, the outdated neurons cannot be deleted by SOINN, resulting in the prediction results of some neurons containing serious errors. Our proposed TN is more accurate than FCM and SOINN. This is because not only is TN capable of detecting and adapting to concept drift, and it is also more robust to noisy data. In summary, our proposed TN can learn the neurons that represent the data distribution from evolving and noisy streaming data.

TLFRNN: Smooth Parameters

In the aforementioned experiment, we focus on validating the performance of the TN algorithm. We do this by proving the regression ability of TLFRNN. From the introduction, we know that the regression performance of our proposed TLFRNN is affected by the smooth parameter. However, it is difficult to set the smooth parameter manually without prior information of the streaming data. So, we propose a method to select the smooth parameter automatically $H(t)$. To substantiate our proposed TLFRNN, we first evaluate the method of selecting the smooth parameter $H(t)$.

We also test the accuracy of our method with a smooth static parameter. The training sets were generated by function Eq. (4.27), and ARMSE was also used to measure accuracy. As for the parameters of TN, we set $\lambda = 200$, $age_{\max} = 50$ and $\lambda = 100$, $age_{\max} = 25$ respectively. We next used the different parameter settings to obtain neurons 10 times. A comparison of the results is shown in Figure 4.13.

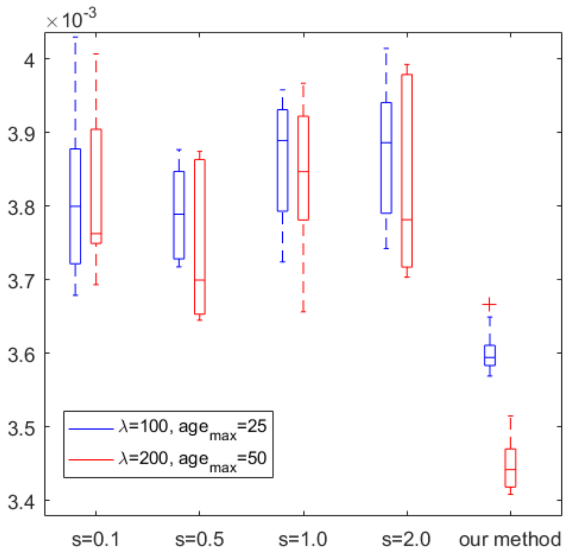


Figure 4.13 : RMSE based on different smooth parameter settings.

Figure 4.13 shows that our method is the most accurate and, if more neurons are generated, our method achieves better accuracy. However, this is not true with the smooth static parameter; the level of accuracy barely changes with an increase in the number of neurons. This also indicates that FSOINN is effective at reaching the optimal state.

TLFRNN: Artificial Dataset

This section presents the experiment results for some of the artificial datasets. The comparison experiments were carried out with non-fuzzy systems, evolving fuzzy systems and our proposed TELFIS. The non-fuzzy systems are FIMTDD [182], AMR [6]. The evolving fuzzy systems include two types: 1). eFRB includes FLEXFIS [138] and Gen-Smart-EFS [137] (an extension of FLEXFIS); 2) eNF includes DENFIS [107], PANFIS [156] and PALM [60]. FIMTDD and AMR are included to demonstrate the need to use fuzzy learning. The reason for including EFSs is to demonstrate the advantages of using topology learning. Table 4.6 shows the parameters of each algorithm. The parameters of each algorithm are selected according to their own criteria within the optimal parameters. If a method applies a window-based updating strategy during the experiments, such as FIMTDD, the size of the time-window is 200. Each algorithm is also evaluated using ARMSE which is calculated using the predicted outputs and the real outputs from the online data.

The first artificial dataset is the drifting hyperplane dataset [62]. This is a well-known drift dataset for evaluating algorithms that deal with concept drift. It contains noise, gradual drifts, and non-recurring drifts. The whole dataset consists of 10 inputs with a uniform distribution over the interval of $[0, 1]$ and 1 output data $y_i \in [0, 1]$; and there are 2000 data ($M = 2000$) in the dataset. The dataset contains four concepts, where each concept holds 500 data. A random variate noise uniformly distributed in the interval of $[-0.1, 0.1]$ is injected into each output y_i (for $i = 1, \ldots, M$). The value of y_i is set to 0 or 1 if its value is less than 0 or greater

Table 4.6 : The parameters of each algorithm.

$Algorithms$	Consequent Parameters Estimation Method	$Number of Use - Defined Parameters$
$FMITDD$	RLS	Learning rate: 0.01, and change probability: 0.05 The minimum number of samples in leaf: 200
AMR	RLS	Error threshold: 50 and minimum number of samples: 200 the magnitude of changes that are allowed: 0.05
$Gen - Smart - EFS$	RLS	A scale factor δ: 1.35 A forgetting factor: 1.0
$FLEXFIS$	RLS	Learning rate: 0.01 Vigilance parameter: 0.3
$DENFIS$	RLS	Learning rate: 0.01 Distance threshold: 0.01
$PANFIS$	RLS	A value was chosen to control whether PANFIS augments its structure or tunes the current structure: 0.1 A value was chosen to yield more rules: 0.01
$PALM$	RLS	Two thresholds of rules merge: 0.5 and 0.5 An adjustment parameter which controls the membership grades :50
$TLFRNN^1$	Maximum likelihood process	Frequency of neuron removal: 100 Lifetime of each age: 25
$TLFRNN^2$	Maximum likelihood process	Frequency of neuron removal: 200 Lifetime of each age: 50

than 1, respectively.

A drifting Friedman's function serves as the second dataset. Friedman's function has linear and nonlinear relations between the input and output variables. The function contains five input variables, and one output variable. The input variables are uniformly distributed over the interval of $[0, 1]$. To create drifting scenarios, one drifting dataset using the original Friedman's function was produced according to [182]. The dataset also has 2000 data and abrupt concept drifts. There are three points of abrupt drift in the training dataset, the first one at $\frac{1}{4}M$ data, the second at $\frac{1}{2}M$ data and the third at $\frac{3}{4}M$ data. During the first stationary period, the Friedman function is modelled. At the first and second points of abrupt drift, the modified functional dependencies are introduced and the regions R_1 and R_2 are expended. The complete description of these drifting data can be found in [68].

The third artificial dataset is the network traffic flow. We use the TCP traffic data that we collected to build an experimental dataset. We collected 12590 traffic data points as a time-series data set. The traffic data are processed to present the amount of traffic in bytes per unit of time and represent a sample. The traffic data is aggregated with time bin 1s, that is the number of packages which arrive within the 1s time interval. To generate noise, we randomly add 1000 samples as the noisy data. If it is a noisy data, the value of traffic data is set to 0.

Table 4.7 : Comparison results on artificial datasets (ARMSE).

	FIMTDD	AMR	Gen-Smart-EFS	FLEXFIS	DENFIS	PANFIS	PALM	TL-EFS[1]	TL-EFS[2]
Hyperplane	4.421	4.478	3.773	3.866	3.764	3.833	3.961	3.669	3.657
Friedman	2.627	3.122	2.875	2.966	2.724	2.759	2.665	2.468	2.453
Network	0.982	1.091	0.832	0.839	0.841	0.867	0.852	0.832	0.820

Table 4.7 shows the results of our comparison based on artificial datasets. By comparing the results obtained from the three datasets, we can see the accuracy

achieved by EFSs is higher than the non-fuzzy systems except on the Friedman dataset. Although the results obtained by EFSs on the Friedman dataset (except for our proposed EFS) are lower than FIMTDD, the performance of EFSs is better than AMR. Therefore, the comparative results prove the advantage of fuzzy learning in processing noisy data. The comparative results also illustrate the ability of the detection mechanism of the current EFSs to adapt well to incremental drift, but not as successfully to abrupt drift. This results in FIMTDD achieving better performance than current EFSs on the Friedman dataset. In contrast, the fact that TLFRNN^1 and TLFRNN^2 obtain the best performance on the three datasets shows that our proposed EFS adapts well to abrupt or incremental drift, thereby proving the effectiveness of the topological network-based detection mechanism of concept drift. Next, comparing all fuzzy systems, we can see our proposed TLFRNN obtains better performance on all datasets whether $\lambda = 200$, $age_{\max} = 50$ or $\lambda = 100$, $age_{\max} = 25$. This phenomenon also proves the neurons obtained by our proposed TN algorithm are more suitable for the distribution of the actual output data. The result obtained by TLFRNN when $\lambda = 200$ and $age_{\max} = 50$ is close to the result obtained by TLFRNN when $\lambda = 100$ and $age_{\max} = 25$. So, this outcome proves the performance of our proposed TLFRNN is not very sensitive to λ and $age_{\max}$.

Regression On Real-world Datasets

In this section, we use real-world datasets to further validate the performance of our algorithm. The datasets were selected from different applications with a wide range of data sizes and dimensionality. First, five datasets were taken from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/ datasets.php), where it was proved in [179] that CCPP (Combined Cycle Power Plant) does not contain any drift and three time-series datasets were taken from different real applications (S&P index, SMEAR, and NN5). The S&P index dataset is a dynamic real-world financial dataset on the Yahoo finance website which contains data on the period from January 3, 1950 to March 12, 2009. This dataset is highly dynamic as evidenced by the two peaks and the valley in the data trend around 2000, 2003, and 2007. The SMEAR dataset comprises a set of 30-min-interval environment observations collected from the SMEAR II station. There are many uncertain data in this dataset. The NN5 datasets comprise 111 time series with 735 observations originating from daily withdrawals at 111 cash machines in England. However, we only combine the first 10 time series of NN5 into D8. Table 4.8 shows the type of datasets and the sample size.

Table 4.8 : Real-world datasets

ID	$Dataset$	SAMPLE SIZE	Attributes
$D1$	$CCPP$	9586	5
$D2$	$Parkinsons$	5875	21
$D3$	CPU	8192	19
$D4$	$California\ Housing$	20640	8
$D5$	$Protein$	45730	9
$D6$	$S\&P\ Index$	14893	1
$D7$	$SMEAR$	140576	43
$D8$	$NN5$	7350	10

The final comparison experiments were carried out based on all the algorithms which are used in the above section. The parameters of each algorithm are the same as in Table 4.6, and the size of the time-window is 200. Table 4.9 shows the results of the real-world experiments. The experiments demonstrate that TLFRNN outperforms the other algorithms on most datasets except on the "Parkinsons" dataset and the SMEAR dataset. Hence, the experiment results prove the superiority of

Table 4.9 : Comparison results on real-world datasets (ARMSE).

	D1	D2	D3	D4	D5	D6	D7	D8
$FMITDD$	0.035	**0.068**	0.039	0.140	0.254	0.064	0.023	0.945
AMR	0.034	0.069	0.039	0.143	0.255	0.089	0.014	0.952
$PALM$	0.053	0.075	0.042	0.139	0.245	0.047	0.033	1.003
$Gen-Smart-EFS$	0.048	0.071	**0.038**	0.133	0.258	0.061	0.018	0.932
$DENFIS$	0.051	0.070	0.042	0.180	0.244	0.055	0.037	0.971
$PANFIS$	0.049	0.073	0.040	0.179	0.253	0.052	0.021	0.959
$PALM$	0.055	0.074	0.041	0.155	0.257	0.042	**0.010**	0.966
$TLFRNN^1$	0.033	0.082	0.039	**0.136**	0.233	**0.036**	0.047	0.933
$TLFRNN^2$	**0.032**	0.083	0.039	0.137	**0.232**	0.037	0.048	**0.927**

using topology learning and introducing randomness for learning evolving fuzzy system. The reason why TLFRNN does not achieve the best performance because the "Parkinsons" dataset is more suitably trained with a linear model. However, its nonlinear learning ability is well demonstrated on the other datasets. As TLFRNN does not achieve the best performance on the SMEAR dataset, this shows that TLFRNN still has limitations on high dimensional datasets and gives a comparable performance to non-fuzzy algorithms (FIMTDD and AMR). The results of TLFRNN^1 and TLFRNN^2 show our proposed algorithm is robust to parameters. In summary, our proposed algorithm is a more accurate alternative to current online regression algorithms for streaming data.

4.4 Summary

It is well-known that the data distribution of streaming data is nonstationary; it can change or evolve. Concept drift refers to the change of data distribution, and a data stream with concept drift is called evolving streaming data. Although some streaming regression algorithms integrate the forgetting mechanism to handle evolving streaming data, the difference between the various types of concept drift is ignored. Hence, we proposed continuous support vector regression (C-SVR) as a method for solving regression problems with evolving streaming data. C-SVR is based on a continuous learning strategy that learns a series of regression models in a series of time windows. However, only one regression model for making a prediction is saved in the computer's memory. A new regression is learned when each new time window arrives, and the last regression model that was learned in the last time window is discarded. A unique difference with our approach is an added similarity term to the Quadratic programming problem that makes the new regression model dependents on the last regression model by transferring some learned knowledge between time windows. The amount of transferred learned knowledge is determined by the extent of the drift as measured by the competency-based method. Like other SVRs designed to work with streaming data, C-SVR solves the QPP in an online manner. Experiments indicate that C-SVR achieves better performance than several online SVR algorithms, especially for datasets with incremental concept drift.

Besides, we proposed a novel evolving-fuzzy-neuro system for a data stream not only with concept drift but with noisy data. In order to improve the performance of evolving-fuzzy-neuro systems for streaming data regression, we propose a novel evolving-fuzzy-neuro system, called the topology learning-based evolving-fuzzy-system (TLFRNN). In TLFRNN, to decrease the sensitivity of prediction accuracy to the system structure, first, we learn multiple fuzzy sets and introduce a randomness layer to assign each fuzzy set a probability. Then, to make fuzzy rules model the nonlinear relationship between inputs and outputs well, we propose a new type of fuzzy rule, and further, a new type of inference is designed. Our designed inference not only fits the nonlinear function well, it also considers the random and fuzzy information simultaneously. Furthermore, the inference of TLFRNN is only influenced by one parameter. This parameter does not need to be learned because it can be decided by a maximum likelihood process. To handle concept drift, a topology network-based mechanism for concept drift is introduced. As the topology network can accurately represent the real data distribution, TLFRNN can adapt to new data distributions easily and rapidly without detecting concept drift. The experiment results show that our proposed algorithm has better performance in relation to streaming data regression.

