HAMID REZA KHOSRAVANI

ARTIFICIAL NEURAL NETWORK MODELS: DATA SELECTION AND ONLINE ADAPTATION

UNIVERSIDADE DO ALGARVE

Faculdade de Ciências e Tecnologia

2017

HAMID REZA KHOSRAVANI

ARTIFICIAL NEURAL NETWORK MODELS: DATA SELECTION AND ONLINE ADAPTATION

Doutoramento em Engenheira Informática

(Especialidade em Inteligência Artificial)

Trabalho efeuado sob a orientação de:

António Eduardo de Barros Ruano e Pedro Miguel Frazão F. Ferreira

UNIVERSIDADE DO ALGARVE

Faculdade de Ciências e Tecnologia

2017

ARTIFICIAL NEURAL NETWORK MODELS:
DATA SELECTION AND ONLINE ADAPTATION

Hamid Reza Khosravani

To my love, Elmira

To my parents, Parvin and Mehdi

To my brother, Ehsan

Acknowledgements

Undoubtedly doing this PhD would have not been possible without the sincere support and guidance that I received from many people throughout four years studying at the University of Algarve. First of all, I would like to express my special thanks to my supervisors Prof. Antonio Ruano and Prof. Pedro Ferreira for all their precious advices and encouragements allowing me to grow as a research scientist. They were not only continuously supporting me but also gave me a chance to find myself in academic atmosphere.

I greatly appreciate the Erasmus Mundus SALAM scholarship program for kindly funding me towards this PhD. I also would like to particularly acknowledge Prof. Hamid Shahbazkia as the coordinator of SALAM scholarship program for all his supports during these years.

Many thanks also to my friend, Sergio Silva, as the CSI laboratory’s administrator for his non-stop help and support in providing all stuffs that I needed to proceed my PhD. I also would like to appreciate Prof. Eslam Nazemi who was one the most influent people in my academic life for his great guidance towards my PhD.

My deepest acknowledgement goes to my beloved, resilient and patient wife Elmira for bearing and accompanying me shoulder to shoulder in ups and downs throughout last four years.

Last but not least, I would like to express my special appreciate to my parents and my brother Ehsan for their ever support and encouragement during my life that enabled me to achieve this goal.

Abstract

Energy consumption has been increasing steadily due to globalization and industrialization. Studies have shown that buildings have the biggest proportion in energy consumption; for example in European Union countries, energy consumption in buildings represents around 40% of the total energy consumption. Hence this PhD was intended towards managing the energy consumed by Heating, Ventilating and Air Conditioning (HVAC) systems in buildings benefiting from Model Predictive Control (MPC) technique. To achieve this goal, artificial intelligence models such as neural networks and Support Vector Machines (SVM) have been proposed because of their high potential capabilities of performing accurate nonlinear mappings between inputs and outputs in real environments which are not noise-free. In this PhD, Radial Basis Function Neural Networks (RBFNN) as a promising class of Artificial Neural Networks (ANN) were considered to model a sequence of time series processes where the RBFNN models were built using Multi Objective Genetic Algorithm (MOGA) as a design platform. Regarding the design of such models, two main challenges were tackled; data selection and model adaptation.

Since RBFNNs are data driven models, the performance of such models relies, to a good extent, on selecting proper data throughout the design phase, covering the whole input-output range in which they will be employed. The convex hull algorithms can be applied as methods for data selection; however the use of conventional implementations of these methods in high dimensions, due to their high complexity, is not feasible. As the first phase of this PhD, a new randomized approximation convex hull algorithm called ApproxHull was proposed for high dimensions so that it can be used in an acceptable execution time, and with low memory requirements. Simulation results showed that applying ApproxHull as a filter data selection method (i.e., unsupervised data selection method) could improve the performance of the classification and regression models, in comparison with random data selection method. In addition, ApproxHull was employed in real applications in terms of three case studies. The first two were in association with applying predictive models for energy saving. The last case study was related to segmentation of lesion areas in brain Computed Tomography (CT) images. The evaluation results showed that applying ApproxHull in MOGA could result in models with an acceptable level of accuracy. Specifically, the results obtained from the third case study demonstrated that ApproxHull is capable of being applied on large size data sets in high dimensions. Besides the random selection method, it was also compared with an entropy based unsupervised data selection method and a hybrid method involving ApproxHull and the entropy based method. Based on the simulation results, for most cases, ApproxHull and the hybrid method achieved a better performance than the others.

In the second phase of this PhD, a new convex-hull-based sliding window online adaptation method was proposed. The goal was to update the offline predictive RBFNN models used in HVAC MPC technique, where these models are applied to processes in which the data input-output range changes over time. The idea behind the proposed method is capturing a new arriving point at each time instant which reflects a new range of data by comparing the point with current convex hull presented via ApproxHull. In this situation the underlying model's parameters are updated based on the new point and a sliding window of some past points. The simulation results showed that not only the proposed method could efficiently update the model while a good level of accuracy is kept but also it was comparable with other methods.

Keywords: Neural Networks; Multi-Objective Genetic Algorithm; Data Selection; Online Adaptation.

Resumo

Devido aos processos de industrialização e globalização o consumo de energia tem aumentado de forma contínua. A investigação sobre o consumo mostra que os edifícios consomem a maior fatia de energia. Por exemplo nos países da União Europeia essa fatia corresponde a cerca de 40% de toda a energia consumida. Assim, esta tese de Doutoramento tem um objetivo prático de contribuir para melhorar a gestão da energia consumida por sistemas *Heating, Ventilating and Air Conditioning (HVAC)* em edifícios, no âmbito de uma estratégia de controlo preditivo baseado em modelos. Neste contexto foram já propostos modelos baseados em redes neuronais artificiais e máquinas de vetores de suporte, para mencionar apenas alguns. Estas técnicas têm uma grande capacidade de modelar relações não-lineares entre entradas e saídas de sistemas, e são aplicáveis em ambientes de operação, que, como sabemos, estão sujeitos a várias formas de ruído. Nesta tese foram consideradas redes neuronais de função de base radial, uma técnica consolidada no contexto da modelação de séries temporais. Para desenhar essas redes foi utilizada uma ferramenta baseada num algoritmo genético multi-objectivo. Relativamente ao processo de desenho destes modelos, esta tese versa sobre dois aspetos menos estudados: a seleção de dados e a adaptação em linha dos modelos.

Uma vez que as redes neuronais artificiais são modelos baseados em dados, a sua performance depende em boa medida da existência de dados apropriados e representativos do sistema/processo, que cubram toda a gama de valores que a representação entrada/saída do processo/sistema gera. Os algoritmos que determinam a figura geométrica que envolve todos os dados, denominados algoritmos *convex hull*, podem ser aplicados à tarefa de seleção de dados. Contudo a utilização das implementações convencionais destes algoritmos em problemas de grane dimensionalidade não é viável do ponto de vista prático. Numa primeira fase deste trabalho foi proposto um novo método randomizado de aproximação ao *convex hull*, cunhado com o nome ApproxHull, apropriado para conjuntos de dados de grande dimensão, de forma a ser viável do ponto de vista das aplicações práticas. Os resultados experimentais mostraram que a aplicação do ApproxHull como método de seleção de dados do tipo filtro, ou seja, não supervisionado, pode melhorar o desempenho de modelos em problemas de classificação e regressão, quando comparado com a seleção aleatória de dados. O ApproxHull foi também aplicado em três casos de estudo relativos a aplicações reais. Nos dois primeiros casos no contexto do desenvolvimento de modelos preditivos para sistemas na área da eficiência energética. O terceiro caso de estudo consiste no desenvolvimento de modelos de classificação para uma aplicação na área da segmentação de lesões em imagens de tomografia computorizada. Os resultados revelaram que da aplicação do método proposto resultaram modelos com uma precisão aceitável. Do ponto de vista da aplicabilidade do método, os resultados mostraram que o ApproxHull pode ser utilizado em conjuntos de dados grandes e com dados de grande dimensionalidade. Para além da comparação com a seleção aleatória de dados, o método foi também comparado com um método de seleção de dados baseado no conceito de entropia e com um método híbrido que resulta da combinação do ApproxHull com o método entrópico. Com base nos resultados experimentais apurou-se que na maioria dos casos estudados o método híbrido conseguiu melhor desempenho que os restantes.

Numa segunda fase do trabalho foi proposto um novo método de adaptação em linha com base no algoritmo ApproxHull e numa janela deslizante no tempo. Uma vez que os processos e sistemas na envolvente do sistema HVAC são variantes no tempo e dinâmicos, o objetivo foi aplicar o método proposto para adaptar em linha os modelos que foram primeiramente obtidos fora de linha. A ideia base do método proposto consiste em comparar cada novo par entrada/saída com o *convex hull* conhecido, e determinar se o novo par tem dados situados fora da gama conhecida. Nessa situação os parâmetros dos modelos são atualizados com base nesse novo ponto e num conjunto de pontos numa determinada janela temporal deslizante. Os resultados experimentais demonstraram não só que o novo método é eficiente na atualização dos modelos e em mantê-los num bom nível de precisão, mas também que era comparável a outros métodos existentes.

Palavras-chave: Redes Neuronais; Algoritmo Genético Multi-Objectivo; Seleção de dados; Adaptação on-line.

Contents

List of Tables

List of Figures

List of Algorithms

List of Acronyms

ANN	Artificial Neural Network
ARIMA	Auto Regressive Integrated Moving Average
ARMA	Auto Regressive Moving Average
ASMOD	Adaptive Spline Modeling of Observation Data
BP	Back Propagation
BSE	Backward Sequential Edition
CDA	Conditional Demand Analysis
CIEMAT	Centre for Energy, Environment and Technology - Centro de Investigaciones Energéticas, MedioAmbientales y Tecnológicas (in Spanish)
CIESOL	Solar Energy Research Center - Centro de Investigación en Energía SOLar (in Spanish)
CNN	Condensed Nearest Neighbor rule
CR	Classification Rate
CT	Computed Tomographic
CVA	Cerebral Vascular Accident
DROP	Decremental Reduction Optimization Procedure
EKF	Extended Kalman Filter
ELM	Extreme Learning Machine
ENN	Edited Nearest Neighbor rule
FIFO	First In First Out
FN	False Negative
FNN	Feedforward Neural Network
FP	False Positive
GA	Genetic Algorithm
GCNN	Generalized Condensed Nearest Neighbor rule
GGA	Generalized Genetic Algorithm
GMCA	Generalized Modified Change Algorithm
GMM	Gaussian Mixture Models
HVAC	Heating, Ventilating and Air Conditioning
HVDM	Heterogeneous Value Difference Metric
IAH	Inside Air Humidity
IAT	Inside Air Temperature

ICF	Iterative Case Filtering
IGA	Intelligent Genetic Algorithm
IMBPC	Intelligent Model Based Predictive Control
k-NN	K nearest neighbors
LMS	Least Mean Square
LS	Least Square
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MaxAE	Maximum Absolute Error
MI	Mutual Information
MLP	Multi-Layer Perceptron
MNV	Mutual Nearest Neighborhood
MOGA	Multi Objective Genetic Algorithm
MOV	Movement Signal
MPC	Model Predictive Control
MRE	Mean Relative Error
MRLS	Multi-innovative Recursive Least Square
NAB	Naive Autoregressive Baseline
NAR	Non-linear AutoRegressive
NARX	Non-linear AutoRegressive with eXogenous inputs
NEN	Nearest-Neighbor algorithm
NSB	Nearest Subclass Classifier
OAT	Outside Air Temperature
OAKM	Optimal Adaptive K-Means
OLS	Orthogonal Least Square
OSC	Object Selection by Clustering
OSR	Outside Solar Radiation
PBLT	Population-Based Incremental Learning
PCA	Principal Components Analysis
PDF	Probability Density Function
PMV	Predicted Mean Vote
POP	Pattern by Ordered Projections
PSR	Prototype Selection by Relevance

QPSO	Quantum Particle Swarm Optimization
RAN	Resource Allocation Network
RBFNN	Radial Basis Function Neural Network
RFOS	Restricted Floating Object Selection
RMSE	Root Mean Square Error
SCADA	Supervisory Control And Data Acquisition
SGBP	Stochastic Gradient Descent Back Propagation
SNN	Selective Nearest Neighbor rule
SSGA	Steady-State Genetic Algorithm
STP	Set Temperature Point
SUS	Stochastic Universal Sampling
TN	True Negative
TP	True Positive
WLS	Weighted Least Square

1. Introduction

1.1. Motivation

In European Union countries, primary energy consumption in buildings represents about 40% of the total energy consumption, and, with variations from country to country, half of this energy is spent for indoor climate conditioning. Advanced techniques for the control of HVAC systems, and in particular, Model Predictive Control (MPC), offer enormous potential for huge savings in building energy consumption. Regarding the application of MPC technique in energy sectors, authors in [1] have shown in simulations that savings of energy in the order of 30% could be obtained, while maintaining a convenient temperature regulation. In a more recent work [2, 3], authors were able to control, in real-time, the air conditioning systems of several rooms in a building. Average savings of 50% were obtained, in summer and winter conditions, while maintaining thermal comfort. The MPC technique uses several ANN models. There are already some available state-of-the-art tools for designing neural network models [4], in terms of input selection, model structure determination and parameter estimation. On the other hand, as ANNs are data driven models, the data used to design the models has a direct influence on the models' performance and, ultimately, on the performance of the HVAC MPC technique. It is very important that the data which is selected to design ANN models involve the boundary samples; those samples that reflect the whole input-output range in which the underlying process is modeled. To catch such samples out of the whole data set, convex hull algorithms [5-12] as one of the fundamental concepts in computational geometry, can be applied (i.e., please refer to Chapter 3).

The standard convex hull algorithms suffer from both time and space in high dimensions (i.e., more than three dimensions). They take $O(n^{\lfloor \frac{d}{2} \rfloor})$ time and space in the worst case where n and d denote the number of samples and the dimension, respectively. Hence this problem prevents us from employing convex hull algorithms in real applications where we are interested in applying them to data sets containing larger numbers of samples and higher dimensions. The first phase of this PhD proposes a state-of-the-art approach to tackle the challenges of standard convex hull algorithms in high dimensions.

Regarding the HVAC MPC technique, the processes involved are time-varying and dynamic. For instance, models designed with winter data will have their performance degraded on summer data. Furthermore, the temperature in a room depends on the occupancy and on equipment being used in the room. Whether the variation has seasonal and/or dynamic origins, it causes changes in the input-output range of the data applied to the models. To deal with these problems, an online adaptation method is required to update the model parameters. The adaptation method should be capable of capturing the new input-output ranges presented by the newly arrived samples throughout time. Based on that newly selected data, the models should become efficiently updated and appropriate for a real time application. The second phase of this PhD is one step towards proposing a state-of-the-art online adaptation method.

1.2. Main contributions

As the first phase of this PhD, a new randomized approximation convex hull algorithm in high dimensions coined *ApproxHull* was proposed. The performance of ApproxHull as a filter type data selection method was evaluated for a number of classification and regression problems. This was done by comparing ApproxHull with other data selection methods including random selection, an entropy based unsupervised method [13] and a hybrid method involving ApproxHull and the entropy based method. Since the main goal of this PhD was employing ApproxHull to construct proper data sets for designing ANN models for the HVAC MPC technique, it was exploited in two related case studies.

In the first case study [14, 15], ApproxHull was used to provide data sets for designing several time series predictive RBFNN models to forecast the one-step-ahead measures of outside climate variables, including outside air temperature, outside relative humidity and outside solar radiation, so that all models were integrated in an intelligent weather station. Additionally, another series of RBFNN models were also designed to predict the one-step-ahead of inside climate variables which have a significant role in the HVAC MPC technique. In order to design these models, the Multi-Objective Genetic Algorithm was applied to select model structures optimized for the specific task.

The second case study resulted from a collaboration between the University of Algarve in Portugal and the University of Almeria in Spain. It was aimed at the development of time series predictive RBFNN models to forecast the one-step-ahead value of the electricity power demand for a building inside the campus of the University of Almeria. As for the previous case study, ApproxHull was applied to supply data sets for the MOGA model design framework. In this case study, a selected MOGA generated model was compared to a RBFNN model designed by means of statistical and analytical tools.

ApproxHull was also exploited by other researchers in the field of biomedical image processing where the goal was presenting an RBFNN based diagnosis system for automatic identification of Cerebral Vascular Accident (CVA) through analysis of Computed Tomographic images. In this case study, ApproxHull was applied on a large size data set in high dimensions (i.e., 1,867,602 samples with 52 features). The simulation results obtained from the case studies showed that not only the proposed method is capable of being applied in real applications but also its performance is comparable with other data selection methods.

In the second phase of this PhD a new convex-hull-based sliding window online adaptation method was proposed. Since at a given time instant the current convex hull reflects the whole input-output range of all observed data, this method enables the comparison of newly arrived samples to the current convex hull to find out whether it presents data outside the known range. In case it does, the current convex hull is updated and consequently the training sliding window is also updated. Afterwards, the model is adjusted based on the modified sliding window. To verify the proposed online adaptation method, a time series predictive RBFNN model for outside air temperature was considered. The simulation results demonstrated that not only the model could be efficiently updated, but it could also preserve relevant input-output pairs that had been presented over time.

As a result of the efforts carried out in this PhD, the following articles were published.

1. RUANO, A. E., MADUREIRA, G., BARROS, O., KHOSRAVANI, H. R., RUANO, M. G. & FERREIRA, P. M. 2014. Seismic detection using support vector machines. *Neurocomputing,* 135**,** 273-283.

2. MESTRE, G., RUANO, A., DUARTE, H., SILVA, S., KHOSRAVANI, H., PESTEH, S., FERREIRA, P. M. & HORTA, R. 2015. An Intelligent Weather Station. *Sensors,* 15**,** 31005–31022.

3. RUANO, A., PESTEH, S., SILVA, S., DUARTE, H., MESTRE, G., FERREIRA, P. M., KHOSRAVANI, H. & HORTA, R. 2015. The IMBPC HVAC system: a complete MBPC solution for existing HVAC systems. *Energy and Buildings*,120, pp- 145-158

4. KHOSRAVANI, H., CASTILLA, M., BERENGUEL, M., RUANO, A. & FERREIRA, P. M. 2016. A Comparison of Energy Consumption Prediction Models Based on Neural Networks of a Bioclimatic Building. *Energies,* 9**,** 57.

5. KHOSRAVANI, H., RUANO, A. & FERREIRA, P. M. 2016. A Convex Hull-based Data Selection Method for Data Driven Models. *Applied Soft Computing*, 47, pp. 515-533

6. RUANO, A. E., MADUREIRA, G., BARROS, O., KHOSRAVANI, H. R., RUANO, M. G. & FERREIRA, P. M. A Support Vector Machine Seismic Detector for Early-Warning Applications. *In:* FERREIRA, P. M., ed. Intelligent Control and Automation Science (ICONS 2013), 2-4 Sept 2013 Chengdu, China. IFAC, 400-405.

7. KHOSRAVANI, H. R., RUANO, A. E. & FERREIRA, P. M. A Simple Algorithm for Convex Hull Determination in High Dimensions. 8th IEEE International Symposium on Intelligent Signal Processing (WISP 2013), Sep, 16-18, 2013 Funchal, Madeira, Portugal. 109 - 114.

8. RUANO, A., KHOSRAVANI, H. R. & FERREIRA, P. M. 2015. A Randomized Approximation Convex Hull Algorithm for High Dimensions. *IFAC-PapersOnLine,* 48**,** 123-128.

9. RUANO, A. E., MESTRE, G., DUARTE, H., SILVA, S., PESTEH, S., KHOSRAVANI, H. R., FERREIRA, P. M. & HORTA, R. A Neural-Network based Intelligent Weather Station. 9th IEEE International Symposium on Intelligent Signal Processing (WISP 2015), 15-17 May 2015 Siena, Italy. 96-101.

10. RUANO, A. E., SILVA, S., PESTEH, S., FERREIRA, P. M., DUARTE, H., MESTRE, G., KHOSRAVANI, H. R. & HORTA, R. Improving a neural networks based HVAC predictive control approach. 9th IEEE International Symposium on Intelligent Signal Processing (WISP 2015), 15-17 May 2015 Siena, Italy. 90-95.

11. RUANO, A., PESTEH, S., SILVA, S., DUARTE, H., MESTRE, G., FERREIRA, P. M., KHOSRAVANI, H. & HORTA, R. PVM-based intelligent predictive control of HVAC systems. 4th IFAC International Conference on Intelligent Control and Automation Sciences (ICONS 2016) 1-3 Jun 2016 Reims, France. IFAC.

1.3. Thesis structure

This dissertation is organized in 9 chapters. Chapter 2 addresses the introduction of the theoretical concepts which were relevant for this PhD. In this chapter, two well-known data-driven models; ANNs and SVMs as well as a number of renowned standard learning algorithms for ANNs are explained. Moreover, the GA, the MOGA, information theory concepts and two statistical tests are presented in this chapter. Chapter 3 introduces a number of standard convex hull algorithms in low and high dimensions. This chapter also discusses an effort done in recent years to deal with the very high time and space complexity of standard convex hull algorithms in high dimensions. Chapter 4 addresses a review of instance selection methods and introduces ApproxHull as the-state-of-the-art in the convex-hull-based data selection domain. In addition, a number of experiments to verify and evaluate the performance of the ApproxHull regarding time and memory requirements are presented and analyzed in Chapter 4. Since, in this PhD, most of the models were designed by the MOGA, Chapter 5 addresses the evaluation of ApproxHull's performance within the MOGA model design framework. To verify and evaluate ApproxHull's performance in real applications, Chapter 6 introduces three case studies in which ApproxHull has been applied to design RBFNN models. To further analyze the ApproxHull's performance, Chapter 7 compares it with three data selection methods, including random selection method, an entropy based unsupervised method and a hybrid method involving ApproxHull and the entropy based method. All methods were applied for classification and regression. In the second phase of the PhD, Chapter 8 addresses a brief overview on online adaptation method and then introduces a new convex-hull-based, sliding-window online adaptation method. Furthermore, to verify and evaluate the performance of the proposed method, several online adaptation experiments associated with two case studies along with the corresponding results and comparisons with other methods are explained in this chapter. Finally, a brief conclusion of all efforts done in this PhD as well as some future work directions are discussed in Chapter 9.

2. Theoretical background

2.1. Introduction

Since this PhD thesis aims to propose new data selection and online model adaptation methods for data driven models, in this chapter two well-known classes of data driven models, ANNs and SVMs, are introduced. Additionally, some basic concepts related to the design of data driven models are explained. This chapter is organized as follows. Section 2.2 addresses the introduction of several types of ANNs. Section 2.3 introduces the SVM as another class of data driven models applied to classification problems. Well-known ANNs supervised learning methods are discussed in Section 2.4. The most common criteria used to evaluate the performance of regression and classification problems are introduced in Section 2.5. As in this study a Genetic Algorithm (GA) was used in the proposed data selection method, and most ANN models were designed by means of the MOGA, Section 2.6 explains the basic concepts of GA and MOGA. Section 2.7 discusses the application of MOGA to the design of ANN models.

As the proposed data selection method was compared with an entropy based unsupervised method proposed in [13], two key concepts of information theory, entropy and mutual information, are explained in Section 2.8. Additionally, in this PhD thesis, two statistical tests were employed to compare the performance of the proposed data selection method with other methods. Section 2.9 describes these two statistical tests.

2.2. Artificial Neural Networks

The fundamental concepts of ANNs were introduced in the 1940s by McCulloch and Pitts [16] with the aim of simulating brain behavior in information processing and computations, where each part of the brain, responsible to perform a particular task, consists of a network with a huge number of neurons as processing units. Since then, ANNs have been used in a wide variety of applications such as image processing, pattern recognition, signal processing, modeling and time series, to mention a few [17]. ANNs are mainly divided into two groups [18]: 1- feed-forward networks 2- recurrent/feedback networks. Each of these two groups, in turn, is divided into several subgroups. Fig. 2.1 shows a brief classification of ANNs.

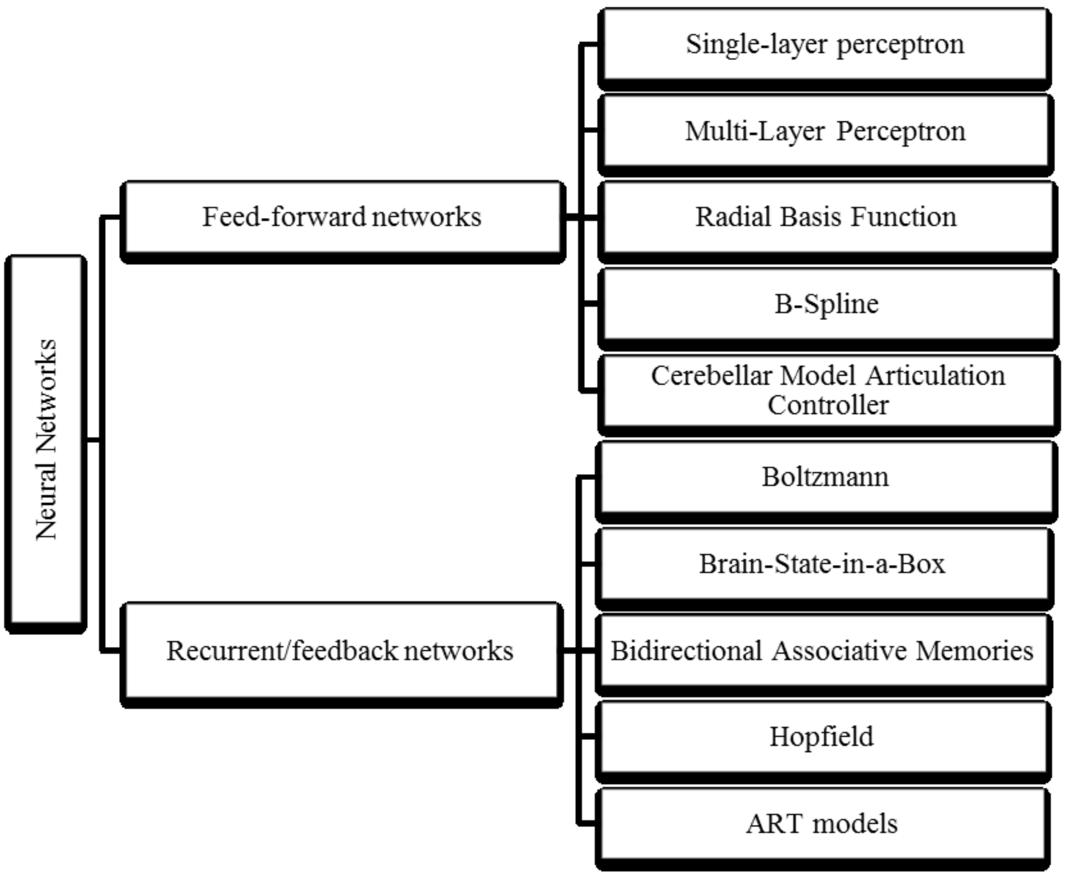


Fig. 2.1. A classification of ANNs [18].

Since in this PhD, specifically, the Multi-Layer Perceptron, Radial Basis Function and B-Spline networks were employed, the following subsections address only these types of networks.

2.2.1. Multi-Layer Perceptron Neural Network

The Multi-Layer Perceptron (MLP), as one of the renowned feed-forward networks, has been extensively applied to classification and regression problems over years. The fully connected structure of the MLP is organized in three types of layers including input, hidden and output layers. Each layer has a number of neurons and each neuron in a layer is connected to all neurons of its predecessor layer via weighted links. Fig. 2.2 illustrates an MLP network with two hidden layers. As it can be seen, each input signal which can be translated into an input variable or a feature is linked to a particular neuron in the input layer while each neuron in the output layer corresponds to a specific output signal/variable. When an input signal is fed into the input layer, several mappings are done through hidden neurons with smooth, nonlinear activation functions to produce the corresponding output signal. Generally speaking, the MLP is a black box function approximator that reflects a nonlinear relationship between input and output signals.

Bounded functions such as sigmoid or hyperbolic tangent are mostly used as the activation functions of the hidden neurons. Eq. (2.1) shows a sigmoid function.

$$\varphi_i^l(\mathbf{w}^l, \mathbf{x}) = \frac{1}{1 + e^{-\left(b_i^l + \sum_{j=1}^{n_{l-1}} w_{i,j}^l \varphi_j^{l-1}(\mathbf{w}^{l-1}, \mathbf{x})\right)}} \tag{2.1}$$

where φ_i^l is the output of the ith neuron at hidden layer l . n_{l-1} and b_i^l denote the number of neurons in hidden layer $l-1$ and the bias of hidden layer l, respectively. $w_{i,j}^l$ refers to the weight connecting the jth neuron in hidden layer $l-1$ to the ith neuron in hidden layer l.

The outputs of the MLP network are obtained by Eq. (2.2) which is a linear combination of the activation functions of the last hidden layer.

$$y_o = b_o^L + \sum_{k=1}^{n_L} w_{o,k}^L \varphi_k^L \tag{2.2}$$

where y_o and L denote the oth output neuron and the number of hidden layers, respectively. In the design process, the structure of the MLP network, which is specified by the number of layers and the number of neurons in each layer, should be determined. This structure determination should be done in such a way that the overfitting phenomenon is avoided. Overfitting refers to a situation where the number of neurons, and consequently the number of parameters, is larger than needed [19].

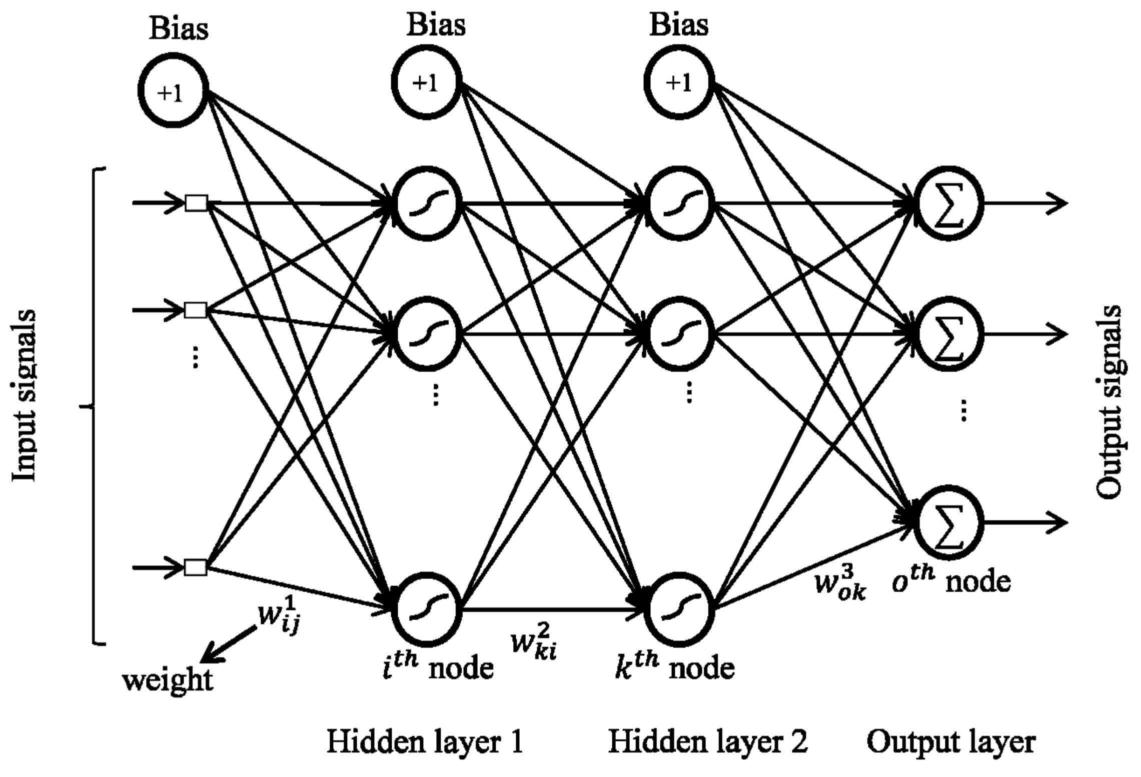


Fig. 2.2. An MLP network with two hidden layers.

