· U C ·

University of Coimbra
Faculty of Sciences and Technology
Department of Electrical and Computer Engineering

Symone Gomes Soares

Ensemble Learning Methodologies for Soft Sensor Development in Industrial Processes

Tese de Doutoramento em Engenharia Electrotécnica e de Computadores, ramo de especialização em Automação e Robótica, orientada pelo Prof. Dr. Rui Alexandre de Matos Araújo e apresentada ao Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

February 2015

· U C ·

Universidade de Coimbra

· U C ·

University of Coimbra
Faculty of Sciences and Technology
Department of Electrical and Computer Engineering

Ensemble Learning Methodologies for Soft Sensor Development in Industrial Processes

by

Symone Gomes Soares

Tese de Doutoramento em Engenharia Electrotécnica e de Computadores, ramo de especialização em Automação e Robótica, orientada pelo Prof. Dr. Rui Alexandre de Matos Araújo e apresentada ao Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Coimbra
February 2015

Agradecimentos

O desenvolvimento desta tese de doutoramento teria sido demasiado difícil, ou até mesmo impossível, sem a contribuição de pessoas e instituições. Agradeço primeiramente a Deus pelas oportunidades concedidas e pela força nos momentos difíceis ao longo desta caminhada de quase cinco anos.

Agradeço ao meu orientador, Professor Dr. Rui Alexandre de Matos Araújo, pela confiança que depositou no meu trabalho, pela orientação e pelos conhecimentos transmitidos. Agradeço também ao Instituto de Sistemas e Robótica da Universidade de Coimbra (ISR-UC) e à empresa AControl pelo suporte na realização deste trabalho.

Agradeço à Fundação para a Ciência e a Tecnologia (FCT) que suportou esta tese através da bolsa de doutoramento que tem a referência SFRH/BD/68515/2010 e que foi apoiada por financiamento do "Programa Operacional Potencial Humano" (POPH) e pelo "Fundo Social Europeu" (FSE). Agradeço ao Projeto "Sistemas Inteligentes de Controlo, Aquisição e Comunicação Industrial" (SInCACI) com referência SInCACI/3120/2009, cofinanciado pelo Quadro de Referência Estratégico Nacional (QREN) no âmbito do "Mais Centro - Programa Operacional Regional do Centro", pela União Europeia através do Fundo Europeu de Desenvolvimento Regional (FEDER), e pela Agência de Inovação (AdI). Agradeço ao Dr. Petr Kadlec, ao M. Eng. Ratko Grbić e ao Dr. Luigi Fortuna por fornecerem *data sets* para o desenvolvimento desta tese.

Um profundo agradecimento aos meus colegas e amigos do ISR-UC, em especial ao Dr. Jérôme Mendes, Ricardo Maia, Dr. Francisco Alexandre, Saeid Rastegar e Alireza Emami. Agradeço em especial ao Professor Dr. Talles Marcelo Gonçalves de Andrade Barbosa pelo incentivo na realização deste trabalho.

Agradeço aos meus amigos, aos quais muitas vezes não disponibilizei tempo e atenção que lhes eram devidos, mas mesmo assim foram fundamentais a longo deste

trabalho.

Um eterno agradecimento a toda a minha família. Aos meus pais pela paciência e amor incondicional sempre demonstrados. Ao meu querido irmão, pelo carinho e amizade em todos os momentos.

Fernando, obrigada por todo apoio, amor, tranquilidade e segurança transmitidos em todos os momentos.

Por fim, agradeço a todos que de alguma forma contribuíram para a realização desta tese. A todos o meu profundo agradecimento.

Abstract

Increasing demands for on-line monitoring and control of industrial processes and their associated variables, and difficulties related to measuring systems have led to the development of predictive models called Soft Sensors (SSs). SSs use computational intelligence methods to estimate difficult-to-measure variables based on some easy-to-measure variables in industrial applications. However, SS development has some difficulties. The performance of the SSs relies on the quality of the data used to extract knowledge during the identification procedure. Other problem is that industrial systems have many complex characteristics (e.g. non-linearity and time-variance). Thus, bringing SSs to real-world industrial applications is a challenge.

This thesis focuses on the development of computational learning methods applied to SSs, with particular emphasis on methodologies for improving the prediction accuracy and the system adaptation, in order to achieve adaptivity and stability in time-varying processes and reduce the maintenance costs. To deal with these issues, this thesis investigates the use of combinations of multiple learning models, a type of structure called ensemble system. These methodologies have demonstrated ability to improve the performance and stability of the systems. However, efficient mechanisms for balancing the diversity, adaptivity, and performance of the models should be investigated and proposed. For this purpose, four main research objectives and research directions are considered.

The first objective is to develop methodologies for the automatic design of Neural Network (NN) ensembles in regression problems. Genetic Algorithm (GA) and Simulated Annealing (SA) methodologies are proposed and compared to select the best subset of models (from a set of models) to be aggregated to the ensemble, taking into account the key factors of ensemble systems (i.e. diversity, number of models, and combination strategy). First, a set of models with high degree of diversity is generated. That is, each model is trained with a different training data set by ap-

plying bootstrap, and the best NN architecture is selected by varying the number of hidden neurons, the activation function, and the synaptic weights initialization. Second, GA and SA are employed to select the best subset of models and the optimal combination.

The second objective is to design an adaptive ensemble for regression which is able to learn samples in the presence of several types of changes and simultaneously retain old information in scenarios where changes may recur. The key idea is to keep a moving data window that slides when a new sample is available. To handle recurring and non-recurring changes, the proposed ensemble uses a new assignment of models' combination weights that takes into account the models' errors on the past and current windows using a discounting factor that decreases or increases the contribution of old windows. New models are launched if the accuracy of the system is decreasing, and inaccurate models can be excluded over time.

The third objective is to design an adaptive ensemble for regression with fast adaptation capability for on-line prediction of variables in time-varying applications. The properties of the proposed ensemble are: on-line inclusion and removal of models to keep only the most accurate models with respect to the current state of the system; dynamic adaptation of the model's combination weights based on their on-line predictions on the most recent samples; and on-line adaptation of the models' parameters.

The fourth objective is to design an on-line ensemble for regression that selects dynamically the best subset models (from a set of models) to form the ensemble. The proposed method employs ordered aggregation to choose the ensemble size and the subset of models based on the minimization of the ensemble error on the newest sample. It is also proposed an adaptive NN using a variable forgetting factor.

The performance and effectiveness of the proposed methodologies are validated and demonstrated using real-world industrial applications, including the estimation of the free lime in a cement kiln process, and other benchmarks for evaluating real-world SS applications. Additionally, experimental results using artificial data sets with several types of changes are presented to demonstrate the effectiveness and accuracy of the proposed methodologies that deal with time-varying environments.

Resumo

A procura crescente por monitorização e controlo *on-line* de processos industriais e suas variáveis associadas, e dificuldades relacionadas com os sistemas de medição disponíveis têm levado ao desenvolvimento de modelos de predição chamados Sensores Virtuais (SVs). SVs utilizam métodos de inteligência computacional para estimar variáveis difíceis de medir tendo por base a utilização de variáveis fáceis de medir em aplicações industriais. Contudo, o desenvolvimento de SVs envolve algumas dificuldades. O desempenho do SV depende da qualidade dos dados utilizados para extrair conhecimento durante o procedimento de identificação. Outro problema é que os sistemas industriais possuem várias características complexas (por exemplo, não-linearidade e variância no tempo). Assim, trazer SVs para aplicações industriais reais é um desafio.

Esta tese foca no desenvolvimento de métodos de aprendizagem computacional aplicados aos SVs, com ênfase específica em metodologias para melhorar a precisão da predição e a adaptação do sistema, de modo a obter adaptabilidade e estabilidade em processos variantes no tempo e reduzir os custos de manutenção. Para lidar com estas questões, esta tese investiga o uso da combinação de múltiplos modelos de aprendizagem, um tipo de estrutura designada por sistemas *ensembles*. Este tipo de metodologia tem demonstrado capacidade de melhorar o desempenho e a estabilidade dos sistemas. Contudo, mecanismos eficientes para balancear a diversidade, adaptabilidade e desempenho dos modelos devem ser investigados e propostos. Assim, quatro principais objetivos de investigação e direções de investigação são considerados.

O primeiro objetivo é desenvolver metodologias para a construção automática de sistemas *ensemble* de Redes Neuronais (RNs) em problemas de regressão. Metodologias baseadas em Algoritmos Genéticos (AG) e *Simulated Annealing* (SA) são propostos e comparados para selecionar o melhor subconjunto de modelos (a partir

de um conjunto de modelos) para constituir o ensemble, tendo em conta os fatores principais de sistemas *ensembles* (ou seja, diversidade, número de modelos e estratégia de combinação). Em primeiro lugar, um conjunto de modelos com elevada diversidade é produzido. Isto é, cada modelo é treinado com diferentes dados de treino utilizando *bootstrap*, e a melhor arquitetura de RN é selecionada variando o número de neurónios na camada oculta, a função de ativação e a inicialização de pesos sinápticos. Em seguida, AG e SA são utilizados para selecionar o melhor subconjunto de modelos e a melhor combinação.

O segundo objetivo é desenvolver um novo sistema *ensemble* adaptativo para regressão que seja capaz de aprender amostras na presença de vários tipos de mudanças e simultaneamente manter informações antigas em cenários em que mudanças podem reaparecer. A ideia principal é manter uma janela deslizante de dados que se move quando uma nova amostra fica disponível. Para tratar mudanças recorrentes e não-recorrentes, o sistema *ensemble* proposto utiliza uma nova atribuição de pesos de combinação de modelos que considera os erros dos modelos nas janelas antigas e recentes, utilizando um fator de desconto que diminui ou aumenta a contribuição de janelas antigas. Novos modelos são incluídos se a precisão do sistema estiver a diminuir, e modelos com baixa precisão podem ser removidos ao longo tempo.

O terceiro objetivo é desenvolver um novo sistema *ensemble* adaptativo para regressão com capacidade de adaptação rápida para a predição *on-line* de variáveis em aplicações variantes no tempo. As propriedades do sistema *ensemble* proposto são: inclusão e remoção *on-line* de modelos para manter apenas os modelos mais precisos em relação ao estado atual do sistema; adaptação dinâmica dos pesos de combinação dos modelos baseada nas predições *on-line* das amostras mais recentes; e adaptação *on-line* dos parâmetros dos modelos.

O quarto objetivo é desenvolver um novo sistema *ensemble* adaptativo para regressão que selecione dinamicamente o melhor subconjunto de modelos (a partir de um conjunto de modelos) para constituir o *ensemble*. O método proposto utiliza agregação ordenada para escolher o tamanho do *ensemble* e o melhor subconjunto de modelos baseados na minimização do erro do *ensemble* na amostra mais recente. Também é proposta uma RN adaptativa utilizando fator de esquecimento variável.

A performance e eficácia das metodologias propostas são validadas e demonstradas utilizando aplicações industriais reais, incluindo a estimação da cal livre num processo de forno de cimenteira, e outros conjuntos de dados importantes para avaliar

aplicações reais de SVs. Além disso, resultados experimentais utilizando conjuntos de dados artificiais com vários tipos de mudanças são apresentados para demonstrar a eficácia e precisão das metodologias propostas que lidam com ambientes variantes no tempo.

Abbreviations and Symbols

General Abbreviations

λ_{DFF}OS-ELM	On-line Sequential Extreme Learning Machine using Directional Forgetting Factor
AddExp	Additive Expert
BP	Back-Propagation
BVD	Bias-variance Decomposition
CSTR	Continuous Stirred-Tank Reactor
DFF	Directional Forgetting Factor
DOER	Dynamic and On-line Ensemble Regression
EBP	Ensemble Before Pruning
ELM	Extreme Learning Machine
EOS-ELM	Ensemble of On-line Sequential Extreme Learning Machine
ES	Ensemble System
FCCU	Fluidized Catalytic Cracking Unit
FF	Forgetting Factor
FPM	First Principle Model
Friedman-GnRG	Global Non-recurring Gradual Friedman data set
Friedman-GRA	Global Recurring Abrupt Friedman data set
Friedman-LA	Local and Abrupt Friedman data set
GA	Genetic Algorithm
GA-NNE	Genetic Algorithm for Designing Neural Network Ensemble
GASEN	Genetic Algorithm based Selective Ensemble
GMM	Gaussian Mixture Model

GPM	Gaussian Process Model
ILLSA	Incremental Local Learning Soft Sensing Algorithm
KF	Kalman Filter
LDO	Light Diesel Oil
LGP	Liquefied Petroleum Gas
LM	Levenberg-Marquardt
LMBP	Levenberg-Marquardt Back-Propagation
LS	Least Squares
ML	Machine Learning
MSE	Mean Squared Error
NCL	Negative Correlation Learning
NIPALS	Non-linear Iterative Partial Least Squares
NN	Neural Network
NNE	Neural Network Ensemble
OA	Ordered Aggregation
OAUE	On-line Accuracy Updated Ensemble
OB	On-line Bagging
OEOA	On-line Ensemble using Ordered Aggregation
OLP	On-line Prediction
OS-ELM	On-line Sequential Extreme Learning Machine
OWE	On-line Weighted Ensemble
PCA	Principal Component Analysis
PCR	Principal Component Regression
PLS	Partial Least Squares
PMPFD	Process Monitoring and Process Fault Detection
RLS	Recursive Least Squares
RPLS	Recursive Partial Least Squares
SA	Simulated Annealing
SA-NNE	Simulated Annealing for Designing Neural Network Ensemble
SD	Standard Deviation

SFDR	Sensor Fault Detection and Reconstruction
SIMPLS	Statistically Inspired Modification of Partial Least Squares
SLFN	Single-hidden Layer Feedforward Network
SRU	Sulfur Recovery Unit
SS	Soft Sensor
SSE	Sum of Squared Error
SVR	Support Vector Regression
SVM	Support Vector Machine
SW	Sliding Window

General Symbols

α	Factor for controlling the inclusion of a new model
θ	Factor for demarcating correct and incorrect predictions
$\boldsymbol{\beta}$	Output synaptic weights vector
$\boldsymbol{\beta}_0$	An initial output synaptic weights vector
$\boldsymbol{\beta}_k$	Output synaptic weights vector of the k-th data block
β_j	Weight connecting the j-th hidden node and the output node
ϑ	Learning parameter of the LMBP algorithm
κ	Discounting factor
ρ	Pruning activation factor
λ_{GASEN}	Threshold for selecting a model for an ensemble in the GASEN algorithm
λ_{NCL}	Correlation penalty term in the cost function of the NCL algorithm
μ	Learning parameter of the LMBP algorithm
$\mathcal{E}$	An ensemble system
ℓ	Number of latent variables
$\mathbf{a}_j$	Weights vector connecting the input nodes and the j-th hidden node
a_{jk}	Weight connecting the j-th hidden node and the k-th input node
b_j	Bias of the j-th hidden node
$\mathbf{D}$	A data set
$\mathbf{D}^0$	An initial training data set
$\mathbf{D}^k$	k-th data set

$\mathbf{D}^{online}$	An on-line data set
$\mathbf{D}^t$	t-th data window
e_n^t	Error of the n-th model on the t-th sample
F	Ensemble system
$F()$	An ensemble output
f	A predictive model
$f()$	Output of a predictive model
f_n	n-th predictive model of an ensemble
$f_n()$	Output of the n-th predictive model of an ensemble
$g()$	Hidden layer activation function of a NN
$\mathbf{H}$	Hidden layer output matrix
$\mathbf{H}_0$	An initial hidden layer output matrix
$\mathbf{H}_k$	Hidden layer output matrix of the k-th data block
$\mathbf{h}_k$	Hidden layer output vector of the k-th data block
$h()$	Output layer activation function of a NN
j	j-th hidden node
L	Number of hidden nodes
$life_n$	Total number of on-line predictions performed by the n-th model
$\mathbf{MSE}^t$	MSE of all the models of the ensemble at the time t
MSE_n^t	MSE of the n-th model at the time t
m	Window's size
N	Number of models of an ensemble
n	n-th model of an ensemble
o_t	Estimated output of the t-th sample
$\mathbf{P}_0$	Initial covariance matrix
$\mathbf{P}_k$	Covariance matrix of the k-th data block
r	Number of inputs
T	Number of samples
T_k	Number of samples of the k-th data set
T_0	Initial number of samples

t	t-th sample
$\mathbf{x}$	Input vector
$\mathbf{x}_t$	Input vector of the t-th sample
x_t^k	k-th input value of an input vector $\mathbf{x}_t$
w_n	Combination weight of the n-th predictive model
$\mathbf{y}$	Output vector
y	Output value
y_t	Output value of the t-th sample

Overview of Soft Sensors (Chapter 2)

σ_x	Standard deviation of the variable x
$\sigma_{x(new)}$	New standard deviation of the variable x
$\sigma_{x(old)}$	Old standard deviation of the variable x
MAD_x	Mean Absolute Deviation (MAD) scale of the variable x
x	An unscaled variable x
$\bar{x}$	Mean of the variable x
$\bar{x}_{(new)}$	New mean of the variable x
$\bar{x}_{(old)}$	Old mean of the variable x
x^*	Median of the variable x
x_t	t-th unscaled sample of the variable x
$x_t^{'}$	t-th scaled sample of the variable x
$x_{max}^{'}$	Maximum value of the scaled variable x
$x_{min}^{'}$	Minimum value of the scaled variable x
x_{max}	Maximum value of the unscaled variable x
x_{min}	Minimum value of the unscaled variable x

Overview of Learning Models Applied to Soft Sensors (Chapter 3)

ℓ_{max}	Maximum number of latent variables
b_{out}	Bias of the output node
$\mathbf{c}$	Output synaptic weights vector
c_j	Weight connecting the j-th hidden node and the output node

E_D	Expectation operator of a data set $\mathbf{D}$
$\mathbf{I}$	Identity matrix
$\mathbf{X}$	Input matrix
$\mathbf{X}_t$	Input matrix of the t-th block data
$\mathbf{y}_t$	Output vector of the t-th block data

Automatic Ensemble Development Using Meta-Heuristics (Chapter 4)

ν	Colling ratio
τ	Temperature
τ_f	Final temperature
τ_i	Initial temperature
$\mathbf{D}_{init}$	An initial (original) data set
$\mathbf{D}_{test}$	A testing data set
$\mathbf{D}_{train}$	A training data set
$\mathbf{D}^n_{train}$	Training data set of the n-th model
$\mathbf{D}_{valid}$	A validation data set
c	Number of bits to represent all the combination strategies
e	Number of individuals selected by elitism for the next generation
G_{max}	Maximum number of generations
$\mathcal{H}$	Hamming distance
K	Number of individuals of the population
MSE^{test}	MSE of the ensemble using a testing data set
$P\%$	Percentage of excluded models (from the ensemble) for obtaining the trimmed mean
$p^{\%}_c$	Crossover probability
$p^{\%}_m$	Mutation probability
$p^{\%}_s$	Selection probability
tr	Number of tries per iteration

An Adaptive Ensemble with Discounting Factor (Chapter 5)

Ψ_n	Total error rate of model f_n
τ_n	Total number of windows where the n-th model has been evaluated

ARE_i^F	Absolute relative error of the ensemble on the i-th sample
ARE_i^n	Absolute relative error of the n-th model on the i-th sample
$\mathcal{D}$	Penalty distribution
d_i	Weight of a sample i
downFactor	Decreasing weight for the samples predicted correctly
upFactor	Increasing weight for the samples predicted incorrectly
ε_n^t	Error rate of the n-th model at the time t

An Adaptive Ensemble Using Ordered Aggregation (Chapter 7)

υ	Parameter of the directional forgetting factor
υ_0	Initial parameter of the directional forgetting factor
γ	Parameter of the directional forgetting factor
γ_0	Initial parameter of the directional forgetting factor
ϵ_t^p	Error prediction of an ordered aggregation at the time t
λ	Forgetting factor
λ_0	Initial forgetting factor
ρ	Parameter of the directional forgetting factor
N_{max}	Maximum number of models of an ensemble
N_{min}	Minimum number of models of an ensemble
$\hat{o}_t^p$	Output prediction of an ordered aggregation at the time t

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Contents

1.1 Main Motivation

In recent decades, the performance requirements for industrial processes have become more difficult to satisfy. Production specifications, environmental and safety regulations, cost and energy efficiency, better plant capacity exploitation are key factors for ensuring safe, robust and low cost production in industries. Additionally, it is necessary to maintain the production with minimum down-time for maintenance. In this scenario, there is a strong need for continuous intelligent control and monitoring of processes in order to optimize the production, and provide mechanisms for efficient monitoring and early control of abnormal situations. For this purpose, reliable measuring systems are valuable tools to cover or support all these objectives.

However, monitoring and controlling industrial systems is a challenge, because such systems suffer from different complex characteristics, such as non-linearity, time-variance, multi-modes, and dangerous operating conditions. Other difficulty

is the large number of interacting variables in the process dynamics. Moreover, installing and maintaining a measurement network system for monitoring a plant with a large number of process variables is never cheap and the required budget can increase significantly the total costs of the plant. Additionally, often, key variables associated to the product quality can be only measured off-line (e.g. by laboratory analyses), introducing significant and undesirable delays and latencies in industrial applications. Due to measurement latencies or by some other reasons, in many cases the measurement of key variables is organized to be done with large sampling intervals/low frequencies. Technological reasons, practical reasons, design reasons, or economical reasons, often dictate such large values of measurement latencies and/or sampling intervals. Thus, it is desirable to develop on-line measuring systems for monitoring complex, dynamical, and time-varying industrial systems with flexibility, reliability, and efficiency.

By these reasons, Soft Sensors (SSs) have been widely investigated and employed as inferential sensing systems for providing on-line estimations of industrial processes' variables. This thesis focuses and proposes intelligent modeling methodologies applied to SS development in industrial processes. Industrial processes are dynamical systems in the sense used in [Narendra and Parthasarathy, 1990]. Additionally, industrial processes are often also time-varying processes/systems, whose characteristics and models change and evolve over time. In this thesis the main focus of attention is on the time-varying characteristic of industrial dynamical processes. In this context, in the sequel in this thesis the term *dynamic* is used with the meaning of *time-varying*.

1.2 Soft Sensor Motivation

In the last two decades, researchers started to use the data being measured and stored in industry for designing predictive models based on such data. These models are called *Soft Sensors*. The term *Soft Sensors* is a combination of the word *software*, because these models are based on computer programs; and the term *sensors* because these models provide similar information as the hardware sensors. SSs are a valuable tool in different industrial fields of application, including pulp industry, refineries, wastewater treatment, copper extraction and fermentation. SSs can help to reduce the need for hardware measuring tools, improve system reliability,

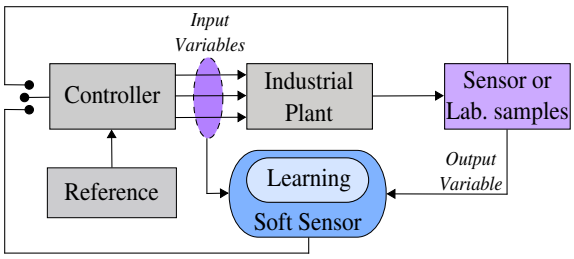


Figure 1.1: An example of a SS connected to a plant.

and offer alternative tools to the implementation of control policies.

The principal application area of SS is the on-line estimation of difficult-to-measure variables (which cannot be automatically measured at all, or can be only measured at high cost, inaccurately, sporadically, or with high delays) based on easy-to-measure variables. Because the estimated variables are often related to the process quality, they are important for process control and process monitoring. A SS can be designed to replace temporally or permanently a sensor in a plant, because the physical sensor may be unavailable in the plant, may have failed or may have been removed for maintenance. Additionally, a SS can be designed for process monitoring and process fault detection, thus being used for detecting the state of the process and possible deviations from the normal conditions of the process. Figure 1.1 shows a case in which a SS is connected to a plant. In this case, the SS can be calibrated using a sensor or laboratory samples, and it can replace the sensor or the laboratory estimates. SSs offer a number of attractive advantages for industries [Fortuna *et al.*, 2006]:

- They are a low-cost alternative to expensive hardware sensors;
- They can operate in parallel with hardware sensors, providing useful information for process monitoring and process fault detection;
- They allow on-line estimations of variables, eliminating the delays introduced by other measurement tools and improving the quality of process control and process monitoring.

SSs can be developed by using first principle models (*model-driven* SSs or *white-box* SSs) that describe the process by physical or chemical background; or by computationally learning models from historical data (*data-driven* SSs or *black-box* SSs).

Model-driven models require physical or chemical knowledge of the systems and processes. Data-driven SSs are implemented using data measured by hardware sensors and thus can alleviate the requirement for physical knowledge, and the physical model development effort. Data-driven models require short development time and can be used for a broad range of applications (flexibility) when compared to model-driven models.

SS development has some difficulties. The performance of SS relies on the quality of the data employed to extract knowledge during the identification procedure. At the preprocessing step, a lot of manual effort is necessary to deal with problems like noise, missing data, inadequate sampling times, outliers, high dimensionality, and others. Additionally, in some applications, the amount of historical data is limited and very long periods of time are necessary to significantly enlarge the amount of data. Small data sets pose some difficulties because they have insufficient information about processes [Fortuna *et al.*, 2009].

Other difficulty is that process plants are rather dynamic, being very difficult for the SS models to react to the changes. Causes for such behavior are changes in the measuring devices, environment changes, and changes of process behavior or of some external process condition. To cope with these effects, methodologies for developing adaptive SS are necessary.

Concerning model learning, the prediction performance of the system can be significantly improved by using ensemble learning algorithms [Polikar, 2012]. The underlying idea is to train a set of models and combine their outputs in order to obtain a final output prediction. Despite the remarkable performance of ensemble systems, more efficient mechanisms for balancing the diversity and adaptivity, and improving performances of the models should be developed. Motivated by these challenges, this thesis investigates and proposes predictive methodologies for SS applications using ensemble learning algorithms.

1.3 Ensemble Learning Motivation

Researches have shown that a combination of multiple models is usually more accurate than any single model [Lan *et al.*, 2009]. The main motivations are the possibility of improving the generalization and stability of the system, including in cases where a small number of samples is available [Fortuna *et al.*, 2009]. An im-

portant issue in ensemble development is the *diversity*. In other words, an accurate ensemble tends to be one where the models make different errors on the same data point. The combination strategy is another important issue for ensuring ensemble accuracy and balancing the diversity between the models.

In ensemble learning, a major drawback is that it is usually necessary a large number of models to ensure the ensemble accuracy. A good way to alleviate this problem is the adequate selection of a subset of models from the original set of models. This approach is known as *ensemble pruning* [Wang and Alhamdoosh, 2013]. It reduces the system complexity, and in some cases, it improves the system accuracy. However, ensemble pruning is a difficult problem whose solution is commonly computationally expensive. Pruning an ensemble with N models requires searching in the space of the $2^N - 1$ non-empty solutions to minimize a cost function correlated with the generalization error. A possible candidate approach to meet this challenge is the application of Genetic Algorithms (GAs) or Simulated Annealing (SA) as an optimization technique to select the best subset of models to be aggregated.

Moreover, most ensemble learning applications are developed off-line and they do not take into account the fact that system/process changes may occur over time. To address this problem, a number of different adaptive mechanisms should be investigated and developed, such as dynamic adaptation of the models' combination weights[1], dynamic adaptation of the models' parameters, dynamic inclusion and removal of models, and dynamic selection of models. Currently, most SS applications using ensemble systems do not add and remove models over time; but these strategies are important keys for improving the prediction performance in time-varying applications. Motivated by these listed problems, this PhD work investigates and develops methodologies for automatic design of ensemble learning systems in order to improve the on-line output prediction in SS applications.

1.4 Thesis Contributions

The main contributions of this research study are:

1. *[Chapter 4], [Soares* et al.*, 2013]:* Design of a new methodology for automatic Neural Network (NN) ensemble development in regression problems. The main

[1] In this thesis, the term *combination weight* refers to the weight/contribution of a model to the final ensemble output.

contribution is the proposal of techniques that select the best subset of models to be aggregated to the ensemble taking into account the key factors of ensemble systems (i.e. diversity, number of models, and combination strategy). First, a set of models with a high degree of diversity is generated. That is, each model is trained with a different training data set by applying bootstrap, and the best NN architecture is selected by varying the number of hidden neurons, the activation function, and the synaptic weights initialization. Then, GA and SA are proposed and compared to select the best subset of models and the optimal combination strategy. Experimental results on two well-known regression data sets and three real-world industrial data sets demonstrate the effectiveness and accuracy of the proposed methodologies over state-of-the-art ensemble approaches.

2. *[Chapter 5], [Soares and Araújo, 2015c]:* Design of a new On-line Weighted Ensemble (OWE) of regressor models which is able to learn incrementally sample by sample in the presence of several types of changes and simultaneously retain old information in scenarios where changes may recur. The key idea is to keep a moving data window that slides when a new sample is available. The main contributions are (2.1) a new dynamic assignment of models' combination weights that takes into account the models' errors on the past and current windows using a discounting factor that decreases or increases the contribution of old windows; (2.2) dynamic removal and inclusion of models; (2.3) regression scope (in contrast with most on-line ensemble applications for handling changes which are devoted to classification tasks); (2.4) thorough analysis of the experimental results using both artificial data sets and industrial data sets; and (2.5) implementation of a new Learn^{++}.NSE algorithm [Elwell and Polikar, 2011] for regression tasks. Experimental results on four artificial data sets (with several types of changes) and two real-world industrial data sets demonstrate that OWE outperforms well-known adaptive ensemble learning methods and adaptive single model learning methods in most cases.

3. *[Chapter 6], [Soares and Araújo, 2015b]:* Design of a new Dynamic and On-line Ensemble Regression (DOER) with fast adaptation capability for on-line prediction of variables in time-varying applications. The contributions (2.2)-(2.4) were incorporated into the DOER method. The new contributions are (3.1)

on-line inclusion and removal of models to keep only the most accurate models with respect to the current state of the system; (3.2) dynamic adaptation of the models' combination weights based on their on-line predictions on the recent samples; and (3.3) on-line adaptation of the models' parameters. Experimental results on four artificial data sets (with several types of changes) and six real-world industrial data sets demonstrate that DOER has higher accuracy in changing environments when compared to well-known adaptive ensemble learning methods and adaptive single model learning methods.

4. *[Chapter 7], [Soares and Araújo, 2015a]:* Design of a new On-line Ensemble using Ordered Aggregation (OEOA). OEOA dynamically selects an optimal ensemble size and composition of the subset of models based on the minimization of the ensemble error on the newest sample. The proposed strategy overcomes the problem of defining the optimal ensemble size, and in most cases it obtains better performance than aggregating all the models. Moreover, the contribution 3 was incorporated into the OEOA method. Other contribution of this work is the proposal of a new on-line and adaptive NN model with variable forgetting factor using the Directional Forgetting Factor (DFF) method [Bobál *et al.*, 2005], called λ_{DFF}OS-ELM (On-line Sequential Extreme Learning Machine using DFF). Experimental results on four artificial data sets (with several types of changes) and five real-world industrial data sets show that λ_{DFF}OS-ELM outperforms the standard Extreme Learning Machine (ELM) [Huang *et al.*, 2006] and On-line Sequential Extreme Learning Machine (OS-ELM) [Liang *et al.*, 2006] algorithms in most cases. Experiments also reveal that λ_{DFF}OS-ELM improves the prediction accuracy of adaptive ensemble learning algorithms in real-world scenarios. Moreover, experimental results show that OEOA delivers more accurate estimations of the output variables in the industrial applications, as well as in several other cases, when compared to the other state-of-the-art ensembles in the literature.

1.5 Thesis Organization

The thesis is organized as follows:

- Chapter 2 provides a background on SS development and SS applications. The background on SS development focuses on the current methodologies and proposes some improvements based on the actual SS applications; a list of the most recent, and state-of-the-art, SS applications is presented.
- Chapter 3 provides an overview of the main machine learning methods applied to SS modeling. The review focuses on single model learning methods and ensemble learning models which are commonly applied to SSs. Moreover, key factors of ensemble learning algorithms are discussed. Finally, the main adaptive learning systems and the main adaptive ensemble learning systems are described.
- Chapter 4 describes the proposed methodologies for automatic NN ensemble development using GA and SA. Experiments are reported to evaluate the effectiveness of the proposed methodologies.
- Chapter 5 presents the OWE ensemble. The discounting factor behavior is also discussed. Experiments are reported to evaluate, and demonstrate the performance and the effectiveness of OWE over state-of-the-art approaches.
- Chapter 6 describes the DOER ensemble. Experiments are reported to evaluate the effectiveness of DOER.
- Chapter 7 presents the OEOA ensemble and the λ_{DFF}OS-ELM model. Experiments are reported to demonstrate the performance and effectiveness of OEOA and λ_{DFF}OS-ELM over state-of-the-art methods.
- Chapter 8 presents concluding remarks. Future research suggestions are also outlined.

Chapter 2

Overview of Soft Sensors

Contents

2.1 Introduction

This Chapter provides an overview of SS applications. First, the main steps to develop data-driven SSs are described. The initial Sections of the Chapter follow the

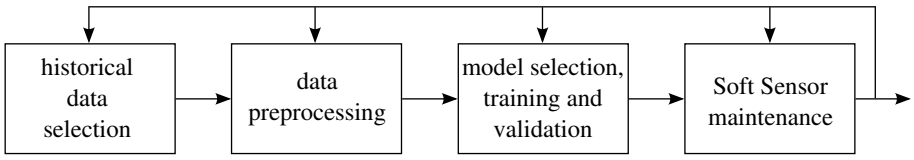


Figure 2.1: The main steps of the SS design.

scheme illustrated in Figure 2.1 [Kadlec *et al.*, 2009]. Section 2.2 describes challenges and techniques in the *historical data selection* step. Section 2.3 describes aspects related to the *data preprocessing* step. Section 2.4 describes challenges and methods related to the *model selection*, *model training*, and *model validation*. Section 2.5 introduces issues related to the *Soft Sensor maintenance*. This Chapter also gives an overview of case studies of SS applications. Section 2.6 describes the main SS application types and provides a list of the most recent, and state-of-the-art, SS publications. The goal is to identify the current research trends, the most common computational methods, and the most popular SS application types. Finally, Section 2.7 provides a list regarding data sets for SS modeling.

2.2 Historical Data Selection

Historical data selection aims to identify how the data are collected and stored in order to select valuable data for SS modeling. The selection takes into account mainly the sampling time.

Multirate systems are abundant in industrial processes since some variables may have slower sampling rates when compared to others. For example, in chemical processes, difficult-to-measure variables (e.g. concentration measurements) are typically obtained after several minutes or hours, while easy-to-measure variables (e.g. temperature measurements) are obtained at each minute and with negligible delay. In this case, a data sample may have missing values of difficult-to-measure variables. In SS applications, the most common procedure is to remove all the data samples that have missing values on the difficult-to-measure variables, so that the retained samples are characterized by the slow rate of the difficult-to-measure variables. Although this technique is straightforward to implement in practice, information may be lost. Several strategies can be employed in order to obtain intermediate values for the difficult-to-measure variables or an appropriate model at a fast sampling rate.

Such common strategies include linear interpolation, polynomial transformation, and data lifting [Lin *et al.*, 2009].

2.3 Data Preprocessing

In data preprocessing, data are processed and transformed so that they can be effectively processed by the learning model. Data preprocessing usually involves the following steps: *data transformation*, *data cleaning*, and *data reduction* [Han and Kamber, 2005].

2.3.1 Data Transformation

In data transformation, data are transformed into forms which are appropriate for the learning process. Different variables may have different numerical magnitudes so that it is important to scale them before the learning process. This procedure is known as *normalization* or *scaling*. Two common scaling techniques are *min-max* normalization and *z-score* (or *zero-mean*) normalization [Fortuna *et al.*, 2006].

The min-max normalization scales the original data into a specific range. Consider that x_{min} and x_{max} are the minimum and maximum values of an unscaled variable x with T samples, where each sample is represented as x_t. The min-max normalization scales each sample x_t into a new range $[x^{'}_{min}, x^{'}_{max}]$ by calculating:

$$x^{'}_t = \frac{x_t - x_{min}}{x_{max} - x_{min}}(x^{'}_{max} - x^{'}_{min}) + x^{'}_{min}, \quad t = 1, \ldots, T, \tag{2.1}$$

where $x^{'}_t$ is a scaled sample of x_t. The z-score normalization scales the original data based on the mean and standard deviation (SD), so that the scaled data have zero mean and unit variance. Each sample x_t of an unscaled variable x is normalized as:

$$x^{'}_t = \frac{x_t - \bar{x}}{\sigma_x}, \quad t = 1, \ldots, T, \tag{2.2}$$

where $\bar{x}$ and σ_x are the mean and SD of x. This method is useful when x_{max} and x_{min} of x are unknown. Because many processes exhibit time-varying behavior and the means and SDs of the variables may change over time, *on-line data scaling* is important to ensure good learning performance and prediction accuracy over time.

Consider an unscaled variable x, where the mean $\bar{x}_{(old)}$ and SD $\sigma_{x(old)}$ are obtained using T (old) samples; when a new unscaled sample x_t is available, it is scaled as [Galicia *et al.*, 2012]:

$$x_t^{'} = \frac{x_t - \bar{x}_{(new)}}{\sigma_{x(new)}}, \tag{2.3}$$

where

$$\bar{x}_{(new)} = \frac{T}{T+1}\bar{x}_{(old)} + \frac{1}{T+1}x_t \ , \tag{2.4}$$

$$\sigma_{x(new)} = \sqrt{\frac{T-1}{T}\sigma^2_{x(old)} + \frac{1}{T}\left(x_t - \bar{x}_{(new)}\right)^2 + \left(\bar{x}_{(new)} - \bar{x}_{(old)}\right)^2} \ , \tag{2.5}$$

and $\bar{x}_{(new)}$ and $\sigma_{x(new)}$ are the new mean and SD of x, respectively.

2.3.2 Data Cleaning

Noisy, incomplete and unreliable data are common in industrial data sets. The data cleaning process addresses the detection of missing and erroneous data. In this Subsection, two main problems are discussed: missing data and outliers.

Missing Data. The main factors that explain why measurements may be missing from data are: because they were forgotten or lost; certain measurements are not applicable for a given variable; or, for a given variable, the designer of the data does not care about the measurements. There are two main ways for dealing with missing data. The first method, known as *ignoring and discarding*, discards samples or variables with missing values. The second method, called *imputation*, aims to replace a missing value using an estimated value. The main imputation procedures include *mean substitution*, where a missing value of a variable x is replaced by the mean of x; *last observation carried forward*, where a missing value is replaced by the last measured value before the missing one [Abusnina and Kudenko, 2013]; and *regression imputation*, where a regression model is used to predict missing values based on the existing data. The choice between these methods depends on the nature and quantity of the missing data. The method of ignoring and discarding is not applicable if the percentage of missing values on the total data is significant. Recent methods based on the expectation-maximization theory have demonstrated ability to deal with missing data [Jin *et al.*, 2012; Khatibisepehr and Huang, 2008].

Outliers. Outliers are inconsistent samples with respect to the majority of the data. Outliers may occur in industrial data due to sensor noise, sensor degradation and/or process disturbances. Outlier detection and treatment are crucial steps because outliers can decrease the SS performance. Common techniques for outlier detection are classified as *univariate* and *multivariate* approaches. Multivariate outlier detection is performed by analyzing the dependence between multiple variables, while univariate outlier detection is performed independently on each variable. The most popular univariate outlier detection is the *3σ edit rule* approach [Fortuna *et al.*, 2006]. In this case, a sample x_t of a variable x (with T samples) is considered an outlier if the following condition is fulfilled:

$$|x_t - \bar{x}| > 3 \cdot \sigma_x, \quad t = 1, \ldots, T, \tag{2.6}$$

where the mean $\bar{x}$ and the standard deviation σ_x are obtained using the T samples of x, and the data is assumed to follow a normal distribution. Robust scaling has been proposed in order to reduce the influence of multiple outliers in estimating the mean and SD. The mean is replaced by the median and the SD is replaced by the Median Absolute Deviation (MAD). This method is known as *Hampel identifier*. The MAD scale of a variable x is defined as [Lin *et al.*, 2007]:

$$\text{MAD}_x = 1.4826 \cdot \text{median}\{|x_t - x^*|\}, \; t = 1, \ldots, T, \tag{2.7}$$

where x^* is the median of the variable x. A sample x_t is an outlier if the following condition is fulfilled:

$$|x_t - x^*| > 3 \cdot \text{MAD}_x. \tag{2.8}$$

Once an outlier is detected, it should be removed or replaced. In SS applications, most authors either remove variables with a high presence of outliers [Ni *et al.*, 2014]; or only discard samples with a presence of outliers [Grbić *et al.*, 2013; Abusnina and Kudenko, 2013]. When the Hampel identifier is used, one strategy is to replace an individual measurement x_t that is detected as an outlier, by the median value x^* of the variable x [Matias *et al.*, 2013].

Multivariate outlier detection is used when the variables are highly correlated, because when only one variable is analyzed, the variability of the other variables are not considered. Popular techniques are Principal Component Analysis (PCA)

and Partial Least Squares (PLS). Statistics using these methods can be employed to detect samples that do not conform with the correlation structure of the data, or that inflate the variance of the data [Fortuna *et al.*, 2006].

2.3.3 Data Reduction

The complexity of a model may depend on the number of variables and/or samples. Data reduction techniques decrease the data into a smaller size, maintaining the integrity and information of the original data set. Such techniques include *dimensionality reduction* which reduces the number of variables, and *numerosity reduction* which reduces the number of samples [Alpaydin, 2004].

Dimensionality Reduction. Dimensionality reduction strategies are classified as *variable extraction* (*feature extraction*) and *variable selection* (*feature selection*).

Variable extraction approaches project the original data into a smaller space so that important variables can be easily identified. The aim is to find a new set of dimensions that is a combination of the original dimensions. PCA is a popular method in industry for reducing high-dimensional inputs into few orthonormal inputs, extracting essential information from the data [Liu *et al.*, 2012c].

Variable selection techniques select a representative subset of variables from the original set of variables. The best subset of variables contains the least number of variables that most contribute to the model accuracy. Industrial data sets may contain a large number of process input variables (e.g. sensors of temperature, flow, pressure, etc). From the SS modeling point of view, high dimensional data sets not only increase the complexity of models, but also can increase the problems of noise, outliers, and missing values, as well as the presence of irrelevant inputs, and can also lead to lower prediction performance. Therefore, variable selection is an important process in SS modeling [Grbić *et al.*, 2013]. Variable selection techniques can be divided into *filter methods* and *wrapper methods* [Chandrashekar and Sahin, 2014].

Filter methods select a subset of variables independently of the choice of the predictor. Such methods rank the variables after some evaluation and then take/retain the best variables. Evaluation metrics include the correlation coefficient [Rogina *et al.*, 2011], mutual information [Vergara and Estéves, 2014], minimum redundancy maximum relevancy criterion using mutual information [Peng *et al.*, 2005], etc. Fil-

ter methods do not make assumptions about the process, and they can overcome overfitting since the variable selection is independent of the learning process [Das, 2001]. However, it is not clear how to determine the cut-off point for demarcating relevant variables.

Wrapper methods select the best subset of variables based on their usefulness to a predictor. That is, the prediction accuracy is used as the metric for the variable selection. Wrapper methods employ some methodology to search the space of all possible subsets of variables. Such methodologies can be optimization algorithms, e.g. GA [Vignolo *et al.*, 2013]. The main advantage of wrapper methods is that the estimated prediction accuracy is the best measurement for the variable selection. However, wrapper methods are computationally expensive because it is necessary to train and test the model for evaluating each subset of variables, making wrapper methods inadequate for high-dimensional data sets.

To take the advantages of both wrapper and filter methods, some researchers consider a hybrid approach of them. Hybrid approaches usually apply a filter method to choose a pool of variables and then a wrapper method is employed to select an optimal subset of variables from the pool of variables [Hsu *et al.*, 2011].

Numerosity Reduction. Few strategies using numerosity reduction have been developed in the SS context. This is because, in most SS applications, only a limited number of samples (e.g. obtained by laboratory analysis) is available. In [Kadlec and Gabrys, 2011], the authors apply *downsampling* to reduce the number of samples to fit a Recursive Partial Least Squares (RPLS) model. In [Han and Kamber, 2005], a number of strategies for numerosity reduction is proposed.

2.4 Model Selection, Training and Validation

This Section describes the key factors in SS modeling: *model selection*, *model training*, and *model validation*.

2.4.1 Model Selection

As explained in Section 1.2, SS models can be classified as *black-box* (*model-driven*) and *white-box* (*data-driven* models). A hybrid approach of them is called *gray-box*

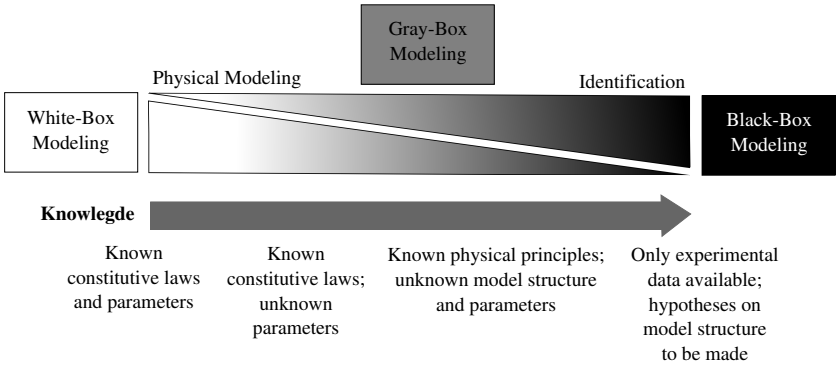


Figure 2.2: An overview of the SS modeling approaches [Fortuna *et al.*, 2006].

model. Figure 2.2 shows an overview of SS models based on physical modeling and identification modeling (computational learning models).

In white-box modeling, mechanistic knowledge obtained from first principles (e.g. physics and chemistry) is employed to design a model, also known as First Principle Model (FPM) [Abonyi, 2002]. In [Escobar *et al.*, 2015], a SS for sensor fault compensation in an evaporator system is modeled using a mathematical model based on algebraic and differential equations. Black-box models have been increasingly employed in SS modeling, since they require little domain expertise of systems and processes, and they can be used for a wide range of SS application types. As industrial processes are complex, and adequate physical or chemical knowledge is often limited, difficult to be obtained and organized in an adequate model, and frequently unavailable in practical terms, black-box models have been increasingly employed in SS modeling. The main disadvantage of black-box models is the dependency on the data quality. Common black-box models for SS modeling are the NN [Rullo *et al.*, 2014; Stanišić *et al.*, 2015], PLS [Shao *et al.*, 2014; Liu *et al.*, 2015], RPLS [He *et al.*, 2015; Ni *et al.*, 2014], and Support Vector Regression (SVR) [Cheng and Liu, 2015; Kaneko and Funatsu, 2014] algorithms. Models which are only partially based both on first principles and on data are called gray-box models. The main advantage of gray-box models is that they exploit the available mechanistic knowledge to improve the model.

Another issue in model selection is the choice between a *linear* model or a *non-linear* model. Some authors suggest to use first a linear model, and then, if it does not have satisfactory performance, one explanation is that the process has a non-linear behavior. However, as almost industrial processes are non-linear, normally SS

designers first consider non-linear models.

A variety of *multivariate statistical* methods are currently employed for SS modeling. Common methods are the PCA, PLS, and RPLS. PCA can be employed to extract meaningful information from the data, detecting and diagnosing abnormal operating conditions in industrial contexts [Liu *et al.*, 2012c]. PLS is able to deal with large dimensional co-linear data by projecting the data into a new space. A number of versions of the PLS exist, including the Non-linear Iterative PLS (NIPALS) [Geladi and Kowalski, 1986] algorithm, and the Statistically Inspired Modification of PLS (SIMPLS) algorithm [Jong, 1993]. SIMPLS calculates PLS components faster and more accurately when compared to the other PLS methods. RPLS has been developed to deal with time-varying processes [Qin, 1998]. In this case, the RPLS model is updated to reflect the current state of the process when new samples become available [Ni *et al.*, 2014].

Nature-inspired modeling is a computational intelligence paradigm for SS modeling, where the goal is to develop algorithms inspired by nature behaviors to deal with real problems [Kadlec and Gabrys, 2007]. Examples of nature-inspired methods include NN and some optimization techniques (e.g. GA). There are many challenges in NN development, including which NN architecture should be chosen, how large the NN should be, and which training algorithm is most suitable. Optimization techniques have been successfully applied to these tasks, including GA [Ding *et al.*, 2012], and SA [Ludermir *et al.*, 2006]. Recently, the Extreme Learning Machine (ELM) has been attracting attention among the scientific community. ELM is a single (or a multiple) hidden-layer NN with faster training time and better performance than other NN models. As many industrial applications exhibit time-varying behavior, it may be difficult for the ELM model to react to changes. Therefore, the research of efficient adaptive mechanisms for ELM models is a promising field in machine learning [Lim *et al.*, 2013].

2.4.2 Model Training

During model training, a phenomenon called *overfitting* may occur. In this case, the model fits too well to the training samples, while presenting poor prediction accuracy (poor generalization capability) on new samples [Caruana *et al.*, 2000]. In the NN context, strategies to avoid overfitting can be applied *before learning*, *during*

learning or *after learning*. In strategies applied before learning, training samples are resampled, or new training samples are artificially created, e.g. by bootstrap [Breiman, 1996] or noise injection [Ho *et al.*, 2010], respectively. Noise injection introduces noises into the input and/or output variables [Fortuna *et al.*, 2009]. It can improve the generalization capability and the convergence of the training, avoiding the possibility of local minina entrapment. Bootstrap is a resampling technique which produces a new training data set by randomly drawing with replacement from the original training data set. It reduces the risk of overfitting that may occur when a model is trained with all samples. Strategies applied during learning aim to overcome the overfitting during the learning procedure. An example is the early stopping criteria [Jeong and Kim, 2005]. The main idea is to inspect the prediction error of a NN on an independent set, a validation data set, so that when the prediction error increases the NN training is stopped to avoid overfitting. In strategies applied after learning, the NN model is trained (with possible overfitting), and then the overfitting is handled. Such methods include pruning techniques, and NN ensembles. The pruning technique eliminates insignificant nodes from a trained NN, since a NN with too many hidden neurons may overfit [Yang and Chen, 2012]. NN ensemble improves the generalization by combining a set of NN models [Lan *et al.*, 2009].

In SS development, one difficulty is to get sufficient data samples for the training. Small data sets offer some challenges due to the insufficient information about the system. Ensemble learning is an efficient way to overcome the limitations associated to small data sets [Polikar, 2006]. In the absence of training samples, bootstrap and noise injection can be employed for drawing different data sets for each model in the ensemble. A well-known ensemble based on bootstrap is Bagging (e.g. **B**ootstrap **agg**regat**ing**) [Breiman, 1996]. Bagging trains each model of the ensemble with a different training data set, obtained by bootstrap. Noise injection can be performed to create different training data sets with the same size as the original training data set, or different training data sets with augmented size when compared to the original training data set [Zhang, 2007].

2.4.3 Model Validation

The model validation procedure verifies whether the SS model can represent the underlying system. In this step, the most relevant issue is to determine the generalization error of the learning model. That is, the capability of the model of performing accurately on unseen samples that were not used during the training procedure. Therefore, the generalization error is obtained using an independent test data set. Common error metrics are cross-validation [Arlot and Celisse, 2010], Mean Squared Error (MSE), root mean squared error [Willmott, 1981], mean absolute error, relative mean absolute error, and correlation coefficient [Ikonomovska, 2012; Fortuna *et al.*, 2006]. Other specific approaches for evaluating on-line learning models can be employed, such as predictive sequential error [Gama *et al.*, 2009].

2.5 Soft Sensor Maintenance

SS maintenance is an important issue that deals with maintaining the SS performance over time. In industrial processes, many factors (e.g. changes in measuring devices due to aging, environment changes, process changes, etc) may lead to the degradation of the SS performance. To deal with these effects, regular model adaptation is necessary to capture all the changes. The main adaptive methodologies include Sliding Window (SW) techniques, recursive learning algorithms, ensemble methods [Kadlec *et al.*, 2011], and just-in-time learning [Saptoro, 2014]. Most SS applications combine a SW technique and learning algorithms [Ni *et al.*, 2014; Facco *et al.*, 2009; Shao *et al.*, 2014]. There are some challenges and problems in the development of adaptive SSs. First, if the model is updated with any abnormal data, its predictive accuracy may deteriorate. Second, in scenarios with recurring changes, the system should conciliate old and new information. Third, the SS should be able to perform well in both gradual and abrupt changes. Therefore, there is still a large amount of work to be done for developing adaptive SSs. The next Chapter discusses the main adaptive mechanisms which can be employed to develop adaptive SSs.

2.6 Soft Sensor Applications

SSs can belong to the following application fields:

- **On-line Prediction (OLP):** the goal is to estimate important variables which cannot be measured on-line using automated or traditional measurement tools. This occurs due to either technological reasons or economical reasons;

- **Process Monitoring and Process Fault Detection (PMPFD):** a SS is employed to monitor the operating state of a process or detect possible process faults;

- **Sensor Fault Detection and Reconstruction (SFDR):** detection and identification of a faulty sensor, and then the sensor is reconstructed or substituted by a SS.

Table 2.1 shows a list of the most recent, and state-of-the-art, SS applications arranged by year. The columns contain the reference, process description, SS application type, main method(s), and information about whether some adaptive mechanism is applied. The list reveals an increasing interest for adaptive methodologies over the past years. A total of 31.25% of the publications in the list do not employ adaptive mechanisms; while 68.75% of the publications in the list use some adaptive method. An interesting aspect observed from Table 2.1 is that the most common SS application field is OLP (contrasting with PMPFD and SFDR). The list also shows a preference for the PLS, RPLS, NN, and SVR algorithms.

2.7 Data Sets for Soft Sensor Modeling

Below, public and private SS data sets are summarized. These data sets can be used to evaluate and validate a SS on different cases and particularities:

1. **Debutanizer column**[1]: the goal is to predict the butane concentration in a debutanizer column of a refinery process using a set of 7 available measurement variables (e.g. temperature and flow) and 2394 samples. The measuring device gives the butane concentration with a delay of about 45 minutes, therefore fast on-line estimation can be the suitable choice [Fortuna *et al.*, 2006];

[1] http://www.springer.com/engineering/control/book/978-1-84628-479-3

Table 2.1: A list of the most recent, and state-of-the-art, SS applications.

Reference	**Process Description**	**SS App. Type**	**Main Method(s)**	**Adaptive Method(s)?**
[Facco *et al.*, 2009]	Polymerization process	OLP	PLS	Yes
[Ahmed *et al.*, 2009]	Polymerization process	OLP	RPLS	Yes
[Shakil *et al.*, 2009]	NO_x and O_2 estimations in boilers	OLP	NN	No
[Liu *et al.*, 2009]	FCCU process	OLP	SVR	Yes
[Liu *et al.*, 2010]	Fermentation process	OLP	SVR	No
[Ge and Song, 2010]	Tennessee Eastman process and distillation process	OLP	PLS, SVR	Yes
[Zhang *et al.*, 2010]	Ozone generation system	OLP	NN	No
[Kadlec and Gabrys, 2011]	Polymerization process	OLP	RPLS	Yes
[Napoli and Xibilia, 2011]	Distillation process	OLP	NN	No
[Soares *et al.*, 2011]	Pulping process	OLP	NN	No
[Xu *et al.*, 2011]	Ammonia synthesis process	OLP	NN	No
[Rogina *et al.*, 2011]	Continuous distillation process	OLP	NN	No
[Chen *et al.*, 2011]	FCCU process	OLP	SVR	Yes
[Jia *et al.*, 2011]	Copper extraction process	OLP	RPLS	Yes
[Liu *et al.*, 2012a]	Air separation process	OLP	PLS	Yes
[Bhattacharya *et al.*, 2012]	Welding process	OLP	NN	No
[Liu *et al.*, 2012b]	Streptokinase fermentation process	OLP	SVR	Yes
[Galicia *et al.*, 2012]	Pulping process	OLP	PLS	Yes
[Zhang *et al.*, 2012]	Continuous annealing process and penicillin fermentation process	PMPFD	PCA	Yes
[Liu *et al.*, 2012c]	Wastewater treatment process	OLP, SFDR	RPLS, PCA	Yes
[Tang *et al.*, 2012]	Grinding process	OLP	PLS, NN	No
[Lingfang and Yechi, 2012]	Oxygen content estimation	OLP	SVR	No
[Grbić *et al.*, 2013]	Powder detergent product process, thermal oxidizer process, and others	OLP	GMM	Yes
[Lv *et al.*, 2013]	NO_x prediction in a boiler	OLP	SVR	Yes
[Pani *et al.*, 2013]	Cement process	OLP	NN	No
[Serpas *et al.*, 2013]	Process monitoring in CSTRs	PMPFD	KF	-
[Liu *et al.*, 2013]	Polymerization production process	OLP	SVR	Yes
[Kim *et al.*, 2013]	Ethylene production process	OLP	PLS	Yes
[Wang and Guo, 2013]	Polymerizing process	OLP	NN	No
[Iliyas *et al.*, 2013]	NO_x prediction in a boiler	OLP	NN	No
[Matias *et al.*, 2013]	Cement process	OLP, SFDR	NN	Yes
[Abusnina and Kudenko, 2013]	Catalyst activation process	OLP	GPM	Yes
[Hu *et al.*, 2013]	Polymerization process and penicillin production process	OLP	PLS	Yes
[Kaneko and Funatsu, 2014]	Alkylaluminum production process and exhaust gas denitration proc.	OLP	SVR	Yes
[Ni *et al.*, 2014]	Three chemical processes	OLP	RPLS	Yes
[Jin *et al.*, 2014]	Fermentation process	OLP	PLS	Yes
[Sharma and Tambe, 2014]	Biochemical processes	OLP	NN, SVR	No
[Ge *et al.*, 2014]	Tennessee Eastman process	OLP, PMPFD	PCR	Yes
[Shao *et al.*, 2014]	SRU process and Debutanizer proc.	OLP	PLS	Yes
[Rullo *et al.*, 2014]	Hydrogen production process	OLP	NN	No
[Xu *et al.*, 2014]	Hydro-isomerization process	OLP	PLS	Yes
[Yuan *et al.*, 2014]	Debutanizer column process and fermentation process	OLP	PCR	Yes
[Stanišić *et al.*, 2015]	Cement process	OLP	NN	Yes
[Escobar *et al.*, 2015]	Evaporation process	OLP, SFDR	FPM	-
[Liu *et al.*, 2015]	Wastewater treatment process	OLP	PLS	Yes
[Cheng and Liu, 2015]	Propylene polymerization process	OLP	SVR	Yes
[He *et al.*, 2015]	Gasoline blending process	OLP	RPLS	Yes
[Soares and Araújo, 2015c]	Polymerization proc. and FCCU proc.	OLP	PLS	Yes
[Soares and Araújo, 2015b]	Cement process and other processes	OLP	NN	Yes
[Soares and Araújo, 2015a]	Cement process and other processes	OLP	NN	Yes

2. **Sulfur Recovery Unit (SRU)**[1]**:** the goal is to predict the hydrogen sulfide (H_2S) and sulfur dioxide (SO_2) concentrations in a SRU of a refinery process. The data set has 5 input variables (mainly related to gas flows) and 10081 samples. The H_2S and SO_2 frequently cause damage to hardware sensors, which are often removed for maintenance. Therefore, a SS can be a valuable tool to predict the H_2S and SO_2 concentrations [Fortuna *et al.*, 2006];

3. **Industrial Fluidized Catalytic Cracking Unit (FCCU)**[2]**:** The FCCU process converts heavy gas oils into lighter hydrocarbon products in a refinery. In traditional operations, gasoline, light diesel oil (LDO), and liquefied petroleum gas (LPG) concentrations can be only measured with a delay of about 8 hours or one day. The fast on-line estimation of these concentrations is important to ensure process quality. The data set has 6 input variables and 104 samples [Liu *et al.*, 2009];

4. **Polymerization process**[2]**:** the objective is to predict the catalyst activity in a polymerization reactor. The catalyst activity values were synthetically produced using chemical reactions equations for simulating a real case. The data covers 1 year of operation of the process plant and contains 15 input variables and 8687 samples [Kadlec and Gabrys, 2011];

5. **Powder detergent production process**[2]**:** the goal is to predict the powder weight in a reactor, since the final product quality depends on the properties of the powder. The powder weight measurements are obtained by laboratory analysis with a frequency of every hour or every half an hour. An appropriate SS can reduce this frequency. The data set consists of 14 input variables and 1962 samples [Grbić *et al.*, 2013];

6. **Thermal oxidizer**[2]**:** the goal is to estimate the nitrogen oxide (NO_x) concentration in a process for air pollution control. The NO_x concentration is measured with gas chromatograph at a low sampling rate; and a SS needs to be designed. The data set consists of 40 input variables and 2053 samples [Grbić *et al.*, 2013];

[2]The data set can be made available for academic purposes by requesting it to the authors.

7. **Cement kiln**[3]**:** the goal is to estimate the free lime (CaO) variable in a cement kiln process, since the product quality is related by the amount of CaO. The data set contains 195 input variables (e.g. temperatures and pressures) and 43469 samples. The input variable samples were recorded with sampling interval of 1 minute, while the output variable samples were obtained with a variety of different sampling intervals ($\geqslant$ 20 minutes) using a laboratory automation system.

2.8 Conclusion

This Chapter presented an overview of SS applications. It described that many issues related to the SS model development and SS maintenance are unaddressed. Concerning model selection, it is necessary to develop additional strategies for improving the adaptivity of single learning algorithms (such as PLS and ELM) in time-varying applications. Concerning model training, strategies to overcome the overfitting and improve the SS performance when an insufficient number of data samples is available should be improved. Concerning SS maintenance, a large amount of work is necessary to develop adaptive SS for dealing with changing environments. The next Chapter will detail these issues. The use of NN models, optimization techniques, and ensemble systems constitute a promising research direction for the development and implementation of improved methodologies for SS applications in industrial processes. NNs have the ability of capturing and mapping complex input-output relationships, and modeling non-linear systems without using prior knowledge; which makes them valuable predictive tools for industrial systems. Ensemble learning has been established as a very promising approach for improving the generalization of learning models. GA and SA optimizations can be employed for automatic ensemble development, for selecting an appropriate number of models and an optimal combination strategy. Additionally, the way ensemble systems may deal with diversity makes them possible strong candidate methods to handle time-varying industrial systems, when adaptation mechanisms are developed and incorporated into the ensemble system.

[3]This data set was provided by "AControl - Automação e Controle Industrial, Lda.", Coimbra, Portugal.

Chapter 3

Overview of Learning Models Applied to Soft Sensors

Contents

3.1 Introduction

Designing intelligent learning systems that can learn from data and adapt to their environments has attracted attention in many fields. In the industrial field, many factors occurring in parallel (e.g. changes, small quantity of data, noise, etc) make difficult the development of such learning and adaptive systems. The goal of this Chapter is to review and describe the main intelligent learning systems applied to SSs. The Chapter is organized as follows. Section 3.2 describes the regression problem and its challenges. Section 3.3 presents two single learning algorithms, PLS and NN, widely employed in SS applications. Section 3.4 describes ensemble learning models. The main objective is to analyze theoretically why ensemble learning models outperform single learning models. Additionally, key factors in ensemble learning systems are discussed. Section 3.5 describes intelligent learning systems that can automatically adapt to their environments. Moreover, a thorough analysis of the main structures of on-line learning systems is provided; and two on-line learning algorithms are described.

3.2 The Regression Problem

Machine learning (ML) methods are computer programs employed to solve a given problem using data or past experience. These programs have the ability of learning from data, where *learning* is the process of obtaining new knowledge. Learning methods are classified into four groups: *supervised learning*, *unsupervised learning*, *semi-supervised learning*, and *reinforcement learning* [Alpaydin, 2004; Mallapragada *et al.*, 2009]. This thesis will consider only supervised learning methods. In supervised learning, the learning (training) process is performed using a set of T samples of the form $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_T, y_T)\}$, each one consisting of values of r independent input variables and one dependent output variable (for the sake of simplicity, this thesis will consider only one output). Each input $\mathbf{x}_t$, of a sample t, is a vector of the form $\mathbf{x}_t = [x_t^1, \ldots, x_t^r]^T$, where the notation x_t^k refers to the k-th input value of $\mathbf{x}_t$; and y_t is the output value of sample t. If the output values are categorical (e.g. "apple", "orange", "lemon"; or "1", "2", "3", "4"; etc), belonging to a countable, and usually fixed, set of finite cardinality, then the supervised learning task is called *classification*. If the output values are numerical, $y \in \mathbb{R}$, then the supervised learn-

ing task is called *regression*. This thesis will consider regression problems. Consider a training data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$ with T training samples. In regression, the main goal is to construct a function (or "machine", model, predictive model, regressor model) f, based on $\mathbf{D}$, which is able to estimate/approximate an output y given an input $\mathbf{x} \in \mathbb{R}^r$, i.e. $f : \mathbb{R}^r \to \mathbb{R}$.

The input and output samples, $\mathbf{x}_t$, and y_t, respectively, may correspond to the input and output samples of a system at the time instant t. However, the components of the input vector $\mathbf{x}_t$ may correspond to variables associated to more than one time instant. Therefore, regression model f may be used as a basis to implement dynamical system models [Narendra and Parthasarathy, 1990] if $\mathbf{x}_t$ is composed of system variables corresponding to different time instants (denoting the existence of system state inside the model). Using this framework, the models proposed in this thesis (Chapters 3 to 7) have the capacity to directly form the basis to implement dynamical system models.