Chapter 5

Streaming Data Regression Under Multi-Outputs Environments

5.1 Introduction

In previous chapters, we have all made a hypothesis about regression, that is, only a single one output needs to be predicted. However, with the arrival of the big data era, many applications are generating huge amounts of streaming data, and thereby resulting in regression analysis becoming increasingly complex. Accordingly, in more and more practical applications, multiple outputs instead of one output are predicted, i.e., multi-output regression [120, 198, 172]. Compared with single-output regression, multi-output regression is more complicated because these outputs maybe have a structure to represent the relationship between outputs. Traditionally, based on the structure of outputs, multi-output regression can be divided into global and local methods [21]. Global methods use the idea of classical learning algorithms for single output, i.e., to predict the multiple outputs as a whole [8]. In contrast, the idea of local methods is simpler, i.e., to decompose the multi-outputs into multiple single outputs, then use classical learning algorithms for single output to predict each output. However, most existing methods are usually computationally and memory demanding, and are unsuitable for dealing with streaming data.

This research aims to provide a system called MORStreaming to address the multi-output regression problems in streaming data. The system uses an instance-based model [4] for regression, which means MORStreaming can easily adapt to a new concept by simply storing new instances, or throwing old instances away.

However, since the training data is streaming data, MORStreaming needs to consider three main challenges: 1) In contrast to the majority of the instance-based learning systems [168], the instances can only be obtained in an online manner because the streaming data arrive sequentially. 2) The structure of outputs is unknown, since we cannot obtain all training data, and the structure of outputs also can change due to concept drift. 3) How to use the instances and the structure of outputs to make a prediction. In order to solve these challenges, in the learning stage, MORStreaming learns rules to reflect the different structures of outputs, then classifying all obtained instances into different rules according to the input of data. In the prediction stage, finding rules which cover predicted data according to the input of predicted data, then only the instances which under these rules can be used to make a prediction of this predicted data.

5.2 MORStreaming: A Multi-Output Regression System for Streaming Data

5.2.1 Preliminary

Multi-output regression is an important branch of structured output learning. Let $\{(X_1, Y_1), (X_2, Y_2), \ldots, (X_N, Y_N)\}$ be a training dataset where X_i is a d-dimensional vector $[X_{i,1} \ldots X_{i,d}]^T$ describing the inputs of the ith sample of the dataset, and Y_i is an m-dimensional vector $[Y_{i,1} \ldots Y_{i,m}]^T$ of the outputs. The problem of multi-output regression is to learn a model $f(X) \to Y$ that maps the input X to the output Y.

Throughout this section, we briefly introduce several state-of-the-art methods for multi-output regression, categorized as local methods and global methods:

- Local methods are based on the idea of converting the multi-output regression problem into m single-output problems, then learning a model for each output, and finally concatenating all m predictions. For example, Hoerl and

Kennard [93] separate the multivariate output into multiple univariate outputs, and then utilize the ridge regression to make a prediction. Next, inspired by stacked generalization that was originally used to deal with multi-label classification, the multi-output regressor stacking method is proposed [77]. The regressor chains method [172] is another important local methods based on the idea of chaining single-output models. Zhang et al. [196] presented a multi-output support vector regression. This type of regression develops a vector virtualization method to build a multi-output model that takes into account the relations between all the outputs.

- Global methods are mainly based on simultaneously predicting all the outputs using a single model that can capture all internal relations between them. Izenman [100] proposed a reduced-rank regression, which adds a rank of constraint on the estimated outputs. Struyf and Dzeroski [173] integrated a constraint-based system into the process of learning multi-objective regression trees Tuia et al. [177] also developed a multi-output support vector regression method but by extending the single-output SVR to multi-outputs while maintaining the advantages of a compact and sparse solution using a cost function.. Aho et al. [5] presented a novel method for learning ensemble rules for multi-output regression and treating multiple numeric outputs as a whole.

In summary, the above multi-output regression methods are usually memory demanding and computationally, i.e., instances need to be processed multiple times, and hence are not suited for dealing with streaming data. In addition, these methods do not have a mechanism to handle the noisy and evolving characteristics of streaming data.

5.2.2 Multiple-output Regression System

In this section, we will introduce the details of our proposed multiple-output regression system, MORStreaming.

MORStreaming

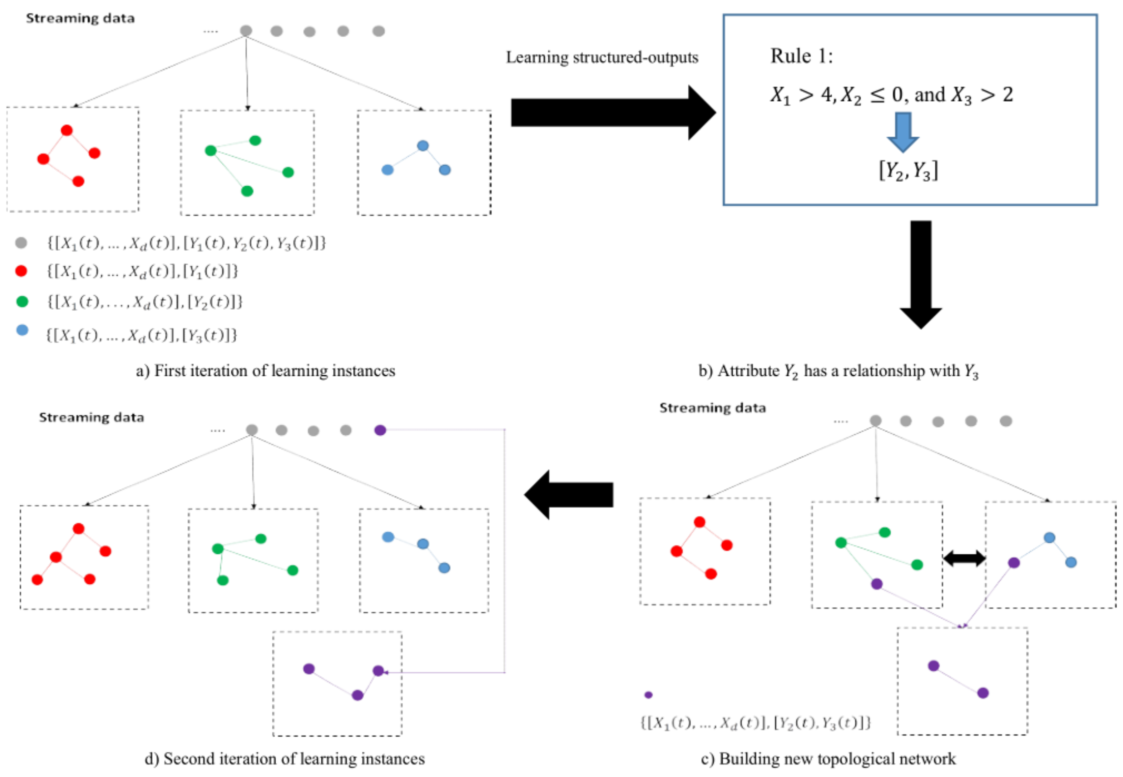

Streaming data
Learning structured-outputs
Rule 1:
$X_1 > 4, X_2 \le 0$, and $X_3 > 2$
$[Y_2, Y_3]$
● $\{[X_1(t), \ldots, X_d(t)], [Y_1(t), Y_2(t), Y_3(t)]\}$
● $\{[X_1(t), \ldots, X_d(t)], [Y_1(t)]\}$
● $\{[X_1(t), \ldots, X_d(t)], [Y_2(t)]\}$
● $\{[X_1(t), \ldots, X_d(t)], [Y_3(t)]\}$
a) First iteration of learning instances
b) Attribute Y_2 has a relationship with Y_3
Streaming data
Streaming data
● $\{[X_1(t), \ldots, X_d(t)], [Y_2(t), Y_3(t)]\}$
d) Second iteration of learning instances
c) Building new topological network

Figure 5.1 : Summary of the MORStreaming process.

The most important component in MORStreaming is to learn instances to create topology networks, where learning instances are constrained by structured-outputs. Assume the training data is streaming data with d-dimensional input attributes and three-dimensional output attributes, $\{[X_1(t), \ldots, X_d(t)], [Y_1(t), Y_2(t), Y_3(t)]\}$. Figure 5.1 illustrates the four-step process of MORStreaming. First, MORStreaming assumes there is no relationship between two arbitrary output attributes, so three topological networks are learnt, based on each output attribute, i.e., the first topology network is learnt by data $\{[X_1(t), \ldots, X_d(t)], [Y_1(t)]\}$, the second topology network is learned by data $\{[X_1(t), \ldots, X_d(t)], [Y_2(t)]\}$ and the third topology network is learnt by data $\{[X_1(t), \ldots, X_d(t)], [Y_3(t)]\}$. After, the first iteration of learning instances is complete, and we learn rules to explore the structure between the output attributes. If a rule is learned to represent the attribute $Y_2(t)$ has a relationship with $Y_3(t)$, then a new topology network is built by combining instances in the second topological network that were covered by R_1, and instances in the third topological network that were covered by R_1. Finally, in the second iteration of learning instances, the new topology network is updated by streaming data that was covered by R_1, otherwise, streaming data is used to update the first, second, and third topology network. Then, a new iteration of learning rules is explored to find new relationships of output attributes. The following sections of this chapter introduce learning in the structured-output and topology network.

Learning Structured-outputs

The most significant difference between single-output and multi-output regression is that multi-output regression needs to consider the structure, or correlation, of outputs. Typically, multi-output predictors can be divided into local and global strategy approaches. However, there are three challenges when using these approaches to handle streaming data: 1). Unlike batch-based methods, the size of streaming data is larger than computer memory. Therefore, the structure of the outputs cannot be obtained or estimated. 2). The structure of outputs is evolving and can change when the distribution of data changes, i.e., concept drift. 3). The structure of outputs is neither local nor global. For example, assuming there are three output attributes $\{Y_1(t), Y_2(t), Y_3(t)\}$ of tth sample $\{X(t), Y(t)\}$, where the attribute Y_1 has a relationship with Y_2 attribute, but the attribute Y_3 is independent of the Y_1 and Y_2 attribute, the structure of these three output attributes should be presented as $\{(Y_1, Y_2), (Y_3)\}$.

To solve above three challenges, the modularity property of a rule set is used to distinguish from the local and global methods. In our proposed MORStreaming system, a rule R is an implication in the form $A \Rightarrow C$ where the consequent C is a kind of correlation of output attributes, and the antecedent A is a conjunction of logical operators based on input attributes,. The logical operators A are called literals (L) and have distinct forms depending on the input attribute. For example, if inputs are real numbers, literals can have the forms $L = (X_1 \leq v)$ or $L = (X_2 > v)$ meaning the value of first input attribute X_1 of tth sample $\{X(t), Y(t)\}$ must be less or equal to v, and second input attribute $X_2(t)$ must be greater than v, respectively. When a rule R covers the tth sample $\{X(t), Y(t)\}$, the consequent C returns a kind of correlation of output attributes, such as $\{(Y_1, Y_2), (Y_3)\}$. R is said to cover a sample $\{X(t), Y(t)\}$ if, and only if, input $X(t)$ satisfies all the literals in A. In addition, a default rule D exists in MORStreaming, and with a set of n learned rules $R = \{R_1, \ldots, R_n\}$ to build a rule set (shown in Figure 5.2). Default rules assume there are no relationships between any two output attributes.

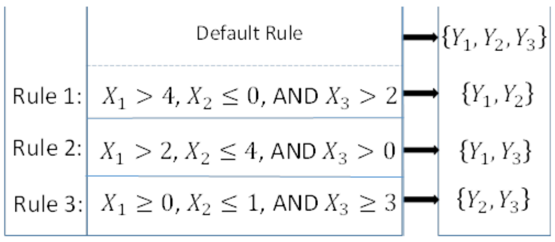

Default Rule
$\{Y_1, Y_2, Y_3\}$
Rule 1:
$X_1 > 4, X_2 \leq 0$, AND $X_3 > 2$
$\{Y_1, Y_2\}$
Rule 2:
$X_1 > 2, X_2 \leq 4$, AND $X_3 > 0$
$\{Y_1, Y_3\}$
Rule 3:
$X_1 \geq 0, X_2 \leq 1$, AND $X_3 \geq 3$
$\{Y_2, Y_3\}$

Figure 5.2 : Rule sets.

To learn the rule set R from streaming data, new instances need to be online learnt, and the rule set can be continuously grown. In MORStreaming, growing the rule set means learning a new rule R_l $A \Rightarrow C$ or a rule R_l is expanded by adding a new literal to the antecedent A_l. Of course, the literal only be added if there is strong evidence that the new literal is the best one among the set of candidates. Hence, a merit function needs to be designed for determining which attribute and split point were selected as a literal. In our proposed MORStreaming, in order to evaluate the merit of splitting an input attribute X_j given the split-point v, the mean-variance ratio (MVR) function. MVR is used and defined as:

$$MVR\left(X_j, v\right) = \frac{1}{m} \sum_{o=1}^{m} VR_o\left(X_j, v\right) \tag{5.1}$$

where $VR_o(X_j, v)$ assesses the merit of splitting the input attribute X_j given the split-point v concerning the output attribute Y_o, and m is the number of output attributes. The variance ratio (VR) is defined as

$$VR_o\left(X_j, v\right) = 1 - \frac{1}{2}\frac{var_o\left(I_L\right)}{var_o(I)} - \frac{1}{2}\frac{var_o\left(I_R\right)}{var_o(I)} \tag{5.2}$$

$$var_o(I) = \frac{1}{m-1} \sum_{i=1}^{m-1} \left(\rho_{i,o}(I) - \bar{\rho}_o(I)\right)^2 \tag{5.3}$$

$$\rho_{i,o}(I) = \left| \frac{cov\left(Y_i(I), Y_o(I)\right)}{\sigma_i \sigma_o} \right| \tag{5.4}$$

where $Y_i(I)$ is the set of ith output attribute of instances that are covered by the rule R_i since its last expansion. $\bar{\rho}_o$ is the mean of the value ρ_o of output attribute $\rho_{i,o}$. E_L is the set of instances $\{X_i, Y_i \in E : X_i \leq v\}$ and E_R is the set of instances $\{X_i, Y_i \in E : X_i > v\}$.