2.2.2. Radial Basis Function Neural Network

The Radial Basis Function (RBF) Neural Network (NN) was firstly proposed by Broomhead and Lowe [20] . It is another type of feed-forward neural network that has received much attention due to its universal approximation and robustness to outlier points. From the structural point of view, RBFNNs, like MLPs, have three types of layers including input, hidden and output layers. Fig. 2.3 illustrates a RBFNN with three layers. As it can be seen in Fig. 2.3, each feature of the input pattern is connected to a node of input layer via a link without weight. The input neurons are only simple sensory nodes passing the input pattern without any changes toward the hidden layer. To each hidden node, a radial basis function is assigned implementing a nonlinear relation between the input and the output spaces. For the most cases, Gaussian radial basis function, thin-plate spline, multiquadrics and inverse multiquadrics are used as the activation function for the hidden neurons as Eq. (2.3).

$$\varphi_i(\mathbf{x}, \mathbf{c}_i, \sigma_i) = e^{-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{2\sigma_i^2}} \tag{2.3}$$

where φ_i denotes the activation function of the i^{th} hidden neuron. $\boldsymbol{c}_i$ and σ_i refer to the corresponding nonlinear parameters, the center and spread of φ_i, respectively. The φ_i are localized functions around each $\boldsymbol{c}_i$, whose localization degree is defined by σ_i [21]. $\mathbf{x}$ is the input pattern.

The output of the RBFNN is obtained by Eq. (2.4) which is a linear combination of the outputs of the hidden layer.

$$y(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i \varphi_i(\mathbf{x}) \tag{2.4}$$

where n is the number of hidden neurons and w_i denotes the corresponding weight of i^{th} hidden neuron. w_0 refers to the bias.

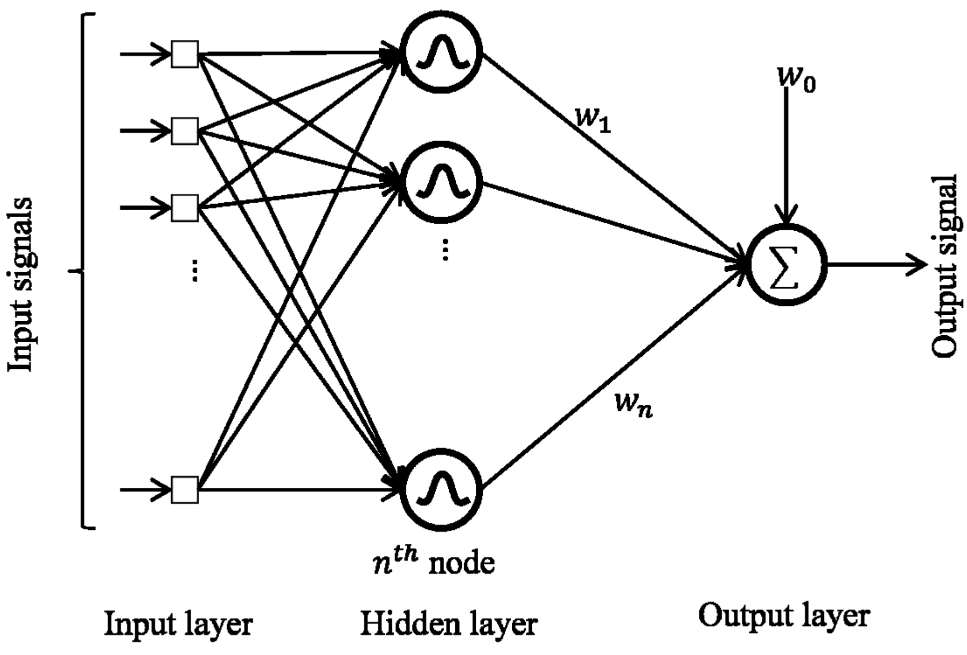


Fig. 2.3. A RBFNN structure.

2.2.3. B-Spline Network

B-spline neural networks belong to the class of networks denoted as lattice-based associative memory networks [22, 23]. In this type of networks, the basis functions are polynomial functions with a predefined order k. The range of each input variable is divided into n_i intervals and throughout the intervals, there are exactly k active functions. The n_i intervals are formed by defining r_i internal knots over the input range as well as by defining the nonlinear parameters of the network [24]. The j^{th} interval of the i^{th} input variable is defined as Eq. (2.5).

$$I_{i,j} = \begin{cases} [\lambda_{i,j-1}, \lambda_{i,j}) & for\, j = 1, \cdots r_i \\ [\lambda_{i,j-1}, \lambda_{i,j}] & if \quad j = r_i + 1 \end{cases} \tag{2.5}$$

where $\lambda_{i,j}$ denotes the j^{th} knot of the i^{th} input variable. According to Eq. (2.5), $\lambda_{i,0}$ and λ_{i,r_i+1} denote the minimum and the maximum value of the input range, respectively. By dividing the range of each input variable into several intervals, the input space is organized into a lattice where for each cell, there exist exactly $\sum_{i=1}^{d} k_i$ active functions where d denotes the input dimension. In case $d = 1$, the j^{th} univariate basis function of order k, denoted by Ψ_k^j, is defined in a recursive manner as Eq. (2.6).

$$\Psi_k^j(x) = \left(\frac{x - \lambda_{j-k}}{\lambda_{j-1} - \lambda_{j-k}}\right)\Psi_{k-1}^{j-1}(x) + \left(\frac{\lambda_j - x}{\lambda_j - \lambda_{j-k+1}}\right)\Psi_{k-1}^{j}(x) \tag{2.6}$$

$$\Psi_1^j(x) = \begin{cases} 1 & if\ x \in I_j, \\ 0 & otherwise \end{cases}$$

In the case of multidimensional input space, the multidimensional basis functions Ψ_k^j are obtained by applying a tensorial product of univariate basis functions defined over input variables as Eq. (2.7).

$$\Psi_k^j(\boldsymbol{x}) = \prod_{i=1}^{d} \Psi_{k_i}^j(x_i) \tag{2.7}$$

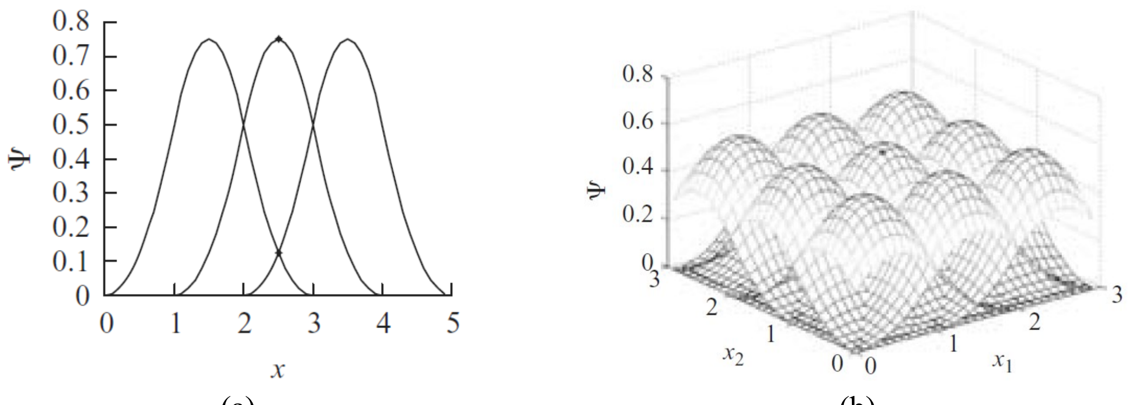
Fig. 2.4 shows an example of two B-Spline quadratic ($k = 3$) functions for one and two dimensional input spaces.

Regarding B-Spline networks, the complexity increases exponentially with respect to the dimension (i.e., the number of variables). In one hand, in order to overcome the complexity, and on the other hand to make a model with a high level of generalization, the ASMOD (Adaptive Spline Modeling of Observation data) algorithm [25, 26] is an efficient design technique to generate parsimonious models using observed data. In this way, a set of low dimensional B-Spline sub-models are generated instead of one high dimensional B-Spline model. Each sub-model depends on a subset of all the input variables. The final model output is a linear combination of sub-models' output, as given by Eq. (2.8),

$$o(x) = \sum_{i=1}^{M} c_i b_i(x) \tag{2.8}$$

where M is the number of sub-models. b_i and c_i denote the i^{th} B-Spline sub-model and its corresponding coefficient, respectively.

The ASMOD algorithm consists of two main steps including refining and pruning. It starts with a simple model with a small number of input variables. In the refining step, it tries to make the simple model complicated by adding and coupling more input variables, by changing the internal structure and by increasing the degree of the basis functions. In the pruning step, the variables are decoupled which results in a number of subsets of input variables. The internal structure is also simplified and the degree of the basis functions is decreased. These two steps are repeated until some termination criteria are met.



(a) (b)

Fig. 2.4. B-Spline quadratic ($k = 3$) functions for (a) one and (b) two dimensional feature spaces [24].

2.3. Support Vector Machines

The Support Vector Machine was proposed by Vapnik and *et al* [27, 28] as an efficient powerful machine learning method for two class classification problems where data are nonlinearly separable with respect to the target feature. The main idea behind SVMs is transferring the original input feature space to a higher dimensional feature space in which data are linearly separable and then finding the optimal hyperplane separating the two classes. The optimal hyperplane has the largest distance to the closest training samples of each class. This distance is so called the maximal margin and the samples located on the margin are marked as support vectors. Fig. 2.5 shows an example of an optimal hyperplane and associated support vectors. As it can be seen, the two classes of samples are separated from each other (i.e., the samples of the first class are shown by unfilled circles and the samples of the second class are shown by black filled circles) by the optimal hyperplane. The samples of both classes surrounded by a red circle are the support vectors.

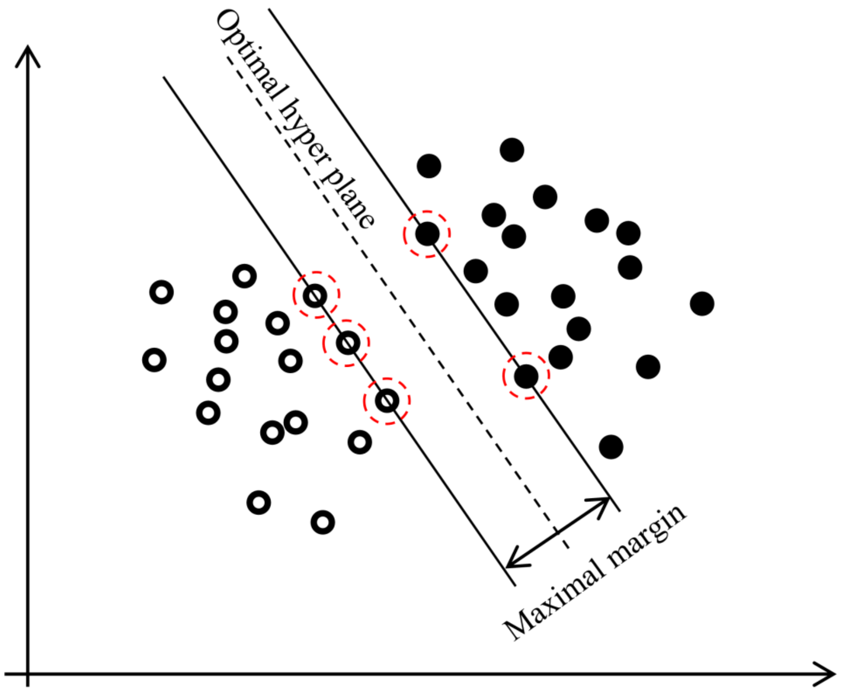


Fig. 2.5. An example of SVM [28].

The determination of the optimal hyperplane is translated into solving a constrained quadratic optimization problem where the Lagrangian stated in Eq. (2.9) should be maximized with respect to α_i subject to the constraints given in Eq. (2.10),

$$L = \sum_{i=1}^{N} \alpha_i - \sum_{i,j=1}^{N} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{2.9}$$

$$\sum_{i=1}^{N} \alpha_i \, y_i = 0 \quad and \quad 0 \leq \alpha_i \leq C \tag{2.10}$$

where N and α_i denote the number of samples of the training set and the corresponding Lagrange multiplier of the ith sample, respectively. $\mathbf{x}_i$ and $y_i \in \{-1, +1\}$ refer to the ith input pattern and the corresponding target, respectively. The target value indicates the class of the input pattern. $K(\mathbf{x}_i, \mathbf{x}_j)$ is the inner-product kernel stated in Eq. (2.11) and C a user specified parameter that establishes a trade-off between the SVM complexity and the number of non-separable patterns, often called the regularization parameter.

$$K(\mathbf{x}_i, \mathbf{x}_j) = \sum_{z=1}^{m} \phi_z(\mathbf{x}_i)\phi_z(\mathbf{x}_j) \tag{2.11}$$

In Eq. (2.11), m is the dimension of the destination feature space (i.e., the higher dimensional feature space) and $\phi_z(\mathbf{x}_i)$ is equal to the z^{th} dimension of the transformed sample $\mathbf{x}_i$ in the destination feature space. Common kernels used in SVMs are given in Eqs. (2.12) to (2.14).

- Homogeneous polynomial $$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i . \mathbf{x}_j)^d \tag{2.12}$$

- Inhomogeneous polynomial $$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i . \mathbf{x}_j + \mathbf{1})^d \tag{2.13}$$

- Gaussian radial basis function $$K(\mathbf{x}_i, \mathbf{x}_j) = exp(-\frac{\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2}{2\sigma^2}) \tag{2.14}$$

The output of a SVM model can be obtained by Eq. (2.15).

$$f(\mathbf{x}) = sign\left(\sum_{i \in SV} y_i \alpha_i^* K(\mathbf{x}, \mathbf{x}_i) - \theta\right) \tag{2.15}$$

where α_i^*s are the solution of the constrained optimization problem stated in Eq. (2.9). SV are the indices of the support vectors in the training set and θ is a user-defined threshold [29].

2.4. Learning methods

In order to fit the parameters of any data driven classification or regression model, learning methods are applied to adjust the model parameters by means of a data set of training samples. Learning methods can be considered from three points of view [18].

In the first aspect, they are categorized into four classes including supervised, unsupervised, combination of supervised and unsupervised, and reinforcement methods. In supervised learning methods, data samples are labeled so that each input pattern corresponds to a target value. In this case the parameters of the model are adjusted based on the comparison between the model outputs and corresponding target values. On the other hand, in unsupervised learning methods, unlabeled data samples are employed. In this case the methods try to group data samples into a number of clusters so that those samples which are more similar to each other are located in the same cluster. In some cases, in order to improve the performance of models, a combination of supervised and unsupervised learning methods is applied to fit the parameters. Finally, reinforcement learning methods are another class of learning methods that try to learn from their consecutive actions and from the interaction with the operational environment.

From a second point of view, learning methods can be categorized into offline or online. In the offline case a set of samples are first collected from the process or system and then processed at the same time by the learning method. In the online case the model is inserted (or simulated) in the operational environment and every time a new pattern is generated, the model parameters are possibly readjusted.

In the third aspect, learning methods can be classified into deterministic or stochastic. If the method follows a specific path to update the parameters, it is considered deterministic. Otherwise it is stochastic in the sense that it follows a randomized behavior to fit the parameters.

Fig. 2.6 briefly illustrates a taxonomy of learning methods. In the following subsections supervised and unsupervised learning methods are described.

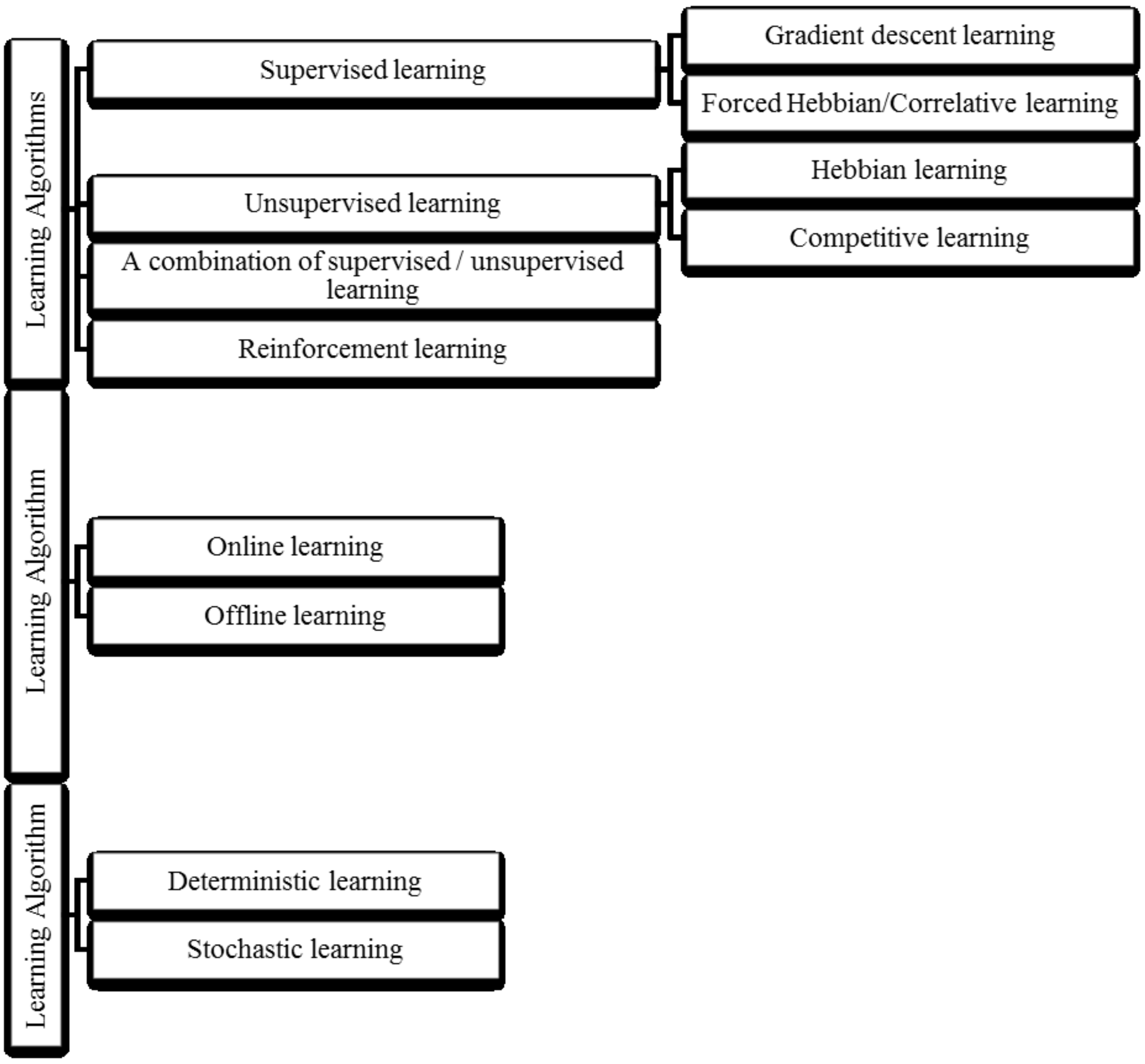


Fig. 2.6. Classification of learning methods [18].

2.4.1. Supervised learning methods

Supervised learning methods adjust the model parameters based on the labeled samples, where each input pattern along with its corresponding target value is presented to the learning method. Then the model parameters are adjusted on the basis of the comparison between the model outputs and corresponding target values. Actually, the goal of supervised learning methods is fitting the parameters so that the error obtained from the comparisons is globally minimized. The following discussion introduces a number of well-known supervised learning methods.

2.4.1.1. Steepest decent method

One of the simplest and the most common gradient based methods is the steepest descent. This method is applied to solve unconstrained optimization problems. Given a cost function $\Omega(w_0, w_1, \dots, w_n)$ where the w_is are linear or nonlinear parameters, the steepest decent method tries to obtain optimal values for w_i that minimize Ω, based on its gradient with respect to each w_i. At the first step the parameters are initialized and then updated in a recursive way over a number of iterations. Eq. (2.16) shows the update of parameters by the steepest decent method,

$$w_k^{(t+1)} = w_k^{(t)} - \alpha \frac{\partial}{\partial w_k^{(t)}} \Omega\left(w_0^{(t)}, w_1^{(t)}, \dots, w_n^{(t)}\right), \quad k = 0,1,2,\dots,n \tag{2.16}$$

where $w_k^{(t)}$ denotes the value of kth parameter in the tth iteration. α denotes the learning rate, indicating the step size that the steepest decent method takes in the direction of a local minima of the cost function Ω. The disadvantage of this method is its likelihood of being trapped into a local minimum instead of obtaining the global minimum. Fig. 2.7, shows an example of minimizing a cost function with two parameters w_0 and w_1. As it can be seen in Fig. 2.7, the minimum value of the cost function that is found by the method depends on the initial point in parameter space. By starting from point a, after a number of iterations the local minimum is achieved at b, while by starting from point c, another local minimum is reached at d.

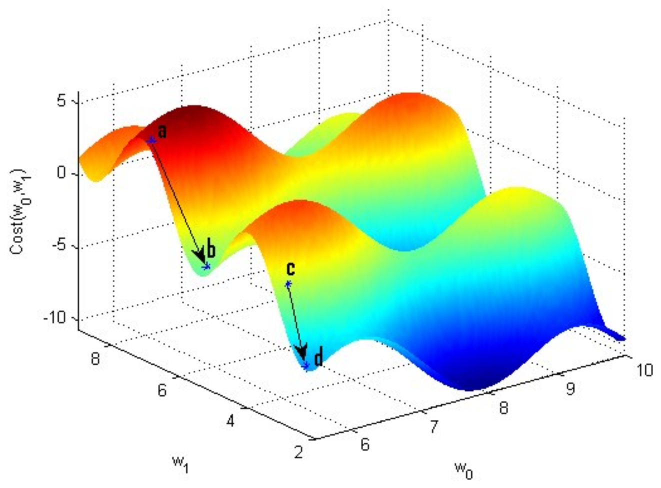


Fig. 2.7. An Example of parameters update by steepest decent method.

2.4.1.1.1. Back propagation technique

In order to apply the steepest descent method in MLP neural networks, the Back Propagation (BP) algorithm is employed. The cost function used in BP is given in Eq. (2.17),

$$\Omega(\mathbf{w}) = \frac{1}{2\mathrm{N}} \times \sum_{i=1}^{N} (\mathrm{y}(\mathbf{x}_i, \mathbf{w}) - t_i)^2 \tag{2.17}$$

where N is the number of samples in the training set. $\mathbf{w}$ denotes the weights as the linear parameters of the MLP. $\mathbf{x}_i$ and t_i refer to the ith input pattern and its corresponding target value, respectively. $\mathrm{y}(\mathbf{x}_i, \mathbf{w})$ denotes the MLP output for $\mathbf{x}_i$ with respect to $\mathbf{w}$. Each weight of the MLP in any layer is updated based on the update rule stated in Eq. (2.18),

$$w_{ab}^{l\,(j)} = w_{ab}^{l\,(j-1)} - \alpha \frac{\partial}{\partial w_{ab}^{l\,(j-1)}} \Omega(\mathbf{w}) \tag{2.18}$$

where $w_{ab}^{l\,(j-1)}$ is the weight of the link connecting neuron a in layer l to neuron b in layer $l-1$ in the $(j-1)$th iteration. α denotes the learning rate and $\frac{\partial}{\partial w_{ab}^{l\,(j-1)}} \Omega(\mathbf{w})$ refers to the gradient of the cost function $\Omega(\mathbf{w})$ with respect to $w_{ab}^{l\,(j-1)}$.

The learning process has two passes: the forward pass and the backward propagation. In the forward pass, the MLP outputs are calculated for each input pattern while in the backward propagation pass, for each node in layer l, the contribution of the neuron to the outputs error of the MLP is computed for each input pattern in $(j-1)$th iteration. Suppose the MLP has L hidden layers so the layer 1 is assigned to the input layer and layer $(L+1)$ corresponds to the output layer. The contribution of neuron a in layer l to the output error of the MLP is defined in terms of the partial gradient (i.e., denoted by $\delta_a^{l\,(j-1)}$) of $\Omega(\mathbf{w})$ with respect to the input of neuron a in layer l in the $(j-1)$th iteration. If neuron a is an output neuron, $\delta_a^{l\,(j-1)}$ is directly computed from the output error of the MLP. In the case that neuron a is a hidden neuron, it is computed in a recursive way as stated in Eq. (2.19),

$$\delta_a^{l\,(j-1)} = g'\left(z_a^{l\,(j-1)}\right) \sum_{k=1}^{N_{l+1}} \delta_k^{(l+1)\,(j-1)}\, w_{ka}^{l+1\ (j-1)}, \qquad 2 \le l \le L \tag{2.19}$$

where g' is the first derivative of the activation function defined in Eq. (2.20) and $z_a^{l\,(j-1)}$ is the input of neuron a in layer l in iteration $(j-1)$th defined in Eq. (2.21). N_{l+1} denotes the number of neurons in layer $l+1$. $w_{ka}^{l+1\ (j-1)}$ refers to the weight of the link connecting neuron k in layer $l+1$ to neuron a in layer l in iteration $(j-1)$th. In Eq. (2.21), $\mathbf{x}_i$ denotes the ith input pattern.

$$g'(z) = g(z)(1-g(z)) \text{ where } g(z) = \frac{1}{1+e^{-z}} \tag{2.20}$$

$$z_a^{l\,(j-1)} = \begin{cases} \mathbf{w}^{1\,(j-1)}\mathbf{x}_i & ,\ l = 2 \\ \sum_{k=1}^{N_{l-1}} w_{ak}^{(l)\ (j-1)} * z_k^{(l-1)\,(j-1)} & ,\ 3 \leq l \leq \mathrm{L} \end{cases} \tag{2.21}$$

Backward propagation is done for all samples and then the gradient of the cost function $\Omega(\mathbf{w})$ with respect to $w_{ab}^{l\ (j-1)}$ is obtained by Eq. (2.22),

$$\frac{\partial}{\partial w_{ab}^{l\ (j-1)}}\Omega(\mathbf{w}) = \frac{1}{m} \times \sum_{i=1}^{m} \varphi_b^{(l-1)\ (j-1)}(\mathbf{x}_i)\delta_a^{(l)\ (j-1)}(\mathbf{x}_i) \tag{2.22}$$

where $\varphi_b^{(l-1)\ (j-1)}$ is the output of neuron b in layer $l-1$ for input pattern $\mathbf{x}_i$ in $(j-1)$th iteration.

2.4.1.2. Newton's method

Since the convergence speed of the steepest descent method is slow, Newton's method was proposed to speed up the learning process. In Newton's method an approximation of the cost function $\Omega(\mathbf{w})$ is considered by using the second order Taylor expansion of $\Omega(\mathbf{w})$ around point $\mathbf{w}^{(t)} = \left(w_0^{(t)}, w_1^{(t)}, \dots, w_n^{(t)}\right)$. This approximation gives rise to the quadratic optimization problem stated in Eq. (2.23).

$$\Omega(\mathbf{w}) \approx \Omega\left(\mathbf{w}^{(t)}\right) + \nabla_{\mathbf{w}^{(t)}} \times (\mathbf{w}-\mathbf{w}^{(t)})^T + \frac{1}{2} \times (\mathbf{w}-\mathbf{w}^{(t)})^T \mathbf{H}^{(t)} (\mathbf{w}-\mathbf{w}^{(t)}) \tag{2.23}$$

Where $\nabla_{\mathbf{w}^{(t)}}$ and $\mathbf{H}^{(t)}$ denote the first and second derivatives of $\Omega(\mathbf{w})$ with respect to $\mathbf{w}$, respectively. $\mathbf{H}^{(t)}$ is also called the Hessian matrix. The minimum value for the estimated cost function $\Omega(\mathbf{w})$ is obtained by solving Eq. (2.24).

$$\nabla_{\mathbf{w}^{(t)}} + \mathbf{H}^{(t)}\left(\mathbf{w}-\mathbf{w}^{(t)}\right) = 0 \tag{2.24}$$

The solution of Eq. (2.24) is obtained by Eq. (2.25) which is the Newton's update method.

$$\mathbf{w} = \mathbf{w}^{(t)} - (\mathbf{H}^{(t)})^{-1}\nabla_{\mathbf{w}^{(t)}} \tag{2.25}$$

In order to update the parameters $\mathbf{w}$ in each iteration by Eq. (2.25), $\mathbf{H}^{(t)}$ must be positive definite which is not always guaranteed by Newton's method [17]. In addition, computing the inverse of $\mathbf{H}^{(t)}$ takes $O(n^3)$ time for each iteration t where n is the number of parameters. In order to deal with this limitation of Newton's method, several efficient methods were proposed. In the following, we will introduce some of them.

2.4.1.3. Quasi-Newton method

The Quasi-Newton method updates $\mathbf{H}^{(t)}$ based on the changes between the gradient of the current iteration and that of the previous one, instead of completely computing $\mathbf{H}^{(t)}$ in each iteration. Several methods have been proposed to gradually update $\mathbf{H}^{(t)}$ including the Davidon–Fletcher–Powell formula (DFP), SR1 formula (Symmetric Rank one), the BHHH method, the BFGS method and the low memory extension of BFGS called L-BFGS [30]. Among these, the BFGS stated in Eq. (2.26) is considered the most effective method for a general unconstrained optimization problem [18].

$$\mathbf{H}_{BFGS}^{(t+1)} = \mathbf{H}^{(t)} + \left(1 + \frac{(\mathbf{q}^{(t)})^T\mathbf{H}^{(t)}\mathbf{q}^{(t)}}{(\mathbf{s}^{(t)})^T\mathbf{q}^{(t)}}\right)\frac{\mathbf{s}^{(t)}(\mathbf{s}^{(t)})^T}{(\mathbf{s}^{(t)})^T\mathbf{q}^{(t)}} - \left(\frac{\mathbf{s}^{(t)}(\mathbf{q}^{(t)})^T\mathbf{H}^{(t)} + \mathbf{H}^{(t)}\mathbf{q}^{(t)}(\mathbf{s}^{(t)})^T}{(\mathbf{s}^{(t)})^T\mathbf{q}^{(t)}}\right) \tag{2.26}$$

where $\mathbf{s}^{(t)} = \mathbf{w}^{(t+1)} - \mathbf{w}^{(t)}$ and $\mathbf{q}^{(t)} = \nabla_{\mathbf{w}^{(t+1)}} - \nabla_{\mathbf{w}^{(t)}}$.