An important issue in ML is the *generalization*, which is the ability of a model to correctly predict samples that have not been presented during the training process. The generalization performance is a useful measure of the quality of a model. The loss of generalization (i.e. when a model cannot generalize well on new samples) is known as *overfitting*. A key problem is how to obtain a powerful model with good generalization. A complex model does not guarantee good generalization, since it usually fits too well on the training samples. Therefore, there is a trade-off between obtaining a model which is not too complex and achieving a good fit to the training samples. This trade-off can be analyzed using the *bias-variance decomposition* (BVD) [Bishop, 2006]. In this analysis, the generalization error of a model is decomposed into two components, *bias* and *variance*. BVD is given by [Brown *et al.*, 2005b]:

$$E_D\{(f(\mathbf{x}) - y)^2\} = \underbrace{(E_D\{f(\mathbf{x})\} - y)^2}_{(\text{bias})^2} + \underbrace{E_D\{(f(\mathbf{x}) - E_D\{f(\mathbf{x})\})^2\}}_{\text{variance}}, \tag{3.1}$$

where $(\mathbf{x}, y)$ is an arbitrary testing point. The expectation operator $E_D\{\cdot\}$ is with respect to a random training data set $\mathbf{D}$ of size T, where each element is independently drawn from an unknown distribution $p(\mathbf{x}, y)$. For the sake of simplicity and

without loss of generality, it is assumed a noise level of zero in the data[1]. The bias measures the difference between the expected prediction and the real output. The variance measures the sensitivity of the estimation with respect to the data set. The main goal of a learner is to minimize the expected loss $E_D\{(f(\mathbf{x}) - y)^2\}$. However, this is not an easy task, because a decrease of the bias term will cause an increase of the variance term, and vice versa. In most cases, a complex model tends to have low bias and high variance, while a simple model tends to have low variance and high bias [Suen *et al.*, 2005]. The model with optimal predictive performance is the one that achieves the best balance between bias and variance.

3.3 Single Learning Models

This Section describes NNs and PLS models. They are powerful learning methods, and are widely employed as data-driven techniques in industry.

3.3.1 Neural Networks

NNs are computational models inspired by biological neuron behaviors and consist of processing elements (neurons) and connections between them (synaptic weights), a neural architecture, and a learning algorithm. NN systems offer a number of attractive properties and capabilities: non-linearity, input-output mapping, adaptivity, generalization, and robustness [Kasabov, 1996; Haykin, 1999]. There are many types of NN architectures, but Feedforward NNs (FNNs), also known as Multilayer Perceptrons (MLPs), are the most popular and successful NN. They consist of one input layer, one or multiple hidden (intermediate) layers, and one output layer. In FNN, during prediction, the information propagates in only one direction (forward), from the input layer to the output layer. Single-hidden Layer Feedforward Networks (SLFNs) are FNN with one hidden layer, and they are the simplest and mostly used FNN because of the good approximation capabilities in many problems. Researches have proved that MLPs are universal approximators [Hornik *et al.*, 1989]. Cybenko [1989] demonstrated that any continuous function can be uniformly approximated

[1]For a non-zero noise level, y in the BVD would be replaced by its expected value $E_D\{y\}$, and a constant (irreducible) variance term σ^2 would be added, representing the variance of the noise [Brown *et al.*, 2005b].

by a continuous NN having a single hidden layer with neurons that use an arbitrary continuous sigmoidal non-linear activation function.

Researches have compared and proven the effectiveness of MLPs over other learning algorithms. Support Vector Machines (SVMs) usually have greater generalization ability than MLPs. SVMs are based on the structural risk minimization principle, and use quadratic programming, which has the global optimal solution. While, MLPs may have problems of convergence to local minima. On the other hand, SVM models require long training time, and the number of parameters increases as the number of training samples increases, while MLPs have fixed structure size as the number of training samples increases [Antón *et al.*, 2013; Abe, 2010]. Other advantage of MLPs over SVMs is that MLPs can have more than one output, while SVMs have only one output. Studies have demonstrated that an Extreme Learning Machine (a type of NN model that uses a SLFN architecture, i.e. only one hidden layer) tends to be faster and have better generalization performance than other NN learning models, SVMs and Gaussian process [Miche *et al.*, 2010]. Below, the SLFN architecture is described.

Single-hidden Layer Feedforward Network Architecture. The architecture of a SLFN includes an input layer with r input nodes, a hidden layer with L hidden nodes, and an output layer with an output node, as shown in Figure 3.1. Hidden

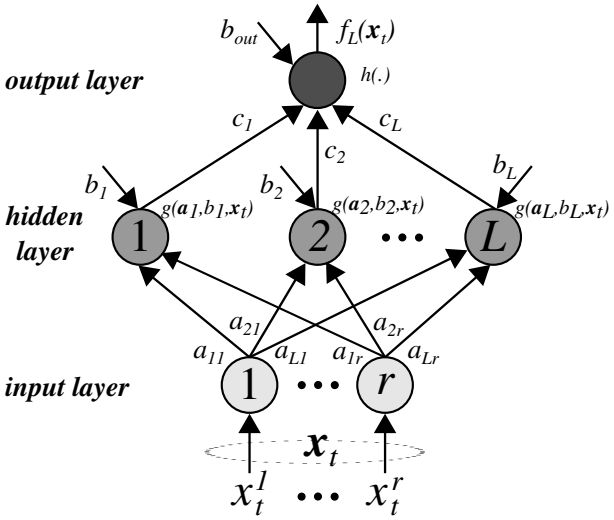


Figure 3.1: A scheme of the SLFN architecture.

nodes and output nodes have activation functions defined as $g(x)$ and $h(x)$, respectively. For example, a *sigmoid* activation function is often used in hidden nodes, $g(x) = 1/(1 + \exp(-x))$, and a *linear* activation function is often used in output nodes, $h(x) = x$. An activation function performs a transformation as the last step to obtain the node's output value. Consider a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$ with T samples (where $\mathbf{x}_t = [x_t^1, \ldots, x_t^r]^T \in \mathbb{R}^r$, and $y_t \in \mathbb{R}$), a standard SLFN model f with L hidden nodes is mathematically represented as:

$$f(\mathbf{x}_t) = h\left(\left(\sum_{j=1}^{L} g\left(\sum_{k=1}^{r} a_{jk}x_t^k + b_j\right)c_j\right) + b_{out}\right) = o_t, \text{ for } t = 1, \ldots, T, \tag{3.2}$$

where $\mathbf{a}_j = [a_{j1}, a_{j2}, \ldots, a_{jr}]^T$ is the vector of weights connecting the r input nodes and the j-th hidden node (for $j = 1, \ldots, L$); $\mathbf{c} = [c_1, c_2, \ldots, c_L]^T$ is the weights vector connecting the L hidden nodes and the output node; b_j is the bias of the j-th hidden node; b_{out} is the bias of the output node; and o_t is the predicted output. Consider that a hidden node j with activation function $g(x)$ can be mathematically represented by $g(\mathbf{a}_j, b_j, \mathbf{x}_t)$, that is:

$$g(\mathbf{a}_j, b_j, \mathbf{x}_t) = g\left(\sum_{k=1}^{r} a_{jk}x_t^k + b_j\right), \text{ for } t = 1, \ldots, T, \tag{3.3}$$

then Equation (3.2) can be rewritten in a simple form as:

$$f(\mathbf{x}_t) = h\left(\left(\sum_{j=1}^{L} g(\mathbf{a}_j, b_j, \mathbf{x}_t)c_j\right) + b_{out}\right) = o_t, \text{ for } t = 1, \ldots, T. \tag{3.4}$$

The network parameters (e.g. synaptic weights and biases) of a SLFN should be adjusted using a learning algorithm in order to reduce the predictive error in all the T samples.

Many learning algorithms have been proposed for this task. The most popular is the Back-Propagation (BP) algorithm, which employs a gradient descent method to tune the NN parameters [Rumelhart *et al.*, 1986]. However, the BP algorithm is very time-consuming and it may result in overfitting. To overcome these limitations, other algorithms were proposed, such as the Levenberg-Marquardt Back-Propagation (LMBP) algorithm [Hagan *et al.*, 1996]. The LMBP achieves good results for non-linear problems.

NN learning algorithms that randomly assign NN parameters have shown good generalization, faster training time and lower computational cost when compared to other algorithms. Schmidt *et al.* [1992] use random hidden synaptic weights (weights connecting input layer and hidden layer) in a SLFN with a sigmoid activation function. However, the universal approximation capability of the proposed solution was not theoretically proved, and the proposed solution is limited to the sigmoid activation function. Pao *et al.* [1994] proposed and proved the universal approximation of a Random Vector Functional-Link (RVFL) approach based on the conventional gradient descent method, in which random hidden synaptic weights can be used in a SLFN with sigmoid or radial basis functions. According to [Huang, 2015], this approach is classified as "semi-random", since hidden node biases are calculated based on the training samples and hidden synaptic weight values. Recently, Huang *et al.* [2006] proposed a SLFN called ELM which has easy parameter tuning, uses random assignment of hidden synaptic weights and biases, and has other advantages, such as fast learning speed, low computational cost, good generalization capability and its universal approximation capability was proved theoretically for a wide variety of types of non-linear piecewise continuous activation functions. The ELM concepts and learning approach can also be used with multiple hidden layer architectures [Huang, 2015]. In this context, the ELM has established itself as an important NN learning architecture. In this thesis (only) the single hidden layer ELM will be considered. For a further discussion about randomness in learning, in NN architectures, and in NN learning, including some works made before ELM, or comparison about NN algorithms that use some form of randomness, references [Huang, 2014, 2015] are suggested.

Below, the ELM and the LMBP learning algorithms for the SLFN architecture are described.

Levenberg-Marquardt Back-Propagation. The LMBP algorithm [Hagan and Menhaj, 1994] was originally proposed to improve the BP algorithm in terms of convergence speed. In the LMBP algorithm, all the network parameters (synaptic weights and biases) are tuned together. Defining $\boldsymbol{\theta}$ as a vector with all the tunable network parameters:

$$\boldsymbol{\theta} = [\mathbf{a}_1; \ldots; \mathbf{a}_L; b_1; \ldots; b_L; \mathbf{c}; b_{out}]^T_{z \times 1}, \tag{3.5}$$

where the number of parameters is $z = r(L + 2) + 1$. At the $(k + 1)$-th algorithm iteration, the values of $\boldsymbol{\theta}$ are updated as:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}_k, \tag{3.6}$$

where

$$\Delta\boldsymbol{\theta}_k = -(\mathbf{J}^T\mathbf{J} + \mu_k\mathbf{I})^{-1}\mathbf{J}^T\mathbf{e}, \tag{3.7}$$

$\mathbf{J}$ is the Jacobian matrix; $\mathbf{I}$ is the identity matrix; μ_k is the learning factor at iteration k; and $\mathbf{e}$ is an error vector of all the samples,

$$\mathbf{e} = \mathbf{y} - \mathbf{o}, \tag{3.8}$$

where $\mathbf{y} = [y_1, \ldots, y_T]^T$ is the real output vector from a training data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^T$; $\mathbf{o} = [o_1, \ldots, o_T]^T$ is the estimated output vector of the NN (with the current parameters $\boldsymbol{\theta}_k$) using $\mathbf{D}$; and $\mathbf{e} = [e_1, \ldots, e_T]^T$. The key step in the LMBP algorithm is the computation of the Jacobian matrix. In [Hagan *et al.*, 1996], this computation is performed by using a variation of the BP algorithm. In this case, to create the Jacobian matrix, it is necessary to obtain the derivatives of the error vector $\mathbf{e}$ with respect to all the network parameters $\boldsymbol{\theta}$ as:

$$\mathbf{J} = \begin{bmatrix} \frac{\partial e_1}{\partial a_{11}} & \cdots & \frac{\partial e_1}{\partial a_{Lr}} & \frac{\partial e_1}{\partial b_1} & \cdots & \frac{\partial e_1}{\partial b_j} & \frac{\partial e_1}{\partial c_1} & \cdots & \frac{\partial e_1}{\partial c_L} & \frac{\partial e_1}{\partial b_{out}} \\ \frac{\partial e_2}{\partial a_{11}} & \cdots & \frac{\partial e_2}{\partial a_{Lr}} & \frac{\partial e_2}{\partial b_1} & \cdots & \frac{\partial e_2}{\partial b_j} & \frac{\partial e_2}{\partial c_1} & \cdots & \frac{\partial e_2}{\partial c_L} & \frac{\partial e_2}{\partial b_{out}} \\ \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \vdots \\ \frac{\partial e_T}{\partial a_{11}} & \cdots & \frac{\partial e_T}{\partial a_{Lr}} & \frac{\partial e_T}{\partial b_1} & \cdots & \frac{\partial e_T}{\partial b_j} & \frac{\partial e_T}{\partial c_1} & \cdots & \frac{\partial e_T}{\partial c_L} & \frac{\partial e_T}{\partial b_{out}} \end{bmatrix}_{T\times z}. \tag{3.9}$$

The LMBP algorithm is summarized in Algorithm 3.1 [Hagan *et al.*, 1996]. In Step 1, the bias and synaptic weights are initialized. In the NN literature, several synaptic weights initialization methods are proposed, such as the uniform random initialization method [Kasabov, 1996], and the Nguyen-Widrow method [Nguyen and Widrow, 1990]. The parameter μ is dynamically tuned using a parameter ϑ. If the SSE value reduces using the new network parameters, then μ is divided by ϑ; otherwise, μ is multiplied by ϑ. When μ is large the algorithm becomes the steepest descent algorithm (with step $1/\mu$), while when μ is small the algorithm becomes the Gauss-Newton method. In [Hagan and Menhaj, 1994], the learning parameters are

Algorithm 3.1 Learning algorithm for LMBP models.

Input: a training data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$; activation functions $g(x)$ and $h(x)$; a number of hidden nodes L; learning parameters μ and ϑ;

1. Initialize synaptic weights $(\mathbf{a}_1, \ldots, \mathbf{a}_L \ ; \ \mathbf{c})$ and biases $(b_1, \ldots, b_L; \ b_{out})$; Set $k = 0$ and $\mu_0 = \mu$; Build vector $\boldsymbol{\theta}_k$ as $\boldsymbol{\theta}$ in Equation (3.5);
2. Present all the inputs to the network and obtain the estimated outputs $\mathbf{o}$; Obtain the error vector $\mathbf{e}$ using Equation (3.8); Obtain the SSE as: $\text{SSE}_k = \sum_{t=1}^{T} (e_t)^2$;
3. Obtain the Jacobian matrix $\mathbf{J}$ with Equation (3.9);
4. Compute $\Delta\boldsymbol{\theta}_k$ with Equation (3.7);
5. Recalculate the SSE $\left(\text{let it be } \text{SSE}_k^{(2)}\right)$ using $\boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}_k$; **if** $\text{SSE}_k^{(2)} < \text{SSE}_k$, **then** let $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \Delta\boldsymbol{\theta}_k$, $\mu_{k+1} = \mu_k/\vartheta$, $k \leftarrow k+1$, and Go to Step 2; **else** let $\mu_k = \mu_k \vartheta$ and Go to Step 4;

set as $\mu = 0.01$ and $\vartheta = 10$.

The LMBP algorithm can be modified to stop when the actual Sum of Squared Errors (SSE) is smaller than a threshold, or using an early stopping criterion [Caruana *et al.*, 2000]. In the case of an early stopping criterion, the SSE of the LMBP model is inspected at a predefined frequency using an independent data set, so that when the SSE increases, the LMBP training is stopped, avoiding overfitting.

Extreme Learning Machine. ELMs were proposed by [Huang *et al.*, 2006]. The input synaptic weights and biases of the SFLN are chosen randomly, and the output synaptic weights are obtained analytically by the Least Squares (LS) method, allowing significant training time reduction when compared to other models. ELMs have demonstrated ability to deal with non-linear problems and exhibit good generalization [Butcher *et al.*, 2013].

Consider a training data set $\mathbf{D}$ with T distinct samples. A standard ELM with $L \leqslant T$ hidden nodes and hidden layer activation function $g(x)$ is mathematically represented as:

$$f(\mathbf{x}_t) = \sum_{j=1}^{L} \beta_j g(\mathbf{a}_j, b_j, \mathbf{x}_t) = o_t, \text{ for } t = 1, \ldots, T, \tag{3.10}$$

where β_j connects the j-th hidden-layer node to the output node [Huang *et al.*,

2006]. It is worth noting that most ELM representations use a linear activation function for the output layer (i.e. $h(x) = x$ in (3.2)), so that for simplicity, $h(x)$ is not used/defined to compute the output prediction $f(\mathbf{x}_t)$. Therefore, in this thesis, only $g(x)$ should be defined for ELM-based models.

If an ELM can approximate the T samples with zero error, then Equation (3.10) can be written as:

$$f(\mathbf{x}_t) = \sum_{j=1}^{L} \beta_j g(\mathbf{a}_j, b_j, \mathbf{x}_t) = y_t, \text{ for } t = 1, \ldots, T. \tag{3.11}$$

The ELM model can be represented as:

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{y}, \tag{3.12}$$

$$\mathbf{H} = \begin{bmatrix} g(\mathbf{a}_1, b_1, \mathbf{x}_1) & \ldots & g(\mathbf{a}_L, b_L, \mathbf{x}_1) \\ \vdots & \ldots & \vdots \\ g(\mathbf{a}_1, b_1, \mathbf{x}_T) & \ldots & g(\mathbf{a}_L, b_L, \mathbf{x}_T) \end{bmatrix}_{T \times L}, \tag{3.13}$$

$$\boldsymbol{\beta} = [\beta_1, \ldots, \beta_L]^T, \qquad \mathbf{y} = [y_1, \ldots, y_T]^T, \tag{3.14}$$

where $\boldsymbol{\beta}$ is the output synaptic weights vector. $\mathbf{H}$ is called the hidden layer output matrix, where the j-th column of $\mathbf{H}$ represents the j-th hidden node output vector with respect to all the input vectors; and the t-th row of $\mathbf{H}$ is the output vector of the hidden layer with respect to $\mathbf{x}_t$.

The learning in ELM is based on finding a solution for vector $\boldsymbol{\beta}$. In most cases, the number of training samples is greater than the number of hidden neurons (i.e. $T > L$); so that $\mathbf{H}$ is a nonsquare matrix and there may not exist a $\boldsymbol{\beta}$ such that $\mathbf{H}\boldsymbol{\beta} = \mathbf{y}$. A solution for $\boldsymbol{\beta}$ can be determined using the LS method as:

$$\hat{\boldsymbol{\beta}} = \mathbf{H}^{\dagger}\mathbf{y}, \tag{3.15}$$

where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse or pseudoinverse [Ben-Israel and Greville, 2003] of matrix $\mathbf{H}$. It will be assumed that $L \leqslant T$. In this condition, if the inverse of $\mathbf{H}^T\mathbf{H}$ exists, then $\mathbf{H}^{\dagger}$ can be obtained as:

$$\mathbf{H}^{\dagger} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T. \tag{3.16}$$

Algorithm 3.2 Learning algorithm for ELM models.

Input: a number of hidden nodes L; a training data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$, where at least L samples are distinct, and $L \leqslant T$; a hidden layer activation function $g(x)$;

1. Randomly assign input synaptic weights $\mathbf{a}_j$ and biases b_j, $j = 1, \ldots, L$;
2. Obtain matrix $\mathbf{H}$ using $\mathbf{D}$ and Equation (3.13);
3. Obtain the output synaptic weight $\boldsymbol{\beta}$ as the values of $\hat{\boldsymbol{\beta}}$ obtained from Equation (3.17);

Substituting Equation (3.16) into Equation (3.15), yields:

$$\hat{\boldsymbol{\beta}} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}. \tag{3.17}$$

Solution $\hat{\boldsymbol{\beta}}$ in (3.17) is the minimum-norm least squares solution of (3.12). It will be assumed that $\mathbf{H}$ is full rank, i.e. rank($\mathbf{H}$) = L. If $\mathbf{H}$ is full rank, then the inverse of $\mathbf{H}^T\mathbf{H}$ in (3.17) exists [Rao and Mitra, 1972]. Theorem II.1 of [Liang *et al.*, 2006] states that if L training samples in $\mathbf{D}$ are distinct, then rank($\mathbf{H}$) = L. The ELM algorithm is summarized in Algorithm 3.2.

3.3.2 Partial Least Squares

PLS has been effectively employed in industrial processes which involve a large number of correlated variables [He *et al.*, 2015; Xu *et al.*, 2014; Liu *et al.*, 2015]. PLS is a linear multivariate regression model that projects the input and output data into a latent space, extracting principal factors with an orthogonal structure and capturing variance in the data.

Consider an input matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T]^T \in \mathbb{R}^{T\times r}$ and an output vector $\mathbf{y} = [y_1, y_2, \ldots, y_T]^T \in \mathbb{R}^{T\times 1}$ from a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$. PLS designs a linear model by decomposing $\mathbf{X}$ and $\mathbf{y}$ into two equations [Qin, 1998]: $\mathbf{X} = \mathbf{W}\mathbf{Z}^T + \mathbf{E}_x$ and $\mathbf{y} = \mathbf{U}\mathbf{q}^T + \mathbf{e}_y$, where $\mathbf{W} \in \mathbb{R}^{T\times\ell}$ and $\mathbf{U} \in \mathbb{R}^{T\times\ell}$ are score matrices that produce ℓ linear combinations/scores (where ℓ is also known as the number of latent variables), with $1 \leqslant \ell \leqslant r$; $\mathbf{Z} \in \mathbb{R}^{r\times\ell}$ and $\mathbf{q} \in \mathbb{R}^{1\times\ell}$ are the loading matrix and loading vector, respectively. $\mathbf{E}_x \in \mathbb{R}^{T\times r}$ and $\mathbf{e}_y \in \mathbb{R}^{T\times 1}$ are input and output data residuals, respectively. PLS designs a regression model by relating the scores of $\mathbf{X}$ and $\mathbf{y}$. The main objective is to build a predictive linear model $\mathbf{y} = \mathbf{X}\mathbf{b} + \mathbf{e}_{reg}$,

where $\mathbf{b} \in \mathbb{R}^{r\times 1}$ is a vector of regression coefficients that are obtained by minimizing $\mathbf{e}_{reg}$.

There are many methods devoted to obtain the PLS vectors and matrices. The most commonly used algorithms are the NIPALS [Geladi and Kowalski, 1986], and the SIMPLS [Jong, 1993]. Both algorithms achieve similar results for problems with only one output y. However, most authors agree that SIMPLS has faster computation with less memory requirements than NIPALS [Martins *et al.*, 2010]. SIMPLS assumes that the variables from $\mathbf{X}$ and $\mathbf{y}$ are scaled to zero mean and unit variance (as proposed in Equation (2.2)), and finds projection directions of $\mathbf{X}$ by obtaining the data cross variance ($\mathbf{X}^T\mathbf{y}$) on an orthogonal subspace. The SIMPLS algorithm for single output problems is summarized in Algorithm 3.3 [Jong, 1993]. The inputs of the algorithm are $\mathbf{X}$, $\mathbf{y}$, and the maximum number of latents ℓ_{max}. The main objective of the algorithm is to obtain a matrix $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_{\ell_{max}}] \in \mathbb{R}^{r\times \ell_{max}}$ that holds vectors of regression coefficients $\mathbf{b}_i$, where each $\mathbf{b}_i$ corresponds to the use of a set of i latent variables ($i = 1, \ldots, \ell_{max}$). In Step 1, matrices and vector $\mathbf{B} \in \mathbb{R}^{r\times \ell_{max}}$, $\mathbf{R} \in \mathbb{R}^{r\times \ell_{max}}$, $\mathbf{W} \in \mathbb{R}^{T\times \ell_{max}}$, $\mathbf{Z} \in \mathbb{R}^{r\times \ell_{max}}$, $\mathbf{q} \in \mathbb{R}^{1\times \ell_{max}}$, $\mathbf{U} \in \mathbb{R}^{T\times \ell_{max}}$, and $\mathbf{V} \in \mathbb{R}^{r\times \ell_{max}}$ are initialized with zero values. The notations $\mathbf{b}_i$, $\mathbf{r}_i$, $\mathbf{w}_i$, $\mathbf{z}_i$, $\mathbf{u}_i$, and $\mathbf{v}_i$ refer to the i-th column vector of $\mathbf{B}$, $\mathbf{R}$, $\mathbf{W}$, $\mathbf{Z}$, $\mathbf{U}$, and $\mathbf{V}$, respectively; and q_i refers to the i-th column element of $\mathbf{q}$. At each iteration i of the algorithm, a new vector of regression coefficients $\mathbf{b}_i$ is obtained. The final matrix $\mathbf{B}$ is useful to determine the prediction error of the model with respect to all the possible numbers of latent variables. Thus, this information can be employed to determine the optimal number of latent variables. Consider an input matrix $\mathbf{X}^{val}$ and an output vector $\mathbf{y}^{val}$ from a validation data set $\mathbf{D}^{val} = \{(\mathbf{x}_t, y_t)\}_{t=T+1}^{T_1}$; and the SSE as the metric for the prediction error. The error of the model for the case of using i latent variables is obtained as: $\text{SSE}_i = \sum_{t=T+1}^{T_1}(y_t - o_t)^2$, where $o_t = \mathbf{x}_t\mathbf{b}_i$.

3.4 Ensemble Learning Models

Ensemble learning models are sets of models that combine in some way their decisions, or their learning algorithms, or different data to achieve accurate predictions. An ensemble learning model is usually more accurate than any single model used separately, and the effectiveness of ensemble systems (ESs) has been shown in different benchmark data sets. Nowadays, ensemble learning represents one of the main

Algorithm 3.3 Learning algorithm for PLS using the SIMPLS method.

Input: an input matrix $\mathbf{X} \in \mathbb{R}^{T \times r}$; an output vector $\mathbf{y} \in \mathbb{R}^{T \times 1}$; and the maximum number of latent variables ℓ_{max} (with $1 \leqslant \ell_{max} \leqslant r$);

1. Initialize with zero values the matrices and vector: $\mathbf{B}$, $\mathbf{R}$, $\mathbf{W}$, $\mathbf{Z}$, $\mathbf{q}$, $\mathbf{U}$, and $\mathbf{V}$;
2. Calculate $\mathbf{s} = \mathbf{X}^T\mathbf{y}$;
3. **for** $i = 1, \ldots, \ell_{max}$
   (a) Obtain: $q_i = 1$, $\mathbf{r}_i = \mathbf{s}q_i$; $\mathbf{w}_i = \mathbf{X}\mathbf{r}_i$; and $\mathbf{w}_i \leftarrow \mathbf{w}_i - \bar{\mathbf{w}}_i$ (where $\bar{\mathbf{w}}_i$ is the mean value of $\mathbf{w}_i$);
   (b) Compute the norm of $\mathbf{w}_i$: $\|\mathbf{w}_i\| = \sqrt{\mathbf{w}_i^T\mathbf{w}_i}$;
   (c) Calculate: $\mathbf{w}_i \leftarrow \mathbf{w}_i/\|\mathbf{w}_i\|$; $\mathbf{r}_i \leftarrow \mathbf{r}_i/\|\mathbf{w}_i\|$; $\mathbf{z}_i = \mathbf{X}^T\mathbf{w}_i$; $q_i = \mathbf{y}^T\mathbf{w}_i$; $\mathbf{u}_i = \mathbf{y}q_i$; $\mathbf{v}_i = \mathbf{z}_i$;
   (d) **if** $i > 1$; **then** $\mathbf{v}_i \leftarrow \mathbf{v}_i - \mathbf{V}(\mathbf{V}^T\mathbf{z}_i)$; **else** $\mathbf{u}_i \leftarrow \mathbf{u}_i - \mathbf{T}(\mathbf{T}^T\mathbf{u}_i)$;
   (e) Calculate: $\mathbf{v}_i \leftarrow \mathbf{v}_i/\|\mathbf{v}_i\| \left(\text{where } \|\mathbf{v}_i\| = \sqrt{\mathbf{v}_i^T\mathbf{v}_i}\right)$; and $\mathbf{s} \leftarrow \mathbf{s} - \mathbf{v}_i(\mathbf{v}_i^T\mathbf{s})$;
   (f) Obtain the regression coefficient vector of the i-th latent: $\mathbf{b}_i = \mathbf{r}_1 q_1 + \ldots + \mathbf{r}_i q_i$;
   (g) Store $\mathbf{b}_i$, $\mathbf{r}_i$, $\mathbf{w}_i$, $\mathbf{z}_i$, q_i, $\mathbf{u}_i$, and $\mathbf{v}_i$ into column i of $\mathbf{B}$, $\mathbf{R}$, $\mathbf{W}$, $\mathbf{Z}$, $\mathbf{q}$, $\mathbf{U}$, and $\mathbf{V}$, respectively;
4. **end for**

Output: matrix of regression coefficients $\mathbf{B}$;

research lines in ML. The main motivations are the possibility of improving the generalization capability and the overall system performance. Despite the remarkable performance of ESs, authors have demonstrated that the ensemble performance depends on several factors, such as the diversity between the models and the combination strategy. Clearly, an adequate strategy should be used to train each ensemble member-model. However, on ESs other key factors exist as will be discussed below.

3.4.1 Theoretical Analyzes of Ensemble Learning Models

Theoretical analysis for ensemble learning algorithms is a key to understand how they work, and prove their efficiency over single learning algorithms. There is no unified theory for these studies, however several theoretical analyzes proved the effectiveness of ESs. This Subsection outlines two main theoretical analyzes for

ESs.

Notation. Consider an ensemble of N regression models and a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$, and consider that the output of the n-th model on the t-th data point is given by $f_n(\mathbf{x}_t)$, and that the ensemble output is given by:

$$F(\mathbf{x}_t) = \sum_{n=1}^{N} w_n f_n(\mathbf{x}_t), \tag{3.18}$$

where w_n is the combination weight of the n-th model, and reflects the contribution of this model to the final ensemble output, and the combination must satisfy the following constraints: $0 \leqslant w_n \leqslant 1$, $\sum_{n=1}^{N} w_n = 1$, and $N > 1$.

The Ambiguity Decomposition. Krogh and Vedelsby [1995] proved that for each data point $(\mathbf{x}_t, y_t)$, the squared error of the ensemble is less than or equal to the average squared error of the ensemble members:

$$(F(\mathbf{x}_t) - y_t)^2 = \sum_{n=1}^{N} w_n \left(f_n(\mathbf{x}_t) - y_t\right)^2 - \sum_{n=1}^{N} w_n \left(f_n(\mathbf{x}_t) - F(\mathbf{x}_t)\right)^2, \tag{3.19}$$

where $F(\mathbf{x}_t)$ is a *convex* combination ($\sum_{n=1}^{N} w_n = 1$) as given in Equation (3.18). This decomposition is known as *ambiguity decomposition* and it is based on the BVD analysis. Details of the proof can be found [Krogh and Vedelsby, 1995; Brown *et al.*, 2005a]. The decomposition is divided into two terms. The first term, $\sum_{n=1}^{N} w_n \left(f_n(\mathbf{x}_t) - y_t\right)^2$, is the weighted average error of the models; while the second term, $\sum_{n=1}^{N} w_n \left(f_n(\mathbf{x}_t) - F(\mathbf{x}_t)\right)^2$, is the *ambiguity term*, and it measures the amount of variability among the models for this data point. Since, the ambiguity term is positive and subtractive from the first term, the ensemble error, $(F(\mathbf{x}_t) - y_t)^2$, is guaranteed to be less than or equal to the weighted average error of the models. The larger the ambiguity term, the larger the ensemble error reduction. However, an increase of the ambiguity term, may also produce an increase of the weighted average error of the models as well. This reveals that diversity itself is not enough, and the best ensemble error reduction can be obtained by a right balance between accuracy and diversity. Unlike the BVD method, the ambiguity decomposition does not take into account the expected error of the ensemble on future data points [Brown

et al., 2005a].

Bias, Variance, and Covariance Decomposition. The BVD method for single learning models (detailed in Section 3.2) can be employed for ensemble learning algorithms, so that Equation (3.1) becomes:

$$E_D\{(F(\mathbf{x}) - y)^2\} = \underbrace{(E_D\{F(\mathbf{x})\} - y)^2}_{(\text{bias})^2} + \underbrace{E_D\{(F(\mathbf{x}) - E_D\{F(\mathbf{x})\})^2\}}_{\text{variance}}, \tag{3.20}$$

where the ensemble combination is given by $F(\mathbf{x}_t) = \frac{1}{N}\sum_{n=1}^{N} f_n(\mathbf{x}_t)$. The BVD method can be reformulated to the *Bias, Variance, and Covariance Decomposition* (BVCD) which includes the correlation among the ensemble members [Brown *et al.*, 2005a,b]. The BVCD method decomposes the ensemble error into tree terms:

$$E\{(F(\mathbf{x}) - y)^2\} = \overline{\text{bias}}^2 + \frac{1}{N}\overline{\text{variance}} + \left(1 - \frac{1}{N}\right)\overline{\text{covariance}},$$

where

$$\overline{\text{bias}}^2 = \left(\frac{1}{N}\sum_{n=1}^{N}(E\{f_n(\mathbf{x})\} - y)\right)^2, \; \overline{\text{variance}} = \frac{1}{N}\sum_{n=1}^{N} E\{(f_n(\mathbf{x}) - E\{f_n(\mathbf{x})\})^2\},$$

$$\overline{\text{covariance}} = \frac{1}{N(N-1)}\sum_{n=1}^{N}\sum_{\substack{k=1 \\ k\neq n}}^{N} E\{(f_n(\mathbf{x}) - E\{f_n(\mathbf{x})\})(f_k(\mathbf{x}) - E\{f_k(\mathbf{x})\})\},$$

and $E\{\cdot\}$ is the expectation operator with respect to different training data sets. The first term, $\overline{\text{bias}}^2$, is the averaged bias of the ensemble members; the second term, $\overline{\text{variance}}$, is the averaged variance of the ensemble members; and the third term, $\overline{\text{covariance}}$ is the averaged covariance of the ensemble members. BVCD shows that the generalization error of the ensemble not only depends on the bias and variance, but also on the covariance between the ensemble members. The bias and variance terms are constrained to be positive-valued, while the covariance term can be negative. The main objective in ES is to decrease the covariance, without increasing the bias and variance terms. According to [Chandra *et al.*, 2006] the covariance term also indicates the *diversity* between the ensemble members. Since it is believed that the more the diversity between the ensemble members, the less

correlated they would be, which implies a reduced error of the ensemble. This is the main reason why diversity is important in ES.

3.4.2 Key Factors in Ensemble Learning Models

The key factor of ensemble systems is to design an ensemble which performs better than random individual ensemble members, and to design members which make different errors on the same sample [Chandra *et al.*, 2006]. That is, *diversity* is important in the ensemble members' decisions. If the models provide the same output, there is nothing to be gained from their aggregation. However, as described in the previous subsection, diversity itself is not enough. An optimal ensemble is the one which achieves a right balance between accuracy and diversity.

During ensemble development, there are several ways to promote diversity. Strategies to promote diversity in ensemble systems are divided into *explicit* and *implicit* diversity methods; while implicit methods rely on randomness to generate diversity, explicit methods deterministically generate diversity. For example, Bagging employs an *implicit* strategy to achieve diversity [Coelho and Nascimento, 2010; Jia and Culver, 2006]. Bagging randomly samples the training data set by applying bootstrap to create a different training data set for each predictor [Breiman, 1996]; at no point a measurement is taken to promote diversity. On the other hand, Boosting is an *explicit* strategy. Boosting directly manipulates the training data set distributions by using specific sample weights to ensure some form of diversity in the set of models [Cristinacce and Cootes, 2007]. The main drawback of this method is that there is no guarantee that it is the *right* way to promote diversity.

Brown *et al.* [2005a] state that the majority of ensemble diversity approaches can be subdivided into three main categories: (*i*) starting the learning with different conditions; (*ii*) altering the set of predictors; (*iii*) altering the trajectory used by the component models in the search space. The first category (*i*) creates each predictor with different initial components. For an ensemble of NN models, training each NN with a different synaptic weights initialization technique may increase the probability of continuing on a different learning trajectory with respect to the other NN models. Approaches in this category usually give poor results, because the predictors are not diverse enough [Castro and Zuben, 2011]. Methods in category (*ii*) aim to modify each ensemble member. Common strategies attempt to manipulate

the training data set that each member receives (e.g., *k-fold cross-validation* [Ries *et al.*, 2007], Bagging, Boosting, or noise injection [Fortuna *et al.*, 2009]), or to alter the model's architecture (e.g., NN models with different architectures or different activation functions), or to design members with heterogeneous learning algorithms [Coelho and Nascimento, 2010]. Approaches in the third category (*iii*) aim to modify the way the search space is traversed, leading different component models to converge to different hypotheses. This category can be subdivided into evolutionary methods and penalty methods. Penalty methods introduce a correlation penalty term into the cost function of the ensemble system so that each model minimizes its error together with the error correlation within the ensemble. On the other hand, evolutionary algorithms can also evolve a population of models using techniques to promote diversity. Penalty methods and evolutionary algorithms can be hybridized. A penalty term can be employed to promote interaction and diversity among the ensemble members and evolutionary algorithms can be used to select the ensemble members [Liu *et al.*, 2000].

Member selection is also a key factor for ensemble development [Soares *et al.*, 2012]. This strategy can lead to better generalization performance. One motivation is that during this process a subset of uncorrelated models (or diverse models) can be selected, promoting the diversity in the ensemble. Several strategies have been employed to select the members for the ensemble, including Genetic Algorithms [Zhou *et al.*, 2002], Particle Swarm Optimization [Yu-Bo and Zhi-Bin, 2011], Bayesian Artificial Immune System [Dondeti *et al.*, 2005], and pruning strategies [Martínez-Muñoz *et al.*, 2009].

During ensemble development some issues are at stake [Re and Valentini, 2012]: how to generate the ensemble members (diversity should be promoted here), how to evaluate the ensemble members, and what member selection method should be employed. Another important issue is what combination strategy should be applied to aggregate the models' outputs. The most common combination strategy is *mean*, where the ensemble's output is obtained as the average of all the models' outputs [Oza and Russell, 2001; Chu and Zaniolo, 2004; Lan *et al.*, 2009]. Other combination strategy is the *weighted mean*, where the ensemble's output is obtained as the weighted average of all the models' outputs. In this case, the combination weight of each model is usually assigned based on the model's prediction accuracy [Dondeti *et al.*, 2005]. The combination strategy is crucial for enhancing the ensemble perfor-

mance [Torres-Sospedra *et al.*, 2005] and balancing the diversity among the ensemble members. An important drawback of most ensemble systems is that usually they consider only one combination strategy during ensemble development.

3.5 Adaptive Learning Systems

In real-world systems, learning algorithms operate in dynamic environments where not only the data but also the data characteristics are evolving continuously. In this case, the target concept to be learned may change over time. This problem is known as *concept drift* in ML [Klinkenberg, 2005]. Learning algorithms must be able to handle dynamic environments and adapt accordingly. Learning algorithms able to automatically adjust their parameters and/or their structures during the on-line phase are known as adaptive learning algorithms. They differ from "off-line learning" algorithms which cannot incorporate new knowledge into the model. When new samples are available, an off-line learned model is usually discarded, and other model is trained with all the stored data. On the other hand, adaptive learning algorithms are equipped with mechanisms which allow them to incorporate/learn new data on-line, and/or receive feedback information about their performances based on incoming data [Kadlec *et al.*, 2011]. In these cases, an on-line learning capability is required.

An ideal on-line learning algorithm should have the following properties [Polikar *et al.*, 2002; Gabrys, 2005]: learn new information from new data; do not require access to previously used data; retain previously acquired important knowledge when learning new knowledge; learn fast from large amounts of data; improve its performance by interacting with other systems; add, retrieve, and remove information/data. This last ability is usually associated to the *stability/plasticity dilemma*, where *stability* describes the ability of the learner of retaining existing important knowledge and remaining stable to irrelevant data (e.g. outlier); while *plasticity* refers to the ability of learning new knowledge [Polikar *et al.*, 2002].

An on-line learning algorithm incrementally processes each new sample or set of samples which is/are continuously arriving. In this scenario, a sequence of steps should be defined to process the new samples. This thesis considers the following steps: (1) the algorithm receives an input (or a set of inputs); (2) the algorithm predicts the output of the sample (or the outputs for a set of samples); (3) the

system reveals the real output value(s) to the algorithm; and (4) the algorithm updates its structure or parameters based on the new sample(s).

It is worth noting that in real-world SS applications, input samples are furnished by the system with low sampling intervals/high frequencies; while output samples are furnished by the system with large sampling intervals/low frequencies. Therefore, steps from (1) to (4) may not be performed sequentially. In some cases (sampling intervals), for a given sample, step (3) and step (4) are not performed, since the real output is not furnished by the system. In these cases, the SS is applied only for on-line prediction, and the on-line learning algorithm of the SS is not updated (i.e. step (4) is not performed). In other cases, for a given sample, step (3) and step (4) are performed, since the real output is furnished by the system. In these cases, the on-line learning algorithm of the SS is updated (i.e. step (4) is performed). Therefore, even if the real outputs are furnished with large sampling intervals, the SS keeps performing on-line predictions.

Below, background of the concept drift problem is given. Moreover, an overview of approaches for dealing with concept drift is given.

3.5.1 The Concept Drift Problem

Recently, the concept drift problem has gained much attention from the ML community. Concept drift happens when the context represented by data changes over time [Klinkenberg, 2005]. For analyzing this problem, consider an on-line learning algorithm where each sample $\mathbf{d} = (\mathbf{x}, y)$ arrives incrementally one by one. Consider a sliding window strategy [Liu *et al.*, 2012a], where a window of fixed size m slides along the data, keeping the most recent m samples:

$$\underbrace{\mathbf{d}_{(1,1)}, \ldots, \mathbf{d}_{(1,m)}}_{\text{window } 1}, \underbrace{\mathbf{d}_{(2,1)}, \ldots, \mathbf{d}_{(2,m)}}_{\text{window } 2}, \ldots, \underbrace{\mathbf{d}_{(t,1)}, \ldots, \mathbf{d}_{(t,m)}}_{\text{window } t}, \underbrace{\mathbf{d}_{(t+1,1)}, \ldots, \mathbf{d}_{(t+1,m)}}_{\text{window } t+1},$$

where $\mathbf{d}_{(k,i)}$ is the i-th sample of window k. For each window k, the data is assumed to follow a distribution $\mathcal{D}_k(\mathbf{x}, y)$. If all the windows are distributed over the same distribution, the concept is considered stable and thus there is no concept drift. Otherwise, if two windows p and q have different data distributions, i.e. $\mathcal{D}_p(\mathbf{x}, y) \neq \mathcal{D}_q(\mathbf{x}, y)$, then there is a concept drift. Learning algorithms to handle the concept drift problem should be able to predict the next data window (e.g. $t + 1$) using the

old data windows (from 1 to t) or a subset of them.

Changes may occur in different forms. In the literature, drifts are classified with respect to their speed, cyclical nature, scope, etc [Minku *et al.*, 2010; Elwell and Polikar, 2011; Zliobaite, 2009]. The drift speed describes the rate at which old concepts are substituted by new concepts. An *abrupt drift* happens when an old concept is abruptly replaced by a new concept; while a *gradual drift* happens when an old concept is slowly substituted by a new concept. Gradual drifts are harder to identify since they typically result in lower rate of change of error, and lower prediction of error, when compared to abrupt drifts.

Drifts can also be classified according to their cyclical nature. A *recurring drift* happens if a previously occurring concept recurs after some time; while a *non-recurring drift* happens if a previously occurring concept cannot recur over time. Recurring drifts may occur due to the cyclic nature of a system (e.g. due to the seasons of the year). Other drift classification is with respect to scope. A *local drift* affects only some regions of the input space; while a *global drift* affects the whole input space. In local drifts, changes depend on the location in the input space. A learning algorithm should detect such changes and adapt only those locations of the model that cover the influenced regions of the input space [Ikonomovska, 2012].

3.5.2 Approaches for Handling Concept Drift

Algorithms to deal with concept drift can be classified as *explicit* or *implicit*. Explicit algorithms employ a drift detection strategy to detect the starting time and severity of a drift. The Early Drift Detection Method (EDDM) is an example of an explicit algorithm to deal with changes [Baena-García *et al.*, 2006]. EDDM measures the distance (interval of time) between two classification errors. It considers that, if the distance increases, then the system is improving its predictions. Otherwise, if the distance decreases, EDDM assumes that the system is learning a new concept, and so a drift is detected. In this case, the system is reset and a new model is trained using a recent set of samples stored since an early drift warning instant is detected. Other examples of drift detection approaches are the *t-test* [Kadlec and Gabrys, 2011], and the *Page-Hinkley test* [Ikonomovska, 2012].

Implicit algorithms do not perform techniques to detect the starting time of a drift. They constantly learn from the environment, adjusting and constructing the

knowledge without explicitly detecting drifts. The main approaches are *instance selection*, *instance weighting*, and *ensemble learning* [Tsymbal, 2004]. In instance selection, a set of relevant samples of the actual concept are selected to build or adapt the model. A common technique is the Sliding Window (SW), also known as moving window. The window can slide on a sample basis (i.e. at each sample) or on a batch basis (i.e. at each set/batch of samples). An important issue in a SW approach is the selection of the window's size. Small windows can provide faster adaptivity, but in more stable phases they can affect the model's performance; while large windows can be more stable but they cannot react faster to the changes. To overcome these issues, adaptive window's size can be employed [Bifet and Gavaldà, 2007].

In instance weighting, samples are weighted according to their age and/or relevance to the current concept, e.g. recursive methods. Recursive methods usually involve down-weighting of the old samples' contribution using a forgetting factor, λ ($0 < \lambda \leqslant 1$) [Qin, 1998]. The forgetting factor indicates the strength of the adaptation. Its value should be flexible so that adaptation can overcome faster and slower changes. Specifically, the forgetting factor works as follows: when λ is close to 0 higher weights are given to the new samples and lower weights are given to the old samples; when λ is close to 1 lower weights are given to the new samples and higher weights are given to the old samples; and when λ is 1, the RLS model is assumed to have "infinite memory". In [Gjerkes *et al.*, 2011], it is proposed a Recursive Least Squares (RLS) algorithm, where the value of λ is adjusted according to the model's prediction error. If the error is small, it is assumed that the estimation is correct and that the process is not changing, so that λ is increased; while if the error is large, then λ is decreased to allow a quick adaptation of the model.

Approaches for Handling Concept Drift in Ensemble Methods. This thesis focuses on ensemble learning algorithms. Table 3.1 lists the main on-line ensembles existing on the literature. An ensemble to deal with concept drift can have the following characteristics: (*i*) adapt the models' combination weights; (*ii*) adapt the models' parameters; and/or (*iii*) add new models or exclude models [Polikar, 2012]. Additionally, other approaches (*iv*) recreate a new ensemble from scratch when a drift is detected [Minku and Yao, 2012; Chu and Zaniolo, 2004].

The removal of models can be performed using an *ensemble pruning* strategy.

Table 3.1: The main on-line ensembles to deal with changing environments.

Approach	Scope	Types of drifts	Drift mechanism	Ensemble learn. mech.	Time step
ACE	classification	mainly recur. drifts	explicit/implicit	(*i*), (*iii*)	sample basis
AddExp	classification/ regression	all the types	explicit/implicit	(*i*), (*ii*), (*iii*)	sample/ batch bases
FLB	classification	gradual and abrupt drifts	explicit/implicit	(*iii*), (*iv*)	batch basis
IBoost	classification	all the types	implicit	(*i*), (*iii*)	sample basis
ILLSA	regression	*	explicit/implicit	(*i*), (*ii*)	sample basis
Learn^{++}.NSE	classification	all the types	implicit	(*i*), (*iii*)	batch basis
OAUE	classification	all the types	implicit	(*i*), (*ii*), (*iii*)	sample basis
OB	classification	*	implicit	(*ii*)	sample basis

* No reference about the types of drifts which the approach can deal with.

In ensemble pruning, a subset of relevant models from the original set of models is selected, and those models that do not contribute to the ensemble's performance are removed. The exclusion of a model from the ensemble can occur when the number of models exceeds a threshold [Nishida and Yamauchi, 2007; Elwell and Polikar, 2009; Chu and Zaniolo, 2004; Kolter and Maloof, 2005]; at a fixed frequency; when a model's parameter reaches a value [Grbovic and Vucetic, 2011]; and/or when the memory usage exceeds a threshold [Brzezinski and Stefanowski, 2014]. Other decision to be taken concerns as to which model should be removed from the ensemble. The excluded model can be the oldest model [Elwell and Polikar, 2009; Chu and Zaniolo, 2004; Kolter and Maloof, 2005], or the model with the worst performance [Nishida and Yamauchi, 2007; Elwell and Polikar, 2009; Kolter and Maloof, 2005; Grbovic and Vucetic, 2011]. In changing environments, identifying the best ensemble pruning strategy is not an easy task. In recurring drifts, there is a risk of removing a model that may be important in the future. Therefore, *weakest first* strategy should be preferred over *oldest first* strategy.

A *time step* defines the time interval at which an on-line ensemble is adapted. An on-line ensemble can be classified as *sample-based* or *batch-based*, when it learns on-line and incrementally on a sample basis, or when it learns from a set of samples, respectively.

Batch-based ensembles tend to be more stable, in the sense that even if a batch

contains an outlier, the system may perform well. Examples of batch-based ensembles are the Learn^{++}.NSE [Elwell and Polikar, 2011, 2009], and the Fast and Light Boosting (FLB) [Chu and Zaniolo, 2004]. In both of these works, for learning purposes, when a new batch is available, the ensembles are employed to predict it. Then, each sample from the batch receives a weight proportional to its prediction error and, a weighted training batch is obtained using the samples' weights. The objective is to train a new model using the weighted training batch. However, ensemble prediction can still be performed on a sample basis. In the FLB algorithm, if a change is detected using statistical decision theory, a new ensemble is created from scratch for fast adaptation of the system to the current concept. This approach may lead the system to a poor performance in scenarios where concepts can recur, since models trained on old concepts are removed. FLB obtains the ensemble's output using the average of the models' outputs. On the other hand, Learn^{++}.NSE obtains the ensemble's output using a weighted average of the models' outputs, where each model's combination weight is calculated using a weighted average of its prediction errors on the old and current batches.

On-line Bagging (OB) is a sample-based ensemble inspired by the Bagging algorithm [Oza and Russell, 2001]. Given a training data set $\mathbf{D}$ with T samples, OB creates a set of N base models, each one trained with a different training data set $\mathbf{D}_n^{'}$ $(n = 1, \ldots, N)$ of size T obtained from $\mathbf{D}$ by bootstrap. When T tends to infinity, a data set $\mathbf{D}_n^{'}$ $(n = 1, \ldots, N)$ may contain K copies of a sample from $\mathbf{D}$. OB assumes that the distribution of K tends to a Poisson distribution. Specifically, during the on-line phase of the OB, when a new sample is available, it is presented K times for retraining each base model, where $K \sim Poisson(\phi)$ and $\phi = 1$. OB uses simple average for combining the models' outputs and no ensemble pruning strategy is applied.

Incremental Boosting (IBoost) [Grbovic and Vucetic, 2011] and Incremental Local Learning Soft Sensing Algorithm (ILLSA) [Kadlec and Gabrys, 2011] are ensembles inspired on the SW concept. IBoost is a sample-based ensemble that keeps a SW with the most recent samples. At a fixed frequency, a model may be added to the ensemble if the ensemble's prediction on the newest sample is incorrect. IBoost has faster adaptation when compared to the batch-based ensembles. ILLSA is an ensemble of RPLS models. ILLSA builds a map for each model using a weighted two-dimensional Parzen Window method. The map stores each model's performance

in the input-output space, such that it can be later employed to estimate the local model's performance given an input data and its prediction. Then models' combination weights on the new sample are calculated by a Bayesian framework using the posterior probabilities of the component-models, given the sample and the prediction of the component-models for the sample. ILLSA does not prune dynamically the ensemble. On-line Accuracy Updated Ensemble (OAUE) [Brzezinski and Stefanowski, 2014], and Adaptive Classifiers-Ensemble system (ACE) [Nishida *et al.*, 2005; Nishida and Yamauchi, 2007] are ensembles for classification tasks that employ a hybridization of strategies of sample-based and batch-based ensembles. Namely, the models' parameters are updated whenever a new sample is available. However, a model is added to, or removed from, the ensemble only after accumulating a number of samples.

In the OAUE, the error of each model f_n, MSE_n^t at time t, is estimated by calculating the Mean Square Error (MSE) using the most recent m samples from a data set $\mathbf{D}$. Then, the combination weight of each model f_n is obtained as: $1/(\text{MSE}_n^t + \text{MSE}_*^t + \epsilon)$, where ϵ is a small positive value, and MSE_*^t is a prediction error threshold used as a reference to the combination weighting strategy. MSE_*^t is obtained using the most recent m samples by $\text{MSE}_*^t = \sum_{z=1}^{Z} p(\omega_z)(1 - p(\omega_z))^2$, where $p(\omega_z)$ is the *a priori* probability (or percentage) of a sample belonging to class ω_z, for a Z-class problem. At time instants defined by a fixed period of m samples, a candidate model is established by training it with the most recent m samples. Afterwards, the candidate model is incrementally trained with the next batch of m samples, and then it is finally added to the ensemble and a new candidate model is established by training it with the new batch of the most recent m samples. If the number of models raises above a threshold, the weakest model is substituted.

Additive Expert (AddExp) is the most popular on-line ensemble for regression [Kolter and Maloof, 2005]. It applies a loss bound to measure the models' performances, and combination weights are adapted according to the current losses and a decreasing factor, ϖ (factor for decreasing combination weights), used to decrease a model's combination weight when it predicts incorrectly. The output values must be set in the interval $[0, 1]$. In AddExp, a new model is included when the total ensemble's loss is greater than a factor φ (factor for adding a new model). The new model's combination weight is set according to a factor ψ (factor for new model combination weight). A model can be removed when the number of models is greater

Algorithm 3.4 A generic on-line batch-based learning algorithm using a single model.

Inputs: a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$ divided into M batches $(\mathbf{D}^1, \ldots, \mathbf{D}^M)$, each one of size m; an on-line supervised learner;

1. $f \leftarrow$ Obtain a model trained with $\mathbf{D}^1$; Set $k = 2$;
2. **while** $k \leqslant M$ **do**:
   (a) Obtain the output prediction of f using $\mathbf{D}^k$;
   (b) Incrementally retrain the existing f using $\mathbf{D}^k$;
   (c) $k \leftarrow k + 1$;
3. **end while**

than a threshold. Two pruning strategies are proposed: *oldest first* or *weakest first*. AddExp does not reveal which samples should be taken for training a new model.

3.5.3 Main Structures of On-line Learning Algorithms

Consider a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$, where samples from $\mathbf{D}$ are given incrementally. Three scenarios are considered. The first is a *batch-based* scenario where $\mathbf{D}$ is partitioned into M batches, $\mathbf{D}^1, \ldots, \mathbf{D}^M$, each one of size m, where $T = M \cdot m$; and at each learning time step, k, a batch is provided for learning, as in Algorithm 3.4. The second is a *sample-based* scenario, where each sample $(\mathbf{x}_t, y_t)$ from $\mathbf{D}$ is sequentially and individually provided for learning, as in Algorithm 3.5.

In the third scenario, the SW scenario defined in Algorithm 3.6, when a new sample is available, an old model trained on the old data window is replaced by a new model trained on the current data window. However, differently from the batch approach, in a SW approach the new data window is obtained from the previous window by adding only one sample, the newest sample, and discarding the oldest sample. Algorithm 3.6 usually outperforms Algorithms 3.4 and 3.5, since in the SW approach the model contains only information about the most recent set of samples. However, Algorithm 3.6 is more computationally expensive, since a new model must be trained for a window at each time step (each sample).

As the standard ELM [Huang *et al.*, 2006] (detailed in Subsection 3.3.1) does not have a retraining strategy, it can be tuned to operate with a SW of fixed size m.

Algorithm 3.5 A generic on-line sample-based learning algorithm using a single model.

Inputs: a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$; number of samples for the initial training phase, m; an on-line supervised learner;

1. **Initialization:** Set the training data as $\mathbf{D}^{train} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{m} \subset \mathbf{D}$;
2. $f \leftarrow$ Obtain a model trained with $\mathbf{D}^{train}$; set $t = m + 1$;
3. **while** $t \leqslant T$ **do**:
   (a) Obtain the output prediction of f using $\mathbf{x}_t$;
   (b) Incrementally retrain the existing f using $(\mathbf{x}_t, y_t)$;
   (c) $t \leftarrow t + 1$;
4. **end while**

Algorithm 3.6 A generic on-line SW learning algorithm using a single model.

Inputs: a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$; window's size, m; a supervised learner;

1. **Initialization:** Set $t = m$ and the window as $\mathbf{D}^t = \{(\mathbf{x}_t, y_t)\}_{t=1}^{m} \subset \mathbf{D}$;
2. $f \leftarrow$ Obtain a model trained with $\mathbf{D}^t$; Set $t = m + 1$;
3. **while** $t \leqslant T$ **do**:
   (a) Slide the window: $\mathbf{D}^t = \mathbf{D}^{t-1} + (\mathbf{x}_t, y_t) - (\mathbf{x}_{t-m}, y_{t-m})$;
   (b) Obtain the output prediction of f using $\mathbf{x}_t$;
   (c) Replace f with a new model trained with $\mathbf{D}^t$;
   (d) $t \leftarrow t + 1$;
4. **end while**

The On-line Sequential Extreme Learning Machine (OS-ELM) [Liang *et al.*, 2006] is an on-line ELM model that can learn data on a batch and/or sample bases. On the on-line phase, once a new sample or a new batch is available, it is employed for retraining, and then it can be discarded.

Lan *et al.* [2009] proposed the sample-based EOS-ELM ensemble, an ensemble of OS-ELM models, which can provide better performance and more stability when compared to the original OS-ELM. Initially, EOS-ELM creates a set of models, all trained using the same activation function and number of hidden neurons, and then

Algorithm 3.7 A generic on-line batch-based ensemble learning algorithm.

Input: a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$ divided into M batches $(\mathbf{D}^1, \ldots, \mathbf{D}^M)$, each one of size m; model error measure, $e()$; an on-line supervised learner; maximum number of models, N;

1. **Initialization:** Set the ensemble as $\mathcal{E} \leftarrow \emptyset$; $n = 1$;
2. $f_n \leftarrow$ Obtain a model trained with $\mathbf{D}^n$; Obtain the prediction error of f_n on $\mathbf{D}^n$ using the error function $e()$; Combination weight f_n based on its prediction error; Set $\mathcal{E} \leftarrow \mathcal{E} \cup \{f_n\}$, and $n \leftarrow n + 1$;
3. **while** $n \leqslant M$ **do**:
   (a) Obtain the output prediction of $\mathcal{E}$ based on the models' combination weights;
   (b) Obtain the prediction error of all the models on $\mathbf{D}^n$ using $e()$;
   (c) Weight all the models based on their prediction errors;
   (d) Incrementally retrain all the existing models using $\mathbf{D}^n$;
   (e) Train a new model f_n with $\mathbf{D}^n$; Obtain the prediction error of f_n on $\mathbf{D}^n$ using $e()$; Weight f_n based on its prediction error; Set $\mathcal{E} \leftarrow \mathcal{E} \cup \{f_n\}$;
   (f) **if** $|\mathcal{E}| > N$ **then** Exclude a model from $\mathcal{E}$;
   (g) $n \leftarrow n + 1$;
4. **end while**

the models' outputs are combined by average. In the on-line phase, when a new sample becomes available, EOS-ELM retrains all the models. EOS-ELM has low diversity between the models, since all the models are trained on the same data and they have the same architectural structure. One alternative is the OB which manipulates the training samples so that each model of the ensemble can be retrained on different samples, increasing the diversity degree between the models.

Other algorithms are the on-line batch-based ensemble and the on-line sample-based ensemble using a SW, detailed in Algorithms 3.7 and 3.8, respectively. They depend on the number of models N, and on $e()$, a generic model error function for measuring the accuracy of the ensemble and the accuracies of the individual models on a data. For example, for the OAUE, the models' errors are obtained using the MSE between the predicted and real outputs on the current window. Algorithm 3.8 depends on a factor for adding a new model on the ensemble. Specifically, a new model is included into the ensemble when the ensemble's prediction error on

Algorithm 3.8 A generic on-line sample-based ensemble learning algorithm using SW.

Input: a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$; window's size, m; model error measure, $e()$; an on-line supervised learner; maximum number of models, N; α, factor for adding a new model;

1. **Initialization:** set $t = m$, $\mathbf{D}^t = \{(\mathbf{x}_t, y_t)\}_{t=1}^{m} \subset \mathbf{D}$, $n = 1$, and set the ensemble as $\mathcal{E} \leftarrow \emptyset$;
2. $f_n \leftarrow$ Obtain a model trained with $\mathbf{D}^t$; Obtain the prediction error of f_n on $\mathbf{D}^n$ or $(\mathbf{x}_t, y_t)$ using the error function $e()$; Weight f_n based on its prediction error; Set $\mathcal{E} \leftarrow \mathcal{E} \cup \{f_n\}$, and $t = m + 1$;
3. **while** $t \leqslant T$ **do**:
   (a) Slide the window: $\mathbf{D}^t = \mathbf{D}^{t-1} + (\mathbf{x}_t, y_t) - (\mathbf{x}_{t-m}, y_{t-m})$;
   (b) Obtain the output predictions of $\mathcal{E}$ based on the models' combination weights;
   (c) Obtain the prediction error of all the models on $\mathbf{D}^n$ or $(\mathbf{x}_t, y_t)$ using $e()$;
   (d) Weight all the models based on their prediction errors on $\mathbf{D}^t$ or $(\mathbf{x}_t, y_t)$;
   (e) Incrementally retrain all the existing models using $(\mathbf{x}_t, y_t)$;
   (f) **if** $e(\mathcal{E})$ on $(\mathbf{x}_t, y_t) > \alpha$ **then** Train a new model f_{n+1} with $\mathbf{D}^t$; Obtain the prediction error of f_n on $\mathbf{D}^n$ or $(\mathbf{x}_t, y_t)$ using $e()$; Weight f_n based on its prediction error; Set $\mathcal{E} \leftarrow \mathcal{E} \cup \{f_{n+1}\}$, and $n \leftarrow n + 1$;
   (g) **if** $|\mathcal{E}| > N$ **then** Exclude a model from $\mathcal{E}$;
   (h) $t \leftarrow t + 1$;
4. **end while**

a new sample is greater than a predefined factor. In contrast, in Algorithm 3.7, a new model is added when a new batch is available. In both algorithms, a model is replaced by a new model, if the number of models exceeds N.

Learn^{++}.NSE and FLB use the scheme presented in the Algorithm 3.7. However, they do not apply Step 3(d), i.e. no retraining of models is applied. AddExp employs a scheme similar to Algorithm 3.8, a robust solution when compared to Algorithm 3.7, since Algorithm 3.8 evaluates the models and the ensemble on every new sample. Additionally, Algorithm 3.8 can add new models at a high frequency when compared to the batch-based ensemble, avoiding the on-line ensemble's degradation. OAUE weights models using the function $e()$ on $\mathbf{D}^t$, while AddExp uses the function $e()$ on $(\mathbf{x}_t, y_t)$. OAUE performs Algorithm 3.8 with some modifications, since new models

are added when m new samples are available/accumulated (that is, at a fixed period of m samples), and these new m samples are grouped to form a new batch. A new model is trained by jointly using the samples of both the new batch and the previous batch. Although the described batch-based algorithms are employed to learn and predict a batch at each iteration of the algorithm, these algorithms can be modified so that the prediction can be performed sample by sample.

3.5.4 On-line Single Learning Models

This Subsection describes two on-line single learning models, OS-ELM and RPLS. They are on-line versions of the ELM and PLS/SIMPLS learning algorithms detailed in Subsections 3.3.1, and 3.3.2, respectively.

On-line Sequential ELM (OS-ELM). The OS-ELM model is an on-line ELM that uses concepts of the RLS algorithm [Haykin, 1996]. The OS-ELM learning consists of two phases: the *initialization phase* and the *sequential learning phase* [Liang *et al.*, 2006]. In the initialization phase, an initial training data set, $\mathbf{D}^0 = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T_0}$ from a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$ (with $T_0 < T$), is considered for designing an initial ELM. In the sequential learning phase, on-line samples are employed either one-by-one or in batches/chunks (with fixed or varying size) for on-line incremental retraining of the ELM, where the $(k+1)$-th chunk of the data set is given by:

$$\mathbf{D}^{k+1} = \{(\mathbf{x}_t, y_t)\}_{t=(\sum_{l=0}^{k} T_l)+1}^{t=\sum_{l=0}^{k+1} T_l}, \tag{3.21}$$

where $k \geqslant 0$ and T_{k+1} is the number of samples in the $(k+1)$-th chunk. The initialization phase is similar to the standard ELM learning. The initial output synaptic weights vector $\boldsymbol{\beta}_0$ is determined as:

$$\boldsymbol{\beta}_0 = (\mathbf{H}_0^T \mathbf{H}_0)^{-1} \mathbf{H}_0^T \mathbf{y}_0, \tag{3.22}$$

where $\mathbf{y}_0 = [y_1, \ldots, y_{T_0}]^T$ is the output vector from $\mathbf{D}^0$; and $\mathbf{H}_0$ is the initial hidden layer output matrix obtained with $\mathbf{D}^0$:

$$\mathbf{H}_0 = \begin{bmatrix} g(\mathbf{a}_1, b_1, \mathbf{x}_1) & \ldots & g(\mathbf{a}_L, b_L, \mathbf{x}_1) \\ \vdots & \ldots & \vdots \\ g(\mathbf{a}_1, b_1, \mathbf{x}_{T_0}) & \ldots & g(\mathbf{a}_L, b_L, \mathbf{x}_{T_0}) \end{bmatrix}_{T_0 \times L} . \tag{3.23}$$

Considering $\mathbf{P}_0 = (\mathbf{H}_0^T \mathbf{H}_0)^{-1}$ as an initial covariance matrix, Equation (3.22) can be written as:

$$\boldsymbol{\beta}_0 = \mathbf{P}_0 \mathbf{H}_0^T \mathbf{y}_0. \tag{3.24}$$

Upon the arrival of $(k+1)$-th chunk, the new output synaptic weights vector $\boldsymbol{\beta}_{k+1}$ is computed using concepts of the RLS algorithm as follows:

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k + \mathbf{P}_{k+1} \mathbf{H}_{k+1}^T (\mathbf{y}_{k+1} - \mathbf{H}_{k+1} \boldsymbol{\beta}_k), \tag{3.25}$$

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \mathbf{P}_k \mathbf{H}_{k+1}^T (\mathbf{I} + \mathbf{H}_{k+1} \mathbf{P}_k \mathbf{H}_{k+1}^T)^{-1} \mathbf{H}_{k+1} \mathbf{P}_k \ , \tag{3.26}$$

$$\mathbf{H}_{k+1} = \begin{bmatrix} g(\mathbf{a}_1, b_1, \mathbf{x}_{(\sum_{l=0}^{k} T_l)+1}) \ldots g(\mathbf{a}_L, b_L, \mathbf{x}_{(\sum_{l=0}^{k} T_l)+1}) \\ \vdots \quad \ldots \quad \vdots \\ g(\mathbf{a}_1, b_1, \mathbf{x}_{\sum_{l=0}^{k+1} T_l}) \ldots g(\mathbf{a}_L, b_L, \mathbf{x}_{\sum_{l=0}^{k+1} T_l}) \end{bmatrix}_{T_{k+1} \times L} , \tag{3.27}$$

$$\mathbf{y}_{k+1} = \left[y_{(\sum_{l=0}^{k} T_l)+1}, \ldots, y_{\sum_{l=0}^{k+1} T_l} \right]^T , \tag{3.28}$$

where $\mathbf{P}_k$ is a covariance matrix of the k-th chunk. For detailed derivation of Equations (3.25) and (3.26) the paper [Liang *et al.*, 2006] is suggested. It is assumed that the samples are such that $\text{rank}(\mathbf{H}_0) = L$, so that $\mathbf{H}_0^T \mathbf{H}_0$ is invertible. Theorem II.1 of [Liang *et al.*, 2006] states that if L training samples in $\mathbf{D}^0$ are distinct, then $\text{rank}(\mathbf{H}_0) = L$. When the $(k+1)$-th chunk contains only one sample, Equations (3.25) and (3.26) can be written using the Sherman-Morrison formula[2] as [Maponi, 2007]:

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k + \mathbf{P}_{k+1} \mathbf{h}_{k+1} (y_{k+1} - \mathbf{h}_{k+1}^T \boldsymbol{\beta}_k), \tag{3.29}$$

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \frac{\mathbf{P}_k \mathbf{h}_{k+1} \mathbf{h}_{k+1}^T \mathbf{P}_k}{1 + \mathbf{h}_{k+1}^T \mathbf{P}_k \mathbf{h}_{k+1}}, \tag{3.30}$$

[2] $(\mathbf{S} + uv^T)^{-1} = \mathbf{S}^{-1} - \frac{\mathbf{S}^{-1} uv^T \mathbf{S}^{-1}}{1 + v^T \mathbf{S}^{-1} u}$.