However, considering the literal is chosen from streaming data, the Hoeffding bound is used to guarantee the new literal is the best one in the candidate set, The Hoeffding bound is defined as

$$\epsilon = \sqrt{\frac{P^2 \ln(1/\delta)}{2n}} \tag{5.5}$$

where P represents a range, δ is probability and n is the number of instances.

This equality can be utilized to determine the minimum number n of instances required to expand a rule R_i, i.e., if $r > \epsilon$, where $r = \mathrm{MVR}_{\mathrm{Best}} - \mathrm{MVR}_{2\ \mathrm{ndBest}}$ means the score difference between two best potential literals, we can state that current literal is the best literal with probability $1-\delta$, and no need to collect more instances. Hence, the rule can then be expanded by this literal. Note that a candidate literal must be the best split for each input attribute. In MORStreaming, score difference r only take values in the range $[0, 1]$ interval, so P is set to 1.

Although Hoeffding bound guarantee MVR will increase in the process of learning rule set, it can not guarantee that VR is increased for all output attributes, and may only be a subset of output attributes. For this reason, after selecting the best literal, only output attributes whose VR is increased, are considered to have a correlation. Let C_l be the current correlation of the multi-outputs according to a rule R_l, and E_{best} is the set of instances in the corresponding best branch. The new correlation C'_l is defined as the set of output attributes that effectively increase in var after the expansion of R_l:

$$C'_l = \left\{ Y_o : Y_o \in C_l \wedge \frac{var_o\left(E_{best}\right)}{var_o(E)} > 1 \right\} \tag{5.6}$$

Besides, to keep saving information for the other output attributes, we added a complementary rule R_o, containing the set of output attributes, that were pruned after the split, into rule set. The antecedent of R_o is equal to the antecedent of R_l before the split, and R_o will only be learned for the output attributes $Y_o \in C'_o$, such that

$$C'_o = C_l / C'_l \tag{5.7}$$

Based on the above introduction, the algorithm for learning a rule set in MORStr-eaming is shown in Algorithm 5.1. In Algorithm 5.1, we can see the rule set R is empty in the initial step and initializing the statistics L_D for the default rule R_D. Then, when a new instance $(X(t), Y(t))$ is available, the statistics necessary to expand the rule R_L are updated. If $r > \epsilon$, this rule is expanded. If the expansion of this rule causes a specialization of the rule R_l on a subset of the current output attributes, a complementary rule R_o is also created and added to the rule set. If no rule covers $X(t)$, the statistics of the default rule L_D are updated, and a new default rule R'_d is created if the default rule R_d is expanded.

Algorithm 5.1. Outputs-structure learning

```
Input: Sequence $\{X(t), Y(t)\}$, $R \leftarrow \emptyset$, $D \leftarrow 0$
Output: Rule set $R = R_1, \ldots, R_m$
1: for each $\{X(t), Y(t)\}$ do
2:   for each $R_l \in S(X(t))$ do
3:       $R_c \leftarrow R_l$
4:       update $\boldsymbol{L_l}$
5:       expanded $\leftarrow$ expend($R_l$)
6:       if expanded = TRUE then.
7:           Compute $C'_o$ as in Eq. (5.6).
8:           $C_o \leftarrow C'_o$
9:           $R \leftarrow R \cup \{R_c\}$
10:      end if
11:  end for
12:  if $S(X(t)) = \emptyset$ then
13:      update $\boldsymbol{L_D}$
14:      expanded $\leftarrow$ expend($D$)
15:      if expanded = TRUE then.
16:          $R \leftarrow R \cup D$
17:          $D \leftarrow 0$
18:      end if
19:  end if
20: end for
```

Learning Instances

As an instance-based learning system, how to learn the instances is critical, because the performance significantly dependents on the quality of instances. Instances with a better quality mean the distribution of the instances is closer to the distribution of the original dataset. In this research, we used our proposed simplified version Gm-SOINN (see Algorithm 4.2) to obtain instances.

Instanced-based Prediction Model

When a sample $\left\{X \in \mathbb{R}^d\right\}$ has to be predicted, rules which cover it need to be found at first, then it is necessary to obtain instances that follow these rules to make a prediction. Assume the rule R_l and its complementary rule R_o cover this sample. Hence, instances $F_l = \left\{X \in \mathbb{R}^d, Y \in \mathbb{R}^l\right\}$ and instances $F_o = \left\{X \in \mathbb{R}^d, Y \in \mathbb{R}^o\right\}$ were obtained respectively, according to rules R_l and R_o, where $l \geq 1$ and $o \geq 1$. Based on these instances, we propose a nonparametric regression function to make the predication of this sample, but our proposed model for multi-output regression. For the sake of notational simplicity, we will assume just two outputs, denoted as $Y_1, Y_2 \in \mathbb{R}$. It is assumed the two output attributes Y_1, Y_2 are correlated, so our proposed multi-output nonparametric regression function [54, 85] directly uses the co-outputs in the expression for the estimator as described below:

$$\widehat{Y_1}(\hat{X}) = \frac{\sum_{i=1}^{n}\left[K_{H_1}^{Y_1}\left(X_i - \hat{X}\right)Y_i + K_{H_2}^{Y_Y Y_2}\left(X_i - \hat{X}\right)Y_2\right]}{\sum_{i=1}^{n}\left[K_{H_1}^{Y_1}\left(X_i - \hat{X}\right) + K_{H_2}^{Y_1 Y_2}\left(X_i - \hat{X}\right)\right]} \tag{5.8}$$

where k^{Y_1} and k^{Y_2} are the kernels that reflect the influence in the predicted output of the Y_1 and Y_2 of instance F_i, respectively.

When one output attribute $\widehat{Y_i}$ is covered by multiple rules, the final values of $\widehat{Y_i}$ is calculated by:

$$\widehat{Y_1}^*(\hat{X}) = \frac{\sum_{k=1}^{N} I_{R_k}\widehat{Y_1}(\hat{X})}{\sum_{k=1}^{N} I_{R_k}} \tag{5.9}$$

where N is the number of rules which cover the input data $\hat{X}$, and I_{R_k} is the number of instances under the rule R_k.

Learning Noise and Concept Drift

Learning noise is very important in streaming data mining because training with noise may impact prediction accuracy. In our system, noise detection is from a local viewpoint because it is performed at each rule. Under each rule, according to the objective of topology learning, a more accurate topology network (i.e., closer to the true data distribution) will be learned simultaneously, as more instances are learned, and thereby resulting in fewer instances connected to this noise. Hence, it meets the deleting condition more readily (refer to details from line 13 to line 15 of Algorithm 4.2).

Learning concept drift is also an important concept in data stream mining. In our researched problem, concept drift can be seen from a global and local viewpoint. From a global viewpoint, there is a new structure of outputs when concept drift occurs. For adapting to this concept drift, a new rule that represents this new structure of outputs will be learnt according to Algorithm 5.1. Furthermore, abundant rules (i.e., conflict with new rules) are deleted to save memory and improve prediction accuracy. Besides, in our proposed Algorithm 4.2, a variable e can be used to record the number of instances to be selected as winners. A rule is also identified as an abundant rule when, if the variable e of all instances is not increased after N iterations. From a local viewpoint (i.e., at each rule), although there is no new structure of concept drift, the relationship between the input and output changed under one rule. To adapt to this concept drift, the state of instances under this rule will be updated according to Algorithm 5.1, and thereby the topological structure will finally adapt to new data distribution.

5.3 Experiments

In this section, we illustrate the advantages of our proposed MORStreaming system by comparing it with other multi-output methods. The evaluations involve different scenarios using both artificial and real-world datasets. All experiments are conducted in Python 3.5 version on Windows 7, running on a PC with system configuration Intel Core i5 processor (2.40 GHz) with 8-GB RAM.

5.3.1 Evaluation Strategy

To evaluate the performance of each multi-ouput methods, we employed a prequential strategy which introduced in [71]. In this strategy, a sample is first evaluated by a learning model, and then used to update the learning model. For considering all the metrics, we calculated their mean value for all data streams. In addition, as suggested by [150], a windowed measurement method also be considered, and a non-overlapping sliding window of size 200 is also used. Aiming at reducing the effects of randomness in our evaluation, all multi-output regression methods were performed at least thirty times for each dataset, and the performance was taken as the average between the repetitions.

In this sense, we validate the performance of algorithms in terms of three aspects, i.e., predictive performance, model size, and running time. To validate predictive performance, the average root means square error (ARMSE) was calculated, considering both an overall measurement using all the arrived samples and errors in the sliding window. The ARMSE is defined as:

$$ARMSE = \frac{1}{d}\sum_{t=1}^{d}\sqrt{\frac{\sum_{i=1}^{N}(y_i^t - \hat{y}_i^t)^2}{N}} \tag{5.10}$$

In addition, the running time (s) spent by each system and the total model size (MB) consumed by the learning models were also recorded. In all cases, metrics were recorded in intervals of 200 samples.

5.3.2 Artificial Datasets

MORStreaming has two defining characteristics. First, it uses a different method to the traditional local or global method. Second, it uses a topology learning method to learn instances. Hence, to illustrate the advantages of using a different method, two extensions of MORStreaming were presented following the idea of classical local strategy (MORStreaming-L) and global strategy (MORStreaming-G). In our local strategy, a rule set is learned independently for each output, and the final prediction results is calculated by concatenating individual prediction results of each rule. In contrast, it is choosing a literal in global strategy based on a compromise on reducing the MVR with respect to all output attributes. Besides, to illustrate the advantages of the topology learning method, an extension of MORStreaming is presented, using a KNN method to learn instances (MORStreaming-K).

Next, to demonstrate the advantages of MORStreaming, we experimented with four artificial datasets: 2Dplanes, FriedD, FriedAsyncD, and MV. The datasets are designed by Duarte and Gama [46], and are used in the researches of multi-output regression for data streams. As for concept drift, FriedD and FriedAsyncD datasets contain one concept drift. To be more specific, the concept drifts in FriedD dataset occur simultaneously for all the output variables in the middle location of the data stream, while the concept drifts in FriedAsyncD occur asynchronously. Lastly, the design of MV dataset was inspired by the homonym dataset [46]. Details of the four artificial datasets are described in Table 5.1.

As for parameters, MORStreaming has two main groups of parameters: expansion rule, and learning instances. In order to expand the rule set, the parameters include the probability δ, the minimum number n of samples which required to expand a rule R_i, and the Hoeffding bound ϵ. However, even though the Hoeffding bound ε will decreases considerably with more instances obtained, it is not effi-

Table 5.1 : Artificial datasets

Dataset	SAMPLE SIZE	Inputs	Outputs
2Dplanes	256000	20	8
FriedD	256000	10	4
FriedAsyncD	256000	10	4
MV	256000	20	9

cient to select literals in practice if only dependent on Hoeffding bound. Hence, a threshold τ is defined in our proposed MORStreaming system, and if $\epsilon < \tau$, the literal with the higher MVR is chosen, and the rule is expanded considering the literal. In order to learn instances, the parameters include parameter λ, which is used to define the frequency of neuron removal, and parameter $age_{\max}$ is defined as the lifetime of each edge. In our proposed MORStreaming system, the parameter λ equals the parameter n in rule expansion. Moreover, due to the KNN algorithm in MORStreaming-K, an important parameter k, representing the number of nearest neighbors, needs to be set. Based on our designed dataset, the parameters refer to the rule expansion were set to $n = 200$ (i.e., equal to the size of the window), $\tau = 0.05$, $\delta = 0.0000001$, and parameters refer to the expansion rule were set to $\lambda = n = 200$, and $age_{\max} = 50$.

The comparative results are shown in Table 5.2, Table 5.3, and Table 5.4, where "*" represents MORStreaming for convenience. In Table 5.2, MORStreaming-L achieved slightly lower ARMSE in artificial datasets than other methods. Table 5.3 shows that MORStreaming-G needed fewer model sizes than other methods. In addition, MORStreaming-K50 required less running time than other artificial dataset's methods (Table 5.4). However, from these three perspectives, MORStreaming outperformed the other methods. The reasons for the higher performance include:

Table 5.2 : Comparative results of ARSME

	2Dplanes	FiredD	FriedAsyncD	MV
$*-L$	**2.736**	**8.643**	**6.976**	**24.112**
$*$	2.741	8.709	7.102	24.539
$*-G$	3.477	10.977	9.234	35.343
$*-K50$	2.845	9.236	8.472	33.684
$*-K200$	2.739	8.716	7.122	24.342

Table 5.3 : Comparative results of Running time

	2Dplanes	FiredD	FriedAsyncD	MV
$*-L$	**89.676**	**401.166**	**403.392**	**689.482**
$*$	97.345	412.008	417.396	709.459
$*-G$	154.023	460.349	506.423	820.943
$*-K50$	92.609	406.755	411.387	701.307
$*-K200$	129.347	434.831	465.391	780.365

1) Compared with MORStreaming-L, although MORStreaming produces a slightly higher error and higher running time than MROStraming-L, MORStreaming needs fewer model sizes than MORStreaming-L; and 2) Compared with MORStreaming-G, MORStreaming has higher predictive accuracy and needs less running time. 3) Compared with MORStreaming-K, the predictive performance of MORStreaming is more stable because MORStreaming-K is dependent on the K value. If the K value becomes large, ARMSE becomes low, but needs more model sizes and needs more running time than MORstreaming. In summary, MORStreaming outperforms MORStreaming-L, and since it can be used to learn a rule set, our proposed method

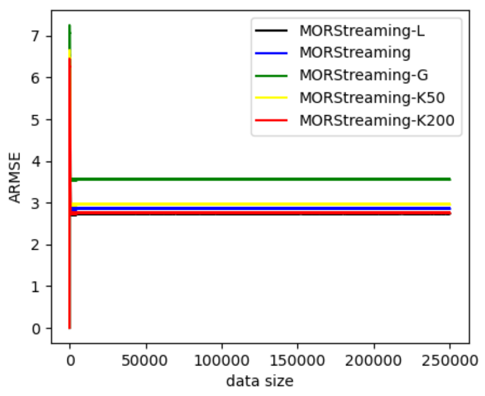
Table 5.4 : Comparative results of Model size

	2Dplanes	FiredD	FriedAsyncD	MV
$*{-}L$	28.290	1203.685	1139.272	1648.291
$*$	15.532	670.717	606.633	805.970
$*{-}G$	**10.652**	**500.134**	**489.306**	**576.004**
$*{-}K50$	12.598	579.060	**553.601**	734.702
$*{-}K200$	17.378	770.631	705.491	979.109

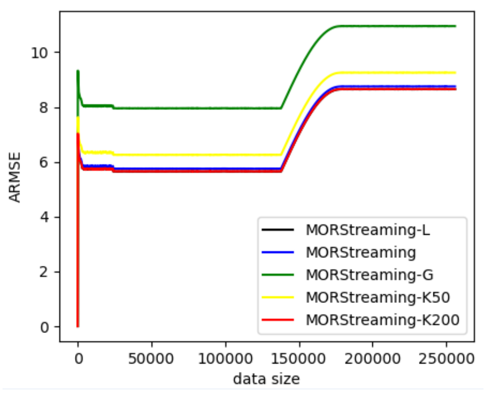
has an advantage over MORStreaming-G. When compared with MORStreaming-K, the higher performance of MORStreaming illustrates the advantage of using a topology learning method to learn instances.