2.4.1.4. Gauss-Newton method

As another alternative to Newton's method, the Gauss-Newton method applies an approximation of $\mathbf{H}^{(t)}$ with the assumption that the underlying problem is a nonlinear least square one with the cost function defined in Eq. (2.27),

$$\Omega(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m} \mathbf{e}_i^2(\mathbf{w}) \text{ }, \mathbf{w} = (w_0, w_1, \dots, w_n) \tag{2.27}$$

where m is the number of training samples and $\boldsymbol{e}_i(\mathbf{w})$ denotes the model output for ith input pattern. $\mathbf{w}$ refers to the parameters vector. The first-order partial derivative of $\Omega(\mathbf{w})$ with respect to each parameter $\boldsymbol{w}_j$ for $j = 0,1,\cdots,n$ is obtained by Eq. (2.28),

$$\nabla_{w_j} = \sum_{i=1}^{m} \mathbf{e}_i \frac{\partial \mathbf{e}_i}{\partial \boldsymbol{w}_j} = \sum_{i=1}^{m} \mathbf{e}_i \mathbf{J}_{ij} \quad , j = 0,1,\dots,n \tag{2.28}$$

where $\mathbf{J}_{ij}$ is the element in ith row and jth column of the $m \times n$ matrix $\mathbf{J}$, which is the so called Jacobean matrix. The relation between the gradient vector $\boldsymbol{\nabla}_w$ and Jacobean matrix $\mathbf{J}$ in tth iteration is expressed by Eq. (2.29) which is the matrix notation of Eq. (2.28).

$$\boldsymbol{\nabla}_{\mathbf{w}^{(t)}} = (\mathbf{J}^{(t)})^T \mathbf{e}^{(t)} \tag{2.29}$$

In practice, the Hessian matrix **H** is a squared matrix of size $n \times n$ whose elements are computed by Eq. (2.30),

$$\mathbf{H}_{jk} = \sum_{i=1}^{m} \left(\frac{\partial \mathbf{e}_i}{\partial \mathbf{w}_j} \frac{\partial \mathbf{e}_i}{\partial \mathbf{w}_k} + \mathbf{e}_i \frac{\partial^2 \mathbf{e}_i}{\partial \mathbf{w}_j \partial \mathbf{w}_k} \right), \quad j,k = 0,1,\dots,n \tag{2.30}$$

where $\mathbf{H}_{jk}$ is the element in jth row and kth column of **H**. The Gauss-Newton method presents an approximation to **H** by eliminating the second term in Eq. (2.30). Hence the approximation of **H** can be stated as Eq. (2.31),

$$\mathbf{H}_{jk} \approx \sum_{i=1}^{m} \left(\frac{\partial \mathbf{e}_i}{\partial \mathbf{w}_j} \frac{\partial \mathbf{e}_i}{\partial \mathbf{w}_k} \right) = \sum_{i=1}^{m} \mathbf{J}_{ij} \mathbf{J}_{ik} \;, \quad j,k = 0,1,\dots,n \tag{2.31}$$

whose matrix notation is given by Eq. (2.32),

$$\mathbf{H}^{(t)} = (\mathbf{J}^{(t)})^T \mathbf{J}^{(t)} \tag{2.32}$$

where $\mathbf{H}^{(t)}$ and $\mathbf{J}^{(t)}$ denote the Hessian and the Jacobean matrix in tth iteration. By replacing Eqs. (2.29) and (2.32) in Eq. (2.25), the Gauss-Newton update rule in tth iteration can be obtained as Eq. (2.33).

$$\mathbf{w} = \mathbf{w}^{(t)} - ((\mathbf{J}^{(t)})^T\mathbf{J}^{(t)})^{-1}(\mathbf{J}^{(t)})^T\mathbf{e}^{(t)} \tag{2.33}$$

2.4.1.5. Levenberg-Marquardt method

Although the approximation of the Hessian matrix **H** in the Gauss-Newton method rprovides increased speed of the method, the invertibility of $((\mathbf{J}^{(t)})^T\mathbf{J}^{(t)})$ is still not guaranteed [31]. To deal with this problem the Levenberg-Marquardt method [32, 33] was proposed. In each iteration the term $\delta\mathbf{I}$ (a diagonal matrix) is added to $((\mathbf{J}^{(t)})^T\mathbf{J}^{(t)})$ to guarantee that it becomes a nonsingular matrix that is invertible. δ is a variable scalar value that changes in every iteration by a given factor and **I** is the identity matrix so that $((\mathbf{J}^{(t)})^T\mathbf{J}^{(t)} + \delta\mathbf{I})$ is invertible. The Levenberg-Marquardt update rule is stated in Eq. (2.34).

$$\mathbf{w} = \mathbf{w}^{(t)} - ((\mathbf{J}^{(t)})^T\mathbf{J}^{(t)} + \delta\mathbf{I})^{-1}(\mathbf{J}^{(t)})^T\mathbf{e}^{(t)} \tag{2.34}$$

The value of δ has a critical role to change the behavior of Levenberg-Marquardt. For small values of δ, $((\mathbf{J}^{(t)})^T\mathbf{J}^{(t)})$ has significant influence in parameters update. In this case, the behavior of the Levenberg-Marquardt method is the same as the Gauss-Newton method. In the case of assigning large values to δ, $((\mathbf{J}^{(t)})^T\mathbf{J}^{(t)})$ does not have significant contribution in parameters update. In this situation, the Levenberg-Marquardt behaves like the steepest decent method where only $(\mathbf{J}^{(t)})^T\mathbf{e}^{(t)}$, as the gradient of cost function, (according to Eq. (2.29)) has an important effect on the update. The method starts with a small value of δ, therefore behaving close to the Gauss-Newton method and continues decreasing δ as long as the error decreases Whenever the error increases in an iteration, the parameters are reset to the values obtained in the previous iteration and δ is made larger with the aim of changing the behavior of the Levenberg-Marquardt method in the direction of the steepest decent method.

By considering the cost function as a least squares problem, the performance of learning algorithms can be improved by separating parameters into linear and nonlinear in the learning process [24]. Suppose $\mathbf{u}$ and $\boldsymbol{\upsilon}$ denote linear and nonlinear parameters, respectively. The model output can be stated as Eq. (2.35).

$$\mathbf{y} = \boldsymbol{\phi}(\boldsymbol{\upsilon})\mathbf{u} \tag{2.35}$$

where $\boldsymbol{\phi}$ denotes the output matrix of the last hidden layer possibly including a column of ones corresponding to the model output bias. By replacing Eq. (2.35) in Eq. (2.27), the following nonlinear least squares problem is obtained,

$$\Omega(\mathbf{u},\boldsymbol{\upsilon}) = \frac{1}{2}\sum_{i=1}^{m} \mathbf{e}_i^2(\mathbf{u},\boldsymbol{\upsilon}) = \frac{\|\mathbf{t} - \boldsymbol{\phi}(\boldsymbol{\upsilon})\mathbf{u}\|_2^2}{2} \tag{2.36}$$

where $\mathbf{t}$ is the target vector. Considering any constant value of $\boldsymbol{\upsilon}$, the optimum value of $\mathbf{u}$ minimizing $\Omega(\mathbf{u},\boldsymbol{\upsilon})$ can be obtained using the pseudo-inverse method:

$$\hat{\mathbf{u}}(\boldsymbol{\upsilon}) = \boldsymbol{\phi}(\boldsymbol{\upsilon})^{+}\mathbf{t} \tag{2.37}$$

By replacing Eq. (2.37) in Eq. (2.36), a new training criterion is obtained where the cost function only depends on the nonlinear parameters $\boldsymbol{\upsilon}$**:**

$$\psi(\boldsymbol{\upsilon}) = \frac{\|\mathbf{t} - \boldsymbol{\phi}(\boldsymbol{\upsilon})\boldsymbol{\phi}(\boldsymbol{\upsilon})^{+}\mathbf{t}\|_2^2}{2} \tag{2.38}$$

To minimize the criterion in Eq. (2.38), the corresponding gradient must be computed. It has been proven in [34] that the gradient of $\psi(\boldsymbol{\upsilon})$ can be determined in such a way that, firstly, the optimal value of $\mathbf{u}$ is obtained by Eq. (2.37) and then it is replaced in Eq. (2.36). Afterwards, the gradient of $\psi(\boldsymbol{\upsilon})$ can be obtained by performing the usual calculation. The new criterion has several advantages when compared to the classic one in Eq. (2.36):

- It decreases the dimension of the optimization problem since only nonlinear parameters are considered.
- It makes the Levenberg-Marquardt method faster since each iteration of the learning process becomes computationally cheaper.
- A small number of iterations is needed to converge to the local minimum of the cost function since the initial values obtained by using Eq. (2.38) are much lower than those obtained by using Eq. (2.36). Moreover, the new criterion results in a faster rate of convergence.

2.4.1.6. Four strategies for training RBFNNs

The strategies considered for training RBFNNs differ on how the centers and spreads of the hidden neurons are computed [17, 35, 36]. Regarding the basic strategy used in the standard RBFNN proposed by Broomhead and Lowe [20], each sample of the training set corresponds to a particular center of the RBFNN producing an interpolating surface which exactly passes throughout all samples of the training set. In case of the presence of a large size training set, a large size RBFNN is produced. Moreover, in application, the exact curve fitting is neither useful nor desirable since it may lead to anomalous interpolation properties [20]. To relax the strict interpolation, three main strategies can be employed.

In the first strategy [37], the center of each hidden neuron corresponds to a random input pattern in the training set and for all hidden neurons, the same spread is considered as given by Eq. (2.39),

$$\sigma = \frac{d_{max}}{\sqrt{2n}} \tag{2.39}$$

where n is the number of centers and d_{max} is the maximum Euclidean distance between the selected centers. Afterwards, the linear parameters $\mathbf{u} = [u_0, u_1, \cdots, u_n,]$ can be obtained with the application of pseudo-inverse method (see Eq. (2.37)).

The second strategy [38] benefits from both supervised and unsupervised learning methods. In this strategy, which is also called self-organized selection of centers, the centers of the hidden neurons are determined by applying for instance a clustering method like k-means or any extended version of that. First, the samples in the training set are grouped into a number of clusters and then the center of each cluster is considered as a center of a hidden neuron. Afterwards, the spread of each hidden neuron can be determined by Eq. (2.39) or other heuristics that have been proposed [24].

Once the centers and the spreads, as nonlinear parameters, are determined, the output linear weights of the RBFNN model can be found as the solution of a linear least square optimization problem (see Eq. 2.37).

The strategies described above determine the non-linear parameters by using stochastic or heuristic methods. Therefore there is no guarantee that the non-linear parameters are the optimal ones in the minimization of the training error criterion. The third strategy has already been mentioned in Section 2.4.1.5 where the linear parameters can be obtained optimally by using the pseudo-inverse operation and then the nonlinear parameters are determined using a

nonlinear least squares optimization problem. This way all the parameters are involved in the minimization of the training error criterion.
The fourth strategy which benefits from Orthogonal Least Squares (OLS) method is an iterative task that starts with an empty hidden layer and continues with adding a center at a time and updating the linear parameters (i.e., weights) until some criteria are met. Some approaches based on OLS can be seen in [39-41].

2.4.1.7. Termination criteria in training process

The training process should be ended when a desired level of accuracy is obtained. In order to achieve this goal, four approaches may be applied to terminate the training process.
The first approach focuses on a fixed number of iterations which is determined as a user-defined threshold. The main disadvantage is that for a given problem it is not clear how many iterations are necessary to guarantee that a desired level of accuracy is obtained.
In order to deal with the problem of the first approach, the second approach simultaneously checks three termination criteria shown in Eqs. (2.40) to (2.42), that reflect the accuracy and parameter convergence of the model [24]. Whenever all criteria are met, the training process ends.

$$\Omega[k-1] - \Omega[\mathrm{k}] < \theta[\mathrm{k}] \tag{2.40}$$

$$\|\mathbf{w}[k-1] - \mathbf{w}[k]\| < \sqrt{\tau_f}.(1 + \|\mathbf{w}[k]\|) \tag{2.41}$$

$$\|\mathbf{g}[k]\| \leq \sqrt[3]{\tau_f}.(1 + |\Omega[k]|) \tag{2.42}$$

$$\theta[\mathrm{k}] = \tau_f.(1 + \Omega[k]) \tag{2.43}$$

In these criteria k denotes the k^{th} iteration. Ω, **w** and **g** refer to the cost function, the parameters vector and the gradient vector, respectively. τ_f is a user-defined threshold denoting a measure of the desired number of correct digits in the cost function. τ_f has a critical role in the training process. For example, assigning a small value to τ_f may lead to have an over-trained model. An over-trained model has a high level of accuracy for the training samples but not an acceptable level of generalization for unseen data [24].
To avoid the over-training phenomenon, the third approach, which is called early stopping method, is considered. The model is evaluated not only on the training samples but also on another set of samples called testing set. In each iteration, the model error is computed for

both training and testing sets. If both training and testing errors are decreasing in comparison to the error in previous iterations, the training process is allowed to continue. In the case that the training error is decreasing but the testing error is increasing, the training process terminates since the model started losing its generalization capability. In this situation, the values of the parameters may be set to the values obtained in an appropriate previous iteration. Fig. 2.8 illustrates how the early stopping method works.

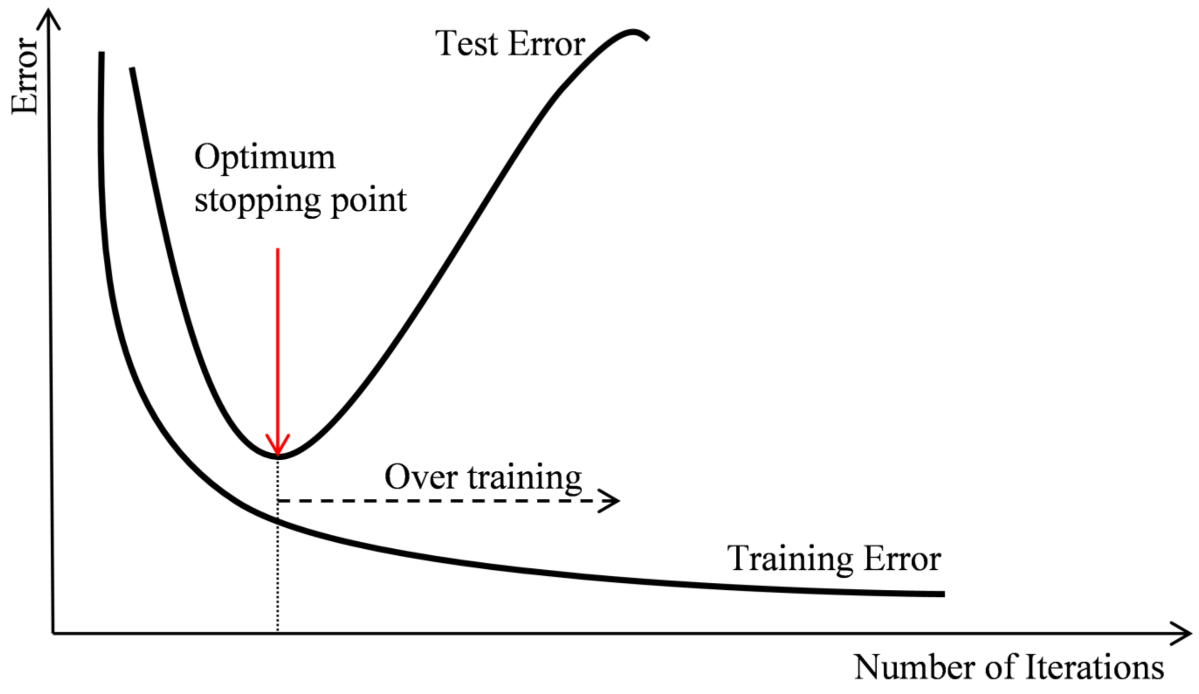


Fig. 2.8. Early stopping method.

2.5. Performance Criteria

Once models are trained, there are various criteria to evaluate their performance. These criteria must allow the comparison of different types of models in terms of their performances. Regarding regression problems, the performance criteria express how much the model's outputs (i.e., predicted values) are close to their corresponding real values (i.e., measured values). Hence, they are specified in terms of the errors obtained between the real and the predicted values. Some criteria used in regression problems are the Root Mean Square Error (RMSE), Mean Absolute Error (MAE), Mean Relative Error (MRE), Mean Absolute Percentage Error (MAPE), Maximum Absolute Error (MaxAE) and standard deviation of predicted values (σ). These can be calculated by Eqs. (2.44) to (2.49).

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{2.44}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{2.45}$$

$$MRE = \frac{1}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{y_i} \tag{2.46}$$

$$MAPE = \frac{100\%}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{|y_i|} \tag{2.47}$$

$$MaxAE = \max(AE(y, \hat{y})) \tag{2.48}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - \bar{\hat{y}}_i(i))^2} \tag{2.49}$$

In Eqs. (2.44) to (2.49), N, y_i and $\hat{y_i}$ denote the number of samples, and the real and predicted values of the output variable for the i^{th} sample, respectively.

Regarding classification problems, the most common criteria are the Classification Rate (CR), the specificity and the sensitivity. These criteria can be calculated by using Eqs. (2.50) to (2.52).

$$Classification\ Rate = \frac{TP + TN}{N} \tag{2.50}$$

$$Specificity = \frac{TP}{TP + FN} \tag{2.51}$$

$$Sensitivity = \frac{TN}{TN + FP} \tag{2.52}$$

TP, TN, FP and FN denote True Positive, True Negative, False Positive and False Negative, respectively. The corresponding definitions are as follows:

TP: The number of positive samples which have been correctly classified by the model.

TN: The number of negative samples which have been correctly classified by the model.

FP: The number of negative samples which have been wrongly classified as positive ones by the model.

FN: The number of positive samples which have been wrongly classified as negative ones by the model.

2.6. Genetic Algorithm

The Genetic Algorithm was inspired by the natural process of evolution and was considered as an optimization method where it, as a guided search method, tries to find an acceptable or satisfactory solution in a search space. The problem being solved by a GA is viewed as a black box system with a number of input parameters and one output parameter. In a more formal way, the input parameters that describe a possible solution are encoded in a representation called the chromosome, whereas the output parameter is usually the result of a function that captures the fitness of the candidate solution to the problem being addressed. The goal is finding a combination of input parameters' values resulting in a satisfactory value for the output parameter [42]. To find an optimal solution, GA starts with an initial population of the potential solutions for the underlying problem. Each solution in the population, termed an individual, is evaluated by the problem specific fitness function reflecting the problem goal. Afterwards, the initial population is evolved by applying genetic operators that mimic the natural process of evolution. The canonical GA considers, mating selection, parent recombination, mutation and replacement operators. This way the initial population is replaced by a new generation by mating the elitists of the initial population so that the individuals in the new generation are expected to be more fit to the problem. The evolution process continues by producing a number of generations, expecting that, eventually after an appropriate number of generations, suitable individuals in terms of their fitness are available. The following subsections describe the GA operators.

2.6.1. Selection

Once an initial population is generated and a fitness value is assigned to each individual in the population, some of them should be selected to produce a new generation. There are several well-known selection methods including roulette wheel (or Fitness Proportionate Selection), Stochastic Universal Sampling (SUS), Tournament selection and Truncation selection. The following briefly introduces these methods.

- *Roulette wheel*

  In this method, firstly, the individuals are sorted in an ascending order based on their fitness values. Suppose that fitness value f_i is assgind to the i^{th} individual. In the second step, for each individual, the corresponding normalized fitness value is computed as:

$$p_i = \frac{f_i}{\sum_{k=1}^{I} f_i} \tag{2.53}$$

  where I is the number of individuals. Accually, the normalized fitness value p_i denotes the probability of i^{th} individual to be selected. In the third step, for each individual, a probability interval based on the accumulated normalized fitness values is computed as:

$$[l_i, l_i + p_i) = [\sum_{j=1}^{i-1} p_j, \sum_{j=1}^{i-1} p_j + p_i) \tag{2.54}$$

  where l_i denotes the lower bound of the interval corresponding to i^{th} individual. It is notable that for the first individual, l_1 is equal to zero. In the fourth step, a random number r from the range $[0,1]$ is selected and then the individual whose interval includes r is selected for mating. This step is repeated for a number of iterations to produce a pool of parents for mating. Based on this method, the individuals with large fitness values have more chance to be selected than the others. They are also likely to be selected more than once throughout the selection process.

- *Stochastic Universal Sampling*

  The Stochastic Universal Sampling method is the extended version of roulette wheel method where multiple evenly spaced pointers are considered at a time instead of using a single pointer to select an individual. Hence, in SUS, multiple individuals are selected at a time. The advantage of SUS is giving a chance to weaker individuals to be selected as well. This method reduces the unfair behavior of the roulette wheel method which mostly ignores the weaker individuals. In SUS method, firstly, a random value r is selected from the range $[0, k]$ where k is obtained as:

$$k = \frac{F}{I} \tag{2.55}$$

  where F and I are the summation of fitness values and the number of individuals that are supposed to be selected by the method, respectively. In the second step, I pointers are generated as:

$$pointers = [p_1, p_{2,\cdots,} p_N], \qquad p_i = r + (i-1) * k \qquad for\ i = 1\ to\ I \tag{2.56}$$

Like roulette wheel method, for each pointer p_i, the individual whose interval includes p_i is selected for mating.

- *Tournament*

  Tournament selection method tries to randomly select a subset of size k (i.e., tournament size) of the current population for several times. In each tournament, the best individual is selected as a parent to reproduce the next generation. If k is a large value, the weaker individuals have a lower chance to be selected. On the other hand, when $k = 1$, tournament method acts as the random selection method.

- *Truncation*

Truncation method is one the simplest selection methods where firstly, the individuals are sorted based on their fitness value and then some proportion, p, of the best individuals are selected.

2.6.2. Recombination

The children are generated through the combination of their parents' genes in form of new chromosomes. Combining the genes is performed by the genetic operator called crossover. The crossover operator is done with a probability called crossover rate which usually takes a large value of probability (e.g., 0.7). A number of crossover methods have been proposed. Among them, single-point, two-point, Cut and Splice and Uniform are well-known crossover methods. Suppose that a chromosome is a binary string so that each gene takes a value from $\{0,1\}$. The follwoing explaines the crossover methods mentioned above.

- *Single-point crossover*

  In single-point method, a common crossover point is selected for both parents and then all genes beyond the point are swapped between the two parents which results in having two children whose lengths are the same as that of their parents. Fig. 2.9 shows an example of single-point crossover method.

- *Two-point crossover*

  In two-point method, two common crossover points are selected rather than one point. In this method, all genes between two points are swapped between the two parents which results in having two children whose lengths are the same as that of their parents. Fig. 2.10 shows an example of two-point crossover method.

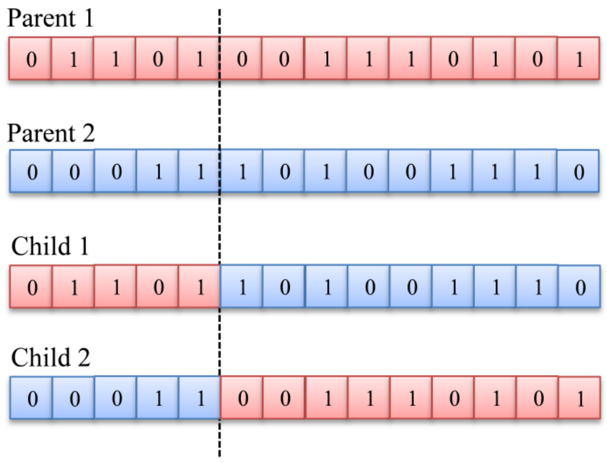


Fig. 2.9. Single-point crossover

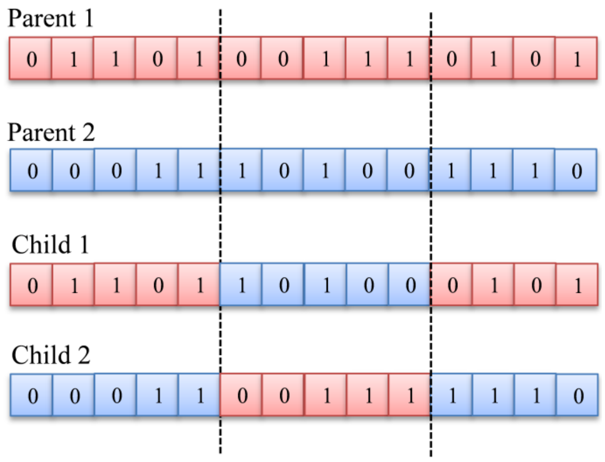


Fig. 2.10. Two-point crossover

- *Cut and Splice crossover*

  Cut and Splice method is similar to single-point method but with the difference that each parent has its own crossover point. All genes beyond each point are swapped between the two parents which results in having two children with different lengths. Fig. 2.11 illustrates an example of Cut and Splice method.

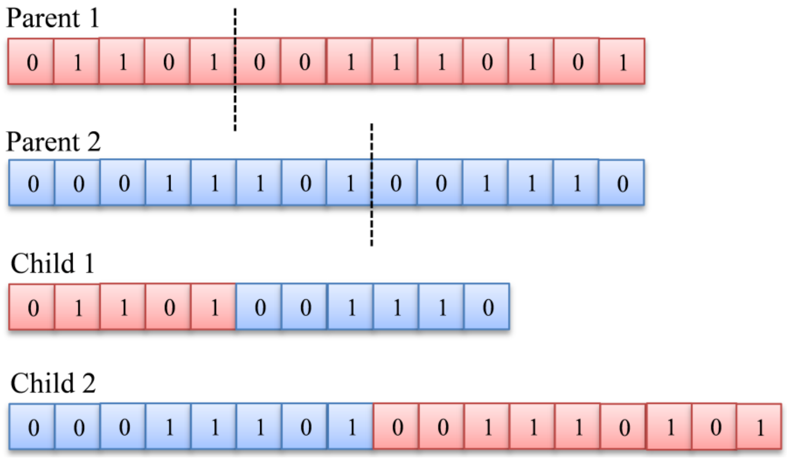


Fig. 2.11. Cut and Splice crossover

- *Uniform crossover*

  Unlike previous methods, in Uniform method, the parents have contribution in producing the children in gene level instead of segment level. In other words, each gene is decided whether to be exchanged with its corresponding gene in the other parent or to be kept unchanged. In this method, each gene is swapped with a fixed probability, typically 0.5 where for each child, approximately half of its genes belong to the first parent and the other half is inherited from the second parent. Fig. 2.12 shows an example of Uniform method.

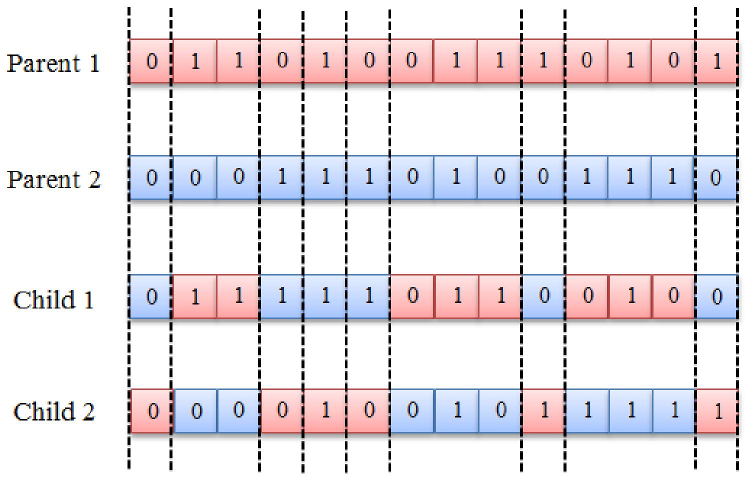


Fig. 2.12. Uniform crossover

2.6.3. Mutation

In order to bring diversity to the next generation, another genetic operator called mutation is performed on a generated child where one or more genes of the corresponding chromosome is/are modified. Unlike the crossover rate, the mutation rate takes a very small value. The common mutation method modifies the value of a gene with a probability $\frac{1}{l}$ where l is the length of the chromosome.

2.6.4. Replacement

Once a population of children is generated and evaluated, the last decision is how to replace the current generation with it. The common way is replacing the current population with all generated children. In some advanced methods, the parents of the current population are allowed to compete with the children where a percentage of the current population migrates to the next generation. Some advanced methods can be found in [43].

2.6.5. Multi-Objective Genetic Algorithm

In the real world, the optimization of an engineering problem is a complicated task due to the presence of multiple objectives which, most of the time, are conflicting with each other. This means that improving one may deteriorate the other. In this case, there is a Pareto-optimal or non-dominated set of solutions in which each solution is not better than the other with respect to the multiple objectives. Fig. 2.13 shows an example of a minimization problem with two objectives. The whole space of solutions is divided into two groups: the shaded region presents the dominated solutions while the solid curve illustrates the non-dominated set of solutions regarding objectives obj.1 and obj.2. As it can be seen in Fig. 2.13, A and B denote two non-dominated solutions.

The goal of a multi-objective optimizer is improving the surface of non-dominated solutions (i.e., the solid curve) in such a way that it approaches the origin (i.e., point ‘O’ in Fig. 2.13) as much as possible.

GAs are well established tools to deal with multi-objective optimization problems [44-46]. In the Multi Objective Genetic Algorithm [4, 24, 46, 47], each individual in the population is evaluated in the space of multiple objectives rather than in one single objective. In addition, at the end of one run of MOGA, a Pareto set of solutions is provided instead of achieving one solution that is better than all others.

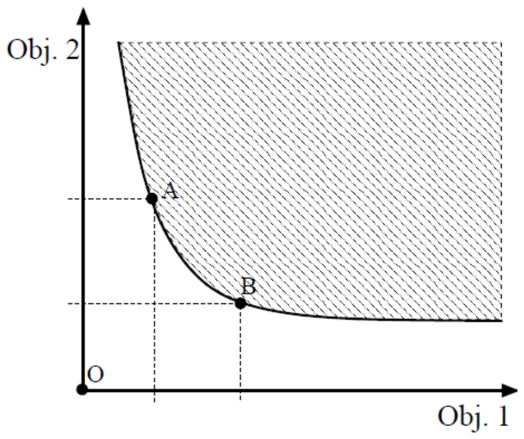


Fig. 2.13. Bi-objective minimization problem. The shaded region presents dominated solutions and the solid curve illustrates non-dominated solutions [47].

An efficient Pareto-based ranking method has been proposed in [43, 47]. In this method, each individual is ranked based on the number of individuals by which they are dominated. For non-dominated individuals, rank 0 is considered; if an individual is dominated by m individuals, rank m will be assigned to it. Fig. 2.14 shows an example of Pareto-based ranking for two objectives that should be minimized.