Algorithm 3.9 Learning algorithm for the OS-ELM model.

Input: a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$; a hidden layer activation function $g(x)$; a number of hidden nodes L; number of samples for the initialization phase T_0 (where $L \leqslant T_0 < T$), where at least L samples are distinct;

1. **Initialization/training phase:** Consider a training data set $\mathbf{D}^0 = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T_0}$;
   (a) Randomly assign input synaptic weights $\mathbf{a}_j$ and biases b_j, $j = 1, \ldots, L$;
   (b) Calculate $\mathbf{H}_0$ using $\mathbf{D}^0$ and Equation (3.23);
   (c) Obtain the output synaptic weight $\boldsymbol{\beta}_0$ through Equation (3.24), where $\mathbf{P}_0 = (\mathbf{H}_0^T\mathbf{H}_0)^{-1}$ and $\mathbf{y}_0 = [y_1, \ldots, y_{T_0}]^T$; Set $k = 0$;
2. **Sequential/on-line learning phase:** Present the $(k+1)$-th chunk $\mathbf{D}^{k+1}$ defined in Equation (3.21);
   (a) Obtain matrix $\mathbf{H}_{k+1}$ using $\mathbf{D}^{k+1}$ and Equation (3.27);
   (b) Set $\mathbf{y}_{k+1}$ using Equation (3.28);
   (c) Obtain $\mathbf{P}_{k+1}$ and $\boldsymbol{\beta}_{k+1}$ using Equations (3.26) and (3.25), respectively;
   (d) Set $k \leftarrow k + 1$; Go to Step 2.

where $\mathbf{h}_{k+1} = [g(\mathbf{a}_1, b_1, \mathbf{x}_{(\sum_{l=0}^{k} T_l)+1}), \ldots, g(\mathbf{a}_L, b_L, \mathbf{x}_{(\sum_{l=0}^{k} T_l)+1})]$. The OS-ELM algorithm is summarized in Algorithm 3.9.

Recursive Partial Least Squares (RPLS). RPLS is widely employed in industrial process monitoring and control [He *et al.*, 2015; Ni *et al.*, 2014]. The main idea is to adapt a PLS model in order to capture all the process changes. In [Qin, 1998], a RPLS algorithm with a SW and fixed forgetting factor that controls the strength of the adaptation is proposed. The main disadvantage is that a fixed forgetting factor may not be sufficient to track all the systems' dynamics. Ahmed *et al.* [2009] developed a RPLS model which is implemented by updating recursively the mean and variance data, and the oldest sample is excluded and the newest sample is included into the model simultaneously. This method can be seen as a SW approach, since the model is always trained using a fixed number of the most recent samples. It allows the adaptation to new events and the partial retention of the process history.

Inspired by this approach, this thesis proposes a recursive SIMPLS algorithm, detailed in Algorithm 3.10. When a new sample is available, the new sample is included to, and the oldest sample is removed from, the previous training data set.

Algorithm 3.10 Learning algorithm for the recursive SIMPLS method.

Input: a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$; window's size, m; the maximum number of latent variables ℓ_{max} (with $1 \leqslant \ell_{max} \leqslant r$);

1. **Initialization:** Set $\mathbf{D}^0 \leftarrow \emptyset$;
2. **for** $t = 1, \ldots, T$:
   (a) **if** $(t > m)$
      i. **then** Slide the window: $\mathbf{D}^t = \mathbf{D}^{t-1} + (\mathbf{x}_t, y_t) - (\mathbf{x}_{t-m}, y_{t-m})$;
      ii. **else** Fill the window: $\mathbf{D}^t = \mathbf{D}^{t-1} + (\mathbf{x}_t, y_t)$;
   (b) **if** $(t = m)$ **then** Go to Step 2(d); **if** $(t < m)$ **then** Go to Step 2;
   (c) Obtain an output prediction of model f using $\mathbf{x}_t$;
   (d) Obtain the current input matrix $\mathbf{X}_t$ and output vector $\mathbf{y}_t$ from $\mathbf{D}^t$;
   (e) **if** $(t = m)$
      i. **then** Obtain the mean and SD of the data $\mathbf{D}^t$;
      ii. **else** Update the mean and SD of the data with the new sample $(\mathbf{x}_t, y_t)$ using Equations (2.4) and (2.5);
   (f) Scale $\mathbf{X}_t$ and $\mathbf{y}_t$ to zero mean and unit variance using Equation (2.2);
   (g) $f \leftarrow$ Obtain a SIMPLS model trained with $\mathbf{X}_t$ and $\mathbf{y}_t$ using Algorithm 3.3;
3. **end for**

The mean and SD of the data are obtained for the first time (when $t = m$) in Step 2(e)i; otherwise the mean and SD are recursively updated with the new sample using Equations (2.4) and (2.5) in Step 2(e)ii. The main objective is to scale the input matrix $\mathbf{X}_t$ and output vector $\mathbf{y}_t$ (to zero mean and unit variance) for the SIMPLS learning procedure (Step 2(g)). In Step 2(g), the best number of latent variables of the SIMPLS model can be obtained by SSE and/or k-fold cross-validation [Arlot and Celisse, 2010]. It is worth noting that Step 2(c) involves some sub-steps: $\mathbf{x}_t$ is scaled using the current mean and SD; the scaled $\mathbf{x}_t$ is presented to the SIMPLS model and so a scaled output prediction is given; and the scaled output prediction is rescaled into the original magnitude/scale (unscaled output).

3.6 Conclusion

This Chapter presented important issues related to intelligent learning systems. Theoretical analyzes showed that the ensemble error is guaranteed to be less than or equal to the average error of the ensemble members. It was described that the success of an ensemble system depends on the diversity of the ensemble members. The ambiguity decomposition showed that diversity itself is not enough, because an increase on the diversity may also produce an increase of the models' errors as well. Therefore, the success of an ensemble system depends on the optimal balance between the diversity and accuracy of the ensemble members. In this case, an optimal combination strategy is an important factor for achieving this balance. However, most ensemble systems consider only one combination strategy during ensemble development. In this context, meta-heuristics optimization are valuable tools to select an appropriate set of accurate and diverse models, and the optimal combination strategy in ensemble systems. In this thesis, meta-heuristics based approaches will be proposed for automatic ensemble development/learning.

This Chapter also described that additional strategies are necessary to guarantee the ensemble performance in dynamic environments. In this case, the ensemble should adapt the model's combination weights, adapt the models' parameters, and/or add new models or exclude models. This Chapter showed that some adaptive ensembles only add or exclude models after accumulating a certain number of samples, which makes difficult the system adaptation to abrupt changes. In this context, having ensembles adapted on every new sample is a key element to achieve superior predictive performance in changing environments. Sample-based learning is one of the elements that are explored in the on-line ensemble learning methods proposed in this thesis.

Chapter 4

Automatic Ensemble Development Using Meta-Heuristics

Contents

4.1 Introduction

In the recent decades ensemble learning has established itself as a valuable strategy within the computational intelligence modeling and ML community. Ensemble learning has proven to be effective in a broad set of ML problems, including feature selection, learning in small data sets, local learning, concept drift theory, among others; and the ensemble learning effectiveness has been recognized in different benchmark data sets [Brown *et al.*, 2005b; Liu *et al.*, 2000; Coelho and Nascimento, 2010]. In this context, Neural Network Ensembles (NNEs) have been widely investigated for both classification and regression problems [Lan *et al.*, 2009]. The main motivation is that the generalization ability of a NNE system can be significantly better than the generalization ability of a single NN model.

Section 3.4 described that the key factors of the ensemble system are diversity, combination strategy, accuracy, and ensemble member selection. Research has encouraged, or explored, diversity by *manipulating the training data set*, e.g. using bootstrap [Oza and Russell, 2001] or noise injection [Zhang, 2007]; or by designing ensembles with *different architectures*, e.g. NN models with different numbers of hidden neurons in NNE systems [Fortuna *et al.*, 2009]; or *heterogeneous learning algorithms*, e.g. an ensemble with NN and SVM models [Coelho and Nascimento, 2010]. Other key factor of an ensemble system is the approach used to combine the individual models in the ensemble. The combination strategy is a way for ensuring ensemble accuracy and balancing the diversity between the individual models.

A major drawback in ensemble learning is that it is usually necessary to combine a large number of models to ensure the ensemble accuracy. A good way to alleviate this problem is the adequate selection of the subset of models from the original set of models [Wang and Guo, 2013] by *ensemble pruning* [Martínez-Muñoz *et al.*, 2009]. The aim is to find a good subset of ensemble members in order to improve

generalization ability, which additionally reduces the system complexity. However, ensemble pruning is a difficult problem whose solution is commonly computationally expensive. Pruning an ensemble with N models requires searching in the space of the $2^N - 1$ non-empty solutions to minimize a cost function correlated with the generalization error.

To address this problem, a number of different meta-heuristics have been developed for model selection. An example is the Genetic Algorithm based Selective Ensemble (GASEN), which trains a set of NNs using bootstrap to increase the diversity among the models. GASEN uses a GA to select an optimal subset of NN models to include in the ensemble. In this strategy, a combination weight derived from the marginal improvement in the *fitness* (measuring the solution quality) associated with including a model in the ensemble is assigned to each model. Then, the models whose combination weights are higher than a fixed threshold are selected for inclusion in the ensemble [Zhou *et al.*, 2002]. The main drawback of GASEN is that the NNs have fixed architectures and the combination techniques are only simple average and weighted average for regression and classification, respectively.

Liu *et al.* [2000] present an automatic strategy for designing ensemble systems using Evolutionary Learning and Negative Correlation Learning (EENCL). Negative Correlation Learning (NCL) generates negatively correlated NN models using a correlation penalty term in the error function to encourage specialization and cooperation among the models. EENCL does not explore the linear combination among the models, and the models' architectures are also predefined. On the other hand, Bayesian Artificial Immune System (BAIS) [Castro and Zuben, 2011] is an immune-inspired methodology for designing NNEs with better generalization ability when compared to the EENCL. Artificial Immune Systems is a computational paradigm inspired by the immunological system of vertebrates, where the immunological system's characteristics of learning and memory are exploited to solve a problem. BAIS introduces diversity in the models' architecture. However, only one combination type is used for designing the NNE systems.

This Chapter proposes and compares GA and SA based approaches for the automatic development of NNEs for regression problems. The main contribution of the proposed method is the development of optimization techniques to select the best subset of models to be aggregated taking into account all the key factors of ensemble systems (i.e. diversity, combination strategy, and ensemble member selection).

First, a set of models with a high degree of diversity is generated. For each model, the proposed approach creates a different training data set by applying bootstrap. Then, the method selects the best model's architecture by varying the number of hidden neurons, the activation functions, and the synaptic weights initializations. Finally, the optimization strategy is employed to select both the best subset of models and the optimal combination strategy for aggregating the subset of models. Experiments on five data sets are reported to evaluate the effectiveness of the proposed methodologies. Results show that the proposed methodologies outperform state-of-the-art approaches including Simple Bagging, NCL, AdaBoost [Cristinacce and Cootes, 2007], and GASEN in terms of generalization ability.

This Chapter is organized as follows. Section 4.2 and Section 4.3 describe the proposed methodologies. Specifically, the design of the initial set of NN models and the proposed combination strategies are described in Section 4.2. Section 4.3 details the proposed methodology for designing NNE systems using GA and SA. Experimental results are detailed and analyzed in Section 4.4. Section 4.5 contains concluding remarks.

4.2 Proposed Methodology: Design of NNs and Combinations

In NN modeling, there are several NN structures and NN parameters that need to be carefully chosen. In the NN structure selection, the number of layers, the number of neurons in each layer, and the activation functions are usually chosen. In the NN parameter selection, the synaptic weights initialization method, and the learning rates are usually selected (besides the NN synaptic weights adaptation during the learning process). Techniques have been proposed for the NN parameter selection and the NN structure selection [Matias *et al.*, 2014]. However, even if the resulting NN is correctly designed, the generalization ability can be a problem [Rosin and Fierens, 1995]. Ensemble learning has been established as a very promising approach for improving the generalization of NN systems [Torres-Sospedra *et al.*, 2005; Dondeti *et al.*, 2005; Yu-Bo and Zhi-Bin, 2011]. The next Subsections describe the data set manipulation performed before training NNs, the design/training of the candidate (or initial set of) NNs, and the proposed combination strategies.

4.2.1 Training, Validation, and Testing Data Sets

Consider an initial data set (original data set) $\mathbf{D}_{init} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$ of size T, where $\mathbf{x}_t \in \mathbb{R}^{r\times 1}$ is the input vector, and $y_t \in \mathbb{R}$ is the output variable. The initial data set is divided into a training data set $\mathbf{D}_{train} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T_0}$, a validation data set $\mathbf{D}_{valid} = \{(\mathbf{x}_t, y_t)\}_{t=T_0+1}^{T_0+T_1}$, and a testing data set $\mathbf{D}_{test} = \{(\mathbf{x}_t, y_t)\}_{t=T_0+T_1+1}^{T=T_0+T_1+T_2}$, of sizes T_0, T_1 and T_2, respectively, and $T = T_0 + T_1 + T_2$.

Bootstrap can be applied to the training data set for promoting diversity in the ensemble. In ML, bootstrap is employed to expand upon a single realization of a distribution or generate different data sets that can provide a better understanding of the mean and variability of the original unknown distribution [Jia and Culver, 2006]. Bootstrap is performed by randomly sampling with replacement from the original training data set $\mathbf{D}_{train}$. To sample with replacement, one sample $\{(\mathbf{x}_t, y_t)\}$ from $\mathbf{D}_{train}$ is randomly selected and then placed into a new training data set $\mathbf{D}_{train}^{b}$. $\mathbf{D}_{train}^{b}$ must contain the same number of samples as $\mathbf{D}_{train}$, i.e. T_0 samples. Random sample selections from $\mathbf{D}_{train}$ continues until $\mathbf{D}_{train}^{b}$ has been filled with T_0 samples. The $\mathbf{D}_{train}^{b}$ data set may include multiple copies of the same sample and no copies of other samples from $\mathbf{D}_{train}$.

Let us assume a set of N candidate NN models, i.e. $f_1, \ldots, f_N$. N different training data sets $\mathbf{D}_{train}^{b}$ are obtained from $\mathbf{D}_{train}$ by bootstrap, i.e. $\mathbf{D}_{train}^{1}, \ldots, \mathbf{D}_{train}^{N}$. Each model f_n is associated and trained with a different training data set $\mathbf{D}_{train}^{n}$. $\mathbf{D}_{valid}$ is used to control the overfitting by early stopping [Jeong and Kim, 2005]. The testing data set $\mathbf{D}_{test}$ is employed to evaluate the ensemble's performance in Section 4.3.

4.2.2 Generation of Candidate Neural Networks

After creating N training data sets, $\mathbf{D}_{train}^{1}, \ldots, \mathbf{D}_{train}^{N}$, the next step is to train N candidate NN models. NN models are implemented using SLNF architecture trained by the LMBP algorithm [Hagan and Menhaj, 1994], described in Subsection 3.3.1. The learning parameters are set according to the authors' suggestions: $\mu = 0.01$, and $\vartheta = 10$.

For each training data set $\mathbf{D}_{train}^{n}$, the topology of the model f_n is chosen from a collection of NN models based on its performance. This evaluation is done using the MSE between the estimated output of the NN and the actual output y in the

validation data set $\mathbf{D}_{valid}$. The collection of models is obtained by varying the number of neurons in the hidden layer L (from 1 to 10); varying among two activation functions (linear $\zeta(x) = x$, and fast hyperbolic tangent $\varrho(x) = 1 - 2/(1 + \exp(2x))$) for the hidden layer activation function $g(x)$ and for the output layer activation function $h(x)$; and three different synaptic weights initialization methods. The employed synaptic weights initialization methods are:

- Randomly initialize the synaptic weights within the interval $[-1/r, 1/r]$, where r is the number of input neurons [Kasabov, 1996];
- Nguyen-Widrow approach: set initial synaptic weights using the Nguyen-Widrow initialization method [Nguyen and Widrow, 1990];
- Randomly initialize the synaptic weights within the interval $[-0.5, 0.5]$, using a uniform distribution [Škutová, 2008].

Therefore, at the end of this process, the best NN model for each training data set $\mathbf{D}^n_{train}$ $(n = 1, \ldots, N)$ is obtained, producing a set of N candidate NN models.

4.2.3 Proposed Combination Strategies

This thesis uses the main combination strategies reported in the literature: *mean*, *trimmed mean*, *median* [Polikar, 2006], and *weighted mean* [Hashem, 1994]. In this Subsection, for the sake of simplicity in describing the combination strategies, and without loss of generality, it is assumed that N candidate NN models are used to constitute an ensemble, that $f_n(\mathbf{x}_t)$ is the output of model f_n, and that $F(\mathbf{x}_t)$ is the ensemble's output. If only a strict subset of N^* candidate models (with $1 < N^* < N$) from the original set of N models is combined, then in the combination strategies described below, only such N^* models are used in the calculation of the combination weight w_n of each model f_n of the subset of combined models. The combination strategies (using a sample $(\mathbf{x}_t, y_t)$) are given by:

1. *Mean*: the ensemble's output is calculated by averaging the N models' predictions: $F(\mathbf{x}_t) = \frac{1}{N} \sum_{n=1}^{N} f_n(\mathbf{x}_t)$;
2. *Trimmed mean*: the ensemble's output is obtained as the trimmed mean of the N predictors' outputs. Trimmed mean excludes the lowest predictors'

outputs and the highest models' outputs before obtaining the *mean*, avoiding extreme outputs. For example, considering a $P\%$ trimmed mean, the *mean* is calculated by removing a percentage of $P\%/2$ of the highest NN models' outputs and a percentage of $P\%/2$ of the lowest NN models' outputs. (This thesis sets $P\%$ as 10%).

3. *Median*: the ensemble's output is the median among all the models' outputs: $F(\mathbf{x}_t) = \underset{n=1,\ldots,N}{\text{median}}\{f_n(\mathbf{x}_t)\}$;

4. *Weighted mean*: the ensemble's output is calculated through a weighted sum of the models' outputs: $F(\mathbf{x}_t) = \sum_{n=1}^{N} w_n f_n(\mathbf{x}_t)$; where each combination weight w_n is related to the accuracy of model f_n, and the combination must satisfy the following constraints: $0 \leqslant w_n \leqslant 1$ and $\sum_{n=1}^{N} w_n = 1$. In this Chapter, the combination weight w_n of a model f_n is obtained using its prediction error (MSE_n) in the validation data set $\mathbf{D}_{valid}$. Specifically, MSE_n is the MSE between the real outputs of $\mathbf{D}_{valid}$ and the estimated outputs of f_n using $\mathbf{D}_{valid}$. A combination weight w_n is obtained as [Dondeti *et al.*, 2005]: $w_n =$ adjusted $\text{MSE}_n / \sum_{k=1}^{N}$ adjusted MSE_k, where the "adjusted MSE_n" is obtained as: adjusted $\text{MSE}_n = (1 -$ average $\text{MSE}_n)$; and the average MSE_n is given by: average $\text{MSE}_n = \text{MSE}_n / \sum_{k=1}^{N} \text{MSE}_k$.

4.3 Proposed Methodology: NNE Design by GA and SA

This Section proposes two different methods for automatic ensemble development: Genetic Algorithm for Designing Neural Network Ensembles (GA-NNE, Subsection 4.3.1), and Simulated Annealing for Designing Neural Network Ensembles (SA-NNE, Subsection 4.3.2). The ensemble construction using GA-NNE and SA-NNE is performed by two main steps:

1. Generation of candidate NN models;

2. Selection of both a subset of NN models and the best combination strategy for aggregating this subset.

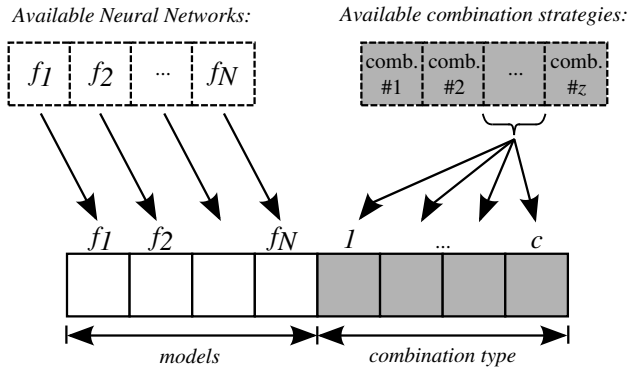


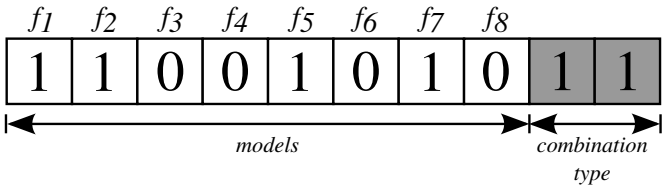
Figure 4.1: Binary solution representation.

The aim is to produce an ensemble which has good performance when compared to an individual NN performance. This objective is achieved by producing and selecting diverse NN models and then selecting the most suitable combination strategy.

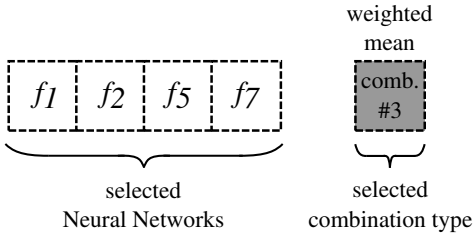
The sub-steps for the generation of candidate NN models are the same as in Subsections 4.2.1 and 4.2.2. For generating N candidate NN models, first N different training data sets are generated by applying bootstrap. Then, the most suitable NN's architecture is chosen for each training data set. At the end of this process N candidate NN models are generated.

For defining the method proposed for the selection of a subset of NN models and a combination strategy using GA-NNE and SA-NNE algorithms, first it is necessary to introduce (i) how the possible solutions are encoded, and (ii) the *fitness* function.

Solution Encoding. A candidate solution to the problem is encoded as a binary string sequence. The solution contains information about the ensemble of NN models to be designed. The solution structure consists of two parts, as illustrated in Figure 4.1. The first part is the *model* section, which contains information about the subset of NN models for composing the ensemble. The second part is the *combination type*, which represents the combination strategy to be employed for aggregating the subset of NN models. As an example, consider a set of N candidate NN models $\{f_1, f_2, \ldots, f_N\}$, where each *locus* of the *model* part is related to the absence "0" or presence "1" of a model f_n in the ensemble system. Consider z as the number of combination strategies, and c as the number of bits to represent them, then $c = \lceil \log_2(z) \rceil$, where $\lceil v \rceil$ is the smallest integer not lower than v. The proposed methodology uses the four combination strategies mentioned in Subsection 4.2.3:



(a) Binary solution representation.



(b) Decoding the solution.

Figure 4.2: Example of a solution.

mean, *trimmed mean*, *median*, and *weighted mean*, with their binary representations being "00", "01", "10", and "11", respectively. Figure 4.2 illustrates an example of a solution representation using a set of eight NN models $\{f_1, f_2, f_3, \ldots, f_8\}$. Figure 4.2(a) shows the binary encoding of the solution, and Figure 4.2(b) shows the decoding of the same solution. The final subset of NN models to compose the ensemble is $\{f_1, f_2, f_5, f_7\}$ and the selected combination strategy is *weighted mean* (i.e., "11").

Fitness Function. For a candidate solution containing a subset of NNs, the *fitness* function is calculated based on the performance of the subset of NNs. This is obtained through the aggregation of the subset of NN models using the selected combination strategy. The subset of NNs is employed to estimate the outputs on the testing data set $\mathbf{D}_{test}$. Then, the estimation error of the subset of NNs is computed using the MSE between the estimated outputs (of the subset of NNs) and the real outputs in the testing data set $\mathbf{D}_{test}$. Here, the notation MSE^{test} is employed to refer to the MSE of this subset of NNs in $\mathbf{D}_{test}$. The *fitness* of a candidate solution containing a subset of NNs is defined by $1/\text{MSE}^{test}$.

4.3.1 Genetic Algorithm for Designing Neural Network Ensemble (GA-NNE)

GAs were proposed by Holland [Holland, 1992] as a global optimization approach inspired by natural evolution and survival of the fittest. GAs use a solution population (*chromosomes*) which evolves by means of *selection*, *crossover*, and *mutation* operators [Sivanandam and Deepa, 2007].

GA-NNE is herein proposed for evolving a population of candidate ensembles [Soares *et al.*, 2013], as shown in Algorithm 4.1. GA-NNE starts by setting the parameters N, c, G_{max}, $p_m^{\%}$, $p_s^{\%}$, $p_c^{\%}$, e, and K. In GA-NNE, each chromosome represents an ensemble to be designed. A chromosome is implemented using a binary solution representation shown in Figure 4.1, and $1/\text{MSE}^{test}$ is the *fitness* function.

In Step 1, a set of N candidate NN models is produced according to Subsection 4.2.2. In Step 2 an initial population P_1 with K individuals is generated, where each individual has length of $(N + c)$ bits.

Step 3 evaluates each individual of P_1 using the *fitness* function. This is done by evaluating the performance of the subset of models (information contained in the *model* part) using all possible combination strategies. Step 4 assigns the combination strategy with the best performance to the chromosome (last c bits, i.e., *combination type* part). This strategy ensures that the ensemble system will always be designed using the optimal combination type. In Step 6, the algorithm loops over G_{max} generations, where the generation number loops over $G = 1, \ldots, G_{max}$. Sub-step 6(a) selects a percentage of $p_s^{\%}$ of the individuals of population P_G by using *Roulette Wheel Selection* [Sivanandam and Deepa, 2007]. In this operation, an individual of P_G is picked to be a parent with a probability proportional to its *fitness*. The individuals selected to be parents are stored sequentially one by one in a population P_{SP}.

Sub-step 6(b) combines the selected parents in P_{SP} to compose a new population of offspring O_G. The crossover operation (i.e. combination of chromosomes for producing new chromosomes) is performed using the population P_{SP} according to a predefined crossover probability $p_c^{\%}$. The crossover probability defines how often crossover will be performed [Sivanandam and Deepa, 2007]. For a given value of $p_c^{\%}$, a percentage of $(1 - p_c^{\%})$ of the offspring are copies from their parents, and a percentage of $p_c^{\%}$ of the offspring are obtained by crossover. In this thesis, $p_c^{\%}$ is set

Algorithm 4.1 Genetic algorithm for designing neural network ensemble (GA-NNE).

Inputs: number of candidate models N; number of bits to represent all the combination strategies c; mutation probability $p_m^{\%}$; selection probability $p_s^{\%}$; crossover probability $p_c^{\%}$; number of individuals selected by elitism e; number of individuals of the population K; maximum number of generations G_{max};

1. Produce N candidate models according to Subsection 4.2.2;
2. Generate randomly an *initial population* P_1 with K individuals;
3. Evaluate the *fitness* of each individual of P_1 with all possible combination strategies;
4. Assign the best combination strategy (on the last c bits) to each individual of P_1 according to the *fitness*;
5. Set generation number as $G \leftarrow 1$;
6. **Repeat**:
   (a) *Select* a percentage $p_s^{\%}$ of the individuals of P_G; and store them sequentially in a population P_{SP} of selected parents;
   (b) Perform *crossover*, with a crossover probability $p_c^{\%}$, on the individuals of P_{SP} to generate a new population of offspring O_G;
   (c) *Mutate* randomly with probability $p_m^{\%}$ on the first N bits (the *model* part) of the individuals in O_G;
   (d) Evaluate the *fitness* of each individual in O_G with all possible combination strategies;
   (e) Assign to each individual in O_G the best combination strategy (on the last c bits) according to the *fitness*;
   (f) *Select* individuals for the new population P_{G+1}:
      i. Set P'_G as a temporary population $P'_G \leftarrow (O_G \cup P_G)$;
      ii. Apply *elitism* by assigning to P_{G+1} the e individuals of P'_G with the best *fitness*;
      iii. *Select* $(K - e)$ individuals of P'_G for P_{G+1};
   (g) Set $G \leftarrow G + 1$;

   until $G = G_{max}$.

to 100%; and the crossover operation is performed by combining two parents from P_{SP} for producing two offspring. To perform each crossover operation, the next set of two parents is sequentially selected from P_{SP}.

The crossover operation is done using uniform crossover with a random mask, where two parents, $parent_1$ and $parent_2$, generate two offspring, $offspring_1$ and $offspring_2$. The method applies crossover only on the *model* part of each individual. First, a random binary mask of length N is produced. If there is a 0 in a bit of the mask, then the corresponding bit from $parent_1$ is copied to the same position in $offspring_1$ and the corresponding bit from $parent_2$ is copied to the same position in $offspring_2$. If there is 1 in the bit of the mask, the corresponding bit from $parent_1$ is copied to the same position in $offspring_2$ and the corresponding bit from $parent_2$ is copied to the same position in $offspring_1$ [Haupt, 2004].

Sub-step 6(c) selects a percentage of $p_m^{\%}$ of the individuals from population O_G to be mutated on the *model* part, where $p_m^{\%}$ is the mutation probability. In this operation, for each selected individual one or more bits can be randomly changed to foster diversity in the offspring population. Sub-steps 6(d), and 6(e) evaluate each individual of O_G with all possible combination strategies and then the best combination type is assigned to the last c bits.

Individuals for composing the new population P_{G+1} are picked in sub-step 6(f). In this operation, *elitism* is applied, by means of which e (with $e < K$) individuals with the best *fitness* from $O_G \cup P_G$ are assigned to P_{G+1} (of size K). Then $(K - e)$ individuals of P_{G+1} are selected from $O_G \cup P_G$ using Roulette Wheel *Selection*.

After G_{max} generations, an individual with the best *fitness* of the last population is selected as the final solution to the problem.

4.3.2 Simulated Annealing for Designing Neural Network Ensemble (SA-NNE)

SA is a meta-heuristic that has proven to be effective in solving many difficult problems, especially combinatorial problems. Annealing is a process to change materials' properties. It is performed by heating a material and then freezing it slowly until it crystallizes. As the heating allows the atoms to move randomly, the cooling process should be slow enough to allow atoms to move themselves to lower energy positions. Considering this procedure as an optimization problem, if the atoms' arrangement

Algorithm 4.2 Simulated annealing for designing neural network ensemble (SA-NNE) (*Maximization problem*).

Inputs: number of candidate models N; number of bits to represent all the combination strategies c; initial temperature τ_i; final temperature τ_f; number of tries per iteration tr; colling factor ν; Hamming distance $\mathcal{H}$;

1. Produce N candidate models according to Subsection 4.2.2;
2. Generate randomly a current solution s_c composed of $(N + c)$ bits;
3. Set $\tau \leftarrow \tau_i$;
4. **Repeat**:
   (a) Set $q \leftarrow 0$;
   (b) **Repeat**:
      i. Select randomly a new $(N + c)$-bits-solution s_n in the neighborhood of s_c defined using a Hamming distance $\mathcal{H}$;
      ii. **if** $eval(s_c) < eval(s_n)$
         A. **then** $s_c \leftarrow s_n$;
         B. **else if** $random[0, 1) < \exp\left(\frac{eval(s_n) - eval(s_c)}{\tau}\right)$
            a. **then** $s_c \leftarrow s_n$;
      iii. Set $q \leftarrow q + 1$;

      until $q = tr$.
   (c) Set $\tau \leftarrow \tau \times \nu$;

   until $\tau \leqslant \tau_f$.

is achieved with the lowest energy level, the arrangement is an optimal solution to the energy minimization problem. SA applies this analogy in order to search for the optimal solution to an optimization problem [Michalewicz and Fogel, 2000]. The main SA's advantage is the ability to avoid becoming trapped at local optima.

This Section develops a SA based approach for designing ensembles of NN models, as detailed in Algorithm 4.2. SA-NNE is able to simultaneously select the best subset of models from a set of candidate models, and the best combination type for aggregating the subset. A solution is encoded according to Figure 4.1 and $1/\text{MSE}^{test}$ is used as the evaluation function, given by $eval()$.

SA-NNE is started by setting the parameters N, c, τ_i, τ_f, tr, ν, and $\mathcal{H}$. Step 1 produces N candidate NN models according to the description in subsection 4.2.2. In Step 2, a current solution s_c of size $(N + c)$ bits is randomly generated.

Step 3 assigns the initial temperature τ_i to the temperature parameter τ. In Step 4, SA-NNE loops over until temperature τ is equal to final temperature τ_f. Temperature τ is gradually decreased according to the cooling ratio $\tau \leftarrow \tau \times \nu$, where ν is the cooling factor. In Step 4(b)i, a new solution s_n is randomly selected in the neighborhood of s_c. This neighborhood is defined using a Hamming distance $\mathcal{H}$, that is, only $\mathcal{H}$ bits at most change from s_c to s_n. If s_n is better than s_c, then the algorithm always accepts s_n as the new s_c. If s_n is worse than s_c, there is a probability of acceptance of s_n that depends on the current value of τ and a random value, as described in Step 4(b)ii. Parameter tr is the maximum number of *tries* allowed for a given value of the temperature parameter.

4.4 Experimental Results

In this Section, experiments to evaluate the proposed GA-NNE and SA-NNE approaches are described. The main objectives of the experiments are: (*i*) to evaluate the performance of a single NN; (*ii*) to analyze the characteristics of the candidate NN models; (*iii*) to evaluate the GA-NNE and SA-NNE performances by varying important parameters; and (*iv*) to compare GA-NNE and SA-NNE to other ensemble techniques.

4.4.1 Data Set Description

Experiments are performed using two benchmark data sets (for regression models) available at Luís Torgo's website [Torgo, 2011], Friedman and Boston Housing; and three real-world industrial data sets. The use of Friedman and Boston Housing data sets allows the control and setting of relevant parameters of the GA-NNE and SA-NNE algorithms; while the industrial data sets are employed to prove the effectiveness of the proposed methodologies in real-world case studies. The benchmarks for regression are summarized as follows:

- *Friedman*: The Friedman function is a well-known function for data generation [Friedman, 1991]. It uses both non-linear and linear relations between

output and inputs. The original Friedman function contains five independent variables, and is defined as follows:

$$y = 10 \sin(\pi x_1 x_2) + 20 \left(x_3 - \frac{1}{2} \right)^2 + 10 x_4 + 5 x_5 + \epsilon \,, \tag{4.1}$$

where $\epsilon \sim N(0, 1)$ is a random variable with normal (Gaussian) probability distribution with zero mean and unit variance. In the data set, the input space is increased by adding other five independent variables $x_6, \ldots, x_{10}$ that do not have influence on y. Each of the variables $x_1, x_2, \ldots, x_{10}$ is uniformly distributed over $[0, 1]$. The data set includes 40768 samples.

- *Boston Housing*: this data set has been applied extensively in literature to benchmark methods. It contains information collected by the United States Census Service about housing in the area of Boston, Massachusetts. The data set consists of 13 independent variables (mainly socio-economic) and 1 output variable (median housing price). The data set is smaller than the Friedman data set, with 506 samples.

The industrial data sets are the debutanizer column, polymerization reactor, and cement kiln data sets. They are described in Section 1.4. Input variable selection was performed to remove variables with missing values and noises, and to select input variables highly correlated to the output [Fortuna *et al.*, 2006] for the polymerization reactor and cement kiln data sets. Despite the fact that input variable selection is an important step in SS applications [Grbić *et al.*, 2013], this thesis focus mainly on the model selection, training and validation steps. Variable selection and dimensionality reduction is not a central topic in this thesis. The data sets are summarized as follows:

- *Polymerization reactor*: The data set covers one year of operation of the process plant and it contains 15 input variables $(x^1, \ldots, x^{15})$, 1 output (the catalyst activity) and 8687 samples. The sampling rate of the output for the last 3000 samples is lower so that the data set was preprocessed as follows: downsample by a factor of 10 of the first 5687 samples; remove x^3, x^4, and x^{15} [Miranda, 2012; Kadlec and Gabrys, 2011], since they are affected by outliers and missing samples; remove x^2, since it is not correlated to the output; remove x^{13}, since it is redundant with x^{12}; and exclude all the samples with missing output values.

At the end of this process, the data set contains 648 samples, 10 inputs, and 1 output.

- *Debutanizer column*: it contains 2394 samples; 1 output (butane concentration); and 7 input variables. No input variable selection was performed.

- *Cement kiln*: input variables and output variable (free lime concentration) have different sampling intervals, so that samples with missing output values were removed from the data set. Moreover, since the data set is high dimensional (195 input variables), only inputs correlated to the output were selected. At the end of this process, the data set contains 701 samples and 45 inputs.

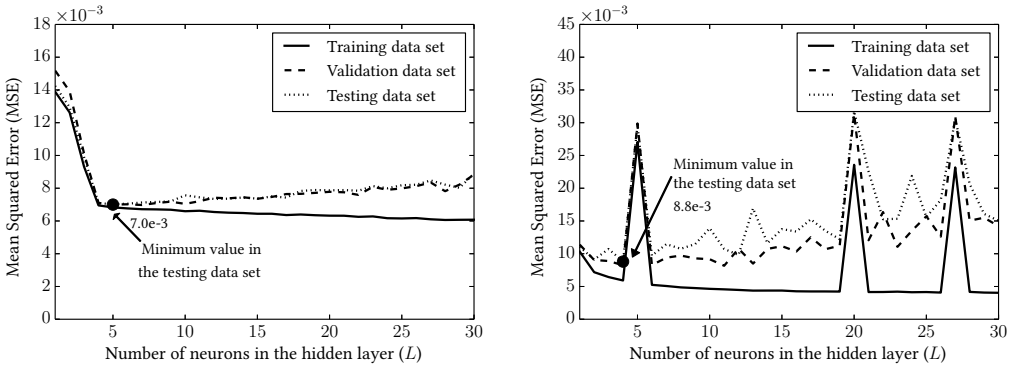
Experiments are organized in runs. Each run is evaluated using 10-*fold cross-validation*, where the data set is split into 10 subsets. The result is the average of the results of the 10 subsets, where each subset is in turn used as a testing data set while the samples of the other 9 subsets are randomly divided into a training data set (90%), and a validation data set (10%). At the end of this process, there are 10 *artificial* data sets each of which consists of training, validation, and testing data sets. Below, the results of MSE^{test} are given by averaging the MSE of all 10 testing subsets.

Subsections from 4.4.2 to 4.4.6 report experimental results using the Friedman and Boston Housing data sets; while Subsection 4.4.7 reports experimental results using all the data sets.

4.4.2 Individual Neural Networks

First, the performance of individual NN models is investigated by altering the number of neurons in the hidden layer L. The experiment was done by setting the synaptic weights initialization approach and activation function according to popularity, where Nguyen-Widrow was selected as the synaptic weights initialization technique, and fast hyperbolic tangent and linear were selected as the activation functions for the hidden layer ($g(x)$) and the output layer ($h(x)$), respectively.

Early stopping criteria is applied as a strategy to control overfitting [Caruana *et al.*, 2000]. Early stopping has been recognized as a good strategy for avoiding overfitting and optimizing the generalization performance of NN models in practice [Jeong and Kim, 2005]. The main idea is to inspect the test error of a NN model



(a) Friedman data set. (b) Boston Housing data set.

Figure 4.3: Performance of the individual NN models.

on an independent set using a validation data set, so that when the validation data set error starts to increase the NN training is stopped to avoid overfitting. In this thesis, early stopping is employed through the following procedures: set the maximum number of epochs as 500; use the results of the first NN training for initializing the current best synaptic weights; then, train the NN calculating the validation data set error every 50 epochs and keep the NN synaptic weights at this current point; if the validation data set error has decreased in comparison to the previous point, continue the NN training and assign the current synaptic weights as the current best NN synaptic weights; if the validation data set error at the current point has risen in comparison to the previous point, terminate the NN training and assign the current best NN synaptic weights as the final NN synaptic weights.

Figure 4.3 shows the performance of the individual NN models, where MSE is obtained by 10-*fold cross-validation*. The experiment reveals that NN models with lower number of neurons in the hidden layer are less prone to overfitting and consequently these NN models have better generalization capability. For this reason, in this Chapter, the maximum number of neurons in the hidden layer is limited to 10.

Moreover, Figures 4.3(a) and 4.3(b) illustrate the minimum MSE value achieved for a NN in the testing data set over all NN models. For the Friedman data set, the minimum value is 7.0×10^{-3}, and the NN has 5 neurons in the hidden layer ($L = 5$); and for the Boston Housing data set, the minimum value is 8.8×10^{-3}, and the NN has 4 neurons in the hidden layer ($L = 4$). In the next experiments, a reduction of the minimum value is observed.

Table 4.1: MSE^{test} results using EBP.

Data Set	Combination Type			
	mean	trimmed mean	median	weighted mean
Friedman	2.100	2.100	2.100	2.100
Boston Housing	4.500	4.400	4.400	4.500

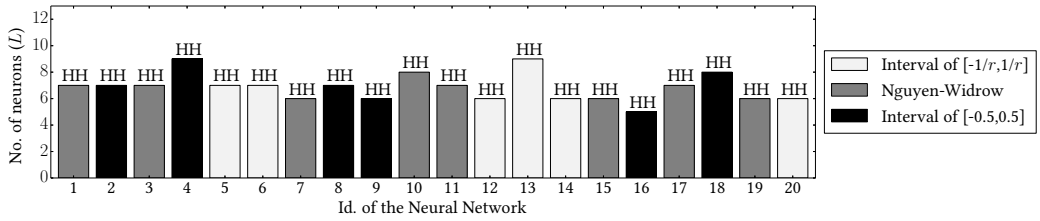
All the MSE values have been multiplied by 10^3 in the table.

4.4.3 Generation of the Candidate Neural Networks

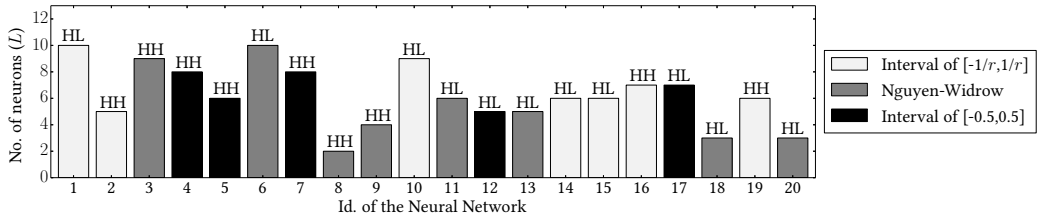
This Subsection details the characteristics of the set of candidate NNs to be used in the next experiments. A set of 20 candidate models ($N = 20$) was generated according to Subsection 4.2.2 by 10-*fold cross-validation*. The set is used by the GA-NNE and SA-NNE approaches. Before doing the experiments with GA-NNE and SA-NNE, all the 20 candidate NN models were aggregated to constitute an ensemble. Therefore, no optimization techniques were employed to select the best subset of models. The term Ensemble Before Pruning (EBP) is employed to refer to this ensemble.

EBP was implemented using *mean*, *trimmed mean*, *median*, and *weighted mean* as combination types. Table 4.1 shows the results of EBP based on the MSE in the testing data set. For the Friedman data set, the combination types have the same value of MSE^{test}. On the other hand, *median* and *trimmed median* outperform *mean* and *weighted mean* for the Boston Housing data set. From the results, it is possible to notice that EBP has good generalization ability when compared to the individual models generated in Subsection 4.4.2.

For the Friedman and Boston Housing data sets, an artificial data set from 10-*fold cross-validation* was randomly chosen to show the characteristics of the candidate NNs. Figure 4.4 shows the NN's properties, such as the number of neurons in the hidden layer, the synaptic weights initialization type, and the activation function types for the hidden layer and the output layer. For the activation functions in the layers the abbreviations displayed in Table 4.2 are used. It is observed that the models from the Friedman data set have a higher number of neurons in the hidden layer when compared to the models from Boston Housing data set. Moreover, for the Friedman data set all the NN models have the fast hyperbolic tangent as activation function for both the hidden layer and the output layer.



(a) Friedman data set.



(b) Boston Housing data set.

Figure 4.4: NN's properties of a subset from 10-*fold cross-validation*.

Table 4.2: Abbreviations for the activation functions in the layers.

Abbreviation	Hidden layer activation function $g(x)$	Output layer activation function $h(x)$
HH	fast hyperbolic tangent	fast hyperbolic tangent
LL	linear	linear
HL	fast hyperbolic tangent	linear
LH	linear	fast hyperbolic tangent

4.4.4 Genetic Algorithm for Designing Neural Network Ensembles

After generating the 20 candidate models, Algorithm 4.1 proceeds with Step 2. Several experiments were performed by varying the GA-NNE's inputs/parameters among the following values:

- *Mutation probability*: $p_m^{\%} \in \{5\%, 10\%, 15\%\}$;
- *Selection probability*: $p_s^{\%} \in \{60\%, 100\%\}$;
- *Population size*: $K \in \{20, 40\}$.

The crossover probability $p_c^{\%}$ is set to 100%, the number of mutated bits for the mutation operations is set to $c = 2$, the maximum number of generations is set to G_{max}=500, and one individual is selected by elitism for the next generation ($e = 1$).

Table 4.3: MSE^{test} results of GA-NNE using the Friedman data set on 12 experiments.

No. exp.	1	2	3	4	5	6	7	8	9	10	11	12
$p_m^{\%}$	5%	5%	5%	5%	10%	10%	10%	10%	15%	15%	15%	15%
$p_s^{\%}$	60%	100%	60%	100%	60%	100%	60%	100%	60%	100%	60%	100%
K	20	40	20	40	20	40	20	40	20	40	20	40
Mean	1.794	1.789	1.794	1.787	1.791	1.788	1.788	1.787	1.793	1.787	1.789	**1.786**
S.D.	0.005	0.002	0.005	0.002	0.003	0.003	0.002	0.001	0.003	0.001	0.003	**0.001**
Min	1.787	1.785	1.785	1.785	1.786	1.785	1.785	1.785	1.787	1.785	1.785	**1.785**
Max	1.808	1.795	1.803	1.795	1.799	1.797	1.794	1.790	1.798	1.791	1.797	**1.788**

Each experiment is composed of 20 runs, and each run consists of a 10-*fold cross-validation*. All the MSE^{test} values have been multiplied by 10^3 in the table.

Table 4.4: MSE^{test} results of GA-NNE using the Boston Housing data set on 12 experiments.

No. exp.	1	2	3	4	5	6	7	8	9	10	11	12
$p_m^{\%}$	5%	5%	5%	5%	10%	10%	10%	10%	15%	15%	15%	15%
$p_s^{\%}$	60%	100%	60%	100%	60%	100%	60%	100%	60%	100%	60%	100%
K	20	40	20	40	20	40	20	40	20	40	20	40
Mean	2.454	2.445	2.458	2.449	2.447	2.445	2.445	2.445	2.447	2.443	2.445	**2.441**
S.D.	0.008	0.004	0.010	0.008	0.007	0.003	0.004	0.003	0.007	0.003	0.004	**0.002**
Min	2.442	2.439	2.441	2.440	2.439	2.440	2.438	2.439	2.439	2.439	2.439	**2.438**
Max	2.472	2.453	2.478	2.471	2.471	2.452	2.457	2.452	2.461	2.449	2.457	**2.450**

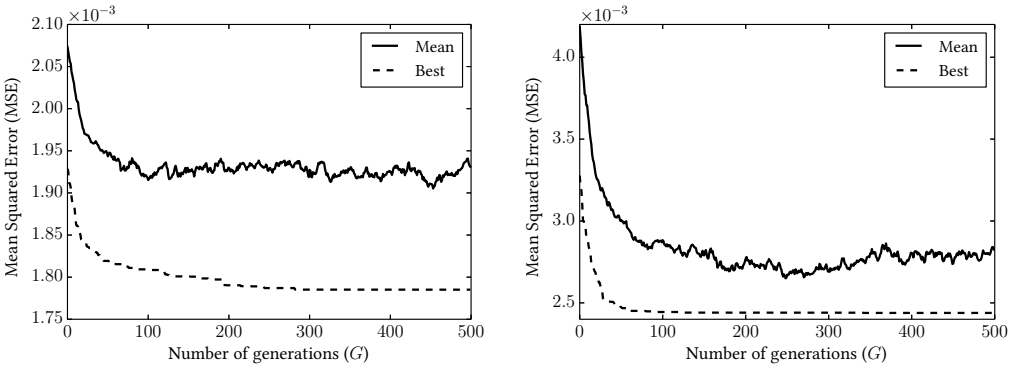
Each experiment is composed of 20 runs, and each run consists of a 10-*fold cross-validation*. All the MSE^{test} values have been multiplied by 10^3 in the table.

Table 4.3 and Table 4.4 show the mean, standard deviation (SD), minimum, and maximum of the MSE obtained with GA-NNE in 12 experiments, with 20 runs on each experiment, using the Friedman and Boston Housing data sets, where 10-*fold cross-validation* is performed for each run.

Considering mean as the metric to evaluate the performance in the experiments, some characteristics are noticed for both data sets. For example, in most experiments, GA-NNE's results improve when the mutation probability ($p_m^{\%}$) increases. Moreover, selection probability $p_s^{\%} = 100\%$ has better performance when compared to $p_s^{\%} = 60\%$. The experiments indicate that improvements are obtained when the population size is $K = 40$.

Table 4.5: GA-NNE - Percentage of combination type selection on the 20 runs of the best experiment.

Data set	Combination type			
	mean	trimmed mean	median	weighted mean
Friedman	0%	0%	**90%**	10%
Boston Housing	0%	0%	**90%**	10%



(a) Friedman data set. (b) Boston Housing data set.

Figure 4.5: GA-NNE - Mean of the MSE (i.e., 1/*fitness*) of the individuals in the population and MSE of the best individual in the population *versus* the number of generations (G) of the best run of the best experiments of Table 4.3 and Table 4.4. The average of the 10 test subsets of the 10-*fold cross-validation* is presented.

The experiments with the best performance in Table 4.3 and Table 4.4 are shown in bold. Considering the best individuals at the end of the 20 runs of experiment 12, the percentage of selection of each combination type is shown in Table 4.5. For both data sets *median* is the most frequently selected strategy.

Considering again experiment 12, the runs with the best MSE performance (i.e., *min* value) are the runs with MSE of 1.785×10^{-3} and 2.438×10^{-3} for the Friedman and Boston Housing data sets, respectively. Figure 4.5 shows the properties of these best runs according to the average of the 10 test subsets of the 10-*fold cross-validation*. *Mean* is the average of the MSE (i.e., 1/*fitness*) of all the individuals in the population in a generation, and *Best* is the best value of MSE for all individuals of the population in a generation. As can be seen, no important improvements are shown after 300 generations for the Friedman data set and 100 generations for the Boston Housing data set.

Table 4.6: MSE^{test} results of SA-NNE using the Friedman data set on 12 experiments.

No. exp.	1	2	3	4	5	6	7	8	9	10	11	12
ν	0.85	0.85	0.85	0.85	0.90	0.90	0.90	0.90	0.95	0.95	0.95	0.95
$\mathcal{H}$	1	1	2	2	1	1	2	2	1	1	2	2
tr	1	2	1	2	1	2	1	2	1	2	1	2
Mean	1.835	1.838	1.811	1.800	1.836	1.836	1.797	1.791	1.839	1.832	1.796	**1.790**
SD	0.010	0.016	0.006	0.007	0.013	0.011	0.006	0.003	0.012	0.009	0.006	**0.003**
Min	1.820	1.818	1.797	1.788	1.811	1.809	1.788	1.785	1.808	1.814	1.785	**1.785**
Max	1.857	1.879	1.822	1.814	1.859	1.864	1.813	1.798	1.855	1.847	1.815	**1.798**

Each experiment is composed of 20 runs, and each run consists of a 10-*fold cross-validation*. All the MSE^{test} values have been multiplied by 10^3 in the table.

4.4.5 Simulated Annealing for Designing Neural Network Ensembles

Using the 20 candidate models, several experiments were carried out by varying the SA-NNE's parameters as follows:

- *Cooling factor*: $\nu \in \{0.85, 0.90, 0.95\}$;
- *Hamming distance*: $\mathcal{H} \in \{1, 2\}$;
- *Number of tries*: $tr \in \{1, 2\}$.

The initial temperature τ_i is set to 1000 and the algorithm stops when the final temperature τ_f is 10^{-20} or less.

Table 4.6 and Table 4.7 show the mean, SD, minimum, and maximum of the MSE obtained with the SA-NNE in 12 experiments, with 20 runs on each experiment, using the Friedman and Boston Housing data sets. Considering the mean to evaluate the performance of all experiments, some patterns are observed. For example, most experiments with a high cooling factor (ν) have better results when compared to experiments with a low cooling factor (ν). In general, a Hamming distance $\mathcal{H} = 2$ outperforms $\mathcal{H} = 1$ and the number of *tries* $tr = 2$ has the best performance.

The experiments with the best performance in Table 4.6 and Table 4.7 are shown in bold. In this case, the best experiment for both the Friedman data set and Boston Housing data set is experiment 12. Considering the best individuals after 20 runs

Table 4.7: MSE^{test} results of SA-NNE using the Boston Housing data set on 12 experiments.

No. exp.	1	2	3	4	5	6	7	8	9	10	11	12
ν	0.85	0.85	0.85	0.85	0.90	0.90	0.90	0.90	0.95	0.95	0.95	0.95
$\mathcal{H}$	1	1	2	2	1	1	2	2	1	1	2	2
tr	1	2	1	2	1	2	1	2	1	2	1	2
Mean	2.656	2.633	2.507	2.480	2.649	2.639	2.484	2.464	2.612	2.602	2.466	**2.454**
SD	0.053	0.055	0.024	0.023	0.059	0.075	0.024	0.013	0.048	0.051	0.019	**0.011**
Min	2.558	2.551	2.463	2.448	2.545	2.533	2.456	2.438	2.535	2.534	2.440	**2.441**
Max	2.792	2.743	2.547	2.529	2.746	2.805	2.542	2.487	2.712	2.743	2.509	**2.484**

Each experiment is composed of 20 runs, and each run consists of a 10-*fold cross-validation*. All the MSE^{test} values have been multiplied by 10^3 in the table.

Table 4.8: SA-NNE - Percentage of combination type selection on the 20 runs of the best experiment.

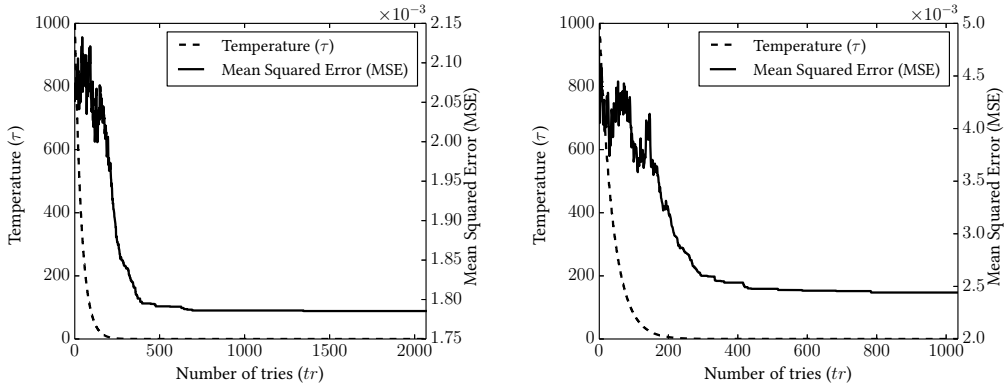
Data set	Combination type			
	mean	trimmed mean	median	weighted mean
Friedman	0%	0%	**90%**	10%
Boston Housing	0%	0%	**90%**	10%

of these best experiments, the percentage of selection of each combination type is shown in Table 4.8. Again, *median* is the most frequently selected combination strategy.

For experiment 12, the run with the best MSE performance has a MSE value of 1.785×10^{-3} for the Friedman data set and 2.441×10^{-3} for Boston Housing data set. The behavior of these runs is shown in Figure 4.6. As can be seen, improvements of the MSE follow the decaying of temperature. In the initial *tries*, it is observed that MSE increases its value. This happens because SA-NNE can more easily accept worse solutions when the temperature is high.

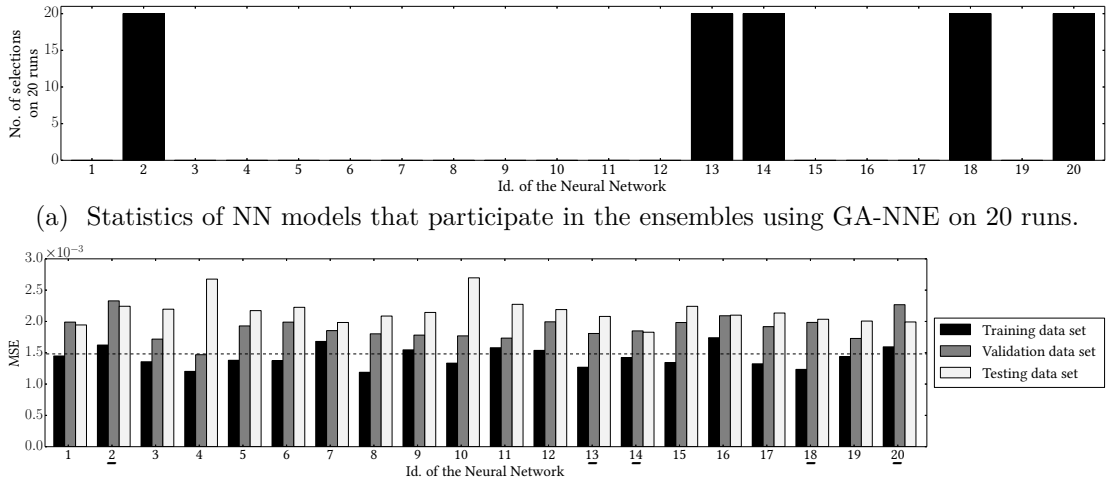
4.4.6 The Models Selected by GA-NNE and SA-NNE

In this Subsection, characteristics of the models selected by GA-NNE and SA-NNE are detailed. The same *artificial* data set from 10-*fold cross-validation* of Subsection 4.4.3 is considered. Here, the results discussed are based on the best experiments



(a) Friedman data set. (b) Boston Housing data set.

Figure 4.6: SA-NNE - Decay of temperature and MSE *versus* number of *tries* (tr). The average of the 10 test subsets of the 10-*fold cross-validation* is presented.



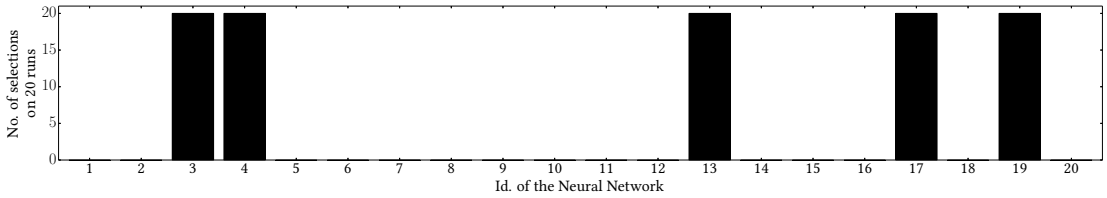
(a) Statistics of NN models that participate in the ensembles using GA-NNE on 20 runs.

(b) Accuracy of each NN in the data set.

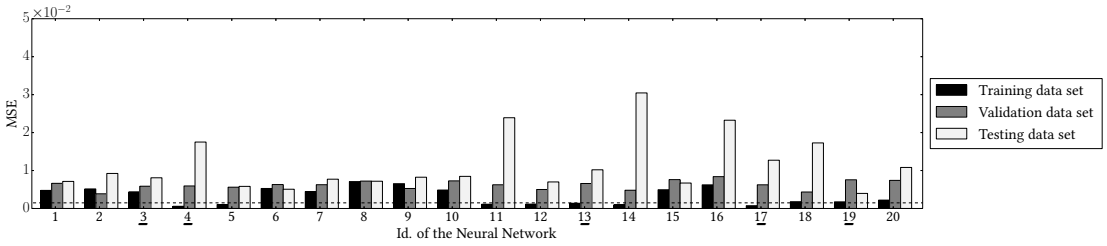
Figure 4.7: Results of the GA-NNE on the best experiment on the Friedman data set. The dashed line represents the MSE^{test} of the ensemble of the best run; underlined numbers represent the selected NN models to design such ensemble.

from Subsection 4.4.4 and Subsection 4.4.5.

Figures 4.7(a), 4.8(a), 4.9(a), and 4.10(a) show the statistics of the numbers of selections of the NN models that participate in the ensembles on the 20 runs, considering the presence of a NN in the final solution of each run. Figures 4.7(b), 4.8(b), 4.9(b), and 4.10(b) display the accuracy of each NN based on the training, validation, and testing data sets. The dashed line represents the MSE^{test} value of the ensemble on the run with minimum MSE value. The underlined NN numbers

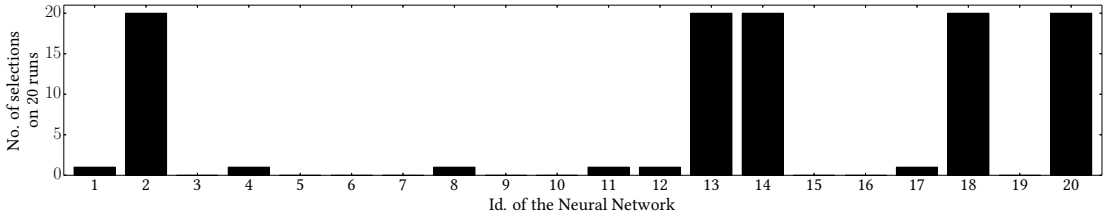


(a) Statistics of NN models that participate in the ensembles using GA-NNE on 20 runs.

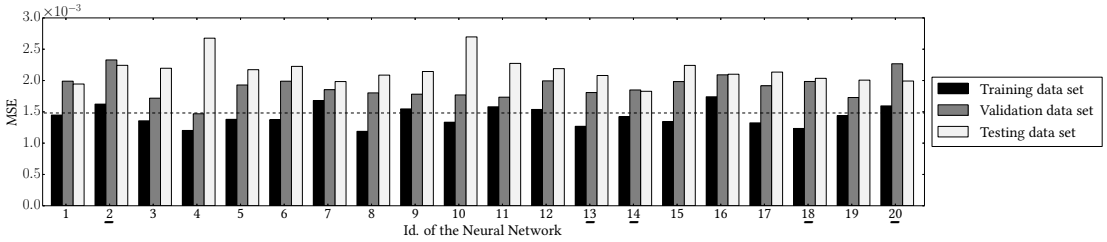


(b) Accuracy of each NN in the data set.

Figure 4.8: Results of the GA-NNE on the best experiment on the Boston Housing data set. The dashed line represents the MSE^{test} of the ensemble of the best run; underlined numbers represent the selected NN models to design such ensemble.



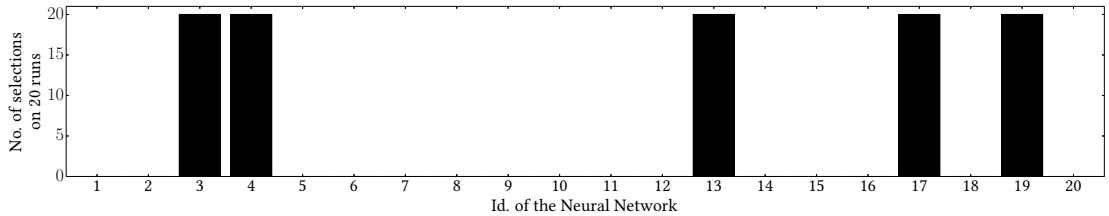
(a) Statistics of NN models that participate in the ensembles using SA-NNE on 20 runs.



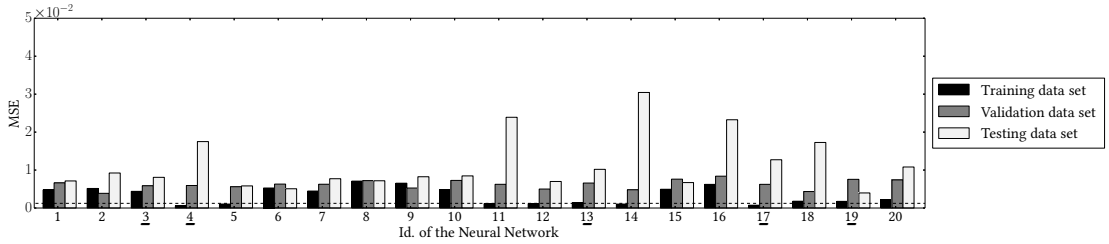
(b) Accuracy of each NN in the data set.