In addition, not only comparing the ARMSE for the whole dataset, the evolution of the ARMSE through time spent was also considered. The line plots are presented for the evaluated datasets, 2Dplanes, FriedD, FiedAsyncD, and MV in Figure 5.3. From these results, different patterns emerge depending on the considered dataset, but considering each dataset itself; all methods presented the same behavior, based on the AMRSE. Since concept drift is present in datasets FriedD and FriedAsyncD, the ARMSE presented a sudden increase in its error, but quickly reaching the convergence. This phenomenon illustrates the efficiency of our proposed mechanism, which used to learn concept drift.

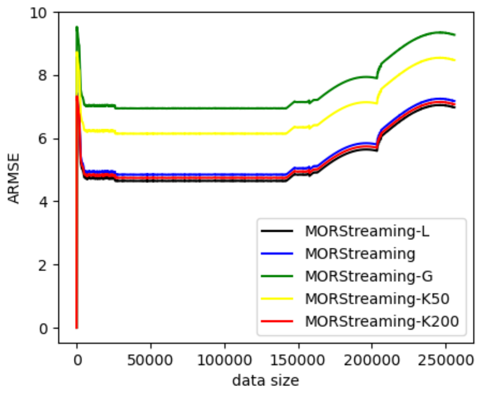
Next, the evolution of the running time was also considered in each dataset (Figure 5.4). From Figure 5.4, we can see similar behaviors, i.e., an approximately linear relationship between the times spent of each multi-output regression method emerged for all datasets. These experimental results illustrate each system meets the requirement of designing a system for streaming data, i.e., time complexity can not be too large. Besides, the experimental results show the number of saved instances



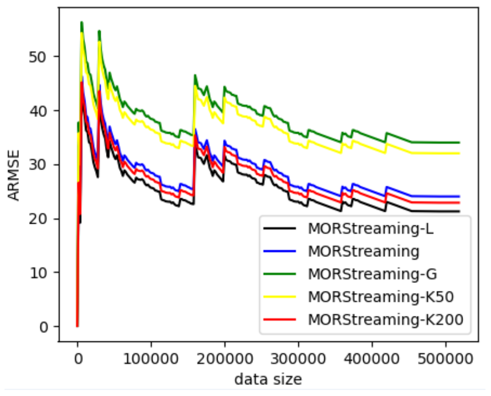
(a) 2DPLANES



(b) FRIEDD



(c) FRIEDASYNCD



(d) MV

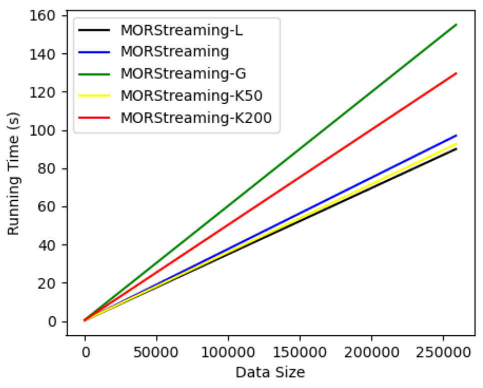
Figure 5.3 : ARMSE based on different datasets.

will impact the time which is spent to learn rules and make predictions.

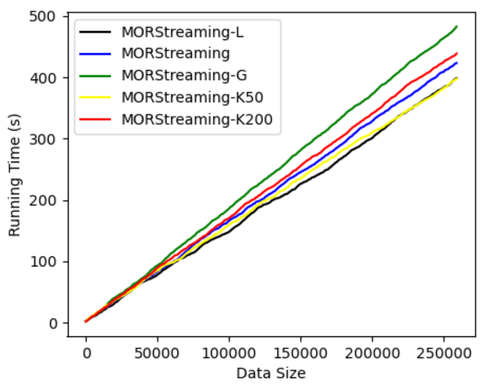
Finally, we compare the evolution of the model size through time. The comparative results are shown in Figure 5.5. Similar to the experimental results of running time, the relationship between the amounts of memory spent by different systems through time was linear for nearly all the datasets.

5.3.3 Real-world Datasets

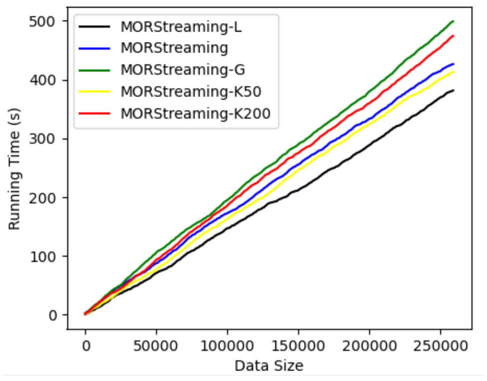
This section presents the experimental results for some of the real-world datasets. The comparison experiments were carried out with rule-based systems, decision tree-based systems, and our proposed MORStreaming system. The rule-based systems



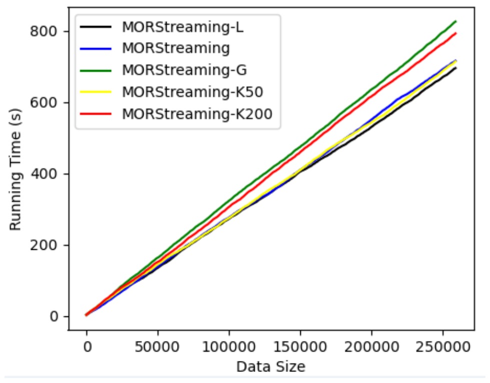
(a) 2DPLANES



(b) FRIEDD



(c) FRIEDASYNCD

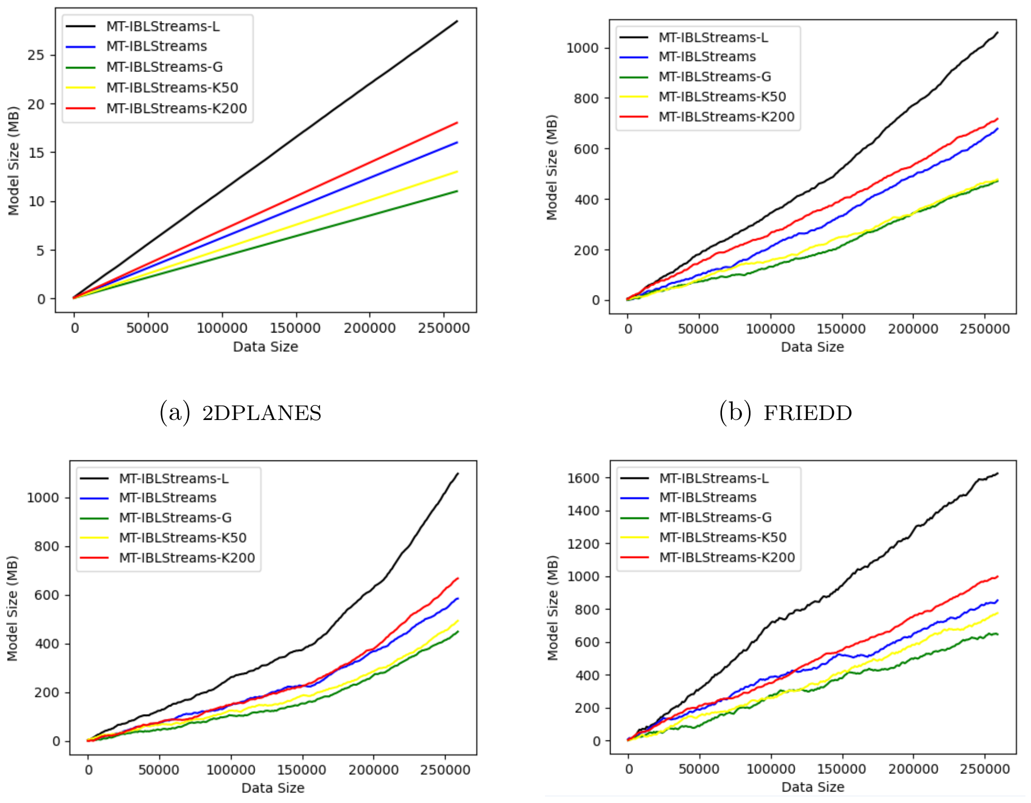


(d) MV

Figure 5.4 : Running time based on different datasets.

are MTR-HT$_{\text{Mean}}$ [46], and MTR-HT$_{\text{Perceptron}}$ [46]. The decision tree-based systems include two types, standard iSOUP-Tree [150], and stacked iSOUP-Tree [150]. Table 5.5 summarizes the main characteristics of each multi-output regression method, including their acronyms, which will be used from here onward.

Next, to better compare the performance, some parameters are fixed for each method following typical configurations of the literature [194]. The attempts of splits were performed once at intervals of 200 samples. The probability δ for the calculation of Hoeffding bound was set to 0.0000001, and the tie-break parameter τ was set to 0.05. In all datasets, we want to provide a 'warm' start for the evaluations, so employee 200 samples to initiate the tree predictors, As for the perceptron weights



(a) 2DPLANES (b) FRIEDD

(c) FRIEDASYNCD (d) MV

Figure 5.5 : Model size based on different datasets.

in AMRules and iSOUP-Tree, they were set to uniform random values in the $[-1, 1]$ interval, and new leaf nodes can inherit their ancestors' weights. The evaluation strategy is the same as the strategy introduced in section IV.A. The comparative experimental results still include predictive performance, running time and model size.

As for real-world datasets, a summary of each dataset used in our evaluations is described in Table 5.6. Their detailed descriptions are as follows:

1) **Bicycles:** The dataset has already been used in two researches for the multi-output regression problem of data streams [46]. This dataset describes the count of rental bikes in an hour, during the period between 2011 and 2012 in

Table 5.5 : Description of the compared methods

Acronym	**Description**
MTR-HT$_{\text{Mean}}$	The variant of MTR-HT which calculates the mean of the outputs at the leaf nodes
MTR-HT$_{\text{Perceptron}}$	The variant of MTR-HT which learns a perceptron model of outputs at the leaf nodes
iSOUP-Tree	The variant of Hoeffding-Tree method which dynamically selects rules between the MTR-HT$_{\text{Mean}}$ and MTR-HT$_{\text{Perceptron}}$
iSOUP-HT	The variant of Hoeffding-Tree method which always use the stacked regressors for making predictions

the “Capital” bikeshare system. The data also contains seasonal and weather information for each rent cases. The task is to predict the count of casual (non-registered), registered and total users.

2) **Eunite03:** The dataset was firstly reported in the competition of the 3rd European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems (2003), and then employed into researches of multi-output regression for data streams [46]. The dataset concerns the continuous production process of manufactured glasses. The input features of each observation include parameters that were employed in producing the glass products, while the outputs refer to the glass quality.

3) **RF1 and RF2:** These two datasets were obtained from the US National Weather and firstly reported by [172], Then, they were employed into researches of multi-output regression for data streams [46]. The “RF1” and “RF2” dataset all describe the river network flows considering a time window of 48 hours in the future, at specific locations. However, the first dataset, RF1,

Table 5.6 : Real-world datasets

Dataset	SAMPLE SIZE	Inputs	Outputs
Bicycles	17379	22	3
*Eunite*03	8064	29	5
*RF*1	9005	64	8
*RF*2	7679	576	8
*SCM*1*d*	9803	280	16
*SCM*20*d*	8966	61	16

uses only the sensor data, whereas the second dataset, RF2, adds a precipitation forecast information (expected rainfall) for each of the measurement sites.

4) **SCM1d and SCM20d:** These two datasets were extracted from the Trading Agent Competition in the Supply Chain Management tournament in 2010. And firstly proposed by [172]. Then, these two datasets were applied into researches of multi-output regression for data streams [46]. Each observation corresponds to a day in the tournament (220 days for each tournament and 18 games during the whole tournament). The input features correspond to the prices considering a specific day in the tournament. Each dataset has 16 outputs, which correspond to the next day mean price (SCM1d) or mean price for 20-days in the future (SCM20d), regarding each product in the simulation.

First, with respect to predictive performance, Table 5.7 summarizes the ARMSE that obtained by each multi-output regression methods. Besides, the smallest ARMSE per dataset is highlighted in bold. As depicted in the table 5.7, MORStreaming was the most accurate system because it obtained the best average rank (2). The iSOUP-

Table 5.7 : Comparison results of ARMSE on real-world datasets

	MTR-HT$_{\text{Mean}}$	MTR-HT$_{\text{Perceptron}}$	iSOUP-Tree	iSOUP-HT	MORStreaming
Bicycles	**86.735 (1)**	141.685 (5)	101.392 (3)	132.971 (4)	87.976 (2)
*Eunite*03	25.869 (3)	26.891 (5)	25.960 (4)	**22.322 (1)**	23.864 (2)
*RF*1	23.156 (4)	28.034 (5)	13.023 (2)	20.042 (3)	**11.035** (1)
*RF*2	26.877 (2)	60.972 (5)	**23.515 (1)**	57.679 (4)	35.959 (3)
*SCM*1*d*	245.912 (2)	354.702 (5)	**215.423 (1)**	317.705 (4)	279.626 (3)
*SCM*20*d*	246.806 (4)	198.315 (5)	147.391 (2)	170.410 (3)	**139.585 (1)**
AVE Rank	2.67	5	2.17	3.17	**2**

HT, which used the stacked perceptron predictors, reached the smallest ARMSE in only one dataset (Eunite03). MTR-HT$_{\text{Mean}}$ also reached the smallest ARMSE in only one dataset (Bicycles). iSOUP-Tree was the second-best performer which reached the second-best average rank (2.17). MTR-HT$_{\text{Perceptron}}$ was the worst predictor.