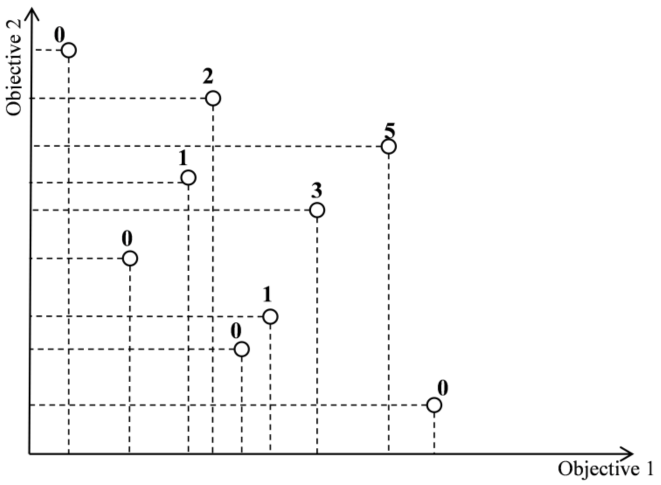


Fig. 2.14. An example of Pareto ranking [48, 49].

In most applications, it is common to assign different priorities to the objectives or define different goals for them which the MOGA tries to achieve. In the case that all objectives have the same priorities, for those individuals which satisfy all goals, their corresponding rank is similarly equal to the number of individuals by which they are dominated. Those individuals which do not meet some goals are penalized by a higher rank. Fig. 2.15 illustrates an example of Pareto-based ranking for two objectives which have the same priorities. For each objective a predefined goal is considered that should be met.

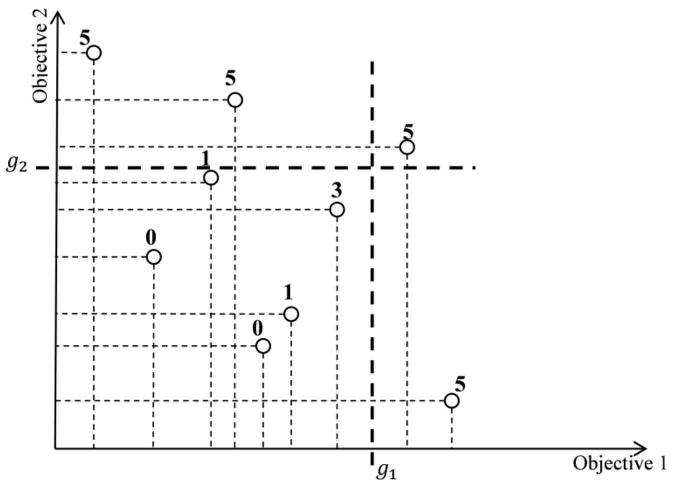


Fig. 2.15. An example of Pareto-based ranking for two objectives with the same priorities. For each objective, a predefined goal is considered [43, 47].

In the case that, objective 2 has higher priority than objective 1, individuals which meet goal g_2 are ranked based on how well they optimize objective 1. Others that do not satisfy g_2 are assigned the worst rank without considering their performance with respect to objective 1. Fig. 2.16 shows an example of this case.

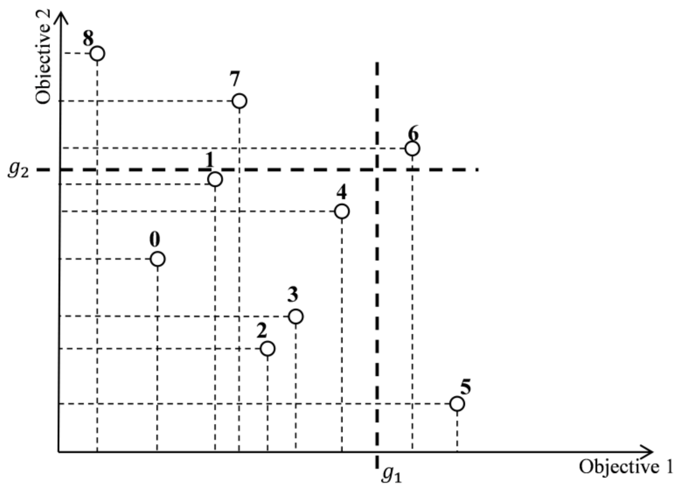


Fig. 2.16. An example of Pareto-based ranking for two objectives in the case that objective 2 has higher priority than objective 1. For each objective, a predefined goal is considered [43, 47].

2.7. Neural network based model design by MOGA

The problem of designing a neural network based model can be divided into two sub-problems as follows [4]:

- *Neural network structure:* It denotes the network inputs, the number of hidden layers, and the number of neurons in each layer.
- *Neural network parameters:* They depend on the model chosen and are usually determined by a suitable learning algorithm.

Since the RBFNN models considered in this thesis were designed by a MOGA, the remaining of this section details the MOGA application to the design of RBFNN models for classification and regression problems.

The output of a RBFNN model is given by Eq. (2.57):

$$o[k] = w_{l+1} + \sum_{m=1}^{l} w_m e^{\frac{\left\| \boldsymbol{i}_j[k] - \boldsymbol{C}(m)] \right\|_2^2}{2\sigma_m^2}} \tag{2.57}$$

In Eq. (2.57), $o[k]$ and $\boldsymbol{i}_j[k]$ denote the model output and the j^{th} input at time instant k, respectively. $\boldsymbol{w}$ represents the vector of the linear weights, $\mathbf{C}(m)$ refers to the vector (extracted from the $\mathbf{C}$ matrix) of the center associated with the m^{th} hidden neuron, σ_{m} is its corresponding spread, and $\| \; \|_2$ represents the Euclidean distance. The network parameters which will be denoted as the parameter vector $\mathbf{p}$, are therefore $\mathbf{C}$, $\boldsymbol{\sigma}$ and $\mathbf{w}$. In order to design a RBFNN model that satisfies a set of defined goals, it is necessary to define a set of quality measures in the form of objectives for each sub-problem mentioned above.

Assume that $\boldsymbol{D} = (\boldsymbol{X}, \boldsymbol{y})$ is a data set composed of N input-output pairs, which is divided into a training set, $\boldsymbol{D}^t$, a generalization or testing set $\boldsymbol{D}^g$ and a validation set $\boldsymbol{D}^v$. Assume also that F is a set of all possible input features (delayed values of the modeled and exogenous variables in time-series regression problems). The problem of designing RBFNN model by MOGA can be expressed as follows:

The Dataset $\boldsymbol{D}$, the allowed range $d \in [d_m, d_M]$ of input features from F and the range $n \in [n_m, n_M]$ of hidden neurons are given as design parameters to the MOGA. After the execution it generates a non-dominated set of RBFNN models that minimize $[\mu_p, \mu_s]$, where

μ_p and μ_s denote a set of objectives related to the RBFNN's parameters $\mathbf{p}$ and its structure, respectively. μ_s includes only one objective,

$$\mu_s = [O(\mu)] \tag{2.58}$$

that denotes the model complexity which is a function of the number of input features and the number of the hidden neurons.

Since the specification of μ_p is different in the classes of problems considered, the following subsections address the specification of μ_p for each class.

2.7.1. Specification of $\boldsymbol{\mu_p}$ in classification problems

In classification problems, we are mainly interested to minimize FP and FN criteria (see Section 2.5). Hence the corresponding objectives for μ_p are considered as:

$$\mu_p = [FP_{\boldsymbol{D}^t}, FN_{\boldsymbol{D}^t}, FP_{\boldsymbol{D}^g}, FN_{\boldsymbol{D}^g}] \tag{2.59}$$

where $FP_{\boldsymbol{D}^t}$ and $FN_{\boldsymbol{D}^t}$ denote the FP and FN on the training set $\boldsymbol{D}^t$, respectively. Similarly, $FP_{\boldsymbol{D}^g}$ and $FN_{\boldsymbol{D}^g}$ refer to the FP and FN on the testing set $\boldsymbol{D}^g$, respectively.

2.7.2. Specification of $\boldsymbol{\mu_p}$ in regression problems

The specification of μ_p in for the case of regression problems relies on the minimization of the error between model outputs and desired values. Therefore, the corresponding objectives for μ_p are defined as:

$$\mu_p = [\varepsilon(\boldsymbol{D^t}), \varepsilon(\boldsymbol{D}^g)] \tag{2.60}$$

where $\varepsilon(\boldsymbol{D^t})$ and $\varepsilon(\boldsymbol{D^g})$ denote the Root Mean Square Errors (RMSE) of the model considering training $\boldsymbol{D}^t$ and the testing set $\boldsymbol{D}^g$.

2.7.2.1. Specification of $\boldsymbol{\mu_p}$ in time series prediction problems

Regarding time series prediction problems, the basic objectives specified for regression problems are also taken into account. Besides these, an additional objective, $\varepsilon(\boldsymbol{D}^s, PH)$, is also considered. Hence the corresponding objectives for μ_p can be defined as:

$$\mu_p = [\varepsilon(\boldsymbol{D}^t), \varepsilon(\boldsymbol{D}^g), \varepsilon(\boldsymbol{D}^s, PH)] \tag{2.61}$$

To understand $\varepsilon(\boldsymbol{D}^s, PH)$, assume $\boldsymbol{E}(\boldsymbol{D}^s, PH)$ is an error matrix defined over the simulation set $\boldsymbol{D}^s$ as expressed in Eq. (2.62), where $\boldsymbol{D}^s$ is composed of a number of consecutive samples with respect to the time instant.

$$E(\boldsymbol{D}^s, PH) = \begin{bmatrix} e[1,1] & e[1,2] & \cdots & e[1,PH] \\ e[2,1] & e[2,2] & \cdots & e[2,PH] \\ \vdots & \vdots & \ddots & \vdots \\ e[m-PH,1] & e[m-PH,2] & \cdots & e[m-PH,PH] \end{bmatrix} \tag{2.62}$$

where $e[i,j]$ is the model prediction error taken from instant i of D^s at step j within the prediction horizon PH. Denoting $\rho(.,i)$ as the RMS function operating over the ith column of its argument matrix, then $\varepsilon(\boldsymbol{D}^s, PH)$ is defined as:

$$\varepsilon(\boldsymbol{D}^s, PH) = \sum_{i=1}^{PH} \rho(\boldsymbol{E}(\boldsymbol{D}^s, PH), i) \tag{2.63}$$

This value is proportional to the area below the curve defined by $\rho(\boldsymbol{E}(\boldsymbol{D}^s, PH), i)$ for i within the prediction horizon, reflecting the model accuracy over the complete prediction horizon for the data set considered.

2.7.3. Model representation in MOGA

Each RBFNN model in the population has a chromosome representation consisting of two components. The first corresponds to the number of hidden neurons and the second one to a string of integers, each one representing the index of a particular feature in F. The chromosome representation is shown in Fig. 2.17.

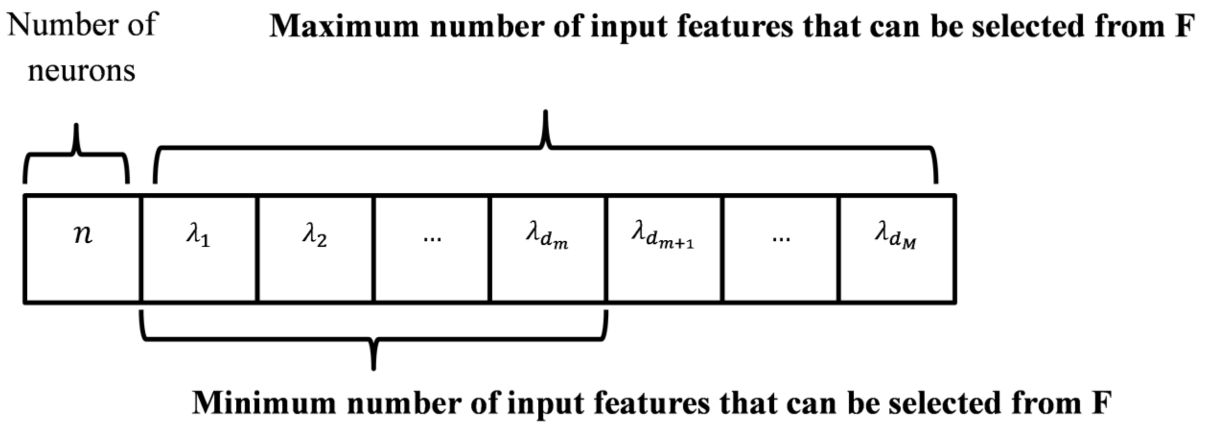

Number of neurons
Maximum number of input features that can be selected from F
n
λ_1
λ_2
…
λ_{d_m}
$\lambda_{d_{m+1}}$
…
λ_{d_M}
Minimum number of input features that can be selected from F

Fig. 2.17. Chromosome representation in MOGA.

Before being evaluated in the MOGA, each model has its parameters determined by a Levenberg-Marquardt algorithm [32, 33] minimizing the error criterion in Eq. (2.38) that exploits the linear-nonlinear relationship of the RBFNN model parameters [34, 50]. The initial values of the nonlinear parameters ($\boldsymbol{C}$ and $\boldsymbol{\sigma}$) are chosen randomly, or by the use of a clustering algorithm, $\boldsymbol{w}$ is determined as a linear least-squares solution, and the procedure is terminated using the early-stopping approach [17] within a maximum number of iterations.

2.7.4. Model design cycle

There are three main actions in the model design cycle: problem definition, solution(s) generation and analysis of results. In the problem definition stage, the data sets, the ranges of features and neurons are defined, as well as the objectives. After this stage, the MOGA execution performs a search to obtain models that satisfy the predefined objectives and goals. In the third stage, the set of models obtained by the MOGA that lie in the Pareto front are analyzed. For this purpose, the performance of the models in the validation set (not involved in the training) is also considered and is of paramount importance. If good solutions are found, the process stops. Otherwise, based on the analysis of results, the search space can be reduced, and/or the objectives and goals can be redefined, therefore restricting the trade-off surface coverage. A more detailed description on the application of the MOGA to the design of ANN models can be found, for instance, in [4, 24].

2.8. Information Theory

Information theory addresses the quantification, storage and communication of information. Entropy, as one of the basic concepts in information theory, measures the expected value of the information contained in any random variable. For a given discrete random variable $\boldsymbol{X}$ with N possible observations, the Shannon entropy is defined as Eq. (2.64) [51].

$$H(\boldsymbol{X}) = -\sum_{i=1}^{N} P(x_i) \log P(x_i) \tag{2.64}$$

where $P(x)$ denotes the Probability Density Function (PDF) of $\boldsymbol{X}$. Mutual Information (MI) as another key concept in information theory measures the dependency between variables. Unlike the correlation coefficient that measures only the linear relationship between variables, MI does not consider any assumption for the underlying relationship. In addition, MI can be defined between groups of variables [52]. MI between two variables $\boldsymbol{X}$ and $\boldsymbol{Y}$ (they can be univariate or multivariate variables) denoted by $I(\boldsymbol{X};\boldsymbol{Y})$ can be interpreted in several ways. Informally, $I(\boldsymbol{X};\boldsymbol{Y})$ measures the amount of information that $\boldsymbol{X}$ and $\boldsymbol{Y}$ share. Formally, $I(\boldsymbol{X};\boldsymbol{Y})$ measures the amount of knowledge of $\boldsymbol{X}$ that reduces the uncertainty about $\boldsymbol{Y}$ and vice versa [53]. It is also translated into the degree of predictability of the second variable knowing the first one [54]. In the case that $\boldsymbol{X}$ (i.e., with N possible observations) and $\boldsymbol{Y}$ (i.e., with M possible observations) are discrete variables, MI is computed by Eq. (2.65).

$$I(\boldsymbol{X};\boldsymbol{Y}) = \sum_{i=1}^{N}\sum_{j=1}^{M} P(x_i, y_j) \log \frac{P(x_i, y_j)}{P(x_i).P(y_j)} \tag{2.65}$$

where $P(.,.)$ and $P(.)$ denote the joint and marginal PDF, respectively. Since the computation of entropy and MI depends on the PDF, the related problematic issue is the estimation of the PDF. The most common methods for PDF estimation are histograms and kernel estimators [55, 56]. Moreover, there is another approach allowing us to directly estimate entropy and MI from data instead of PDF estimation. Some methods for estimating entropy and MI can be seen in [57, 58].

2.9. Overview of two statistical tests

In order to compare the performance of machine learning methods on classification and regression problems, from a statistical point of view, several statistical tests have been proposed. In this way, at the first step, the two involved classification or regression methods are independently applied on to the same data sets. At the second step, based on the corresponding evaluation results (i.e., they can be in terms of RMSE and classification rate for regression and classification problems, respectively) obtained by each method, the comparison of the methods' performances are statistically verified by rejecting or accepting a null hypothesis (i.e., the null hypothesis is equivalent to the assumption that the two methods perform equally well). In this section, the two statistical tests which have been applied in this thesis are explained.

2.9.1. Sign test

Applying Sign test [59] is one of the simple ways to compare the performances of two methods. In this test, two methods are compared with each other in terms of the number of times that the first method has performed better than the second one. This number is also known as the number of wins. In case of tie, the corresponding count is evenly split between them; if there is an odd number of them, one is ignored. In case that multiple methods should be compared, pairwise comparisons are organized in a matrix. Typically, for a large number L of data sets, the critical number of wins with the significance level of $\alpha = 0.05$ is equal to $L/2 + (1.96\sqrt{L})/2$ or $L/2 + \sqrt{L}$. The first method performs better than the second one, if its number of wins is greater than or equal to the critical value.

2.9.2. Wilcoxon signed-ranks test

According to the suggestion of [60], a Wilcoxon signed-ranks test [59] is a proper alternative for t-test. Firstly, Wilcoxon signed-ranks test is safer than t-test since it does not assume the normal distributions. Secondly, the outliers have less effect on the Wilcoxon's performance than they have on the t-test. This is a non-parametric test, which ranks the differences in performances of two methods for each data set, ignoring the signs, and compares the ranks for the positive and the negative differences. Assume that d_i is the difference between the performance scores of the two methods on the ith out of L data sets. The differences are ranked according to their absolute values; average ranks are assigned in case of ties. Let R^+ be the sum of ranks for the data sets on which the second method outperformed the first, and R^- the sum of ranks for the opposite. Ranks of $d_i = 0$ are split evenly among the sums; if there is an odd number of them, one is ignored. Eqs. (2.66) and (2.67) present R^+ and R^-, respectively.

$$R^+ = \sum_{d_i>0} Rank(d_i) + \frac{1}{2}\sum_{d_i=0} Rank(d_i) \tag{2.66}$$

$$R^- = \sum_{d_i<0} Rank(d_i) + \frac{1}{2}\sum_{d_i=0} Rank(d_i) \tag{2.67}$$

Assume that T is the minimum of the sums, $T = \min(R^+, R^-)$. For a small number of data sets (i.e., $L < 60$), most books in general statistics contain a table of exact critical values of T based on some different significance levels of α. For a large number of data sets (i.e., $L \geq 60$), the statistic $z = \frac{T - \frac{1}{4}L(L+1)}{\sqrt{\frac{1}{24}L(L+1)(2L+1)}}$ is distributed approximately normally. With $\alpha = 0.05$, the null hypothesis can be rejected if z is smaller than -1.96.

3. Convex hull algorithms and state of the art

3.1. Introduction

The convex hull of a dataset is a widely known concept in computational geometry. It has been applied in many fields such as computer graphics, pattern recognition, image processing, file searching, statistics, cartography, metallurgy, etc. For example, in some applications, the size of an object can be computed through its image. In case that some pixels of the image might be lost or not be visible, convex hull can provide us an approximated shape of the underlying object. In machine vision applications, convex hull can be applied to detect collisions while navigating over a field of obstacles, where the objects can be substituted with their corresponding convex hull.

The convex hull of a set of data can be presented in terms of vertices and facets where the vertices refer to the boundary points of the data set and the facets denote the connections among the vertices. Since convex hull vertices are useful and informative points reflecting the whole range of data, convex hull can also be considered in the data selection phase in machine learning and data mining tasks.

This chapter is intended to address the explanation of the basic concept of convex hull as well as introducing some standard convex hull algorithms applicable to low and high dimensions. The rest of this chapter is organized as follows: the definition of convex hull along with an overview of convex hull algorithms are explained in Section 3.2. In Section 3.3, some standard convex hull algorithms for low dimensions are introduced. As a state-of-the-art, a proposed convex hull algorithm in high dimensions is introduced in Section 3.4 and finally some conclusions are drawn in Section 3.5.

3.2. An overview of convex hull algorithms

From a computational geometry's point of view, an object in Euclidean space is convex if for any pair of points within the object, the straight line segment that joins them is also within the object. A set S is convex if, for any pair, $x, y \in S$, and all $t \in [0,1]$, the point $(1-t)x + ty$ is in S otherwise S is a concave set. Moreover, if S is a convex set, for any $u_1, u_2, \dots, u_r \in S$, and any nonnegative numbers $\{\lambda_1, \lambda_2, \dots, \lambda_r\}: \sum_{i=1}^{r} \lambda_i = 1$, the vector $\sum_{i=1}^{r} \lambda_i u_i$ is called a convex combination of $u_1, u_2, \dots, u_r$. Intuitively, Figs. 3.1(a) and 3.1(b) illustrate convex and concave sets, respectively.

Fig. 3.1. (a): convex set, (b): concave set

According to the definitions above, the convex hull or convex envelope of a set X of points in the Euclidean space can be defined in terms of convex sets or convex combinations [61-63]:

- the minimal convex set containing X, or
- the intersection of all convex sets containing X, or
- the set of all convex combinations of points in X.

Based on the definition of convex hull, a k-simplex is a k-dimensional polytope which is the convex hull of $k + 1$ affinely independent points. Intuitively, 0-simplex, 1-simplex, 2-simplex and 3-simplex correspond to a point, a line segment, a triangle and a tetrahedron, respectively. Generally, a k-simplex consists of the elements called i-faces where $i \leq k - 1$. 0-faces, 1-faces and $(k - 1)$-faces are called vertices, edges and facets of the k-simplex, respectively. Fig. 3.2 shows the convex hull of a set of points.

Convex hull algorithms can be categorized from three points of view. An algorithm can be deterministic or randomized depending on the order of vertices found. If the order is fixed from execution to execution, the algorithm is deterministic [6]; otherwise, it is randomized [12]. Furthermore, an algorithm can be considered as a real or approximation algorithm. If it is capable of identifying all vertices of the real convex hull, the algorithm is classified as real [8]; otherwise, it is an approximation [10, 64]. Finally, we can also classify convex hull algorithms into offline and online algorithms. The former uses all the data to compute the convex hull, while the latter employs newly arrived points to adapt an already existing convex hull [5]. Fig 3.3 shows the main categories of convex hull algorithms from the three points of view.

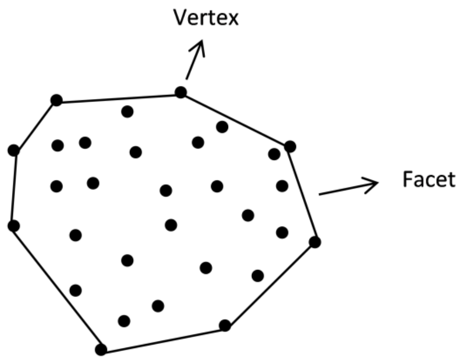


Fig. 3.2. Convex hull of a set of points.

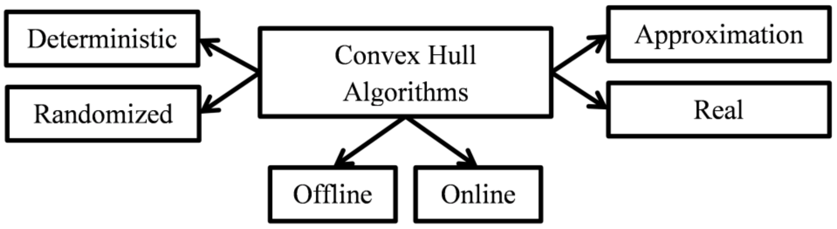


Fig. 3.3. Categories of convex hull algorithms.

3.3. Introduction of convex hull algorithms in two and three dimensions

In one dimension, the convex hull vertices of a set of n points are the minimum and maximum values (i.e., the corresponding convex hull involves two vertices). Hence the time complexity of finding convex hull in one dimension is $O(n)$. For 2 and 3-dimensional Euclidean space, some standard algorithms have been proposed so that the time complexity of most of them is $O(n \log n)$. For 2-dimensional Euclidean space, some basic algorithms have been proposed which are strongly similar to standard sort algorithms where they produce the convex hull vertices in a counterclockwise order. The following introduces some standard real algorithms as well as approximation, online and randomized algorithms in two and three dimensions.

3.3.1. Graham's scan

Graham's scan [6] is one of earliest real deterministic offline algorithm in two dimensions. This algorithm which outputs the vertices in counterclockwise order works based on three elements including the angle between each point and the center of points, the distance of each point to the center and the *left turn* concept.

Three points $\boldsymbol{p_1} = (p_{11}, p_{12})$, $\boldsymbol{p_2} = (p_{21}, p_{22})$ and $\boldsymbol{p_3} = (p_{31}, p_{32})$ make a *left turn* when $\begin{vmatrix} p_{11} & p_{12} & 1 \\ p_{21} & p_{22} & 1 \\ p_{31} & p_{32} & 1 \end{vmatrix}$ is positive where $|.|$ denotes the determinant operation. The positive value demonstrates that the three points are in counterclockwise order while the non-positive value refers to clockwise order corresponding to the *right turn*.

In Graham's scan, first, all points are lexicographically sorted with respect to the polar angle and the distance from the center of points. In the second step, the lowest leftmost point, called the start point, as well as the two consecutive points after that are inserted in the vertices list. Then the algorithm starts traversing the points onwards in a circular way. At each traverse, the new point is compared with the last two vertices found in the previous traverses. If the new and the last two vertices make a *left turn* then the new point is inserted into the list and the traverse progresses; otherwise, the last vertex is deleted from the list and again the *left turn* examination is done. This backward elimination is repeated as long as the *left turn* examination is not met. The algorithm stops when all points are traversed. Fig. 3.4 illustrates an example of applying Graham's scan on a set of 10 points. As it can be seen in Fig. 3.4, the origin of coordinates is transferred to the center point and then all points are sorted with respect to the polar angle and the distance from the center. Consequently, the sorted list is obtained as $\{\boldsymbol{p_2}, \boldsymbol{p_3}, \boldsymbol{p_4}, \boldsymbol{p_5}, \boldsymbol{p_6}, \boldsymbol{p_7}, \boldsymbol{p_8}, \boldsymbol{p_9}, \boldsymbol{Start}, \boldsymbol{p_1}\}$. In Fig. 3.4, $\boldsymbol{Start}$ is the lowest rightmost point which is definitely a vertex of convex hull. Three points, $\boldsymbol{Start}$, $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ are selected as vertices of convex hull and then the next point which is $\boldsymbol{p_3}$ is examined. As it can be seen in Fig. 3.4, triple $(\boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_3})$ makes a *left turn* so $\boldsymbol{p_3}$ is selected as a convex hull vertex. In the next traverse, point $\boldsymbol{p_4}$ is considered. As it can be observed in Fig. 3.4, triple $(\boldsymbol{p_2}, \boldsymbol{p_3}, \boldsymbol{p_4})$ makes a *right turn* so $\boldsymbol{p_3}$ is removed from the vertices list and then triple $(\boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_4})$ is examined. Since this triple makes a *left turn*, the algorithm traverses the next point which is $\boldsymbol{p_5}$. Triple $(\boldsymbol{p_2}, \boldsymbol{p_4}, \boldsymbol{p_5})$ makes a *left turn* therefore $\boldsymbol{p_5}$ is inserted to the vertices list and it allows the algorithm to examine triple $(\boldsymbol{p_4}, \boldsymbol{p_5}, \boldsymbol{p_6})$. As this triple makes a *right turn*, $\boldsymbol{p_5}$ is removed from the vertices list and then triple $(\boldsymbol{p_2}, \boldsymbol{p_4}, \boldsymbol{p_6})$ is examined that allows the algorithm to traverse the next point $\boldsymbol{p_7}$. This procedure continues until all points are traversed. Finally Graham's scan outputs the vertices list as $\{\boldsymbol{Start}, \boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_4}, \boldsymbol{p_6}, \boldsymbol{p_7}, \boldsymbol{p_8}\}$. The time complexity of Graham's scan is $O(n \log n)$.

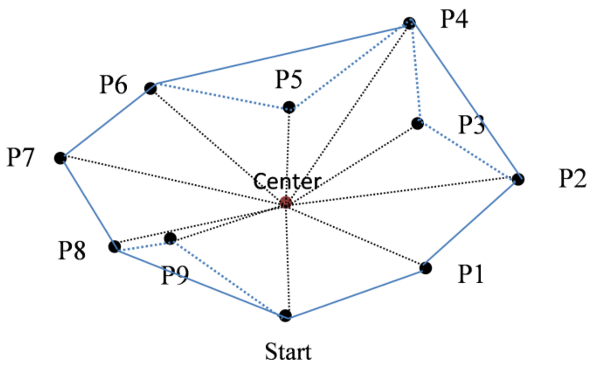


Fig. 3.4. Graham's scan on ten points.

3.3.2. Jarvis's march

Jarvis's march [7] is another instance of real deterministic offline methods . It works based on the theorem stating that a segment line between two points is an edge of the convex hull in planar space if and only if all the remaining points are located in the same side of the edge [5]. The algorithm starts with the lowest rightmost point as the new origin (i.e., the original coordinates is transferred to the new origin) which is a vertex of convex hull. Then the point with the smallest angle with respect to the positive x axis is selected as the second vertex of the convex hull. In the next step, the second vertex is set as the new origin and then another point with the smallest angle with respect to the positive x axis is selected as a new vertex. This procedure continues until it gets to the highest rightmost point. From this point, the algorithm continues to find a new point with the smallest angle with respect to the negative x axis. The algorithm terminates when we get to the lowest rightmost point. Fig. 3.5 illustrates an example of applying Jarvis's march on a set of 10 points. As it can be seen in Fig. 3.5, the algorithm starts with $\boldsymbol{Start}$ as the lowest rightmost point which is definitely a vertex of the convex hull. In next step, the coordinates are transferred to $\boldsymbol{Start}$ as the origin and then $\boldsymbol{p_1}$ as a point with the smallest angle with respect to the positive x axis is selected as a convex hull vertex. In next step, the coordinates are transferred to $\boldsymbol{p_1}$ as the origin and then $\boldsymbol{p_2}$ as a point with the smallest angle with respect to the positive x axis is selected as a convex hull vertex. In next step $\boldsymbol{p_4}$ is selected as another vertex. Since $\boldsymbol{p_4}$ is the highest rightmost point, for next steps, the smallest angle is considered with respect to the negative x axis. Considering $\boldsymbol{p_4}$ as the origin, $\boldsymbol{p_6}$ is selected as another vertex rather than $\boldsymbol{p_5}$. This procedure continues until we get to the $\boldsymbol{Start}$. Ultimately, Jarvis's march results in a vertices list $\{\boldsymbol{Start}, \boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_4}, \boldsymbol{p_6}, \boldsymbol{p_7}, \boldsymbol{p_8}\}$.