Figure 4.9: Results of the SA-NNE on the best experiment on the Friedman data set. The dashed line represents the MSE^{test} of the ensemble of the best run; underlined numbers represent the selected NN models to design such ensemble.

represent the NN models selected for designing the ensemble of this run. As can be seen, all the cases design ensembles with MSE^{test} values lower than the NN with the lowest MSE^{test} value. This confirms that the ensemble is more accurate than any single model in the ensemble.



(a) Statistics of NN models that participate in the ensembles using SA-NNE on 20 runs.



(b) Accuracy of each NN in the data set.

Figure 4.10: Results of the SA-NNE on the best experiment on the Boston Housing data set. The dashed line represents the MSE^{test} of the ensemble of the best run; underlined numbers represent the selected NN models to design such ensemble

In Figures 4.7, 4.8, 4.9, and 4.10, it is observed that the most common models selected by all runs are also selected by the best run, i.e., the one with the minimum MSE value. Common characteristics are noticed for the GA-NNE and SA-NNE approaches. Specifically, GA-NNE and SA-NNE select the same models for designing the ensemble and the same combination type. For the Friedman data set, as depicted in Figures 4.7(b) (GA-NNE) and 4.9(b) (SA-NNE), the selected NN models for designing the ensemble are $\{2, 13, 14, 18, 20\}$, $\text{MSE}^{test} = 1.481 \times 10^{-3}$ and *median* is the combination type. For the Boston Housing data set, as displayed in Figures 4.8(b) (GA-NNE) and 4.10(b) (SA-NNE), the selected NN models for aggregating the ensemble are $\{3, 4, 13, 17, 19\}$, $\text{MSE}^{test} = 1.231 \times 10^{-3}$ and *median* is the combination type.

4.4.7 Comparisons of the Ensemble Systems

In this Subsection, the proposed GA-NNE and SA-NNE methodologies are compared to other ensemble systems. The ensemble systems include Simple Bagging, GASEN, NCL, and AdaBoost.

As mentioned before, Bagging creates an ensemble where each model is trained by a different training data set using bootstrap resampling. Bagging is a common

Table 4.9: Comparison of ensemble systems: MSE^{test} results using the Friedman data set.

	AdaBoost	Simple Bagging	NCL	EBP				Pruned Bagging		
				mean t.	mean	median	w. mean	GASEN	SA-NNE	GA-NNE
Mean	24.465	7.081	2.438	2.100	2.100	2.100	2.100	1.914	1.790	**1.786**
SD	0.000	0.027	0.029	-	-	-	-	0.034	0.003	**0.001**
Min	25.465	7.036	2.396	-	-	-	-	1.862	1.785	**1.785**
Max	25.465	7.138	2.510	-	-	-	-	1.964	1.798	**1.788**

The results are for 20 runs, except for EBP (one run). All the MSE^{test} values have been multiplied by 10^3 in the table.

technique applied to GASEN, GA-NNE, and SA-NNE. However, as these approaches employ pruning techniques for selecting the best subset of models, in this Subsection the Bagging strategy is applied for designing an ensemble without pruning technique, i.e. all the candidate NN models are aggregated. Additionally, in this ensemble the NN component-models have fixed architecture and parameters since most Bagging applications apply this strategy. To distinguish from other approaches (GASEN, GA-NNE, and SA-NNE), this ensemble of NN models is called "Simple Bagging". In the experiments, Simple Bagging is composed of 20 NN models using *mean* as the combination strategy. The Nguyen-Widrow method is employed for synaptic weights initialization. Fast hyperbolic tangent and linear activation functions are used for the hidden layer and output layer, respectively.

The number of neurons in the hidden layer was chosen using the experiment in Subsection 4.4.2. Specifically, this number was selected according to the performance in the validation data set (see Figure 4.3). Early stopping (as detailed in Subsection 4.4.2) was chosen for controlling overfitting.

GASEN employs a GA to select the appropriate subset of NN models to constitute the ensemble [Zhou *et al.*, 2002]. GASEN assigns a combination weight to each model, and then models with combination weights higher than a specified threshold λ_{GASEN} are selected to compose the ensemble. In the GASEN procedure, combination weights evolve using a GA and the *fitness* function is characterized by the generalization error of the ensemble. Experiments were done using the code available at the website `http://lamda.nju.edu.cn/files/Gasen.zip`.

The experiment and parameter setting were performed according to [Zhou *et al.*,

Table 4.10: Comparison of ensemble systems: MSE^{test} results using the Boston Housing data set.

	AdaBoost	Simple	NCL	EBP				Pruned Bagging		
		Bagging		mean t.	mean	median	w. mean	GASEN	SA-NNE	GA-NNE
Mean	19.162	9.129	7.817	4.500	4.400	4.400	4.500	6.254	2.454	**2.441**
SD	0.000	0.509	0.092	-	-	-	-	0.564	0.011	**0.002**
Min	19.162	8.104	7.629	-	-	-	-	5.457	2.441	**2.438**
Max	19.162	10.176	8.016	-	-	-	-	7.440	2.484	**2.450**

The results are for 20 runs, except for EBP (one run). All the MSE^{test} values have been multiplied by 10^3 in the table.

Table 4.11: Comparison of ensemble systems: MSE^{test} results using the polymerization reactor data set.

	AdaBoost	Simple	NCL	EBP				Pruned Bagging		
		Bagging		mean t.	mean	median	w. mean	GASEN	SA-NNE	GA-NNE
Mean	9.419	6.872	1.690	1.556	1.261	0.962	1.548	3.362	0.365	**0.361**
SD	0.000	0.377	0.016	-	-	-	-	3.474	0.005	**0.001**
Min	9.419	6.360	1.665	-	-	-	-	1.759	0.361	**0.359**
Max	9.419	7.528	1.719	-	-	-	-	17.768	0.378	**0.364**

The results are for 20 runs, except for EBP (one run). All the MSE^{test} values have been multiplied by 10^3 in the table.

Table 4.12: Comparison of ensemble systems: MSE^{test} results using the cement kiln data set.

	AdaBoost	Simple	NCL	EBP				Pruned Bagging		
		Bagging		mean t.	mean	median	w. mean	GASEN	SA-NNE	GA-NNE
Mean	18.533	21.899	10.600	10.384	10.285	10.621	10.383	9.766	7.683	**7.634**
SD	0.000	1.834	0.340	-	-	-	-	0.517	0.044	**0.016**
Min	18.533	17.865	9.914	-	-	-	-	8.842	7.628	**7.616**
Max	18.533	24.454	11.264	-	-	-	-	10.925	7.806	**7.668**

The results are for 20 runs, except for EBP (one run). All the MSE^{test} values have been multiplied by 10^3 in the table.

2002], where the genetic operators are set to the default values of the Genetic Algorithm Optimization Toolbox (GAOT) [Houck *et al.*, 1996]. The predefined threshold

Table 4.13: Comparison of ensemble systems: MSE^{test} results using the debutanizer column data set.

	AdaBoost	Simple Bagging	NCL	EBP				Pruned Bagging		
				mean t.	mean	median	w. mean	GASEN	SA-NNE	GA-NNE
Mean	23.739	10.595	13.508	9.894	9.894	9.902	9.894	12.128	8.731	**8.701**
SD	0.000	0.100	0.203	-	-	-	-	0.153	0.026	**0.006**
Min	23.739	10.485	13.197	-	-	-	-	11.736	8.700	**8.691**
Max	23.739	10.897	13.947	-	-	-	-	12.375	8.796	**8.715**

The results are for 20 runs, except for EBP (one run). All the MSE^{test} values have been multiplied by 10^3 in the table.

λ_{GASEN} is set to 0.05. The initial set of candidate models is composed of 20 NN models. Each NN has just one hidden layer with five hidden neurons, where the NN is trained using the BP algorithm. Other NN's parameters are set to the default values, such as hyperbolic tangent sigmoid as activation function for the hidden layer, linear activation function for the output layer, and the training stops when the number of iterations reaches 100. GASEN uses simple average for combining the models' outputs.

NCL produces an ensemble of NN models using negative correlation [Liu and Yao, 1999]. The aim is to train the NN models in parallel and use a correlation penalty term λ_{NCL} in their cost function for ensuring specialization and cooperation among the individual NN models. In this thesis, the value of λ_{NCL} is determined using $\lambda_{NCL} = \frac{N}{N-1}$, where N is the number of NN models [Brown *et al.*, 2005a]. NCL is tested using the code available at Gavin Brown's website `http://www.cs.man.ac.uk/~gbrown/projects/nc/NCL.zip`. A set of 20 NN models is produced by the BP algorithm. The models' architecture and parameters are the same of as the ones of GASEN. NCL also uses simple average for combining the models' outputs.

In this thesis, the AdaBoost algorithm uses the GentleBoost logistic regression method as detailed in [Cristinacce and Cootes, 2007]. A weak learner is selected at each round and the residual displacements from the real output and predicted output are adjusted. In this model, the training samples have equal weight. After R rounds a strong regressor function $F(\mathbf{x})$ is the final output, where the weak learners have the same combination weights. GentleBoost was implemented using

the AdaBoost Toolbox [Cordiner, 2009]. The best performance of the boosting ensemble was achieved in 50 rounds.

For GA-NNE and SA-NNE, the parameters are set according to the best experiments of Subsection 4.4.4 and Subsection 4.4.5, respectively. That is, for GA-NNE, parameters are set to $p_m^{\%} = 15\%$, $p_s^{\%} = 100\%$, and $K = 40$; and for SA-NNE, parameters are set to $\nu = 0.95$, $\mathcal{H} = 2$, and $tr = 2$.

Tables 4.9-4.13 show the experimental results of MSE^{test} in the two benchmark data sets and three real-world industrial data sets using different ensemble systems on 20 runs, except for EBP where the results are for one run. As pointed out in Subsection 4.4.3, EBP is an ensemble with all the candidate NN models used by GA-NNE and SA-NNE. Therefore, EBP is an intermediate ensemble obtained before performing these pruning techniques.

It can be seen that ensembles of NN models (e.g. NCL, Simple Bagging, EBP, GASEN, SA-NNE, and GA-NNE) outperform AdaBoost in most data sets. NCL also obtains more accurate predictions than Simple Bagging and AdaBoost in most cases. NCL can produce NN ensembles with good generalization ability (compared to Simple Bagging). However, within an/each NCL ensemble, component-models have the same architecture, making the ensemble have a low degree of diversity.

Simple Bagging presents the worst generalization ability when compared to the other NNEs with the bootstrap technique (namely EBP, GASEN, GA-NNE, and SA-NNE). The main issue is that Simple Bagging achieves diversity just by manipulating the training data set while the models have the same architecture and parameters. Moreover, Simple Bagging does not employ any strategy for selecting the best subset of models and combination type.

On the other hand, EBP considerably outperforms Simple Bagging in terms of generalization ability. The success is attributed to the several diversity levels employed by EBP, for example, using a different synaptic weights initialization and a different architecture for each model in the ensemble.

It is observed that pruned Bagging systems (e.g. GASEN, SA-NNE, and GA-NNE) have better results when compared to ensembling all techniques (e.g. EBP, and Simple Bagging), except the lower performance obtained by GASEN for the Boston Housing data set.

As pointed out before, here EBP aggregates all the candidate models used by GA-NNE and SA-NNE. Therefore, it is noticed that the proposed GA-NNE and

Table 4.14: Average number of the selected NN models using different ensemble systems.

Data Set	GASEN	SA-NNE	GA-NNE
Friedman	**4.17**	5.31	5.10
Boston housing	**4.87**	5.69	5.81
Polymerization reactor	**3.74**	6.33	6.60
Cement kiln	5.85	4.96	**4.94**
Debutanizer column	**5.38**	5.56	6.01

SA-NNE approaches achieve good results when compared to EBP. The results prove the efficiency of models subset selection, and combination type selection during the ensemble development to obtain good generalization ability for the Friedman, Boston Housing, and industrial data sets.

Table 4.14 shows the average number of selected models by GASEN, GA-NNE, and SA-NNE. The results were obtained by averaging the number of selected models of the best individuals after the 20 runs on experiments described in the Table 4.9 and Table 4.10. Table 4.14 shows that SA-NNE and GA-NNE select a higher number of models when compared to GASEN.

4.5 Conclusion

NNE has established itself as a valuable tool for computational intelligence modeling. The main motivation is that the generalization ability of the system can be significantly improved. Most studies consider the following key factors during the ensemble development: diversity among the models, subset selection of models, and optimal combination strategy. Since there is no automatic procedure to implement these steps, this Chapter proposes and compares two approaches for automatic development of NNE: GA-NNE and SA-NNE.

The main contributions of the proposed methodologies are the selection of the subset of models and combination type providing a high degree of diversity among the models. First, models are generated by starting the learning with different conditions (synaptic weights initialization methods), using different training data sets (applying bootstrap), and using models with different learning parameters and architectures. Second, two optimization techniques, GA and SA, are used to select

the best subset of models and the optimal combination strategy.

The GA-NNE and SA-NNE methodologies proposed in this Chapter, obtained a superior performance when compared to well-known ensemble systems, including Simple Bagging, NCL, AdaBoost, and GASEN. This success results from the diversity among the NN models, and the optimal selection of the subset of models and combination type. These are crucial factors to ensure the ensemble performance in terms of generalization ability. Moreover, experiments have shown that GA-NNE and SA-NNE have good performance when compared to a single model and the aggregation of all candidate models (EBP). The results also revealed that GA-NNE and SA-NNE obtained a similar performance.

Therefore, the aim of proposing methodologies for automatic NNE development (taking into account diversity and accuracy) by meta-heuristics was reached in this Chapter. The experiments revealed that the proposed approach designs a pool of diverse NN models with different synaptic weights initialization strategies and number of hidden neurons; and that GA-NNE and SA-NNE can select a reduced subset of accurate NN models for building an ensemble system. The performance and effectiveness of the proposed methodologies in industrial problems has been demonstrated. According to the experimental results, the proposed methodologies can deliver more accurate estimations of key variables in industrial applications than well-known ensemble systems. However, adaptive mechanisms (such as the ones described in Section 3.5) may be necessary to guarantee the performance of the proposed ensembles in time-varying environments.

Chapter 5

An Adaptive Ensemble with Discounting Factor

Contents

5.1 Introduction

On-line learning applications where the target concept may change over time pose serious problems. Underlying changes may make the model designed on old data, inconsistent with new data. Section 3.5 described that this problem is called concept drift. A challenge in on-line learning models is to adapt and handle changes without being informed about them, and make use of the past experiences in situations where

old contexts may reappear. Reusing previously acquired knowledge can enhance the learning in terms of accuracy and processing time [Gomes *et al.*, 2014].

Ensemble learning has been proven itself as a valuable tool to handle concept drifts [Brzezinski and Stefanowski, 2014]. As mentioned in Subsection 3.5.2, ensembles to deal with concept drift can combine a subset of the following strategies: (*i*) adaptation of the models' combination weights; (*ii*) adaptation of the models' parameters; (*iii*) and/or add new models or exclude models. Ensembles can be classified as *batch-based* or *sample-based* if they are adapted when a batch of data or a sample is available, respectively. Unfortunately, most ensembles to deal with concept drift are batch-based and focus on classification tasks [Chu and Zaniolo, 2004]. Adaptations on a batch basis usually require a long time, and batch data may not reflect the current state of the system. However, even if the system is adapted on a sample basis, existing algorithms may adapt slowly to changes and cannot conciliate old and new information [Oza and Russell, 2001; Lan *et al.*, 2009], in scenarios in which changes may recur.

As described in Subsection 3.5.2, Learn^{++}.NSE [Elwell and Polikar, 2011] is a batch-based ensemble, inspired by the Boosting [Drucker, 1997], and it can conciliate old and new knowledge. The combination weight of each model is assigned using a weighted average of its errors on the current and old batches by a sigmoid function with two slope parameters. However, the parameters' setting is not an easy task, since Learn^{++}.NSE is sensitive to their values. AddExp [Kolter and Maloof, 2005] is a sample-based ensemble which uses a loss bound to obtain the models' errors, and models' combination weights are adapted according to the models' actual losses and a decreasing factor, employed to reduce a model's combination weight when it performs poorly. ILLSA is an ensemble for SS applications [Kadlec and Gabrys, 2011]. On the training phase, a set of models is designed, where each model is trained with samples of a different concept contained on the training data; while on the on-line phase, for each incoming sample, the models' combination weights are dynamically adapted using a Bayesian framework. One drawback is that few models are designed if the training data contains few concepts; and new models are not launched on the on-line phase. Unfortunately, most SS applications using ensemble systems do not add and remove models over time [Ge and Song, 2014; Lv *et al.*, 2013; Fortuna *et al.*, 2009], which makes difficult the system adaptation to process changes.

This Chapter proposes an on-line weighted ensemble of regressor models (OWE) which is able to learn incrementally sample by sample in the presence of several types of changes and simultaneously retain old knowledge in recurring scenarios. OWE is inspired by Learn^{++}.NSE [Elwell and Polikar, 2011]. But unlike Learn^{++}.NSE, in the OWE, the ensemble is adapted on a sample basis, leading the system to faster recovery from changes and increasing the system accuracy. Additional and new strategies are proposed to increase the OWE's accuracy. The experiments indicate that OWE outperforms Learn^{++}.NSE in all tests.

The key idea is to keep a fixed-size SW that slides along data when a new sample is available. Then, the error of each model on the current window is determined using a Boosting strategy [Feely, 2000; Shrestha and Solomatine, 2006] that assigns small errors to the models that predict accurately the samples predicted poorly by the ensemble. To handle recurring and non-recurring changes, OWE uses a new method for assigning the models' combination weights that takes into account the models' errors on the past and recent windows using a discounting factor that decreases or increases the contribution of old windows. OWE launches new models if the system's accuracy is decreasing, and it can remove inaccurate models for reducing memory and computational time. Experiments on artificial data sets and industrial data sets are detailed to evaluate, and demonstrate the performance and the effectiveness of the OWE over the state-of-the-art concept drift approaches.

The main contributions of OWE and of this Chapter are: (1) a new on-line sample-based ensemble of regressor models which can conciliate old and new knowledge using a discounting factor; (2) dynamic removal and inclusion of models (while, most ensembles for SS applications do not perform these tasks); (3) regression scope (while most ensemble applications for handling concept drifts are devoted to classification tasks); (4) thorough analysis of the experimental results using both artificial data sets and industrial data sets, demonstrating faster adaptation capability and accuracy of the OWE over the main state-of-the-art approaches; and (5) implementation of a new Learn^{++}.NSE algorithm for regression tasks.

This Chapter is organized as follows. Section 5.2 describes the OWE. In Section 5.3 the results are presented and discussed. Section 5.4 presents the concluding remarks.

5.2 On-line Weighted Ensemble of Regressor Models

This Section details the On-line Weighted Ensemble of Regressor Models (OWE) algorithm. OWE incorporates all the main adaptive mechanisms to deal with the problem of concept drift (Subsection 3.5.2); that is: ($A1$) *instance selection*, ($A2$) *instance weighting*, and ($A3$) *ensemble learning*. OWE employs the common assumption that the most recent data provides the best and most relevant representation of the current concept and near-future concept; and only this input data should be kept [Brzezinski and Stefanowski, 2014; Klinkenberg, 2005] (but the ensemble model keeps information about other past concepts). For this purpose, a fixed-size SW is used to keep the most recent set of samples ($A1$). These samples are employed to obtain the ensemble's accuracy based on the prediction errors, and to train a new model. Additionally, OWE also incorporates instance weighting mechanisms ($A2$) based on Boosting [Feely, 2000; Shrestha and Solomatine, 2006]. That is, a weighted distribution of the ensemble's error on the current window is obtained, and then the error of each model is calculated based on its contribution to the ensemble. This contribution is seen as the ability of a model to predict accurately the samples poorly predicted by the ensemble.

Boosting algorithms were firstly developed for solving binary classifications problems. Freund and Schapire [1997] proposed the first regression boosting algorithm called *AdaBoost.R*. The main idea is to map each regression sample into an infinite set of binary classification samples. Although it has theoretical proof of its convergence, the number of classification samples grows linearly in each iteration, hindering its practical application. Drucker [1997] proposed the *AdaBoost.R2* algorithm, a modification of the *AdaBoost.R*, that has promising results. *AdaBoost.R2* uses loss functions to convert regression loss into the domain of classification loss. Big Error Margin (BEM) boosting [Feely, 2000] is quite similar to *AdaBoost.R2*. However, BEM is less sensitive to noise and the system can handle weak learners with larger errors. In BEM, the absolute predictive error of a sample is compared to a predefined threshold, so that the corresponding sample is demarcated as incorrect or correct. The absolute error has a problem when the variability of the magnitude of real values is very high. To overcome this drawback, in OWE, absolute relative error (ARE), which adapts to the magnitude of the real value, is employed as the er-

ror measure, as in the AdaBoost.RT algorithm [Shrestha and Solomatine, 2006]. In OWE, a threshold is used to demarcate an incorrectly or correctly predicted sample based on the ARE.

OWE is an ensemble learning ($A3$) algorithm that takes into account that the data exhibits time-varying behavior. The main adaptive ensemble mechanisms used in OWE for dealing with concept drift are: (i) adaptation of the models' combination weights (with respect to their contributions on the recent and old windows); (iii) dynamic inclusion of models when the ensemble's performance is degrading; and removal of models over time. Adaptation of the models' parameters (ii) is not employed for old models to retain information about past scenarios, so that the ensemble can perform well in old and past scenarios.

The Steps of the OWE algorithm are detailed in Algorithm 5.1. The inputs of the algorithm are: a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t) \,|\, \mathbf{x}_t \in \mathbb{R}^{r\times 1}, y_t \in \mathbb{R}, t = 1, \ldots, T\}$, where $\mathbf{x}_t$ is a vector of r inputs, y_t is the output and $(\mathbf{x}_t, y_t)$ is a sample; the window's size, m; a factor for demarcating correct and incorrect predictions, θ; a factor for controlling the inclusion of a new model, α; a discount factor, κ; where $0 < \theta, \alpha < 1$ and $0 \leqslant \kappa < 1$; a generic supervised learning algorithm for regression, Weak Learner; a pruning activation factor, ρ; and the maximum number of models, N (enforced if and only if (iff) ρ is activated). In Step 1, the initialization of some variables is done. Variable k denotes the number of models in the ensemble, f_k denotes the most recently designed model, and t is the time step. $\mathbf{D}^t$ is a data window (of size m) produced at time t.

The penalty distribution (or window data weights), $\mathcal{D} = [d_1, \ldots, d_m]^T$, holds the weights of each sample on the current data window, where each weight is based on the ensemble prediction accuracy on a sample. $\mathcal{D}$ is firstly initialized to be a uniform distribution $[1/m, \ldots, 1/m]^T$. The penalty distribution is not employed during the model training as instance/sample weighting, e.g. for resampling the training data set. It is considered that all the training samples have the same weight/contribution during the training process.

Step 2 is repeated when a new sample from data set $\mathbf{D}$ becomes available. The data window $\mathbf{D}^t$ is firstly filled with the first m samples of $\mathbf{D}$, if t is not greater than m (Step 2(a)ii). Otherwise, the window slides along $\mathbf{D}$ (Step 2(a)i). In Step 2(b), the algorithm is directed to create the first model if t is equal to m. In Step 2(c), the ensemble $F(\cdot)$ is used to predict the new sample. The ensemble's output

Algorithm 5.1 On-line weighted ensemble of regressor models (OWE).

Inputs: a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$; window's size m; factor for demarcating correct and incorrect predictions θ; factor for controlling the inclusion of a new model α; discounting factor κ; a supervised learning algorithm, Weak Learner; pruning activation factor ρ; maximum number of models N (enforced if ρ is activated);

1. **Initialization:** Set $s = 1$; Number of models $k = 0$; $\mathbf{D}^0 \leftarrow \emptyset$; $\mathcal{D} = [d_1, \ldots, d_m]^T$, with $d_i = 1/m$, for $i = 1, \ldots, m$;
2. **for** $t = 1, \ldots, T$:
   (a) **if** $(t > m)$
      i. **then** Slide the window: $s \leftarrow s + 1$; $\mathbf{D}^t = \mathbf{D}^{t-1} + (\mathbf{x}_t, y_t) - (\mathbf{x}_{t-m}, y_{t-m})$;
      ii. **else** Fill the window: $\mathbf{D}^t = \mathbf{D}^{t-1} + (\mathbf{x}_t, y_t)$;
   (b) **if** $(t = m)$ **then** Go to Step 2(h)i; **else if** $(t < m)$ **then** Go to Step 2;
   (c) Predict y_t as: $F(\mathbf{x}_t) = \left(\sum_{n=1}^{k} \log(1/\Psi_n)\, f_n(\mathbf{x}_t)\right) / \sum_{n=1}^{k} \log(1/\Psi_n)$;
   (d) Obtain the error of the ensemble on $(\mathbf{x}_t, y_t)$: $\mathrm{ARE}_t^F = |(F(\mathbf{x}_t) - y_t)/y_t|$;
   (e) Count the number of samples incorrectly predicted by the ensemble: $totalSamples = \sum_{i=s}^{t} [\![\mathrm{ARE}_i^F > \theta]\!]$; where for any condition ϕ, $[\![\phi]\!] = 1$ if ϕ is true, and $[\![\phi]\!] = 0$ if ϕ is false;
   (f) Calculate *upFactor* and *downFactor*: $upFactor = m/totalSamples$, and $downFactor = 1/upFactor$;
   (g) Update and normalize the distribution $\mathcal{D}$:
      i. Set $d_i = 1/m$, for $i = s, \ldots, t$;
      ii. **if** $(\mathrm{ARE}_i^F > \theta)$, **then** $d_i \leftarrow d_i \times upFactor$, **else** $d_i \leftarrow d_i \times downFactor$, for $i = s, \ldots, t$;
      iii. $d_i^{(new)} \leftarrow d_i / \sum_{i=s}^{t} d_i$, for $i = s, \ldots, t$; $d_i \leftarrow d_i^{(new)}$, for $i = s, \ldots, t$;
   (h) **if** $|(F(\mathbf{x}_t) - y_t)/y_t| > \alpha$
      i. **then** Set $k \leftarrow k + 1$; set $\tau_k = 0$; Call Weak Learner and obtain a new model f_k trained with $\mathbf{D}^t$; Obtain the error of f_k on $\mathbf{D}^t$ as $\mathrm{ARE}_i^k = |(f_k(\mathbf{x}_i) - y_i)/y_i|$, for $i = s, \ldots, t$; Include f_k into the ensemble; **if** $(k = 1)$, **then** $\mathrm{ARE}_i^F \leftarrow \mathrm{ARE}_i^k$ for $i = s + 1, \ldots, t$;
   (i) Evaluate the models using $\mathbf{D}^t$ and obtain their current error rates using $\mathcal{D}$, (for $n = 1, \ldots, k$):
      $\mathrm{ARE}_t^n = |(f_n(\mathbf{x}_t) - y_t)/y_t|$;
      $\tau_n \leftarrow \tau_n + 1$; $\varepsilon_n^{\tau_n} = \sum_{i=s}^{t} d_i [\![\mathrm{ARE}_i^n > \theta]\!]$;
   (j) Obtain the total error rate of each model on the past and recent windows using κ:
      $\Psi_n = \left(\sum_{p=1}^{\tau_n} \kappa^{\tau_n - p} \varepsilon_n^p\right) / \sum_{p=1}^{\tau_n} \kappa^{\tau_n - p}$, for $n = 1, \ldots, k$;
   (k) **if** $(\rho = 1)$ **and** $(k > N)$, **then**
      Exclude the model f_n where $n = \mathrm{argmax}_{q=1,\ldots,k-1}(\Psi_q)$, and set $k \leftarrow k - 1$; and
      Renumber models: $\sigma_q \leftarrow \sigma_{q+1}$, for $\sigma_q \in \{f_q, \tau_q, \varepsilon_q^p, \Psi_q\}$, $q = n, \ldots, k$, $p = 1, \ldots, \tau_q$;
3. **end for**

is obtained by a weighted sum of the models' outputs using a (natural) logarithm function. Step 2(d) obtains the ARE of the most recent sample $(\mathbf{x}_t, y_t)$ predicted by the ensemble. Each error ARE_t^F of a sample $(\mathbf{x}_t, y_t)$ is obtained using the real output value y_t, and the predicted output of the ensemble $F(\mathbf{x}_t)$.

In Step 2(e), samples of $\mathbf{D}^t$ are demarcated as incorrectly or correctly predicted by the ensemble using a threshold θ. The objective is to count the total number of samples incorrectly predicted by the ensemble for obtaining the new penalty distribution. OWE employs concepts similar to the BEM boosting [Feely, 2000] and to the AdaBoost.RT [Shrestha and Solomatine, 2006] boosting regression algorithms. The strategy works as follows: if an error ARE_i^δ (of a sample $i \in \mathbf{D}^t$ predicted by a component model, $\delta = 1, \ldots, k$, or the ensemble, $\delta = F$; Steps 2(e), 2(g)ii, 2(i)) is greater than θ, then sample i is demarcated as incorrect, otherwise as correct. As in the AdaBoost.RT and BEM algorithms, OWE is sensitive to the setting of θ. The methods perform well when θ is between 0 and 0.4.

Step 2(f) determines the values of variables *upFactor* and *downFactor* using *totalSamples*. The variable *upFactor* increases the weights of the samples predicted incorrectly by the ensemble and *downFactor* decreases the weights of the samples predicted correctly by the ensemble. In Step 2(g), the values of $\mathcal{D}$ are updated. First, in Step 2(g)i, distribution $\mathcal{D}$ is reinitialized to be uniform. In Step 2(g)ii, each weight d_i of a sample i is obtained. The main idea is to assign larger weights to the samples predicted incorrectly by the ensemble and small weights to the samples predicted correctly by the ensemble.

In Step 2(h), a new model is launched to the ensemble if the absolute relative error of the ensemble on the newest sample is greater than α. The new model is trained using all samples of $\mathbf{D}^t$. Additionally, the absolute relative errors of the new model with respect to all the samples of the current window $\mathbf{D}^t$ are obtained; where each error of sample i is given by ARE_i^k (for $i = s, \ldots, t$ and $i \in \mathbf{D}^t$). If the new model is the first generated model of the ensemble, then the obtained values are assigned to the absolute relative error values of the ensemble with respect to $\mathbf{D}^t$.

Then, all the models are evaluated based on their predictions of $\mathbf{D}^t$ and their contributions to the ensemble (using $\mathcal{D}$) in the Step 2(i). In summary, the prediction error of a model f_n on $\mathbf{D}^t$ is given by $\text{ARE}_s^n, \ldots, \text{ARE}_t^n$, where ARE_t^n is the model's error on the most recent sample and is obtained at each iteration of the algorithm. The current error rate, ε_n^t, of model n at sample t is calculated using the sum of the

weights (from $\mathcal{D}$) of the samples demarcated as incorrectly predicted by model f_n. Therefore, since the sum of the elements of $\mathcal{D}$ is 1, ε_n^t can assume values between 0 and 1. As mentioned before, large errors are given to the models that predict poorly the samples predicted incorrectly by the ensemble; and small errors are given to the models that predict correctly the samples predicted incorrectly by the ensemble. In this Step 2(i), the variable τ_n for each model f_n is incremented. It holds the total number of windows where model f_n has been evaluated.

The total error rate Ψ_n of each model f_n is calculated using a discounting factor that weights the model's errors on the past and recent windows (Step 2(j)). The discounting factor weights recent errors more heavily than old errors for obtaining the total error rate Ψ_n of a model f_n as:

$$\Psi_n = \frac{\Psi_{n,t}^{(1)}}{\Psi_{n,t}^{(2)}} = \frac{\sum_{p=1}^{\tau_n} \kappa^{\tau_n - p} \varepsilon_n^p}{\sum_{p=1}^{\tau_n} \kappa^{\tau_n - p}}, \tag{5.1}$$

where κ is the discounting factor with $0 \leqslant \kappa < 1$; ε_n^p denotes the error of model f_n on a window p; and let at sample t

$$\Psi_{n,t}^{(1)} = \sum_{p=1}^{\tau_n} \kappa^{\tau_n - p} \varepsilon_n^p, \tag{5.2}$$

$$\Psi_{n,t}^{(2)} = \sum_{p=1}^{\tau_n} \kappa^{\tau_n - p}. \tag{5.3}$$

Equation (5.1) can be rewritten as:

$$\Psi_n = \frac{1}{\sum_{p=1}^{\tau_n} \kappa^{\tau_n - p}} \left(\kappa^{\tau_n - 1} \varepsilon_n^1 + \ldots + \kappa^1 \varepsilon_n^{\tau_n - 1} + \kappa^0 \varepsilon_n^{\tau_n} \right). \tag{5.4}$$

The sequence of weights $\kappa^{\tau_n - p}$ is *decreasing* with the oldness of the window, $\tau_n - p$, so that $\kappa^{\tau_n - 1} \leqslant \ldots \leqslant \kappa^1 < \kappa^0$, as can be observed in the Figures 5.1 and 5.2. Also, smaller values of κ imply that lower weights are given to the errors on the old windows. This case works well in non-recurring drifts, since more importance is given to the current scenario/concept. Larger values of κ imply that larger weights are given to the errors on the old windows. This case performs well in scenarios with

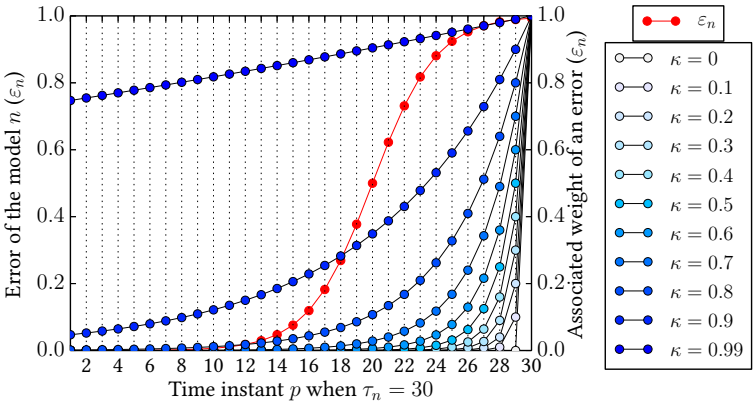


Figure 5.1: Error weighting of a model in OWE.

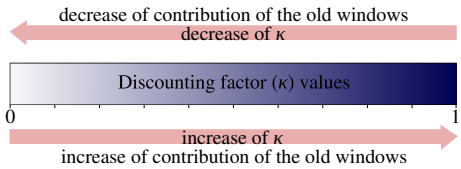


Figure 5.2: Discounting factor behavior in OWE.

recurring drifts, since more importance is given to the errors on the old windows. The proposed approach employs concepts of the discounted Mean Square Forecast Error (MSFE) method [Shen *et al.*, 2008]. It uses discounting factors for obtaining weights of each forecast in an ensemble system. This strategy is simple and easy to apply, and it is simple to tune the discount factor value. Other strategy is the Weighted Majority Algorithm (WMA) which combines a set of models using a weighted majority vote of the models' predictions [Littlestone and Warmuth, 1994]. When a model incorrectly classifies a sample, then WMA decreases its combination weight by a constant. Therefore, the combination weight is discounted only when it performs poorly on the newest sample. AddExp also employs the same concept. Learn^{++}.NSE employs a sigmoid function with two parameters to decrease or increase the contribution of the old batches. However, the parameters' setting is not an easy task, and Learn^{++}.NSE is sensitive to their values.

In Step 2(k), if the pruning factor ρ is activated by the user, then a model is removed if the number of models is greater than the threshold N. The pruning strategy removes the model with the largest total error rate Ψ_n. Note that a new model f_k created at sample-iteration t is never removed by the pruning strategy at

that same iteration t.

Using the value of τ_n at sample t, and from (5.2)-(5.3), at sample $t+1$, the computations of Equation (5.1), which are used to obtain Ψ_n in Step 2(j), can be recursively implemented with the following two recursions:

$$\Psi^{(1)}_{n,t+1} = \kappa\Psi^{(1)}_{n,t} + \kappa^0\varepsilon_n^{(\tau_n+1)}, \tag{5.5}$$

$$\Psi^{(2)}_{n,t+1} = \kappa\Psi^{(2)}_{n,t} + \kappa^0, \tag{5.6}$$

initialized (at the sample when model n is created) with $\Psi^{(1)}_{n,t} = \varepsilon_n^1$, and $\Psi^{(2)}_{n,t} = \kappa^0 = 1$. This implementation prevents the unbounded growth of the memory costs of saving ε_n^p and the computational costs related to the implementation of (5.1) for a single sample for the case of models which might never happen to be removed. Past values of ε_n^p do not need to be stored in memory, so that in Step 2(k) the values ε_n^p do not need to be updated/saved if in Step 2(j) $\Psi_n = \Psi^{(1)}_{n,t}/\Psi^{(2)}_{n,t}$ is computed using (5.1), (5.5)-(5.6).

5.3 Experimental Results

Experimental results are detailed in this Section to compare OWE to the state-of-the-art approaches. The approaches are evaluated on different scenarios using artificial data sets and industrial data sets. The use of artificial data sets allows the control of relevant parameters and to empirically evaluate the algorithms in several types of changes. There is a lack of artificial data sets to simulate changing environments for regression tasks. In this Chapter, it is used the hyperplane data set proposed by [Kolter and Maloof, 2005], a benchmark for evaluating algorithms that deal with concept drifts for both classification and regression tasks; and the drifting Friedman's function proposed by [Ikonomovska, 2012], a recent data set created for evaluating regression algorithms in changing environments. The real-world data sets enable us to evaluate the merit of the proposed approach in real-world problems, and compare it to the most recent works in real-life problems. However, it may not be possible to precisely state when drifts occur or if there is any drift at some specific time instant. This Chapter uses two well-known industrial data sets (previously described in Section 2.7) widely employed to evaluate algorithms for dynamic system modeling: the polymerization reactor data set, which has slowly changing process

dynamics due to the catalyst decay over the period of one year [Kadlec and Gabrys, 2011]; and the Fluidized Catalytic Cracking Unit (FCCU) data set, a benchmark for evaluating dynamic systems [Liu *et al.*, 2009]. The tests have been performed in the Matlab environment, running on a PC equipped with an Intel Core i7-2600 3.4GHz process of 4 cores and 8GB of RAM.

5.3.1 Data Set Description

Drifting Hyperplane Data Set. It is a well-known drifting data set used to evaluate algorithms that deal with concept drift [Minku *et al.*, 2010]. It contains noise, and gradual and non-recurring drifts, and is similar to the one proposed in [Kolter and Maloof, 2005] (AddExp). The whole data set consists of 10 inputs with uniform distribution over the interval $[0, 1]$, 1 output, $y_t \in [0, 1]$, and 2000 samples ($T = 2000$). The data set contains 4 concepts, where each concept holds 500 samples. The outputs of the data set are given by

- *Concept* 1: $y_t = (x_t^1 + x_t^2 + x_t^3)/3$, for $t = 1, \ldots, \frac{T}{4}$;
- *Concept* 2: $y_t = (x_t^2 + x_t^3 + x_t^4)/3$, for $t = (\frac{T}{4} + 1), \ldots, \frac{T}{2}$;
- *Concept* 3: $y_t = (x_t^4 + x_t^5 + x_t^6)/3$, for $t = (\frac{T}{2} + 1), \ldots, \frac{3T}{4}$;
- *Concept* 4: $y_t = (x_t^7 + x_t^8 + x_t^9)/3$, for $t = (\frac{3T}{4} + 1), \ldots, T$;

where x_t^z denotes the value of the input z of a sample t, where the sample input vector is $\mathbf{x}_t = [x_t^1, \ldots, x_t^r]^T \in \mathbb{R}^{r \times 1}$, r is the number of inputs, and $z = 1, \ldots, r$. A random variate noise uniformly distributed in the interval of $[-0.1, 0.1]$ is injected/added to each output sample y_t (for $t = 1, \ldots, T$). The value of y_t is clipped to 0 or 1 if its value is less than 0 or greater than 1, respectively.

Drifting Friedman's Function. As described in Chapter 4, the Friedman's function contains 5 input variables, $x^1, \ldots, x^5$, and 1 output variable, y_t:

$$y_t = 10 \sin(\pi x_t^1 x_t^2) + 20(x_t^3 - 0.5)^2 + 10x_t^4 + 5x_t^5 + \epsilon, \tag{5.7}$$

where $\epsilon \sim N(0, 1)$ is a zero-mean, unit-variance Gaussian random variable. The input space is enlarged by including other 5 input variables $x^6, \ldots, x^{10}$ that are not

relevant for predicting y. The input variables are uniformly distributed over the interval of $[0, 1]$. To create drifting scenarios, three drifting/artificial data sets using the Friedman's function were produced according to [Ikonomovska, 2012], each one with 2000 samples ($T = 2000$). After generating the input values for all samples of all data sets (with $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$), the output values of each data set are produced as follows.

1. *Local and abrupt drift data set* (Friedman-LA). Drifts in two different regions (local drift), R_1, and R_2, of the input space are introduced, where $R_1 = \{x^2 < 0.3 \wedge x^3 < 0.3 \wedge x^4 > 0.7 \wedge x^5 < 0.3\}$ and $R_2 = \{x^2 > 0.7 \wedge x^3 > 0.7 \wedge x^4 < 0.3 \wedge x^5 > 0.7\}$. The data set contains 3 points of abrupt drifts. The output values are obtained as:

   - Obtain the output y_t using (5.7), (for $t = 1, \ldots, \frac{T}{4}$);
   - (1^{st} *point*) **if** $\mathbf{x}_t \in R_1$, **then** Obtain y_t as $y_t^{R_1} = 10x_t^1x_t^2 + 20(x_t^3 - 0.5) + 10x_t^4 + 5x_t^5 + \epsilon$, **else** Obtain y_t using (5.7), for $t = \left(\frac{T}{4} + 1\right), \ldots, \frac{T}{2}$;
   - (2^{nd} *point*) **if** $\mathbf{x}_t \in R_2$, **then** Obtain y_t as $y_t^{R_2} = 10\cos(x_t^1x_t^2) + 20(x_t^3 - 0.5) + e^{x_t^4} + 5(x_t^5)^2 + \epsilon$, **else** Obtain y_t using (5.7), for $t = \left(\frac{T}{2} + 1\right), \ldots, \frac{3T}{4}$;
   - (3^{rd} *point*) remove the inequalities $x^4 > 0.7$ and $x^4 < 0.3$ from R_1 and R_2, respectively; **if** $\mathbf{x}_t \in R_1$, **then** Obtain y_t using $y_t^{R_1}$, **else if** $\mathbf{x}_t \in R_2$, **then** Obtain y_t using $y_t^{R_2}$, **else** Obtain y_t using (5.7), for $t = \left(\frac{3T}{4} + 1\right), \ldots, T$.

2. *Global recurring abrupt drift data set* (Friedman-GRA). It simulates global, abrupt and recurring drifts in two drift points of the data. The output values are produced as follows:

   - Obtain the output y_t using (5.7), for $t = 1, \ldots, \frac{T}{2}$;
   - (1^{st} *point*) Obtain the output y_t as $y_t = y_t^{glr_1} = 10 \sin(\pi x_t^4 x_t^5) + 20(x_t^2 - 0.5)^2 + 10x_t^1 + 5x_t^3 + \epsilon$; for $t = \left(\frac{T}{2} + 1\right), \ldots, \frac{3T}{4}$;
   - (2^{nd} *point*) Obtain the output y_t using (5.7), for $t = \left(\frac{3T}{4} + 1\right), \ldots, T$.

3. *Global non-recurring gradual drift data set* (Friedman-GnRG). It is produced by gradually introducing samples which belong to a different function in contrast to the samples of the initial function. Two points are introduced to simulate gradual changes. After each point, samples of an old concept are

gradually replaced by samples of a new concept by increasing their probability of being included in the data. This thesis considers that the probability of acceptance depends on a sigmoid function, and a random value uniformly distributed over the $[0, 1]$ interval. The output values are produced as follows, where $p = 0.02$, and $q = M/8$:

- Obtain the output y_t using (5.7), for $t = 1, \ldots, \frac{T}{2}$;
- (1^{st} *point*) **if** $random[0, 1) \leqslant (1+e^{-p(t-\frac{T}{2}-q)})^{-1}$, **then** obtain y_t using the function $y_t^{glr_1}$, **else** obtain y_t using (5.7), for $t = \left(\frac{T}{2} + 1\right), \ldots, \frac{3T}{4}$;
- (2^{nd} *point*) **if** $random[0, 1) \leqslant (1 + e^{-p(t-\frac{3T}{4}-q)})^{-1}$, **then** obtain y_t using function $y_t = y_t^{glr_2} = 10 \sin(\pi x_t^2 x_t^5) + 20 (x_t^4 - 0.5)^2 + 10x_t^3 + 5x_t^1 + \epsilon$, **else** obtain y_t using $y_t^{glr_1}$, for $t = \left(\frac{3T}{4} + 1\right), \ldots, T$.

Polymerization Reactor Data Set. The aim is to predict the catalyst activity in a reactor (for more details see Section 2.7). The data set was preprocessed as described in Subsection 4.4.1. Thus, it contains 648 samples, 10 inputs, and 1 output.

FCCU Data Set. The aim is to estimate the gasoline, light diesel oil (LDO) and liquefied petroleum gas (LGP) concentrations in a refinery (for more details see Section 2.7). The data set has 104 samples, 6 inputs, and 3 outputs.

5.3.2 Approach Setup and Description

Experiments are performed by comparing OWE to the RPLS, AddExp, ILLSA, and Learn^{++}.NSE. RPLS is a widely used algorithm in on-line process modeling to adapt to process changes. Its popularity is motivated by its reduced computational time and computer memory requirements [Qin, 1998]; and by its robustness under collinearity, measurement error and high dimensionality of input space, which are common characteristics in most industrial data sets. AddExp and Learn^{++}.NSE are two well-known adaptive ensembles for dealing with concept drifts; they are suitable to compare OWE to adaptive sample-based ensembles (AddExp) and adaptive batch-based ensembles (Learn^{++}.NSE), respectively. On the other hand, ILLSA is a popular adaptive ensemble for soft sensing.

For the OWE, AddExp, and Learn^{++}.NSE algorithms, ensembles are designed with and without pruning strategy. Ensembles with pruning strategy are termed as *pruned*. For each pruned ensemble, the pruning strategy was selected according to the best result indicated by its authors. In the pruned AddExp, the model with the lowest combination weight is excluded. In the pruned Learn^{++}.NSE [Elwell and Polikar, 2009], the model n with the largest current error is excluded. In all the pruned algorithms, the maximum number of models in the ensemble (N) is set to 20. This choice was considered the best suitable for all the ensembles, since the maximum number of models usually varies between 15 and 30 [Elwell and Polikar, 2009; Kolter and Maloof, 2005; Nishida and Yamauchi, 2007], and the use of more models linearly increases the processing time of the experiments.

Either PLS or RPLS [Ahmed *et al.*, 2009] is used as base model in the methods studied in the experiments. The PLS was implemented using the SIMPLS algorithm [Jong, 1993] (described in Algorithm 3.3); while the RPLS was implemented using the recursive SIMPLS algorithm (proposed in Algorithm 3.10). The optimal number of latents ℓ is determined by 10-fold cross-validation using the sum of squared prediction errors between the real output and the predicted output.

The following structure is performed in all the algorithms (except ILLSA). Consider a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$ with T samples. The first model is designed using the first m samples from $\mathbf{D}$, while the other $(T - m)$ samples of $\mathbf{D}$ are grouped to form the on-line data to simulate an on-line scenario. Each method is evaluated using MSE, which is calculated using the predicted outputs and the real outputs from the on-line data. MSE is a widely used metric to evaluate models. It provides a quadratic loss function that penalizes larger errors.

The component-models of the AddExp were implemented using RPLS models, each one trained using the most recent m samples. The parameters are set based on the pilot studies from [Kolter and Maloof, 2005]: $\varpi = 0.5$ (factor for decreasing combination weights), $\psi = 0.1$ (factor for new model combination weight), and $\varphi = 0.05$ (factor for adding a new model). For more details, see Subsection 3.5.2. In the experiments, the ensemble is adapted on a sample basis. If a new model should be included at time step t, its training data is obtained as: $\mathbf{D}^t = \{(\mathbf{x}_i, y_i)\}_{i=t-m+1}^{t} \subset \mathbf{D}$. As the AddExp requires the output data to be normalized to the interval of $[0, 1]$, the outputs of all the data sets for all the methods are normalized to this interval.

As the Learn^{++}.NSE is an algorithm for classification tasks, a new scheme is here

proposed to adapt it for regression tasks. Learn^{++}.NSE was implemented using a boosting regression algorithm, the AdaBoost.RT [Shrestha and Solomatine, 2006]. Each time step of the Learn^{++}.NSE consists of a batch of samples (it can be viewed as the window's size). Therefore, the batches are considered to have size m. The Weak Learner is the SIMPLS. In Learn^{++}.NSE, the slope parameters of the sigmoid function are set according to the authors' suggestions [Elwell and Polikar, 2011]: $a = 0.5$, and $b = 10$,

The ILLSA was implemented according to the works [Kadlec and Gabrys, 2011; Miranda, 2012]. The ILLSA is simulated by dividing the data set $\mathbf{D}$ into two data sets: 30% of $\mathbf{D}$ is used as the training data set (the initial samples of $\mathbf{D}$), for building the pool of RPLS models; and 70% of $\mathbf{D}$ is used as an on-line data set (the ending samples of $\mathbf{D}$). Therefore, the on-line data set of ILLSA is different from the other methods. Here, even if only one concept is detected on the training data set to train one model, two models are designed to ensure that an ensemble is designed. ILLSA does not have an ensemble pruning strategy. The size of the initial window (n^{init}) is set as m. In each experiment, the optimal values of the kernel size (σ) and the kernel size for the adaptation masks (σ^{adapt}) are chosen by 10-fold cross-validation (using the training data) using values in the range of $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$.

The OWE and pruned OWE are implemented according to Algorithm 5.1, where the SIMPLS Algorithm 3.3 is used as the Weak Learner. Based on pilot studies, θ is set to 0.05 for the OWE, pruned OWE, and Learn^{++}.NSE; since in the tests, large values of θ produce unstable systems. For the pruned OWE, ρ is set to 1, and N is set to 20; and for the OWE, ρ is set to 0. For each experiment, the result is obtained by averaging 20 independent runs.

5.3.3 Analysis of OWE Parameters

The parameters' setting is discussed in this Subsection. In on-line ensembles, the frequency of adding new models may impact on the ensemble's performance, and the discounting factor κ can be tuned according to the data characteristics. Based on pilot studies, tests of the OWE algorithm are conducted by varying α from 0.01 to 0.1 in steps of 0.01 (α lower than 0.01 produces a very large number of models and increases the computational time, while α greater than 0.1 may produce inaccurate ensembles); varying κ from 0 to 0.95 (in steps of 0.05) and also using $\kappa = 0.99$

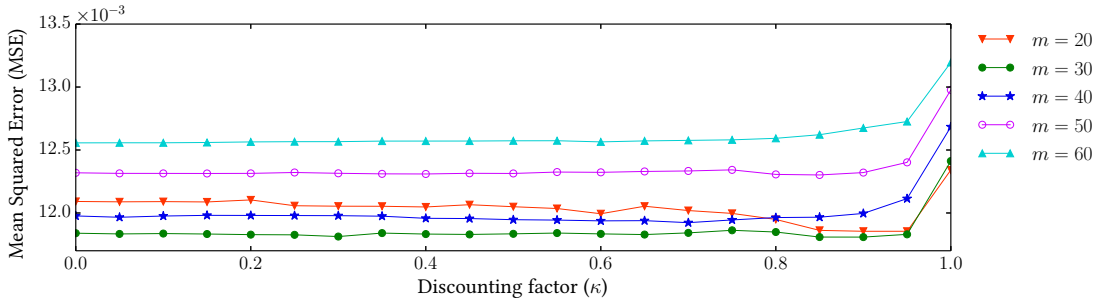
(its values range over all the interval of $0 \leqslant \kappa < 1$ for analyzing its behavior); and varying m in the following ranges:

- $m \in \{20, 30, 40, 50, 60\}$, for the artificial data sets;
- $m \in \{10, 15, 20, 25, 30\}$, for the industrial data sets.

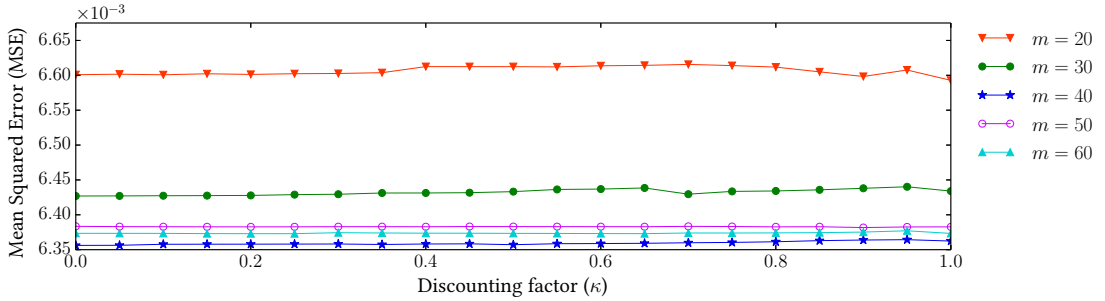
These ranges of m are sufficient to analyze the behaviors with the other parameters.

It has been observed that for the hyperplane data set and the Friedman-GnRG data set, the OWE's performance improves when α increases; while for the other artificial data sets, OWE has almost constant accuracy when α varies. For the industrial data sets, the accuracy of OWE slightly improves when α decreases. From the performed analysis it is seen that the most adequate α depends on the characteristics of each data set, i.e. in data sets that require faster adaptation capability, α should be set to a low value for including new models at a higher frequency. For data sets with small amounts of concept changes and that require low adaptation, α should be set to a large value for adding new models at a low frequency. In the tests below, α is set to 0.10 for the artificial data sets, and to 0.04 for the industrial data sets.

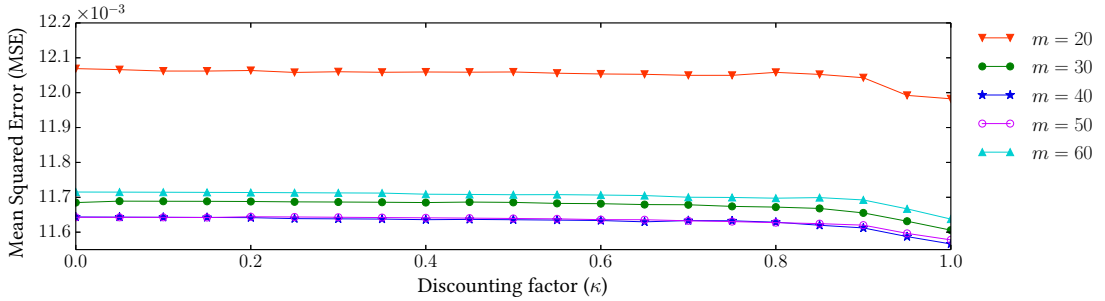
The MSE results of the OWE when κ varies using all the data sets are shown in Figure 5.3 and Figure 5.4. The tests show that for the hyperplane data set and Friedman-GnRG data set, OWE performs well when κ takes small values; and in this case, the MSE increases substantially when κ is 0.99. This happens because both data sets have non-recurring drifts and more importance should be given to the current concept. Therefore, the ensemble has better performance when the total error rate of each model is assigned by decreasing the contribution of the old window errors, and consequently, increasing heavily the contribution of the recent window errors. In contrast, on the Friedman-GRA data set, which is a recurring drift data set, OWE improves significantly its performance when κ is large, since the total error of the models takes more into account the errors on the old windows. For the Friedman-LA data set, OWE has almost constant accuracy when κ varies. In the polymerization reactor data set, it is observed that OWE performs well when κ is low; while in the FCCU data set, OWE oscillates its performance when κ varies. In the tests below, κ is set to 0.2 for all the data sets, except for the Friedman-GRA where κ is set to 0.99.



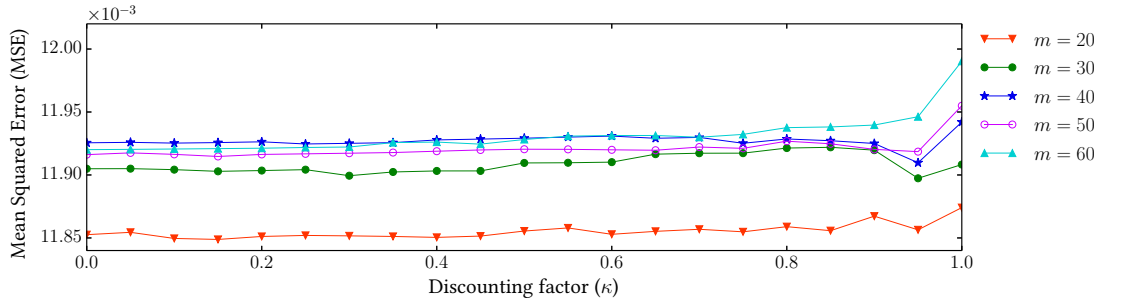
(a) Hyperplane data set.



(b) Friedman-LA data set.

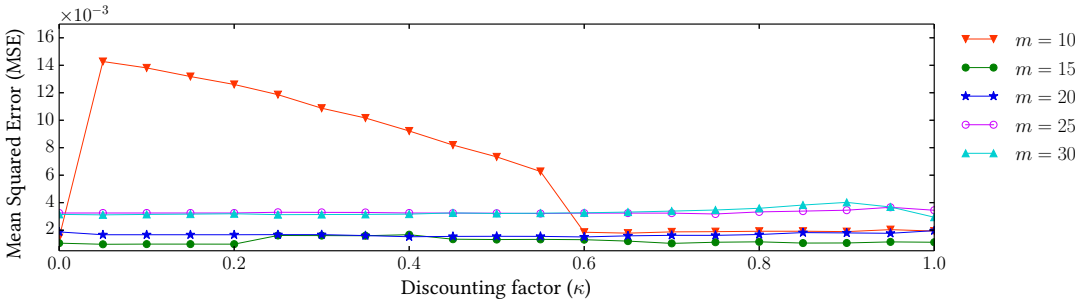


(c) Friedman-GRA data set.

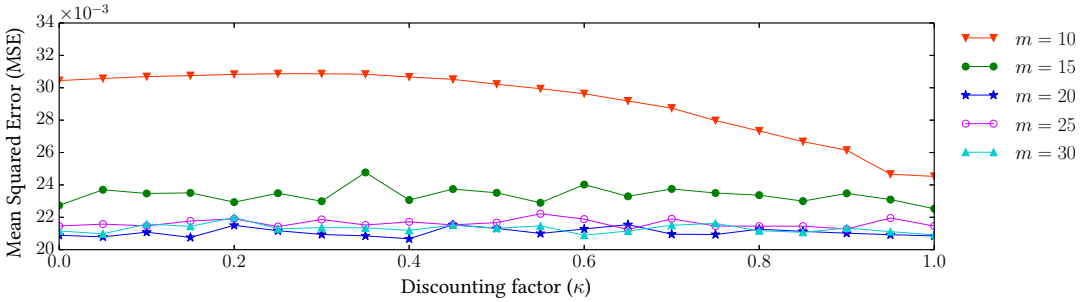


(d) Friedman-GnRG data set.

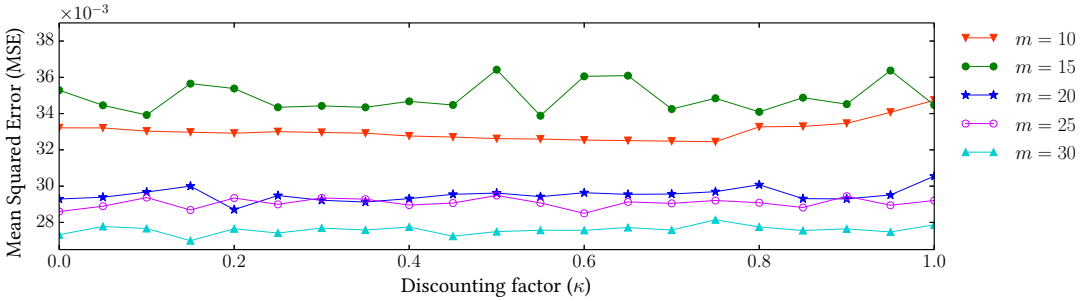
Figure 5.3: Artificial data sets: OWE's accuracy using different values of the discounting factor (κ).



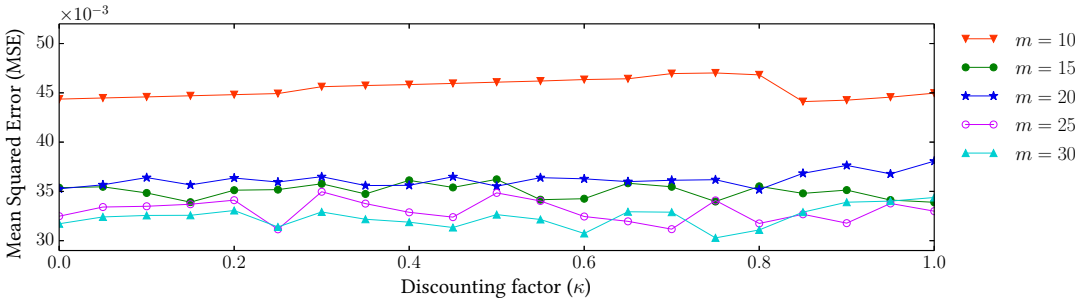
(a) Polymerization reactor data set.



(b) FCCU data set (gasoline concentration).

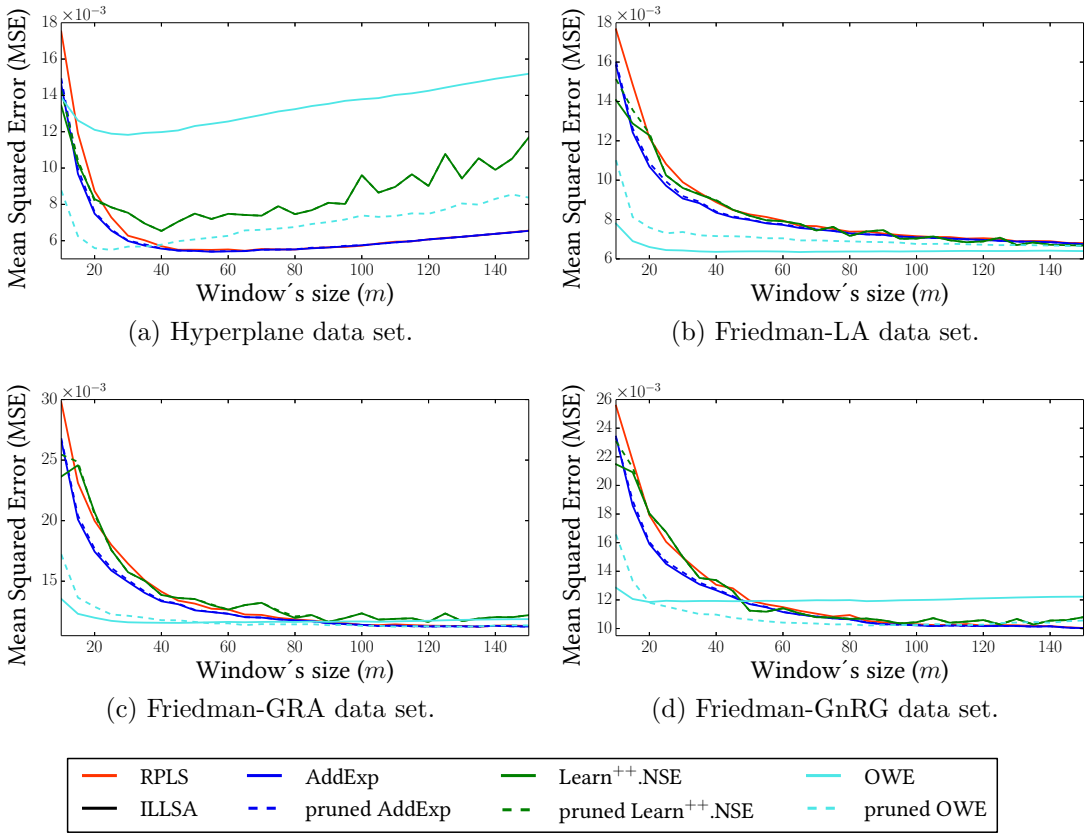


(c) FCCU data set (LDO concentration).



(d) FCCU data set (LPG concentration).

Figure 5.4: Industrial data sets: OWE's accuracy using different values of the discounting factor (κ).



(a) Hyperplane data set.

(b) Friedman-LA data set.

(c) Friedman-GRA data set.

(d) Friedman-GnRG data set.

Figure 5.5: MSE results of the approaches in the artificial data sets using different window's sizes (m).