Table 5.8 : Comparison results of running time on real-world datasets

	MTR-HT$_{\text{Mean}}$	MTR-HT$_{\text{Perceptron}}$	iSOUP-Tree	iSOUP-HT	MORStreaming
Bicycles	**7.925 (1)**	14.685 (5)	14.166 (3)	14.671 (4)	10.679 (2)
*Eunite*03	**11.650 (1)**	15.303 (3)	16.622 (4)	17.656 (5)	12.646 (2)
*RF*1	175.856 (2)	220.333 (5)	191.379 (3)	203.267 (4)	**173.591 (1)**
*RF*2	**150.467 (1)**	171.913 (3)	187.870 (5)	186.942 (4)	159.159 (2)
*SCM*1*d*	**809.125 (1)**	816.528 (4)	814.768 (3)	850.753 (5)	811.162 (2)
*SCM*20*d*	**100.944 (1)**	122.349 (5)	115.231 (4)	112.749 (3)	109.138 (2)
AVE Rank	**1.17**	4.17	3.67	4.17	1.83

With respect to the running times for the compared methods, the simplest alternative of the Hoeffding tree-based method, MTR-HT$_{\text{Mean}}$, is the fastest predictor in all datasets, as shown in Table 5.8. Again, the shortest running times per dataset are

Table 5.9 : Comparison results of model size on real-world datasets

	MTR-HT$_{\text{Mean}}$	MTR-HT$_{\text{Perceptron}}$	iSOUP-Tree	iSOUP-HT	MORStreaming
Bicycles	4.467 (2)	4.561 (3)	4.631 (4)	5.042 (5)	**2.566 (1)**
*Eunite*03	8.792 (4)	8.695 (3)	8.021 (2)	9.477 (5)	**7.064 (1)**
*RF*1	11.562 (4)	11.434 (2)	11.523 (3)	18.042 (5)	**11.035 (1)**
*RF*2	37.277 (2)	37.697 (3)	38.711 (4)	39.767 (5)	**33.959 (1)**
*SCM*1*d*	665.412 (3)	667.025 (4)	664.232 (2)	707.033 (5)	**649.162 (1)**
*SCM*20*d*	275.068 (2)	278.631 (3)	283.915 (4)	307.045 (5)	**273.188 (1)**
AVE Rank	2.83	3	3.17	5	**1**

in bold. Our proposed MORStreaming needs more running time than MTR-HT$_{\text{Mean}}$, but a faster running time than MTR-HT$_{\text{Perceptron}}$ (the reason is it needs more time to build perception models), so it achieved the second smallest average rank in terms of running time. In general, iSOUP-HT performed as fast as iSOUP-Tree for the majority of the datasets.

Lastly, the model size (MB) of each method are summarized in Table 5.9. From Table 5.9, we can see our proposed MORStreaming system spent fewer memory resources than other competitors. In contrast, iSOUP-HT spent more memory resources than the other competitor's strategies. Excluding our proposed MORStreaming and iSOUP-HT, the remaining datasets compared methods only differed in small amounts, regardless of the compared dataset.

5.4 Summary

To solve the multi-output regression problem of streaming data, we propose a new multi-output regression system, called MORStreaming. In MORStreaming, the characteristics of streaming data such as sequential, infinite, noisy, and evolving factors are considered at the same time. The establishment of MORStreaming system faces two challenges. The first challenge is there is an existing structure in outputs due to the number of outputs greater than one, and the structured-output can change due to concept drift. The second challenge is the need for online learning instances.

Based on these two considerations, we first proposed a structured-outputs learning algorithm to learn specific rules to represent the structure of the outputs. The algorithm is built on adaptive rules, so the rules not only can be learned in an online manner but can also fit the drift of the structured-outputs. We then propose an online instances learning algorithm. Our proposed online instances learning algorithm is able to sequentially handle input data by following the rules that we learned, and saving a topology network, which significantly smaller than the size of the original streaming data. As for evolving and noisy characteristics, a topology network-based detection mechanism is introduced to adapt instances to new data distribution and to delete noisy instances. Experimental results show that our proposed algorithm demonstrates improved generation ability and can adapt well to drift.

Chapter 6

Conclusion and Future Study

6.1 Conclusions

The development of the Internet of Things and Big data generates an increasing demand for real-time prediction in modern life. Hence, as the main type of real-time prediction problem, regression is getting more and more attention. Conventional batch-based regression algorithms are built on a static assumption of independent and identically distributed (i.i.d) data and therefore are not suitable to make a real-time prediction for streaming data. Many subsequent studies have proved that streaming regression algorithms are effective approaches to solve the regression problem of streaming data. In streaming algorithms, data are processed sequentially as well and can be examined in only a few passes (typically just one). In addition, streaming algorithms use limited memory and limited processing time per item. Hence, the streaming algorithm represents a dynamic technique of supervised learning and unsupervised learning that can be applied when training data becomes available gradually over time, or its size is out of system memory limits. The aim of streaming algorithms is to adapt the learning model to new data without forgetting its existing knowledge, so it does not retrain the model. However, recent research of streaming regression algorithms still poses several unsolved and challenging problems in this area, i.e., 1) most of the existing streaming regression algorithms only can handle precise data, but streaming data in many real-world applications with a lot of noisy data. The reason for that is the environment of generating streaming data is a noisy environment. The noisy data impacts the learning process of many streaming regression algorithms, and thereby resulting in the performance of many streaming regression algorithms decrease dramatically; 2) more and more studies on streaming data show that the data distribution is nonstationary, it can change or evolve. Concept drift refers to this unpredictable change of data distribution in streaming data. The performance of a regression algorithm may become worse when concept drift occurs. Hence, concept drift detection and adaptation techniques are effective approaches to solve the problem of distribution changes; 3) in many real-world applications, the regression problem of streaming data becomes more complicated. Two or more outputs instead of single one output need to be predicted, i.e., multi-output regression. However, the multi-output regression only has been discussed extensively for offline, static settings. Few works address how to solve this problem for streaming data.

To solve the above-mentioned challenges, this thesis proposes five concrete algorithms. The findings of this study are summarized as follows:

i. Proposed an online robust support vector regression for noisy streaming data regression. Online robust support vector regression (ORSVR) is an exact incremental regression algorithm for handling data streams that transform the classical v-SVR into a dual regression model. ORSVR captures the characteristics of data distributions very well, which makes ORSVR robust to noise. Additionally, ORSVR determines the upper and lower-bound functions by breaking the large QPP associated with SVR into two smaller QPPs and solving each simultaneously. The KKT conditions are met for each new sample and maintained for existing samples, but each bound of the divided QPP is simpler than in classical v-SVR, which results in faster incremental learning speed. However, the ORSVR requires an additional constraint to be compatible with incremental learning. Therefore, the ORSVR approach incorporates several new methods that constitute the incremental learning algorithm for

ORSVR. One method introduces a procedure for preparing the initial solution prior to incremental learning. The other is an adjustment step to ensure all the weights of the support vectors are greater than 0. The experimental results demonstrate that ORSVR successfully handles noisy data and is faster than other incremental SVR algorithms.

ii. Proposed a Gaussian membership-based self-organizing incremental neural network (Gm-SOINN) to online filter noisy data. Our proposed Gm-SOINN method can learn a topological structure in an online manner, from an unlabeled dataset. Due to introducing fuzzy logic, unlike other SOINN-based methods, Gm-SOINN uses a Gaussian membership to indicate the degree to which nodes are identified as a winner (closest node). In addition, based on the Gaussian membership, some theoretical and technical innovations such as a recursive method of updating nodes and a non-parameter density estimation method, which is called eGMM were proposed. When comparing with other SOINN-based methods, these innovations make Gm-SOINN does not have the stability-plasticity dilemma and need fewer user-decided parameters. When comparing with the fuzzy logic system, Gm-SOINN is not only robust to noise but also obtains better performance in a stationary and nonstationary environment. Besides, although fuzzy logic is introduced, Gm-SOINN does not require to design a method to learn the fuzzy rules. More important, topology learning makes Gm-SOINN can handle some problems, such as Vector Quantization, that cannot directly be handled in fuzzy learning systems.

iii. Proposed a continuous support vector regression for evolving streaming data. Continuous support vector regression (C-SVR) is based on a continuous learning strategy that learns a series of regression model in a series of time windows. However, only one regression model for making a prediction is saved in the computer's memory. A new regression model is learned when each new time window arrives, and the last regression model that was learned in the last time window is discarded. A unique difference with our approach is an added similarity term to the Quadratic programming problem that makes the new regression model dependents on the last regression model by transferring some learned knowledge between time windows. The amount of transferred learned knowledge is determined by the extent of the drift as measured by the competency-based method. Like other online or incremental support vector regression designed to work with streaming data, C-SVR solves the quadratic programming problem in an online manner but achieves better performance than other streaming regression algorithms.

iv. Proposed a novel evolving-fuzzy-neuro system, called the topology learning-based fuzzy random neural network, for streaming data not only with concept drift but with noisy data. In the topology learning-based fuzzy random neural network (TLRFNN), to decrease the sensitivity of prediction accuracy to the system structure, first, we learn multiple fuzzy sets and introduce a randomness layer to assign each fuzzy set a probability. Then, to make fuzzy rules model the nonlinear relationship between inputs and outputs well, we propose a new type of fuzzy rule, and further, a new type of inference is designed. Our designed inference not only fits the nonlinear function well, it also considers the random and fuzzy information simultaneously. Furthermore, the inference of TLRFNN is only influenced by one parameter. This parameter does not need to be learned because it can be decided by a maximum likelihood process. To handle concept drifts, a topology network-based mechanism for concept drift is introduced. As the topology network can accurately represent the real data distribution, TLRFNN can adapt to new data distributions easily and rapidly without detecting concept drift. The experiment results show that our proposed algorithm has better performance in relation to streaming data regression.

v. Proposed an online multi-output regression system, called MORStreaming. To solve the characteristics of streaming data such as sequential, infinite, noisy, and evolving factors at the same time. The establishment of the MORStreaming system faces two challenges. The first challenge is there is an existing structure in outputs due to the number of outputs greater than one, and the structured-output can change due to concept drift. The second challenge is the need for online learning instances. Based on these two considerations, we first proposed a structured-outputs learning algorithm to learn specific rules to represent the structure of the outputs. The algorithm is built on adaptive rules, so the rules not only can be learned in an online manner but can also fit the drift of the structured-outputs. We then propose an online instances learning algorithm. Our proposed online instances learning algorithm is able to sequentially handle input data by following the rules that we learned, and saving a topology network, which significantly smaller than the size of the original streaming data. As for evolving and noisy characteristics, a topology network-based detection mechanism is introduced to adapt instances to new data distribution and to delete noisy instances.

6.2 Future Study

Although we proposed some streaming regression algorithms to solve several unsolved and challenging problems in this area, these algorithms still have some drawbacks. Hence, in a future study, we need to solve the following drawbacks:

i. Theoretically, a decremental learning paradigm for ORSVR could be designed in a similar manner. Further, both the incremental and decremental versions of ORSVR may benefit from leave-one-out cross-validation and/or learning with limited memory in terms of efficiency. However, ORSVR still needs some improvement for use with real-world noisy data streams. From our analysis, we find too many samples were saved into computer memory to slow down the learning speed. Hence, we need to design a method to filter unimportance samples.

ii. In Gm-SOINN, two parameters must be determined by the user, and the two parameters will influence the result of Gm-SOINN. However, it is difficult to automatically determine such two parameters. The reason is the optimal choice of such parameters is different for different tasks, and thereby, it is difficult to give a standard of such parameters for every task. Although these parameters are not so sensitive, we remain hopeful that some methods are useful to automatically deduce the optimal choice of such parameters for the task. Such problems will be addressed in subsequent studies.

iii. In our proposed C-SVR, a fixed-size time window is used to detect concept drift. However, in future studies, we intend to look beyond fixed-size time windows with a similar TAI-SVR method that involves varying the size of the time windows to further improve the accuracy of C-SVR. The reason for this is the interval between the occurrences of two concept drifts is not necessarily fixed, so using time windows of varying sizes is helpful to determine the degree of concept drifts more accurately.

iv. In our proposed evolving-fuzzy-neuro system, the neurons in the fuzzy set layer can be learned by our proposed clustering algorithm. Although our proposed clustering algorithm robust to noise and can accurately represent the real data distribution, a mechanism for over-fitting prevention should be introduced to prevent saving too many neurons.

v. In our proposed MORStreaming, to improve the prediction accuracy of our proposed MORStreaming, we consider the concept drift problem. However, compared with the detection of concept drift in streaming data with a single output: 1) the training data becomes more complex when data with multiple-outputs. This complexity includes the volume of data becoming larger, and a change in the number and scale of each feature of the data stream; 2) the underlying distribution of the streaming data with multiple-output becomes more complex. Specifically, all outputs may have the same underlying distribution, or each output has a unique underlying distribution; and 3) the correlation of outputs becomes more complex. That is, when streaming data a single output, we do not need to consider if the output has a correlation with other outputs. However, in multiple-output streaming data, a output may be correlated with other outputs.