The time complexity of Jarvis's march is $O(nf)$ where n and f denote the number of points and the number of convex hull vertices, respectively. Hence this algorithm is an example of output-sensitive algorithms where the time complexity depends not only on the input size but also on the output size. In the worst case, when all points are located on the hull (i.e., no point is identified as an inner point), the time complexity is $O(n^2)$.

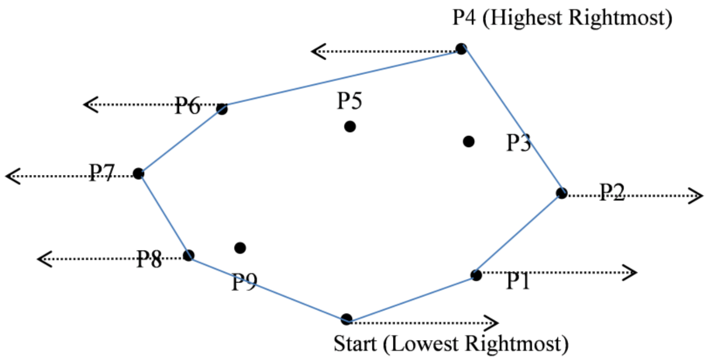


Fig. 3.5. Jarvis's march on ten points.

3.3.3. Quickhull

Quickhull [8] as a promising real deterministic offline algorithm is faster than other proposed algorithms in two dimensions and it can be extended to more than two dimensions. The idea behind Quickhull is growing the current convex hull in each iteration by finding the furthest point with respect to the facets of the current convex hull. In each iteration, the current convex hull is presented in terms of both vertices and facets. The algorithm starts with an initial convex hull which is the maximum 2-simplex being translated to a triangle with maximum area (i.e., the initial convex hull has three vertices). In the next step, the points inside the initial convex hull are marked as inner points and then removed from the set of points. In this step, the initial convex hull with three facets divides the whole space into three subspaces. Afterwards, inside each subspace, the point which has the maximum distance to its corresponding facet is marked. Among the marked points, the point with maximum distance, called the furthest point, is selected as a new vertex of the convex hull. In the next step, two new facets are generated in such a way that each new facet involves the furthest point and one of the two vertices of the corresponding facet. Consequently, a new triangle is generated. Afterwards, the points inside the triangle are marked as the inner points and are removed from the set. Then the corresponding facet of the furthest point is removed to update the current convex hull. This procedure continues with the current convex hull and stops when no furthest point is identified. Fig. 3.6 illustrates the steps of Quickhull applied on a set of 20 points.

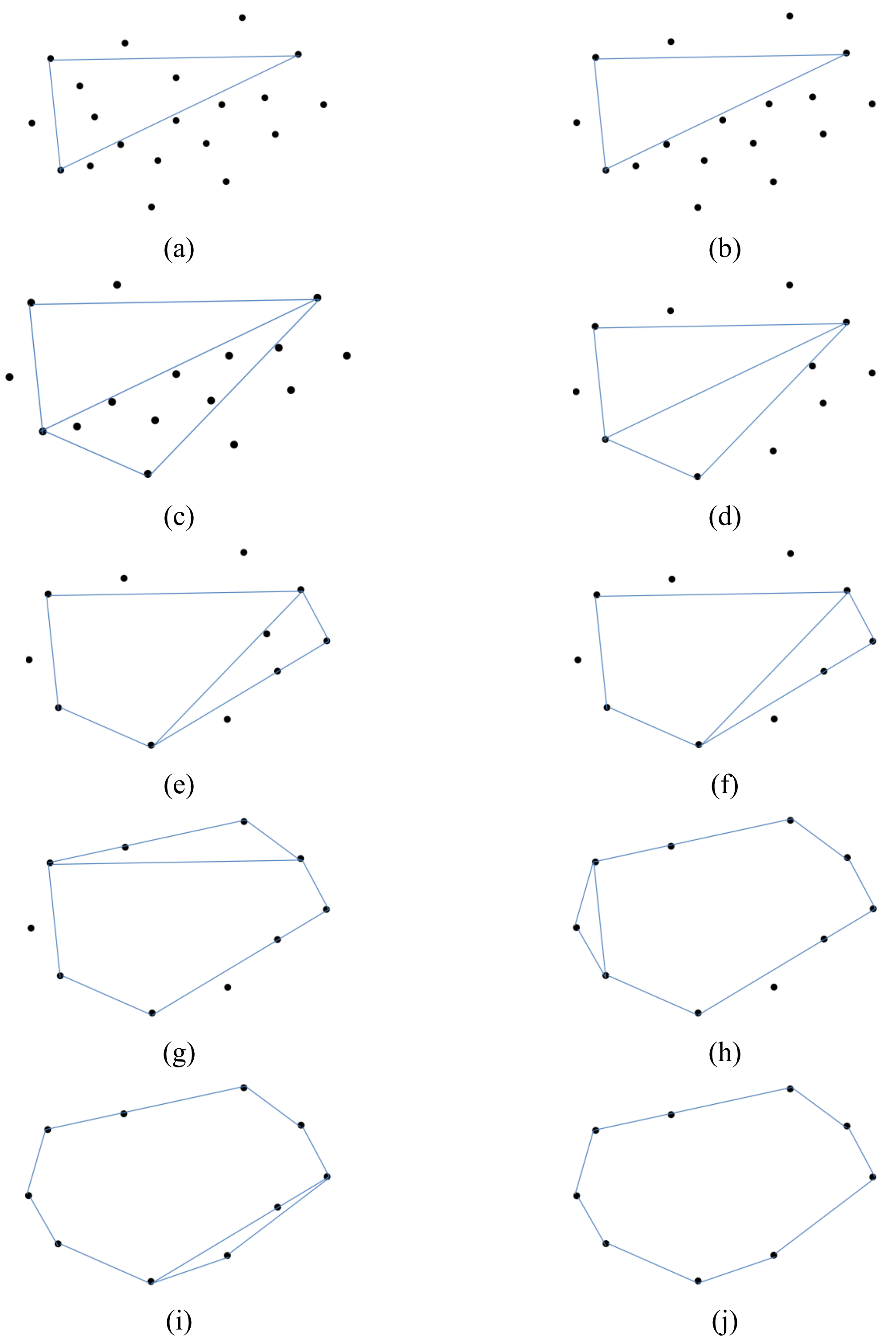


Fig. 3.6. The steps of Quickhull applied on 20 points.

As it can be observed in Fig. 3.6, in each step, the furthest point with respect to the current convex hull is identified and marked as a convex hull vertex. Afterwards, two new facets are generated and based on those, the inner points are removed from the underlying set. Then the corresponding facet of the furthest point is removed. This procedure continues until no new vertex is found.

From the time complexity point of view, for dimensions $d \leq 3$, Quickhull runs in time $O(n \log r)$, where n and r are the number of all points and the number of processed points, respectively. In the worst case, the time complexity is $O(n^2)$. For $d \geq 4$, Quickhull runs in time $O(nf_r/r)$, where f_r is the maximum number of facets for r vertices. Since $f_r = O(r^{\lfloor \frac{d}{2} \rfloor} / \lfloor \frac{d}{2} \rfloor !)$, for high dimensions, a massive number of facets would be generated for r vertices. Consequently, Quickhull is not feasible for high dimensions, both in terms of execution time and memory requirements.

3.3.4. A divide and conquer based convex hull algorithm

Preparata and Hong [9] proposed a convex hull algorithm based on the divide and conquer technique. This algorithm starts with sorting set S of points with respect to the first dimension denoted as x (i.e., the second dimension is denoted as y). Then S is divided into two equal size subsets S_1 and S_2 so that the first half of points is assigned to S_1 and the second one to S_2. The algorithm is recursively performed on S_1 and S_2. Sorting points produces a set of nonintersecting sub-convex hulls throughout the execution of the algorithm. Any standard convex hull algorithm in 2-dimensional space can be applied to obtain the sub-convex hull of each subset. Another phase of the algorithm is merging any two sub-convex hulls. Fig. 3.7 shows the steps of the divide and conquer based algorithm applied on a set of 20 points. As it can be seen in Fig. 3.7, the underlying set is divided into the subsets recursively. Generating sub-convex hulls and merging them are also done in a recursive manner. Sorting the points takes $O(n \log n)$ operations. The time complexities of generating a set of sub-convex hulls and merging them are $O(n \log n)$ and $O(n)$, respectively. To sum up, the time complexity of the algorithm is $O(n \log n)$. The extended version of the algorithm in 3-dimensional Euclidean space also takes $O(n \log n)$ time.

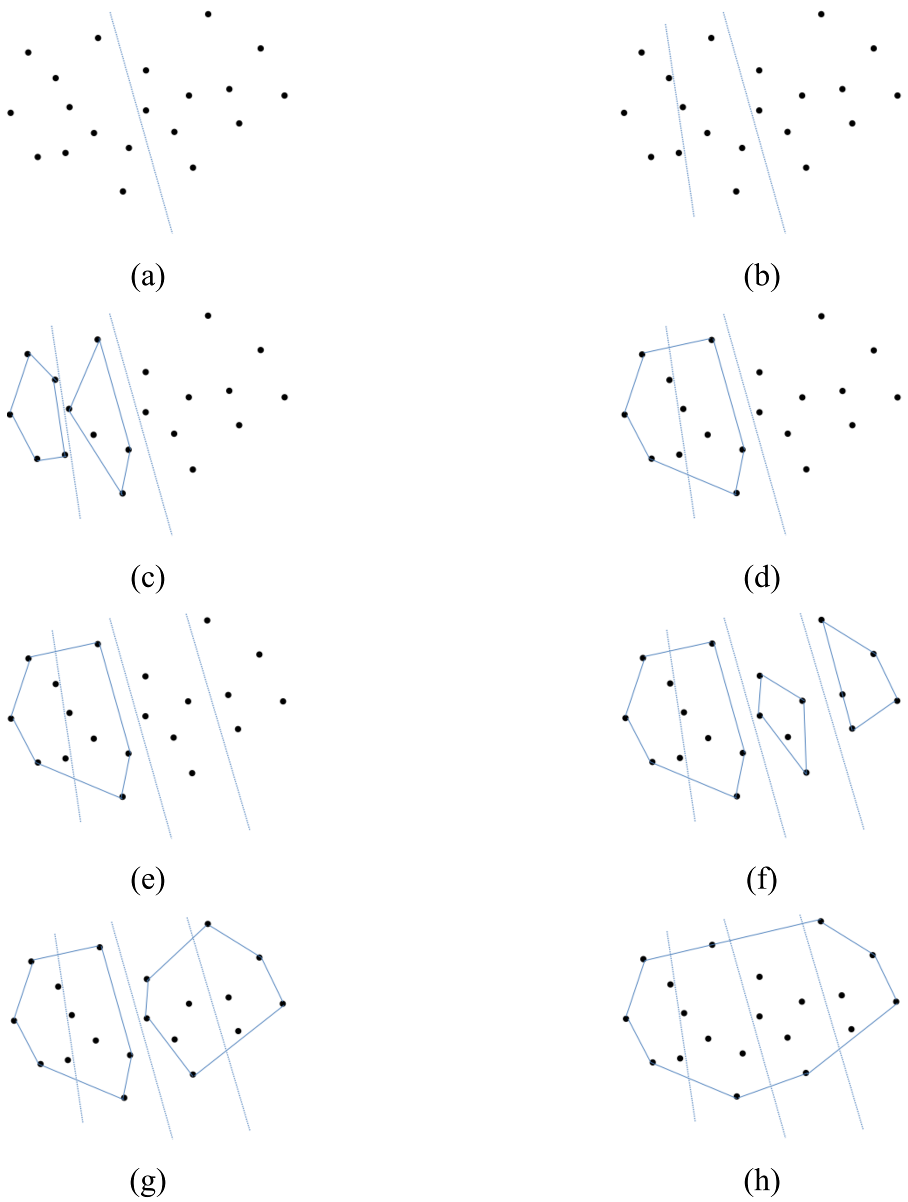
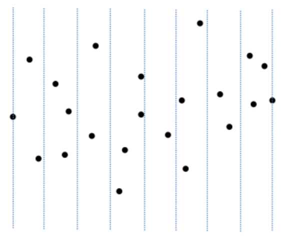


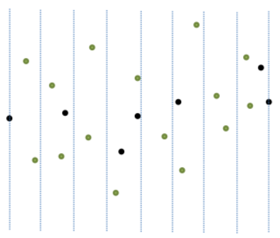
Fig. 3.7. The steps of the divide and conquer based algorithm applied on 20 points.

3.3.5. Approximation Algorithms for Convex Hulls

As mentioned earlier, there exists a subclass of convex hull algorithms called approximation algorithms where a subset of vertices of real convex hull is obtained. The approximation algorithms are proper for real time applications. Approximation algorithms are also suitable for statistical applications in which data observations are not accurate. Since the obtained real convex hull from inaccurate data is not the same as the one obtained from the accurate data, one can rely on an approximation convex hull. Authors in [10] proposed approximation algorithms for two and three dimensions. The basic idea behind these algorithms is selecting a subset S of the whole set containing n points and then applying any convex hull algorithm on

S to obtain an approximation convex hull. The key point in these algorithms is proposing a method based on data partitioning to select the subset S. In two dimensions, firstly the minimum and maximum points with respect to x axis (i.e., the first dimension) are included in S. In second step, the points are partitioned into k equally spaced strips with respect to x axis. In third step, in each strip, the minimum and maximum points with respect to y (i.e., the second dimension) are included in S and finally a convex hull algorithm is applied on S to obtain an approximation convex hull. The steps of the algorithm applied on 22 points (i.e., $n = 22$) with 8 equally spaced strips (i.e., $k = 8$) are illustrated in Fig. 3.8. As it can be seen in Fig. 3.8(c), one outer point which is a vertex of the real convex hull has not been identified by the approximation algorithm. The time complexity of the approximation algorithm in two dimensions is $\theta(n + k)$. In a 3-dimensional Euclidean space, firstly, the minimum and maximum points with respect to both x and y axis are included in S. In the second step, the points are partitioned into a $k \times k$ grid of squares obtained with respect to x and y axis. In the third step, in each square, the minimum and maximum points with respect to z axis (i.e., the third dimension) are also included in S and finally any convex hull algorithm for three dimensions can be applied on S to result in an approximated convex hull. The time complexity of the approximation algorithm in three dimensions is $\theta(n + k^2 \log k)$.


(a)


(b)

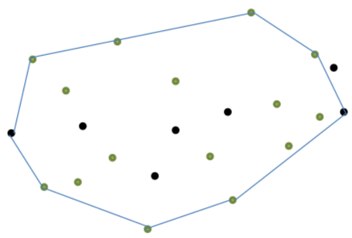

(c)

Fig. 3.8. The steps of the approximation convex hull algorithm applied on 22 points.

3.3.6. Online convex hull algorithms

Unlike offline algorithms, online convex hull algorithms process points at a time in the sense that the current convex hull is gradually updated whenever a new arriving point is received. The online convex hull problem can be described as follows: firstly, the convex hull of a given N points $p_1, \cdots, p_N$ is identified and then the convex hull is updated by the new arriving point p_i. Throughout the convex hull update process, three cases may happen. In the first case, the new arriving point is an inner point meaning that the new point is located inside the current convex hull. In this case, the new point is rejected and the current convex hull is kept unchanged. In the second case, the new point is an outer point meaning that the point is located outside the current convex hull. This situation causes elimination of some vertices of the current convex hull which have been already converted to inner points. In the third case, the new point is an outer point but does not affect the vertices of the current convex hull. In this case, the new point is appended into the list of vertices.

One of the earliest online convex hull algorithms in 2-dimensional spaces was proposed by Preparat and Shamos [11]. The main idea behind their algorithm is updating the current convex hull benefiting from two support lines (i.e., left and right support lines). A support line is a line which passes through the new arriving point and one of the vertices of the current convex hull so that the remaining points lie in the same side of the line. If no support line is founded, it means that the new point is an inner point. In case that the new point is an outer point, all vertices between two support lines are marked as inner points and will be eliminated in the update process. Fig. 3.9 shows the update process of the online convex hull algorithm based on the support lines.

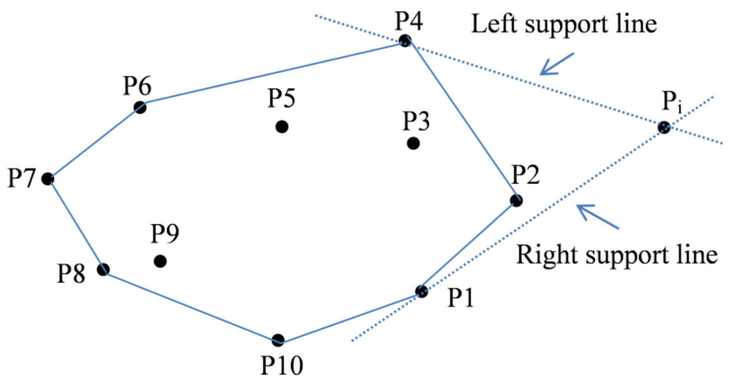


Fig. 3.9. The update process of the online convex hull algorithm based on support lines.

The online algorithm takes $\theta(n \log n)$ time for obtaining the convex hull of n points with $\theta(\text{logn}\, n)$ update time. The extended version of this algorithm in 3-dimensional Euclidean space also takes $\theta(n\, \text{logn}\, n)$ time.

3.3.7. Randomized algorithms

Unlike deterministic convex hull algorithms, randomized algorithms construct the structure of convex hull in a random manner. They are similar to online algorithms in the sense that the convex hull is incrementally formed due to processing random points at a time. In online algorithms, however, the initial convex hull is formed based on a limited number of points while in random algorithms, all points are available to be processed. Therefore, some information of the resulting convex hull can be obtained before it is constructed.

In randomized incremental algorithms, a convex hull is incrementally constructed in three steps. In the first step, an unprocessed random point is selected. In the second step, the boundary of visible facets with respect to the point is identified. This boundary is called horizon ridges. Afterwards, new facets are generated using the point and the horizon ridges. Finally, the visible facets as well as the inner points are eliminated. This procedure is repeated until no unprocessed point remains [8]. In this method, the convex hull is presented as a set of a finite number of extreme points which are the convex hull vertices. Hence, this representation of the convex hull is known as vertex representation or V-representation [65]. The time complexity of such randomized algorithms is $O(n \log n)$ for $d \leq 3$. Convex hull can also be defined in terms of the intersection of a finite number of half-spaces in the form of a system of linear inequalities as follows:

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1d}x_d & \leq & b_1 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{md}x_d & \leq & b_m \end{array}$$

Each inequality denotes a half-space where m and d are the number of halfspaces defining the convex polytope and the dimension, respectively. The concise form of the above system can be represented in the form of matrix inequality as $\boldsymbol{Ax} \leq \boldsymbol{b}$ where each row of $\boldsymbol{A}$ and $\boldsymbol{b}$ together correspond to the definition of a supporting hyperplane of the convex polytope in terms of normal and offset, respectively. This presentation of convex hull is known as half-space representation or H-representation [65].

One of the earliest randomized convex hull algorithms which provides H-representation of the convex hull was proposed by Clarkson and Shor [12]. In each iteration of the algorithm, an unprocessed random half-space is added into the current convex polytope by intersecting it with the previous half-spaces enveloping the convex polytope. The algorithm takes $O(m + n\log n)$ expected time and $O(n)$ space in the worst case, where n and m denote the number of points and the number of intersecting pairs reported.

3.4. Introduction of convex hull algorithms in higher dimensions

In higher dimensions $d \geq 4$, two main methods are considered to identify the convex hull: 1-gift wrapping method [5, 66] 2- beneath-beyond method [63].

The idea behind the gift wrapping method is constructing the convex hull by starting a facet and finding the adjacent facets. This procedure is iteratively conducted for each new identified facet. Since each facet is linked to its corresponding adjacent facets, the way in which the convex hull formed is like wrapping around a convex polytope in d-dimensional space. Jarvis's march algorithm introduced in Section 3.3.2 is a special case of the gift wrapping method in a 2-dimensional space. Like Jarvis's march, the gift wrapping method in high dimensions is an output-sensitive method whose time complexity also depends on the size of output f which can be the number of facets of the generated convex hull. Therefore, the time complexity of the gift wrapping method in high dimensions is $O(nf)$. Based on the upper bound theory, the number of generated facets is $O(v^{\lfloor \frac{d}{2} \rfloor})$ where v is the number of convex hull vertices. In the worst case where $n = v$, the time complexity of the gift wrapping method is $O(n^{\lfloor \frac{d}{2} \rfloor + 1})$. An improved version of the original gift wrapping method was proposed by Seidel [66] where the algorithm takes $O(n^2 + f\log n)$ time. Based on the upper bound theory, in the worst case, the time complexity of the algorithms is $O(n^{\lfloor \frac{d}{2} \rfloor}\log n)$.

The Beneath-beyond method is considered as an incremental approach constructing the convex hull by adding one point into the current convex hull at a time. The update process of the current convex hull includes adding new facets into the current convex hull and removing the visible facets with respect to the new point. The Quickhull algorithm, as a deterministic incremental algorithm stated in Section 3.3.3, and also the randomized incremental algorithms, described in Section 3.3.7, where the convex hull is presented in terms of vertices (i.e., V-representation) are special beneath-beyond methods in a 2-dimensionl Euclidean space. The time complexity of the beneath-beyond method in high dimensions is $O(n\log n + n^{\lfloor \frac{(d+1)}{2} \rfloor})$ with $O(n^{\lfloor \frac{d}{2} \rfloor})$ space [63]. The method was improved by [67] through derandomizing the randomized incremental algorithm proposed by [12] . The time complexity of the improved version is $O(n \log n + n^{\lfloor \frac{d}{2} \rfloor})$. Since both time and memory requirement increase exponentially with respect to dimension $d \geq 4$ with a fixed number of samples, applying the traditional convex hull algorithms in very high dimensions with a huge number of samples is not feasible.

As the state of the art, one of the recent work done to overcome these challenges in high dimensions is the proposal of an approximation convex hull algorithm by Wangs and *et al.* [68]. The proposed algorithm represents the convex hull in terms of vertices with the aim of including extreme points in the training set for online adaptation process of SVM models. The algorithm is based on samples partitioning where for each partition, the corresponding sub-convex hull is obtained and then the union of vertices of all sub-convex hulls is considered as the set of vertices of an approximation convex hull. The algorithm results an approximation convex hull throughout three steps. In the first step, $d + 1$ samples are selected as the vertices of the initial convex hull so that these samples can constitute a d-simplex as large as possible where d is the dimension. Since the d-simplex has $d + 1$ facets, it divides the space into $d + 1$ partitions. For example, Fig. 3.10 illustrates a 2-simplex which is translated into a triangle. As it can be seen in Fig. 3.10, the vertices of the 2-simplex are $\{x_1, x_2, x_3\}$. Assume o is the center of the 2-simplex. As can be seen in Fig. 3.10, the rays ox_1, ox_2 and ox_3 divide the samples outside the 2-simplex into three partitions named P_1, P_2 and P_3.

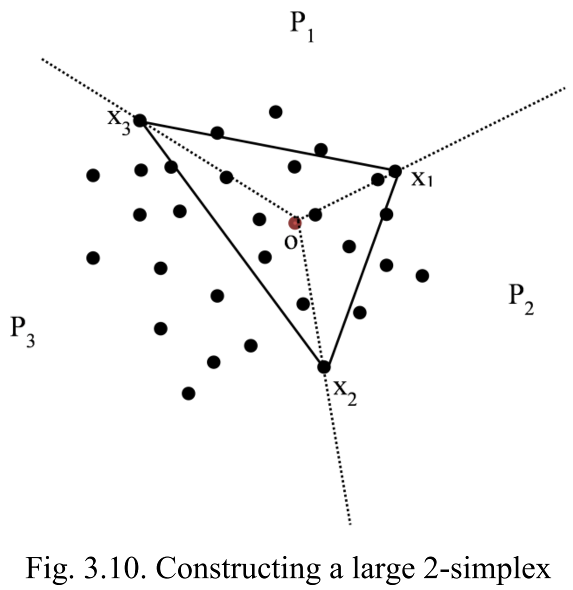


Fig. 3.10. Constructing a large 2-simplex

In the second step, each partition whose number of samples is greater than a user-defined threshold L is divided into d new partitions based on the furthest sample to the corresponding facet of the partition. Afterwards, the furthest sample is appended into the list of convex hull vertices. This procedure continues until there exists no partition whose number of samples exceeds L. For example, in Fig. 3.11, partition P_3 is divided into two new smaller partitions named P_{31} and P_{32} based on the furthest sample named x_4 to the facet x_2x_3. As it can be seen in Fig. 3.11, the facet x_2x_3 is removed and two new facets x_2x_4 and x_3x_4 are generated.

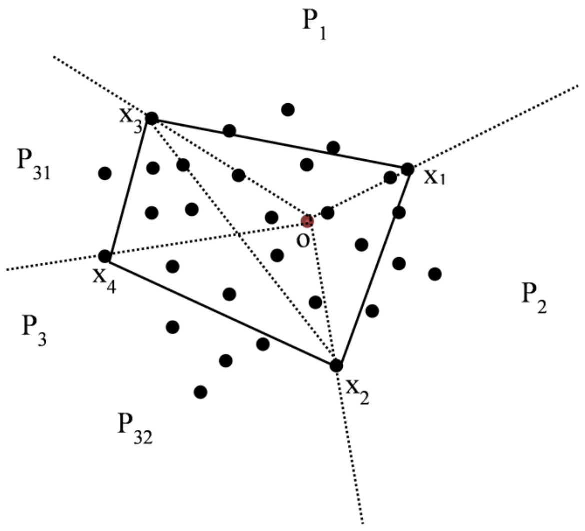


Fig. 3.11. Partitioning step

At the end of this step, each facet is considered as a sub-convex hull including d vertices. In the third step, each sub-convex hull is tried to be expanded by identifying the furthest sample to the whole sub-convex hull and appending it into the list of the sub-convex hull vertices. This procedure continues until an approximation convex hull with at most M vertices is obtained, where M is a user-defined threshold. The time complexity of the algorithm is at most $O(nd^4)$ where n is the number of samples. Although the algorithm can cope with the time complexity in high dimensions, as it will be shown in Section 4.4.1, it presents some points as vertices of the approximation convex hull that do not belong to the vertices of the corresponding real convex hull.

3.5. Conclusions

Convex hull, as one of the fundamental concepts in computational geometry, has been applied in a wide variety of applications such as data selection, image processing, pattern recognition, collision detection, file searching, cluster analysis, etc. Convex hull algorithms can be considered from three points of view: deterministic or randomized, in terms of vertices order, real or approximation, depending if all convex hull points are found, or not, and offline and online, depending on the use of data.

To the best of our knowledge, the standard algorithms in two and three dimensions that have been proposed by far, present the real convex hull in the time complexity $O(n \log n)$. Moreover, the proposed standard real algorithms in high dimensions (i.e., more than three dimensions) take $O(n^{\lfloor \frac{d}{2} \rfloor})$ time and space in the worst case; where n and d denote the number of samples and the dimension, respectively. Since in practice, applying the standard real algorithm on a huge number of samples in high dimensions is not feasible, approximation algorithms have received much attention to cope with challenges in high dimensions. As one of the state of the art, Wangs and *et al.* [68] proposed an approximation algorithm in high dimensions with the time complexity $O(nd^4)$. Since this algorithm marks some points as the convex hull vertices which do not belong to the vertices of the real convex hull, a randomized approximation convex hull algorithm for high dimensions is proposed in Section 4.3. The proposed algorithm not only overcomes the time and space complexity in high dimensions, but also presents a subset of informative vertices of the corresponding real convex hull.

4. A convex hull-based data selection method for data driven models

4.1. Introduction

As stated in Section 3.4, the standard convex hull algorithms suffer from high time and space complexity in high dimensions (i.e., the latter being $O(n^{\lfloor \frac{d}{2} \rfloor})$ where n and d are the number of samples and dimensions, respectively.). Hence, in practice, they cannot be applied in high dimensions. For example, if Quickhull [8], a real deterministic convex hull algorithm described in Section 3.3.3, is applied to an artificial dataset including 1000 uniformly distributed random samples in just 9 dimensions, in a computer with Ubuntu Linux OS, Intel Core i5 processor and 4 Gigabytes of RAM, one can see that it suffers from insufficient memory.

A small number of efforts have been done to overcome these problems in high dimensions. The approximation algorithm proposed in [68] has significantly reduced the time complexity to $O(nd^4)$. The problem of the algorithm, as reported previously, is that some points which are marked as convex hull vertices do not belong to the real convex hull.

This chapter introduces a randomized approximation convex hull algorithm called ApproxHull , with the aim of being applied as a filter data selection method to design data driven models. ApproxHull not only is capable of being applied on large size data sets in high dimensions but also presents a set of informative vertices which all belong to the real convex hull. The main application of ApproxHull in the data selection phase is constructing a training set that reflects the whole input-output range of the design data. To do that, the training set incorporates the convex hull points obtained from ApproxHull, as well as some random points from the whole data.

This chapter is organized as follows. A review on instance selection methods is presented in Section 4.2. ApproxHull is introduced in Section 4.3. To verify and evaluate the performance of the algorithm, a number of experiments were carried out. Section 4.4 explains the simulation results obtained. The run time and memory requirements of ApproxHull are discussed in Section 4.5 and 4.6, respectively. Finally, some conclusions are given in Section 4.7.

4.2. A review on instance selection methods

In many machine learning and data mining problems two basic tasks have to be considered: feature selection and instance selection. The former denotes choosing a subset from all available features so that the selected subset has the strongest relation to the model output and yields improved model performance. The latter refers to sample selection where we are interested in selecting a subset of useful and informative data samples (denoted by S) among all existing data samples (denoted by T). The goal is that the model obtained using S can maintain or even exceed the performance level (for instance, accuracy) that would be attained using T. The instance selection process not only helps decreasing the run time of the training process but also has the benefit of reducing the memory requirements of learning algorithms. This is important when classification or regression tasks rely on existing large-size training sets.

Generally speaking, instance selection methods can be classified from the search direction and selection criterion points of view. Regarding the search direction, the methods are categorized as incremental or decremental. In the former, the selection process starts with $S = \emptyset$ and progresses iteratively by inserting selected samples from T into S. In the latter, in contrast, the selection process starts with $S = T$ and superfluous samples are discarded from S in an iterative manner. Finally, for both methods, we have $S \subset T$ by the end of the selection process [69, 70].

From the selection criterion point of view, instance selection methods are classified as wrapper or filter methods. Wrapper methods use a model as a selection criterion, where the performance of the model is evaluated based on a subset of samples iteration by iteration to select those samples which have the most contribution on the model accuracy. Most works found in literature on wrapper methods relate to classification tasks. Unlike wrapper methods, filter methods employ a model independent selection function to choose informative samples [69]. This means that the accuracy of the model does not have any contribution in the selection criterion; instead, a selection rule is applied.

Fig. 4.1 shows the two main classes of instance selection methods along with their subgroups. The following details wrapper and filter methods along with some related works.