5.3.4 Experimental Results Using Artificial Data Sets

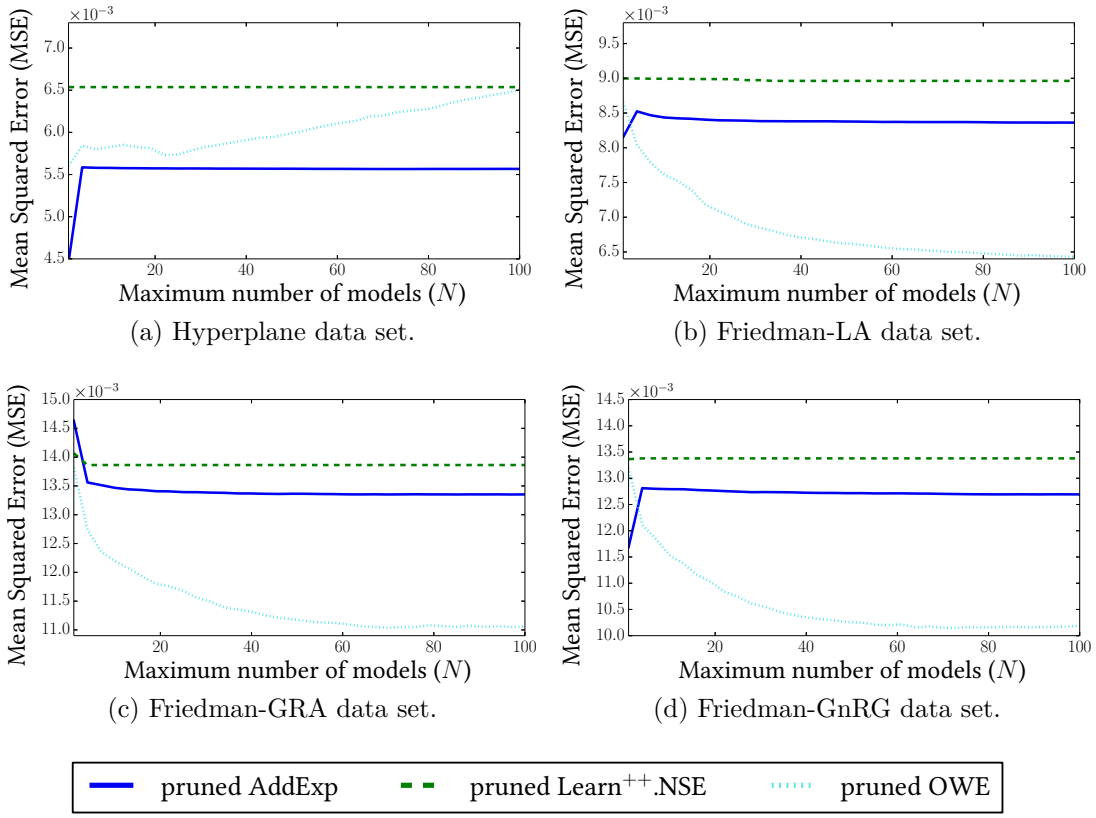
Results. In this Subsection, results of the algorithms using the artificial data sets are detailed and analyzed. Results of the ILLSA can be hidden in some figures due to their large errors when compared to the other methods, making difficult the analysis of the experiments. The first test aims to determine the impact of window's size m on the algorithms' accuracies. Figure 5.5 shows the algorithms' errors when m varies from 10 to 150 (in steps of 5). This range was chosen in pilot tests that indicated that, for all the approaches, the accuracies do not significantly improve when m is greater than 150 (see Figure 5.5). As can be observed, m is a factor that may influence the algorithms' accuracies. The results indicate different behaviors when m varies. But in most cases, the approaches' accuracies increase when m is

large. Figures 5.5b and 5.5c show that all the algorithms' errors do not significantly reduce after $m = 100$.

Table 5.1 shows the average and Standard Deviation (SD) of the MSE results of all the approaches on all values of m; bold font is used to indicate the best result in each data set. The MSE results reveal that OWE and pruned OWE have the best accuracies in most cases. The exception is in the hyperplane data set, where AddExp outperforms (6.34×10^{-3}) OWE (13.35×10^{-3}) and pruned OWE (6.95×10^{-3}) on average. In the Friedman-LA data set, OWE and pruned OWE have achieved the lowest errors (6.46×10^{-3} and, 7.11×10^{-3}, respectively); while other approaches have MSE greater than 8×10^{-3} on average. In the Friedman-GRA data set, OWE and pruned OWE also outperform (11.78×10^{-3}, and 11.81×10^{-3}, respectively) AddExp, pruned AddExp, RPLS, Learn^{++}.NSE, and pruned Learn^{++}.NSE (13.08×10^{-3}, 13.13×10^{-3}, 13.67×10^{-3}, 13.87×10^{-3}, and 13.91×10^{-3}, respectively) on average. In the Friedman-GnRG data set, pruned OWE has the lowest error (10.86×10^{-3}), while other methods have errors greater than 12×10^{-3}. OWE and pruned OWE have the lowest SD of the MSE when compared to the other approaches.

A test is applied to show the impact of the ensemble size (maximum number of models) on the algorithms' accuracies. Figure 5.6 shows the MSE errors of the pruned ensembles when the maximum number of models is increased (m is set to 40). For the drifting Friedman data sets, the Learn^{++}.NSE and the AddExp have constant accuracies when the maximum number of models increases; while the OWE tends to reduce the error when the maximum number of models increases. However, for the hyperplane data set, the OWE performs better when the maximum number of models is small. Table 5.2 shows some details of all the approaches on the artificial data sets. The values are calculated by obtaining the average and standard deviation of the running (computation) time and the number of models using different values of m for each algorithm. Learn^{++}.NSE and pruned Learn^{++}.NSE have the lowest running time among all the methods (including the single model RPLS). OWE and AddExp produce larger number of models when compared to ILLSA and Learn^{++}.NSE. It can be observed that the ILLSA produces fewer models than the other methods.

Discussion. ILLSA has achieved the largest error when compared to the other approaches. This is because, in ILLSA, each model is trained with a different concept of the training data set; and, if the training data contains few concepts (for example,



(a) Hyperplane data set. (b) Friedman-LA data set.

(c) Friedman-GRA data set. (d) Friedman-GnRG data set.

Figure 5.6: Pruned ensembles' errors (MSE) using the artificial data sets when the maximum number of models varies (for $m = 40$).

the Friedman-GRA data set, which contains one concept in the training data set), few models are built and they may be insufficient to deal with the dynamics of the data during the on-line phase. Regarding Learn^{++}.NSE and RPLS, they rarely outperform OWE and AddExp algorithms. In the Learn^{++}.NSE, the ensemble is adapted only when a new batch is available, requiring a long period of time for system adaptation. In contrast, sample-based ensembles (such as the OWE and the AddExp) have faster adaptation capability, since the ensembles are adapted on a sample basis. The RPLS is also adapted on a sample basis, but it employs few strategies to deal with concept drifts: the only strategy is to recursively include each new sample into the model at each time.

AddExp has good prediction performance when compared to the Learn^{++}.NSE and RPLS algorithms; and to the OWE for some values of m (mainly in the hyper-

Table 5.1: Artificial data sets: average and standard deviation of the MSE by varying the value of m.

Data set/Approach	*Hyperplane*	*Friedman-LA*	*Friedman-GRA*	*Friedman-GnRG*
RPLS	6.62 (2.46)	8.46 (2.53)	13.67 (4.24)	12.37 (3.72)
ILLSA	-	-	-	38.13 (9.90)
AddExp	**6.34 (1.80)**	8.07 (1.97)	13.08 (3.34)	11.87 (3.00)
Pruned AddExp	6.37 (1.87)	8.12 (2.03)	13.13 (3.42)	11.91 (3.04)
Learn^{++}.NSE	8.72 (1.63)	8.18 (1.94)	13.87 (3.46)	12.29 (3.15)
Pruned Learn^{++}.NSE	8.75 (1.73)	8.25 (2.13)	13.91 (3.73)	12.35 (3.35)
OWE	13.35 (1.06)	**6.46 (0.28)**	**11.78 (0.37)**	12.04 (0.19)
Pruned OWE	6.95 (0.96)	7.11 (0.82)	11.81 (1.17)	**10.86 (1.28)**

MSE values have been multiplied by 10^3;
m is varied from 10 to 150 (in steps of 5).

Table 5.2: Artificial data sets: average and standard deviation of the number of models and running time (minutes) of the approaches by varying the value of m.

Data set / Approach	*Hyperplane*		*Friedman-LA*		*Friedman-GRA*		*Friedman-GnRG*	
	n. of models	run. time	n. of models	run. time	n. of models	run. time	n. of models	run. time
RPLS	1.00 (0.00)	0.36 (0.04)	1.00 (0.00)	0.36 (0.02)	1.00 (0.00)	0.36 (0.06)	1.00 (0.00)	0.35 (0.01)
ILLSA	3.34 (1.78)	0.13 (0.10)	2.86 (2.03)	0.09 (0.08)	3.32 (2.84)	0.08 (0.09)	2.80 (2.48)	0.06 (0.05)
AddExp	1005.83 (87.81)	7.28 (0.48)	1023.48 (92.85)	6.90 (0.38)	1220.07 (100.34)	8.25 (1.14)	1186.28 (95.11)	7.76 (0.50)
Pruned AddExp	20.00 (0.00)	0.81 (0.03)	20.00 (0.00)	0.81 (0.01)	20.00 (0.00)	0.95 (0.20)	20.00 (0.00)	0.91 (0.12)
Learn^{++}.NSE	41.00 (41.68)	0.07 (0.12)	41.00 (41.68)	0.06 (0.07)	41.00 (41.68)	0.07 (0.12)	41.00 (41.68)	0.06 (0.06)
Pruned Learn^{++}.NSE	18.45 (2.47)	0.04 (0.02)	18.45 (2.47)	0.04 (0.02)	18.45 (2.47)	0.05 (0.05)	18.45 (2.47)	0.04 (0.04)
OWE	1201.90 (22.67)	3.52 (0.19)	792.48 (40.08)	2.52 (0.15)	1190.17 (39.96)	3.94 (0.40)	1144.10 (37.22)	3.47 (0.32)
Pruned OWE	20.00 (0.00)	0.61 (0.02)	20.00 (0.00)	0.52 (0.01)	20.00 (0.00)	0.74 (0.16)	20.00 (0.00)	0.70 (0.12)

m is varied from 10 to 150 (in steps of 5).

plane data set). AddExp outperforms RPLS since AddExp is a dynamic ensemble of models, while RPLS is composed of only one model. This shows that an ensemble is usually more accurate than any single model. AddExp outperforms Learn^{++}.NSE, because in the AddExp, the ensemble is adapted when a sample is available (rather than on a batch basis). AddExp has worse performance when compared to the OWE for the drifting Friedman data sets. In the AddExp, when a new sample is available, all the models are re-trained using such new sample. After some time, the models become very similar. This occurs because the models start to contain only information about the recent samples, since the recursive learning excludes the oldest samples. In this way, the old models start to lose information about the old scenarios. In recurring drifts, if the system becomes to lose information about the past concepts, the system takes more time to react to them when they recur.

In summary, the results in the data sets indicate that the OWE has better or comparable performance to the other state-of-the-art methods. The good accuracy of OWE is attributed to the development of a set of mechanisms. For example, OWE keeps a set of diverse models trained with different parts of the data so that when an old concept recurs, old models can be re-activated and the system performs well. Additionally, in the pruned OWE, the pruning strategy removes the model with the worst performance on the old and current windows. This strategy is important to maintain the ensemble's performance in recurring scenarios, since it reduces the probability of excluding good models that belong to old concepts. OWE dynamically launches new models if the ensemble's performance is poor on the newest sample. Furthermore, the models' combination weights are obtained by taking into account their accuracies on the recent and past windows.

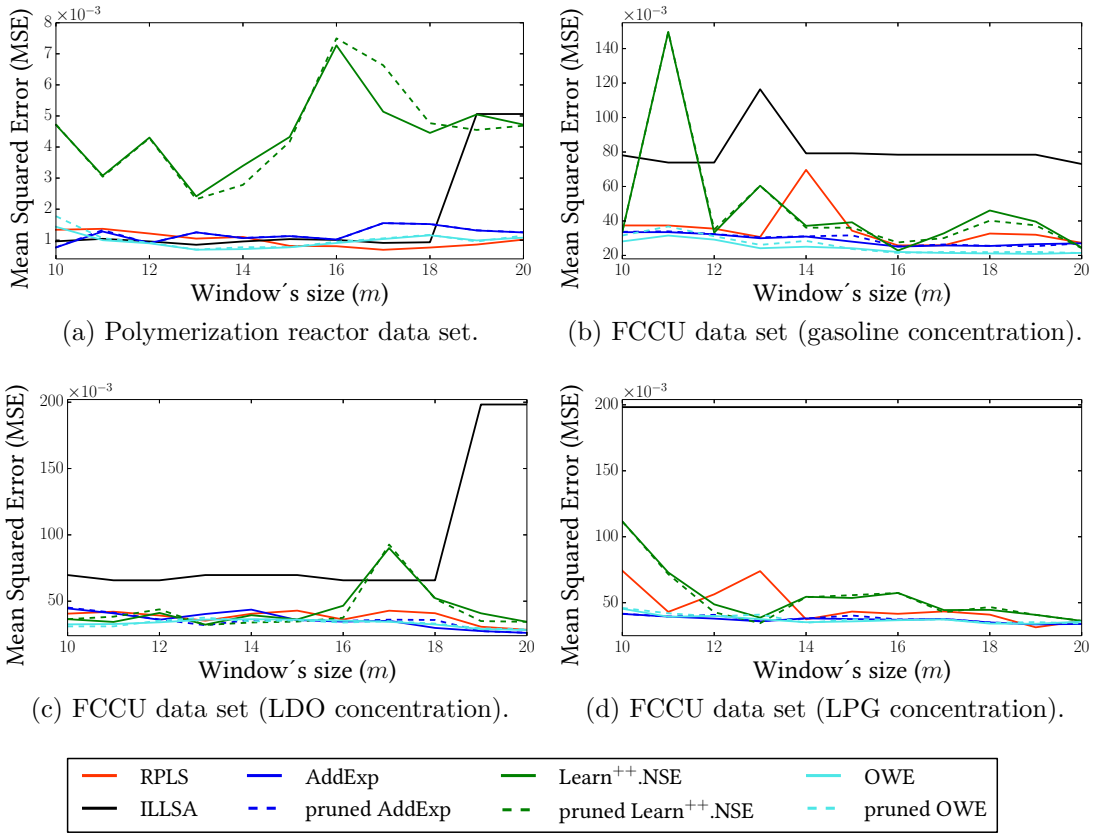
Common behaviors are observed in the experiments. For example, pruned ensembles outperform ensembles without pruning strategy in the non-recurring drifts; and ensembles without pruning strategy outperform pruned ensembles in the recurring drifts. In recurring drifts, since an old concept recurs, the pruning strategy may exclude models trained on an old concept. On the other hand, in non-recurring drifts, the pruning strategy can be seen as a way to remove redundant models and keep the most accurate set of models that maximize the performance on the current concept.

5.3.5 Experimental Results Using Industrial Data Sets

Results. Experimental results of the algorithms using industrial data sets are presented in this Subsection. The algorithms' accuracies when the window's sizes vary from 10 to 20 (in steps of 1) are shown in Figure 5.7. This range was chosen in pilot tests that indicated that, for all the approaches, the accuracies do not significantly improve when m is greater than 20. The test indicates that the OWE has the lowest error when compared to the other methods in most values of m. For the FCCU data set, most approaches achieve smaller errors for large m. Table 5.3 shows the average and standard deviation of the MSE results of all the algorithms for all values of m, where bold font indicates the best result in each data set. The MSE results indicate that OWE and pruned OWE have the best results in most data sets. In the polymerization reactor data set, OWE (0.97×10^{-3}) slightly outperforms the RPLS (1×10^{-3}) on average; while other approaches have MSE greater than 1×10^{-3}. In the gasoline concentration prediction (FCCU data set), OWE, pruned OWE, and AddExp have achieved the lowest errors on average, i.e. 24.48×10^{-3}, 26.05×10^{-3}, and 28.92×10^{-3}, respectively; while most of the other approaches have errors larger than 30×10^{-3}. In the LDO concentration estimation (FCCU data set), pruned OWE and OWE have the lowest errors, i.e. 33.23×10^{-3} and 33.41×10^{-3}, respectively; and other methods have errors larger than 35×10^{-3}. For the LPG concentration estimation (FCCU data set), AddExp has the best accuracy (i.e. 37.08×10^{-3}), followed by OWE (i.e. 37.39×10^{-3}). Figures from 5.8 to 5.11 show the estimated outputs of each algorithm on its best window's size m as evaluated by the MSE.

Figure 5.12 shows the performance of the pruned ensembles when N increases (m is set to 20). In most cases, it is observed that the error tends to decrease when N increases. Table 5.4 shows interesting details of all the approaches. The test shows that OWE produces more models than the other methods. However, in most cases, OWE has smaller running time when compared to AddExp. For the polymerization reactor data set, ILLSA has produced a larger number of models when compared to the artificial data sets, because this data set contains more concept changes in the training data set.

Discussion. It can be observed that ILLSA and the Learn^{++}.NSE are more sensitive to the value of m. The RPLS and ILLSA have achieved better accuracy in the polymerization reactor data set when compared to their performances in the



(a) Polymerization reactor data set.

(b) FCCU data set (gasoline concentration).

(c) FCCU data set (LDO concentration).

(d) FCCU data set (LPG concentration).

Figure 5.7: MSE results of the approaches in the industrial data sets using different window's sizes (m).

artificial data sets. Possibly, in the polymerization reactor data set, the dynamics of the process are well represented for designing the set of models in the ILLSA; and for training the model in the RPLS. The AddExp and the OWE algorithms outperform the Learn^{++}.NSE approach. As mentioned before, Learn^{++}.NSE uses batch learning and it takes longer time to adapt to the changes. In most cases, the tests indicate that the ensembles without pruning usually have equal or superior performance when compared to their pruned versions. Therefore, this result reveals that a larger number of models may lead to a better ensemble's accuracy. This performance may be related to the diversity among the models or because the data sets have recurring behavior and, consequently old models are necessary to be re-activated during predictions.

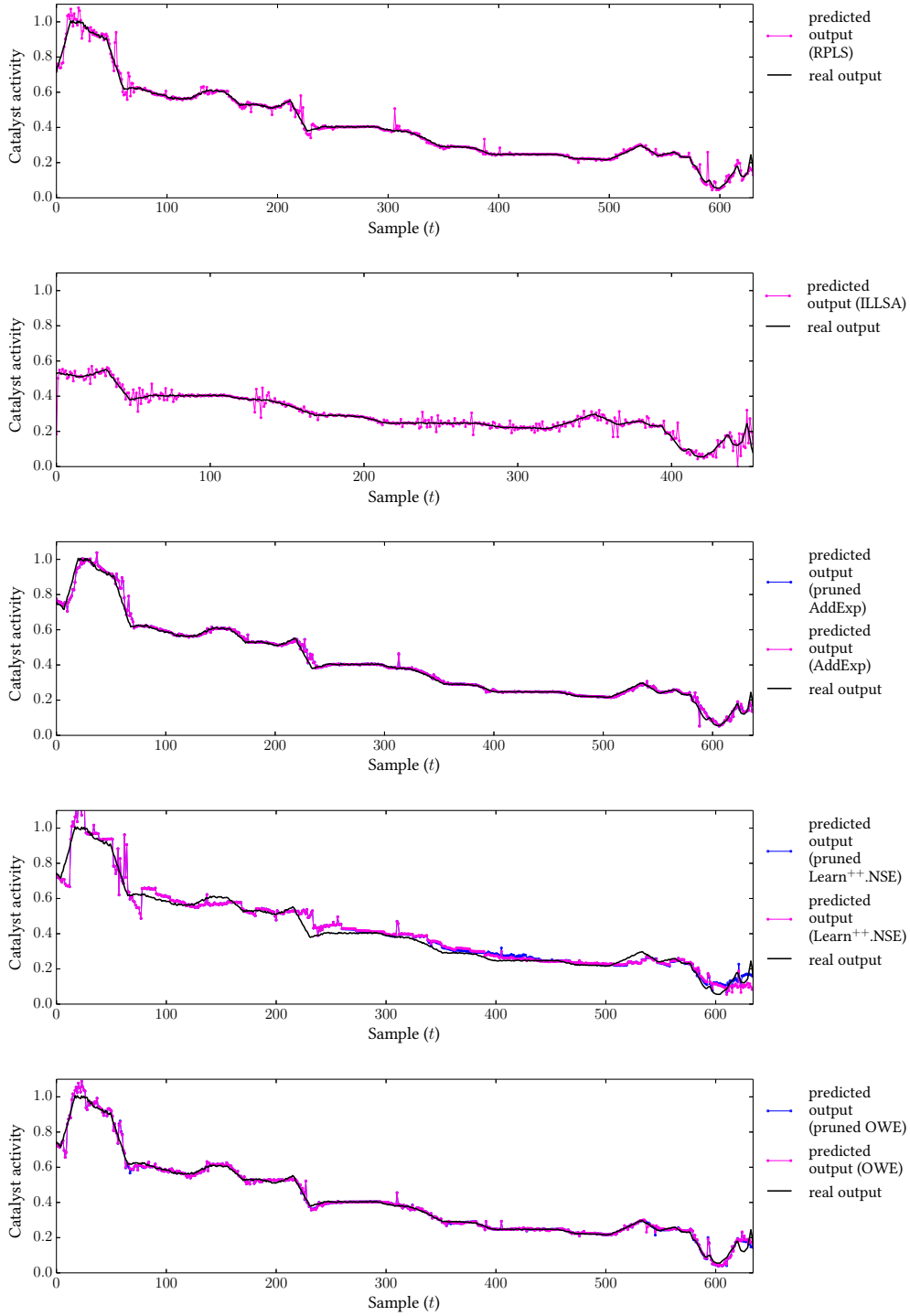


Figure 5.8: The predicted outputs of all the algorithms using the polymerization reactor data set.

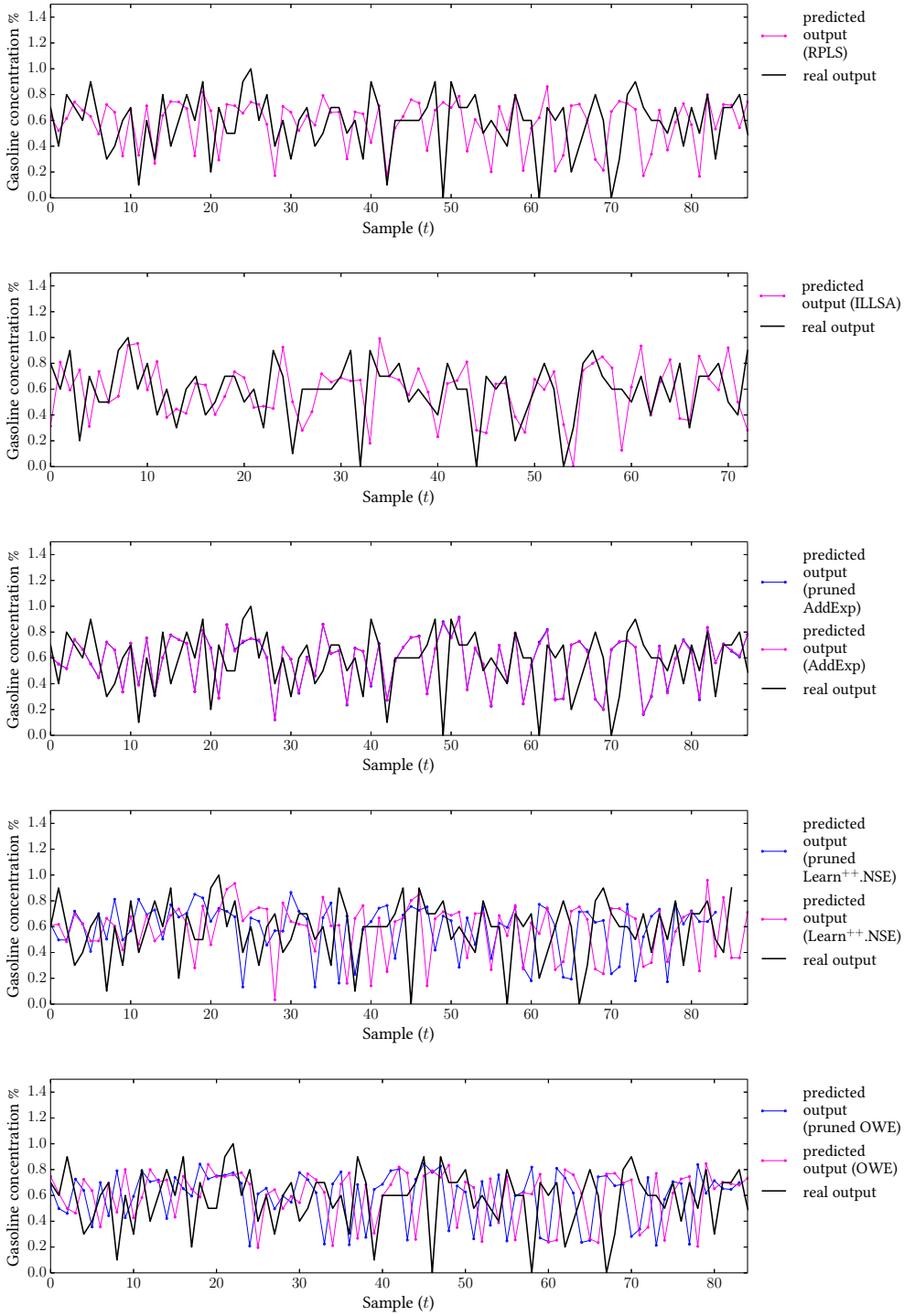


Figure 5.9: The predicted outputs of all the algorithms using the FCCU data set (gasoline concentration).

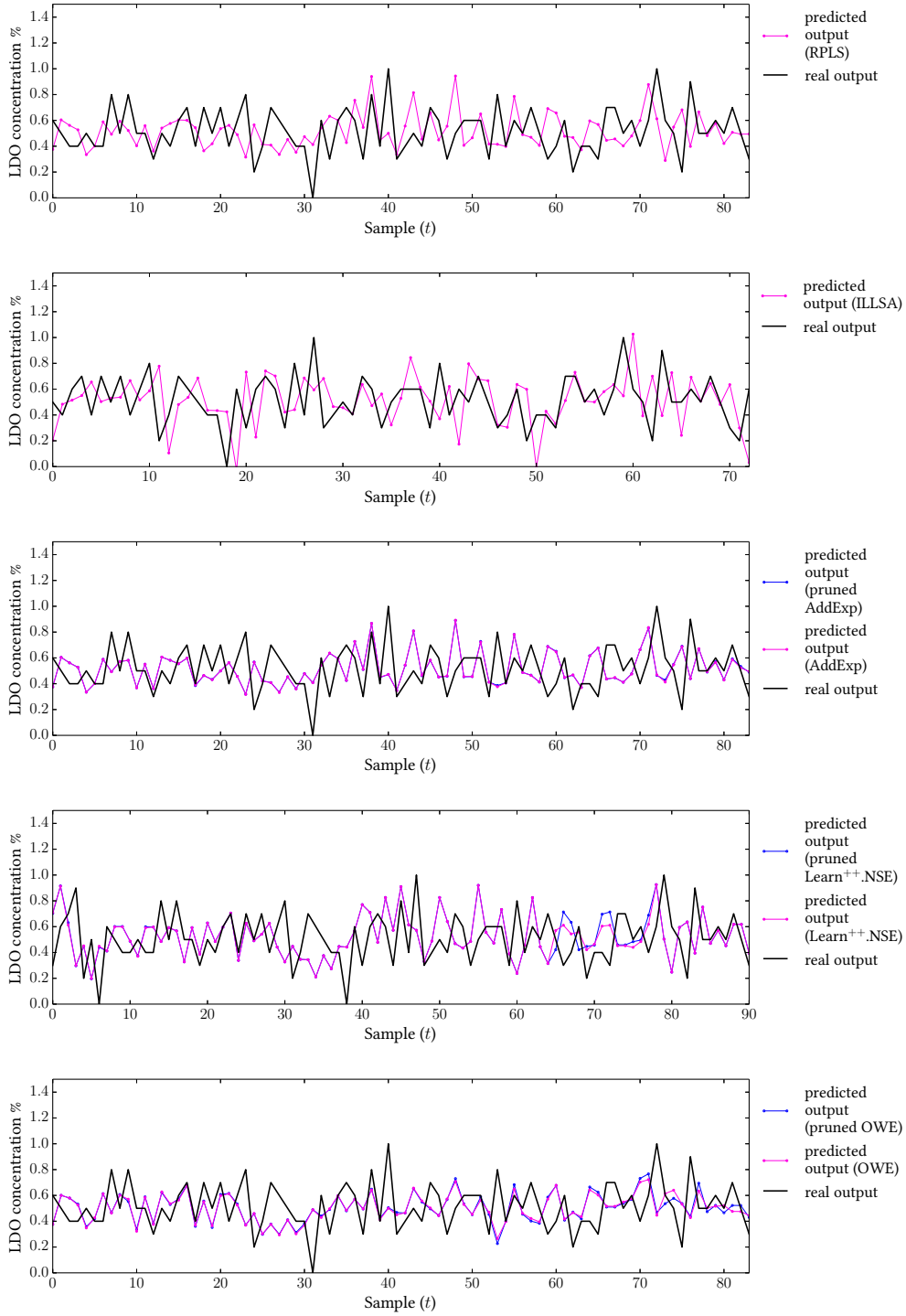


Figure 5.10: The predicted outputs of all the algorithms using the FCCU data set (LDO concentration).

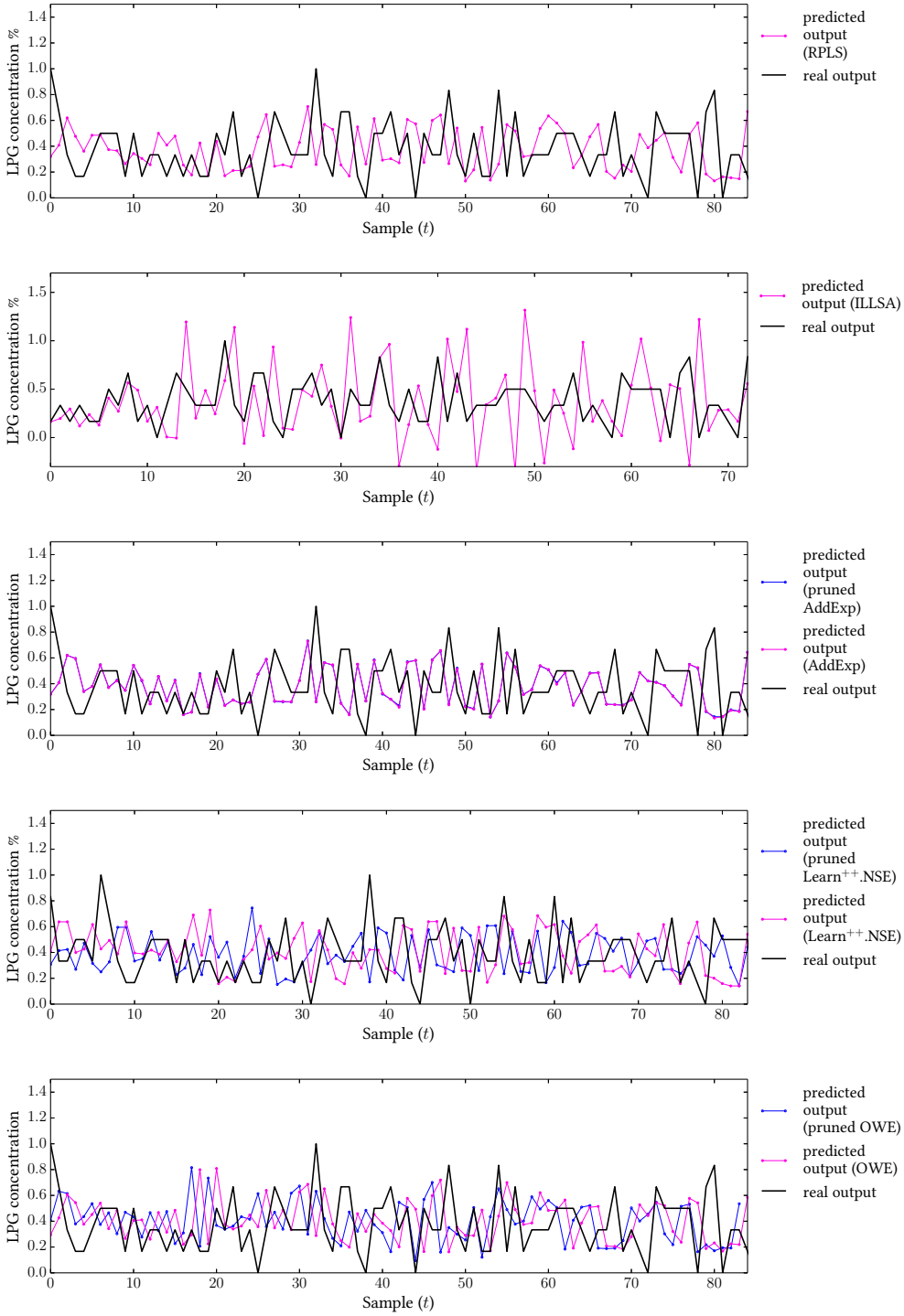
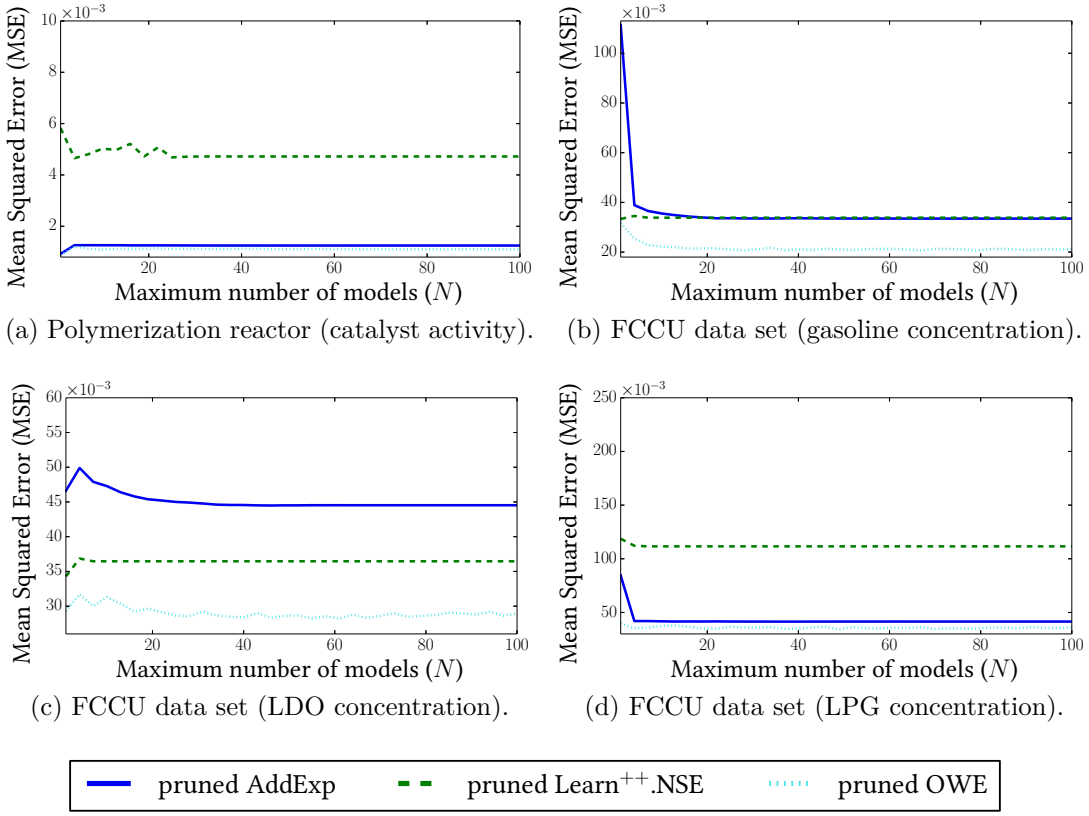


Figure 5.11: The predicted outputs of all the algorithms using the FCCU data set (LPG concentration).



(a) Polymerization reactor (catalyst activity).

(b) FCCU data set (gasoline concentration).

(c) FCCU data set (LDO concentration).

(d) FCCU data set (LPG concentration).

Figure 5.12: Pruned ensembles' errors (MSE) using the industrial data sets when the maximum number of models varies.

Table 5.3: Industrial data sets: average and standard deviation of the MSE by varying the value of m.

Data set/ Approach	*Polymerization reactor (catalyst activity)*	*FCCU (gasoline concentration)*	*FCCU (LDO concentration)*	*FCCU (LPG concentration)*
RPLS	1.00 (0.23)	35.33 (12.13)	38.16 (4.91)	47.43 (14.54)
ILLSA	1.71(1.66)	80.68(12.07)	91.28(52.93)	198.27(0.00)
AddExp	1.19 (0.24)	28.92 (3.23)	35.86 (6.20)	**37.08 (2.37)**
Pruned AddExp	1.19 (0.24)	29.35 (3.48)	35.22 (5.42)	37.65 (2.72)
Learn^{++}.NSE	4.44 (1.27)	47.23 (35.49)	44.02 (16.35)	54.86 (21.47)
Pruned Learn^{++}.NSE	4.50 (1.54)	46.42 (35.50)	42.79 (17.46)	54.03 (21.95)
OWE	**0.97 (0.21)**	**24.48 (3.66)**	33.41 (2.77)	37.39 (3.28)
Pruned OWE	1.03 (0.29)	26.05 (5.25)	**33.23 (2.80)**	38.12 (3.69)

The MSE values have been multiplied by 10^3;
m is varied from 10 to 20 (in steps of 1).

Table 5.4: Industrial data sets: average and standard deviation of the number of models and running time (minutes) of the approaches by varying the value of m.

Data set / Approach	*Polymerization reactor (catalyst activity)*		*FCCU (gasoline concentration)*		*FCCU (LDO concentration)*		*FCCU (LPG concentration)*	
	n. of models	run. time	n. of models	run. time	n. of models	run. time	n. of models	run. time
RPLS	1.00 (0.00)	0.182 (0.047)	1.00 (0.00)	0.019 (0.005)	1.00 (0.00)	0.021 (0.001)	1.00 (0.00)	0.017 (0.004)
ILLSA	31.45(15.40)	0.578 (0.234)	2.00 (0.00)	0.005 (0.002)	2.00 (0.00)	0.006 (0.000)	2.00 (0.00)	0.006 (0.001)
AddExp	44.64 (5.39)	0.205 (0.053)	73.27 (5.46)	0.065 (0.020)	70.36 (1.91)	0.071 (0.002)	72.36 (2.80)	0.064 (0.014)
Pruned AddExp	20.00 (0.00)	0.163 (0.048)	20.00 (0.00)	0.053 (0.014)	20.00 (0.00)	0.058 (0.001)	20.00 (0.00)	0.049 (0.013)
Learn^{++}.NSE	44.91 (10.38)	0.070 (0.018)	6.82 (1.72)	0.005 (0.002)	6.82 (1.72)	0.005 (0.001)	6.82 (1.72)	0.004 (0.001)
Pruned Learn^{++}.NSE	20.00 (0.00)	0.054 (0.013)	6.82 (1.72)	0.005 (0.002)	6.82 (1.72)	0.005 (0.001)	6.82 (1.72)	0.004 (0.001)
OWE	192.64 (17.87)	0.322 (0.090)	80.27 (3.61)	0.054 (0.014)	79.91 (3.75)	0.060 (0.002)	83.18 (3.84)	0.055 (0.012)
Pruned OWE	20.00 (0.00)	0.192 (0.052)	20.00 (0.00)	0.048 (0.014)	20.00 (0.00)	0.054 (0.002)	20.00 (0.00)	0.049 (0.012)

m is varied from 10 to 20 (in steps of 1).

5.4 Conclusion

In on-line applications, changes may happen over time and thus additional adaptive strategies are necessary to guarantee the ensemble performance in changing environments. The main contribution of the ensemble proposed in this Chapter, OWE, is the ability to learn incrementally sample by sample in presence of several types of changes, and retain old knowledge in recurring scenarios by using a discounting factor. Moreover, the proposed ensemble is adapted whenever a new sample is available, and thus it can achieve better performance than ensembles adapted only when a set of samples is available. The proposed ensemble also dynamically removes and adds models over time.

The methodology proposed in this Chapter was compared to RPLS, ILLSA, AddExp, and Learn^{++}.NSE using artificial data sets with concept drifts, and real-world industrial data sets. The results have shown that, in most experiments, OWE achieves better accuracy than other state-of-the-art methods, and in some cases, OWE has comparable accuracy to the other state-of-the-art approaches. RPLS assumes that samples that fall outside the moving window are irrelevant for the learning, and such method does not have capability to use the old acquired data, since the oldest samples are discarded. Other methods able to conciliate previous data and current data (e.g. Learn^{++}.NSE) may perform poorly since a long time is required for system adaptation.

In this Chapter, the tests show notable behaviors. Results show that, in most cases, ensemble learning outperforms learning using only one single model. The tests also show that OWE has capability to deal with the concept drifts. The analysis reveals that the frequency of including a new model to the ensemble (α), the contribution of old windows over new windows (κ), and the maximum number of models are important issues in on-line ensembles that deal with changing environments.

Other important issues are shown in this Chapter. In ensemble learning, the re-training of all models on the same data can produce very similar models. In this specific case, the ensemble loses information about the old scenarios, leading the ensemble to a poor accuracy in scenarios where an old concept can recur. In recurring drifts, ensembles without pruning strategies are usually more accurate than pruned ensembles. Since in recurring drifts old concepts can recur, the pruning strategy may remove important models trained on these old concepts. OWE monitors the

models' performances on the current and old windows so that when an old concept recurs, old and accurate models can be re-activated.

Despite the attractive characteristics of the OWE, its accuracy is related to the windows' size and the α value; since the prediction of a data set with time-varying behavior depends on the values of α and windows' size m. To cover these limitations, as a future work, a variable window size that adapts according to the process dynamics and an adaptive setting of α that is automatically adjusted according to the change of characteristics (e.g. when a change occurs, α would be set to a low value to include new models in a high frequency) can be proposed. Moreover, as a future work, the author suggests the introduction of other pruning strategies and methods to dynamically adjust other OWE's parameters over time.

Therefore, the objective of proposing an adaptive ensemble to deal with several types of changes and simultaneously retain old information in scenarios where changes may recur was reached in this Chapter. Tests with artificial data sets with different types of changes showed that the proposed ensemble learning algorithm is able to learn in changing environments and simultaneously retain old information in scenarios where changes may recur. The success of the proposed ensemble is mainly attributed to the use of the discounting factor that can increase or decrease the contribution of old models. The experimental results showed that, in most cases, the proposed methodology can estimate key variables in soft sensing applications more accurately than well-known adaptive ensemble methods.

Chapter 6

An Adaptive Ensemble with Fast Adaptation Capability

Contents

6.1 Introduction

Industrial plants are rather dynamic, being very difficult for the models to react to the changes, and thus leading to a deterioration of the model accuracy. The main reasons for such changes are the *sensor drift* and/or *process drift*. Sensor drift is a temporal shift of a sensor (which cannot be predicted or defined) due to aging or

environment changes [Vergara *et al.*, 2012]. Process drift is related to the changes of process behavior or to some external process conditions over time [Chincholkar and Herrmann, 2008]. Sensor drift and process drift are difficult to detect and handle, since many other factors may be changing the process conditions in parallel. In ML, all these drifting problems are summarized under the term concept drift [Tsymbal, 2004] (see Section 3.5.1).

To cope with all these effects, the development of predictive models with adaptive capability is necessary. Different strategies for on-line adaptation have been proposed in the literature. Recently, Just-in-time Learning (JITL) has been raising much attention in modeling industrial systems with time-varying behavior [Jin *et al.*, 2014]. JITL designs a local model using samples which are similar to a testing sample. Once the testing sample is predicted, the local model is discarded. The main drawback is the high computational time to continuously train a new model from scratch whenever a new sample is available. Other strategies are the recursive methods such as the RLS [Gjerkes *et al.*, 2011], the RPLS [Qin, 1998], or the OS-ELM [Liang *et al.*, 2006], where model's parameters are recursively adapted over time. Recursive methods perform well in cases where the process dynamics are well represented in the training data set. However, these methods are not efficient and sufficient to deal with new process dynamic characteristics occurring in new samples, becoming difficult to adapt quickly to abrupt changes of the process.

Adaptive sample-based ensembles for regression inspired by SW have been used as predictive models for industrial processes. One of the first methods to introduce this concept is the ILLSA [Kadlec and Gabrys, 2011]. Kaneko and Funatsu [2014] propose an ensemble of on-line SVM models. On the on-line phase, a new model is added at a fixed frequency using the current data window. In [Lv *et al.*, 2013] the training data is partitioned into different subsets using the Fuzzy C-means clustering algorithm. Then, each subset is employed to train a Least Squares-SVM model. No adaptive mechanisms are employed to the ensemble. However, all these listed ensembles do not include and exclude models during on-line operation; but the on-line inclusion and exclusion of models can be an important factor for improving ensemble prediction performance.

The frequency of including a new model in the ensemble is a key factor in on-line ensembles. Ensemble applications usually add a new model only at a predefined fixed frequency (e.g. batch frequency) [Brzezinski and Stefanowski, 2014]. However,

results indicate that when new models are added at a sample frequency, the ensemble can adapt quickly to the changes and the system's performance is improved significantly [Kolter and Maloof, 2005]. Another important issue is the dynamic removal of models from the ensemble, since the used memory and computational resources may be increasing considerably; and some models may contain little information about the current state of the process.

In Chapter 5, an on-line weighted ensemble (OWE) of regression models, which can include and remove models over time, was proposed. OWE achieves good performance in several types of changes by tuning a discounting factor. It does not retrain models over time. This characteristic is important to maintain the system performance in recurring changes, since it avoids that models lose knowledge about the past data. However, in some applications, it is not possible to maintain knowledge about the past scenarios, due to limited memory capacity of the system; and applications may have non-recurring and abrupt behaviors so that all the models of the ensemble should be tuned to reflect the current state of the process. In this outlined scenario, retraining models is one of the key factors for covering this challenge. Since old models trained on past data can be adapted to the new concepts using new samples, and thus the system can acquire more adaptation capability.

This Chapter proposes a new dynamic and on-line ensemble regression (DOER) approach of OS-ELM models with fast adaptation capability for on-line prediction of variables measured at low sampling frequency and on a sample basis in applications with time-varying behavior. In this thesis, the term "fast adaptation capability" is related to a property of the ensemble system of adapting all models quickly to the current state of the process. This work incorporates three contributions from Chapter 5: dynamic removal and inclusion of models; regression scope; and thorough experimental analysis using artificial and industrial data sets. Moreover, DOER brings together desired properties which are not given by OWE method proposed in Chapter 5: (1) on-line inclusion and removal of models to keep only the most accurate models with respect to the *current state of the system*; (2) dynamic adaptation of the models' combination weights based on their on-line predictions on the *recent samples*; and (3) on-line adaptation of the models' parameters (i.e. on-line model retraining). Therefore, the ensemble system is tunned to reflect the current state of the system, so that it can adapt faster to the changes.

Experiments on four artificial data sets and six real-world industrial data sets

are reported to evaluate the effectiveness of the DOER. Results show that DOER has high adaptation capability, and DOER is not only comparable to the state-of-the-art approaches, but in most cases, DOER has better accuracy when compared to them.

This Chapter is organized as follows. Section 6.2 describes the DOER algorithm. The experiments are reported and analyzed in Section 6.3. Finally, Section 6.4 contains concluding remarks.

6.2 Dynamic and On-line Ensemble Regression

DOER offers together the following strategies or characteristics which are not jointly given by previous works from other authors in the literature: (1) *on-line ensemble learning* - most ensembles are developed *off-line* and do not take into account that the process or data may exhibit time-varying behaviors; (2) *regression scope*, as there is a lack of on-line regression ensembles; (3) *sample-based ensemble* which offers higher accuracy and faster adaptivity when compared to batch-based ensembles; (4) *adaptation of the models' combination weights*, DOER incorporates dynamic adaptation of the models' combination weights based on the models' accuracies on the most recent samples - DOER assigns high combination weights to the most accurate models, allowing that inaccurate models do not degrade the ensemble's performance; (5) *adaptation of the models' parameters*, leading the system to a faster adaptation in changing environments; (6) *dynamic inclusion and removal of models*, new models are launched to the ensemble and models that do not contribute to the ensemble are excluded; the (7) *pruning strategy* removes the model with the worst accuracy on the most recent samples, therefore old and accurate models can be kept; DOER can work with (8) *unnormalized data*; and (9) *overfitting control*, the model evaluation does not consider the performance on the training phase.

6.2.1 DOER Description

DOER builds a dynamic sample-based ensemble of weighted models based on the SW approach. A data window of fixed size is maintained, and when a new sample is available, it is included into the window, and the oldest sample is removed from the window. The main idea is to add a new model trained with the data window when

Algorithm 6.1 Dynamic and on-line ensemble regression (DOER).

Input: a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$; window's size m; an on-line supervised learner; factor for controlling the inclusion of a new model α; maximum number of models N;

1. **Initialization:** set $\mathcal{E} \leftarrow \emptyset$ (ensemble), $t = m$, $k = 1$, and $\mathbf{D}^t = \{(\mathbf{x}_t, y_t)\}_{t=1}^{m} \subset \mathbf{D}$;
2. $f_k \leftarrow$ Obtain a new model trained with $\mathbf{D}^t$; Set $life_k = 0$, $\text{MSE}_k^t = 0$; $w_k = 1$, and $\mathcal{E} \leftarrow \mathcal{E} \cup f_k$;
3. **while** $t \leqslant T$ **do**:
   (a) Slide the window: $t \leftarrow t + 1$; $\mathbf{D}^t = \mathbf{D}^{t-1} + (\mathbf{x}_t, y_t) - (\mathbf{x}_{t-m}, y_{t-m})$;
   (b) Predict y_t as: $F(\mathbf{x}_t) = \left(\sum_{n=1}^{k} w_n f_n(\mathbf{x}_t)\right) / \sum_{n=1}^{k} w_n$;
   (c) For all models $f_n \in \mathcal{E}$: obtain the prediction error e_n^t on $\mathbf{x}_t$ as $e_n^t = (y_t - f_n(\mathbf{x}_t))^2$; and Set $life_n \leftarrow life_n + 1$;
   (d) Obtain MSE_n^t for each model $f_n \in \mathcal{E}$ using Equation (6.2);
   (e) Weight all models from $\mathcal{E}$ using Equations (6.3), and (6.4);
   (f) Incrementally retrain all models from $\mathcal{E}$ using $(\mathbf{x}_t, y_t)$;
   (g) **if** $|(F(\mathbf{x}_t) - y_t)/y_t| > \alpha$
      i. $f_0 \leftarrow$ Obtain a new model trained with $\mathbf{D}^t$; Set $life_0 = 0$, $\text{MSE}_0^t = 0$, and $w_0 = 1$;
      ii. **if** $k < N$
         A. **then** Include f_0 into $\mathcal{E}$: Set $k \leftarrow k + 1$, $f_k \leftarrow f_0$, and $\mathcal{E} \leftarrow \mathcal{E} \cup f_k$;
         B. **else** Replace model f_z by f_0, where $z = \text{argmax}_{v=1,\ldots,k}\left(\text{MSE}_v^t\right)$; Set $f_z \leftarrow f_0$;
4. **end while**

the ensemble's performance is not satisfactory on the newest sample of the window. The proposed on-line ensemble regression method is presented in Algorithm 6.1.

The algorithm starts by defining some *inputs*: a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t) | \mathbf{x}_t \in \mathbb{R}^{r \times 1}, y_t \in \mathbb{R}, t = 1, \ldots, T\}$, where $(\mathbf{x}_t, y_t)$ is the sample given at time t, $\mathbf{x}_t$ is a vector of r input variables, and y_t is the output variable; the window's size, m; a generic on-line supervised learner; a factor for controlling the inclusion of a new model, α; and the maximum number of models, N. In Step 1, some variables are set: $\mathcal{E}$ denotes the set of models; k is the number of models; and $\mathbf{D}^t$ corresponds to the current data window of size m, at time t, where $\mathbf{D}^t$ initially receives the first m samples from $\mathbf{D}$. Step 2 creates the first model for the ensemble. It is trained with the initial data window.

On the on-line phase of the algorithm (from Step 3 to Step 4), for each new incoming sample, the window slides along the data (Step 3(a)). This operation excludes the oldest sample, $(\mathbf{x}_{t-m}, y_{t-m})$ from the window and includes the newest sample, $(\mathbf{x}_t, y_t)$, into the window. The ensemble prediction $F(\mathbf{x}_t)$ of a new input sample $\mathbf{x}_t$ is obtained using a weighted sum of the models' outputs (Step 3(b)).

The error of each model f_n from the ensemble $\mathcal{E}$ $(n = 1, \ldots, k)$ on the newest sample $(\mathbf{x}_t, y_t)$ is calculated as (Step 3(c)):

$$e_n^t = (y_t - f_n(\mathbf{x}_t))^2, \tag{6.1}$$

where $f_n(\mathbf{x}_t)$ is the predicted output of model f_n using the input sample $\mathbf{x}_t$. In Step 3(c), the variable $life_n$ is also incremented. It denotes the total number of on-line evaluations performed with a model f_n. After this, the current error of each model on the current data window, MSE_n^t, is obtained as (Step 3(d)):

$$\mathrm{MSE}_n^t = \begin{cases} 0, & \text{if } life_n = 0, \\ \frac{life_n - 1}{life_n} \cdot \mathrm{MSE}_n^{t-1} + \frac{1}{life_n} \cdot e_n^t, & \text{if } 1 \leqslant life_n \leqslant m, \\ \mathrm{MSE}_n^{t-1} + \frac{e_n^t}{m} - \frac{e_n^{t-m}}{m}, & \text{if } life_n > m. \end{cases} \tag{6.2}$$

This approach is similar to one proposed by OAUE [Brzezinski and Stefanowski, 2014]. The aim is to estimate the average of the predictive error of f_n on the most recent m samples using the MSE. Equation (6.2) works like an adaptive MSE. A new model initially receives MSE_n^t equal to 0. As it performs on-line predictions and the variable life $life_n$ is incremented, the window of errors is also enlarged up to a maximum width m. If $life_n > m$ at a time t, the new error e_n^t is considered to compute MSE_n^t and the old error e_n^{t-m} is eliminated in the calculation of MSE_n^t. Note that only errors observed on the on-line phase are considered to calculate MSE_n^t.

Step 3(e) dynamically assigns the current combination weight w_n of each model f_n according to its error on the window, MSE_n^t, as:

$$w_n = \exp\left(-\frac{\mathrm{MSE}_n^t - \mathrm{med}(\mathbf{MSE}^t)}{\mathrm{med}(\mathbf{MSE}^t)}\right), \tag{6.3}$$

where

$$\mathbf{MSE}^t = [\mathrm{MSE}_1^t, \ldots, \mathrm{MSE}_k^t], \tag{6.4}$$

and med($\mathbf{MSE}^t$) is the median value of the models' errors, $\mathbf{MSE}^t$. Equation (6.3) transforms the combination weights in such a way that a model f_n with MSE_n^t around the median value receives a combination weight equal to 1, while models with MSE_n^t lower than the median have their combination weights exponentially increased, and models with MSE_n^t larger than the median have their combination weights exponentially decreased. This strategy allows that models with low accuracies do not impact negatively the ensemble's performance. On the other hand, more "credit" is given to the models that have high accuracy. In Step 3(f), all the models are retrained, keeping the models updated on the current scenario.

Step 3(g) evaluates if a new model should be included to the ensemble. The criterion includes a new model when the absolute relative error of the ensemble on the newest sample is greater than α. The new model f_0 is trained using the samples from the current data window, $\mathbf{D}^t$, where its combination weight is initially set as 1. Therefore, it receives the same combination weight as a model with error around the median error, med($\mathbf{MSE}^t$). This criterion smooths the contribution of a new model at the time $t+1$, the period in which this model will be evaluated on-line for the first time.

In Step 3(g)ii, if the number of models of the ensemble (k) is lower than N, then the value of k is incremented by 1 and the new model f_0 is attributed to f_k. Otherwise, if k is greater than or equal to N, then f_0 replaces the least accurate model f_n of the ensemble. The criterion substitutes the model f_n with the highest error MSE_n^t. Therefore, a new model created in iteration t is never excluded by the pruning strategy at the same time t.

6.3 Experimental Results

In this Section, experiments are performed with the DOER and the results are detailed and compared to state-of-the-art approaches. Four artificial data sets and six real-world data sets are employed to evaluate the algorithms' effectiveness on different changing scenarios. The tests have been performed on the Matlab environment, running on a PC equipped with an Intel Core i7-2600 3.4 GHz processor of 4 cores and 8 GB of RAM.

Table 6.1: Specifications of the industrial data sets used in the experiments.

Data set	Before preprocessing		After preprocessing		Data Set
	# Samples	# Inputs	# Samples	# Inputs	Size
polymerization reactor	8687	15	**648**	**10**	*small*
cement kiln process	43469	195	**701**	**45**	*small*
debutanizer column	2394	7	**1836**	**7**	*medium*
powder detergent production	1962	14	**1962**	**7**	*medium*
thermal oxidizer	2053	39	**2053**	**6**	*medium*
SRU, output 1	10081	5	**6909**	**5**	*large*
SRU, output 2	10081	5	**6806**	**5**	*large*

6.3.1 Data Set Description

The artificial data sets are the hyperplane data set [Kolter and Maloof, 2005]; the local and abrupt drift data set (Friedman-LA); the global recurring abrupt drift data set (Friedman-GRA); and the global non-recurring gradual drift data set (Friedman-GnRG) [Ikonomovska, 2012]. For more details, see Subsection 5.3.1.

Six real-world data sets are considered [Fortuna *et al.*, 2006; Kadlec and Gabrys, 2011; Grbić *et al.*, 2013], as listed in Table 6.1. They aim to predict important variables in industrial applications. Most industrial processes exhibit some kind of time-varying behavior, and so these data sets are crucial to evaluate the proposed methodologies. Details and descriptions of the data sets can be found in Section 2.7. Preprocessing was applied to select input variables in every single data set, as the strategy described in Subsection 4.4.1. Unlike the other Chapters, this Chapter uses *Hampel identifier* [Lin *et al.*, 2007] (as described in Subsection 2.3.2) to detect outliers. For data sets with a large number of outliers, outliers were replaced by the median value. In this case, for a given variable, an outlier was replaced by the median value of the corresponding variable. For data sets with few number of outliers, outliers were removed.

6.3.2 Approach Setup and Description

Experiments are performed by comparing DOER to four on-line strategies using a single model OS-ELM algorithm; and six on-line ensemble algorithms (EOS-ELM, AddExp, On-line Bagging (OB), Learn^{++}.NSE, OWE, and OAUE). The single models are designed using the main structures of on-line learning algorithms presented in

Subsection 3.5.3: sample-based OS-ELM (**OS-ELM***s*), as in Algorithm 3.5; batch-based OS-ELM (**OS-ELM***b*), as in Algorithm 3.4; sample-based OS-ELM using a SW (**OS-ELM***s***-SW**), as in Algorithm 3.6; and batch-based OS-ELM using a SW (**OS-ELM***b***-SW**), which is a modified version of Algorithm 3.6, where samples are given in batches.

The following structure is employed to evaluate all the approaches. Consider a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$ with T samples. The initial single model, or the first model of the ensemble, or the pool of models (depending on the approach) is created using the first m samples from $\mathbf{D}$, i.e. $\mathbf{D}^0 = \{(\mathbf{x}_t, y_t)\}_{t=1}^{m}$. The other $(T-m)$ samples, $\mathbf{D}^{online} = \{(\mathbf{x}_t, y_t)\}_{t=m+1}^{T}$, are grouped to form the on-line data to simulate an on-line scenario. For each approach, its performance (accuracy) is evaluated using the mean and the standard deviation of the MSE between the real and the predicted outputs of the on-line data set $\mathbf{D}^{online}$ in 20 trials.

The OS-ELM (described in Subsection 3.5.4) is the base model for all the ensembles, except Learn^{++}.NSE and OWE. As the Learn^{++}.NSE and OWE ensembles do not employ model retraining, their base model is the ELM algorithm. ELM and OS-ELM were implemented using Algorithm 3.2 and Algorithm 3.9, respectively. The number of training samples to train an ELM model is m; while for an OS-ELM, the number of training samples in the initialization/training phase T_0 is equal to m. For both OS-ELM and ELM, the hidden layer activation function $g(x)$ is *sigmoid*. The number of neurons in the hidden layer L is selected by varying it in the interval of $[1, 20]$. This interval may be adjusted to $[1, m]$ if $m < 20$, since L should not be greater than m in order to comply with the assumptions in the ELM algorithms. The value of L is selected based on the best performance on a *10-fold cross-validation* using the training data set in 1 trial, where the best number of neurons is selected as the one that maximizes the mean testing performance on the 10-folds.

For the OS-ELM model, on-line data scaling of the input and output variables is applied using Equations (2.3)-(2.5) to attain zero-mean and unit-variance, where on the on-line phase, the mean and standard deviation of each variable are recursively adapted using (2.4)-(2.5) as new samples are available [Galicia *et al.*, 2012]. The AddExp requires that the output samples are normalized to the $[0, 1]$ interval. Therefore, the outputs of all data sets were firstly normalized to this interval for facilitating the comparison of AddExp with the other methods.

For the on-line ensembles, the maximum number of models is 15, i.e. $N = 15$.

This value was chosen to reduce the processing time and memory, since there is not a considerable improvement of the ensembles' performances when N further increases. The on-line ensemble learning algorithms were implemented as follows:

- **EOS-ELM** [Lan *et al.*, 2009]. The training data set is $\mathbf{D}^0$. All the models are trained with the same activation function and L, where L is selected as the most frequent best number of neurons on 20 trials of *10-fold cross-validation* on $\mathbf{D}^0$, and at each trial the best number of neurons is selected as the one that maximizes the mean testing performance on the 10-folds. On the on-line phase, all the models are retrained and combined by average.

- **OB**. It was implemented according to the structure in [Oza and Russell, 2001], where L can be different for each model and the training data set for creating the first model is $\mathbf{D}^0$. On the on-line phase, all models are retrained and combined by average.

- **AddExp** [Kolter and Maloof, 2005]. It was implemented according to the "pruned AddExp" description in Subsection 5.3.2, where the first model is created using $\mathbf{D}^0$.

- **Learn^{++}.NSE** [Elwell and Polikar, 2011, 2009]. It was implemented according to the "pruned Learn^{++}.NSE" description in Subsection 5.3.2. The first model is created using $\mathbf{D}^0$.

- **OAUE**. It is an ensemble for classification tasks; so that the models' combination weights are obtained as $1/(\text{MSE}_n^t)$ to convert OAUE for regression tasks, where MSE_n^t is obtained using Equation (6.2). OAUE was implemented according to [Brzezinski and Stefanowski, 2014], where each batch/block is considered to have size m. If the number of models is greater than N, then the weakest model is replaced.

- **OWE** [Soares and Araújo, 2015c]. It was implemented according to Algorithm 5.1 and the "pruned OWE" description in Subsection 5.3.2, with the ELM as the base model. The factor for demarcating incorrect and correct predictions is set to $\theta = 0.05$, and the pruning activation factor ρ is set to 1. The discount factor κ is set to 0.2 for all data sets, except for Friedman-GRA where κ is set to 0.99, because it has a recurring nature. The factor for including a new

model is set to $\alpha = 0.10$ for artificial data sets; and $\alpha = 0.04$ for industrial data sets.

- **DOER**. It was implemented according to the structure proposed in Section 6.2. The values of m and α can vary on each experiment and analysis.

6.3.3 Analysis of DOER Parameters

The parameter setting is discussed in this Subsection, since in SW approaches the window's size is a factor that may influence the system's accuracy; and in on-line ensemble systems that add dynamically new models, the frequency of including models may also impact the ensemble's performance. Tests of the DOER are conducted by setting α in the range of $\alpha \in \{0.04, 0.06, 0.08, 0.10\}$, and m in the following ranges:

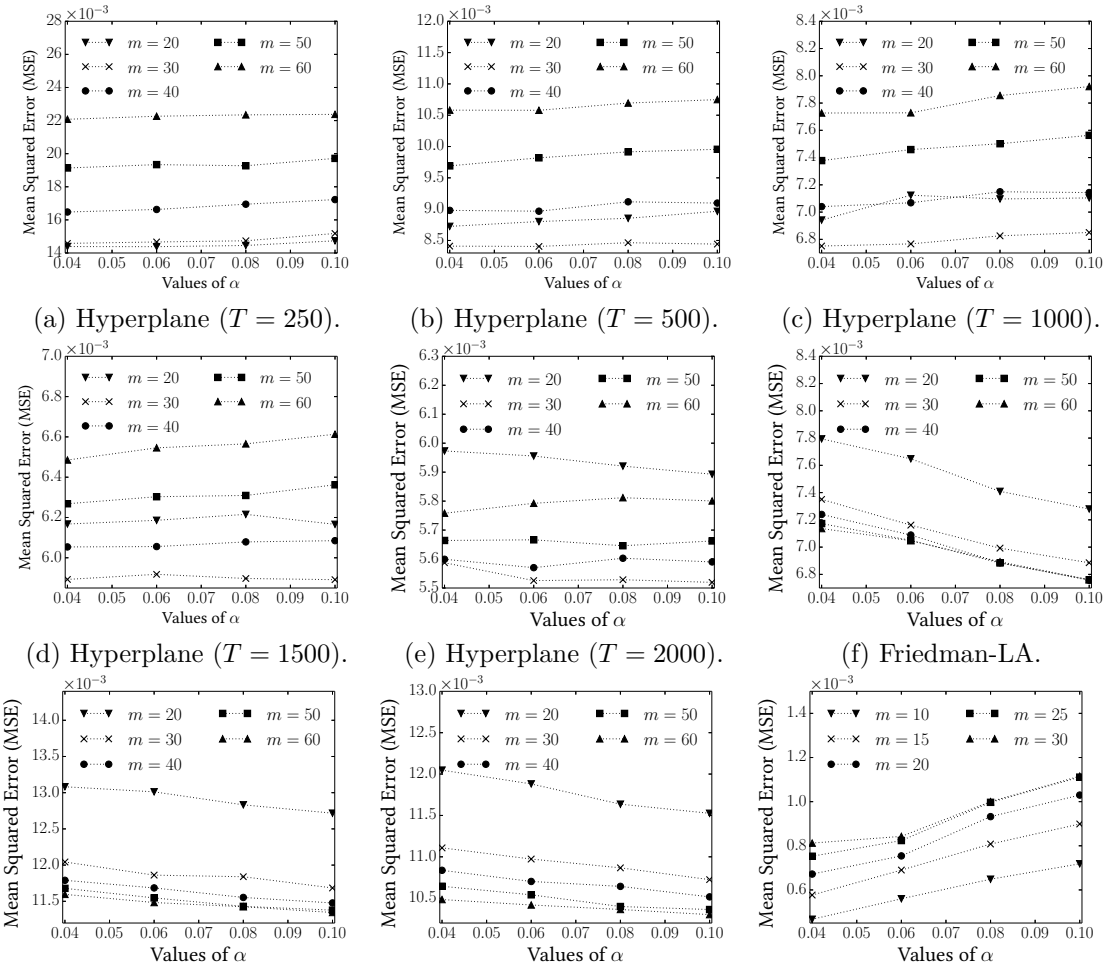
- $m \in \{10, 15, 20, 25, 30\}$, for the real data sets of small size;
- $m \in \{20, 30, 40, 50, 60\}$, for the real data sets of medium size and the artificial data sets; and
- $m \in \{30, 60, 90, 120, 150\}$, for the real data sets of large size.