Furthermore, this thesis identifies the following directions as future work:

i. streaming data regression for scarce data. Scare data may manifest in different ways such as imbalanced data, insufficient data, and streaming data with uneven time stamps.

ii. regression for streaming data with temporal dependency. The most important challenge of this problem is drift adaptation with temporal dependency because it needs solid theoretical guarantees other than the learning theory.

iii. regression for multiple data streams. Real-world applications such as online decision-making system often require a method for handling multiple streams at the same time. Therefore, a drift detection and adaptation framework for multiple data streams is needed

Appendix

A. 1

Assuming that N samples have already been handled, when a new sample X_c arrives, Q_{ij} is updated as:

$$Q'_{ij} = \frac{1}{(N+1)} K\left(X_i, X_j\right) \tag{6.1}$$

Then, the margin functions can be updated as:

$$\begin{bmatrix} h'_{1s_1} \\ \vdots \\ h'_{1s_s} \end{bmatrix} = \begin{bmatrix} 1 & Q'_{s_1 s_1} & \dots & Q'_{s_1 s_s} \\ \vdots & \vdots & \ddots & \vdots \\ i & Q'_{s_s s_1} & \dots & Q'_{s_s s_s} \end{bmatrix} \begin{bmatrix} B_1 \\ \alpha_{11} \\ \vdots \\ \alpha_{1s} \end{bmatrix} \tag{6.2}$$

Following ORSVR, when a new sample X_c arrives, the weight α_{1i} is updated. We assume the updated weight to be α'_{1i}, so if the weight α_{1c} is set to 0 according to (11), we have

$$\sum_{i=1}^{N} \alpha'_{1i} = C_1 v_1 (N+1) \tag{6.3}$$

Therefore, the sum of the increment of every weight equals $C_1 v_1$, and we have

$$[1, \dots, 1] \begin{bmatrix} \Delta\alpha_{1i} \\ \vdots \\ \Delta\alpha_{1s} \end{bmatrix} = C_1 v_1 \tag{6.4}$$

According to the KKT conditions of the supporting sets, we have

$$\begin{bmatrix} C_1 v_1(N+1) \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & Q'_{s_1 s_1} & \dots & Q'_{s_1 s_s} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q'_{s_s s_1} & \dots & Q'_{s_s s_s} \end{bmatrix} \begin{bmatrix} B'_1 \\ \alpha'_{11} \\ \vdots \\ \alpha'_{1s} \end{bmatrix} \tag{6.5}$$

Based on (49) and (51), we have

$$\begin{bmatrix} C_1 v_1 \\ -h'_{1s_1} \\ \vdots \\ -h'_{1s_s} \end{bmatrix} = \begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & Q'_{s_1 s_1} & \dots & Q'_{s_1 s_s} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q'_{s_s s_1} & \dots & Q'_{s_s s_s} \end{bmatrix} \begin{bmatrix} \Delta B_1 \\ \Delta\alpha_{11} \\ \vdots \\ \Delta\alpha_{1s} \end{bmatrix} \tag{6.6}$$

So

$$\begin{bmatrix} \Delta B_1 \\ \Delta\alpha_{11} \\ \vdots \\ \Delta\alpha_{1s} \end{bmatrix} = \begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & Q'_{s_1 s_1} & \dots & Q'_{s_1 s_s} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q'_{s_s s_1} & \dots & Q'_{s_s s_s} \end{bmatrix}^{-1} \begin{bmatrix} C_1 v_1 \\ -h'_{1s_1} \\ \vdots \\ -h'_{1s_s} \end{bmatrix} \tag{6.7}$$

Combining (52) and (54), we have

$$\begin{bmatrix} B'_1 \\ \alpha'_{1s_1} \\ \vdots \\ \alpha'_{1s_s} \end{bmatrix} = \begin{bmatrix} B_1 \\ \alpha_{1s_1} \\ \vdots \\ \alpha_{1s_s} \end{bmatrix} + \begin{bmatrix} \Delta B_1 \\ \Delta\alpha_{11} \\ \vdots \\ \Delta\alpha_{1s} \end{bmatrix} = \begin{bmatrix} B_1 \\ \alpha_{1s_1} \\ \vdots \\ \alpha_{1s_s} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 & 1 & \dots & 1 \\ 1 & Q'_{s_1 s_1} & \cdots & Q'_{s_1 s_s} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Q'_{s_s s_1} & \cdots & Q'_{s_s s_s} \end{bmatrix}^{-1}}_{R} \begin{bmatrix} C_1 v_1 \\ \vdots \\ -h' \\ -h' \end{bmatrix} \tag{6.8}$$

Thus, Eq. (3.23) is proved.

Bibliography

[1] M. Abaszade and S. Effati, "Stochastic support vector regression with probabilistic constraints," *Applied Intelligence*, 2017. [Online]. Available: http://link.springer.com/10.1007/s10489-017-0964-6

[2] A. Abdelhalim and I. Traore, "A new method for learning decision trees from rules," in *2009 International Conference on Machine Learning and Applications*, 2009, pp. 693–698.

[3] C. C. Aggarwal, P. S. Yu, J. Han, and J. Wang, "- a framework for clustering evolving data streams," in *Proceedings 2003 VLDB Conference*, J.-C. Freytag, P. Lockemann, S. Abiteboul, M. Carey, P. Selinger, and A. Heuer, Eds. San Francisco: Morgan Kaufmann, 2003, pp. 81 – 92. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780127224428500161

[4] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine learning*, vol. 6, no. 1, pp. 37–66, 1991.

[5] T. Aho, B. Ženko, and S. Džeroski, "Rule ensembles for multi-target regression," in *2009 Ninth IEEE International Conference on Data Mining*, 2009, pp. 21–30.

[6] E. Almeida, C. Ferreira, and J. Gama, "Adaptive model rules from data streams," in *Machine Learning and Knowledge Discovery in Databases*, H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 480–492.

[7] E. Alpaydin, *Introduction to machine learning.* MIT press, 2020.

[8] M. Alvarez and N. D. Lawrence, “Sparse convolved gaussian processes for multi-output regression,” in *Advances in neural information processing systems*, 2009, pp. 57–64.

[9] R. Anderson, Y. S. Koh, G. Dobbie, and A. Bifet, “Recurring concept meta-learning for evolving data streams,” *Expert Systems with Applications*, vol. 138, p. 112832, 2019.

[10] P. Angelov and R. Buswell, “Identification of evolving fuzzy rule-based models,” *IEEE Transactions on Fuzzy Systems*, vol. 10, no. 5, pp. 667–677, 2002.

[11] P. P. Angelov and X. Zhou, “Evolving fuzzy-rule-based classifiers from data streams,” *IEEE Transactions on Fuzzy Systems*, vol. 16, no. 6, pp. 1462–1475, 2008.

[12] M. Awad and R. Khanna, *Support Vector Regression.* Berkeley, CA: Apress, 2015, pp. 67–80.

[13] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno, “Early Drift Detection Method,” *4th ECML PKDD International Workshop on Knowledge Discovery from Data Streams*, vol. 6, no. August 2014, pp. 77–86, 2006. [Online]. Available: http://www.lsi.upc.edu/ abifet/EDDM.pdf

[14] Bai Su, Yi-Dong Shen, and Wei Xu, “Modeling concept drift from the perspective of classifiers,” in *2008 IEEE Conference on Cybernetics and Intelligent Systems*, 2008, pp. 1055–1060.

[15] R. D. Baruah and P. Angelov, “Evolving fuzzy systems for data streams: A survey,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 6, pp. 461–476, 2011.

[16] D. M. Bates and D. G. Watts, *Nonlinear regression analysis and its applications.* Wiley New York, 1988, vol. 2.

[17] V. Bentkus, "On hoeffding's inequalities," *Ann. Probab.*, vol. 32, no. 2, pp. 1650–1673, 04 2004. [Online]. Available: https://doi.org/10.1214/009117904000000360

[18] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. null, p. 281–305, Feb. 2012.

[19] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl, "Moa: Massive online analysis, a framework for stream classification and clustering," in *Proceedings of the First Workshop on Applications of Pattern Analysis*, ser. Proceedings of Machine Learning Research, T. Diethe, N. Cristianini, and J. Shawe-Taylor, Eds., vol. 11. Cumberland Lodge, Windsor, UK: PMLR, 01–03 Sep 2010, pp. 44–50.

[20] H. Borchani, A. M. Martínez, A. R. Masegosa, H. Langseth, T. D. Nielsen, A. Salmerón, A. Fernández, A. L. Madsen, and R. Sáez, "Modeling concept drift: A probabilistic graphical model based approach," in *Advances in Intelligent Data Analysis XIV*, E. Fromont, T. De Bie, and M. van Leeuwen, Eds. Cham: Springer International Publishing, 2015, pp. 72–83.

[21] H. Borchani, G. Varando, C. Bielza, and P. Larrañaga, "A survey on multi-output regression," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 5, no. 5, pp. 216–233, 2015.

[22] A. Cano and B. Krawczyk, "Evolving rule-based classifiers with genetic programming on gpus for drifting data streams," *Pattern Recognition*, vol. 87, pp. 248–268, 2019.

[23] Q. Cao, Y. Pei, K. Akbudak, A. Mikhalev, G. Bosilca, H. Ltaief, D. Keyes, and J. Dongarra, “Extreme-scale task-based cholesky factorization toward climate and weather prediction applications,” in *Proceedings of the Platform for Advanced Scientific Computing Conference*, ser. PASC ’20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3394277.3401846

[24] G. A. Carpenter and S. Grossberg, “The art of adaptive pattern recognition by a self-organizing neural network,” *Computer*, vol. 21, no. 3, pp. 77–88, 1988.

[25] G. A. Carpenter, S. Grossberg, N. Marcuzon, J. H. Reynolds, and D. B. Rosen, “Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analogue Multidimensional Maps,” *IEEE Trans. Neural Networks*, vol. 3, pp. 698–713, 1992.

[26] G. Cauwenberghs and T. A. Poggio, “Incremental and decremental support vector machine learning,” in *NIPS*, 2000, pp. 409–415. [Online]. Available: http://papers.nips.cc/paper/1814-incremental-and-decremental-support-vector-machine-learning

[27] T. Chai and R. R. Draxler, “Root mean square error (rmse) or mean absolute error (mae)?” *Geoscientific Model Development Discussions*, vol. 7, no. 1, pp. 1525–1534, feb 2014.

[28] S. Chandrasekaran and M. J. Franklin, “Streaming queries over streaming data,” in *VLDB’02: Proceedings of the 28th International Conference on Very Large Databases.* Elsevier, 2002, pp. 203–214.

[29] C.-C. Chang and C.-J. Lin, “Training v-support vector regression: theory and algorithms,” *Neural computation*, vol. 14, no. 8, pp. 1959–1977, 2002.

[30] P.-C. Chang and C.-H. Liu, “A tsk type fuzzy rule based system for stock price prediction,” *Expert Systems with applications*, vol. 34, no. 1, pp. 135–144, 2008.

[31] M. Charikar, L. O’Callaghan, and R. Panigrahy, “Better streaming algorithms for clustering problems,” in *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, 2003, pp. 30–39.

[32] M. Chen, S. Mao, and Y. Liu, “Big data: A survey,” *Mobile networks and applications*, vol. 19, no. 2, pp. 171–209, 2014.

[33] Y. Chen, J. Xiong, W. Xu, and J. Zuo, “A novel online incremental and decremental learning algorithm based on variable support vector machine,” *Cluster Computing*, vol. 22, no. 3, pp. 7435–7445, 2019.

[34] Z. Y. Chen and Z. P. Fan, “Dynamic customer lifetime value prediction using longitudinal data: An improved multiple kernel SVR approach,” *Knowledge-Based Systems*, vol. 43, pp. 123–134, 2013.

[35] Chung-Chun Kung and Chih-Chien Lin, “Fuzzy c-regression model with a new cluster validity criterion,” in *2002 IEEE World Congress on Computational Intelligence. 2002 IEEE International Conference on Fuzzy Systems. FUZZ-IEEE’02. Proceedings (Cat. No.02CH37291)*, vol. 2, 2002, pp. 1499–1504 vol.2.

[36] L. Cohen, G. Avrahami-Bakish, M. Last, A. Kandel, and O. Kipersztok, “Real-time data mining of non-stationary data streams from sensor networks,” *Information Fusion*, vol. 9, no. 3, pp. 344–353, 2008.

[37] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[38] T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi, “An information-theoretic approach to detecting changes in multi-dimensional data streams,”

in *In Proc. Symp. on the Interface of Statistics, Computing Science, and Applications*, 2006.

[39] A. de Haro-García, G. Cerruela-García, and N. García-Pedrajas, "Instance selection based on boosting for instance-based learners," *Pattern Recognition*, vol. 96, p. 106959, 2019.

[40] S. J. Delany, P. Cunningham, A. Tsymbal, and L. Coyle, "A case-based technique for tracking concept drift in spam filtering," in *Applications and Innovations in Intelligent Systems XII*, A. Macintosh, R. Ellis, and T. Allen, Eds. London: Springer London, 2005, pp. 3–16.

[41] P. Diamond and H. Tanaka, *Fuzzy Regression Analysis.* Boston, MA: Springer US, 1998, pp. 349–387.

[42] G. Ditzler and R. Polikar, "Incremental learning of concept drift from streaming imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 10, pp. 2283–2301, 2013.

[43] F. Dong, G. Zhang, J. Lu, and K. Li, "Fuzzy competence model drift detection for data-driven decision support systems," *Knowledge-Based Systems*, vol. 143, pp. 284–294, 2018.

[44] D. M. dos Reis, P. Flach, S. Matwin, and G. Batista, "Fast unsupervised online drift detection using incremental kolmogorov-smirnov test," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 1545–1554.

[45] N. R. Draper and H. Smith, *Applied regression analysis.* John Wiley & Sons, 1998, vol. 326.

[46] J. Duarte and J. Gama, "Multi-target regression from high-speed data streams

with adaptive model rules," in *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, 2015, pp. 1–10.

[47] P. D'Urso, R. Massari, and A. Santoro, "Robust fuzzy regression analysis," *Information Sciences*, vol. 181, no. 19, pp. 4154–4174, 2011.

[48] J. Dutta and C. Lalitha, "Optimality conditions in convex optimization revisited," *Optimization Letters*, vol. 7, no. 2, pp. 221–229, 2013.

[49] E. E. Elattar, J. Goulermas, and Q. H. Wu, "Electric load forecasting based on locally weighted support vector regression," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 4, pp. 438–447, 2010.

[50] N. Elgendy and A. Elragal, "Big data analytics: a literature review paper," in *Industrial conference on data mining.* Springer, 2014, pp. 214–227.

[51] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Transactions on Neural Networks*, vol. 22, no. 10, pp. 1517–1531, 2011.

[52] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2275–2285, 2004.

[53] C. Fahy, S. Yang, and M. Gongora, "Ant colony stream clustering: A fast density clustering algorithm for dynamic data streams," *IEEE Transactions on Cybernetics*, vol. 49, no. 6, pp. 2215–2228, 2019.

[54] J. Fan, "Design-adaptive nonparametric regression," *Journal of the American statistical Association*, vol. 87, no. 420, pp. 998–1004, 1992.

[55] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A Library for Large Linear Classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008.