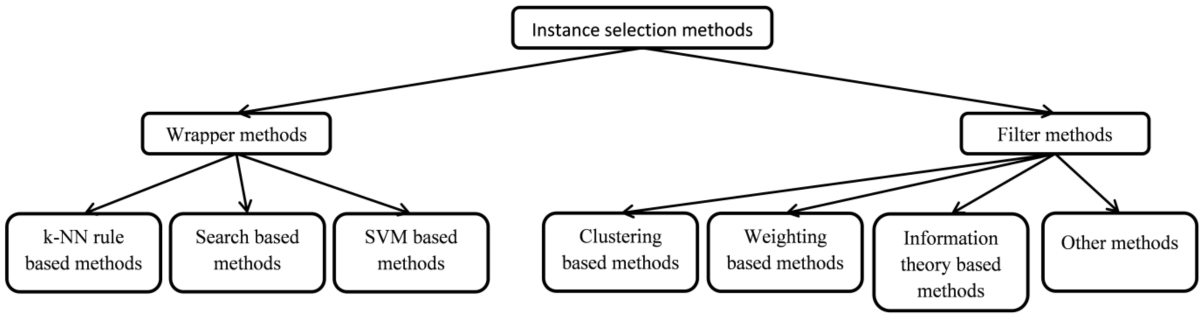


Fig. 4.1. Classification of instance selection methods

4.2.1. Wrapper instance selection methods

Collectively, wrapper methods may be further subcategorized into three groups. The first gathers methods which are based on $k-NN$ (K Nearest Neighbors) classifiers [71], whereas the second group involves a broad class of wrapper methods that can be based on any classifier. The second group which is mostly based on search algorithms tries to find an optimal set S from T to keep the classifier in a desirable level of accuracy. The third group benefits from SVM [27] where they are applied to constitute set T_s, containing only support vectors of T , which is used for $K-NN$ classifiers. The following addresses the three groups of wrapper instance selection methods.

4.2.1.1. $K-NN$ rule based methods

One of the earliest incremental method called CNN (Condensed Nearest Neighbor rule) was proposed by Hart [72]. It focuses on misclassified samples as critical samples that matter the most to the $k-NN$ classifier to ensure that unlabeled samples which are similar to the misclassified ones are correctly classified [73]. This method constitutes set S from set T by randomly selecting samples of each class. Afterwards, all samples in T are classified by the $1-NN$ rule using S as the training set. Then all misclassified samples are included in S to ensure that the unlabeled samples similar to the misclassified samples are correctly classified. The main disadvantage of this method is to allow noisy samples to be included in S since they are mostly misclassified based on their neighbors. As another version of the CNN, Ritter *et al.*[74] proposed the SNN (Selective Nearest Neighbor rule). In this method, set S is composed in such a way that, for each sample in set T, its nearest neighbor can be found in set S. Hence, a sample of T is correctly classified based on the $1-NN$ rule using S. In classification tasks, the samples which are close to the decision boundary involve useful information to discriminate classes from each other. Since in the CNN and some of its variations samples are randomly selected from each class without considering their position with respect to the decision boundary, the boundary samples may be selected occasionally. To deal with this problem, some extended versions of the CNN were proposed.

Gowda and Krishna [75] proposed a method in which the set S is formed using the concept of *mutual nearest neighborhood* for selecting the boundary samples. If for two samples x_i and x_j, x_j being the m^{th} nearest neighbor of x_i and correspondingly , x_i being the n^{th} nearest neighbor of x_j, the mutual neighborhood value of sample x_i with respect to sample x_j is defined as $MNV(x_i, x_j) = m + n$. The proposed method, firstly, measures MNV for each sample of T with respect to its nearest sample in the opposite class. Samples that are close to the decision boundary have a low MNV. Afterwards, the samples of T are sorted in ascending order based on their MNV and the first sample of T is inserted in S; the remaining samples are classified based on the $1 - NN$ rule using S. Then, misclassified samples are included in S. This process is repeated iteration by iteration until no misclassified sample is detected. As another extended version of the CNN, GCNN (Generalized Condensed Nearest Neighbor rule) was introduced by Chou *et al* [76]. In this method, sample x as a prototype of T is included in S if it violates the absorption criterion $\|x - q\| - \|x - p\| > \delta$ where p is the nearest neighbor of x in the class to which x belongs and q is the nearest neighbor of x in the opposite class. δ is a user defined threshold and $\|.\|$ denotes the 2-norm operation.

So far, all introduced methods were based on $1 - NN$ rule. The authors in this literature have also proposed the methods based on $k - NN$ where $k > 1$. One of the earliest decremental method known as ENN (Edited Nearest Neighbor rule) using $k - NN$ rule was proposed by Wilson [77]. In this method, $3 - NN$ rule is applied to remove the noisy samples from T in such a way that each sample of T is classified using three nearest neighbors where the majority class is considered for labeling the sample. Then, the misclassified samples are removed from T. Finally, the reduced set T is considred as set S to classify new samples using $1 - NN$ rule. An extended version of the ENN called $all\ k - NN$ was introduced by Tomek [78]. In this method, the ENN is repeated for different values of k on set T and those samples which are incorrectly classified for at least one value of k are removed from T.

A family of five incremental methods coined DROP1 to DROP5 (Decremental Reduction Optimization Procedure, 1 to 5) can be seen in [70]. These methods, which are based on the $k - NN$ rule, remove noisy samples using the *associates* concept. The associates of sample x

are those samples for which x is one of their k nearset neighbors. These methods remove sample x from T whenever its associates are correctly classified without considering x. This is because, in most cases, the majority of associates of a noisy sample belong to the opposite class. Brighton and Mellish [79] introduced a method known as ICF (Iterative Case Filtering) which applies two concepts, *reachability* and *coverage*, corresponding to the neighborhood and associate sets, respectively. The reachable set does not have a fixed size; instead, it is bounded by the number of nearest samples from the opposite class. The ICF method focuses on removing noisy and superfluous samples from T. At the first stage, it applies the method ENN proposed by Wilson [77] to remove noisy samples. Afterwards, it tries to discard the superfluous samples relying on reachable and coverage sets. In this method, each sample x which meets the condition $|reachable(x)| > |coverage(x)|$ is discarded from T. It is translated to this fact that sample x is far from the decision border. Hence x, as a superfluous sample, can be removed from T since their neighbors can correctly classify new arriving similar samples to the x.

4.2.1.2. Instance selection methods based on search algorithms

So far, all the introduced methods were based on the $k-NN$ classifier model. In this literature, there is another group of wrapper instance selection methods which are mostly based on search algorithms. In this group of wrapper methods, the instance selection process is carried out by evaluating an arbitrary classifier model iteratively. Among search algorithms, evolutionary algorithms, as general-purpose search algorithms, have received much attention in the literature of instance selection process. Specifically, GA-based methods have been considered in this domain (for further information about GA, please consult Section 2.6). In this group of instance selection methods, each chromosome $\boldsymbol{c}$ corresponds to a subset S of T which is commonly presented by a binary string as $\boldsymbol{c} = [0,1,1,0,1,1,0,0,\cdots,0]$ so that $|\boldsymbol{c}| = N$ where N is the size of T. Each element $\boldsymbol{c}_j$ of $\boldsymbol{c}$ for $j = 1,2,\cdots,N$ denotes the presence or absence of jth sample of T in S [69, 80, 81].

When GA is customized for instance selection problem, the fitness value of a chromosome representing a subset S of T is computed in terms of the model accuracy and the percentage of instance reduction as Eq. (4.1) [80, 81].

$$Fitness(S) = \lambda . C_{rate}(S) + (1-\lambda). P_{reduction}(S) \tag{4.1}$$

where C_{rate} and $P_{reduction}$ denote the classification rate obtained by the evaluation of the model using S and the percentage of instance reduction of S with respect to T, respectively. In order to compromise between the model accuracy and the size of S, λ is usually set to 0.5. The percentage of instance selection is also computed by Eq. (4.2) [80, 81].

$$P_{reduction}(S) = \frac{\#(T) - \#(S)}{\#(T)} \, . 100 \tag{4.2}$$

where $\#(.)$ denotes the size of underlying set.

In the literature of evolutionary instance selection, some variants of GA have been applied from the most classic version to the most complex versions. Six evolutionary instance selection methods including Generational Genetic Algorithm (GGA), Steady-State Genetic Algorithm (SSGA), CHC Adaptive Search Algorithm, Intelligent Genetic Algorithm (IGA), Steady-State Memetic Algorithm and Population-Based Incremental Learning (PBLI) have been employed in [80, 81]. Moreover, some works that use GA in the application of instance selection to improve the accuracy of $k - NN$ classification can be seen in [82-84].

As other works related to search based instance selection methods, the authors in [85, 86] introduced the instance selection methods based on Tabu search [87] to find the best subset S from T for $1 - NN$ classification. Tabu search, so called adaptive memory programming, is considered as a very efficient and straightforward optimization method to solve NP-hard problems.

Tabu search benefits from short-term memory, called Tabu list, and neighborhood exploration which make it distinctive from other search methods in terms of low computational cost and better space exploration. Tabu search method starts with an initial solution S_i. Afterwards, a set of possible moves with respect to the current solution S_c is considered, in a sense that each move is a neighbor solution of the S_c , with a little bit modification in S_c. In the next step, all neighbor solutions are evaluated and the best one is selected considering those Tabu moves which have been previously inserted into the Tabu list (i.e., a short-term memory which is usually managed by FIFO policy to keep track the recently examined solutions). Finally, the Tabu list is checked to see if the best solution already exists within. If not, it will be inserted in the Tabu list. This process continues until a termination criterion is met. The termination criterion is:

1. exceeding a given number of iterations or
2. when there is no improvement with respect to the overall best solutions throughout a given number of consecutive iterations.

As a further effort in applying search based methods, authors in [88] proposed Backward Sequential Edition method (BSE) as a decremental selection method benefiting from backward sequential search [89] to select an optimum set S from T. This method iteratively discards a sample from the current set S which has the minimum contribution on the classification accuracy. This procedure stops when the classification accuracy starts to decrease. Since this method is based on backward sequential method, the eliminated samples have no chance to be reconsidered in the selection process in a forward manner. In order to deal with this disadvantage of BSE, authors in [90] introduced the Restricted Floating Object Selection (RFOS) which relies on sequential floating search [91] in a restricted manner due to its extreme run time that allows the eliminated samples to be reconsidered in a forward direction.

4.2.1.3. SVM based methods

In this group of methods, SVMs are applied to reduce the size of T. Authors in [92] proposed an SVM based instance selection method. This method uses the algorithm DROP2 proposed in [70] to discard noisy samples from T_s. T_s contains the support vectors obtained by applying SVM on T. Then, a new sample x is classified by $1-NN$ rule using T_s.

The SVM based method proposed in [93], uses $k-means$ clustering algorithm to cluster T_s and then each support vector v of T_s is assigned a weight based on the proportion of its class label in the cluster to which v belongs. It is defined as $\frac{N(class(v)}{N_c}$ where $N(class(v))$ and N_c denote the number of samples in the cluster which have the same class label as v and total number of samples in the cluster of v. Afterwards, a new sample x is classified using $k-NN$ rule in such a way that firstly the weights of the nearest neighbors of the same class are summed up and then the class label corresponding to the maximum summation is considered as the class label of x.

4.2.2. Filter instance selection methods

As mentioned earlier, unlike wrapper methods, in filter methods, a classifier independent criterion is applied to select instances. Mainly, filter methods can be organized into three groups including clustering based methods, weighting based methods and information theory based methods. Besides these groups, a few methods were proposed which do not belong to any specific group. The following describes three main groups of filter methods as well as some other methods.

4.2.2.1. Clustering based methods

Among all efforts done in filter methods, some clustering based methods have been proposed with the aim of obtaining a set of prototypes so that each of them is a representative of a group of original instances in T. The idea behind this group of filter methods is clustering the original instances and then considering the centers of obtained clusters as a set of prototypes representing all instances in T where finally unlabeled samples are classified based on the obtained set of prototypes. Authors in [94] introduced Generalized Modified Change Algorithm (GMCA) in which a merging strategy is exploited to merge two same-class nearest clusters and then the new center is considered as a new prototype. The method proposed in [95] called Nearest Subclass Classifier (NSB) clusters each class separately using Maximum Variance Cluster Algorithm [96] where the number of clusters is different from class to class as the distribution of samples may be different from class to class. In [97], a method known as Object Selection by Clustering (OSC) was presented to select both border and interior instances using clustering. In this method, the centers of homogeneous clusters are considered as prototypes representing the interior instances while from heterogeneous clusters, the border instance P is selected. Instance P in cluster C_j is a border instance if it is the nearest neighbor of another instance in cluster C_j with different class label.

4.2.2.2. Weighting based methods

Weighting based methods, as another group of filter methods, work as follows: In the first step, a weight is assigned to each instance; then, a percentage of instances based on a user-defined threshold on their weights is selected as a subset S of T. Authors in [98] proposed a new approach based on instance weighting where weights σ_i (i.e., corresponding to the i^{th} sample) are obtained by minimizing cost function $J(\sigma)$ using a gradient descent method. $J(\sigma)$ is a function of $Weighted\,Prototype$ dissimilarity between each instance and its corresponding nearest neighbor, and also between the instance and its nearest enemy (i.e., the nearest neighbor of the opposite class). Those instances whose weights are larger than a user-defined threshold are removed from the whole training set T.

As another weighting based method, Prototype Selection by Relevance (PSR) was introduced in [99]. The idea behind the method is based on the fact that some instances in T which belong to the same class are more relevant than the others and they should be selected. Hence, this method assigns a weight to each instance reflecting its amount of relevance. The relevance of each instance is computed based on the average similarity of the instance to the others. Heterogeneous Value Difference Metric (HVDM) [100] is applied as the similarity function. Afterwards, r most relevant instances of each class are selected and through them, some border instances bringing useful information of class discrimination regions are also chosen.

4.2.2.3. Information theory based methods

Recently another group of filter instance selection methods specifically for regression tasks has received attentions. This group of methods benefits from information theory to select a subset S from T so that S contains the most informative samples which have the most contribution in model fitting. Authors in [101] proposed a Mutual Information based method for instance selection aimed to be applied in time series prediction. In fact, MI between two random variables measures how much information of one of two variables reduces uncertainty of the other. In this work, MI was applied to compute how much information can be obtained about the target variable using the information of input variables in the form of input patterns. The basic idea behind the work is that if the amount of MI loss due to the absence of an input pattern $\boldsymbol{x}_i$ in the whole training set T is similar to that due to absence of each of its k nearest neighbors, the input pattern $\boldsymbol{x}_i$ should be selected for subset S. As another effort in exploiting of information theory in instance selection for time series prediction, a MI based methodology was presented in [53]. The idea behind this work is selecting those instances which share a significant amount of MI with the current predicted instance at each step of the prediction horizon. In this method, at each step of prediction horizon, the current input patterns are ranked based on MI between them and the current predicted instance. Then, those input patterns whose ranks are greater than a user-defined threshold constitute a subset S of the current training set. Ferreira in [13] proposed an unsupervised selection method based on Shannon's information entropy [51, 102] which measures the amount of information content of the data. In this method, firstly, the probability of the presence of each instance in T is estimated using a kernel based density estimation proposed in [56] known as Parzen window method. Afterwards, a fitness value based on information entropy is assigned to each instance where this value reflects the amount of informativeness of the instance. Finally the subset S with size k is selected from T using SUS method [103].

4.2.2.4. Other methods

Besides the three groups of filter methods, some works have been done which do not belong to any specific group. Authors in [104] presented POP (Pattern by Ordered Projections) method to remove interior instances and select some border instances based on the $Weakness$ concept. An instance P_i of class C_j is a border instance if P_i is the nearest neighbor for an instance of another class C_k, otherwise it is an interior instance. The concept $Weakness$ indicates how many times an instance is not a border instance with respect to each of its features' values. This method removes the irrelevant instance P, which is the instance whose $Weakness$ is equal to m , where m denotes the number of features.

The method proposed in [105] applies kd-trees structure [106] which are binary trees to select a subset S of T. Based on kd-trees structure, the root of the tree includes all instances in T. Afterwards, T is partitioned into two groups so that one of them corresponds to the left child and the other corresponds to the right one. The separation of T is performed using the $Maxdiff$ criterion. To calculate $Maxdiff$, first the feature with maximum distance along with consecutive samples (i.e., samples are sorted in ascending order with respect to the feature) is considered and then the value of the corresponding feature of the sample which has the maximum distance with its successor is considered as a pivot to split T. Those samples whose values of the corresponding feature is less than or equal to the pivot constitute the left child and the remaining forms the right one. The $Maxdiff$ criterion is satisfied if the pivot value is greater than a user-defined threshold, otherwise the variance along features is considered. If the maximum variance is greater than another user-defined threshold, the mean value through the corresponding feature is considered as a pivot. This procedure is repeated for each child. In the case that neither $Maxdiff$ criterion nor the maximum variance is satisfied, the algorithm is terminated and the samples which are located in the leaves of the tree are considered as a subset S of T.

Regarding the design of ANNs and SVMs as two examples of well-established data driven machine learning approaches for classification and regression tasks, some filter instance selection methods including Principal Components Analysis (PCA), convex hull and decision tree have been proposed [68, 107-110]. In the design phase of such models, it is very important that subset S covers the whole input-output range in which the underlying process is modeled. To achieve this goal, convex hull algorithms can be employed to identify the boundary points reflecting the whole range of data.

4.3. ApproxHull: A randomized approximation convex hull algorithm for high dimensions

In this Section, ApproxHull as a randomized approximation convex hull algorithm for high dimensions is introduced, providing a subset of all possible vertices of the corresponding real convex hull in a stochastic manner. ApproxHull, which was inspired by Quickhull [8] (please see Section 3.3.3) tries to identify some informative vertices of the real convex hull, relaying on two fundamental concepts hyperplane [111, 112] and convex hull distance [68]. Hence before addressing ApproxHull, these two concepts are explained in the following sections.

4.3.1. Hyperplane

Any hyperplane in a d-dimensional Euclidean space partitions it into two subspaces; positive and negative subspaces. Any point in the positive subspace has a positive distance to the hyperplane while the points located in the negative subspace have a negative distance to the hyperplane. Computing the equation of a hyperplane based on some predetermined points, which lie on the hyperplane, is intensively applied in computational geometry. Some convex hull algorithms like Quickhull need to compute the corresponding hyperplane equations of the current convex hull's facets to find the next vertices. In the following we shall describe how these equations can be obtained, starting by introducing the distance of a point to an hyperplane.

4.3.1.1. Hyperplane distance

Suppose $\boldsymbol{p} = [p_1, p_2, \dots, p_d]^T$ is a point, $\boldsymbol{F}$ is a d-vertex facet (each facet of d-dimensional convex hull involves exactly d vertices), and H is the corresponding hyperplane of facet $\boldsymbol{F}$ in a d-dimensional Euclidean space. The general equation of an hyperplane H is given as:

$$a_1x_1 + a_2x_2 + \cdots + a_dx_d + b = 0 \tag{4.3}$$

where $\boldsymbol{n} = [a_1, a_2, \dots, a_d]^T$ and b are the normal vector and the offset of H, respectively.
The normalized distance from point $\boldsymbol{p}$ to hyperplane H is computed by Eq. (4.4).

$$d_s(\boldsymbol{p}, H) = \frac{a_1 p_1 + a_2 p_2 + \cdots a_d p_d + \mathrm{b}}{\sqrt{{a_1}^2 + {a_2}^2 + \cdots {a_d}^2}} \tag{4.4}$$

4.3.1.2. Hyperplane computation

Suppose that facet $\boldsymbol{F} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_d]^T$ consists of d vertices $\boldsymbol{v}_i = [v_{i1}, v_{i2}, \cdots, v_{id}]^T$, for $i = 1,2, \cdots, d$, and also that $\boldsymbol{c} = [c_1, c_2, \cdots, c_d]^T$ is the center point of the current convex hull . Since any vertex $\boldsymbol{v}_i$ is located on the hyperplane H which includes facet $\boldsymbol{F}$, Eq. (4.3) is satisfied by $\boldsymbol{v}_i$ as (4.5).

$$a_1 v_{i1} + a_2 v_{i2} + \cdots + a_d v_{id} = -b \tag{4.5}$$

By adding $(-a_1 c_1 - a_2 c_2 - \cdots - a_d c_d)$ to the both side of Eq. (4.5), Eq. (4.6) is obtained.

$$a_1(v_{i1} - c_1) + a_2(v_{i2} - c_2) + \cdots + a_d(v_{id} - c_d) = -(a_1 c_1 + a_2 c_2 + \cdots + a_d c_d + b) \tag{4.6}$$

Suppose that point $\boldsymbol{c}$ is located on the negative subspace with respect to H. Hence the distance of $\boldsymbol{c}$ to H is negative as stated in (4.7).

$$d_s(\boldsymbol{c}, H) = \frac{a_1 c_1 + a_2 c_2 + \cdots a_d c_d + \mathrm{b}}{\sqrt{{a_1}^2 + {a_2}^2 + \cdots {a_d}^2}} < 0 \tag{4.7}$$

According to (4.7), $(a_1 c_1 + a_2 c_2 + \cdots a_d c_d + \mathrm{b})$ should have a negative value. Assume this negative value is equal to -1. By replacing $(a_1 c_1 + a_2 c_2 + \cdots a_d c_d + \mathrm{b})$ with -1 in the right side of Eq. (4.6), Eq. (4.8) is obtained.

$$a_1(v_{i1} - c_1) + a_2(v_{i2} - c_2) + \cdots + a_d(v_{id} - c_d) = 1 \tag{4.8}$$

Since facet $\boldsymbol{F}$ consists of d vertices, a system of equations can be obtained based on all vertices:

$$\begin{bmatrix} a_1(v_{11}-c_1) & + & a_2(v_{12}-c_2) & + & \cdots & + & a_d(v_{1d}-c_d) \\ a_1(v_{21}-c_1) & + & a_2(v_{22}-c_2) & + & \cdots & + & a_d(v_{2d}-c_d) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_1(v_{d1}-c_1) & + & a_2(v_{d2}-c_2) & + & \cdots & + & a_d(v_{dd}-c_d) \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \tag{4.9}$$

By solving (4.9), the normal vector $\boldsymbol{n} = [a_1, a_2, \cdots, a_d]^T$of H is obtained. Afterwards, the offset b of H is obtained using Eq. (4.10).

$$b = -1 - (a_1c_1 + a_2c_2 + \cdots + a_dc_d) \tag{4.10}$$

4.3.2. Convex hull distance

Given a set $P = \{x_i\}_{i=1}^{n} \subset \mathbb{R}^d$ and a point $x \in \mathbb{R}^d$, the Euclidean distance between x and the convex hull of P, denoted by $conv(P)$, can be computed by solving the quadratic optimization problem stated in (4.11).

$$\begin{aligned} &\min_{a} \frac{1}{2} a^T Q a - c^T a \\ &s.t.\ e^T a = 1, a \geq 0 \end{aligned} \tag{4.11}$$

where $e = [1,1,\cdots,1]^T, Q = X^TX$ and $c = X^Tx,$ with $X = [x_1, x_2, \dots, x_n]$.
Suppose that the optimal solution of (4.11) is a^*; then the distance of point x to $conv(P)$ is given by Eq. (4.12).

$$d_c\big(x, conv(P)\big) = \sqrt{x^Tx - 2c^Ta^* + a^{*^T}Qa^*} \tag{4.12}$$

4.3.3. The Proposed Algorithm

The idea behind ApproxHull is inspired from Quickhull, where the vertices of the real convex hull are identified, based on the hyperplane distance of samples to the facets of the current convex hull. Like Quickhull, ApproxHull is an incremental algorithm; it starts with an initial convex hull and then the current convex hull grows iteratively by adding the new vertices into it. In order to overcome the challenges of time complexity and memory requirements in high dimensions, ApproxHull has two main properties. Firstly, it is an approximation algorithm which is translated into obtaining a subset of the most informative vertices of the real convex hull using a user-defined threshold. On the contrary, Quickhull finds the total vertices of the real convex hull which prevents it to be applicable in large, high dimensional datasets. Secondly, the convex hull obtained by the ApproxHull is only given by vertices, whereas Quickhull presents the convex hull in terms of both vertices and facets, which makes it infeasible to be run for high dimensional datasets, due to the problem of high time complexity and memory requirements.

A pre-processing phase is performed on the original data set before applying ApproxHull. Duplicated rows (equal samples) and columns (equal features), rows with missing values, and rows having non-numerical values are removed to decrease the risk of generating a singular matrix corresponding to a random invalid facet in ApproxHull.

ApproxHull consists of five main steps:

Step 1: Scaling each dimension to the range [-1, 1].

Step 2: Identifying the maximum and minimum samples with respect to each dimension. These samples are considered as vertices of the initial convex hull.

Step 3: Generating a population of k facets based on the current vertices of convex hull. In this step, the validity of all generated facets is checked in each iteration. A facet F of d points in d dimensions is valid if F is a full rank matrix. In ApproxHull, to guarantee that the population contains valid facets, two actions are considered. First, when d vertices of the current convex hull are selected to constitute a facet of the population, its validity is checked. Invalid facets are ignored, being substituted by another combination of d vertices of the current convex hull until a valid facet is found. Second, in the case that there is a potential of generating invalid facets iteratively, in order to reduce the time being spent to ignore the invalid facets and generate the valid substitutions for them, the joggling method used in Quickhull [113] can be employed as an optional action in the data preprocessing phase. Joggling the input is performed to solve precision error in computational geometry context. Mainly, in joggling (also called random perturbation) the input of each cell of the data set is modified by a small random quantity (positive or negative) to solve the problem of coplanar points that have the potential of generating invalid facets.

Step 4: Identifying the furthest points to each facet in the current facets population as new vertices of convex hull, if they have not been detected before. To detect the furthest points (i.e., those samples whose hyperplane distances are maximum with respect to a particular facet), firstly, the corresponding hyperplane equation of the facet is obtained by Algorithm 4.1 in terms of the normal vector and the offset. Secondly, the hyperplane distance of samples to the corresponding hyperplane is computed by Eq. (4.4). In fact, Algorithm 4.1 computes the hyperplane equation relying on the fact that the distance of the center point (a row vector whose elements are obtained by averaging each dimension of the dataset) to the hyperplane is a negative value. Algorithm 4.1 obtains the equation of the corresponding hyperplane equation of the facet in form $Ax = b$ where A and b are normal and offset of the hyperplane equation respectively.

Step 5: Updating the current convex hull by adding the newly found vertices to the current set of vertices.

Steps 3 to 5 are executed iteratively until one of the following two termination criteria is met:

- There are no newly found vertices in Step 4
- Let dc be the maximum of the approximated distances of the furthest points to the current convex hull in each iteration. If there are new vertices as a consequence of Step 4, and the difference between the maximum and the minimum of dc over the w last iterations is less than a user-defined threshold β (default value of 0.1), the algorithm ends.

Algorithm 4.1: Obtaining the corresponding hyperplane of a facet

Input: $DS = \{x_i\}_{i=1}^{n} \subseteq R^d$ as a set of samples and $F = \{v_i\}_{i=1}^{d}$ as a particular facet so that v_i is a row vector which denotes a specific sample in DS.

```
1. Let $c$ is a row vector which denotes the center point of all samples in $DS$.
2. $U = \{u_i \mid u_i = v_i - c\}_{i=1}^{d}$
3. $A = \{\}$
4. $b = \{\}$
5. $A = U^{-1}e$ where $e = [1,1,\dots,1]^T$
6. $b = 1 + cA$
7. $t = \sqrt{\sum_{i=1}^{d} a_i^2}$ where $a_i \in A$ $for\ i = 1,2,\dots,d$
8. n$A = \{\frac{a_i}{t} \mid a_i \in A\}_{i=1}^{d}$
9. $nb = \frac{b}{t}$
```

Output: nA and nb.

The basic idea behind the second criterion is to avoid selecting new vertices that are very close to the current convex hull, not contributing this way with new information. As the convex hull generated by ApproxHull grows iteratively, the dc has a descending trend over iterations. Hence when the difference between the maximum and minimum of dc over the w last iterations is small, meaning the new found vertices are very close to the current convex hull, they can be ignored and ApproxHull can be terminated.

In the ApproxHull algorithm the facets population size, the sliding window width, and the user-defined threshold β, can be tuned to manage the number of vertices of the approximated convex hull. As an example, in the experimental part of this work we are interested in having at most half of the training set points from the convex hull and the remaining points selected randomly from the complete data set. The value $\beta = 0.1$, which was obtained by trial and error, satisfies our expectation for a facet population size <1000 and a sliding window width <10, in cases where the data was uniformly or normally sampled. In other applications where the data size (number of samples and dimensions) is high and the data distribution is unknown, the value of β can be tuned differently to meet the user-defined specification of the maximum percentage of approximated convex hull vertices in the training set.

Since computing the distance from a point to the current convex hull by solving the quadratic optimization problem defined in Eq. (4.11) is complex and time consuming in high dimensions, in ApproxHull the approximated distance of a newly found vertex to the current convex hull is computed based on $2 * d$ vertices, where d denotes dimension, which are the nearest neighbors to the newly found vertex in the current convex hull.

In Step 3 of ApproxHull, in order to generate a population of facets based on the vertices of the current convex hull, two policies were tested: 1- a stochastic policy; 2- a GA based policy. In the first policy, k facets are generated in such a way that each vertex of a specific facet is generated by random selection among the vertices of the current convex hull. The stochastic policy algorithm is summarized in Algorithm 4.2.

In the GA-based policy, k facets are generated by a GA so that $k = p * ng$, where p and ng denote the population size and the number of generations of the GA, respectively. In each generation of the GA execution, a new population of ps facets is created after employing crossover and mutation operators. In this policy, the population of each generation is appended to the total population so that, in the end, there is a total population of facets with size $p * ng$. The GA-based algorithm is summarized in Algorithm 4.3.

Since the idea behind ApproxHull is generating many different facets to help finding new vertices of the real convex hull, the diversity of generated facets is an important issue in the algorithm. In other words, more diversity in facets population provides a higher chance of detecting new vertices. Hence, the fitness value of a facet can be defined in terms of the inverse of occurrence ratio of its vertices over the current population. The fitness value of a specific facet is high if, in average, the occurrence ratio of its vertices over the current population is low. In ApproxHull with GA-based Policy, the fitness value of a facet in the population is measured by (4.13):

$$\frac{p * d}{\sum_{i=1}^{d} N_i} \tag{4.13}$$

where p, d and N_i are the population size, dimension and number of facets in the current population which share the i^{th} vertex of the facet, respectively.

In each iteration of the GA, parents are selected for mating using the Roulette Wheel method. Uniform crossover [114] is applied with a swapping probability of 0.5. The crossover probability is set to 0.7. For mutation, a vertex of a facet is selected randomly and replaced with another random vertex which has not been seen in the current population. The mutation probability is set to 0.05. Fig. 4.2 illustrates a simplified flowchart of the operations involved in ApproxHull.