Five data sets using the hyperplane data were generated by varying T in the following range in order to study different rates of concept drift:

- $T \in \{250, 500, 1000, 1500, 2000\}$.

The smaller the value of T, the larger is the rate of concept drift, since concept drift episodes with the same overall concept-state transitions occur in intervals of smaller duration (lower number of samples). For all data sets, the average errors on the 20 trials are shown in Figure 6.1 and Figure 6.2.

The results indicate different behaviors when m takes different values. Figures 6.1(a)-6.1(e) indicate that m is related to the rate of the concept drift. That is, in data sets where the rate of concept drift is large, DOER has better accuracy when m is small; while in data sets where the rate of concept drift is small, DOER has better accuracy when m is large. This can be observed in Figure 6.1(a) (hyperplane $T = 250$), where DOER has the best performance for $m = 20$, and has the worst performance for $m = 60$; and in Figure 6.1(e) (hyperplane $T = 2000$), where DOER has the best performance for $m = 30$, and has the worst performance for $m = 20$.



(a) Hyperplane ($T = 250$). (b) Hyperplane ($T = 500$). (c) Hyperplane ($T = 1000$).

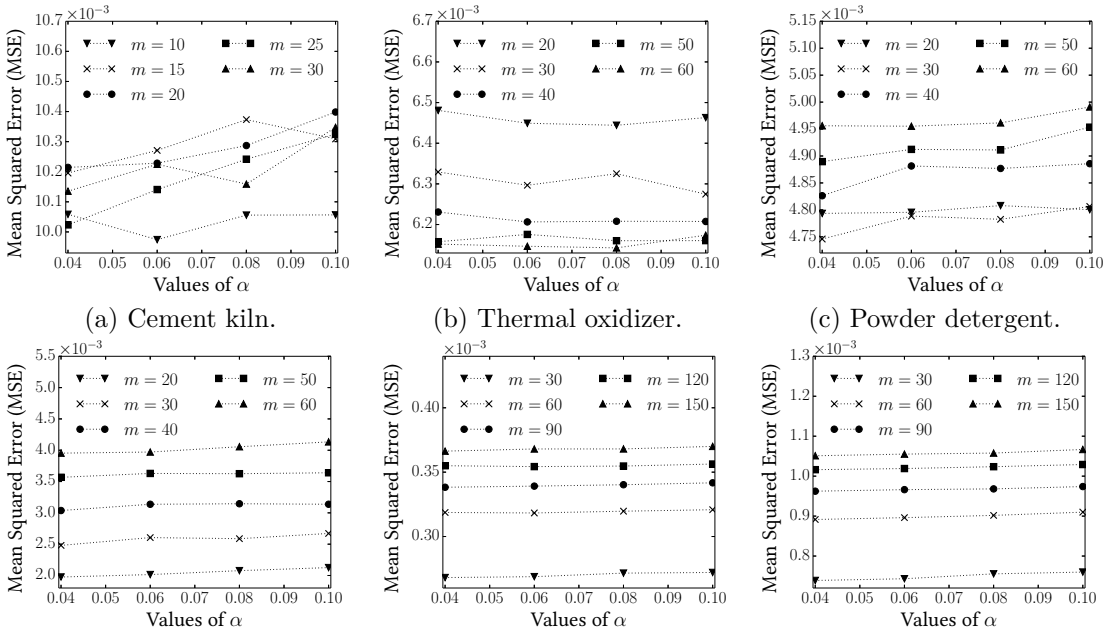
(d) Hyperplane ($T = 1500$). (e) Hyperplane ($T = 2000$). (f) Friedman-LA.

(g) Friedman-GRA. (h) Friedman-GnRG. (i) Polymerization reactor

Figure 6.1: Average of errors of the DOER algorithm on all the data sets using different values of m and α. (Part 1).

For the Friedman data sets, the tests indicated that the DOER's performance in general increases when m increases. Otherwise, for the real-world data sets, DOER usually has high accuracy when m is small. This indicates that the real-world data sets contain concepts of large concept drift rate.

The results also indicate that α is related to the rate of concept drift. For example, for the hyperplane with $T = 1000$ (Figure 6.1(c)), DOER has a considerable improvement of accuracy when α has small values; while for the hyperplane with $T = 2000$ (Figure (6.1(e)), DOER has better accuracy when α is large. The same trend as in the hyperplane data set is observed in the Friedman data sets: DOER



(a) Cement kiln. (b) Thermal oxidizer. (c) Powder detergent.

(d) Debutanizer column. (e) SRU (output 1). (f) SRU (output 2).

Figure 6.2: Average of errors of the DOER algorithm on all the data sets using different values of m and α. (Part 2).

improves the performance when α is larger. Otherwise, for the polymerization reactor and cement kiln data sets, DOER improves significantly the accuracy when α is low. For the other real-world data sets, α does not affect substantially the DOER's performance. Therefore, the experiments reveal that the most adequate α depends on the change characteristics of each data set. For example, for data sets with high rate of changes and that require faster adaptation capability (as some industrial data sets), α should be set to a small value for including new models with high frequency so that the ensemble is adapted quickly to the new changes. On the contrary, for data sets with low rate of changes and that require low adaptation capability, α should be set to a large value for adding models at a low frequency. In the experiments below, the hyperplane data set has 2000 samples ($T = 2000$); and α is set to 0.10 for the artificial data sets, and α is set to 0.04 for the real-world data sets.

6.3.4 Comparing DOER to Other Approaches

Tables 6.2, 6.3, 6.4, and 6.5 report the experimental results based on the MSE of all the approaches using different values of m. For the artificial data sets, in most cases, the performances of all the methods improve when m increases. For the SW approaches, such as the OS-ELMs-SW and the OS-ELMb-SW, the ideal window's size m should be such that the window contains relevant samples that maximize the representativeness of the current concept: a small value of m can provide faster adaptivity, but in more stable phases the model's performance can decrease; while a large value of m can provide stability, but the system cannot recover faster to the changes. For example, since the artificial data sets contain concepts of large time durations, the SW approaches have high performance for large windows. In other approaches, large windows offer low adaptation capability to the system. This strategy is important to data sets where concepts are replaced more slowly by new concepts, as in the artificial data sets. This can be observed in the Learn^{++}.NSE, where the ensemble's accuracy improves when m increases, since the ensemble is only adapted when a batch is available. Other important remark for the artificial data sets is that in most cases the on-line single models have low accuracies when compared to the on-line ensembles, demonstrating that an ensemble is usually more accurate than any single model.

DOER outperforms the other approaches on the data sets with non-recurring concepts, i.e. the hyperplane and Friedman-GnRG data sets. On the other hand, the DOER has inferior accuracy when compared to OAUE and AddExp in the Friedman-LA data set; and the DOER has accuracy comparable to the accuracy of the AddExp method in the Friedman-GRA data set, a data set with recurring concepts. This reveals that DOER can also deal with recurring changes. Therefore, the experiments using the artificial data sets seem to support that, in terms of accuracy, DOER is not only comparable to the other approaches of the state-of-the-art, but in most cases DOER has higher performance.

In this Chapter, it can be observed that OWE has inferior performance when compared to AddExp in the Friedman data sets; while in Chapter 5, OWE achieved higher accuracy when compared to AddExp. This is mainly attributed to the change of learning algorithm. Chapter 5 employs PLS and RPLS as learning algorithms; while this Chapter employs ELM and OS-ELM as learning algorithms.

In the real-world data sets, different characteristics are noticed when m increases or decreases. In the OS-ELMs, m is the number of samples to be used on the training phase. Therefore, when m increases the model performs better, and consequently its on-line performance is improved, as can be seen in the powder detergent data set, where for $m = 20$ the MSE is 7.09×10^{-3}, and for $m = 60$ the MSE is 6.48×10^{-3}. On the other hand, in the OS-ELMb, m has impact not only in the number of samples to be employed on the training, but m also influences the time interval of the model adaptation. So that, when m is small, the time interval is small, and consequently the model will be adapted at a high frequency. In this case, for the OS-ELMb, low values of m offer higher adaptation and performance. This can be seen in the debutanizer column data sets, where for $m = 60$ the MSE is 13.85×10^{-3}, and when $m = 10$ the MSE 11.90×10^{-3}. In the OS-ELMs-SW, m holds the number of samples employed to train a new model. Therefore, data sets with fewer concepts operate better when m is large (large window), while data sets with more concepts perform well with small windows, ensuring faster adaptation. The OS-ELMb-SW and Learn^{++}.NSE are batch-based algorithms, and m has impact on the time interval of the system adaptation, as in the OS-ELMb. Therefore, low values of m guarantee faster adaptation to changes. For example, for the Learn^{++}.NSE using the thermal oxidizer data set, when $m = 20$ the MSE is 12.14×10^{-3}, and when $m = 60$ the MSE is 24.15×10^{-3}. In EOS-ELM and OB, m works as in OS-ELMs, i.e. m refers to the size of the initial training data set. Therefore, these algorithms improve their performances when m increases. As shown in the cement kiln data set for the EOS-ELM, when for $m = 10$ the MSE is 22.88×10^{-3}, and for $m = 30$ the MSE is 19.13×10^{-3}. In the AddExp, m only refers to the number of samples to be used to train a new model. The results show that variations of m have small impact on the AddExp's performance, as indicated in the standard deviation of the last column of the tables. In the OAUE, m is applied to measure the models' performances, to define the number of samples to train a new model ($2 \times m$), and to define the time interval for including a new model in the ensemble. Possibly, m affects mainly the time interval for including a new model, since, in most cases, approaches that add new models at a high frequency (DOER, OWE, and AddExp) have a higher accuracy when compared to other approaches. This is because, in such high frequency approaches such as DOER, OWE, and AddExp, ensembles contain mainly models trained on the samples of the most recent concept.

Table 6.2: MSE results of the on-line learning algorithms using the hyperplane data set, the Friedman-LA data set and the Friedman-GRA data set.

Method/Data set		Window's size					All the
		($m = 20$)	($m = 30$)	($m = 40$)	($m = 50$)	($m = 60$)	window's sizes
hyperplane data set							
OS-ELMs	(*on-line single model*)	23.00 (2.28)	21.39 (0.86)	21.36 (1.20)	21.35 (0.67)	21.37 (0.57)	21.69 (0.73)
OS-ELMb	(*on-line single model*)	22.42 (1.75)	21.53 (0.47)	21.66 (0.57)	21.83 (0.61)	21.92 (0.23)	21.87 (0.34)
OS-ELMs-SW	(*on-line single model*)	23.81 (0.93)	14.97 (0.38)	11.64 (0.29)	10.23 (0.31)	9.59 (0.32)	14.05 (5.84)
OS-ELMb-SW	(*on-line single model*)	24.46 (1.62)	16.00 (1.11)	12.32 (0.47)	11.93 (0.50)	11.57 (0.66)	15.25 (5.44)
EOS-ELM	(*on-line ensemble*)	19.09 (0.20)	19.24 (0.24)	19.51 (0.19)	19.69 (0.17)	19.78 (0.13)	19.46 (0.29)
OB	(*on-line ensemble*)	19.10 (0.37)	19.24 (0.24)	19.61 (0.21)	19.66 (0.17)	19.80 (0.23)	19.48 (0.30)
Learn^{++}.NSE	(*on-line ensemble*)	16.54 (0.88)	11.38 (0.47)	10.07 (0.40)	10.21 (0.44)	10.13 (0.46)	11.67 (2.78)
OAUE	(*on-line ensemble*)	19.70 (0.17)	19.55 (0.12)	19.59 (0.18)	19.63 (0.12)	19.67 (0.16)	19.63 (0.06)
AddExp	(*on-line ensemble*)	7.26 (0.24)	6.73 (0.15)	6.85 (0.15)	6.93 (0.13)	7.22 (0.11)	7.00 (0.23)
OWE	(*on-line ensemble*)	7.91 (0.19)	6.47 (0.11)	6.33 (0.09)	6.37 (0.09)	6.48 (0.08)	6.71 (0.67)
DOER	(*on-line ensemble*)	**5.89 (0.12)**	**5.52 (0.07)**	**5.59 (0.06)**	**5.66 (0.06)**	**5.80 (0.07)**	**5.69 (0.15)**
Friedman-LA data set							
OS-ELMs	(*on-line single model*)	12.81 (4.53)	9.55 (3.14)	8.13 (1.99)	7.41 (0.64)	7.63 (0.66)	9.11 (2.23)
OS-ELMb	(*on-line single model*)	11.91 (3.65)	8.48 (1.54)	8.63 (1.24)	7.98 (0.79)	7.60 (0.73)	8.92 (1.72)
OS-ELMs-SW	(*on-line single model*)	20.87 (0.62)	15.59 (0.56)	13.08 (0.26)	11.81 (0.31)	10.71 (0.26)	14.41 (4.04)
OS-ELMb-SW	(*on-line single model*)	20.90 (1.47)	15.60 (0.56)	13.29 (0.51)	11.69 (0.45)	10.80 (0.35)	14.45 (4.04)
EOS-ELM	(*on-line ensemble*)	9.61 (1.32)	6.89 (0.35)	6.58 (0.37)	6.53 (0.14)	**6.39 (0.10)**	7.20 (1.36)
OB	(*on-line ensemble*)	8.45 (0.67)	6.85 (0.23)	6.58 (0.17)	6.56 (0.09)	6.46 (0.09)	6.98 (0.83)
Learn^{++}.NSE	(*on-line ensemble*)	12.37 (0.60)	10.01 (0.39)	9.02 (0.27)	8.37 (0.22)	8.13 (0.26)	9.58 (1.72)
OAUE	(*on-line ensemble*)	**6.55 (0.17)**	**6.51 (0.15)**	**6.51 (0.15)**	**6.49 (0.11)**	6.59 (0.18)	**6.53 (0.04)**
AddExp	(*on-line ensemble*)	7.16 (0.22)	6.84 (0.13)	6.73 (0.11)	6.70 (0.14)	6.64 (0.10)	6.81 (0.21)
OWE	(*on-line ensemble*)	9.31 (0.20)	7.89 (0.10)	7.44 (0.07)	7.24 (0.06)	7.12 (0.07)	7.80 (0.89)
DOER	(*on-line ensemble*)	7.28 (0.13)	6.89 (0.06)	6.76 (0.07)	6.76 (0.05)	6.76 (0.05)	6.89 (0.23)
Friedman-GRA data set							
OS-ELMs	(*on-line single model*)	18.87 (4.20)	18.00 (4.84)	16.60 (3.34)	15.21 (3.20)	15.07 (2.12)	16.75 (1.68)
OS-ELMb	(*on-line single model*)	22.57 (5.92)	19.71 (7.07)	15.93 (3.49)	14.87 (2.08)	15.12 (1.66)	17.64 (3.38)
OS-ELMs-SW	(*on-line single model*)	36.11 (0.74)	26.59 (0.93)	21.61 (0.63)	19.16 (0.50)	17.54 (0.46)	24.20 (7.49)
OS-ELMb-SW	(*on-line single model*)	36.56 (1.80)	26.83 (1.69)	21.52 (0.90)	19.89 (0.84)	17.76 (0.67)	24.51 (7.52)
EOS-ELM	(*on-line ensemble*)	15.75 (2.58)	13.15 (1.25)	12.42 (0.67)	12.09 (0.24)	11.98 (0.22)	13.08 (1.56)
OB	(*on-line ensemble*)	14.79 (0.91)	13.19 (0.78)	12.27 (0.21)	12.08 (0.19)	11.99 (0.15)	12.87 (1.18)
Learn^{++}.NSE	(*on-line ensemble*)	28.43 (2.17)	21.06 (1.01)	17.76 (0.69)	16.62 (0.73)	15.49 (0.55)	19.87 (5.22)
OAUE	(*on-line ensemble*)	12.46 (0.36)	12.21 (0.25)	12.25 (0.22)	12.26 (0.23)	12.44 (0.36)	12.32 (0.12)
AddExp	(*on-line ensemble*)	**12.41 (0.26)**	11.78 (0.17)	11.58 (0.17)	11.48 (0.13)	11.50 (0.13)	11.75 (0.39)
OWE	(*on-line ensemble*)	14.54 (0.33)	12.71 (0.13)	12.09 (0.18)	11.77 (0.12)	11.56 (0.10)	12.54 (1.20)
DOER	(*on-line ensemble*)	12.72 (0.20)	**11.68 (0.12)**	**11.48 (0.11)**	**11.38 (0.11)**	**11.35 (0.11)**	**11.72 (0.57)**

The MSE values have been multiplied by 10^3. Average and SD of MSE are obtained on 20 trials of the algorithms. Bold values indicate the lowest error of a data set on a window's size. The last column reports the average and SD of the error of each approach on all window's sizes.

Table 6.3: MSE results of the on-line learning algorithms using the Friedman-GnRG data set, the polymerization reactor data set and the cement kiln data set.

Method/Data set		Window's size					All the
		$m = 20$	$m = 30$	$m = 40$	$m = 50$	$m = 60$	window's sizes
Friedman-GnRG data set							
OS-ELMs	(*on-line single model*)	19.19 (4.08)	15.68 (2.10)	14.74 (1.09)	14.27 (0.99)	14.20 (1.43)	15.61 (2.08)
OS-ELMb	(*on-line single model*)	19.72 (4.76)	16.22 (4.00)	15.08 (2.04)	14.98 (2.04)	14.43 (1.51)	16.09 (2.13)
OS-ELMs-SW	(*on-line single model*)	30.34 (0.87)	23.12 (0.63)	19.36 (0.51)	17.11 (0.45)	15.77 (0.44)	21.14 (5.85)
OS-ELMb-SW	(*on-line single model*)	30.70 (1.06)	23.53 (1.42)	19.55 (0.85)	16.53 (0.69)	15.83 (0.52)	21.23 (6.10)
EOS-ELM	(*on-line ensemble*)	13.68 (1.23)	12.37 (0.23)	12.36 (0.22)	12.32 (0.19)	12.26 (0.12)	12.60 (0.61)
OB	(*on-line ensemble*)	13.77 (0.67)	12.43 (0.23)	12.28 (0.17)	12.28 (0.13)	12.29 (0.11)	12.61 (0.65)
Learn^{++}.NSE	(*on-line ensemble*)	22.49 (1.16)	17.51 (0.47)	15.51 (0.48)	13.92 (0.52)	13.45 (0.47)	16.58 (3.67)
OAUE	(*on-line ensemble*)	12.52 (0.21)	12.58 (0.23)	12.51 (0.26)	12.54 (0.26)	12.53 (0.15)	12.54 (0.03)
AddExp	(*on-line ensemble*)	11.57 (0.18)	11.06 (0.12)	10.90 (0.14)	10.74 (0.13)	10.72 (0.12)	11.00 (0.35)
OWE	(*on-line ensemble*)	14.43 (0.30)	12.03 (0.15)	11.22 (0.13)	10.73 (0.10)	10.54 (0.13)	11.79 (1.58)
DOER	(*on-line ensemble*)	**11.53 (0.15)**	**10.72 (0.07)**	**10.52 (0.09)**	**10.36 (0.07)**	**10.30 (0.07)**	**10.69 (0.50)**
		$m = 10$	$m = 15$	$m = 20$	$m = 25$	$m = 30$	
Polymerization reactor data set							
OS-ELMs	(*on-line single model*)	10.16 (1.31)	10.65 (1.23)	14.38 (4.65)	9.42 (0.79)	8.91 (0.96)	10.70 (2.16)
OS-ELMb	(*on-line single model*)	11.94 (1.76)	12.65 (2.70)	19.14 (10.70)	14.47 (2.08)	14.22 (2.14)	14.48 (2.81)
OS-ELMs-SW	(*on-line single model*)	0.49 (0.09)	0.62 (0.09)	0.83 (0.21)	0.99 (0.20)	1.02 (0.20)	0.79 (0.23)
OS-ELMb-SW	(*on-line single model*)	1.79 (0.47)	2.70 (0.48)	5.26 (0.98)	4.30 (0.46)	5.84 (1.31)	3.98 (1.71)
EOS-ELM	(*on-line ensemble*)	8.14 (0.41)	8.08 (0.42)	13.75 (2.80)	7.76 (0.38)	5.90 (0.50)	8.72 (2.95)
OB	(*on-line ensemble*)	8.42 (0.40)	8.03 (0.28)	11.18 (1.26)	7.57 (0.30)	5.79 (0.19)	8.20 (1.95)
Learn^{++}.NSE	(*on-line ensemble*)	4.18 (0.68)	5.30 (0.61)	6.39 (0.68)	6.27 (0.89)	8.42 (1.91)	6.11 (1.57)
OAUE	(*on-line ensemble*)	2.77 (0.21)	3.13 (0.25)	2.99 (0.46)	2.99 (0.32)	3.58 (0.28)	3.09 (0.30)
AddExp	(*on-line ensemble*)	2.80 (0.09)	2.78 (0.13)	2.73 (0.14)	2.75 (0.15)	2.61 (0.13)	2.73 (0.07)
OWE	(*on-line ensemble*)	0.55 (0.06)	0.77 (0.09)	0.99 (0.10)	1.06 (0.11)	1.29 (0.10)	0.93 (0.29)
DOER	(*on-line ensemble*)	**0.47 (0.03)**	**0.58 (0.04)**	**0.67 (0.07)**	**0.75 (0.06)**	**0.81 (0.06)**	**0.66 (0.14)**
Cement kiln data set							
OS-ELMs	(*on-line single model*)	22.92 (2.52)	23.24 (2.06)	22.58 (3.26)	21.96 (3.14)	19.67 (2.11)	22.07 (1.42)
OS-ELMb	(*on-line single model*)	25.31 (1.70)	26.06 (2.04)	25.74 (2.77)	25.12 (2.51)	26.62 (2.24)	25.77 (0.60)
OS-ELMs-SW	(*on-line single model*)	12.72 (3.88)	12.11 (0.50)	12.41 (0.68)	12.12 (0.43)	12.44 (0.60)	12.36 (0.25)
OS-ELMb-SW	(*on-line single model*)	16.42 (1.66)	20.45 (2.55)	20.28 (2.95)	21.73 (3.01)	27.07 (2.84)	21.19 (3.84)
EOS-ELM	(*on-line ensemble*)	22.88 (0.95)	22.01 (1.94)	22.77 (1.75)	18.99 (2.62)	19.13 (2.69)	21.16 (1.94)
OB	(*on-line ensemble*)	20.61 (0.85)	19.10 (1.15)	18.99 (1.10)	16.86 (0.70)	16.42 (0.65)	18.40 (1.73)
Learn^{++}.NSE	(*on-line ensemble*)	17.31 (0.65)	21.99 (1.78)	22.42 (1.57)	23.44 (2.09)	27.06 (1.96)	22.44 (3.50)
OAUE	(*on-line ensemble*)	11.98 (0.26)	12.64 (0.32)	12.48 (0.39)	12.55 (0.56)	13.02 (0.40)	12.53 (0.37)
AddExp	(*on-line ensemble*)	12.13 (0.25)	12.29 (0.25)	12.33 (0.28)	12.03 (0.33)	11.98 (0.35)	12.15 (0.15)
OWE	(*on-line ensemble*)	11.33 (0.32)	12.05 (0.43)	12.51 (0.42)	13.08 (0.62)	13.38 (0.56)	12.47 (0.82)
DOER	(*on-line ensemble*)	**10.06 (0.16)**	**10.20 (0.21)**	**10.21 (0.12)**	**10.02 (0.27)**	**10.14 (0.20)**	**10.13 (0.08)**

The MSE values have been multiplied by 10^3. Average and SD of MSE are obtained on 20 trials of the algorithms. Bold values indicate the lowest error of a data set on a window's size. The last column reports the average and SD of the error of each approach on all window's sizes.

Table 6.4: MSE results of the on-line learning algorithms using the thermal oxidizer data set, the powder detergent data set and the debutanizer column data set.

Method/Data set		Window's size					All the
		$m = 20$	$m = 30$	$m = 40$	$m = 50$	$m = 60$	window's sizes
Thermal oxidizer data set							
OS-ELMs	(*on-line single model*)	17.86 (6.37)	14.67 (2.93)	15.15 (4.73)	14.33 (3.55)	14.41 (3.00)	15.28 (1.48)
OS-ELMb	(*on-line single model*)	20.36 (5.35)	18.42 (5.26)	17.06 (2.67)	18.04 (3.26)	19.50 (2.98)	18.68 (1.29)
OS-ELMs-SW	(*on-line single model*)	8.12 (0.35)	8.14 (0.31)	8.19 (0.41)	8.07 (0.46)	8.03 (0.48)	8.11 (0.06)
OS-ELMb-SW	(*on-line single model*)	17.23 (16.29)	20.69 (5.86)	30.19 (31.66)	27.96 (8.57)	29.27 (7.06)	25.07 (5.77)
EOS-ELM	(*on-line ensemble*)	14.22 (1.88)	12.89 (1.50)	11.30 (0.70)	11.52 (0.72)	11.37 (0.82)	12.26 (1.27)
OB	(*on-line ensemble*)	14.80 (1.24)	12.21 (0.71)	11.45 (0.54)	11.15 (0.42)	11.50 (0.48)	12.22 (1.49)
Learn^{++}.NSE	(*on-line ensemble*)	12.14 (1.12)	17.92 (2.94)	18.26 (4.19)	28.35 (13.94)	24.15 (10.30)	20.16 (6.24)
OAUE	(*on-line ensemble*)	6.90 (0.08)	7.19 (0.08)	7.41 (0.16)	7.72 (0.19)	7.83 (0.14)	7.41 (0.38)
AddExp	(*on-line ensemble*)	6.84 (0.13)	6.63 (0.07)	6.55 (0.10)	6.54 (0.08)	6.60 (0.07)	6.63 (0.12)
OWE	(*on-line ensemble*)	8.39 (0.46)	9.02 (1.14)	8.76 (0.43)	8.41 (0.38)	8.40 (0.39)	8.60 (0.28)
DOER	(*on-line ensemble*)	**6.48 (0.06)**	**6.33 (0.06)**	**6.23 (0.06)**	**6.16 (0.07)**	**6.15 (0.07)**	**6.27 (0.14)**
Powder detergent data set							
OS-ELMs	(*on-line single model*)	7.09 (0.62)	7.05 (0.67)	6.86 (0.61)	6.58 (0.29)	6.48 (0.28)	6.81 (0.28)
OS-ELMb	(*on-line single model*)	7.77 (0.74)	7.64 (0.60)	7.62 (0.64)	7.46 (0.30)	7.48 (0.33)	7.59 (0.13)
OS-ELMs-SW	(*on-line single model*)	6.79 (0.29)	6.63 (0.39)	6.40 (0.25)	6.30 (0.24)	6.04 (0.20)	6.43 (0.29)
OS-ELMb-SW	(*on-line single model*)	11.24 (1.21)	10.09 (1.32)	9.79 (1.05)	10.43 (1.22)	9.98 (0.70)	10.31 (0.57)
EOS-ELM	(*on-line ensemble*)	6.79 (0.60)	6.24 (0.37)	6.02 (0.28)	5.91 (0.11)	6.00 (0.10)	6.19 (0.36)
OB	(*on-line ensemble*)	6.54 (0.23)	6.11 (0.15)	5.91 (0.08)	5.83 (0.08)	5.89 (0.06)	6.06 (0.29)
Learn^{++}.NSE	(*on-line ensemble*)	7.67 (0.40)	7.40 (0.28)	7.71 (1.08)	8.07 (0.52)	7.95 (0.32)	7.76 (0.26)
OAUE	(*on-line ensemble*)	5.07 (0.06)	5.23 (0.07)	5.33 (0.08)	5.44 (0.06)	5.59 (0.14)	5.33 (0.20)
AddExp	(*on-line ensemble*)	5.32 (0.08)	5.27 (0.08)	5.28 (0.08)	5.31 (0.10)	5.30 (0.08)	5.30 (0.02)
OWE	(*on-line ensemble*)	6.09 (0.14)	6.20 (0.18)	6.15 (0.13)	6.06 (0.11)	6.15 (0.11)	6.13 (0.05)
DOER	(*on-line ensemble*)	**4.79 (0.06)**	**4.75 (0.05)**	**4.83 (0.07)**	**4.89 (0.07)**	**4.96 (0.05)**	**4.84 (0.08)**
Debutanizer column data set							
OS-ELMs	(*on-line single model*)	10.86 (0.69)	10.28 (0.74)	10.10 (0.68)	9.94 (0.39)	9.79 (0.50)	10.19 (0.41)
OS-ELMb	(*on-line single model*)	11.90 (0.50)	12.45 (0.52)	12.72 (0.62)	13.18 (0.55)	13.85 (0.56)	12.82 (0.74)
OS-ELMs-SW	(*on-line single model*)	2.73 (0.27)	3.40 (0.28)	4.24 (0.30)	5.51 (0.76)	5.67 (0.38)	4.31 (1.28)
OS-ELMb-SW	(*on-line single model*)	22.03 (18.83)	20.24 (3.53)	34.70 (7.11)	27.48 (2.87)	35.83 (8.77)	28.06 (7.11)
EOS-ELM	(*on-line ensemble*)	10.56 (0.31)	10.02 (0.28)	9.86 (0.32)	9.58 (0.21)	9.25 (0.25)	9.85 (0.49)
OB	(*on-line ensemble*)	10.38 (0.20)	9.83 (0.16)	9.77 (0.15)	9.49 (0.18)	9.28 (0.13)	9.75 (0.42)
Learn^{++}.NSE	(*on-line ensemble*)	12.83 (2.34)	15.38 (2.53)	24.73 (4.06)	20.68 (4.69)	25.21 (3.89)	19.76 (5.53)
OAUE	(*on-line ensemble*)	7.32 (0.08)	7.92 (0.09)	8.25 (0.11)	8.54 (0.13)	8.73 (0.14)	8.15 (0.56)
AddExp	(*on-line ensemble*)	5.77 (0.17)	5.94 (0.12)	6.01 (0.20)	6.25 (0.12)	6.38 (0.11)	6.07 (0.24)
OWE	(*on-line ensemble*)	6.00 (0.39)	7.09 (0.27)	9.93 (0.73)	13.21 (1.03)	14.59 (1.48)	10.16 (3.73)
DOER	(*on-line ensemble*)	**1.98 (0.05)**	**2.48 (0.07)**	**3.04 (0.08)**	**3.57 (0.16)**	**3.95 (0.14)**	**3.00 (0.80)**

The MSE values have been multiplied by 10^3. Average and SD of MSE are obtained on 20 trials of the algorithms. Bold values indicate the lowest error of a data set on a window's size. The last column reports the average and SD of the error of each approach on all window's sizes.

Table 6.5: MSE results of the on-line learning algorithms using the SRU data set.

Method/Data set		Window's size ($m = 30$)	($m = 60$)	($m = 90$)	($m = 120$)	($m = 150$)	All the window's sizes
SRU data set (output 1)							
OS-ELMs	(*on-line single model*)	0.51 (0.01)	0.51 (0.01)	0.51 (0.01)	0.51 (0.00)	0.51 (0.00)	0.51 (0.00)
OS-ELMb	(*on-line single model*)	0.55 (0.01)	0.58 (0.03)	0.59 (0.08)	0.63 (0.04)	0.63 (0.09)	0.59 (0.04)
OS-ELMs-SW	(*on-line single model*)	0.39 (0.03)	0.46 (0.03)	0.46 (0.01)	0.46 (0.01)	0.47 (0.01)	0.45 (0.03)
OS-ELMb-SW	(*on-line single model*)	1.92 (1.61)	1.82 (0.45)	2.11 (0.54)	2.14 (0.82)	2.19 (0.72)	2.03 (0.16)
EOS-ELM	(*on-line ensemble*)	0.51 (0.00)	0.50 (0.00)	0.50 (0.00)	0.50 (0.00)	0.50 (0.00)	0.50 (0.00)
OB	(*on-line ensemble*)	0.50 (0.00)	0.50 (0.00)	0.50 (0.00)	0.50 (0.00)	0.50 (0.00)	0.50 (0.00)
Learn^{++}.NSE	(*on-line ensemble*)	0.98 (0.28)	1.18 (0.19)	1.29 (0.25)	1.33 (0.21)	1.68 (0.54)	1.29 (0.26)
OAUE	(*on-line ensemble*)	0.44 (0.00)	0.47 (0.00)	0.48 (0.00)	0.48 (0.00)	0.49 (0.00)	0.47 (0.02)
AddExp	(*on-line ensemble*)	0.49 (0.01)	0.49 (0.00)	0.49 (0.00)	0.49 (0.00)	0.49 (0.00)	0.49 (0.00)
OWE	(*on-line ensemble*)	0.60 (0.02)	0.66 (0.02)	0.64 (0.02)	0.65 (0.02)	0.66 (0.01)	0.64 (0.03)
DOER	(*on-line ensemble*)	**0.27 (0.00)**	**0.32 (0.00)**	**0.34 (0.00)**	**0.35 (0.00)**	**0.37 (0.00)**	**0.33 (0.04)**
SRU data set (output 2)							
OS-ELMs	(*on-line single model*)	1.45 (0.01)	1.45 (0.00)	1.45 (0.01)	1.44 (0.01)	1.44 (0.01)	1.44 (0.01)
OS-ELMb	(*on-line single model*)	1.58 (0.05)	1.60 (0.03)	1.66 (0.12)	1.81 (0.35)	1.70 (0.10)	1.67 (0.09)
OS-ELMs-SW	(*on-line single model*)	1.10 (0.11)	1.35 (0.10)	1.37 (0.03)	1.40 (0.04)	1.40 (0.03)	1.32 (0.13)
OS-ELMb-SW	(*on-line single model*)	4.37 (0.58)	6.58 (1.91)	6.02 (1.37)	6.47 (2.26)	7.36 (2.64)	6.16 (1.11)
EOS-ELM	(*on-line ensemble*)	1.41 (0.01)	1.41 (0.01)	1.41 (0.01)	1.41 (0.00)	1.41 (0.00)	1.41 (0.00)
OB	(*on-line ensemble*)	1.41 (0.00)	1.42 (0.00)	1.41 (0.00)	1.41 (0.00)	1.41 (0.00)	1.41 (0.00)
Learn^{++}.NSE	(*on-line ensemble*)	2.33 (0.24)	3.19 (0.75)	3.75 (0.85)	4.03 (0.94)	4.13 (1.08)	3.49 (0.74)
OAUE	(*on-line ensemble*)	1.31 (0.01)	1.37 (0.01)	1.39 (0.00)	1.39 (0.01)	1.39 (0.01)	1.37 (0.04)
AddExp	(*on-line ensemble*)	1.39 (0.01)	1.38 (0.01)	1.38 (0.01)	1.38 (0.01)	1.38 (0.01)	1.38 (0.00)
OWE	(*on-line ensemble*)	1.59 (0.04)	1.95 (0.04)	1.98 (0.04)	1.93 (0.05)	1.93 (0.06)	1.88 (0.16)
DOER	(*on-line ensemble*)	**0.74 (0.01)**	**0.89 (0.01)**	**0.96 (0.01)**	**1.02 (0.01)**	**1.05 (0.01)**	**0.93 (0.12)**

The MSE values have been multiplied by 10^3. Average and SD of MSE are obtained on 20 trials of the algorithms. Bold values indicate the lowest error of a data set on a window's size. The last column reports the average and SD of the error of each approach on all window's sizes.

The results show that when m decreases, the OAUE's performance increases. In the DOER, in general, when m is low, DOER achieves high accuracy. The value of m is related to the number of samples to be used for training a new model and to measure the models' performances. As can be seen, m plays an important role on all the approaches.

The OS-ELMs-SW has better performance in the real-world data sets than in the artificial data sets, being a remarkable improvement of performance when compared to what happens in similar comparisons made with the other on-line methods. OS-ELMs-SW has high accuracy in scenarios where the most recent samples are sufficient to describe the system. However, the OS-ELMs-SW requires high computational time when compared to the other algorithms, since in the OS-ELMs-SW, a new model is trained when a new sample is available. The OS-ELMb-SW has low computation time, because a new model is trained only when a batch is available. However, the OS-ELMb-SW has larger error than the OS-ELMs-SW, since in the OS-ELMb-SW, the system is adapted on a batch basis, taking more time to adapt to the new concepts.

The experiments indicated that the presented industrial applications require faster adaptation capability. Therefore, on-line ensembles with few adaptive mechanisms have worse performances, as for example the EOS-ELM and OB which employ few adaptive ensemble mechanisms, since only the models' parameters (retraining of models) are adapted. On both the methods, the ensembles react slowly to changes. The OB slightly outperforms the EOS-ELM. In the OB, each new sample can be presented more times for retraining each model in comparison to the EOS-ELM, so that the OB can adapt faster.

Learn^{++}.NSE employs more adaptive ensemble mechanisms when compared to EOS-ELM and OB. However, the Learn^{++}.NSE's performance is usually worse when compared to them, since the ensemble is adapted on a batch basis. Additionally, Learn^{++}.NSE does not perform retraining of the models. In contrast to Learn^{++}.NSE, OAUE retrains all the models when a sample is available. However, OAUE adds new models to the ensemble at a low frequency (batch frequency) when compared to AddExp and DOER; and in most cases, the tests show that AddExp and DOER outperform OAUE. AddExp employs the same adaptive ensemble mechanisms ((*i*), (*ii*), and (*iii*), defined in Section 3.5.2) as the DOER. However, the AddExp's performance is inferior in the Tables 6.3 and 6.5. As described before, in

the AddExp, new models trained on the new concepts take more time to have their combination weights significantly increased. In scenarios that require faster adaptation to the new concepts, this strategy may fail. In contrast to AddExp, DOER assigns high combination weights to the new and accurate models if they have low errors on the recent samples.

Table 6.6 and Table 6.7 show the processing time (in seconds) of all the approaches in all on-line samples (AOS) and per on-line sample (POS) using all the data sets. The results reveal that sample-based windowing techniques (OS-ELMs-SW, DOER, and AddExp) are more time consuming when compared to the sample-based techniques that do not use a window (EOS-ELM, OB, etc). This is because, sample-based windowing techniques require that more models are trained. Therefore, in most cases, DOER is more time consuming than other ensembles, since more models are designed. Nevertheless, in most cases, DOER gives better accuracy than other methods. For the real-world data sets, all the approaches can perform on-line prediction for each sample in less than 1 second.

Table 6.6: Processing time (seconds) of all the approaches in all on-line samples (AOS) and per on-line sample (POS) when $m = 60$ using the artificial data sets.

Method/	Hyperplane	Friedman-LA	Friedman-GRA	Friedman-GnRG
Data Set	AOS / POS	AOS / POS	AOS / POS	AOS / POS
OS-ELMs	1.7 / 8.7×10^{-4}	1.4 / 7.0×10^{-4}	1.4 / 7.0×10^{-4}	1.8 / 9.0×10^{-4}
OS-ELMb	1.5 / 7.4×10^{-4}	1.1 / 5.4×10^{-4}	1.1 / 5.6×10^{-4}	1.1 / 5.5×10^{-4}
OS-ELMs-SW	1089.1 / 5.5×10^{-1}	909.1 / 4.6×10^{-1}	1124.6 / 5.7×10^{-1}	1130.7 / 5.7×10^{-1}
OS-ELMb-SW	7.7 / 3.9×10^{-3}	11.2 / 5.7×10^{-3}	7.7 / 3.9×10^{-3}	8.3 / 4.2×10^{-3}
EOS-ELM	41.4 / 2.1×10^{-2}	44.3 / 2.2×10^{-2}	44.1 / 2.2×10^{-2}	38.8 / 2.0×10^{-2}
OB	43.7 / 2.2×10^{-2}	45.6 / 2.3×10^{-2}	50.7 / 2.6×10^{-2}	45.2 / 2.3×10^{-2}
Learn^{++}.NSE	36.2 / 1.8×10^{-2}	38.6 / 2.0×10^{-2}	36.1 / 1.8×10^{-2}	38.3 / 1.9×10^{-2}
OAUE	79.2 / 4.0×10^{-2}	76.5 / 3.9×10^{-2}	74.6 / 3.8×10^{-2}	75.7 / 3.8×10^{-2}
AddExp	555.0 / 2.8×10^{-1}	724.7 / 3.7×10^{-1}	801.5 / 4.1×10^{-1}	910.2 / 4.6×10^{-1}
OWE	455.6 / 2.3×10^{-1}	410.7 / 2.1×10^{-1}	539.5 / 2.7×10^{-1}	754.6 / 3.8×10^{-1}
DOER	707.0 / 3.6×10^{-1}	634.8 / 3.2×10^{-1}	825.1 / 4.2×10^{-1}	837.1 / 4.2×10^{-1}

6.3.5 Discussion

In summary, the results in the previous Subsections reaffirm the superiority of DOER over the other methodologies in both the real scenarios and the artificial scenarios.

Table 6.7: Processing time (seconds) of all the approaches in all on-line samples (AOS) and per on-line sample (POS) when $m = 30$ using the real-world data sets.

Method/	Polymerization reactor	Cement kiln	Thermal oxidizer	Powder detergent
Data set	AOS / POS	AOS / POS	AOS / POS	AOS / POS
OS-ELM*s*	0.6 / 9.5×10^{-4}	0.6 / 9.3×10^{-4}	1.4 / 6.8×10^{-4}	1.3 / 6.5×10^{-4}
OS-ELM*b*	0.6 / 9.0×10^{-4}	0.6 / 8.2×10^{-4}	1.1 / 5.4×10^{-4}	1.1 / 5.5×10^{-4}
OS-ELM*s*-SW	365.9 / 5.9×10^{-1}	428.7 / 6.4×10^{-1}	1190.7 / 5.9×10^{-1}	1024.7 / 5.3×10^{-1}
OS-ELM*b*-SW	7.2 / 1.2×10^{-2}	9.2 / 1.4×10^{-2}	29.1 / 1.4×10^{-2}	29.6 / 1.5×10^{-2}
EOS-ELM	21.2 / 3.4×10^{-2}	22.8 / 3.4×10^{-2}	43.4 / 2.1×10^{-2}	37.4 / 1.9×10^{-2}
OB	8.9 / 1.4×10^{-2}	12.0 / 1.8×10^{-2}	33.0 / 1.6×10^{-2}	42.7 / 2.2×10^{-2}
Learn^{++}.NSE	14.4 / 2.3×10^{-2}	18.5 / 2.8×10^{-2}	65.0 / 3.2×10^{-2}	57.1 / 3.0×10^{-2}
OAUE	15.0 / 2.4×10^{-2}	37.3 / 5.6×10^{-2}	137.8 / 6.8×10^{-2}	132.5 / 6.9×10^{-2}
AddExp	95.5 / 1.5×10^{-1}	242.8 / 3.6×10^{-1}	635.1 / 3.1×10^{-1}	429.1 / 2.2×10^{-1}
OWE	84.8 / 1.4×10^{-1}	211.1 / 3.1×10^{-1}	669.4 / 3.3×10^{-1}	468.8 / 2.4×10^{-1}
DOER	140.0 / 2.3×10^{-1}	294.2 / 4.4×10^{-1}	896.4 / 4.4×10^{-1}	597.2 / 3.1×10^{-1}

	Debutanizer column	SRU (output 1)	SRU (output 2)
	AOS / POS	AOS / POS	AOS / POS
OS-ELM*s*	1.2 / 6.7×10^{-4}	3.9 / 5.6×10^{-4}	4.0 / 5.8×10^{-4}
OS-ELM*b*	1.1 / 5.9×10^{-4}	3.1 / 4.6×10^{-4}	3.1 / 4.6×10^{-4}
OS-ELM*s*-SW	1206.4 / 6.7×10^{-1}	3966.1 / 5.8×10^{-1}	3533.8 / 5.2×10^{-1}
OS-ELM*b*-SW	27.2 / 1.5×10^{-2}	93.2 / 1.4×10^{-2}	97.0 / 1.4×10^{-2}
EOS-ELM	38.6 / 2.1×10^{-2}	116.6 / 1.7×10^{-2}	123.2 / 1.8×10^{-2}
OB	40.7 / 2.3×10^{-2}	126.4 / 1.8×10^{-2}	131.2 / 1.9×10^{-2}
Learn^{++}.NSE	55.9 / 3.1×10^{-2}	215.4 / 3.1×10^{-2}	205.5 / 3.0×10^{-2}
OAUE	124.4 / 6.9×10^{-2}	705.3 / 1.0×10^{-1}	668.7 / 9.9×10^{-2}
AddExp	545.2 / 3.0×10^{-1}	200.0 / 2.9×10^{-2}	752.8 / 1.1×10^{-1}
OWE	679.6 / 3.8×10^{-1}	522.9 / 7.6×10^{-2}	551.4 / 8.1×10^{-2}
DOER	799.1 / 4.4×10^{-1}	912.3 / 1.3×10^{-1}	810.1 / 1.1×10^{-1}

SW algorithms using a single model assume that samples that fall outside the window are irrelevant, and such algorithms do not have capability to handle the previously acquired data, since old data are discarded. On the other hand, algorithms able to retain the previously acquired knowledge (e.g. EOS-ELM, and OB) have difficulty to adapt quickly to changes, since the old data are still relevant to the learned model. But even if the algorithm is able to conciliate previous data and current data, some algorithms have slow adaptation capability, because a long time is required to introduce new knowledge to the system.

The results in this Chapter revealed interesting characteristics of the proposed

DOER method. DOER attains an error lower than the best single model, the OS-ELM*s*-SW. The results also indicate that DOER has capability to deal with changing environments; and they also indicate that the correct setting of the factor of including a new model to the ensemble, α, is an important issue to control the system's adaptation capability. For example, in scenarios with fewer concept changes, α should be set to approximately 0.10; while in scenarios that require fast adaptation (as in the cement kiln data set or in the polymerization reactor data set), α should be set to 0.04, since low values of α do not improve significantly the DOER's performance; while in other scenarios (e.g. SRU data set), the DOER's performance does not improve when α varies. It has also been observed that the accuracy of each approach is related to the setting of m. Therefore, m should be set using some knowledge about the data or using a value proportional to the data set's size.

In contrast to other approaches, DOER does not perform any down-weighting mechanism of the old models. Additionally, a recently created model can have a contribution similar to the other accurate models from the ensemble, if they predict well the recent samples. Therefore, old and new models can have the same contribution to the ensemble if they predict accurately the recent set of samples. It has been observed that this characteristic is important to increase the ensemble's accuracy. Moreover, the DOER's weighting strategy decreases exponentially the contribution of models that perform poorly on the current window, not allowing that they affect the ensemble's accuracy.

6.4 Conclusion

This Chapter proposed a dynamic and on-line ensemble regression (DOER) method for on-line prediction of variables in changing environments, with application, for example, for predicting variables measured with significant delay, as in soft sensing applications. The main contribution of this Chapter is the proposal of an on-line ensemble for regression that incorporates three different levels of adaptation (dynamic inclusion and removal of models, models' combination weights adaptation, and models' parameters adaptation), which enable to maintain the system's performance in changing environments. DOER was shown to have higher accuracy when compared to state-of-the-art approaches in scenarios that require adaptation

capability, and with non-recurring drifts. According to the simulation results of soft sensing applications, the proposed method can deliver more accurate predictions of the key variables in industrial processes when compared to the traditional SW approach using a single model, commonly used in soft sensing applications. Therefore, the proposed method can be designed for practical use in industrial applications, reducing the time and maintenance costs of traditional systems (e.g. laboratory measurement systems).

The window's size setting may have an important role in some data sets, e.g. the debutanizer column data set. Therefore, in these cases, it is important to apply extra experiments using the proposed method to define the window's size. Also, as a future work, it is proposed a variable window's size that adapts according to the process dynamics and characteristics. Moreover, it seems to be interesting to propose an adaptive setting of α. In this way, α should be set to a high value when a change occurs, and to a low value when no changes are detected. Additionally, future efforts can also be devoted to apply fast dynamic optimization strategies for on-line selection of the best subset of models from the ensemble.

Therefore, the aim of proposing an adaptive ensemble with fast adaptation capability for on-line predictions in changing environments was reached in this Chapter. The experimental results showed that the proposed methodology can predict important variables in industrial SS applications more accurately than well-known single learning algorithms and ensemble learning algorithms.

Chapter 7

An Adaptive Ensemble Using Ordered Aggregation

Contents

7.1 Introduction

Many practical systems, such as industrial plants, exhibit time-varying behavior, being very difficult for ELM models to react to the changes. Recently, ELM models for dynamic environments have been proposed. The most popular is the OS-ELM model [Liang *et al.*, 2006]. In [Deng *et al.*, 2014], it is proposed an On-line Sequential Reduced Kernel ELM that is incrementally updated based on the new samples' confidence estimation. In [Matias *et al.*, 2013], a forgetting factor (FF) is introduced to the OS-ELM algorithm. When the FF value is close to 1 more contribution is given to the old samples, and when the FF value is close to 0 more importance is given to the recent samples. Since a fixed value for the FF may not be sufficient to track all the system dynamics, a variable FF for the OS-ELM is proposed in [Lim *et al.*, 2013]. The FF is adapted using a gradient (derivative) descent method, derived from a cost function of the RLS. This method depends on the appropriate step size and takes too many iterations to converge to the appropriate FF value. RLS with FF discounts continuously the old data even when the new data does not carry sufficient information, producing a phenomenon known as *windup* [Cao and Schwartz, 2000]. As a consequence, values of the information matrix will tend to zero and the model gain will tend to be unbounded, so that the model becomes sensitive to noises. The directional FF (DFF) method can overcome this effect [Bobál *et al.*, 2005]. It considers that the data has directions, and the old samples are forgotten only in some specific directions.

Although on-line ELM models have shown good performance in real-world applications, Chapter 6 showed that a combination of multiple on-line ELM models (ensemble systems) is more accurate than a single on-line ELM model. Chapter 4 showed that the optimal subset selection of models from a set of models (ensemble pruning) is a valuable tool to increase the ensemble accuracy. In off-line ensembles, a subset of models is usually selected by GA [Soares *et al.*, 2013] or Greedy optimizations [Partalas *et al.*, 2008] based on the ensemble error and/or ensemble diversity. However, these methods are computationally expensive for on-line applications, so that other methods should be preferred. The Ordered Aggregation (OA) technique uses some measure in order to produce a decreasing order of the best models for a given data. In [Coelho and Zuben, 2006] models are ordered according their accuracies on a validation data set; while in [Lazarevic and Obradovic, 2001] models are

ranked according to their accuracies and their diversities.

This Chapter proposes a new sample-based On-line Ensemble of regressor models using Ordered Aggregation (OEOA) which is able to provide on-line prediction of variables in changing environments. The proposed ensemble incorporates all the previous contributions and characteristics of DOER (Chapter 6). However, OEOA dynamically selects an optimal ensemble size and composition of the subset of models based on the minimization of the ensemble error on the newest sample. Then, the models are ordered based on their on-line prediction errors and the best models of the ordered sequence are employed to obtain the ensemble's output. OEOA builds an ensemble based on a SW. A new model is trained (with samples of the current window) and added, if the current ensemble's error is higher than a threshold. The error of each model is obtained using a window that is filled with its predictive errors on the most recent on-line samples. The models' combination weights are dynamically assigned according to their prediction errors. Inaccurate models are removed for assuring adaptation of the ensemble in changing environments. The proposed ensemble overcomes the problem of defining the optimal ensemble size, and in most cases it obtains better performance than aggregating all the models.

As a base model for the ensemble, this Chapter proposes an OS-ELM model using DFF (termed as λ_{DFF}OS-ELM), a new algorithm which shows superior accuracy in industrial applications when compared to the well-known OS-ELM model. Experiments are reported to demonstrate the performance, effectiveness, and faster adaptation capability and accuracy of the proposed methods. Furthermore, a thorough analysis of the experimental results using on-line ensembles of the state-of-the-art and OEOA, is presented and reveals that the performances of these methods can be significantly improved using λ_{DFF}OS-ELM as the base model in industrial data sets.

The Chapter is organized as follows. Section 7.2 describes the λ_{DFF}OS-ELM algorithm. Section 7.3 describes the OEOA algorithm. The experimental results are presented and analyzed in Section 7.4. Section 7.5 presents concluding remarks.

7.2 An OS-ELM Model with DFF

An OS-ELM model with variable FF, called λ_{DFF}OS-ELM, is proposed in this Section. It is based on the assumption that *a priori* selection of the FF may not

be able to track all the system dynamics. Using the DFF method, λ_{DFF}OS-ELM is adapted only when the new data contains sufficient information so that the windup phenomenon [Cao and Schwartz, 2000] is avoided.

Similarly to the OS-ELM algorithm, described in Subsection 3.5.4, the λ_{DFF}OS-ELM has two phases: *initialization phase* and *on-line learning phase*. In the initialization phase, a training data set of size m, $\mathbf{D}^0 = \{(\mathbf{x}_t, y_t)\}_{t=1}^{m}$ from a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$ (with $m < T$), is used to train an initial model. In the on-line learning phase, on-line samples from a data set $\mathbf{D}^{online} = \{(\mathbf{x}_t, y_t)\}_{t=m+1}^{T}$ are given incrementally one-by-one for on-line retraining of the model.

Similarly to ELM and OS-ELM, the main objective is to obtain an output synaptic weights vector $\boldsymbol{\beta}$, so that $\mathbf{H}\boldsymbol{\beta} = \mathbf{y}$. For more details, see Subsection 3.3.1 and Subsection 3.5.4. In the initialization phase, the initial output synaptic weights vector $\boldsymbol{\beta}_0$ is obtained as:

$$\boldsymbol{\beta}_0 = \mathbf{P}_0\mathbf{H}_0^T\mathbf{y}_0, \tag{7.1}$$

where $\mathbf{P}_0 = (\mathbf{H}_0^T\mathbf{H}_0)^{-1}$; $\mathbf{y}_0 = [y_1, \ldots, y_m]^T$ (from $\mathbf{D}_0$); $\mathbf{H}_0$ is the initial hidden layer output matrix and it is obtained from Equation (3.23), where L is the number of hidden nodes; and $\boldsymbol{\beta}_0 = [\beta_1, \ldots, \beta_L]^T$.

In the on-line learning phase, when a new sample $(\mathbf{x}_t, y_t)$ from $\mathbf{D}^{online}$ is available, it is employed to obtain a new output synaptic weights vector $\boldsymbol{\beta}_{k+1}$ using the RLS with DFF [Mendes *et al.*, 2013; Bobál *et al.*, 2005] as follows:

$$\boldsymbol{\beta}_{k+1} = \boldsymbol{\beta}_k + \frac{\mathbf{P}_k\mathbf{h}_{k+1}}{1 + \xi_{k+1}}\hat{e}_{k+1} \ , \tag{7.2}$$

$$\hat{e}_{k+1} = y_t - \mathbf{h}_{k+1}^T\boldsymbol{\beta}_k, \tag{7.3}$$

$$\xi_{k+1} = \mathbf{h}_{k+1}^T\mathbf{P}_k\mathbf{h}_{k+1}, \tag{7.4}$$

$$\mathbf{h}_{k+1} = [g(\mathbf{a}_1, b_1, \mathbf{x}_t), \ldots, g(\mathbf{a}_L, b_L, \mathbf{x}_t)], \tag{7.5}$$

$k = t - m - 1$ and $k \geqslant 0$. If $\xi_{k+1} = 0$, then the covariance matrix $\mathbf{P}_k$ is obtained by $\mathbf{P}_{k+1} = \mathbf{P}_k$. Otherwise, if $\xi_{k+1} > 0$, then $\mathbf{P}_{k+1}$ is updated as:

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \frac{\mathbf{P}_k\mathbf{h}_{k+1}\mathbf{h}_{k+1}^T\mathbf{P}_k}{\varepsilon_{k+1}^{-1} + \xi_{k+1}} \ , \tag{7.6}$$

$$\varepsilon_{k+1} = \lambda_k - \frac{1-\lambda_k}{\xi_{k+1}}, \tag{7.7}$$

where λ_k is the FF at the k-th iteration, and it should be initialized as $0 < \lambda_0 \leqslant 1$. The FF for the $(k+1)$-th iteration is obtained as [Bobál *et al.*, 2005; Bobál and Chalupa, 2008], [Kulhavý, 1985, cited in [Bobál and Chalupa, 2008]]:

$$\lambda_{k+1} = \left\{1 + (1+\rho)\left[\ln(1+\xi_{k+1}) + \left(\frac{(\upsilon_{k+1}+1)\eta_{k+1}}{1+\xi_{k+1}+\eta_{k+1}} - 1\right)\frac{\xi_{k+1}}{1+\xi_{k+1}}\right]\right\}^{-1} \tag{7.8}$$

where

$$\eta_{k+1} = \hat{e}^2_{k+1}/\gamma_{k+1}, \tag{7.9}$$

$$\gamma_{k+1} = \lambda_k\left(\gamma_k + \frac{\hat{e}^2_{k+1}}{1+\xi_{k+1}}\right), \tag{7.10}$$

$$\upsilon_{k+1} = \lambda_k(\upsilon_k + 1), \tag{7.11}$$

and ρ is a positive constant. The initial values of γ and υ (i.e. γ_0 and υ_0) should be set between 0 and 1. The λ_{DFF}OS-ELM learning algorithm is summarized in Algorithm 7.1. Similarly to the OS-ELM algorithm, it is assumed that L distinct samples, with $L \leqslant m < T$, are included among the m samples contained in $\mathbf{D}^0$.

7.3 An On-line Ensemble Using Ordered Aggregation

OEOA designs an ensemble using a SW. It employs the common assumption that the most recent data provides the best and most relevant representation of the current state of the process and of the near-future state; thus only this data should be kept [Brzezinski and Stefanowski, 2014; Klinkenberg, 2005]. A data window of fixed size is kept, and when a new sample is available, it is added to the window, and the oldest sample is removed from the window. The data window is employed to train a new model when the ensemble's performance is deteriorating, and to obtain the models' prediction errors. Similarity to DOER, the main strategies of OEOA for achieving faster adaptivity in time-varying environments are: *sample-based ensemble* which offers higher performance and faster adaptivity when compared to batch-based ensembles; *models' combination weights adaptation*; *models' parameters adaptation*;

Algorithm 7.1 Learning algorithm for λ_{DFF}OS-ELM.

Input: a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$; a hidden layer activation function $g(x)$; a number of hidden nodes L; number of samples for the initialization phase m, including L distinct samples (where $L \leqslant m < T$); λ_0; γ_0; υ_0; ρ;

1. **Initialization/training phase:** consider a training data set $\mathbf{D}^0 = \{(\mathbf{x}_t, y_t)\}_{t=1}^{m}$;
   (a) Randomly assign input synaptic weights $\mathbf{a}_j$ and biases b_j, $j = 1, \ldots, L$;
   (b) Calculate matrix $\mathbf{H}_0$ using $\mathbf{D}^0$ and Equation (3.23);
   (c) Obtain the output synaptic weight $\boldsymbol{\beta}_0$ through Equation (3.24), where $\mathbf{P}_0 = (\mathbf{H}_0^T \mathbf{H}_0)^{-1}$, and $\mathbf{y}_0 = [y_1, \ldots, y_m]^T$;
2. **On-line learning phase:** consider a testing data set $\mathbf{D}^{online} = \{(\mathbf{x}_t, y_t)\}_{t=m+1}^{T}$; set $t = m$;

(a) **While** $t < T$ **do:**
   i. Set $t \leftarrow t + 1$; $k = t - m - 1$;
   ii. Obtain sample $(\mathbf{x}_t, y_t)$ from $\mathbf{D}^{online}$;
   iii. Obtain vector $\mathbf{h}_{k+1}$ using Equation (7.5);
   iv. Obtain $\boldsymbol{\beta}_{k+1}$, $\hat{e}_{k+1}$, and ξ_{k+1} using Equations (7.2)-(7.4), respectively;
   v. Obtain $\mathbf{P}_{k+1}$:

$$\mathbf{P}_{k+1} = \begin{cases} \mathbf{P}_k, & \text{if } \xi_{k+1} = 0, \\ \text{as Equations (7.6)-(7.7)}, & \text{if } \xi_{k+1} > 0; \end{cases}$$

   vi. Calculate η_{k+1}, γ_{k+1}, and υ_{k+1} using Equations (7.9)-(7.11), respectively;
   vii. Compute λ_{k+1} using Equation (7.8);

(b) **end while**

and *dynamic inclusion and removal of models*. But unlike DOER, OEOA selects dynamically the best subset of models from a set of models using an OA approach.

Firstly, consider a regression problem with a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$, where $\mathbf{x}_t \in \mathbb{R}^r$ and $y_t \in \mathbb{R}$, and a window's size m (with $m < T$). Consider an ensemble $\mathcal{E}$ with N_{max} models, where $\mathcal{E} = \{f_1, \ldots, f_{N_{max}}\}$ and $f_n \in \mathcal{E}$ represents a model. OEOA has two phases: creation of an initial pool of N_{max} models, and on-line learning phase. In the first phase, an initial data window $\mathbf{D}^t$ with the first m samples of $\mathbf{D}$ is employed to train the initial pool of models. In the second phase, samples $t = m + 1, \ldots, T$ from $\mathbf{D}$ are given one-by-one for on-line prediction and on-line learning. Therefore, for each t, a data window $\mathbf{D}^t$ keeps the most recent m

samples.

Before introducing the OEOA algorithm, Subsection 7.3.1 describes the models' characteristics. Then, Subsection 7.3.2 details the OEOA algorithm.

7.3.1 OEOA Component Models

Each model f_n from ensemble $\mathcal{E}$ is initially trained with samples from a data window $\mathbf{D}^t$ using the initialization phase of an on-line supervised learner (e.g. OS-ELM or λ_{DFF}OS-ELM). Similarly to the DOER approach (for more details, see Subsection 6.2.1), in the OEOA algorithm, the main parameters associated to a model f_n are: $life_n$ which denotes the total number of on-line predictions performed by f_n; w_n which is the combination weight of model f_n; and MSE_n^t which denotes the total prediction error of f_n at time t. The value of MSE_n^t is computed using Equation (6.2), and as described in Subsection 6.2.1.

7.3.2 OEOA Algorithm Description

OEOA selects a subset of its models to participate in forming the ensemble prediction. Ensemble model selection usually involves selecting an optimal subset of models by searching the space of all models' combinations. However, the computational complexity of such an approach is exponential in the number of models: an ensemble with N_{max} models involves searching a space of $(2^{N_{max}} - 1)$ non-empty solutions to minimize a cost function. OEOA sorts models according to their errors obtained on the on-line predictions to avoid exhaustive search. Then, the best N models (with $N \leqslant N_{max}$) in the ordered sequence are selected as the optimal subset of models for predicting each incoming sample. When the real output is available, the optimal subset size is determined so as to minimize the ensemble prediction error on the newest sample.

The proposed OEOA method is summarized in Algorithm 7.2. Factor α controls the inclusion of a new model based on the prediction error on the newest sample, as in the OWE and DOER algorithms, where $0 < \alpha < 1$. An ensemble $\mathcal{E} = \{f_1, \ldots f_{N_{max}}\}$ with $N_{max} > 1$ models is considered. To avoid the problem of reaching a small size of the subset of models [Santos *et al.*, 2009], a variable N_{min} was included to control the minimum size of the optimal subset in the ordered aggregation, where $1 < N_{min} \leqslant N_{max}$.

Algorithm 7.2 Learning algorithm for OEOA.

Input: a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$; m; an on-line supervised learner; factor for controlling the inclusion of a new model α; maximum N_{max} and minimum N_{min} number of models of the ensemble (where $1 < N_{min} \leqslant N_{max}$);

Creating a pool of N_{max} models:

1. Set $\mathcal{E} \leftarrow \emptyset$; $t = m$; number of considered best models $N = N_{max}$; current window $\mathbf{D}^t = \{(\mathbf{x}_t, y_t)\}_{t=1}^{m} \subset \mathbf{D}$;
2. **for** $n = 1, \ldots N_{max}$ **do**:
   (a) $f_n \leftarrow$ obtain a new model trained with $\mathbf{D}^t$ using the initialization phase of the on-line supervised learner (e.g. λ_{DFF}OS-ELM);
   (b) Set $life_n = 0$, $\text{MSE}_n^t = 0$, $w_n = 1$; Include f_n into the ensemble: $\mathcal{E} \leftarrow \mathcal{E} + \{f_n\}$;
3. **end for**
4. Build the vector of the MSE of the models: $\mathbf{MSE}^t = [\text{MSE}_1^t, \ldots, \text{MSE}_{N_{max}}^t]$;

On-line learning phase:

5. **for** $t = m+1, \ldots, T$ **do**:
   (a) Receive a new sample $(\mathbf{x}_t, y_t)$;
   (b) Slide the window: $\mathbf{D}^t = \mathbf{D}^{t-1} + (\mathbf{x}_t, y_t) - (\mathbf{x}_{t-m}, y_{t-m})$;
   (c) Get models' predictions $\mathbf{o}_t$, where $\mathbf{o}_t = [o_t^1, \ldots, o_t^{N_{max}}]$, and $o_t^i = f_i(\mathbf{x}_t)$;
   (d) Obtain the final prediction of the optimal subset:
   $F(\mathbf{x}_t) \leftarrow \text{OutputOEOA}(\mathcal{E}, N_{max}, N, \mathbf{MSE}^{t-1}, \mathbf{o}_t)$;
   (e) **if** $N_{min} \neq N_{max}$, determine a new value for N:
   Set minError $= \infty$;
   for $p = N_{min}, \ldots, N_{max}$ **do:**
      i. $\hat{o}_t^p \leftarrow \text{OutputOEOA}(\mathcal{E}, N_{max}, p, \mathbf{MSE}^{t-1}, \mathbf{o}_t)$;
      ii. Determine the error as $\epsilon_t^p = (y_t - \hat{o}_t^p)^2$;
      iii. **if** $\epsilon_t^p <$ minError, **then** set minError $= \epsilon_t^p$; $N = p$;
   end for
   (f) Update the models (for $n = 1, \ldots, N_{max}$):
      i. Obtain the error e_n^t of f_n for input $\mathbf{x}_t$: $e_n^t = (y_t - o_t^n)^2$;
      ii. Set $life_n \leftarrow life_n + 1$;
      iii. Obtain MSE_n^t using Equation (6.2);
      iv. Incrementally retrain model f_n using sample $(\mathbf{x}_t, y_t)$ and using (one iteration of) the on-line learning phase of the on-line supervised learner (e.g. λ_{DFF}OS-ELM);
   (g) Build vector $\mathbf{MSE}^t = [\text{MSE}_1^t, \ldots, \text{MSE}_{N_{max}}^t]$;
   (h) **if** $|(F(\mathbf{x}_t) - y_t)/y_t| > \alpha$
      i. $f_0 \leftarrow$ obtain a new model trained with $\mathbf{D}^t$ using the initialization phase of the on-line supervised learner; Set $life_0 = 0$, $\text{MSE}_0^t = 0$, and $w_0 = 1$;
      ii. Replace model $f_z \in \mathcal{E}$ by f_0, where $z = \text{argmax}_{n=1,\ldots,N_{max}} \left(\text{MSE}_n^t\right)$:
      $f_z \leftarrow f_0$; $life_z \leftarrow life_0$;
6. **end for**

Algorithm 7.3 OutputOEOA: output prediction based on the OA of the best models.