[56] W. Fan, "Systematic data selection to mine concept-drifting data streams," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 128–137. [Online]. Available: https://doi.org/10.1145/1014052.1014069

[57] W. Fan and A. Bifet, "Mining big data: current status, and forecast to the future," *ACM SIGKDD explorations newsletter*, vol. 14, no. 2, pp. 1–5, 2013.

[58] G. Fasano and A. Franceschini, "A multidimensional version of the Kolmogorov–Smirnov test," *Monthly Notices of the Royal Astronomical Society*, vol. 225, no. 1, pp. 155–170, 03 1987. [Online]. Available: https://doi.org/10.1093/mnras/225.1.155

[59] S. Feng and C. L. P. Chen, "Fuzzy broad learning system: A novel neuro-fuzzy model for regression and classification," *IEEE Transactions on Cybernetics*, vol. 50, no. 2, pp. 414–424, 2020.

[60] M. M. Ferdaus, M. Pratama, S. G. Anavatti, and M. A. Garratt, "Palm: An incremental construction of hyperplanes for data stream regression," *IEEE Transactions on Fuzzy Systems*, vol. 27, no. 11, pp. 2115–2129, 2019.

[61] L. Florack and A. Kuijper, "The topological structure of scale-space images," *Journal of Mathematical Imaging and Vision*, vol. 12, no. 1, pp. 65–79, 2000.

[62] J. H. Friedman and C. B. Roosen, "An introduction to multivariate adaptive regression splines," *Statistical Methods in Medical Research*, vol. 4, no. 3, pp. 197–217, 1995.

[63] B. Fritzke, “A Growing Neural Gas Learns Topologies,” *Advances in Neural Information Processing Systems*, vol. 7, pp. 625–632, 1995.

[64] ——, “Fast learning with incremental rbf networks,” *Neural processing letters*, vol. 1, no. 1, pp. 2–5, 1994.

[65] ——, “A self-organizing network that can follow non-stationary distributions,” in *Artificial Neural Networks — ICANN'97*, W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 613–618.

[66] C. fu Lin and S. de Wang, “Training algorithms for fuzzy support vector machines with noisy data,” *Pattern Recognition Letters*, vol. 25, no. 14, pp. 1647–1656, 2004.

[67] S. Furao and O. Hasegawa, “An incremental network for on-line unsupervised classification and topology learning,” *Neural networks*, vol. 19, no. 1, pp. 90–106, 2006.

[68] S. Furao, T. Ogura, and O. Hasegawa, “An enhanced self-organizing incremental neural network for online unsupervised learning,” *Neural Networks*, vol. 20, no. 8, pp. 893–903, 2007.

[69] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, “Mining data streams: A review,” *SIGMOD Rec.*, vol. 34, no. 2, p. 18–26, Jun. 2005. [Online]. Available: https://doi.org/10.1145/1083784.1083789

[70] P. Gaillard, M. Aupetit, and G. Govaert, “Learning topology of a labeled data set with the supervised generative gaussian graph,” *Neurocomputing*, vol. 71, no. 7, pp. 1283 – 1299, 2008, progress in Modeling, Theory, and Application of Computational Intelligenc. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231208000635

[71] J. Gama, *Knowledge discovery from data streams.* CRC Press, 2010.

[72] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence – SBIA 2004*, A. L. C. Bazzan and S. Labidi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 286–295.

[73] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[74] A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International journal of information management*, vol. 35, no. 2, pp. 137–144, 2015.

[75] E. A. Gehan, "A generalized two-sample wilcoxon test for doubly censored data," *Biometrika*, vol. 52, no. 3/4, pp. 650–653, 1965. [Online]. Available: http://www.jstor.org/stable/2333721

[76] N. A. Ghani, S. Hamid, I. A. T. Hashem, and E. Ahmed, "Social media big data analytics: A survey," *Computers in Human Behavior*, vol. 101, pp. 417–428, 2019.

[77] S. Godbole and S. Sarawagi, "Discriminative methods for multi-labeled classification," in *Pacific-Asia conference on knowledge discovery and data mining.* Springer, 2004, pp. 22–30.

[78] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet, "A survey on ensemble learning for data stream classification," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–36, 2017.

[79] H. M. Gomes, J. P. Barddal, L. E. B. Ferreira, and A. Bifet, "Adaptive random forests for data stream regression," in *26th European Symposium on Artificial*

Neural Networks, ESANN 2018, Bruges, Belgium, April 25-27, 2018, Bruges, Belgium, 2018. [Online]. Available: https://hal.telecom-paris.fr/hal-02412400

[80] P. M. Gonçalves Jr and R. S. M. De Barros, “Rcd: A recurring concept drift framework,” *Pattern Recognition Letters*, vol. 34, no. 9, pp. 1018–1025, 2013.

[81] R. Gray, “Vector quantization,” *IEEE ASSP Magazine*, vol. 1, no. 2, pp. 4–29, 1984.

[82] B. Gu, V. S. Sheng, Z. Wang, D. Ho, S. Osman, and S. Li, “Incremental learning for nu-support vector regression,” *Neural Networks*, vol. 67, pp. 140 – 150, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608015000696

[83] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O’Callaghan, “Clustering data streams: Theory and practice,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, pp. 515–528, 2003.

[84] G. S. Gurjar and S. Chhabria, “A review on concept evolution technique on data stream,” in *2015 International Conference on Pervasive Computing (ICPC)*. IEEE, 2015, pp. 1–3.

[85] L. Györfi, M. Kohler, A. Krzyzak, and H. Walk, *A distribution-free theory of nonparametric regression*. Springer Science & Business Media, 2006.

[86] T. Hagerup and C. Rüb, “A guided tour of chernoff bounds,” *Information Processing Letters*, vol. 33, no. 6, pp. 305 – 308, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/002001909090214I

[87] S. K. Halgamuge, “Self-evolving neural networks for rule-based data processing,” *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2766–2773, 1997.

[88] H. Hammouri, G. Bornard, and K. Busawon, “High gain observer for structured multi-output nonlinear systems,” *IEEE Transactions on automatic control*, vol. 55, no. 4, pp. 987–992, 2010.

[89] P.-Y. Hao, “Pair-v-svr: A novel and efficient pairing nu-support vector regression algorithm,” *IEEE transactions on neural networks and learning systems*, vol. 28, no. 11, pp. 2503–2515, 2016.

[90] A. Haque, L. Khan, M. Baron, B. Thuraisingham, and C. Aggarwal, “Efficient handling of concept drift and concept evolution over stream data,” in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 2016, pp. 481–492.

[91] C. Hennig, “Methods for merging Gaussian mixture components,” *Advances in Data Analysis and Classification*, vol. 4, no. 1, pp. 3–34, 2010.

[92] T. R. Hoens, R. Polikar, and N. V. Chawla, “Learning from streaming data with concept drift and imbalance: an overview,” *Progress in Artificial Intelligence*, vol. 1, no. 1, pp. 89–101, 2012.

[93] A. E. Hoerl and R. W. Kennard, “Ridge regression: Biased estimation for nonorthogonal problems,” *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[94] B.-J. Hou, L. Zhang, and Z.-H. Zhou, “Learning with feature evolvable streams,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 1417–1427. [Online]. Available: http://papers.nips.cc/paper/6740-learning-with-feature-evolvable-streams.pdf

[95] G. Huang, S. Member, S. Song, C. Wu, and K. You, “Robust Support Vector Regression for Uncertain Input and Output Data,” *IEEE TRANSACTIONS*

ON NEURAL NETWORKS AND LEARNING SYSTEMS, vol. 23, no. 11, pp. 1690–1700, 2012.

[96] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 97–106. [Online]. Available: https://doi.org/10.1145/502512.502529

[97] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges.* Springer Nature, 2019.

[98] E. Ikonomovska, J. Gama, and S. Džeroski, "Online tree-based ensembles and option trees for regression on evolving data streams," *Neurocomputing*, vol. 150, pp. 458 – 470, 2015, special Issue on Information Processing and Machine Learning for Applications of Engineering Solving Complex Machine Learning Problems with Ensemble Methods Visual Analytics using Multidimensional Projections. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231214012338

[99] E. Ikonomovska, S. Jafarpour, and A. Dasdan, "Real-time bid prediction using thompson sampling-based expert selection," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1869–1878. [Online]. Available: https://doi.org/10.1145/2783258.2788586

[100] A. J. Izenman, "Reduced-rank regression for the multivariate linear model," *Journal of multivariate analysis*, vol. 5, no. 2, pp. 248–264, 1975.

[101] M. Janssen, H. van der Voort, and A. Wahyudi, "Factors in-

fluencing big data decision-making quality," *Journal of Business Research*, vol. 70, pp. 338 – 345, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0148296316304945

[102] M. Jaworski, P. Duda, and L. Rutkowski, "New splitting criteria for decision trees in stationary data streams," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 6, pp. 2516–2529, 2017.

[103] F. V. Jensen *et al.*, *An introduction to Bayesian networks.* UCL press London, 1996, vol. 210.

[104] J.-Y. Jeong, J.-S. Kang, and C.-H. Jun, "Regularization-based model tree for multi-output regression," *Information Sciences*, vol. 507, pp. 240–255, 2020.

[105] J. R. B. Junior and M. do Carmo Nicoletti, "An iterative boosting-based ensemble for streaming data classification," *Information Fusion*, vol. 45, pp. 66–78, 2019.

[106] T. Kailath, "The divergence and bhattacharyya distance measures in signal selection," *IEEE Transactions on Communication Technology*, vol. 15, no. 1, pp. 52–60, 1967.

[107] N. Kasabov and Qun Song, "DENFIS: dynamic evolving neural-fuzzy inference system and its application for time-series prediction," *IEEE Transactions on Fuzzy Systems*, vol. 10, no. 2, pp. 144–154, apr 2002. [Online]. Available: http://ieeexplore.ieee.org/document/995117/

[108] D. Kifer, S. Ben-David, and J. Gehrke, "Detecting change in data streams," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, ser. VLDB '04. VLDB Endowment, 2004, p. 180–191.

[109] W. Kim, J. Park, J. Yoo, H. J. Kim, and C. G. Park, "Target localization

using ensemble support vector regression in wireless sensor networks," *IEEE Transactions on Cybernetics*, vol. 43, no. 4, pp. 1189–1198, 2013.

[110] G. Klir and B. Yuan, *Fuzzy sets and fuzzy logic.* Prentice hall New Jersey, 1995, vol. 4.

[111] J. Z. Kolter and M. A. Maloof, "Using additive expert ensembles to cope with concept drift," in *Proceedings of the 22nd International Conference on Machine Learning*, ser. ICML '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 449–456. [Online]. Available: https://doi.org/10.1145/1102351.1102408

[112] H. Kondylakis, N. Dayan, K. Zoumpatianos, and T. Palpanas, "Coconut palm: Static and streaming data series exploration now in your palm," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 1941–1944.

[113] I. Koychev, "Gradual forgetting for adaptation to concept drift," *Proceedings of ECAI 2000 Workshop on Current Issues in Spatio-Temporal Reasoning*, pp. 101–106, 2000.

[114] G. Krempl, I. Žliobaite, D. Brzeziński, E. Hüllermeier, M. Last, V. Lemaire, T. Noack, A. Shaker, S. Sievi, M. Spiliopoulou, and J. Stefanowski, "Open challenges for data stream mining research," *SIGKDD Explor. Newsl.*, vol. 16, no. 1, p. 1–10, Sep. 2014. [Online]. Available: https://doi.org/10.1145/2674026.2674028

[115] G. Krempl, I. Žliobaite, D. Brzeziński, E. Hüllermeier, M. Last, V. Lemaire, T. Noack, A. Shaker, S. Sievi, M. Spiliopoulou *et al.*, "Open challenges for data stream mining research," *ACM SIGKDD explorations newsletter*, vol. 16, no. 1, pp. 1–10, 2014.

[116] R. Kurle, B. Cseke, A. Klushyn, P. van der Smagt, and S. Günnemann, "Continual learning with bayesian neural networks for non-stationary data," in *International Conference on Learning Representations*, 2019.

[117] A. Labrinidis and H. V. Jagadish, "Challenges and opportunities with big data," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 2032–2033, 2012.

[118] K. D. Lawrence, *Robust regression: analysis and applications.* Routledge, 2019.

[119] S.-W. Lee, J.-H. Kim, J. Jun, J.-W. Ha, and B.-T. Zhang, "Overcoming catastrophic forgetting by incremental moment matching," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4652–4662. [Online]. Available: http://papers.nips.cc/paper/7051-overcoming-catastrophic-forgetting-by-incremental-moment-matching.pdf

[120] C. Li, F. Wei, W. Dong, X. Wang, Q. Liu, and X. Zhang, "Dynamic structure embedded online multiple-output regression for streaming data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 2, pp. 323–336, 2019.

[121] A. LIAW, "Classification and regression by randomforest," *R News*, vol. 2, pp. 18–22, 2002.

[122] C. P. Lim and R. F. Harrison, "An incremental adaptive network for on-line supervised learning and probability estimation," *Neural Networks*, vol. 10, no. 5, pp. 925 – 939, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608096001232

[123] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, “A symbolic representation of time series, with implications for streaming algorithms,” in *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, 2003, pp. 2–11.

[124] K.-P. Lin and P.-F. Pai, “A fuzzy support vector regression model for business cycle predictions,” *Expert Systems with Applications*, vol. 37, no. 7, pp. 5430–5435, 2010. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0957417410001107

[125] N. Littlestone, “From on-line to batch learning,” in *Proceedings of the second annual workshop on Computational learning theory*, 2014, pp. 269–284.

[126] J. Liu, V. Vitelli, E. Zio, and R. Seraoui, “A novel dynamic-weighted probabilistic support vector regression-based ensemble for prognostics of time series data,” *IEEE Transactions on Reliability*, vol. 64, no. 4, pp. 1203–1213, 2015.

[127] J. Liu and E. Zio, “An adaptive online learning approach for support vector regression: Online-svr-fid,” *Mechanical Systems and Signal Processing*, vol. 76-77, pp. 796 – 809, 2016.

[128] ——, “A svr-based ensemble approach for drifting data streams with recurring patterns,” *Applied Soft Computing*, vol. 47, pp. 553 – 564, 2016.