Algorithm 4.2: ApproxHull with Stochastic Policy

Input: $DS = \{x_i\}_{i=1}^{n} \subseteq R^d$ as a set of samples obtained after the preprocessing of the original data, p denotes the population size of facets in d-dimensional space , w is an integer value as width of the sliding window and β as a user-defined threshold.

```
1. Scaling each dimension of $DS$ to the range $[-1, 1]$.
2. Let $V$ denotes the maximum and minimum
       samples with respect to each dimension in
     $DS$.
3. $NotFound = False$
4. $Diff = False$
5. $iteration = 1$
6. $DC = \{\}$
7. **While** (not $NotFound$ and not $Diff$) **do**
8.   Let $P$ be an empty population.
9.   **For** $(i = 1; i \le p; i++)$ **do**
10.    $isValid = False$
11.    **While** (not $isValid$) **do**
12.      $j = 1$
13.      Let $F$ be an empty facet.
14.      **While**$(j \le d)$ **do**
15.        Select randomly a vertex $v$ from $V$
16.        **If** ($v$ is not in $F$) **then**
17.          $F = F \cup \{v\}$
18.          $j = j + 1$
19.        **End If**
20.      **End While**
21.      **If** $(\det(F) \ne 0)$ **then**
22.        $isValid = True$
23.      **End If**
24.    **End While**
25.    $P = P \cup \{F\}$
26.  **End For**
27.  $newV = \{\}$
28.  **For each** facet $F$ in $P$ **do**
29.
Let $H$ be the corresponding hyperplane
      equation of facet $F$ which is obtained
         by Algorithm 4.1.
30.    $mds = max_{x_i \in DS} d_s(x_i, H)$
31.    $FP = \{x | x \in DS \; and \; d_s(x, H) =$
$mds\}$
32.    **For each** point $fp$ in $FP$ **do**
33.      **If** ($fp$ is not in $V$) **do**
34.        $newV = newV \cup \{fp\}$
35.      **End If**
36.    **End For**
37.  **End For**
38.  **If** $(newV = \{\})$ **then**
39.    $NotFound = True$
40.  **End If**
41.  **If** (not $NotFound$) **then**
42.    $D = \emptyset$
43.    **For** $(i = 1; i \le |newV|; i++)$ **do**
44.      $N_i = \emptyset, T = V$
45.      **For** $(k = 1; k \le 2 * d; k++)$ **do**
46.            $nn = \arg\min_{v_j \in T, j = 1, \cdots, |T|} \|z_i - v_j\|_2$
             where $z_i \in newV$
47.        $N_i = N_i \cup \{nn\}$
48.        $T = T \backslash \{nn\}$
```

```
49.       End For
50.       $D = D \cup \{d_c(z_i, N_i)\}$
51.     End For
52.     $dc = \max D$
53.     $DC = DC \cup \{dc\}$
54.     If $(iteration \geq w)$ then
55.       Let $dc_{min}$ be the minimum of $dc$ in
          $DC$ over $w$ last iterations.
56.       Let $dc_{max}$ be the maximum of $dc$ in
                  $DC$ over $w$ last iterations.
57.       If $((dc_{max} - dc_{min}) < \beta)$ then
58.         $Diff = True$
59.       Else
60.         $V = V \cup \{newV\}$
61.       End If
62.     End If
63.   End If
64.   $iteration = iteration + 1$
65. End While
Output: V
```

Algorithm 4.3: ApproxHull with GA-based Policy

```
Input: $DS = \{x_i\}_{i=1}^{n} \subseteq R^d$ as a set of samples obtained after the preprocessing of the original data, $p$ as population size, $pc$ as crossover probability, $pm$ as mutation probability, $ng$ as number of generation for GA, $w$ is an integer value as width of the sliding window and $\beta$ as a user-defined threshold.
1. Scaling each dimension of $DS$ to the range [-1, 1].
2. Let $V$ denotes the maximum and minimum
     samples with respect to each dimension in $DS$.
3. $NotFound = False$
4. $Diff = False$
5. $iteration = 1$
6. $DC = \{\}$
7. While (not $NotFound$ and not $Diff$) do
8.   Let $P$ be an empty population with maximum
       size $p * ng$.
9.   Let $G$ be an empty population with maximum
       size $p$.
10.  For $(i = 1; i \leq p; i++)$ do
11.    $isValid = False$
12.    While (not $isValid$) do
13.      $j = 1$
14.      Let $F$ be an empty facet.
15.      While$(j \leq d)$ do
16.        Select randomly a vertex $v$ from $V$
17.        If ($v$ is not in $F$) then
18.          $F = F \cup \{v\}$
19.          $j = j + 1$
20.        End If
21.      End While
22.      If $(\det(F) \neq 0)$ then
23.        $isValid = True$
24.      End If
25.    End While
26.    $G = G \cup \{F\}$
27.  End For
28.  $ga\_Iteration = 1$
29.  While $(ga\_Iteration \leq ng)$ do
30.    $P = P \cup G$
31.    $Offsprings = \{\}$
32.    For each $indv$ in $G$ do
33.      Compute the corresponding fitness value of
         $indv$.
34.    End For
35.    Let $MPool$ denotes an empty mating pool.
36.    For $(i = 1; i \leq p; i++)$ do
37.      Select one random parent $P1$ from $G$
         using Roulette Wheel method.
38.      $MPool = MPool \cup \{P1\}$
39.    End For
40.    $j = 0$
41.    While $(j < p)$ do
42.      Select two random parents $P1$ and $P2$ from
         $MPool$.
43.      Let $rc$ is a random number from range [0, 1].
44.      If $(rc \leq pc)$
45.        Do uniform crossover on $P1$ and P2 and
           consider $children$ for the result of crossover.
46.        For each $child$ in $children$ do
```

```
47.          Let $rm$ is a random number from
                                range $[0, 1]$
48.          If ($rm \le pm$)
49.            Do mutation on $child$
50.          End If
51.        End For
52.        For each $child$ in $children$ do
53.          If (det($child$) $\neq 0$) then
54.            $Offsprings = Offsprings \cup \{child\}$
55.          End If
56.        End For
57.        $j = |Offsprings|$
58.      End If
59.    End While
60.    Let G includes the first $p$ individuals of
       $Offsprings$.
61.    $ga\_Iteration = ga\_Iteration + 1$
62.  End While
63.  $newV = \{\}$
64.  For each facet $F$ in $P$ do
65.    Let $H$ be the corresponding hyperplane
       equation of facet $F$ which is obtained
       by Algorithm 4.1.
66.    $mds = max_{x_i \in DS} d_s(x_i, H)$
67.    $FP = \{x | x \in DS \ and \ d_s(x, H) = mds\}$
68.    For each point $fp$ in $FP$ do
69.      If ($fp$ is not in $V$) do
70.        $newV = newV \cup \{fp\}$
71.      End If
72.    End For
73.  End For
74.  If ($newV = \{\}$) then
75.    $NotFound = True$
76.  End If
77.  If (not $NotFound$) then
78.    $D = \emptyset$
79.    For ($i = 1; i \le |newV|; i++$) do
80.      $N_i = \emptyset, T = V$
81.      For ($k = 1; k \le 2 * d; k++$) do
82.            $nn = \arg\underset{v_j \in T, j = 1, \cdots, |T|}{min} \|z_i - v_j\|_2$
           where $z_i \in newV$
83.        $N_i = N_i \cup \{nn\}$
84.        $T = T \backslash \{nn\}$
85.      End For
86.      $D = D \cup \{d_c(z_i, N_i)\}$
87.    End For
88.    $dc = \max D$
89.    $DC = DC \cup \{dc\}$
90.    If ($iteration \ge w$) then
91.      Let $dc_{min}$ be the minimum of $dc$ in $DC$
         over $w$ last iterations.
92.      Let $dc_{max}$ be maximum of $dc$ in $DC$
         over $w$ last iterations.
93.      If (($dc_{max} - dc_{min}$) < β) then
94.        $Diff = True$
95.      Else
96.        $V = V \cup \{newV\}$
97.      End If
98.    End If
```

```
99.   **End If**
100. $iteration = iteration + 1$
101. **End While**
**Output:** V
```

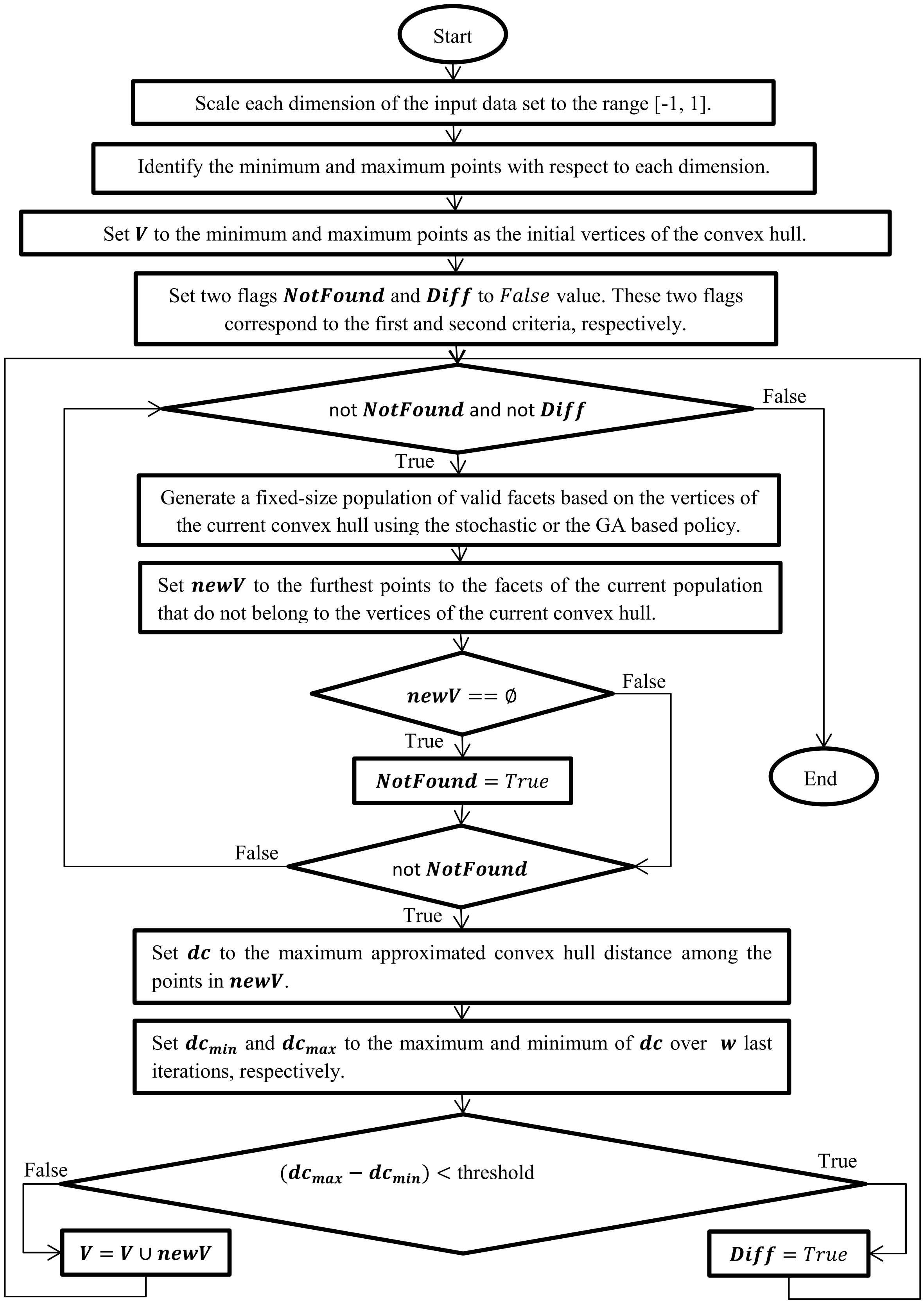

Start
Scale each dimension of the input data set to the range [-1, 1].
Identify the minimum and maximum points with respect to each dimension.
Set $\boldsymbol{V}$ to the minimum and maximum points as the initial vertices of the convex hull.
Set two flags $\boldsymbol{NotFound}$ and $\boldsymbol{Diff}$ to $False$ value. These two flags correspond to the first and second criteria, respectively.
not $\boldsymbol{NotFound}$ and not $\boldsymbol{Diff}$
False
True
Generate a fixed-size population of valid facets based on the vertices of the current convex hull using the stochastic or the GA based policy.
Set $\boldsymbol{newV}$ to the furthest points to the facets of the current population that do not belong to the vertices of the current convex hull.
$\boldsymbol{newV} == \emptyset$
False
True
$\boldsymbol{NotFound} = True$
End
False
not $\boldsymbol{NotFound}$
True
Set $\boldsymbol{dc}$ to the maximum approximated convex hull distance among the points in $\boldsymbol{newV}$.
Set $\boldsymbol{dc_{min}}$ and $\boldsymbol{dc_{max}}$ to the maximum and minimum of $\boldsymbol{dc}$ over $\boldsymbol{w}$ last iterations, respectively.
False
$(\boldsymbol{dc_{max}} - \boldsymbol{dc_{min}}) <$ threshold
True
$\boldsymbol{V} = \boldsymbol{V} \cup \boldsymbol{newV}$
$\boldsymbol{Diff} = True$

Fig. 4.2. Flow chart of ApproxHull.

4.4. Simulation results

Three experiments were executed to evaluate ApproxHull performance and its effect on the accuracy in classification and approximation tasks. The algorithm has been implemented in Python and C languages, and was executed in a computer with Ubuntu Linux OS, Intel Core i5 processor and 4 Gigabytes of RAM.

4.4.1. Experiment 1

ApproxHull was applied on four artificial datasets named UDS1, UDS2, UDS3 and UDS4. All datasets are composed of uniformly distributed random samples which were generated by the built-in MATLAB function *rand.* The description of the datasets is given in Table 4.1.

TABLE 4.1. DESCRIPTION OF ARTIFICIAL DATASETS CONSISTING OF UNIFORMLY DISTRIBUTED RANDOM SAMPLES. DIM AND #S DENOTE THE NUMBER OF DIMENSIONS AND SAMPLES, RESPECTIVELY.

Dataset Name	dim	#S
UDS1	3	4000
UDS2	4	4000
UDS3	5	4000
UDS4	6	4000

Since Quickhull is a deterministic algorithm, in this experiment it is considered as a baseline to which ApproxHull and Wang's algorithm [68], both being approximation convex hull algorithms, are compared. We use two criteria for comparison: P and R defined in Eq. (4.14) and Eq. (4.15).

$$P = \frac{\#(V_R \cap V_P)}{\#V_P} * 100 \tag{4.14}$$

$$R = \frac{\#(V_R \cap V_P)}{\#V_R} * 100 \tag{4.15}$$

V_R is the set of vertices obtained by employing the Quickhull algorithm and V_P is the set of vertices obtained by applying one of the other algorithms. Basically, criterion P shows the precision of an algorithm in approximating the Quickhull results, while criterion R denotes how much the results obtained by an algorithm are similar to those obtained by Quickhull.

In this experiment, ApproxHull considering the two facet generation policies and Wang's algorithm were executed for ten runs. For the latter, L was set to $0.01n$ for all datasets, and M was set as $M>=0.02n$, $M>=0.07n$, $M>=0.1n$ and $M>=0.14n$, for UDS1, UDS2, UDS3 and UDS4, respectively, n being the number of samples.

For ApproxHull with Stochastic Policy the sliding window size, w, was set to 10 for all datasets and p (population size) was set to 4000, 5000, 6000, and 7000 for datasets UDS1, UDS2, UDS3 and UDS4, respectively. For ApproxHull with GA-based Policy, w was also set to 10 for all datasets and the number of generations, ng, was set to 50. p (population size) was set to 80, 100, 120 and 140 for datasets UDS1, UDS2, UDS3 and UDS4, respectively.

Figs. 4.3 and 4.4 show the average values of P and R for the results obtained on datasets UDS1 to UDS4 by ApproxHull (with both policies) and by Wang's algorithm. By analyzing Fig. 4.3 it may be concluded that ApproxHull identifies only vertices that belong to the real convex hull, while Wang's algorithm selects some vertices which are not in the real convex hull. Moreover, according to Fig. 4.4, ApproxHull, using either the Stochastic Policy or GA-based Policy, detects more vertices of the real convex hull than Wang's algorithm. Fig. 4.4 also shows that ApproxHull with Stochastic Policy could identify more vertices of the real convex hull in comparison to ApproxHull employing the GA-based policy.

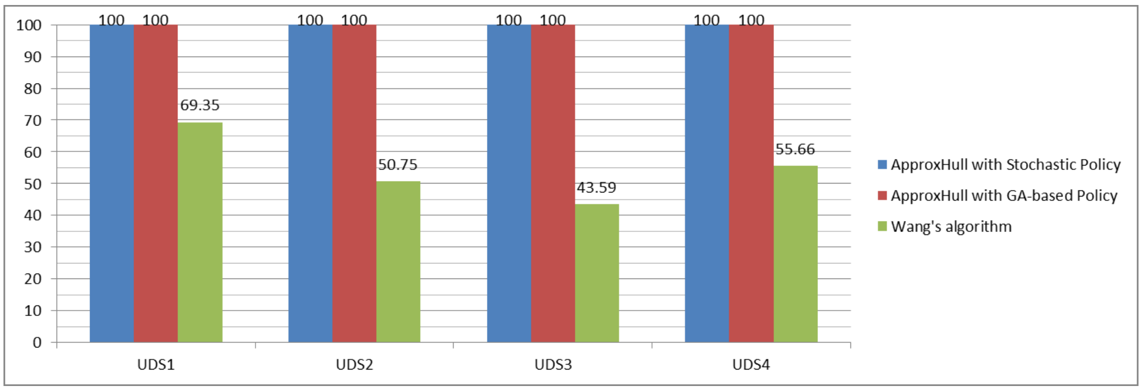


Fig. 4.3. Average value of criterion P for ApproxHull with both policies and Wang's algorithm on UDS1 to UDS4.

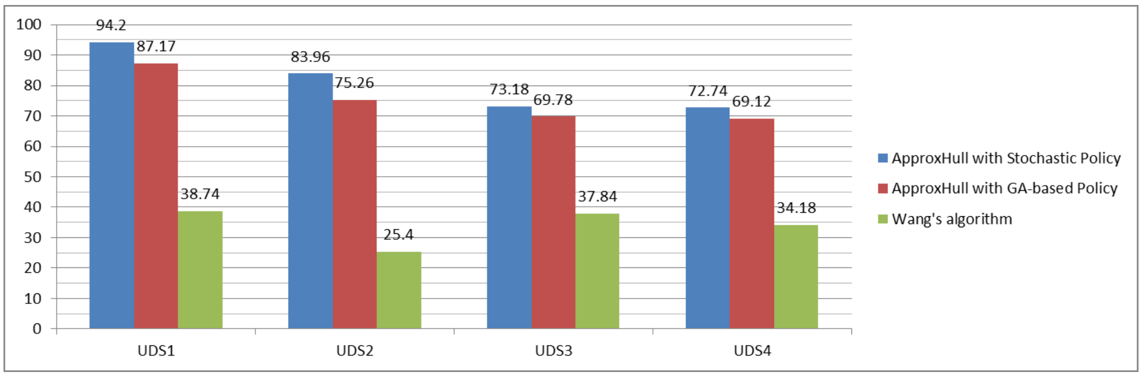


Fig. 4.4. Average value of criterion *R* for ApproxHull with both policies and the Wang's algorithm on UDS1 to UDS4.

Another experiment was conducted using normally distributed random samples which were generated by built-in MATLAB function *normrnd*, where the mean and the standard deviation were set to 0 and 1, respectively. In this experiment, ApproxHull with both policies was applied on four normally distributed artificial datasets for ten runs. The experiments were executed in the same conditions as described before.

Fig. 4.5 shows the average value of *R* for the results obtained by ApproxHull using both policies on datasets NDS1 to NDS4. The results in terms of criterion *P* were 100%, as in the previous case. Tables 4.2 and 4.3 also show the run time of ApproxHull using both policies on datasets in both cases.

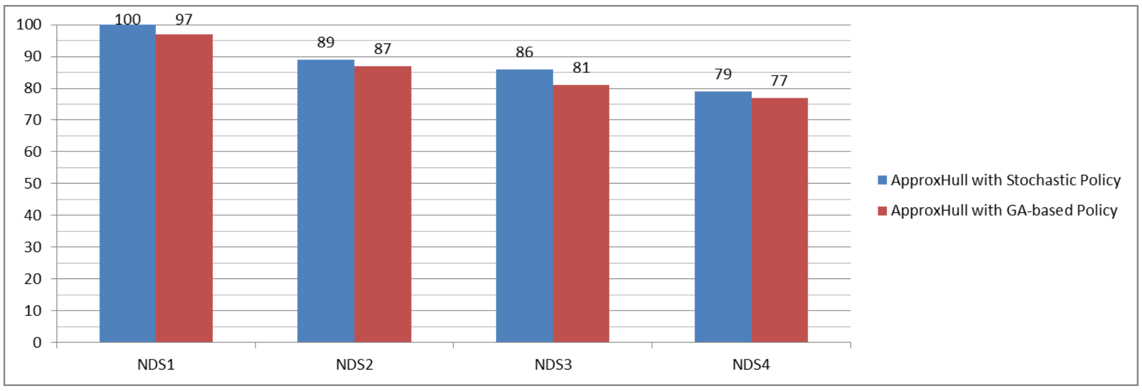


Fig. 4.5. Average value of criterion R for ApproxHull using both policies on NDS1 to NDS4.

TABLE 4.2. RUN TIME (IN SECONDS) OF APPROXHULL WITH STOCHASTIC POLICY AND GA-BASED POLICY ON DATASETS UDS1-4.

	UDS1	UDS2	UDS3	UDS4
Stochastic Policy	11.50	25.86	55.44	115.59
GA-based Policy	17.11	58.66	208.03	323.68

TABLE 4.3. RUN TIME (IN SECONDS) OF APPROXHULL WITH STOCHASTIC POLICY AND GA-BASED POLICY ON DATASETS NDS1-4.

	NDS1	NDS2	NDS3	NDS4
Stochastic Policy	6.30	11.86	30.33	50.20
GA-based Policy	10.06	19.24	90.06	156.78

Analyzing Fig. 4.4 and Fig. 4.5, ApproxHull with Stochastic Policy on both groups of artificial datasets has a better performance than the version employing the GA-based policy. In addition, according to Tables 4.2 and 4.3, the former is faster than the latter for all cases considered. For this reason, the Stochastic Policy will be used subsequently.

4.4.2. Experiment 2

In this experiment, ApproxHull was applied as a method for data selection in classification tasks. In order to evaluate the accuracy of the classification model, two cases were considered. In the first, ten training datasets were generated by random selection of samples from the whole dataset. In the second case, ten training datasets were generated, each one of them incorporating vertices of the approximated convex hull (which were obtained by ApproxHull) as well as random samples from the remaining dataset. The algorithm was applied separately for positive and negative classes. The datasets employed for classification were taken from [115]. The MATLAB SVM tool with Gaussian RBF (Radial Basis Function) kernel was used to design classifiers in both scenarios. The description of each dataset along with their corresponding hyper-parameters' values for the SVM classifiers is given in Table 4.4. In this experiment, the CR criterion stated in Section 2.5 was used. Table 4.5 shows the results obtained in the two cases for the datasets described in Table 4.4.

TABLE 4.4. DESCRIPTION OF THE DATASETS USED IN CLASSIFICATION. #F, #DS, #TR, #TE ARE THE NUMBER OF FEATURES, TOTAL NUMBER OF SAMPLES, NUMBER OF TRAINING SAMPLES AND TEST SAMPLES, RESPECTIVELY. C AND $\boldsymbol{\gamma}$ ARE THE SVM HYPER-PARAMETERS.

Dataset	Class1 / Class2	#F	#DS	#TR	#TE	C	γ
Breast Cancer	"Malignant" / "Benign"	30	569	376	193	1	0.05
Parkinson	"Yes" / "No"	26	1040	686	354	200	0.1
Satellite	"Red Soil" / "Grey Soil"	36	2033	1342	691	500	0.1142
Letter	"A" / "B"	16	1555	1026	529	1	0.6576
Cover Type	"Douglasfir" / "Krummholz"	54	37877	24999	12878	1	0.5

TABLE 4.5. AVERAGE CLASSIFICATION RATE FOR TEST DATASET IN TWO CASES FOR ALL DATASETS IN TABLE IV. $\boldsymbol{CR_{Te}(1)}$ AND $\boldsymbol{CR_{Te}(2)}$ DENOTE THE CLASSIFICATION RATES FOR THE TEST DATASET USING RANDOM SELECTION AND USING APPROXHULL, RESPECTIVELY.

Dataset	$CR_{Te}(1)$	$CR_{Te}(2)$	$CR_{Te}(2) - CR_{Te}(1)$
Breast Cancer	0.963	0.981	0.018
Parkinson	0.656	0.667	0.011
Satellite	0.990	1.000	0.010
Letter	0.993	1.000	0.007
Cover Type	1.000	1.000	0.000

According to the fourth column of Table 4.5, for all datasets the data selection mechanism employing ApproxHull has improved the accuracy of the corresponding classifiers, in comparison with the random data selection method. For the Breast Cancer and the Letter datasets, the highest and lowest improvements were achieved, respectively. For the Cover Type, both algorithms achieved perfect classification. The average classification rate for datasets Satellite, Letter and Cover Type, in the second case is equal to 1 which means that perfect classification is obtained for these datasets.

Additionally, as the previous datasets were balanced data sets, the same procedure, using the same SVM tool, was employed to a problem of automatic diagnosis of CVAs, from CT images. The application is described in, for instance, [116, 117] and, for the point of view of this paper, is a binary classification problem, using 51 features, with the aim of classifying each pixel in the intracranial area of each CT slice as normal, or abnormal (corresponding to a lesion). Using 150 CT slices corresponding to 7 exams, we had 1,867,602 pixels, from which

64,786 (around 3.5%) were abnormal. This is clearly a very bad-balanced problem. Approxhull has been applied for this large data set, and training and test datasets with the sizes of 20,000 and 14,000 were constructed. The average values of the classification rate obtained for the test sets, over 10 experiments, for random selection and using Approxhull were 0.972 and 0.983, respectively. This example demonstrates that, for a completely unbalanced problem, the use of ApproxHull again achieved better results than random selection, and that ApproxHull is applicable to large datasets.

In order to assess the statistical significance of the last results, as well as the ones presented in Table 4.5, we used the Wilcoxon signed-ranks test discussed in Section 2.9.2. Since in this case, L, the number of data sets, is equal to 60, the corresponding value of statistic z is equal to -5.35 which clearly indicates that the improvements obtained with ApproxHull are statistically significant.

4.4.3. Experiment 3

Experiment 3 was conducted to find out how much improvement can be obtained for regression models by employing ApproxHull for data selection.

As in Experiment 2, two approaches were analyzed for comparison: 1) generating ten training datasets by random selection; 2) generating ten training datasets by applying ApproxHull, together with random selection. The datasets which are used for regression were taken from [115, 118]. The description of each dataset is given in Table 4.6.

TABLE 4.6. DESCRIPTION OF THE DATASETS USED IN REGRESSION. #F, #DS, #TR, #TE AND #VAL ARE THE NUMBER OF FEATURES, TOTAL SAMPLES, TRAINING SAMPLES, TEST SAMPLES AND VALIDATION SAMPLES, RESPECTIVELY.

Dataset	#F	#DS	#TR	#TE	#VAL
Puma	32	8192	4915	1638	1639
Bank	32	8192	4915	1638	1639
CompAct	21	8192	4915	1638	1639
Concrete	8	1030	618	206	206
Skillcraft	18	3338	2003	667	668

The MLP implemented in MATLAB was employed with two hidden layers and the output layer with one linear neuron. For all datasets except the Concrete dataset, both hidden layers had ten sigmoidal neurons. For the Concrete dataset, both hidden layers employed five sigmoidal neurons. The Levenberg-Marquardt method (Please see Section 2.4.1.5) is employed to train the model, terminating if one of the following conditions is met: early-stopping, the number of training iterations exceeds 100 iterations, or the three criteria described in Section 2.4.1.7, where $\tau_f = 10^{-3}$.

The RMSE criterion is employed to evaluate the accuracy of the models. Tables 4.7 and 4.8 show the results obtained in the test and validation for the datasets described in Table 4.6.

Table 4.7 shows the average RMSE for the test datasets (i.e., data used for early stopping) in the two mentioned cases. As it may be seen in the sixth column, the regression models which resulted from the data selected by ApproxHull have a lower approximation error. Table 4.8 shows the average RMSE for the validation sets (i.e., data not used in the model design). Again, it may be concluded that the use of ApproxHull in the data selection phase, decreases the error for all datasets except for Skillcraft, which has an identical value.

TABLE 4.7. AVERAGE RMSE FOR THE TEST DATASETS IN TWO CASES FOR ALL DATASETS IN TABLE VI. $\boldsymbol{E_{Te}(1)}$ AND $\boldsymbol{E_{Te}(2)}$ DENOTE RMSE FOR TEST DATASET IN FIRST CASE (RANDOM SELECTION) AND SECOND CASE (DATA SELECTION USING APPROXHULL) RESPECTIVELY.

Dataset	Initial $E_{Te}(1)$	$E_{Te}(1)$	Initial $E_{Te}(2)$	$E_{Te}(2)$	$E_{Te}(1) - E_{Te}(2)$
Puma	0.336	0.076	0.326	0.073	0.003
Bank	0.293	0.209	0.260	0.195	0.014
CompAct	0.193	0.082	0.166	0.049	0.033
Concrete	0.374	0.161	0.329	0.143	0.018
Skillcraft	0.428	0.404	0.382	0.337	0.067

TABLE 4.8. AVERAGE RMSE FOR THE VALIDATION DATASETS IN TWO CASES FOR ALL DATASETS IN TABLE VI. $\boldsymbol{E_{Val}(1)}$ AND $\boldsymbol{E_{Val}(2)}$ DENOTE RMSE FOR VALIDATION DATASET IN FIRST CASE (RANDOM SELECTION) AND SECOND CASE (DATA SELECTION USING APPROXHULL) RESPECTIVELY.

Dataset	Initial $E_{Val}(1)$	$E_{Val}(1)$	Initial $E_{Val}(2)$	$E_{Val}(2)$	$E_{Val}(1) - E_{Val}(2)$
Puma	0.339	0.076	0.324	0.073	0.003
Bank	0.296	0.209	0.259	0.194	0.015
CompAct	0.194	0.061	0.169	0.048	0.013
Concrete	0.359	0.162	0.329	0.147	0.015
Skillcraft	0.406	0.334	0.382	0.334	0.000

For completeness, the second and the fourth columns in the last two tables illustrate the average initial values (before training was performed) of the RMSEs, for both approaches, for the test and validation datasets.

Performing the Wilcoxon signed-ranks test on the RMSEs of the test and validation data sets, with $L = 50$, the z values obtained are -4.73 and -4.18, respectively. In the same way as in the classification problems, the improvements obtained with Approxhull were considered to be statistically significant.

To summarize the results, among the 15 performance values presented in Tables 4.5, 4.7 and 4.8, as well as for the CVA problem, the use of ApproxHull for data selection achieves better results than those obtained by using random data selection in 14 cases, and achieves equal performance in 2 cases.