Input: Ensemble $\mathcal{E}$; maximum number of models of the ensemble N_{max}; number of models to be aggregated Q; MSE errors of all the models $\mathbf{MSE}^{t-1}$; predicted outputs of all the models $\mathbf{o}_t$;

1. Sort the elements of $\mathbf{MSE}^{t-1}$ in ascending order forming $\mathbf{MSE}^{t-1}_{Sort} = [\mathrm{MSE}^{t-1}_{ix_1}, \ldots \mathrm{MSE}^{t-1}_{ix_{N_{max}}}]$ and return a vector of indexes $\mathbf{IX}_{Sort} = [ix_1, \ldots ix_{N_{max}}]$ which contains the position of each element of $\mathbf{MSE}^{t-1}_{Sort}$ in vector $\mathbf{MSE}^{t-1}$;
2. Assign to $\mathbf{MSE}^{t-1}_{Top} = [\mathrm{MSE}^{t-1}_{ix_1}, \ldots \mathrm{MSE}^{t-1}_{ix_Q}]$ and $\mathbf{IX}_{Top} = [ix_1, \ldots ix_Q]$ the first Q elements from $\mathbf{MSE}^{t-1}_{Sort}$ and $\mathbf{IX}_{Sort}$, respectively;
3. **for** each $n \in \mathbf{IX}_{Top}$ **do:**

$$w_n = \begin{cases} 1, & \text{if } life_n = 0, \\ \exp\left(-\frac{\mathrm{MSE}^{t-1}_n - \mathrm{median}\left(\mathbf{MSE}^{t-1}_{Top}\right)}{\mathrm{median}\left(\mathbf{MSE}^{t-1}_{Top}\right)}\right), & \text{if } life_n > 0; \end{cases} \tag{7.12}$$

4. **end for**
5. Obtain the output prediction: $\hat{o}_{Top} = \left(\sum_{n \in \mathbf{IX}_{Top}} w_n o^n_t)\right) / \sum_{n \in \mathbf{IX}_{Top}} w_n$;

Output: $\hat{o}_{Top}$;

In Step 2 a pool of N_{max} models is created using the initialization phase of a generic on-line supervised learner. The models are trained using the initial window $\mathbf{D}^t = \{(\mathbf{x}_t, y_t)\}_{t=1}^m \subset \mathbf{D}$. When a new sample $(\mathbf{x}_t, y_t)$ is available (Step 5(a)), the window slides along the data (Step 5(b)). This operation adds the new sample $(\mathbf{x}_t, y_t)$ to the window and excludes the oldest sample $(\mathbf{x}_{t-m}, y_{t-m})$ from the window. In Step 5(d), the final output of the optimal subset of N models is given. It is obtained by a weighted sum of the models' outputs. This step is performed using the Algorithm 7.3, an algorithm that obtains the output prediction based on the ordered aggregation of the best models.

Algorithm 7.3 obtains a vector of indexes, $\mathbf{IX}_{Top} = [ix_1, \ldots ix_Q]$, of the Q best performing models of the ensemble with respect to the $\mathbf{MSE}^{t-1}$. The MSE values of this subset of models are kept in vector $\mathbf{MSE}^{t-1}_{Top}$. Step 3 of Algorithm 7.3 aims to obtain only the combination weights of the subset of models. Similarly to the DOER algorithm, Equation (7.12) transforms combination weights in such a way

that a model f_n with MSE_n^{t-1} around the median value receives a combination weight equal to 1. Models with MSE_n^{t-1} lower than the median have their combination weights exponentially increased, while models with MSE_n^{t-1} larger than the median have their combination weights exponentially decreased. A model with $life_n = 0$ created at time t is initialized with combination weight equal to 1. This criterion smooths the contribution of a new model at the time $t+1$, the time at which such model will be evaluated on-line for the first time.

Then, the question is how to determine the optimal subset size N for the next iteration. N is chosen so as to minimize the square error on the newest sample $(\mathbf{x}_t, y_t)$:

$$N = \underset{p=N_{min},\ldots,N_{max}}{\text{argmin}} \left(\epsilon_t^p\right), \tag{7.13}$$

where $\epsilon_t^p = (y_t - \hat{o}_t^p)^2$, and $\hat{o}_t^p$ is the output prediction of an ordered aggregation with the best p models of the ensemble with respect to the $\mathbf{MSE}^{t-1}$. This strategy may obtain a small value of N and induce the inclusion of only new models, since a new model f_n is initialized with $\text{MSE}_n^t = 0$. To prevent this case and ensure stability to the ensemble, N_{min} should be large (> 5). It is worth noting that when $N_{min} = N_{max}$ no ordered strategy is employed, and the ensemble has a fixed number of models, since $N = N_{max}$ in all iterations. This strategy is called as "OEOA without ordering". In this case, the algorithm has lower processing time and the algorithm has similar performance when compared to the DOER algorithm. However, the main advantage of the "OEOA with ordering" (specifically the main advantage of having $N_{min} \neq N_{max}$) is that it is not necessary to tune the ensemble size (but only N_{min} and N_{max}), and then the algorithm dynamically selects the optimal ensemble size; and in some cases, this strategy has higher accuracy when compared to ensembles of fixed size.

In Step 5(f) the parameters of all models are updated. All the models are re-trained, keeping the models updated on the current state of the process. Step 5(h) evaluates if a new model should be added. The criterion adds a new model when the absolute relative error of the ensemble on the newest sample is greater than α. The new model f_0 is trained using the samples from the current data window $\mathbf{D}^t$; and f_0 replaces the least accurate model of the ensemble. The criterion substitutes the model f_z with the highest error, MSE_z^t. A new model created at iteration t is never excluded by the pruning strategy at the same time t.

Table 7.1: Specifications of the real-world data sets used in the experiments.

Data set	# Samples	# Inputs (bef. preproc.)	# Inputs (af. preproc.)	Data Set Size
polymerization reactor	**648**	15	**10**	*small*
cement kiln process	**701**	195	**45**	*small*
powder detergent production	**1962**	14	**14**	*medium*
thermal oxidizer	**2053**	39	**39**	*medium*
debutanizer column	**2394**	7	**7**	*medium*

7.4 Experimental Results

In this Section, four artificial data sets and five real-world data sets with time-varying behavior are employed to demonstrate the predictive performance of λ_{DFF}OS-ELM and OEOA over state-of-the-art approaches. The experiments have been performed on the Matlab environment, running on a PC equipped with an Intel Core i7-4700MQ 2.4GHz-3.4GHz processor of 4 cores and 8GB of RAM.

7.4.1 Data Set Description

Artificial data sets. They are the same data sets as the ones employed in Chapter 5 and Chapter 6: the hyperplane data set [Kolter and Maloof, 2005]; the local and abrupt drift data set (Friedman-LA); the global recurring abrupt drift data set (Friedman-GRA); and the global non-recurring gradual drift data set (Friedman-GnRG) [Ikonomovska, 2012]. For more details, see Subsection 5.3.1.

Industrial data sets. Five real-world data sets are considered in this Chapter, as detailed in Table 7.1. Most industrial processes exhibit some kind of time-varying behavior, and so these data sets are crucial to evaluate the proposed methodologies. Input variable selection was performed to the cement kiln and polymerization reactor [Kadlec and Gabrys, 2011] data sets, as described in Subsection 4.4.1. No input variable selection was performed to the powder detergent production and thermal oxidizer data sets [Grbić *et al.*, 2013] (unlike Chapter 6). Additionally, no outlier detection and outlier treatment were applied in this Chapter. The aim is to analyze the algorithms' behaviors when erroneous data are available in the on-line learning process. Details and descriptions of all the data sets can be found in Section 2.7 and Subsection 4.4.1.

7.4.2 Evaluation Methodology

The same evaluation methodology that was used in Chapter 6 is applied in this Chapter. That is, consider a data set $\mathbf{D} = \{(\mathbf{x}_t, y_t)\}_{t=1}^{T}$ with T samples. The single model, the first model of the ensemble, or the pool of models (depending on the approach) is designed using the first m samples from $\mathbf{D}$ in the initialization phase of the learner. The remaining $(T - m)$ samples of $\mathbf{D}$ are arranged to form the on-line data to simulate an on-line scenario, where samples are given incrementally one-by-one. Each approach is evaluated using the mean and standard deviation of the MSE between the predicted outputs and the real outputs on the on-line data in 20 trials. In the experiments below, only the MSE on the on-line data is reported.

7.4.3 Approach Description and Setup

Tests are performed by comparing λ_{DFF}OS-ELM and OEOA to other state-of-the-art methods. The accuracies of the following single model learning algorithms are compared:

- *ELM*: standard ELM [Huang *et al.*, 2006], implemented as Algorithm 3.2.
- *OS-ELM*: sample-based OS-ELM [Liang *et al.*, 2006], implemented as Algorithm 3.9.
- λ_{DFF}*OS-ELM*: OS-ELM using a variable FF, implemented as Algorithm 7.1, where $\lambda_0 = 1$; $\gamma_0 = 10^{-3}$; $\upsilon_0 = 10^{-6}$, and $\rho = 0.99$ (the parameters are set as recommend by [Bobál *et al.*, 2005]).

For all the models, the hidden layer activation function $g(x)$ is *sigmoid*. The number of neurons in the hidden layer L is selected by *10-fold cross-validation*, as detailed in Subsection 6.3.2. Each OS-ELM or λ_{DFF}OS-ELM model is created by firstly training it with m samples (e.g. belonging to $\mathbf{D}^0$, for OS-ELM or λ_{DFF}OS-ELM, or belonging to $\mathbf{D}^t$ in OEOA) using the initialization phase of the learner; and then, whenever a new on-line sample is available, the model is retrained using the on-line learning phase of the learner.

Experiments are also conduced by comparing the effectiveness of the following on-line ensemble learning algorithms, which were implemented as described in Chapter 6: AddExp, DOER, EOS-ELM, Learn^{++}.NSE, OAUE, OB, OEOA, and OWE.

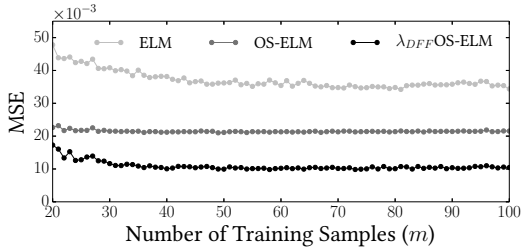
DOER and OWE were implemented using $\alpha = 0.04$ for the industrial data sets; and $\alpha = 0.10$ for the artificial data sets. OEOA was implemented according to Algorithm 7.2. Its parameters setting will be discussed in Subsection 7.4.5.

As Learn^{++}.NSE and OWE do not employ model retraining, their base model is the ELM. For the other ensembles, tests are performed using λ_{DFF}OS-ELM and OS-ELM as base models. For all the learning algorithms, the data are scaled as described in Chapter 6, Subsection 6.3.2. In all experiments, small values of m (training data set size) are considered, since in real-world setups of SS applications, it is often difficult to get sufficient data for modeling.

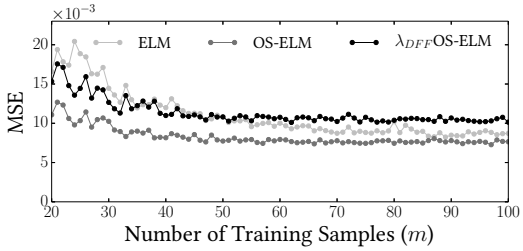
7.4.4 Comparison of Single Model Learning Algorithms

λ_{DFF}OS-ELM is evaluated and compared to ELM and OS-ELM. For each model, the results are averaged over 20 trials. It has been observed that, for medium size data sets, small windows (e.g. $m = 10$) lead to a significant increase in computational time, and in some cases no improvement in the accuracy of the system is observed. Thus, large windows were chosen for artificial data sets and real-world data sets of medium size. On the other hand, previous tests in Chapter 6 have shown that, in real-world data sets of small size, better performance is obtained when small windows are selected. Therefore, the experiments are conducted by varying m from 20 to 100 in steps of 1 for artificial data sets and real-world data sets of medium size; and varying m from 10 to 50 in steps of 1 for real-world data sets of small size (see Table 7.1). Figure 7.1 shows the MSE results of each algorithm as a function of m in all data sets. Table 7.2 shows the average and standard deviation of the MSE and processing time over all values of m for all the algorithms. The processing time considers the time spent on the training and on-line phases.

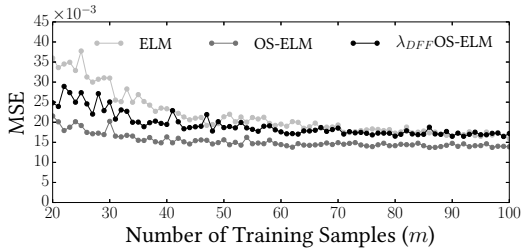
As observed in Figure 7.1, for the artificial data sets, the algorithms tend to decrease their errors as m increases. For the real-world data sets, in most cases, OS-ELM and λ_{DFF}OS-ELM methods keep their performances as m increases. For the Friedman data sets, it is observed that OS-ELM outperforms λ_{DFF}OS-ELM. This reveals that λ_{DFF}OS-ELM may not track scenarios with local and abrupt drifts (Friedman-LA), global recurring abrupt drifts (Friedman-GRA), and global non-recurring gradual drifts (Friedman-GnRG). This is because, λ_{DFF}OS-ELM forgets old information over time. λ_{DFF}OS-ELM has the best performance in the hyper-



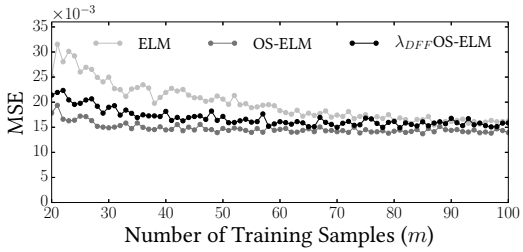
(a) Hyperplane data set.



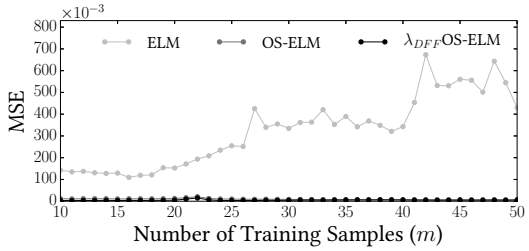
(b) Friedman-LA data set.



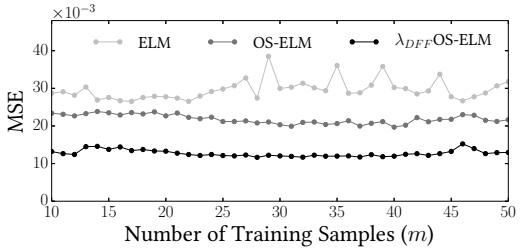
(c) Friedman-GRA data set.



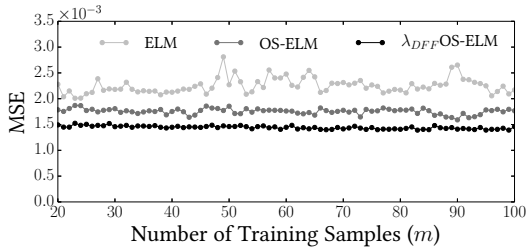
(d) Friedman-GnRG data set.



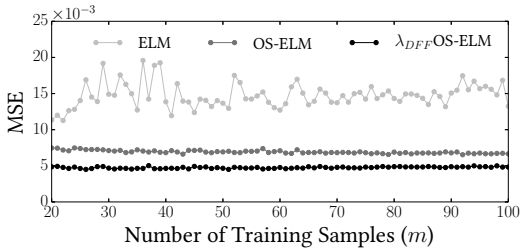
(e) Polymerization reactor data set



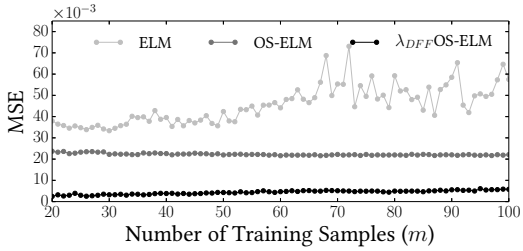
(f) Cement kiln data set.



(g) Thermal oxidizer data set.



(h) Powder detergent data set.



(i) Debutanizer column data set.

Figure 7.1: Performance of the single learning algorithms when the number of training samples m increases.

Table 7.2: Average and SD of the MSE[1] and processing time (seconds) of the single model learning algorithms by varying m.

Data set	Approach	MSE	Processing time
Hyperplane[2]	ELM	37.120 (2.741)	1.481 (0.259)
	OS-ELM	21.472 (0.323)	1.950 (0.241)
	λ_{DFF}OS-ELM	**10.867 (1.350)**	**1.408 (0.202)**
Friedman-LA[2]	ELM	11.032 (3.024)	1.477 (0.015)
	OS-ELM	**8.285 (1.151)**	2.007 (0.197)
	λ_{DFF}OS-ELM	11.359 (1.618)	**1.332 (0.161)**
Friedman-GRA[2]	ELM	21.497 (5.349)	1.487 (0.016)
	OS-ELM	**15.299 (1.654)**	1.518 (0.208)
	λ_{DFF}OS-ELM	19.253 (2.921)	**1.294 (0.197)**
Friedman-GnRG[2]	ELM	19.735 (3.738)	1.475 (0.013)
	OS-ELM	**14.775 (0.965)**	2.035 (0.176)
	λ_{DFF}OS-ELM	16.801 (1.683)	**1.411 (0.159)**
Polymerization reactor[3]	ELM	323.506 (160.943)	**0.710 (0.075)**
	OS-ELM	9.152 (2.983)	0.878 (0.100)
	λ_{DFF}OS-ELM	**4.891 (2.015)**	0.715 (0.112)
Cement kiln[3]	ELM	29.608 (2.666)	0.757 (0.092)
	OS-ELM	21.825 (1.192)	0.888 (0.106)
	λ_{DFF}OS-ELM	**12.736 (0.882)**	**0.666 (0.099)**
Powder detergent[2]	ELM	14.751 (1.699)	1.473 (0.035)
	OS-ELM	6.922 (0.218)	1.660 (0.314)
	λ_{DFF}OS-ELM	**4.756 (0.119)**	**1.457 (0.145)**
Thermal oxidizer[2]	ELM	2.250 (0.148)	1.788 (0.326)
	OS-ELM	1.755 (0.055)	2.071 (0.298)
	λ_{DFF}OS-ELM	**1.443 (0.031)**	**1.518 (0.154)**
Debutanizer column[2]	ELM	45.426 (8.936)	**1.629 (0.337)**
	OS-ELM	22.243 (0.493)	1.688 (0.322)
	λ_{DFF}OS-ELM	**4.398 (0.883)**	1.693 (0.175)

[1]The values have been multiplied by 10^3.
[2]The values of m are varied from 20 to 100 (in steps of 1).
[3]The values of m are varied from 10 to 50 (in steps of 1).

plane data set that contains non-recurring abrupt drift. For the real-world data sets, λ_{DFF}OS-ELM obtained the lowest MSE. From Table 7.2, it is noted that, in terms of processing time, λ_{DFF}OS-ELM outperforms OS-ELM and ELM in most cases.

7.4.5 Analysis of OEOA Parameters

The frequency of adding new models (which is related to α) may impact on the performances of OEOA, OWE, and DOER: small values of α generate large numbers of new models and increase the computational time; and large values of α may produce an inaccurate ensemble in changing environments, since new models are rarely added to the ensemble. Previous tests in Chapter 6 were conducted by varying α from 0.04 to 0.1 in steps of 0.02. It has been shown that α is related to the rate of concept change. That is, in data sets where concepts have large sizes (e.g. hyperplane data, where each concept has 500 samples), α should be set to a large value; while in data sets with concepts of small sizes (e.g. most industrial data sets due to the dynamics), α should be set to a small value. Therefore, for the OEOA, OWE, and DOER, in this Chapter, α is set to 0.10 for the artificial data sets, and 0.04 for the real-world data sets. The training size m (or window size) is also related to the rate of change of the data. In data sets which have a large rate of change, the system has better accuracy when m is small; while in data sets which have a small rate of change, the system has better accuracy when m is large. This characteristic can be observed in the experiments of the next Subsection.

Experiments were done to evaluate the effect of the minimum number of models (N_{min}) and maximum number of models (N_{max}) in the OEOA algorithm. The experiments use the cement kiln data set with $m = 10$ and $\alpha = 0.04$. The base models are λ_{DFF}OS-ELM and OS-ELM. The ELM model is not used since it does not have retraining. The first test aims to show the OEOA algorithm when $N_{min} = N_{max}$, namely, no OA is employed (see Figure 7.2(a)). The test reveals that when λ_{DFF}OS-ELM is the base model, the error is reduced as the number of models increases. When OS-ELM is the base model, the best performances are not obtained with the largest ensemble sizes. Figures 7.2(b) and 7.2(c) show the OEOA's performance when N_{min} varies and N_{max} is fixed. In Figure 7.2(b), N_{max} is set to 15, and in Figure 7.2(c), N_{max} is set to 30. When λ_{DFF}OS-ELM is the base model, as an overall tendency the experiment shows that if N_{min} increases, then the ensemble accuracy increases; while when OS-ELM is the base model, if N_{min} increases the ensemble error increases, after having obtained the best accuracy for some value of N_{min}. Thus, the adequate setting of N_{min} may depend on the base model. The test shows that λ_{DFF}OS-ELM outperforms OS-ELM as base model for the cement kiln

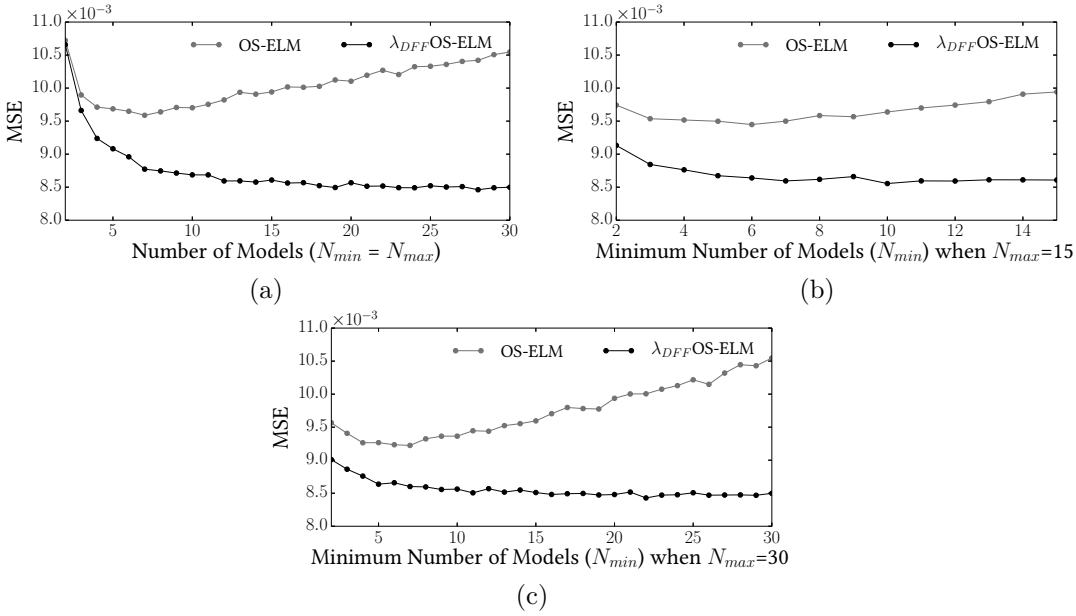


Figure 7.2: Experiments using different values of N_{min} and N_{max} in the OEOA algorithm for the cement kiln data set.

data set. This is because, λ_{DFF}OS-ELM models are able to forget old information and track better the dynamics of this data set.

7.4.6 Comparison of On-line Ensemble Learning Algorithms

In this Subsection, results of the on-line ensemble learning algorithms are compared. For the ensembles of fixed size (i.e. all ensembles except OEOA with the ordering strategy), the maximum number of models (N_{max}) is set to 15. This choice was considered the best suitable for all the ensembles, since in literature N_{max} usually varies between 15 and 30 [Elwell and Polikar, 2011; Kolter and Maloof, 2005], and the processing time of the experiments increases as N_{max} increases. Pilot tests indicate that, overall, the results and conclusions do not change significantly when N_{max} increases, for example to $N_{max} = 30$.

The OEOA is tested in two scenarios. In the first scenario $N_{min} = N_{max}$ and thus no OA is employed. In the second scenario $N_{min} \neq N_{max}$, and thus OA is tested. For each data set, the following pairs of values of (N_{min}, N_{max}) are tested in OEOA: $(5, 15)$, $(5, 30)$, $(10, 15)$, and $(10, 30)$; and for each data set and base model, and over all the tested values of m, the pair with the lowest average MSE error is

Table 7.3: Results of the on-line ensemble learning algorithms using the hyperplane data set and the Friedman-LA data set.

	Approach	Ensemble size	Average and SD of MSE for different values of m					Av. and SD on all values of m	
			$m = 20$	$m = 40$	$m = 60$	$m = 80$	$m = 100$	MSE	Proc. time (min.)
Hyperplane data set									
OS-ELM	AddExp	$N_{max} = 15$	7.23 (0.26)	6.77 (0.17)	7.16 (0.14)	7.69 (0.15)	8.10 (0.10)	7.39 (0.52)	9.84 (0.30)
	DOER	$N_{max} = 15$	5.87 (0.11)	5.57 (0.05)	5.80 (0.06)	6.18 (0.06)	6.50 (0.05)	5.98 (0.36)	7.72 (0.69)
	EOS-ELM	$N_{max} = 15$	19.20 (0.69)	19.61 (0.19)	19.79 (0.16)	20.03 (0.12)	20.14 (0.10)	19.75 (0.37)	0.52 (0.11)
	OAUE	$N_{max} = 15$	19.68 (0.20)	19.59 (0.12)	19.76 (0.27)	19.74 (0.17)	19.85 (0.19)	19.72 (0.10)	1.29 (0.35)
	OB	$N_{max} = 15$	19.01 (0.26)	19.55 (0.19)	19.77 (0.20)	20.04 (0.14)	20.16 (0.14)	19.71 (0.46)	0.44 (0.13)
	OEOA	$N_{min} = 15$, $N_{max} = 15$	**5.82 (0.07)**	**5.54 (0.07)**	5.80 (0.05)	6.16 (0.04)	**6.49 (0.06)**	**5.96 (0.37)**	7.08 (0.95)
	OEOA	$N_{min} = 10$, $N_{max} = 15$	5.87 (0.09)	**5.54 (0.05)**	**5.76 (0.07)**	**6.15 (0.08)**	**6.49 (0.08)**	**5.96 (0.37)**	7.25 (0.83)
λ_{DFF}OS-ELM	AddExp	$N_{max} = 15$	7.26 (0.23)	6.47 (0.14)	6.51 (0.16)	6.58 (0.12)	6.70 (0.14)	6.70 (0.32)	7.91 (0.69)
	DOER	$N_{max} = 15$	6.30 (0.09)	5.51 (0.06)	5.58 (0.07)	5.77 (0.05)	6.04 (0.06)	5.84 (0.33)	8.13 (0.75)
	EOS-ELM	$N_{max} = 15$	8.82 (1.77)	6.35 (0.28)	6.31 (0.14)	6.37 (0.10)	6.51 (0.12)	6.87 (1.09)	0.47 (0.03)
	OAUE	$N_{max} = 15$	10.21 (0.83)	8.49 (0.59)	8.23 (0.43)	8.51 (0.51)	8.89 (0.37)	8.87 (0.78)	1.20 (0.67)
	OB	$N_{max} = 15$	8.57 (0.73)	6.38 (0.15)	6.41 (0.12)	6.59 (0.16)	6.84 (0.12)	6.96 (0.92)	0.50 (0.01)
	OEOA	$N_{min} = 15$, $N_{max} = 15$	6.23 (0.10)	5.53 (0.05)	5.57 (0.08)	5.76 (0.05)	6.01 (0.07)	5.82 (0.30)	7.07 (0.86)
	OEOA	$N_{min} = 10$, $N_{max} = 30$	**6.01 (0.09)**	**5.44 (0.05)**	**5.50 (0.07)**	**5.67 (0.06)**	**5.88 (0.08)**	**5.70 (0.24)**	7.19 (0.76)
ELM	Learn^{++}.NSE	$N_{max} = 15$	23.63 (1.17)	12.92 (1.35)	11.47 (0.49)	10.33 (0.43)	11.97 (0.40)	14.06 (5.43)	0.54 (0.09)
	OWE	$N_{max} = 15$	**7.91 (0.19)**	**6.32 (0.08)**	**6.48 (0.08)**	**6.87 (0.08)**	**7.29 (0.10)**	**6.97 (0.64)**	
Friedman-LA									
OS-ELM	AddExp	$N_{max} = 15$	7.23 (0.16)	6.71 (0.12)	6.66 (0.09)	6.60 (0.08)	6.59 (0.07)	6.76 (0.27)	5.99 (0.47)
	DOER	$N_{max} = 15$	7.29 (0.11)	6.75 (0.06)	6.75 (0.05)	6.72 (0.05)	6.67 (0.06)	6.84 (0.26)	6.29 (0.36)
	EOS-ELM	$N_{max} = 15$	8.98 (1.87)	6.53 (0.14)	6.49 (0.13)	**6.45 (0.14)**	**6.42 (0.10)**	6.97 (1.12)	0.55 (0.01)
	OAUE	$N_{max} = 15$	**6.62 (0.14)**	**6.48 (0.09)**	6.53 (0.10)	6.64 (0.15)	6.56 (0.13)	6.56 (0.07)	1.60 (0.94)
	OB	$N_{max} = 15$	8.70 (0.86)	6.60 (0.15)	6.45 (0.08)	6.46 (0.08)	6.43 (0.07)	6.93 (0.99)	0.71 (0.03)
	OEOA	$N_{min} = 15$, $N_{max} = 15$	7.22 (0.12)	6.72 (0.06)	6.76 (0.05)	6.70 (0.05)	6.65 (0.04)	6.81 (0.23)	5.55 (0.36)
	OEOA	$N_{min} = 10$, $N_{max} = 30$	6.86 (0.09)	**6.48 (0.04)**	**6.48 (0.07)**	6.47 (0.05)	6.46 (0.05)	**6.55 (0.17)**	6.48 (0.43)
λ_{DFF}OS-ELM	AddExp	$N_{max} = 15$	8.84 (0.17)	7.99 (0.13)	7.70 (0.09)	7.49 (0.10)	7.37 (0.07)	7.88 (0.59)	7.13 (0.50)
	DOER	$N_{max} = 15$	8.79 (0.12)	7.51 (0.08)	7.18 (0.06)	**6.94 (0.05)**	**6.80 (0.06)**	7.45 (0.80)	5.95 (1.52)
	EOS-ELM	$N_{max} = 15$	11.34 (2.25)	8.12 (0.34)	7.73 (0.13)	7.64 (0.15)	7.55 (0.11)	8.48 (1.61)	0.45 (0.01)
	OAUE	$N_{max} = 15$	**6.64 (0.15)**	**6.62 (0.10)**	**6.77 (0.09)**	**6.94 (0.09)**	7.10 (0.16)	**6.81 (0.21)**	1.23 (0.70)
	OB	$N_{max} = 15$	9.48 (0.68)	7.45 (0.12)	7.27 (0.15)	7.20 (0.11)	7.11 (0.09)	7.70 (1.00)	0.48 (0.01)
	OEOA	$N_{min} = 15$, $N_{max} = 15$	8.73 (0.11)	7.53 (0.06)	7.17 (0.06)	6.96 (0.04)	**6.80 (0.05)**	7.44 (0.77)	5.99 (0.27)
	OEOA	$N_{min} = 10$, $N_{max} = 30$	8.40 (0.08)	7.48 (0.06)	7.15 (0.06)	6.95 (0.05)	6.81 (0.05)	7.36 (0.63)	6.65 (0.41)
ELM	Learn^{++}.NSE	$N_{max} = 15$	20.45 (0.93)	13.30 (0.53)	10.79 (0.39)	9.57 (0.37)	8.93 (0.25)	12.61 (4.69)	0.71 (0.28)
	OWE	$N_{max} = 15$	**9.26 (0.14)**	**7.41 (0.07)**	**7.10 (0.05)**	**6.89 (0.06)**	**6.76 (0.06)**	**7.49 (1.02)**	6.62 (0.78)

The MSE values have been multiplied by 10^3. Average and SD of MSE and processing time are obtained on 20 trials of the algorithms. The last two columns report the average and SD of MSE error and processing time of each approach on all window's sizes, respectively.

Table 7.4: Results of the on-line ensemble learning algorithms using the Friedman-GRA data set and Friedman-GnRG data set.

	Approach	Ensemble size	Average and SD of MSE for different values of m					Av. and SD on all values of m	
			$m = 20$	$m = 40$	$m = 60$	$m = 80$	$m = 100$	MSE	Proc. time (min.)
	Friedman-GRA data set								
OS-ELM	AddExp	$N_{max} = 15$	12.40 (0.29)	11.55 (0.16)	11.51 (0.13)	11.46 (0.12)	11.46 (0.12)	11.67 (0.41)	8.78 (1.29)
OS-ELM	DOER	$N_{max} = 15$	12.63 (0.16)	11.43 (0.10)	11.30 (0.12)	11.17 (0.08)	11.08 (0.06)	11.52 (0.63)	9.40 (0.97)
OS-ELM	EOS-ELM	$N_{max} = 15$	16.30 (3.18)	12.42 (0.64)	11.92 (0.11)	11.84 (0.13)	11.85 (0.11)	12.87 (1.94)	0.55 (0.01)
OS-ELM	OAUE	$N_{max} = 15$	12.45 (0.22)	12.29 (0.28)	12.16 (0.14)	12.33 (0.30)	12.28 (0.29)	12.30 (0.10)	1.66 (0.97)
OS-ELM	OB	$N_{max} = 15$	14.83 (1.48)	12.26 (0.31)	11.99 (0.11)	11.94 (0.11)	11.88 (0.10)	12.58 (1.27)	0.74 (0.13)
OS-ELM	**OEOA**	$N_{min} = 15$, $N_{max} = 15$	12.60 (0.16)	11.40 (0.10)	11.33 (0.08)	11.14 (0.09)	11.04 (0.07)	11.50 (0.63)	8.92 (1.11)
OS-ELM	**OEOA**	$N_{min} = 10$, $N_{max} = 30$	**11.96 (0.13)**	**11.16 (0.11)**	**11.02 (0.05)**	**10.97 (0.09)**	**10.89 (0.08)**	**11.20 (0.43)**	9.63 (0.70)
λ_{DFF}OS-ELM	AddExp	$N_{max} = 15$	14.24 (0.38)	12.60 (0.19)	12.15 (0.13)	11.87 (0.18)	11.63 (0.12)	12.50 (1.04)	8.63 (0.38)
λ_{DFF}OS-ELM	DOER	$N_{max} = 15$	14.47 (0.23)	12.21 (0.13)	11.61 (0.11)	11.27 (0.08)	11.09 (0.10)	12.13 (1.38)	7.74 (0.30)
λ_{DFF}OS-ELM	EOS-ELM	$N_{max} = 15$	16.03 (1.52)	13.69 (1.21)	12.46 (0.42)	12.50 (0.37)	12.16 (0.26)	13.37 (1.60)	0.45 (0.01)
λ_{DFF}OS-ELM	OAUE	$N_{max} = 15$	**12.23 (0.31)**	**11.90 (0.42)**	11.88 (0.19)	11.90 (0.25)	11.97 (0.41)	11.98 (0.15)	1.25 (0.79)
λ_{DFF}OS-ELM	OB	$N_{max} = 15$	15.09 (0.82)	12.20 (0.50)	11.74 (0.21)	11.53 (0.25)	11.44 (0.17)	12.40 (1.53)	0.48 (0.01)
λ_{DFF}OS-ELM	**OEOA**	$N_{min} = 15$, $N_{max} = 15$	14.51 (0.16)	12.16 (0.10)	11.62 (0.10)	11.27 (0.10)	**11.04 (0.08)**	12.12 (1.40)	8.81 (1.19)
λ_{DFF}OS-ELM	**OEOA**	$N_{min} = 10$, $N_{max} = 30$	13.66 (0.15)	12.07 (0.10)	**11.57 (0.10)**	**11.17 (0.08)**	**11.04 (0.09)**	**11.90 (1.06)**	8.87 (0.40)
ELM	Learn^{++}.NSE	$N_{max} = 15$	37.56 (10.02)	21.34 (0.93)	17.81 (0.64)	16.06 (0.55)	16.03 (0.69)	21.76 (9.09)	0.68 (0.29)
ELM	OWE	$N_{max} = 15$	**14.54 (0.33)**	**12.10 (0.17)**	**11.59 (0.13)**	**11.29 (0.10)**	**11.21 (0.08)**	**12.15 (1.38)**	7.89 (2.19)
	Friedman-GnRG data set								
OS-ELM	AddExp	$N_{max} = 15$	11.63 (0.19)	10.87 (0.12)	10.70 (0.09)	10.67 (0.11)	10.62 (0.07)	10.90 (0.42)	7.54 (1.25)
OS-ELM	DOER	$N_{max} = 15$	11.54 (0.16)	10.51 (0.09)	10.27 (0.06)	10.15 (0.06)	9.98 (0.07)	10.49 (0.61)	8.90 (0.85)
OS-ELM	EOS-ELM	$N_{max} = 15$	13.73 (1.53)	12.40 (0.44)	12.27 (0.13)	12.27 (0.15)	12.20 (0.10)	12.57 (0.65)	0.56 (0.00)
OS-ELM	OAUE	$N_{max} = 15$	12.54 (0.14)	12.45 (0.17)	12.53 (0.19)	12.66 (0.26)	12.71 (0.45)	12.58 (0.11)	1.37 (1.10)
OS-ELM	OB	$N_{max} = 15$	13.70 (0.52)	12.30 (0.19)	12.22 (0.10)	12.25 (0.10)	12.27 (0.12)	12.55 (0.65)	0.71 (0.05)
OS-ELM	**OEOA**	$N_{min} = 15$, $N_{max} = 15$	11.51 (0.17)	10.53 (0.09)	10.26 (0.11)	10.14 (0.11)	9.98 (0.06)	10.48 (0.61)	8.15 (1.16)
OS-ELM	**OEOA**	$N_{min} = 10$, $N_{max} = 30$	**10.99 (0.13)**	**10.23 (0.06)**	**10.05 (0.06)**	**9.97 (0.06)**	**9.84 (0.06)**	**10.22 (0.45)**	9.04 (0.57)
λ_{DFF}OS-ELM	AddExp	$N_{max} = 15$	13.02 (0.21)	11.60 (0.18)	11.12 (0.14)	10.81 (0.12)	10.59 (0.11)	11.43 (0.97)	8.39 (0.69)
λ_{DFF}OS-ELM	DOER	$N_{max} = 15$	13.25 (0.17)	11.20 (0.07)	10.56 (0.10)	10.28 (0.08)	**10.03 (0.07)**	11.06 (1.30)	8.31 (0.53)
λ_{DFF}OS-ELM	EOS-ELM	$N_{max} = 15$	14.63 (2.59)	12.07 (0.74)	11.65 (0.35)	11.40 (0.43)	11.09 (0.25)	12.17 (1.42)	0.46 (0.01)
λ_{DFF}OS-ELM	OAUE	$N_{max} = 15$	**11.97 (0.22)**	11.60 (0.21)	11.49 (0.30)	11.30 (0.31)	11.21 (0.30)	11.52 (0.30)	1.22 (0.53)
λ_{DFF}OS-ELM	OB	$N_{max} = 15$	13.72 (1.00)	11.06 (0.26)	10.77 (0.18)	10.59 (0.10)	10.52 (0.17)	11.33 (1.35)	0.49 (0.01)
λ_{DFF}OS-ELM	**OEOA**	$N_{min} = 15$, $N_{max} = 15$	13.15 (0.13)	11.21 (0.10)	10.58 (0.09)	**10.26 (0.07)**	10.04 (0.07)	11.05 (1.26)	8.44 (1.36)
λ_{DFF}OS-ELM	**OEOA**	$N_{min} = 10$, $N_{max} = 30$	12.46 (0.14)	**11.04 (0.09)**	**10.50 (0.09)**	**10.26 (0.07)**	10.05 (0.05)	**10.86 (0.97)**	8.55 (0.60)
ELM	Learn^{++}.NSE	$N_{max} = 15$	29.73 (1.03)	19.42 (0.84)	15.50 (0.57)	14.16 (0.44)	13.34 (0.58)	18.43 (6.73)	0.68 (0.31)
ELM	OWE	$N_{max} = 15$	**14.44 (0.26)**	**11.21 (0.14)**	**10.52 (0.07)**	**10.30 (0.09)**	**10.06 (0.09)**	**11.31 (1.80)**	9.80 (1.12)

The MSE values have been multiplied by 10^3. Average and SD of MSE and processing time are obtained on 20 trials of the algorithms. The last two columns report the average and SD of MSE error and processing time of each approach on all window's sizes, respectively.

Table 7.5: Results of the on-line ensemble learning algorithms using the polymerization reactor data set and the cement kiln data set.

	Approach	Ensemble size	Average and SD of MSE for different values of m					Av. and SD on all values of m	
			$m = 10$	$m = 20$	$m = 30$	$m = 40$	$m = 50$	MSE	Proc. time (min.)
	Polymerization reactor data set								
OS-ELM	AddExp	$N_{max} = 15$	2.80 (0.09)	2.73 (0.14)	2.67 (0.20)	2.52 (0.17)	2.75 (0.21)	2.70 (0.11)	1.28 (0.35)
OS-ELM	DOER	$N_{max} = 15$	0.47 (0.04)	0.66 (0.06)	0.80 (0.07)	0.99 (0.08)	1.22 (0.09)	0.83 (0.29)	1.41 (0.51)
OS-ELM	EOS-ELM	$N_{max} = 15$	8.09 (0.42)	14.18 (3.07)	6.01 (0.25)	4.54 (0.26)	4.12 (0.20)	7.39 (4.10)	0.26 (0.04)
OS-ELM	OAUE	$N_{max} = 15$	2.76 (0.23)	2.93 (0.19)	3.59 (0.34)	3.74 (0.42)	4.06 (0.37)	3.42 (0.55)	0.17 (0.04)
OS-ELM	OB	$N_{max} = 15$	8.32 (0.57)	10.73 (1.47)	5.99 (0.34)	4.49 (0.25)	4.27 (0.24)	6.76 (2.75)	0.28 (0.03)
OS-ELM	**OEOA**	$N_{min} = 15$, $N_{max} = 15$	0.50 (0.02)	0.68 (0.05)	0.81 (0.07)	1.10 (0.12)	1.37 (0.08)	0.89 (0.35)	1.36 (0.45)
OS-ELM	**OEOA**	$N_{min} = 5$, $N_{max} = 30$	**0.40 (0.06)**	**0.52 (0.06)**	**0.69 (0.09)**	**0.87 (0.10)**	**1.12 (0.09)**	**0.72 (0.28)**	1.38 (0.36)
λ_{DFF}OS-ELM	AddExp	$N_{max} = 15$	0.97 (0.12)	1.33 (0.19)	1.47 (0.16)	1.58 (0.25)	1.62 (0.15)	1.39 (0.26)	0.31 (0.06)
λ_{DFF}OS-ELM	DOER	$N_{max} = 15$	**0.29 (0.02)**	0.46 (0.04)	0.54 (0.05)	0.64 (0.09)	0.88 (0.05)	0.56 (0.22)	0.88 (0.29)
λ_{DFF}OS-ELM	EOS-ELM	$N_{max} = 15$	0.68 (0.13)	2.94 (2.42)	1.50 (0.38)	2.23 (0.26)	2.25 (0.21)	1.92 (0.86)	0.22 (0.04)
λ_{DFF}OS-ELM	OAUE	$N_{max} = 15$	0.81 (0.13)	1.45 (0.40)	1.76 (0.47)	2.31 (0.34)	2.40 (0.51)	1.75 (0.65)	0.40 (0.14)
λ_{DFF}OS-ELM	OB	$N_{max} = 15$	1.03 (0.16)	2.13 (0.86)	2.13 (0.27)	2.75 (0.26)	2.61 (0.17)	2.13 (0.68)	0.21 (0.03)
λ_{DFF}OS-ELM	**OEOA**	$N_{min} = 15$, $N_{max} = 15$	**0.29 (0.03)**	0.48 (0.03)	0.52 (0.05)	0.62 (0.05)	0.87 (0.06)	0.56 (0.21)	0.94 (0.32)
λ_{DFF}OS-ELM	**OEOA**	$N_{min} = 5$, $N_{max} = 30$	0.30 (0.03)	**0.43 (0.05)**	**0.51 (0.04)**	**0.60 (0.07)**	**0.82 (0.09)**	**0.53 (0.19)**	1.25 (0.30)
ELM	Learn^{++}.NSE	$N_{max} = 15$	2.89 (0.42)	5.74 (0.87)	6.74 (1.30)	8.05 (2.69)	16.83 (2.99)	8.05 (5.26)	0.26 (0.09)
ELM	OWE	$N_{max} = 15$	**0.55 (0.06)**	**0.94 (0.10)**	**1.33 (0.11)**	**1.90 (0.12)**	**2.42 (0.27)**	**1.43 (0.75)**	1.85 (0.81)
	Cement kiln data set								
OS-ELM	AddExp	$N_{max} = 15$	12.13 (0.26)	12.22 (0.23)	12.02 (0.35)	11.97 (0.24)	11.86 (0.34)	12.04 (0.14)	3.39 (1.10)
OS-ELM	DOER	$N_{max} = 15$	10.03 (0.19)	10.25 (0.21)	10.20 (0.23)	10.26 (0.20)	10.13 (0.18)	10.17 (0.10)	3.17 (1.20)
OS-ELM	EOS-ELM	$N_{max} = 15$	22.40 (1.17)	22.05 (2.00)	17.33 (2.31)	18.59 (2.64)	17.88 (1.98)	19.65 (2.39)	0.28 (0.05)
OS-ELM	OAUE	$N_{max} = 15$	12.07 (0.32)	12.67 (0.32)	12.97 (0.40)	13.47 (0.64)	14.99 (1.15)	13.23 (1.11)	0.49 (0.10)
OS-ELM	OB	$N_{max} = 15$	20.08 (0.86)	18.70 (1.09)	16.52 (1.12)	16.46 (0.96)	16.13 (0.91)	17.58 (1.73)	0.31 (0.05)
OS-ELM	**OEOA**	$N_{min} = 15$, $N_{max} = 15$	9.96 (0.18)	10.22 (0.14)	10.08 (0.23)	10.21 (0.20)	10.15 (0.23)	10.12 (0.11)	2.85 (1.10)
OS-ELM	**OEOA**	$N_{min} = 5$, $N_{max} = 30$	**9.26 (0.17)**	**9.59 (0.17)**	**9.56 (0.29)**	**9.73 (0.25)**	**9.69 (0.22)**	**9.56 (0.18)**	3.18 (1.16)
λ_{DFF}OS-ELM	AddExp	$N_{max} = 15$	9.13 (0.24)	9.44 (0.25)	9.35 (0.23)	9.67 (0.32)	9.78 (0.31)	9.47 (0.26)	2.47 (0.72)
λ_{DFF}OS-ELM	DOER	$N_{max} = 15$	8.61 (0.11)	8.86 (0.11)	8.78 (0.15)	8.97 (0.16)	8.97 (0.20)	8.84 (0.15)	2.68 (0.97)
λ_{DFF}OS-ELM	EOS-ELM	$N_{max} = 15$	10.03 (0.63)	10.20 (1.33)	9.15 (1.05)	9.49 (0.91)	9.46 (1.14)	9.67 (0.44)	0.24 (0.05)
λ_{DFF}OS-ELM	OAUE	$N_{max} = 15$	9.61 (0.30)	9.55 (0.28)	9.45 (0.35)	9.70 (0.56)	10.17 (0.81)	9.70 (0.28)	0.57 (0.20)
λ_{DFF}OS-ELM	OB	$N_{max} = 15$	9.44 (0.43)	9.36 (0.34)	8.94 (0.21)	9.19 (0.28)	9.16 (0.20)	9.22 (0.19)	0.23 (0.04)
λ_{DFF}OS-ELM	**OEOA**	$N_{min} = 15$, $N_{max} = 15$	8.60 (0.09)	8.88 (0.13)	8.77 (0.15)	8.93 (0.15)	8.92 (0.19)	8.82 (0.14)	2.70 (1.05)
λ_{DFF}OS-ELM	**OEOA**	$N_{min} = 10$, $N_{max} = 30$	**8.57 (0.09)**	**8.71 (0.13)**	**8.67 (0.08)**	**8.81 (0.08)**	**8.79 (0.11)**	**8.71 (0.10)**	4.40 (2.02)
ELM	Learn^{++}.NSE	$N_{max} = 15$	23.12 (3.03)	24.47 (1.71)	28.51 (2.03)	26.75 (3.51)	33.44 (5.59)	27.26 (4.03)	0.31 (0.07)
ELM	OWE	$N_{max} = 15$	**11.23 (0.41)**	**12.67 (0.69)**	**13.04 (0.67)**	**13.55 (0.65)**	**13.16 (0.49)**	**12.73 (0.89)**	3.40 (1.63)

The MSE values have been multiplied by 10^3. Average and SD of MSE and processing time are obtained on 20 trials of the algorithms. The last two columns report the average and SD of MSE error and processing time of each approach on all window's sizes, respectively.

Table 7.6: Results of the on-line ensemble learning algorithms using the powder detergent data set and the thermal oxidizer data set.

	Approach	Ensemble size	Average and SD of MSE for different values of m					Av. and SD on all values of m	
			$m = 20$	$m = 40$	$m = 60$	$m = 80$	$m = 100$	MSE	Proc. time (min.)
	Powder detergent data set								
OS-ELM	AddExp	$N_{max} = 15$	5.13 (0.09)	5.11 (0.10)	5.21 (0.09)	5.29 (0.11)	5.37 (0.06)	5.22 (0.11)	5.14 (1.11)
OS-ELM	DOER	$N_{max} = 15$	4.53 (0.06)	4.76 (0.09)	4.97 (0.08)	5.09 (0.08)	5.19 (0.06)	4.91 (0.27)	7.47 (0.22)
OS-ELM	EOS-ELM	$N_{max} = 15$	6.92 (0.70)	6.04 (0.47)	6.14 (0.37)	5.94 (0.17)	5.83 (0.22)	6.17 (0.43)	0.54 (0.01)
OS-ELM	OAUE	$N_{max} = 15$	5.09 (0.07)	5.36 (0.11)	5.59 (0.17)	5.72 (0.16)	5.79 (0.20)	5.51 (0.29)	1.36 (0.79)
OS-ELM	OB	$N_{max} = 15$	6.43 (0.26)	5.87 (0.15)	5.80 (0.15)	5.76 (0.12)	5.72 (0.08)	5.92 (0.29)	0.68 (0.05)
OS-ELM	**OEOA**	$N_{min} = 15$, $N_{max} = 15$	4.51 (0.06)	4.74 (0.08)	4.95 (0.10)	5.07 (0.09)	5.17 (0.11)	4.89 (0.27)	6.54 (0.42)
OS-ELM	**OEOA**	$N_{min} = 5$, $N_{max} = 30$	**4.31 (0.09)**	**4.51 (0.11)**	**4.69 (0.11)**	**4.84 (0.13)**	**4.96 (0.11)**	**4.66 (0.26)**	7.66 (0.52)
λ_{DFF}OS-ELM	AddExp	$N_{max} = 15$	4.07 (0.15)	4.05 (0.10)	4.15 (0.12)	4.20 (0.08)	4.26 (0.10)	4.15 (0.09)	4.48 (0.68)
λ_{DFF}OS-ELM	DOER	$N_{max} = 15$	4.00 (0.04)	4.17 (0.07)	4.18 (0.08)	4.27 (0.07)	4.40 (0.08)	4.20 (0.15)	7.02 (2.22)
λ_{DFF}OS-ELM	EOS-ELM	$N_{max} = 15$	3.86 (0.14)	**3.78 (0.08)**	**3.75 (0.05)**	3.86 (0.07)	**3.84 (0.06)**	**3.82 (0.05)**	0.45 (0.01)
λ_{DFF}OS-ELM	OAUE	$N_{max} = 15$	3.89 (0.08)	3.91 (0.10)	3.96 (0.13)	4.20 (0.19)	4.21 (0.17)	4.03 (0.16)	1.45 (1.00)
λ_{DFF}OS-ELM	OB	$N_{max} = 15$	**3.84 (0.08)**	3.79 (0.06)	3.80 (0.09)	**3.84 (0.08)**	3.89 (0.06)	3.83 (0.04)	0.46 (0.00)
λ_{DFF}OS-ELM	**OEOA**	$N_{min} = 15$, $N_{max} = 15$	3.99 (0.07)	4.11 (0.10)	4.19 (0.11)	4.26 (0.09)	4.35 (0.10)	4.18 (0.14)	6.23 (0.27)
λ_{DFF}OS-ELM	**OEOA**	$N_{min} = 10$, $N_{max} = 30$	3.87 (0.05)	3.87 (0.06)	3.90 (0.06)	3.99 (0.06)	4.10 (0.08)	3.95 (0.10)	6.54 (0.56)
ELM	Learn^{++}.NSE	$N_{max} = 15$	9.94 (1.14)	11.51 (1.66)	12.01 (1.68)	11.77 (2.47)	11.07 (1.69)	11.26 (0.82)	0.64 (0.28)
ELM	OWE	$N_{max} = 15$	**5.88 (0.17)**	**6.28 (0.21)**	**6.67 (0.20)**	**7.06 (0.20)**	**7.12 (0.18)**	**6.60 (0.53)**	7.36 (1.57)
	Thermal oxidizer data set								
OS-ELM	AddExp	$N_{max} = 15$	1.45 (0.04)	1.40 (0.04)	1.38 (0.03)	1.38 (0.03)	1.39 (0.03)	1.40 (0.03)	1.53 (0.26)
OS-ELM	DOER	$N_{max} = 15$	**1.12 (0.01)**	1.13 (0.01)	1.12 (0.01)	1.13 (0.01)	1.13 (0.00)	1.13 (0.01)	7.41 (2.24)
OS-ELM	EOS-ELM	$N_{max} = 15$	1.66 (0.10)	1.64 (0.13)	1.79 (0.08)	1.78 (0.13)	1.70 (0.12)	1.72 (0.07)	0.66 (0.07)
OS-ELM	OAUE	$N_{max} = 15$	1.19 (0.01)	1.26 (0.01)	1.29 (0.02)	1.34 (0.03)	1.39 (0.03)	1.30 (0.07)	1.64 (0.97)
OS-ELM	OB	$N_{max} = 15$	1.64 (0.03)	1.57 (0.05)	1.65 (0.04)	1.70 (0.05)	1.66 (0.04)	1.64 (0.04)	0.79 (0.07)
OS-ELM	**OEOA**	$N_{min} = 15$, $N_{max} = 15$	**1.12 (0.01)**	1.13 (0.01)	1.12 (0.01)	1.13 (0.00)	1.14 (0.00)	1.13 (0.01)	7.41 (2.41)
OS-ELM	**OEOA**	$N_{min} = 5$, $N_{max} = 15$	**1.12 (0.01)**	**1.12 (0.01)**	**1.11 (0.01)**	**1.11 (0.01)**	**1.12 (0.01)**	**1.12 (0.01)**	6.54 (2.17)
λ_{DFF}OS-ELM	AddExp	$N_{max} = 15$	1.32 (0.04)	1.28 (0.05)	1.27 (0.05)	1.28 (0.04)	1.26 (0.05)	1.28 (0.02)	0.80 (0.16)
λ_{DFF}OS-ELM	DOER	$N_{max} = 15$	**1.17 (0.02)**	**1.19 (0.02)**	1.18 (0.02)	**1.17 (0.02)**	1.18 (0.02)	**1.18 (0.01)**	6.29 (1.80)
λ_{DFF}OS-ELM	EOS-ELM	$N_{max} = 15$	1.26 (0.04)	1.27 (0.04)	1.24 (0.05)	1.23 (0.05)	1.27 (0.09)	1.25 (0.02)	0.54 (0.08)
λ_{DFF}OS-ELM	OAUE	$N_{max} = 15$	**1.17 (0.03)**	1.22 (0.04)	1.24 (0.07)	1.27 (0.07)	1.27 (0.07)	1.23 (0.04)	1.64 (0.86)
λ_{DFF}OS-ELM	OB	$N_{max} = 15$	1.22 (0.04)	1.20 (0.03)	1.19 (0.03)	1.19 (0.04)	1.20 (0.03)	1.20 (0.01)	0.57 (0.05)
λ_{DFF}OS-ELM	**OEOA**	$N_{min} = 15$, $N_{max} = 15$	1.18 (0.03)	**1.19 (0.01)**	**1.17 (0.02)**	1.18 (0.02)	**1.17 (0.02)**	**1.18 (0.01)**	6.73 (2.74)
λ_{DFF}OS-ELM	**OEOA**	$N_{min} = 10$, $N_{max} = 15$	1.18 (0.02)	1.20 (0.02)	1.18 (0.02)	1.18 (0.02)	**1.17 (0.01)**	**1.18 (0.01)**	8.48 (3.09)
ELM	Learn^{++}.NSE	$N_{max} = 15$	1.36 (0.10)	1.50 (0.13)	1.59 (0.09)	1.69 (0.14)	1.76 (0.15)	1.58 (0.16)	0.83 (0.26)
ELM	OWE	$N_{max} = 15$	**1.12 (0.01)**	**1.16 (0.01)**	**1.17 (0.01)**	**1.19 (0.01)**	**1.21 (0.01)**	**1.17 (0.03)**	6.88 (2.65)

The MSE values have been multiplied by 10^3. Average and SD of MSE and processing time are obtained on 20 trials of the algorithms. The last two columns report the average and SD of MSE error and processing time of each approach on all window's sizes, respectively.

Table 7.7: Results of the on-line ensemble learning algorithms using the debutanizer column data set.

	Approach	Ensemble size	Average and SD of MSE for different values of m					Av. and SD on all values of m	
			$m = 20$	$m = 40$	$m = 60$	$m = 80$	$m = 100$	MSE	Proc. time (min.)
	Debutanizer column data set								
OS-ELM	AddExp	$N_{max} = 15$	7.28 (0.19)	7.94 (0.20)	8.87 (0.15)	9.59 (0.15)	10.14 (0.18)	8.77 (1.17)	8.62 (1.10)
	DOER	$N_{max} = 15$	2.61 (0.21)	3.62 (0.12)	5.72 (0.18)	6.73 (0.20)	7.73 (0.22)	5.28 (2.13)	14.02 (1.82)
	EOS-ELM	$N_{max} = 15$	23.22 (0.82)	21.41 (0.54)	20.59 (0.48)	20.86 (0.45)	20.45 (0.25)	21.30 (1.13)	0.58 (0.03)
	OAUE	$N_{max} = 15$	15.22 (0.30)	17.86 (0.31)	18.72 (0.29)	19.04 (0.26)	19.30 (0.23)	18.03 (1.66)	1.81 (1.06)
	OB	$N_{max} = 15$	22.61 (0.45)	21.25 (0.19)	20.58 (0.26)	20.56 (0.25)	20.57 (0.25)	21.11 (0.88)	0.74 (0.03)
	OEOA	$N_{min} = 15$, $N_{max} = 15$	2.60 (0.20)	3.60 (0.11)	5.68 (0.17)	6.73 (0.16)	7.77 (0.17)	5.28 (2.15)	12.59 (1.63)
	OEOA	$N_{min} = 5$, $N_{max} = 30$	**2.17 (0.20)**	**3.07 (0.15)**	**4.92 (0.17)**	**5.87 (0.19)**	**6.90 (0.27)**	**4.59 (1.95)**	13.12 (3.95)
λ_{DFF}OS-ELM	AddExp	$N_{max} = 15$	1.39 (0.24)	1.94 (0.26)	2.66 (0.35)	**2.77 (0.31)**	3.26 (0.36)	2.40 (0.73)	3.68 (0.87)
	DOER	$N_{max} = 15$	1.55 (0.19)	2.14 (0.10)	3.18 (0.14)	3.74 (0.15)	4.31 (0.17)	2.99 (1.14)	16.56 (1.37)
	EOS-ELM	$N_{max} = 15$	**1.11 (0.16)**	1.89 (0.34)	2.82 (0.21)	2.84 (0.30)	**3.05 (0.16)**	2.34 (0.82)	0.50 (0.02)
	OAUE	$N_{max} = 15$	2.19 (0.44)	2.75 (0.31)	3.31 (0.30)	3.32 (0.26)	3.47 (0.30)	3.01 (0.53)	1.68 (0.98)
	OB	$N_{max} = 15$	2.03 (0.17)	3.09 (0.23)	3.96 (0.19)	3.95 (0.23)	4.30 (0.16)	3.47 (0.92)	0.53 (0.02)
	OEOA	$N_{min} = 15$, $N_{max} = 15$	1.55 (0.13)	2.15 (0.11)	3.20 (0.11)	3.72 (0.14)	4.29 (0.22)	2.98 (1.12)	8.75 (1.07)
	OEOA	$N_{min} = 10$, $N_{max} = 30$	1.19 (0.09)	**1.77 (0.12)**	**2.42 (0.23)**	2.84 (0.14)	3.07 (0.14)	**2.26 (0.78)**	12.13 (2.13)
ELM	Learn^{++}.NSE	$N_{max} = 15$	26.49 (4.69)	42.15 (4.83)	51.06 (8.00)	72.20 (15.63)	98.71 (31.13)	58.12 (28.07)	0.77 (0.36)
	OWE	$N_{max} = 15$	**9.19 (0.61)**	**19.58 (1.18)**	**31.18 (1.52)**	**29.67 (1.54)**	**32.76 (1.80)**	**24.48 (9.98)**	16.79 (3.09)

The MSE values have been multiplied by 10^3. Average and SD of MSE and processing time are obtained on 20 trials of the algorithms. The last two columns report the average and SD of MSE error and processing time of each approach on all window's sizes, respectively.

presented in the results of the experiments. The simulation results are presented in Tables 7.3, 7.4, 7.5, 7.6, and 7.7. Several values of m were tested, as described in these tables. For each problem, the simulation was conducted 20 times. The average and standard deviation values of the MSE and processing time (in minutes) are presented. The processing time considers the time spent on both the training and on-line phases.

As described in Subsection 7.4.4, λ_{DFF}OS-ELM has poor performance when compared to OS-ELM in the Friedman data sets. This can be observed in Table 7.3 where most ensembles have better performance when OS-ELM is the base model. The exceptions are for the OAUE in the Friedman-GRA and Friedman-GnRG data sets; and for the OB in the Friedman-GnRG data set; where the error is reduced when λ_{DFF}OS-ELM is the base model. For the other data sets, in most cases, the ensembles' errors reduce significantly when λ_{DFF}OS-ELM is the base model. The reduction can be observed mainly in the debutanizer column data set. For example, for the EOS-ELM with OS-ELM as base model, the average of the MSE over all tested values of m is 21.3×10^{-3}; and with λ_{DFF}OS-ELM as base model, the average of the MSE is reduced to 2.34×10^{-3}; and for the OB with OS-ELM as base model, the average of MSE on all values of m is 21.11×10^{-3}; and with λ_{DFF}OS-ELM as base model, the MSE is reduced to 3.47×10^{-3}. OB and EOS-ELM are ensembles with few adaptive mechanisms, since only retraining of models is employed, and no combination weights' adaptation and no dynamic selection of models are employed. However, they significantly improve their performances when λ_{DFF}OS-ELM is the base model. Additionally, they have low processing time when compared to the other approaches.

OWE outperforms Learn^{++}.NSE in all cases. This is because, Learn^{++}.NSE is adapted on a batch basis; while OWE is adapted on a sample basis. Therefore, OWE adapts faster to changes. The best performances of OWE are achieved mainly in the artificial data sets and the thermal oxidizer data set. In the thermal oxidizer data set, the average of the MSE on all values of m for the OWE is 1.17×10^{-3}; while for OEOA with ordering ($N_{min} \neq N_{max}$) and λ_{DFF}OS-ELM as base model, the average MSE is 1.18×10^{-3}. In contrast to OWE and Learn^{++}.NSE, OAUE retrains all the models at each new sample. However, OAUE includes new models into the ensemble at a low frequency when compared to AddExp, OEOA, OWE, and DOER. AddExp employs the same adaptive ensemble mechanisms as the OEOA. However, AddExp

has an error larger than the error on OEOA in all cases. In the AddExp, new models take more time to have their combination weights significantly increased. In scenarios that require faster adaptation to the new concepts, this method for assigning combination weights may fail. In contrast to AddExp, OEOA assigns large combination weights to the new and accurate models if they have low errors on the newest samples.