[129] W. Liu, B. Schmidt, G. Voss, and W. Muller-Wittig, “Streaming algorithms for biological sequence alignment on gpus,” *IEEE transactions on parallel and distributed systems*, vol. 18, no. 9, pp. 1270–1281, 2007.

[130] V. Losing, B. Hammer, and H. Wersing, “Knn classifier with self adjusting memory for heterogeneous concept drift,” in *2016 IEEE 16th international conference on data mining (ICDM)*. IEEE, 2016, pp. 291–300.

[131] C.-J. Lu, T.-S. Lee, and C.-C. Chiu, "Financial time series forecasting using independent component analysis and support vector regression," *Decision Support Systems*, vol. 47, no. 2, pp. 115 – 125, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167923609000323

[132] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.

[133] J. Lu, A. Liu, Y. Song, and G. Zhang, "Data-driven decision support under concept drift in streamed big data," *Complex & Intelligent Systems*, vol. 6, no. 1, pp. 157–163, 2020.

[134] N. Lu, J. Lu, G. Zhang, and R. Lopez De Mantaras, "A concept drift-tolerant case-base editing technique," *Artificial Intelligence*, vol. 230, pp. 108–133, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.artint.2015.09.009

[135] N. Lu, G. Zhang, and J. Lu, "Concept drift detection via competence models," *Artificial Intelligence*, vol. 209, no. 1, pp. 11–28, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.artint.2014.01.001

[136] F. F. Lubis, Y. Rosmansyah, and S. H. Supangkat, "Gradient descent and normal equations on cost function minimization for online predictive using linear regression with multiple variables," in *2014 International Conference on ICT For Smart Society (ICISS)*, 2014, pp. 202–205.

[137] E. Lughofer, C. Cernuda, S. Kindermann, and M. Pratama, "Generalized smart evolving fuzzy systems," *Evolving systems*, vol. 6, no. 4, pp. 269–292, 2015.

[138] E. D. Lughofer, "FLEXFIS: A robust incremental learning approach for evolving Takagi-Sugeno fuzzy models," *IEEE Transactions on Fuzzy Systems*,

vol. 16, no. 6, pp. 1393–1410, 2008.

[139] A. J. Ma, P. C. Yuen, W. W. Zou, and J.-H. Lai, “Supervised spatio-temporal neighborhood topology learning for action recognition,” *IEEE transactions on circuits and systems for video technology*, vol. 23, no. 8, pp. 1447–1460, 2013.

[140] J. Ma, J. Theiler, and S. Perkins, “Accurate on-line support vector regression,” *Neural Computation*, vol. 15, no. 11, pp. 2683–2703, 2003. [Online]. Available: https://doi.org/10.1162/089976603322385117

[141] A. McAfee, E. Brynjolfsson, T. H. Davenport, D. Patil, and D. Barton, “Big data: the management revolution,” *Harvard business review*, vol. 90, no. 10, pp. 60–68, 2012.

[142] L. L. Minku, A. P. White, and X. Yao, “The impact of diversity on online ensemble learning in the presence of concept drift,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 5, pp. 730–742, 2010.

[143] B. Mirza, Z. Lin, and N. Liu, “Ensemble of subset online sequential extreme learning machine for class imbalance and concept drift,” *Neurocomputing*, vol. 149, pp. 316–329, 2015.

[144] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to linear regression analysis.* John Wiley & Sons, 2012, vol. 821.

[145] H. Mouss, D. Mouss, N. Mouss, and L. Sefouhi, “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)*, vol. 2, 2004, pp. 815–818.

[146] M. Narwaria and W. Lin, “Objective image quality assessment based on support vector regression,” *IEEE Transactions on Neural Networks*, vol. 21, no. 3, pp. 515–519, 2010.

[147] K. Nishida and K. Yamauchi, “Detecting concept drift using statistical testing,” in *International conference on discovery science.* Springer, 2007, pp. 264–269.

[148] L. O’callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, “Streaming-data algorithms for high-quality clustering,” in *Proceedings 18th International Conference on Data Engineering.* IEEE, 2002, pp. 685–694.

[149] O. A. Omitaomu, M. K. Jeong, and A. B. Badiru, “Online support vector regression with varying parameters for time-dependent data,” *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 41, no. 1, pp. 191–197, 2011.

[150] A. Osojnik, P. Panov, and S. Džeroski, “Tree-based methods for online multi-target regression,” *Journal of Intelligent Information Systems*, vol. 50, no. 2, pp. 315–339, 2018.

[151] N. R. Pal and J. C. Bezdek, “On cluster validity for the fuzzy c-means model,” *IEEE Transactions on Fuzzy Systems*, vol. 3, no. 3, pp. 370–379, 1995.

[152] B. S. Parker and L. Khan, “Detecting and tracking concept class drift and emergence in non-stationary fast data streams,” in *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[153] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2012.

[154] X. Peng, D. Chen, L. Kong, and D. Xu, “Interval twin support vector regression algorithm for interval input-output data,” *International Journal of*

Machine Learning and Cybernetics, vol. 6, no. 5, pp. 719–732, 2015.

[155] B. Pfahringer, G. Holmes, and R. Kirkby, "New options for hoeffding trees," in *AI 2007: Advances in Artificial Intelligence*, M. A. Orgun and J. Thornton, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 90–99.

[156] M. Pratama, S. G. Anavatti, P. P. Angelov, and E. Lughofer, "Panfis: A novel incremental learning machine," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 55–68, 2014.

[157] M. Pratama, J. Lu, E. Lughofer, G. Zhang, and M. J. Er, "An incremental learning of concept drifts using evolving type-2 recurrent fuzzy neural networks," *IEEE Transactions on Fuzzy Systems*, vol. 25, no. 5, pp. 1175–1192, 2017.

[158] B. L. Pulito, T. R. Damarla, and S. Nariani, "A two-dimensional shift invariant image classification neural network which overcomes the stability/plasticity dilemma," in *1990 IJCNN International Joint Conference on Neural Networks*, 1990, pp. 825–833 vol.2.

[159] D. Puthal, S. Nepal, R. Ranjan, and J. Chen, "Dlsef: A dynamic key-length-based efficient real-time security verification model for big data stream," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 2, Dec. 2016. [Online]. Available: https://doi.org/10.1145/2937755

[160] L. Rettig, M. Khayati, P. Cudré-Mauroux, and M. Piórkowski, "Online anomaly detection over big data streams," in *Applied Data Science*. Springer, 2019, pp. 289–312.

[161] R. T. Rockafellar, "Augmented lagrange multiplier functions and duality in nonconvex programming," *SIAM Journal on Control*, vol. 12, no. 2, pp. 268–285, 1974.

[162] P. Rojanavasu, H. H. Dam, H. A. Abbass, C. Lokan, and O. Pinngern, "A self-organized, distributed, and adaptive rule-based induction system," *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 446–459, 2009.

[163] C. Romer, A. Kandel, and M. Friedman, "On fuzzy random variables," in *Proceedings of 1995 IEEE International Conference on Fuzzy Systems.*, vol. 2, 1995, pp. 987–992 vol.2.

[164] Rui Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005.

[165] D. Sahoo, S. C. Hoi, and B. Li, "Online multiple kernel regression," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*. New York, New York, USA: ACM Press, 2014, pp. 293–302. [Online]. Available: 10.1145/2623330.2623712. http://dl.acm.org/citation.cfm?doid=2623330.2623712

[166] S. Scardapane, D. Comminiello, M. Scarpiniti, and A. Uncini, "Online Sequential Extreme Learning Machine with Kernels," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 9, pp. 2214–2220, 2015.

[167] J. C. Schlimmer and R. H. Granger, "Incremental learning from noisy data," *Machine learning*, vol. 1, no. 3, pp. 317–354, 1986.

[168] A. Shaker and E. Hu, "IBLStreams : a system for instance-based classification and regression on data streams," *Evolving Systems*, vol. 3, pp. 235–249, 2012.

[169] I. Škrjanc, J. Iglesias, A. Sanchis, D. Leite, E. Lughofer, and F. Gomide, "Evolving fuzzy and neuro-fuzzy approaches in clustering, regression, identification, and classification: A Survey," *Information Sciences*, vol. 490, pp. 344–368, 2019.

[170] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[171] Y. Song, J. Lu, H. Lu, and G. Zhang, "Fuzzy clustering-based adaptive regression for drifting data streams," *IEEE Transactions on Fuzzy Systems*, vol. 28, no. 3, pp. 544–557, 2020.

[172] E. Spyromitros-Xioufis, G. Tsoumakas, W. Groves, and I. Vlahavas, "Multi-target regression via input space expansion: treating targets as inputs," *Machine Learning*, vol. 104, no. 1, pp. 55–98, 2016.

[173] J. Struyf and S. Džeroski, "Constraint based induction of multi-objective regression trees," in *International Workshop on Knowledge Discovery in Inductive Databases*. Springer, 2005, pp. 222–233.

[174] A. Tapus, G. Ramel, L. Dobler, and R. Siegwart, "Topology learning and recognition using bayesian programming for mobile robot navigation," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 4. IEEE, 2004, pp. 3139–3144.

[175] M. Tscherepanow, "TopoART: A Topology Learning Hierarchical ART Network," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2010, vol. 6354 LNCS, no. PART 3, pp. 157–167.

[176] A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen, "Dynamic integration of classifiers for handling concept drift," *Information Fusion*, vol. 9, no. 1, pp. 56 – 68, 2008, special Issue on Applications of Ensemble Methods. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1566253506001138

[177] D. Tuia, J. Verrelst, L. Alonso, F. Perez-Cruz, and G. Camps-Valls, "Multioutput support vector regression for remote sensing biophysical parameter estimation," *IEEE Geoscience and Remote Sensing Letters*, vol. 8, no. 4, pp. 804–808, 2011.

[178] D. Vidaurre, C. Bielza, and P. Larrañaga, "A survey of l1 regression," *International Statistical Review*, vol. 81, no. 3, pp. 361–387, 2013.

[179] L.-Y. Wang, C. Park, K. Yeon, and H. Choi, "Tracking concept drift using a constrained penalized regression combiner," *Computational Statistics and Data Analysis*, vol. 108, pp. 52 – 69, 2017.

[180] W. Wang, "An incremental learning strategy for support vector regression," *Neural processing letters*, vol. 21, no. 3, pp. 175–188, 2005.

[181] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean, "Characterizing concept drift," *Data Mining and Knowledge Discovery*, vol. 30, no. 4, pp. 964–994, 2016.

[182] A. Wibisono, W. Jatmiko, H. A. Wisesa, B. Hardjono, and P. Mursanto, "Traffic big data prediction and visualization using fast incremental model trees-drift detection (fimt-dd)," *Knowledge-Based Systems*, vol. 93, pp. 33 – 46, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950705115004165

[183] S. Wu, M. J. Er, and Y. Gao, "A fast approach for automatic generation of fuzzy rules by generalized dynamic fuzzy neural networks," *IEEE Transactions on Fuzzy Systems*, vol. 9, no. 4, pp. 578–594, 2001.

[184] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding, "Data mining with big data," *IEEE transactions on knowledge and data engineering*, vol. 26, no. 1, pp. 97–107, 2013.

[185] Y. Xing, X. Shi, F. Shen, K. Zhou, and J. Zhao, "A Self-Organizing Incremental Neural Network based on local distribution learning," *Neural Networks*, vol. 84, pp. 143–160, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2016.08.011

[186] D. Xu, Y. Shi, I. W. Tsang, Y.-S. Ong, C. Gong, and X. Shen, "Survey on multi-output learning," *IEEE transactions on neural networks and learning systems*, 2019.

[187] J. Xu, *Topological structure and analysis of interconnection networks.* Springer Science & Business Media, 2013, vol. 7.

[188] L. Yang, F. Chao, and Q. Shen, "Generalized adaptive fuzzy rule interpolation," *IEEE Transactions on Fuzzy Systems*, vol. 25, no. 4, pp. 839–853, 2017.

[189] C.-Y. Yeh, C.-W. Huang, and S.-J. Lee, "A multiple-kernel support vector regression approach for stock market price forecasting," *Expert Systems with Applications*, vol. 38, no. 3, pp. 2177 – 2186, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417410007876

[190] H. Yu, "Incremental learning of bayesian networks from concept-drift data," in *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, 2019, pp. 701–704.

[191] H. Yu, J. Lu, and G. Zhang, "An online robust support vector regression for data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. Early Access, pp. 1–14, 2020.

[192] T. Zhai, Y. Gao, H. Wang, and L. Cao, "Classification of high-dimensional evolving data streams via a resource-efficient online ensemble," *Data Mining and Knowledge Discovery*, vol. 31, no. 5, pp. 1242–1265, 2017.

[193] H. Zhang, X. Xiao, and O. Hasegawa, “A load-balancing self-organizing incremental neural network,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 6, pp. 1096–1105, 2014.

[194] L. Zhang, J. Lin, and R. Karim, “Sliding window-based fault detection from high-dimensional data streams,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 2, pp. 289–303, 2017.

[195] P. Zhang, X. Zhu, and Y. Shi, “Categorizing and mining concept drifting data streams,” in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’08. New York, NY, USA: Association for Computing Machinery, 2008, p. 812–820. [Online]. Available: https://doi.org/10.1145/1401890.1401987

[196] W. Zhang, X. Liu, Y. Ding, and D. Shi, “Multi-output ls-svr machine in extended feature space,” in *2012 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications (CIMSA) Proceedings*, 2012, pp. 130–134.

[197] P. Zhao, L.-W. Cai, and Z.-H. Zhou, “Handling concept drift via model reuse,” *Machine Learning*, vol. 109, no. 3, pp. 533–568, 2020.

[198] X. Zhen, M. Yu, X. He, and S. Li, “Multi-target regression via robust low-rank learning,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 2, pp. 497–504, 2018.

[199] Y. Zhu, K. M. Ting, and Z. Zhou, “Multi-label learning with emerging new labels,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 10, pp. 1901–1914, 2018.

[200] Zhuo Li, Shi-Zhong He, and Shaohu Tan, “A refined on-line rule/parameter adaptive fuzzy controller,” in *Proceedings of 1994 IEEE 3rd International*

Fuzzy Systems Conference, 1994, pp. 1472–1477 vol.3.

[201] P. Zikopoulos, C. Eaton *et al.*, *Understanding big data: Analytics for enterprise class hadoop and streaming data.* McGraw-Hill Osborne Media, 2011.