In most cases, OEOA with OA significantly reduces the ensemble error when compared to OEOA without OA - for example, in the debutanizer column and cement kiln data sets. In other cases, OEOA with OA has similar performance when compared to OEOA without OA - for example, in the hyperplane data set with OS-ELM as the base model. In most cases, it can also be observed that OEOA with OA has better performance when N_{min} and N_{max} are large; and the sets with the best performances are $(N_{min}, N_{max}) = (10, 30)$ and $(N_{min}, N_{max}) = (5, 30)$. DOER, and OEOA without OA, have similar performances, since they have similar methods for the assignment of combination weights and for the selection of models. However, DOER starts the system by creating an ensemble with one model; while OEOA starts the system by creating an initial pool of N_{max} models. The results indicate that OEOA without OA slightly outperforms DOER in most cases. However, in the polymerization reactor data set, with OS-ELM as base model, the average of the MSE over all values of m for the DOER is 0.83×10^{-3}; while for the OEOA without OA it is 0.89×10^{-3}.

The results reveal that OEOA with OA is more time consuming when compared to the OEOA without OA. This is because the OA strategy requires more time to compute the best subset size. Additionally, the results show that sample-based ensembles with SW strategies (DOER, OEOA, OWE, and AddExp) require more processing time than batch-based ensembles (Learn^{++}.NSE, and OAUE). This is because, sample-based ensembles with SW train more models over time. Nevertheless, these ensembles outperform batch-based ensembles in prediction performance.

7.5 Conclusion

An on-line ensemble of regressor models using an OA method which is able to predict on-line variables in changing environments is proposed in this Chapter. The main contribution of the proposed ensemble is that it overcomes the problems of defining

the optimal ensemble size and selecting of the set of most relevant models. These problems are solved by minimizing the ensemble's error on the newest sample. The results have shown that this strategy obtains better performance than combining all the models, in most cases. The proposed ensemble (OEOA) was shown to deliver more accurate estimations of the output variables in industrial applications, as well as in several other cases, when compared to the other state-of-the-art ensembles in the literature. This Chapter also proposed the λ_{DFF}OS-ELM model, an on-line ELM model using variable FF. λ_{DFF}OS-ELM was shown to have higher accuracy when compared to the OS-ELM; and it also improves the performance of well-known state-of-the-art ensembles. In most cases, OEOA and λ_{DFF}OS-ELM have high accuracy and fast adaptivity in non-recurring abrupt drifts (hyperplane data set), and in real-world applications. Thus, the proposed methods can be built for real industrial applications, reducing the time and maintenance costs of traditional measurement systems, such as laboratory measurement systems.

Therefore, the aim of proposing an adaptive ensemble with dynamic selection of models was reached in this chapter, and the proposed ensemble also includes all the other adaptation mechanisms (models' parameters adaptation, models' combination weights adaptation, removal and inclusion of models). The experiments showed that the proposed ensemble is able to overcome the problem of defining the optimal ensemble size, and additionally perform on-line selection of models. Additionally, the aim of proposing an adaptive NN (λ_{DFF}OS-ELM model) which can be dynamically adapted over time was reached in this Chapter. Unlike the standard OS-ELM model, the proposed λ_{DFF}OS-ELM model can be dynamically adapted, according to the changes in system characteristics, using a variable forgetting factor. The proposed adaptive NN model can improve significantly the predictive accuracy of ensemble systems in industrial applications.

Chapter 8

Conclusion

SSs offer a number of advantages for industries. SSs help to reduce the need for hardware measuring tools, improve system reliable and offer alternative tools to the implementation of control policies. Taking these facts into account, technology companies have started to propose and sell software based on SSs for on-line estimation of quality variables [Siemens, 2015]. Although SSs have been employed in industry for estimating quality variables, SS methods (e.g. data preprocessing and learning methods) can also be applied to other application fields, such as, health monitoring, meteorological prediction, financial applications, etc. Chapter 2 identified the current research trends in historical data selection, data preprocessing, computational learning algorithms and SS maintenance. Additionally, Chapter 3 reported the current trends in single on-line learning algorithms and ensemble on-line learning algorithms.

Motivated by the current problems and challenges in SS modeling, the thesis proposed several methodologies for improving the predictive performance on SS applications. The main contribution of the thesis is the proposal of several methodologies for automatic design of ensemble learning systems in order to improve the on-line output prediction in SS applications. Experimental results using real-world data sets showed that, in most cases, the proposed methodologies are able to provide more accurate estimations of the quality variables in industrial applications than the state-of-the-art methods in the literature. The thesis also reported experimental results of the methodologies using artificial data sets with several types of changes. Results of these data sets were important to validate and prove the performance of

the proposed methodologies over the state-of-the-art approaches.

In Chapter 4, a methodology for automatic NN ensemble development in regression tasks was proposed. The main contribution is the proposal of techniques that select the best subset of models to be aggregated to the ensemble taking into account the key factors of ensemble systems (i.e. diversity, number of models, and combination strategy). The proposed approach employs GA and SA to select models and the optimal combination strategy. The proposed methodologies obtained superior performance when compared to state-of-the-art ensemble systems (Bagging, NCL, AdaBoost, and GASEN) in two well-known regression data sets and three real-world industrial data sets. Although the proposed methodology can achieve good results, it does not incorporate adaptive mechanisms which may be necessary to guarantee the ensemble performance in time-varying applications. Therefore, future research topics can be considered to improve the accuracy of the proposed methodologies: dynamic meta-heuristics to dynamically include and remove models over time; dynamic selection of the combination strategy; and dynamic tunning of the NNs' architectures.

Chapter 5 proposed an adaptive ensemble (OWE) which is able to learn incrementally sample by sample in the presence of several types of changes and simultaneously retain old information in scenarios where changes may recur. The main contributions of OWE are dynamic assignment of models' combination weights that takes into account the models' errors on the past and current windows using a discounting factor that decreases or increases the contribution of old windows; dynamic removal and inclusion of models; and regression scope. Four artificial data sets with concept drifts and two well-known real-world industrial data sets were employed to compare the proposed approach to Learn^{++}.NSE, ILLSA, AddExp, and RPLS. The tests showed that OWE has capability to deal with the concept drifts. The experimental results revealed that, in most cases, OWE achieves better accuracy when compared to the state-of-the-art approaches, and in some cases, OWE has comparable accuracy to the state-of-the-art approaches. Despite the attractive characteristics of OWE, its accuracy may be related to the windows' size. To cover these limitations, a variable window's size that adapts according to the process dynamics can be investigated as a future research topic. In this case, the actual window's size should lead the window to contain relevant samples that maximize the representativeness of the current concept, avoiding that the actual window contains samples of an old

concept. For example, the window's size can be automatically reduced if a concept change is detected, so that old samples are discarded; and the window's size can be automatically increased, when the system is learning a new concept [Lazarescu *et al.*, 2004].

In Chapter 6, a dynamic and on-line ensemble regression (DOER) approach of OS-ELM models with fast adaptation capability for on-line prediction was proposed. DOER brings together desired properties which are not given by OWE: on-line inclusion and removal of models to keep only the most accurate models with respect to the current state of the system; dynamic adaptation of the models' combination weights based on their on-line predictions on the recent samples; and on-line model retraining. Experiments on four artificial data sets and six real-world industrial data sets were reported to evaluate the effectiveness of DOER over OS-ELM*s*, OS-ELM*b*, OS-ELM*s*-SW, OS-ELM*b*-SW, OB, Learn^{++}.NSE, OAUE, AddExp, and OWE. Results showed that DOER has high adaptation capability. DOER is not only comparable to the state-of-the-art approaches, but in most cases, DOER has better accuracy when compared to them. The experimental results showed that the proposed methodology can more accurately predict important variables in SS applications when compared to all the tested state-of-the-art approaches. Overall, the proposed DOER method has limited capability for predicting local and abrupt drift data sets, and gradual recurring data sets. This may happen because the proposed method loses information about the past scenarios. Moreover, the window's size setting may have an important role in some data sets (e.g. the debutanizer column data set). In these cases, it is important to apply extra experiments using the proposed method to define the window's size. Therefore, strategies for selecting the best window's size can be considered as a future research topic. Moreover, it seems to be interesting to propose an adaptive setting of α. In this way, α should be set to a high value when a change occurs, and to a low value when no changes are detected.

Chapter 7 proposed an adaptive ensemble (OEOA) that dynamically selects an optimal ensemble size and composition of the subset of models based on the minimization of the ensemble error on the newest sample. The proposed approach overcomes the problem of defining the optimal ensemble size and composition, and in most cases it obtains better performance when compared to the case where all models are aggregated. Chapter 7 also proposed an on-line sequential ELM model

using the DFF method. In most cases, OEOA delivers more accurate estimations of the output variables in industrial applications when compared to the other state-of-the-art ensembles in the literature. Also, the proposed λ_{DFF}OS-ELM model showed higher accuracy when compared to the OS-ELM; and the integration of the proposed λ_{DFF}OS-ELM also improves the performance of well-known state-of-the-art ensembles. Other strategies for ordering aggregation can be considered as a future work. Specifically, ordering aggregation of models according to their accuracies and their diversities.

In this thesis, problems related to the learning of SS models were addressed; and methodologies were proposed and developed in order to overcome such problems. The proposed methods were compared with state-of-the art approaches: SA-NNE and GA-NNE achieved higher accuracy than all the state-of-the-art approaches for all data sets (Chapter 4); OWE, in most cases, had better performance when compared to the state-of-the-art approaches (Chapter 5); DOER, an adaptive ensemble with fast adaptation capability, was successfully tested in predicting important variables in industrial applications (Chapter 6); OEOA aims to dynamically select the best number of models to be aggregated into the ensemble, the results showed that OEOA can deliver accurate on-line estimations of key variables in industrial processes (Chapter 7). Therefore, all the proposed methods can be successfully applied to SS applications. However, there are still general aspects in SS modeling that need future work, such as:

- Data preprocessing approaches should be developed to track environmental changes. Methodologies for dynamic selection of variables that influence the model output and their respective time delays should be investigated. Techniques for on-line data scaling, on-line outlier detection, and for on-line overcoming of missing data should be proposed;

- In SS applications, the real output samples for the SS adaptation are given sporadically or with high delays. In such cases, the application of the SS may correspond to situations where prediction may be performed in every sampling interval, while model learning and adaptation may be applicable only on a subset of the sampling intervals (e.g. when the output is available). In this case, additional tests can be employed to measure the influence of the availability of the real output in the SS performance [Kadlec and Gabrys,

2011]. Semi-supervised learning methods can also be considered to estimate the real output samples when unlabeled samples (i.e. samples containing only input data) are given [Ge and Song, 2011].

- Change detection and change classification techniques should be investigated, so that the SS model can be adapted based on the type of change;
- Model maintenance strategies should be continued to be researched and improved to avoid the SS degradation. Further mechanisms to reduce the maintenance and implementation costs should be developed.

Bibliography

[Abe, 2010] Chigeo Abe. *Support Vector Machines for Pattern Classification*. Advances in Computer Vision and Pattern Recognition. Springer-Verlag London, 2nd ed., 2010. (Cited in page 29).

[Abonyi, 2002] Janos Abonyi. *Fuzzy Model Identification for Control*. Birkhäuser, 2002. (Cited in page 16).

[Abusnina and Kudenko, 2013] Ali Abusnina and Daniel Kudenko. Adaptive Soft Sensor Based on Moving Gaussian Process Window. In: *IEEE Int. Conf. on Industrial Technology*, ICIT'13, pp. 1051–1056. February 25-28 2013. (Cited in pages 12, 13, and 21).

[Ahmed *et al.*, 2009] Faisal Ahmed, Salman Nazir, and Yeong Yeo. A Recursive PLS-based Soft Sensor for Prediction of the Melt Index During Grade Change Operations in HDPE Plant. *Korean Journal of Chemical Engineering*, vol. 26, pp. 14–20, January 2009. (Cited in pages 21, 55, and 104).

[Alpaydin, 2004] Ethem Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2004. (Cited in pages 14, and 26).

[Antón *et al.*, 2013] Juan Carlos Álvarez Antón, Paulino José García Nieto, Cecilio Blanco Viejo, and José Antonio Vilán Vilán. Support Vector Machines Used to Estimate the Battery State of Charge. *IEEE Transactions on Power Electronics*, vol. 28, no. 12, pp. 5919–5926, December 2013. (Cited in page 29).

[Arlot and Celisse, 2010] Sylvain Arlot and Alain Celisse. A Survey of Cross-validation Procedures for Model Selection. *Statistics Surveys*, vol. 4, pp. 40–79, 2010. (Cited in pages 19, and 56).

[Baena-García *et al.*, 2006] Manuel Baena-García, José del Campo-Ávila, Raúl Fidalgo, Albert Bifet, Ricard Gavaldà, and Rafael Morales-Bueno. Early Drift Detection Method. In: *ECML/PKDD 2006 Workshop on Knowledge Discovery from Data Streams*, pp. 1–10. September 18 2006. (Cited in page 44).

[Ben-Israel and Greville, 2003] Adi Ben-Israel and Thomas N.E. Greville. *Generalized Inverses: Theory and Applications*. Springer-Verlag, New York, USA, 2nd ed., 2003. (Cited in page 34).

[Bhattacharya *et al.*, 2012] Sandip Bhattacharya, Kamal Pal, and Surjya K. Pal. Multi-sensor Based Prediction of Metal Deposition in Pulsed Gas Metal Arc Welding Using Various Soft Computing Models. *Applied Soft Computing*, vol. 12, no. 1, pp. 498–505, January 2012. (Cited in page 21).

[Bifet and Gavaldà, 2007] Albert Bifet and Ricard Gavaldà. Learning from Time-Changing Data with Adaptive Windowing. In: *Proc. of the 2007 SIAM Int. Conf. on Data Mining*, pp. 443–448. April 26-28 2007. (Cited in page 45).

[Bishop, 2006] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006. (Cited in page 27).

[Bobál *et al.*, 2005] V. Bobál, J. Böhm, J. Fessl, and J. Machácek. *Digital Self-tuning Controllers: Algorithms, Implementation and Applications*. Advanced Textbooks in Control and Signal Processing. Springer, 2005. (Cited in pages 7, 150, 152, 153, and 160).

[Bobál and Chalupa, 2008] Vladimir Bobál and Petr Chalupa. *Self-Tuning Controllers Simulink Library*. Department of Control Theory, Institute of Information Technologies, Faculty of Applied Informatics, Tomas Bata University in Zlín, Zlín, Czech Republic, March 2008.
`http://www.mathworks.com/matlabcentral/fileexchange/8381-stcsl-standard-version`.
(Cited in page 153).

[Breiman, 1996] Leo Breiman. Bagging Predictors. *Machine Learning*, vol. 24, no. 2, pp. 123–140, August 1996. (Cited in pages 18, and 40).

[Brown *et al.*, 2005a] Gavin Brown, Jeremy Wyatt, Rachel Harris, and Xin Yao. Diversity Creation Methods: A survey and Categorisation. *Information Fusion*, vol. 6, no. 1, pp. 5–20, March 2005. (Cited in pages 38, 39, 40, and 87).

[Brown *et al.*, 2005b] Gavin Brown, Jeremy L. Wyatt, and Peter Tiňo. Managing Diversity in Regression Ensembles. *Journal of Machine Learning Research*, vol. 6, pp. 1621–1650, December 2005. (Cited in pages 27, 28, 39, and 60).

[Brzezinski and Stefanowski, 2014] Dariusz Brzezinski and Jerzy Stefanowski. Combining Block-based and Online Methods in Learning Ensembles from Concept Drifting Data Streams. *Information Sciences*, vol. 265, pp. 50–67, May 2014. (Cited in pages 46, 48, 92, 94, 126, 130, 134, and 153).

[Butcher *et al.*, 2013] J. B. Butcher, D. Verstraeten, B. Schrauwen, C. R. Day, and P. W. Haycock. Reservoir Computing and Extreme Learning Machines for Non-linear Time-series Data Analysis. *Neural Networks*, vol. 38, pp. 76–89, February 2013. (Cited in page 33).

[Cao and Schwartz, 2000] Liyu Cao and Howard Schwartz. A Directional Forgetting Algorithm Based on the Decomposition of the Information Matrix. *Automatica*, vol. 36, no. 11, pp. 1725–1731, November 2000. (Cited in pages 150, and 152).

[Caruana *et al.*, 2000] Rich Caruana, Steve Lawrence, and Lee Giles. Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping. In: *Proceedings Neural Information Processing Systems Conference*, NIPS'00, pp. 402–408. November 28-30 2000. (Cited in pages 17, 33, and 74).

[Castro and Zuben, 2011] Pablo A. Dalbem Castro and Fernando José Von Zuben. Learning Ensembles of Neural Networks by Means of a Bayesian Artificial Immune System. *IEEE Transactions on Neural Networks*, vol. 22, no. 2, pp. 304–316, February 2011. (Cited in pages 40, and 61).

[Chandra *et al.*, 2006] Arjun Chandra, Huanhuan Chen, and Xin Yao. Trade-Off Between Diversity and Accuracy in Ensemble Generation. In: *Multi-Objective Machine Learning*, vol. 16 of *Studies in Computational Intelligence*, pp. 429–464. Springer, 2006. (Cited in pages 39, and 40).

[Chandrashekar and Sahin, 2014] Girish Chandrashekar and Ferat Sahin. A Survey on Feature Selection Methods. *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, January 2014. (Cited in page 14).

[Chen *et al.*, 2011] Kun Chen, Jun Ji, Haiqing Wang, Yi Liu, and Zhihuan Song. Adaptive Local Kernel-based Learning for Soft Sensor Modeling of Nonlinear Processes. *Chemical Engineering Research and Design*, vol. 89, no. 10, pp. 2117–2124, October 2011. (Cited in page 21).

[Cheng and Liu, 2015] Zhong Cheng and Xinggao Liu. Optimal Online Soft Sensor for Product Quality Monitoring in Propylene Polymerization Process. *Neurocomputing*, vol. 149, Part C, pp. 1216–1224, February 2015. (Cited in pages 16, and 21).

[Chincholkar and Herrmann, 2008] Mandar Chincholkar and Jeffrey W. Herrmann. Estimating Manufacturing Cycle Time and Throughput in Flow Shops with Process Drift and Inspection. *International Journal of Production Research*, vol. 46, no. 24, pp. 7057–7072, December 2008. (Cited in page 126).

[Chu and Zaniolo, 2004] Fang Chu and Carlo Zaniolo. Fast and Light Boosting for Adaptive Mining of Data Streams. In: *Advances in Knowledge Discovery and Data Mining*, vol. 3056 of *Lecture Notes in Computer Science*, pp. 282–292. Springer Berlin Heidelberg, 2004. (Cited in pages 41, 45, 46, 47, and 92).

[Coelho and Nascimento, 2010] André L. V. Coelho and Diego S. C. Nascimento. On the Evolutionary Design of Heterogeneous Bagging Models. *Neurocomputing*, vol. 73, no. 16-18, pp. 3319–3322, October 2010. (Cited in pages 40, 41, and 60).

[Coelho and Zuben, 2006] G. P. Coelho and F. J. Von Zuben. The Influence of the Pool of Candidates on the Performance of Selection and Combination Techniques in Ensembles. In: *Proc. of the Int. Joint Conf. on Neural Networks*, IJCNN'06, pp. 5132–5139. July 16-21 2006. (Cited in page 150).

[Cordiner, 2009] Alister Cordiner. AdaBoost Toolbox - A Matlab Toolbox for Adaptive Boosting. Tech. rep., School of Computer Science and Software Engineering, University of Wollongong, Wollongong, Australia, 2009.
`http://thedeadbeef.files.wordpress.com/2010/07/techreport_boosting.pdf`. (Cited in page 88).

[Cristinacce and Cootes, 2007] David Cristinacce and Tim F. Cootes. Boosted Regression Active Shape Models. In: *Proc. of the British Machine Vision Conference*, vol. 2, pp. 880–889. BMVA Press, September 10-13 2007. (Cited in pages 40, 62, and 87).

[Cybenko, 1989] G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989. (Cited in page 28).

[Das, 2001] Sanmay Das. Filters, Wrappers and a Boosting-Based Hybrid for Feature Selection. In: *Proc. of the 8th Int. Conf. on Machine Learning*, ICML'01, pp. 74–81. June 28 - July 1 2001. (Cited in page 15).

[Deng *et al.*, 2014] Wan-Yu Deng, Qing-Hua Zheng, and Zhong-Min Wang. Cross-person Activity Recognition Using Reduced Kernel Extreme Learning Machine. *Neural Networks*, vol. 53, pp. 1–7, May 2014. (Cited in page 150).

[Ding *et al.*, 2012] Shifei Ding, Li Xu, Chunyang Su, and Fengxiang Jin. An Optimizing Method of RBF Neural Network Based on Genetic Algorithm. *Neural Computing and Applications*, vol. 21, no. 2, pp. 333–336, March 2012. (Cited in page 17).

[Dondeti *et al.*, 2005] Satyanarayana Dondeti, Kamarajan Kannan, and Rajappan Manavalan. Genetic Algorithm Optimized Neural Networks Ensemble for Estimation of Mefenamic Acid and Paracetamol in Tablets. *Acta Chimica Slovenica*, vol. 52, no. 4, pp. 440–449, December 2005. (Cited in pages 41, 62, and 65).

[Drucker, 1997] Harris Drucker. Improving Regressors using Boosting Techniques. In: *Proc. of the 14th Int. Conf. on Machine Learning*, ICML'97, pp. 107–115. Morgan Kaufmann Publishers Inc., 1997. (Cited in pages 92, and 94).

[Elwell and Polikar, 2009] Ryan Elwell and Robi Polikar. Incremental Learning in Nonstationary Environments with Controlled Forgetting. In: *Proc. of the Int. Joint Conf. on Neural Networks*, pp. 771–778. June 14-19 2009. (Cited in pages 46, 47, 104, and 134).

[Elwell and Polikar, 2011] Ryan Elwell and Robi Polikar. Incremental Learning of Concept Drift in Nonstationary Environments. *IEEE Transactions on*

Neural Networks, vol. 22, no. 10, pp. 1517–1531, October 2011. (Cited in pages 6, 44, 47, 92, 93, 105, 134, and 165).

[Escobar *et al.*, 2015] R. F. Escobar, C. M. Astorga-Zaragoza, J. A. Hernández, D. Juárez-Romero, and C. D. García-Beltrán. Sensor Fault Compensation Via Software Sensors: Application in a Heat Pump's Helical Evaporator. *Chemical Engineering Research and Design*, vol. 93, pp. 473–482, January 2015. (Cited in pages 16, and 21).

[Facco *et al.*, 2009] Pierantonio Facco, Franco Doplicher, Fabrizio Bezzo, and Massimiliano Barolo. Moving Average PLS Soft Sensor for Online Product Quality Estimation in an Industrial Batch Polymerization Process. *Journal of Process Control*, vol. 19, no. 3, pp. 520–529, March 2009. (Cited in pages 19, and 21).

[Feely, 2000] R. Feely. *Predicting Stock Market Volatility Using Neural Networks.* Bachelor's thesis (unpublished), Department of Computer Science, Trinity College Dublin, 2000. (Cited in pages 93, 94, and 97).

[Fortuna *et al.*, 2009] L. Fortuna, S. Graziani, and M. G. Xibilia. Comparison of Soft-Sensor Design Methods for Industrial Plants Using Small Data Sets. *IEEE Transactions on Instrumentation and Measurement*, vol. 58, no. 8, pp. 2444–2451, August 2009. (Cited in pages 4, 18, 41, 60, and 92).

[Fortuna *et al.*, 2006] Luigi Fortuna, Salvatore Graziani, Alessandro Rizzo, and Maria Gabriella Xibilia. *Soft Sensors for Monitoring and Control of Industrial Processes (Advances in Industrial Control).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. (Cited in pages xxi, 3, 11, 13, 14, 16, 19, 20, 22, 73, and 132).

[Freund and Schapire, 1997] Yoav Freund and Robert E. Schapire. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119 – 139, August 1997. (Cited in page 94).

[Friedman, 1991] Jerome H. Friedman. Multivariate Adaptive Regression Splines. *The Annals of Statistics*, vol. 19, no. 1, pp. 1–67, March 1991. (Cited in page 72).

[Gabrys, 2005] B. Gabrys. *Do Smart Adaptive Systems Exist? - Introduction*, chap. 1, pp. 1–17. Springer, 2005. (Cited in page 42).

[Galicia *et al.*, 2012] Hector J. Galicia, Q. Peter He, and Jin Wang. Comparison of the Performance of a Reduced-order Dynamic PLS Soft Sensor with Different Updating Schemes for Digester Control. *Control Engineering Practice*, vol. 20, no. 8, pp. 747–760, August 2012. (Cited in pages 12, 21, and 133).

[Gama *et al.*, 2009] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. Issues in Evaluation of Stream Learning Algorithms. In: *Proc. of the 15th Int. Conf. on Knowledge Discovery and Data Mining*, KDD'09, pp. 329–338. June 28 - July 1 2009. (Cited in page 19).

[Ge and Song, 2010] Zhiqiang Ge and Zhihuan Song. A Comparative Study of Just-in-time-learning Based Methods for Online Soft Sensor Modeling. *Chemometrics and Intelligent Laboratory Systems*, vol. 104, no. 2, pp. 306–317, December 2010. (Cited in page 21).

[Ge and Song, 2011] Zhiqiang Ge and Zhihuan Song. Semisupervised Bayesian Method for Soft sensor Modeling with Unlabeled Data Samples. *AIChE Journal*, vol. 57, no. 8, pp. 2109–2119, August 2011. (Cited in page 179).

[Ge and Song, 2014] Zhiqiang Ge and Zhihuan Song. Ensemble Independent Component Regression Models and Soft Sensing Application. *Chemometrics and Intelligent Laboratory Systems*, vol. 130, pp. 115–122, January 2014. (Cited in page 92).

[Ge *et al.*, 2014] Zhiqiang Ge, Zhihuan Song, and Manabu Kano. External Analysis-based Regression Model for Robust Soft Sensing of Multimode Chemical Processes. *AIChE Journal*, vol. 60, no. 1, pp. 136–147, January 2014. (Cited in page 21).

[Geladi and Kowalski, 1986] Paul Geladi and Bruce R. Kowalski. Partial Least-Squares Regression: A Tutorial. *Analytica Chimica Acta*, vol. 185, pp. 1–17, 1986. (Cited in pages 17, and 36).

[Gjerkes *et al.*, 2011] Henrik Gjerkes, Joze Malensek, Anze Sitar, and Iztok Golobic. Product Identification in Industrial Batch Fermentation using a Variable Forget-

ting Factor. *Control Engineering Practice*, vol. 19, no. 10, pp. 1208–1215, October 2011. (Cited in pages 45, and 126).

[Gomes *et al.*, 2014] João Bártolo Gomes, Mohamed Medhat Gaber, Pedro A. C. Sousa, and Ernestina Menasalvas. Mining Recurring Concepts in a Dynamic Feature Space. *IEEE Trans. on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 95–110, January 2014. (Cited in page 92).

[Grbić *et al.*, 2013] Ratko Grbić, Dražen Slišković, and Petr Kadlec. Adaptive Soft Sensor for Online Prediction and Process Monitoring Based on a Mixture of Gaussian Process Models. *Computers & Chemical Engineering*, vol. 58, pp. 84–97, November 2013. (Cited in pages 13, 14, 21, 22, 73, 132, and 159).

[Grbovic and Vucetic, 2011] Mihajlo Grbovic and Slobodan Vucetic. Tracking Concept Change with Incremental Boosting by Minimization of the Evolving Exponential Loss. In: *Proc. of the European Conf. on Machine Learning and Knowledge Discovery in Databases - Volume Part I*, pp. 516–532. Springer, 2011. (Cited in pages 46, and 47).

[Hagan and Menhaj, 1994] Martin T. Hagan and Mohammad B. Menhaj. Training Feedforward Networks with the Marquardt Algorithm. *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 489–501, November 1994. (Cited in pages 31, 32, and 63).

[Hagan *et al.*, 1996] Martin T. Hagan, Howard B. Demuth, and Mark Beale. *Neural Network Design*. PWS Publishing, Boston, MA, USA, 1996. (Cited in pages 30, and 32).

[Han and Kamber, 2005] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005. (Cited in pages 11, and 15).

[Hashem, 1994] Sherif Hashem. Optimal Linear Combinations of Neural Networks. *Neural Networks*, vol. 10, no. 4, pp. 599–614, June 1994. (Cited in page 64).

[Haupt, 2004] Randy L. Haupt Sue Ellen Haupt. *Practical Genetic Algorithms*. Wiley-Interscience, 2nd ed., 2004. (Cited in page 70).

[Haykin, 1996] Simon Haykin. *Adaptive Filter Theory*. Prentice-Hall, Upper Saddle River, NJ, USA, 3rd ed., 1996. (Cited in page 53).

[Haykin, 1999] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd ed., 1999. (Cited in page 28).

[He *et al.*, 2015] Kaixun He, Feng Qian, Hui Cheng, and Wenli Du. A Novel Adaptive Algorithm with Near-infrared Spectroscopy and Its Application in Online Gasoline Blending Processes. *Chemometrics and Intelligent Laboratory Systems*, vol. 140, pp. 117–125, August 2015. (Cited in pages 16, 21, 35, and 55).

[Ho *et al.*, 2010] Kevin I.-J. Ho, Chi-Sing Leung, and John Sum. Convergence and Objective Functions of Some Fault/Noise-Injection-Based Online Learning Algorithms for RBF Networks. *IEEE Transactions on Neural Networks*, vol. 21, no. 6, pp. 938–947, June 2010. (Cited in page 18).

[Holland, 1992] John H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA, 1992. (Cited in page 68).

[Hornik *et al.*, 1989] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. (Cited in page 28).

[Houck *et al.*, 1996] Christopher R. Houck, Jeffery A. Joines, and Michael G. Kay. A Genetic Algorithm for Function Optimization: A Matlab Implementation. Tech. rep., North Carolina State University, Raleigh, NC, USA, 1996. `http://www.daimi.au.dk/~pmn/Matlab/dochome/toolbox/GAOT/`. (Cited in page 86).

[Hsu *et al.*, 2011] Hui-Huang Hsu, Cheng-Wei Hsieh, and Ming-Da Lu. Hybrid Feature Selection by Combining Filters and Wrappers. *Expert Systems with Applications*, vol. 38, no. 7, pp. 8144–8150, July 2011. (Cited in page 15).

[Hu *et al.*, 2013] Yi Hu, Hehe Ma, and Hongbo Shi. Enhanced Batch Process Monitoring Using Just-in-time-learning Based Kernel Partial Least Squares. *Chemometrics and Intelligent Laboratory Systems*, vol. 123, pp. 15–27, April 2013. (Cited in page 21).

[Huang, 2014] Guang-Bin Huang. An Insight into Extreme Learning Machines: Random Neurons, Random Features and Kernels. *Cognitive Computation*, vol. 6, no. 3, pp. 376–390, September 2014. (Cited in page 31).

[Huang, 2015] Guang-Bin Huang. What are Extreme Learning Machines? Filling the Gap between Frank Rosenblatt's Dream and John von Neumann's Puzzle. *Cognitive Computation*, vol. 7, no. 3, pp. 263–278, May 2015. (Cited in page 31).

[Huang *et al.*, 2006] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme Learning Machine: Theory and Applications. *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, December 2006. (Cited in pages 7, 31, 33, 49, and 160).

[Ikonomovska, 2012] Elena Ikonomovska. *Algorithms for Learning Regression Trees and Ensembles on Evolving Data Streams*. PhD thesis, Jožef Stefan Int. Postgraduate School, Ljubljana, Slovenia, October 2012. (Cited in pages 19, 44, 100, 102, 132, and 159).

[Iliyas *et al.*, 2013] Surajdeen A. Iliyas, Moustafa Elshafei, Mohamed A. Habib, and Ahmed A. Adeniran. RBF Neural Network Inferential Sensor for Process Emission Monitoring. *Control Engineering Practice*, vol. 21, no. 7, pp. 962–970, July 2013. (Cited in page 21).

[Jeong and Kim, 2005] Dae-Il Jeong and Young-Oh Kim. Rainfall-runoff Models using Artificial Neural Networks for Ensemble Streamflow Prediction. *Hydrological Processes*, vol. 19, no. 19, pp. 3819–3835, December 2005. (Cited in pages 18, 63, and 74).

[Jia *et al.*, 2011] Run-Da Jia, Zhi-Zhong Mao, Yu-Qing Chang, and Lu-Ping Zhao. Soft-Sensor for Copper Extraction Process in Cobalt Hydrometallurgy Based on Adaptive Hybrid Model. *Chemical Engineering Research and Design*, vol. 89, no. 6, pp. 722–728, June 2011. (Cited in page 21).

[Jia and Culver, 2006] Yanbing Jia and Teresa B. Culver. Bootstrapped Artificial Neural Networks for Synthetic Flow Generation with a Small Data Sample. *Journal of Hydrology*, vol. 331, no. 3-4, pp. 580–590, December 2006. (Cited in pages 40, and 63).

[Jin *et al.*, 2014] Huaiping Jin, Xiangguang Chen, Jianwen Yang, and Lei Wu. Adaptive Soft Sensor Modeling Framework Based on Just-in-time Learning and Kernel Partial Least Squares Regression for Nonlinear Multiphase Batch Processes. *Computers & Chemical Engineering*, vol. 71, pp. 77–93, December 2014. (Cited in pages 21, and 126).

[Jin *et al.*, 2012] Xing Jin, Siyun Wang, Biao Huang, and Fraser Forbes. Multiple Model Based LPV Soft Sensor Development with Irregular/Missing Process Output Measurement. *Control Engineering Practice*, vol. 20, no. 2, pp. 165–172, February 2012. (Cited in page 12).

[Jong, 1993] Sijmen de Jong. SIMPLS: An Alternative Approach to Partial Least Squares Regression. *Chemometrics and Intelligent Laboratory Systems*, vol. 18, no. 3, pp. 251–263, March 1993. (Cited in pages 17, 36, and 104).

[Kadlec and Gabrys, 2007] P. Kadlec and B. Gabrys. Nature-Inspired Adaptive Architecture for Soft Sensor Modelling. In: *3rd European Symposium on Nature-inspired Smart Information Systems*, NiSIS'2007, pp. 407–414. November 26-27 2007. (Cited in page 17).

[Kadlec and Gabrys, 2011] Petr Kadlec and Bogdan Gabrys. Local Learning-based Adaptive Soft Sensor for Catalyst Activation Prediction. *AIChE Journal*, vol. 57, no. 5, pp. 1288–1301, August 2011. (Cited in pages 15, 21, 22, 44, 47, 73, 92, 101, 105, 126, 132, 159, and 178).

[Kadlec *et al.*, 2009] Petr Kadlec, Bogdan Gabrys, and Sibylle Strandt. Data-driven Soft Sensors in the Process Industry. *Computers & Chemical Engineering*, vol. 33, no. 4, pp. 795–814, April 2009. (Cited in page 10).

[Kadlec *et al.*, 2011] Petr Kadlec, Ratko Grbić, and Bogdan Gabrys. Review of Adaptation Mechanisms for Data-driven Soft Sensors. *Computers & Chemical Engineering*, vol. 35, no. 1, pp. 1–24, January 2011. (Cited in pages 19, and 42).

[Kaneko and Funatsu, 2014] Hiromasa Kaneko and Kimito Funatsu. Adaptive Soft Sensor Based on Online Support Vector Regression and Bayesian Ensemble Learning for Various States in Chemical Plants. *Chemometrics and Intelligent Laboratory Systems*, vol. 137, pp. 57–66, October 2014. (Cited in pages 16, 21, and 126).

[Kasabov, 1996] Nikola K. Kasabov. *Foundations of Neural Networks, Fuzzy Systems, and Knowledge Engineering.* MIT Press, Cambridge, MA, USA, 1st ed., 1996. (Cited in pages 28, 32, and 64).

[Khatibisepehr and Huang, 2008] Shima Khatibisepehr and Biao Huang. Dealing with Irregular Data in Soft Sensors: Bayesian Method and Comparative Study. *Industrial & Engineering Chemistry Research*, vol. 47, no. 22, pp. 8713–8723, October 2008. (Cited in page 12).

[Kim *et al.*, 2013] Sanghong Kim, Ryota Okajima, Manabu Kano, and Shinji Hasebe. Development of Soft-sensor Using Locally Weighted PLS with Adaptive Similarity Measure. *Chemometrics and Intelligent Laboratory Systems*, vol. 124, pp. 43–49, May 2013. (Cited in page 21).

[Klinkenberg, 2005] Ralf Klinkenberg. Meta-Learning, Model Selection, and Example Selection in Machine Learning Domains with Concept Drift. In: *Proc. of Annual Workshop of the Special Interest Group on Machine Learning, Knowledge Discovery, and Data Mining*, FGML'05, pp. 164–171. October 2005. (Cited in pages 42, 43, 94, and 153).

[Kolter and Maloof, 2005] Jeremy Zico Kolter and Marcus A. Maloof. Using Additive Expert Ensembles to Cope with Concept Drift. In: *Proc. of the 22nd Int. Conf. on Machine Learning*, pp. 449–456. ACM, August 7-11 2005. (Cited in pages 46, 48, 92, 100, 101, 104, 127, 132, 134, 159, and 165).

[Krogh and Vedelsby, 1995] Anders Krogh and Jesper Vedelsby. Neural Network Ensembles, Cross Validations, and Active Learning. In: *Advances in Neural Information Processing Systems*, vol. 8, pp. 231–238. MIT Press, 1995. (Cited in page 38).

[Kulhavý, 1985] Rudolf Kulhavý. *Probabilistic Identification of Time-Variable Systems with Unknown Model of Parameter Evolution.* PhD thesis, Institute of Information Theory and Automation of Czechoslovak Academy of Sciences, Praha, Czechoslovakia, 1985. (in Czech). (Cited in page 153).

[Lan *et al.*, 2009] Yuan Lan, Yeng Chai Soh, and Guang-Bin Huang. Ensemble of Online Sequential Extreme Learning Machine. *Neurocomputing*, vol. 72, no. 13-15, pp. 3391–3395, August 2009. (Cited in pages 4, 18, 41, 50, 60, 92, and 134).

[Lazarescu *et al.*, 2004] Mihai M. Lazarescu, Svetha Venkatesh, and Hung H. Bui. Using Multiple Windows to Track Concept Drift. *Intelligent Data Analysis*, vol. 8, no. 1, pp. 29–59, January 2004. (Cited in page 177).

[Lazarevic and Obradovic, 2001] Aleksandar Lazarevic and Zoran Obradovic. Effective Pruning of Neural Network Classifier Ensembles. In: *Proc. of the Int. Joint Conf. on Neural Networks*, vol. 2 of *IJCNN'01*, pp. 796–801. July 15-19 2001. (Cited in page 150).

[Liang *et al.*, 2006] Nan-Ying Liang, Guang-Bin Huang, P. Saratchandran, and N. Sundararajan. A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks. *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411–1423, November 2006. (Cited in pages 7, 35, 50, 53, 54, 126, 150, and 160).

[Lim *et al.*, 2013] Jun-Seok Lim, Seokjin Lee, and Hee-Suk Pang. Low Complexity Adaptive Forgetting Factor for Online Sequential Extreme Learning Machine (OS-ELM) for Application to Nonstationary System Estimations. *Neural Computing and Applications*, vol. 22, no. 3-4, pp. 569–576, March 2013. (Cited in pages 17, and 150).

[Lin *et al.*, 2007] Bao Lin, Bodil Recke, Jørgen K. H. Knudsen, and Sten Bay Jørgensen. A systematic Approach for Soft Sensor Development. *Computers & Chemical Engineering*, vol. 31, no. 5-6, pp. 419–425, May 2007. (Cited in pages 13, and 132).

[Lin *et al.*, 2009] Bao Lin, Bodil Recke, Torben M. Schmidt, Jørgen K. H. Knudsen, and Sten Bay Jørgensen. Data-Driven Soft Sensor Design with Multiple-Rate Sampled Data: A Comparative Study. *Industrial & Engineering Chemistry Research*, vol. 48, no. 11, pp. 5379–5387, May 2009. (Cited in page 11).

[Lingfang and Yechi, 2012] Sun Lingfang and Wang Yechi. Soft-sensing of Oxygen Content of Flue Gas Based on Mixed Model. *Energy Procedia*, vol. 17, Part A, pp. 221–226, 2012. (Cited in page 21).

[Littlestone and Warmuth, 1994] Nick Littlestone and Manfred K. Warmuth. The Weighted Majority Algorithm. *Information and Computation*, vol. 108, no. 2, pp. 212–261, February 1994. (Cited in page 99).

[Liu *et al.*, 2010] Guohai Liu, Dawei Zhou, Haixia Xu, and Congli Mei. Model Optimization of SVM for a Fermentation Soft Sensor. *Expert Systems with Applications*, vol. 37, no. 4, pp. 2708–2713, April 2010. (Cited in page 21).

[Liu *et al.*, 2012a] Jialin Liu, Ding-Sou Chen, and Ming-Wei Lee. Adaptive Soft Sensors Using Local Partial Least Squares with Moving Window Approach. *Asia-Pacific Journal of Chemical Engineering*, vol. 7, pp. S134–S144, February 2012. (Cited in pages 21, and 43).

[Liu *et al.*, 2009] Yi Liu, Naiping Hu, Haiqing Wang, and Ping Li. Soft Chemical Analyzer Development Using Adaptive Least-Squares Support Vector Regression with Selective Pruning and Variable Moving Window Size. *Industrial & Engineering Chemistry Research*, vol. 48, no. 12, pp. 5731–5741, May 2009. (Cited in pages 21, 22, and 101).

[Liu *et al.*, 2012b] Yi Liu, Zengliang Gao, Ping Li, and Haiqing Wang. Just-in-Time Kernel Learning with Adaptive Parameter Selection for Soft Sensor Modeling of Batch Processes. *Industrial & Engineering Chemistry Research*, vol. 51, no. 11, pp. 4313–4327, February 2012. (Cited in page 21).

[Liu *et al.*, 2013] Yi Liu, Zengliang Gao, and Junghui Chen. Development of Soft-sensors for Online Quality Prediction of Sequential-reactor-multi-grade Industrial Processes. *Chemical Engineering Science*, vol. 102, pp. 602–612, October 2013. (Cited in page 21).

[Liu *et al.*, 2012c] Yiqi Liu, Daoping Huang, Yan Li, and Xuefeng Zhu. Development of a Novel Self-Validating Soft Sensor. *Korean Journal of Chemical Engineering*, vol. 29, no. 9, pp. 1135–11439, September 2012. (Cited in pages 14, 17, and 21).

[Liu *et al.*, 2015] Yiqi Liu, Yongping Pan, and Daoping Huang. Development of a Novel Adaptive Soft-sensor Using Variational Bayesian PLS With Accounting For Online Identification of Key Variables. *Industrial & Engineering Chemistry Research*, vol. 54, no. 1, pp. 338–350, 2015. (Cited in pages 16, 21, and 35).

[Liu and Yao, 1999] Yong Liu and Xin Yao. Ensemble Learning Via Negative Correlation. *Neural Networks*, vol. 12, no. 10, pp. 1399–1404, December 1999. (Cited in page 87).

[Liu *et al.*, 2000] Yong Liu, Xin Yao, and Tetsuya Higuchi. Evolutionary Ensembles with Negative Correlation Learning. *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 4, pp. 380–387, November 2000. (Cited in pages 41, 60, and 61).

[Ludermir *et al.*, 2006] Teresa B. Ludermir, Akio Yamazaki, and Cleber Zanchettin. An Optimization Methodology for Neural Network Weights and Architectures. *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1452–1459, November 2006. (Cited in page 17).

[Lv *et al.*, 2013] You Lv, Jizhen Liu, Tingting Yang, and Deliang Zeng. A Novel Least Squares Support Vector Machine Ensemble Model for NOx Emission Prediction of a Coal-fired Boiler. *Energy*, vol. 55, pp. 319–329, June 2013. (Cited in pages 21, 92, and 126).

[Mallapragada *et al.*, 2009] Pavan Kumar Mallapragada, Rong Jin, Anil K. Jain, and Yi Liu. SemiBoost: Boosting for Semi-Supervised Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 11, pp. 2000–2014, Novemver 2009. (Cited in page 26).

[Maponi, 2007] P. Maponi. The Solution of Linear Systems by Using the Sherman-Morrison Formula. *Linear Algebra and Its Applications*, vol. 420, no. 2-3, pp. 276–294, January 2007. (Cited in page 54).

[Martínez-Muñoz *et al.*, 2009] Gonzalo Martínez-Muñoz, Daniel Hernández-Lobato, and Alberto Suárez. An Analysis of Ensemble Pruning Techniques Based on Ordered Aggregation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 2, pp. 245–259, February 2009. (Cited in pages 41, and 60).

[Martins *et al.*, 2010] João Paulo A. Martins, Reinaldo F. Teófilo, and Márcia M. C. Ferreira. Computational Performance and Cross-Validation Error Precision of Five PLS Algorithms using Designed and Real Data Sets. *Journal of Chemometrics*, vol. 24, no. 6, pp. 320–332, June 2010. (Cited in page 36).

[Matias *et al.*, 2013] Tiago Matias, Dulce Gabriel, Francisco Souza, Rui Araújo, and J. Costa Pereira. Fault Detection and Replacement of a Temperature Sensor

in a Cement Rotary Kiln. In: *Proc. of the 18th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, ETFA'13, pp. 1–8. September 10-13 2013. (Cited in pages 13, 21, and 150).

[Matias *et al.*, 2014] Tiago Matias, Francisco Souza, Rui Araújo, and Carlos Henggeler Antunes. Learning of a Single-hidden Layer Feedforward Neural Network using an Optimized Extreme Learning Machine. *Neurocomputing*, vol. 129, pp. 428–436, April 2014. (Cited in page 62).

[Mendes *et al.*, 2013] Jérôme Mendes, Rui Araújo, and Francisco Souza. Adaptive Fuzzy Identification and Predictive Control for Industrial Processes. *Expert Systems with Applications*, vol. 40, no. 17, pp. 6964–6975, December 2013. (Cited in page 152).

[Michalewicz and Fogel, 2000] Zbigniew Michalewicz and David B. Fogel. *How to Solve it: Modern Heuristics*. Springer-Verlag, Berlin, Germany, 2000. (Cited in page 71).

[Miche *et al.*, 2010] Yoan Miche, Antti Sorjamaa, Patrick Bas, Olli Simula, Christian Jutten, and Amaury Lendasse. OP-ELM: Optimally Pruned Extreme Learning Machine. *IEEE Transactions on Neural Networks*, vol. 21, no. 1, pp. 158–162, January 2010. (Cited in page 29).

[Minku and Yao, 2012] Leandro L. Minku and Xin Yao. DDD: A New Ensemble Approach for Dealing with Concept Drift. *IEEE Trans. on Knowledge and Data Engineering*, vol. 24, no. 4, pp. 619–633, April 2012. (Cited in page 45).

[Minku *et al.*, 2010] Leandro L. Minku, Allan P. White, and Xin 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, May 2010. (Cited in pages 44, and 101).

[Miranda, 2012] Luís Filipe Pereira Miranda. *Metodologias de Inteligência Computacional para Projecto de Sensores Virtuais em Processos Industriais*. Master's thesis, Dep. of Electrical and Computer Engineering, University of Coimbra, Coimbra, Portugal, 2012. (Cited in pages 73, and 105).

[Napoli and Xibilia, 2011] Giuseppe Napoli and Maria Gabriella Xibilia. Soft Sensor Design for a Topping Process in the Case of Small Datasets. *Computers & Chemical Engineering*, vol. 35, no. 11, pp. 2447–2456, November 2011. (Cited in page 21).

[Narendra and Parthasarathy, 1990] Kumpati S. Narendra and Kannan Parthasarathy. Identification and Control of Dynamical Systems Using Neural Networks. *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4–27, March 1990. (Cited in pages 2, and 27).

[Nguyen and Widrow, 1990] Derrick Nguyen and Bernard Widrow. Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights. In: *Proc. of the IEEE Int. Joint Conf. on Neural Networks*, vol. 3 of *IJCNN'90*, pp. 21–26. June 17-21 1990. (Cited in pages 32, and 64).

[Ni *et al.*, 2014] Wangdong Ni, Steven D. Brown, and Ruilin Man. A Localized Adaptive Soft Sensor for Dynamic System Modeling. *Chemical Engineering Science*, vol. 111, pp. 350–363, May 2014. (Cited in pages 13, 16, 17, 19, 21, and 55).

[Nishida and Yamauchi, 2007] Kyosuke Nishida and Koichiro Yamauchi. Adaptive Classifiers-Ensemble System for Tracking Concept Drift. In: *Int. Conf. on Machine Learning and Cybernetics*, vol. 6, pp. 3607–3612. August 19-22 2007. (Cited in pages 46, 48, and 104).

[Nishida *et al.*, 2005] Kyosuke Nishida, Koichiro Yamauchi, and Takashi Omori. ACE: Adaptive Classifiers-Ensemble System for Concept-Drifting Environments. In: *Multiple Classifier Systems*, vol. 3541 of *Lecture Notes in Computer Science*, pp. 176–185. Springer, 2005. (Cited in page 48).

[Oza and Russell, 2001] Nikunj C. Oza and Stuart Russell. Experimental Comparisons of Online and Batch Versions of Bagging and Boosting. In: *Proc. of the 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD'01, pp. 359–364. 2001. (Cited in pages 41, 47, 60, 92, and 134).

[Pani *et al.*, 2013] Ajaya Kumar Pani, Vamsi Krishna Vadlamudi, and Hare Krishna Mohanta. Development and Comparison of Neural Network Based Soft Sensors for Online Estimation of Cement Clinker Quality. *ISA Transactions*, vol. 52, no. 1, pp. 19–29, January 2013. (Cited in page 21).

[Pao *et al.*, 1994] Yoh-Han Pao, Gwang-Hoon Park, and Dejan J. Sobajic. Learning and Generalization Characteristics of the Random Vector Functional-link Net. *Neurocomputing*, vol. 6, no. 2, pp. 163–180, April 1994. Backpropagation, Part IV. (Cited in page 31).

[Partalas *et al.*, 2008] Ioannis Partalas, Grigorios Tsoumakas, Evaggelos V. Hatzikos, and Ioannis Vlahavas. Greedy Regression Ensemble Selection: Theory and an Application to Water Quality Prediction. *Information Sciences*, vol. 178, no. 20, pp. 3867–3879, October 2008. (Cited in page 150).

[Peng *et al.*, 2005] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature Selection Based on Mutual Information: Criteria of Max-dependency, Max-relevance, and Min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1226–1238, August 2005. (Cited in page 14).

[Polikar, 2006] Robi Polikar. Ensemble Based Systems in Decision Making. *IEEE Circuits and Systems Magazine*, vol. 6, no. 3, pp. 21–45, Third Quarter 2006. (Cited in pages 18, and 64).

[Polikar, 2012] Robi Polikar. Ensemble Learning. In: *Ensemble Machine Learning: Methods and Applications*, pp. 1–34. Springer, New York, NY, USA, 2012. (Cited in pages 4, and 45).

[Polikar *et al.*, 2002] Robi Polikar, Jeff Byorick, Stefan Krause, Anthony Marino, and Michael Moreton. Learn++: A Classifier Independent Incremental Learning Algorithm for Supervised Neural Networks. In: *Proc. IEEE Int. Joint Conf. on Neural Networks*, vol. 2 of *IJCNN'02*, pp. 1742–1747. May 12-17 2002. (Cited in page 42).

[Qin, 1998] S. Joe Qin. Recursive PLS Algorithms for Adaptive Data Modeling. *Computers & Chemical Engineering*, vol. 22, no. 4-5, pp. 503–514, January 1998. (Cited in pages 17, 35, 45, 55, 103, and 126).

[Rao and Mitra, 1972] C. Radhakrishna Rao and Sujit Kumar Mitra. Generalized Inverse of a Matrix and Its Applications. In: *Proc. 6th Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Theory of Statistics*, pp. 601–620. University of California Press, Berkeley, CA, USA, 1972. (Cited in page 35).

[Re and Valentini, 2012] Matteo Re and Giorgio Valentini. *Ensemble Methods: A Review*, chap. 26, pp. 563–582. Chapman & Hall/CRC. CRC press, 2012. (Cited in page 41).

[Ries *et al.*, 2007] Michal Ries, Olivia Nemethova, and Markus Rupp. Performance Evaluation of Mobile Video Quality Estimators. In: *Proc. of the 15th European Signal Processing Conference*, pp. 159–163. Poznań, Poland, September 3-7 2007. (Cited in page 41).

[Rogina *et al.*, 2011] A. Rogina, I. Šiško, I. Mohler, Ž. Ujević, and N. Bolf. Soft Sensor for Continuous Product Quality Estimation (In Crude Distillation Unit). *Chemical Engineering Research and Design*, vol. 89, no. 10, pp. 2070–2077, October 2011. (Cited in pages 14, and 21).

[Rosin and Fierens, 1995] Paul L. Rosin and Freddy Fierens. Improving Neural Network Generalisation. In: *Proceedings of International Geoscience and Remote Sensing Symposium (IGARSS'95)*, vol. 2, pp. 1255–1257. Firenze, Italy, 1995. (Cited in page 62).

[Rullo *et al.*, 2014] P. Rullo, L. Nieto Degliuomini, M. García, and M. Basualdo. Model Predictive Control to Ensure High Quality Hydrogen Production for Fuel Cells. *International Journal of Hydrogen Energy*, vol. 39, no. 16, pp. 8635–8649, May 2014. (Cited in pages 16, and 21).

[Rumelhart *et al.*, 1986] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Internal Representations by Error Propagation. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1: Foundations, pp. 318–362. MIT Press, 1986. (Cited in page 30).

[Santos *et al.*, 2009] Eulanda M. Dos Santos, Robert Sabourin, and Patrick Maupin. Overfitting Cautious Selection of Classifier Ensembles with Genetic Algorithms. *Information Fusion*, vol. 10, no. 2, pp. 150–162, April 2009. (Cited in page 155).

[Saptoro, 2014] Agus Saptoro. State of the Art in the Development of Adaptive Soft Sensors based on Just-in-Time Models. *Procedia Chemistry*, vol. 9, pp. 226–234, 2014. (Cited in page 19).

[Schmidt *et al.*, 1992] Wouter F. Schmidt, Martin A. Kraaijveld, and Robert P. W. Duin. Feed Forward Neural Networks with Random Weights. In: *Proc. of the 11th IAPR International Conference on Pattern Recognition. Conference B: Pattern Recognition Methodology and Systems*, vol. II, pp. 1–4. August 1992. (Cited in page 31).

[Serpas *et al.*, 2013] Mitch Serpas, Yunfei Chu, and Juergen Hahn. Fault Detection Approach for Systems Involving Soft Sensors. *Journal of Loss Prevention in the Process Industries*, vol. 26, no. 3, pp. 443–452, May 2013. (Cited in page 21).

[Shakil *et al.*, 2009] M. Shakil, M. Elshafei, M. A. Habib, and F. A. Maleki. Soft sensor for NO_x and O_2 Using Dynamic Neural Networks. *Computers & Electrical Engineering*, vol. 35, no. 4, pp. 578–586, July 2009. (Cited in page 21).

[Shao *et al.*, 2014] Weiming Shao, Xuemin Tian, and Ping Wang. Local Partial Least Squares Based Online Soft Sensing Method for Multi-output Processes with Adaptive Process States Division. *Chinese Journal of Chemical Engineering*, vol. 22, no. 7, pp. 828–836, July 2014. (Cited in pages 16, 19, and 21).

[Sharma and Tambe, 2014] Suraj Sharma and Sanjeev S. Tambe. Soft-sensor Development for Biochemical Systems Using Genetic Programming. *Biochemical Engineering Journal*, vol. 85, pp. 89–100, April 2014. (Cited in page 21).

[Shen *et al.*, 2008] Shujie Shen, Gang Li, and Haiyan Song. An Assessment of Combining Tourism Demand Forecasts Over Different Time Horizons. *Journal of Travel Research*, vol. 47, no. 2, pp. 197–207, November 2008. (Cited in page 99).

[Shrestha and Solomatine, 2006] D. L. Shrestha and D. P. Solomatine. Experiments with AdaBoost.RT, An Improved Boosting Scheme for Regression. *Neural Computation*, vol. 18, no. 7, pp. 1678–1710, July 2006. (Cited in pages 93, 94, 95, 97, and 105).

[Siemens, 2015] Siemens. *INCA Sensor: Soft Sensors for Non-Measurable Quality Variables*, 2015.
`http://mall.industry.siemens.com/mall/en/WW/Catalog/Products/10017370`.
(Cited in page 175).

[Sivanandam and Deepa, 2007] S. N. Sivanandam and S. N. Deepa. *Introduction to Genetic Algorithms*. Springer, 2007. (Cited in page 68).

[Soares *et al.*, 2011] Symone Soares, Rui Araújo, Pedro Sousa, and Francisco Souza. Design and Application of Soft Sensor Using Ensemble Methods. In: *Proc. of the 16th IEEE Int. Conf. on Emerging Technologies and Factory Automation*, ETFA'11, pp. 1–8. September 5-9 2011. (Cited in page 21).

[Soares *et al.*, 2012] Symone Soares, Carlos Antunes, and Rui Araújo. A Genetic Algorithm for Designing Neural Network Ensembles. In: *Proc. of the 14th Int. Conf. on Genetic and Evolutionary Computation Conference*, GECCO'12, pp. 681–688. ACM, Philadelphia, PA, USA, July 7-11 2012. (Cited in page 41).

[Soares *et al.*, 2013] Symone Soares, Carlos Henggeler Antunes, and Rui Araújo. Comparison of a Genetic Algorithm and Simulated Annealing for Automatic Neural Network Ensemble Development. *Neurocomputing*, vol. 121, pp. 498–511, December 2013. (Cited in pages 5, 68, and 150).

[Soares and Araújo, 2015a] Symone Gomes Soares and Rui Araújo. An Adaptive Ensemble of On-line Extreme Learning Machines with Variable Forgetting Factor for Dynamic System Prediction. *Neurocomputing*, vol. X, no. Y, pp. ppp–ppp, 2015. URL `http://dx.doi.org/10.1016/j.neucom.2015.07.035`. (Accepted). (Cited in pages 7, and 21).

[Soares and Araújo, 2015b] Symone Gomes Soares and Rui Araújo. A Dynamic and On-line Ensemble Regression for Changing Environments. *Expert Systems with Applications*, vol. 42, no. 6, pp. 2935–2948, April 2015. (Cited in pages 6, and 21).

[Soares and Araújo, 2015c] Symone Gomes Soares and Rui Araújo. An On-line Weighted Ensemble of Regressor Models to Handle Concept Drifts. *Engineering Applications of Artificial Intelligence*, vol. 37, pp. 392–406, January 2015. (Cited in pages 6, 21, and 134).

[Stanišić *et al.*, 2015] Darko Stanišić, Nikola Jorgovanović, Nikola Popov, and Velimir Čongradac. Soft Sensor for Real-time Cement Fineness Estimation. *ISA Transactions*, vol. 55, no. 0, pp. 250–259, March 2015. (Cited in pages 16, and 21).

[Suen *et al.*, 2005] Yuk Lai Suen, Prem Melville, and Raymond J. Mooney. Combining Bias and Variance Reduction Techniques for Regression Trees. In: *Machine Learning: ECML 2005*, vol. 3720 of *Lecture Notes in Computer Science*, pp. 741–749. Springer, 2005. (Cited in page 28).

[Tang *et al.*, 2012] Jian Tang, Dianhui Wang, and Tianyou Chai. Predicting Mill Load Using Partial Least Squares and Extreme Learning Machines. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 16, no. 9, pp. 1585–1594, September 2012. (Cited in page 21).

[Torgo, 2011] Luís Torgo. *Regression Datasets*. Laboratory of Artificial Intelligence and Decision Support (LIAAD), University of Porto, 2011. `http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html`. (Cited in page 72).

[Torres-Sospedra *et al.*, 2005] Joaquín Torres-Sospedra, Mercedes Fernández-Redondo, and Carlos Hernández-Espinosa. A Research on Combination Methods for Ensembles of Multilayer Feedforward. In: *Proc. IEEE Int. Joint Conf. on Neural Networks*, vol. 2 of *IJCNN'05*, pp. 1125–1130. July 31 - August 04 2005. (Cited in pages 42, and 62).

[Tsymbal, 2004] Alexey Tsymbal. The Problem of Concept Drift: Definitions and Related Work. Tech. rep., The University of Dublin, Trinity College, Department of Computer Science, Dublin, Ireland, April 2004. (Cited in pages 45, and 126).

[Vergara *et al.*, 2012] Alexander Vergara, Shankar Vembu, Tuba Ayhan, Margaret A. Ryan, Margie L. Homer, and Ramón Huerta. Chemical Gas Sensor Drift Compensation Using Classifier Ensembles. *Sensors and Actuators B: Chemical*, vol. 166-167, pp. 320–329, May 2012. (Cited in page 126).

[Vergara and Estéves, 2014] Jorge R. Vergara and Pablo A. Estéves. A Review of Feature Selection Methods Based on Mutual Information. *Neural Computing and Applications*, vol. 24, no. 1, pp. 175–186, January 2014. (Cited in page 14).

[Vignolo *et al.*, 2013] Leandro D. Vignolo, Diego H. Milone, and Jacob Scharcanski. Feature Selection for Face Recognition Based on Multi-objective Evolutionary Wrappers. *Expert Systems with Applications*, vol. 40, no. 13, pp. 5077–5084, October 2013. (Cited in page 15).

[Škutová, 2008] Jolana Škutová. Weights Initialization Methods for MLP Neural Networks. *Tran. of the VŠB - Technical University of Ostrova, Mechanical Series*, vol. LIV, no. 2, pp. 147–152, 2008. (Cited in page 64).

[Wang *et al.*, 2015] Di Wang, Ping Wang, and Yan Ji. An Oscillation Bound of the Generalization Performance of Extreme Learning Machine and Corresponding Analysis. *Neurocomputing*, vol. 151, Part 2, pp. 883–890, March 2015. (No citations).

[Wang and Alhamdoosh, 2013] Dianhui Wang and Monther Alhamdoosh. Evolutionary Extreme Learning Machine Ensembles with Size Control. *Neurocomputing*, vol. 102, pp. 98–110, February 2013. (Cited in page 5).

[Wang and Guo, 2013] Jiesheng Wang and Qiuping Guo. Locally Weighted Kernel Principal Component Regression Model for Soft Sensing of Nonlinear Time-Variant Processes. *Instrumentation Science & Technology*, vol. 41, no. 1, pp. 18–36, February 2013. (Cited in pages 21, and 60).

[Willmott, 1981] Cort J. Willmott. On the Validation of Models. *Physical Geography*, vol. 2, no. 2, pp. 184–194, 1981. (Cited in page 19).

[Xu *et al.*, 2014] Ouguan Xu, Yongfeng Fu, Hongye Su, and Lijuan Li. A Selective Moving Window Partial Least Squares Method and Its Application in Process Modeling. *Chinese Journal of Chemical Engineering*, vol. 22, no. 7, pp. 799–804, July 2014. (Cited in pages 21, and 35).

[Xu *et al.*, 2011] Wei Xu, Lingbo Zhang, and Xingsheng Gu. Soft Sensor for Ammonia Concentration at the Ammonia Converter Outlet Based on an Improved Particle Swarm Optimization and BP Neural Network. *Chemical Engineering Research and Design*, vol. 89, no. 10, pp. 2102–2109, October 2011. (Cited in page 21).

[Yang and Chen, 2012] Shih-Hung Yang and Yon-Ping Chen. An Evolutionary Constructive and Pruning Algorithm for Artificial Neural Networks and its Prediction Applications. *Neurocomputing*, vol. 86, pp. 140–149, June 2012. (Cited in page 18).

[Yu-Bo and Zhi-Bin, 2011] Tian Yu-Bo and Xie Zhi-Bin. Particle-Swarm-Optimization-Based Selective Neural Network Ensemble and Its Application to

Modeling Resonant Frequency of Microstrip Antenna. In: *Microstrip Antennas*, pp. 69–82. InTech, April 2011. (Cited in pages 41, and 62).

[Yuan *et al.*, 2014] Xiaofeng Yuan, Zhiqiang Ge, and Zhihuan Song. Locally Weighted Kernel Principal Component Regression Model for Soft Sensing of Nonlinear Time-Variant Processes. *Industrial & Engineering Chemistry Research*, vol. 53, no. 35, pp. 13736–13749, August 2014. (Cited in page 21).

[Zhang, 2007] G. Peter Zhang. A Neural Network Ensemble Method with Jittered Training Data for Time Series Forecasting. *Information Sciences*, vol. 177, no. 23, pp. 5329–5346, December 2007. (Cited in pages 18, and 60).

[Zhang *et al.*, 2010] Haichuan Zhang, Zhongyang Liu, Dongwei Xu, and Ninghui Wang. Soft-Sensor of Ozone Concentration in Ozone Generation System. *Ozone: Science & Engineering*, vol. 32, no. 1, pp. 56–60, March 2010. (Cited in page 21).

[Zhang *et al.*, 2012] Yingwei Zhang, Shuai Li, and Yongdong Teng. Dynamic Processes Monitoring Using Recursive Kernel Principal Component Analysis. *Chemical Engineering Science*, vol. 72, pp. 78–86, April 2012. (Cited in page 21).

[Zhou *et al.*, 2002] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. Ensembling Neural Networks: Many Could Be Better Than All. *Artificial Intelligence*, vol. 137, no. 1-2, pp. 239–263, May 2002. Code available at `http://lamda.nju.edu.cn/files/Gasen.zip`. (Cited in pages 41, 61, and 85).

[Zliobaite, 2009] Indre Zliobaite. Learning under Concept Drift: an Overview. Tech. rep., Faculty of Mathematics and Informatics, Vilnius University, 2009. (Cited in page 44).


FCT
Fundação para a Ciência e a Tecnologia
MINISTÉRIO DA EDUCAÇÃO E CIÊNCIA


mais
CENTRO
Programa Operacional Regional do Centro


adi
agência de inovação


POPH
PROGRAMA OPERACIONAL POTENCIAL HUMANO


QREN
QUADRO
DE REFERÊNCIA
ESTRATÉGICO
NACIONAL
PORTUGAL 2007-2013


GOVERNO DA REPÚBLICA
PORTUGUESA


UNIÃO EUROPEIA
Fundo Social Europeu