4.5. Run time analysis

The ApproxHull run time with Stochastic Policy depends on five factors including the size of the involved dataset (i.e., the number of samples and features), population size (input parameter p), number of iterations, number of vertices of convex hull found, and on the distribution of samples in the dataset. In order to analyze the dependency of the run time with these factors, two experiments were conducted. First, the algorithm was applied to all the datasets described in Tables 4.4 and 4.6 for ten times. For all datasets, p (population size) and w (width of sliding window) were set to 1000 and 5, respectively. Fig. 4.6 shows the average percentage of total samples identified as vertices of convex hull for each dataset described in Tables 4.4 and 4.6.

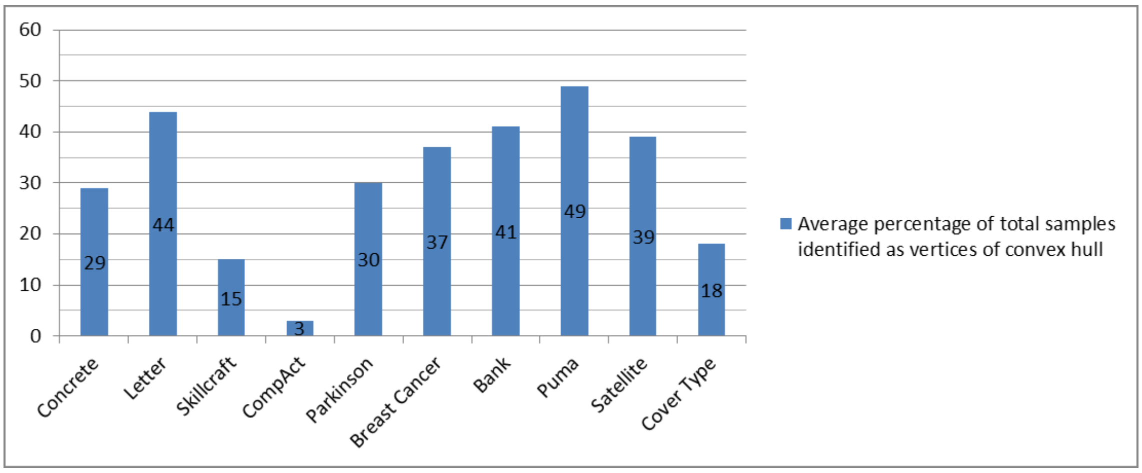


Fig. 4.6. Average percentage of total samples identified as vertices of convex hull for each dataset described in Tables 4.4 and 4.6.

Fig. 4.7 illustrates the average number of iterations that were used to terminate the algorithm for each dataset. The corresponding average run time for each dataset is given in Table 4.9. As it can be seen in this table, the highest and lowest average run times are related to datasets Cover Type and Concrete, respectively. Cover Type is the largest dataset in terms of number of samples and features while Concrete has the smallest number of features and is the second smallest dataset with respect to the number of samples. Although datasets Bank and Puma have the same size, the average run time for Bank is larger than that for Puma, because the average number of iterations for Bank is larger than that for Puma. This specific result related to datasets Puma and Bank reveals the fact that the distribution of samples can influence the run time.

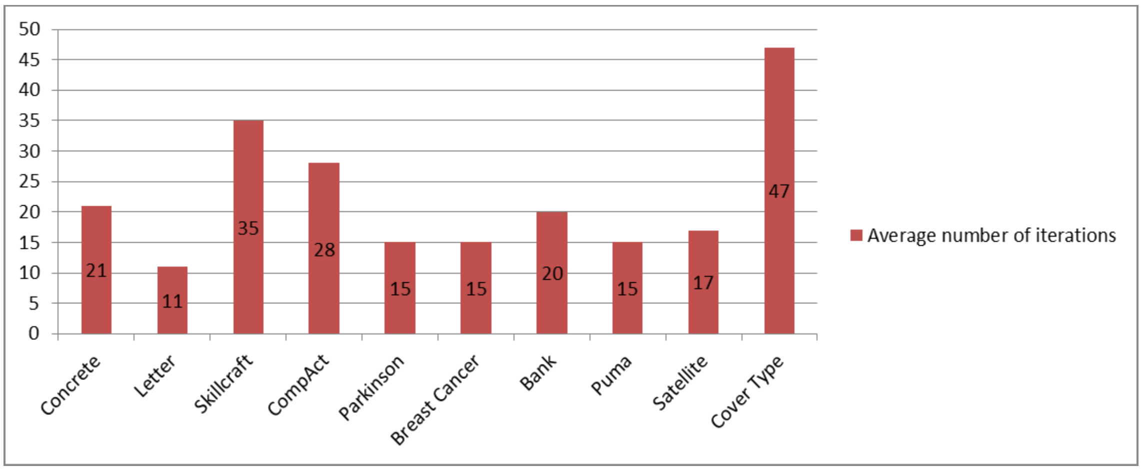


Fig. 4.7. Average number of iterations in ApproxHull for each dataset described in Tables 4.4 and 4.6.

TABLE 4.9. AVERAGE RUN TIME OF APPROXHULL ON DATASETS DESCRIBED IN TABLES 4.4 AND 4.6.

Dataset	Average Run Time (in seconds)
Concrete	11.78
Letter	19.13
Skillcraft	37.70
ComAct	37.39
Breast Cancer	8.10
Bank	257.34
Puma	174.24
Satellite	62.80
Cover Type	1280.16

Another experiment based on two groups of artificial datasets was conducted to clarify further the relationship between the run time of ApproxHull and the above mentioned factors.

The first group included thirty datasets which were composed of uniformly distributed random samples. The number of samples and the dimensions are in ranges [1000, 5000] and [4, 30] respectively. The second group employed the same number of datasets, in the same conditions, but using normally distributed random samples.

Figs. 4.8 and 4.9 show the effect of dataset size (i.e., number of samples and dimensions) on the run time of ApproxHull on five datasets with different number of samples and dimensions. For all datasets of both groups, population size (i.e., input parameter p) is set to 2000. It can be seen that in both groups of datasets, for a constant number of samples, by increasing the dimension, the run time of ApproxHull rises; for a constant dimension, by raising the number of samples, the run time also increases. It can also be observed that an increase in dimension is translated into a larger increase of the run time than an increase of the number of samples.

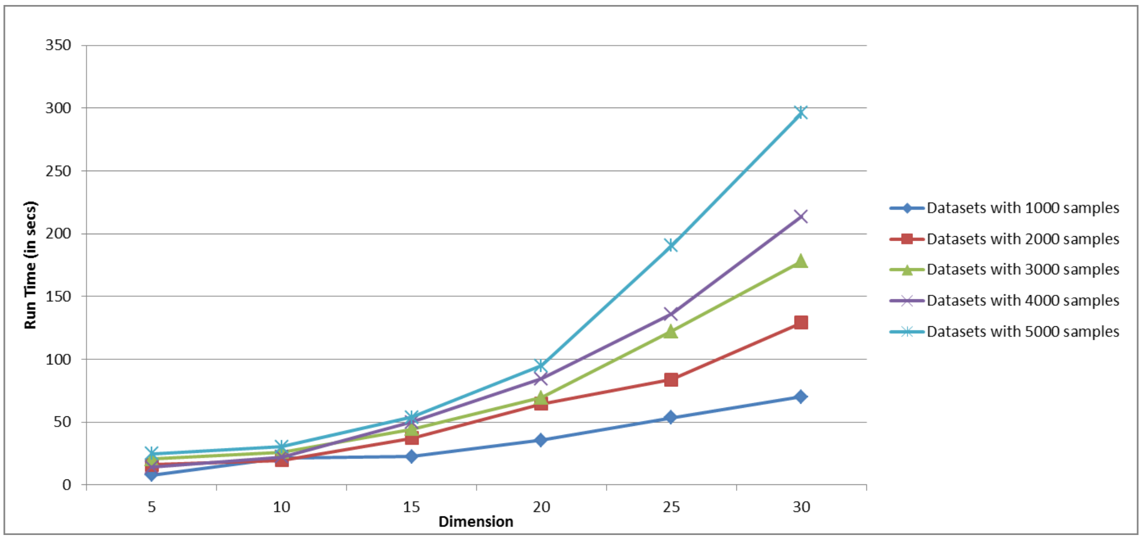


Fig. 4.8. Relationship between the size of five datasets containing uniformly distributed random samples and the run time of ApproxHull.

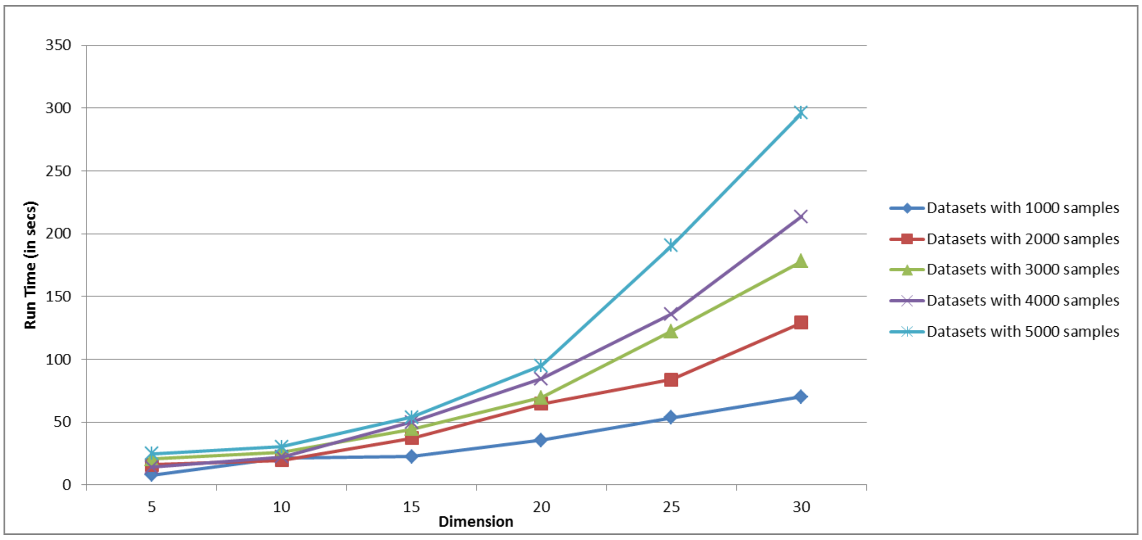


Fig. 4.9. Relationship between the size of five datasets containing normally distributed random samples and the run time of ApproxHull.

Figs. 4.10 and 4.11 illustrate the influence of population size (i.e., input parameter p) on the run time of ApproxHull for both groups of datasets. For each group, six datasets with 5000 samples are considered, with varying dimension. As it can be seen, on the one hand the population size has less influence on the run time in comparison with the effect of dataset size; on the other hand, if we enlarge the population size, this is not always translated into a run time increase (although this usually happens).

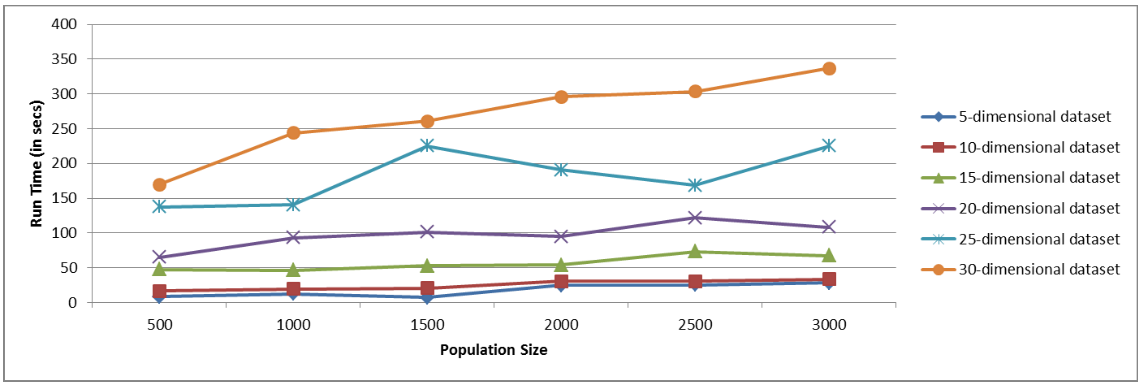


Fig. 4.10. Relationship between the population size (input parameter $\boldsymbol{p}$) and the run time of ApproxHull on six datasets containing uniformly distributed random samples.

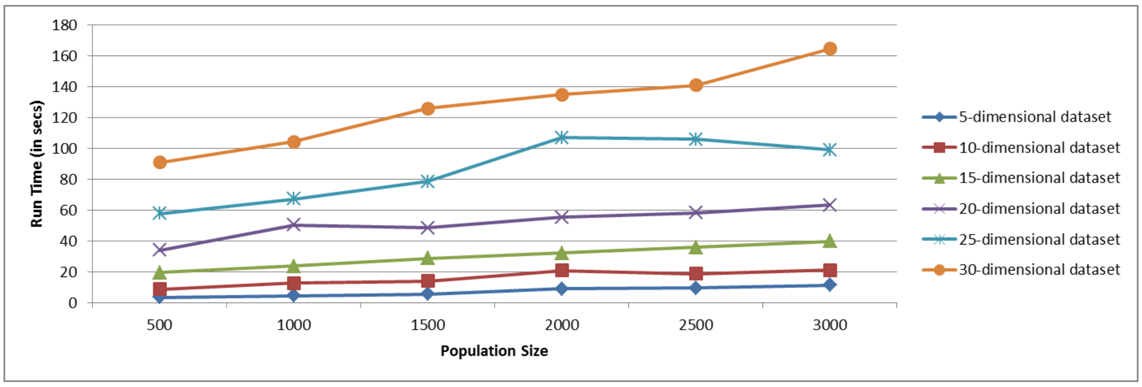


Fig. 4.11. Relationship between the population size (input parameter $\boldsymbol{p}$) and the run time of ApproxHull on six datasets containing normally distributed random samples.

As an example, in Fig 4.10 for the 25-dimensional dataset, by increasing population size from 1500 to 2000, the run time decreases. This happened because the number of iterations is equal to 44 in the case where the population size is set to 1500, whereas it is equal to 19 in the case where the population size is set to 2000. The corresponding number of iterations for different population sizes shown in Figs. 4.10 and 4.11 are given in Fig. 4.12 and 4.13.

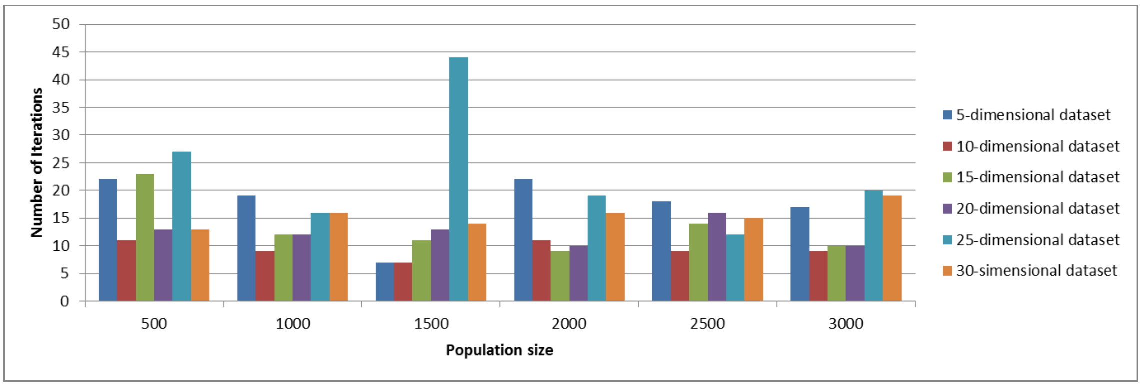


Fig. 4.12. Number of iterations in ApproxHull with six values for population sizes on six datasets containing 5000 uniformly distributed random samples.

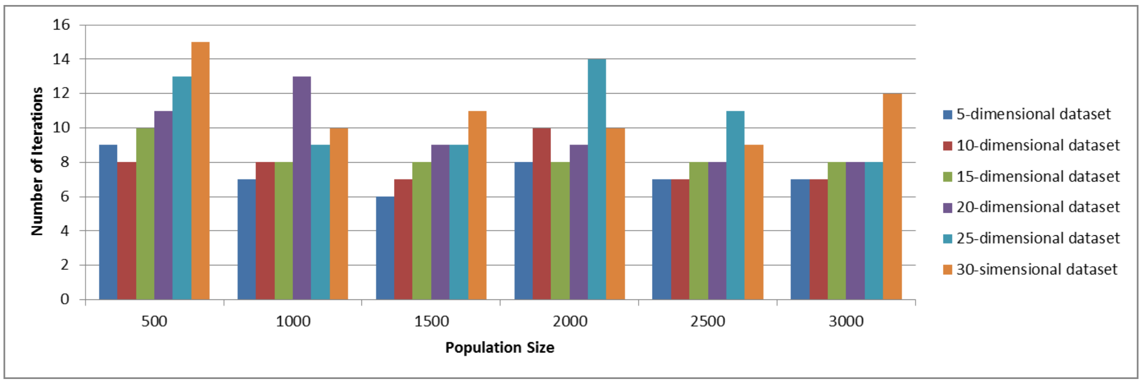


Fig. 4.13. Number of iterations in ApproxHull with six values for population sizes on six datasets containing 5000 normally distributed random samples.

Similarly, for the second group of datasets, increasing the population size does not always lead to a longer run time. For example, in Fig. 4.11, for the 20-dimensional dataset, by increasing population size from 1000 to 1500, the run time decreases because the corresponding number of iterations for population size 1000 is equal to 13, while it is equal to 9 for population size 1500.

From Fig. 4.14 and 4.15, it may be seen that in both group of datasets, an increase in population size does not always lead to an increase in percentage of samples identified as vertices of the convex hull.

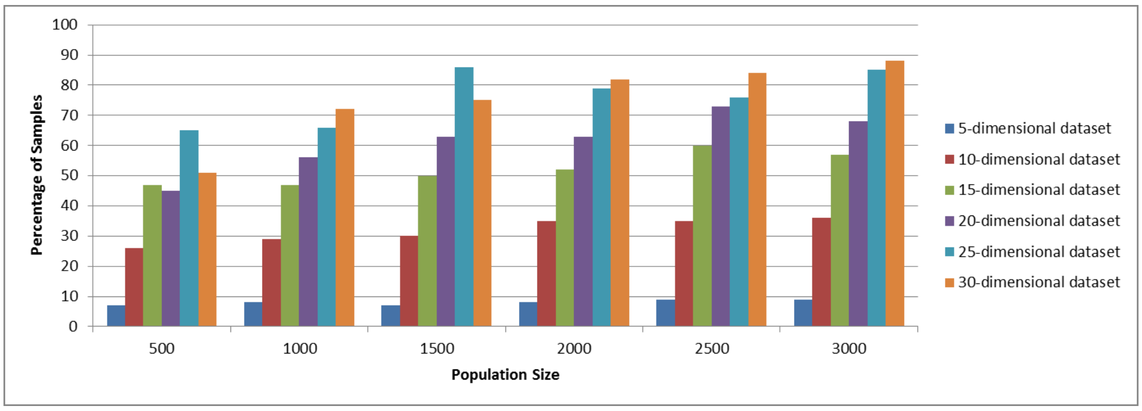


Fig. 4.14. Percentage of samples identified as vertices of convex hull by employing ApproxHull with six values for population sizes on six datasets containing 5000 uniformly distributed random samples.

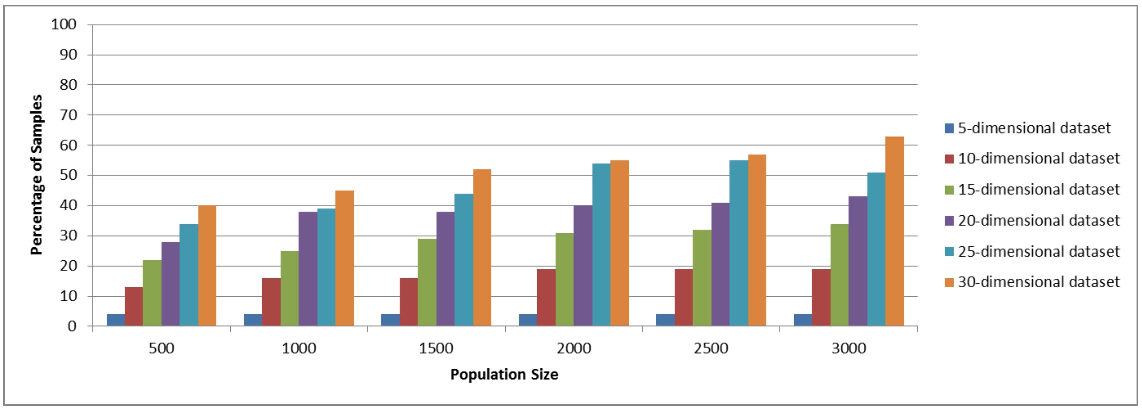


Fig. 4.15. Percentage of samples identified as vertices of convex hull by employing ApproxHull with six values for population sizes on six datasets containing 5000 normally distributed random samples.

From a data distribution point of view, Table 4.10 shows that the minimum and maximum run times of ApproxHull on datasets containing normally distributed random samples (i.e., the first group of datasets) are smaller than those on datasets containing uniformly distributed random samples (i.e., the second group of datasets). Correspondingly, Table 4.11 illustrates that the minimum and the maximum percentages of samples identified as vertices of convex hull from datasets involving normally distributed random samples are less than those from datasets involving uniformly distributed random samples.

TABLE 4.10. CORRESPONDING MINIMUM AND MAXIMUM RUN TIME OF APPROXHULL ON DATASETS USED IN FIG. 4.8 AND 4.9. RTMIN_1 AND RTMIN_2 DENOTE THE MINIMUM RUN TIME IN THE FIRST AND THE SECOND GROUP OF DATASETS RESPECTIVELY. RTMAX_1 AND RTMAX_2 DENOTE THE MAXIMUM RUN TIME IN THE FIRST AND THE SECOND GROUP OF DATASETS, RESPECTIVELY.

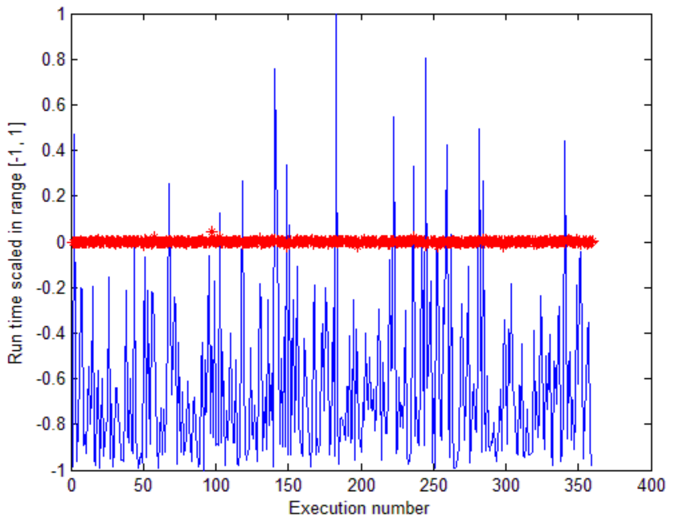
Cases	RTMIN_1	RTMIN_2	RTMAX_1	RTMAX_2
Datasets with 1000 samples	7.81	6.31	70.16	46.83
Datasets with 2000 samples	15.82	8.96	128.98	84.73
Datasets with 3000 samples	20.42	9.26	178.17	99.28
Datasets with 4000 samples	13.92	7.64	213.69	117.64
Datasets with 5000 samples	24.87	9.09	295.9	134.82

TABLE 4.11. CORRESPONDING MINIMUM AND MAXIMUM PERCENTAGE OF SAMPLES IDENTIFIED AS VERTICES OF CONVEX HULL FROM DATASETS USED IN FIG. 4.14 AND 4.15 PMIN_1 AND PMIN_2 DENOTE THE MINIMUM PERCENTAGE IN FIRST AND SECOND GROUP OF DATASETS. PMAX_1 AND PMAX_2 DENOTE THE MAXIMUM PERCENTAGE IN FIRST AND SECOND GROUP OF DATASETS.

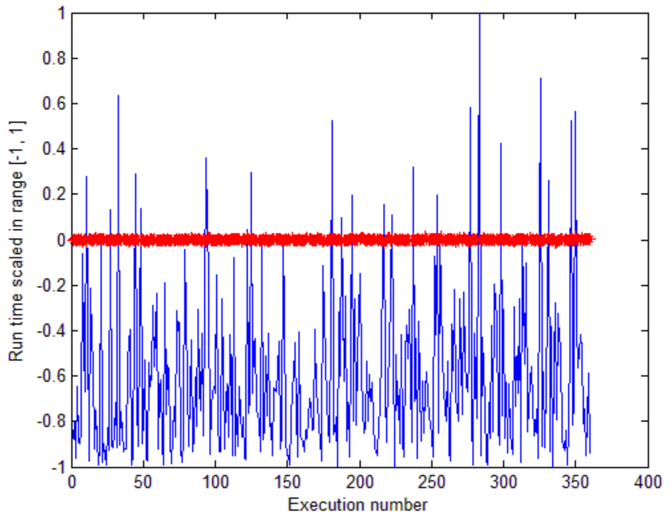
Dataset	PMIN_1	PMIN_2	PMAX_1	PMAX_2
5-dimensional dataset	7	4	9	4
10-dimensional dataset	26	13	36	19
15-dimensional dataset	47	22	60	34
20-dimensional dataset	45	28	73	43
25-dimensional dataset	65	34	86	55
30-dimensional dataset	51	40	88	63

In order to extract an approximate mathematical model specifying the relationship between the run time of ApproxHull as a function of the dataset size, population size, number of iteration and number of convex hull vertices, we employed the ASMOD algorithm [25] on the data obtained in the last experiment described above. The ASMOD algorithm is, as discussed in Section 2.2.3, a design technique for B-spline neural networks.

For each group of datasets, we collected 360 records of data by running ApproxHull on the corresponding datasets. According to the mathematical model which was obtained for each group of datasets, the time complexity of ApproxHull for both group of datasets can be approximated as $O(n^2d^3v^3 + i^3p^3)$ where n, d, v, i and p denote the number of samples, dimension, number of convex hull vertices found, number of iterations and population size, respectively. In order to assess the accuracy of the model obtained for the run time of ApproxHull, Fig. 4.16 presents the run time of ApproxHull (data scaled in a range [-1, 1[) and the error obtained by the model in two groups of datasets. As it can be seen, for both groups of datasets, a good accuracy for the model has been obtained.



(a)



(b)

Fig. 4.16. Run time of ApproxHull and the error obtained by the model on two groups of datasets. (a) First group: uniformly distributed random samples; (b) Second group: normally distributed random samples.

4.6. Memory requirements analysis

As mentioned in Section 3.3.3, Quickhull as a standard convex hull algorithm suffers from insufficient memory in high dimensions. In this section, to compare empirically ApproxHull with Quickhull in terms of memory requirements, both algorithms were applied to four artificial datasets described in Table 4.12. All datasets are composed of uniformly distributed random samples.

TABLE 4.12. DESCRIPTION OF THE ARTIFICIAL DATASETS CONSISTING OF UNIFORMLY DISTRIBUTED RANDOM SAMPLES. DIM AND #S DENOTE THE NUMBER OF DIMENSIONS AND SAMPLES RESPECTIVELY.

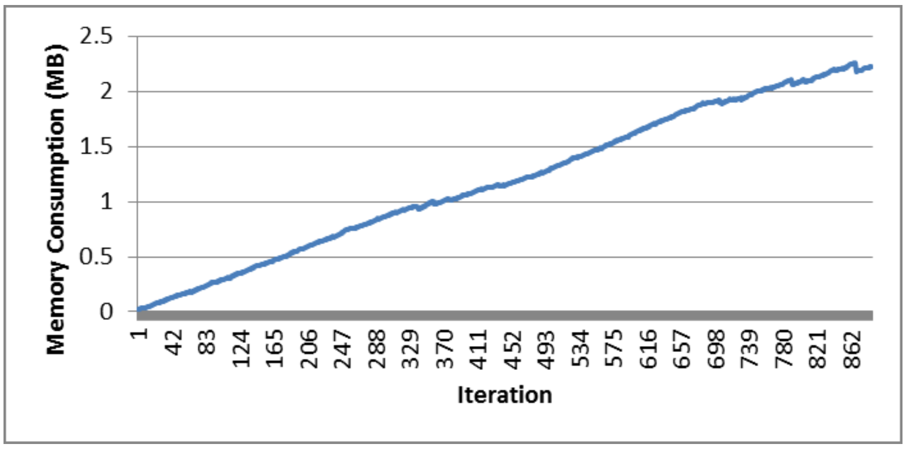
Dataset Name	dim	#S
DS1	5	4000
DS2	6	4000
DS3	7	4000
DS4	8	3500

Since facets in both ApproxHull and Quickhull are the principal objects to which a considerable amount of memory is allocated, this section addresses memory requirements for the generated facets in each iteration for both algorithms. For datasets DS1 to DS3, the sliding window size, w, was set to 10 and for DS4 it was set to 15. The population size, p, was set to 6000, 7000, 8000 and 9000 for datasets DS1, DS2, DS3 and DS4, respectively.

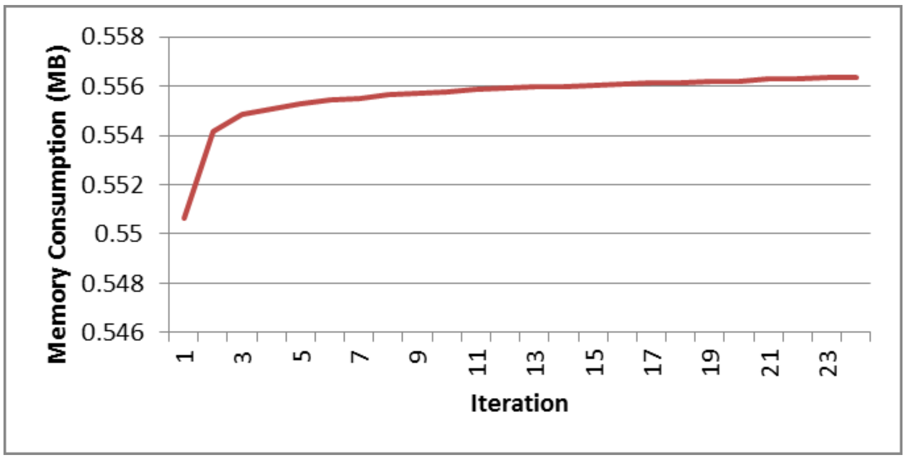
Figs. 4.17 to 4.20 show the trend of memory consumption over all iterations in both algorithms on datasets DS1 to DS4. An analysis of these figures shows that memory allocation in Quickhull for all generated facets in each iteration is much larger than that in ApproxHull. As it can be seen in the figures, the trend of memory consumption for all generated facets in ApproxHull is approximately constant in the last iterations. By increasing the dimension, the trend of memory consumption in Quickhull is linearly increasing, translating into a large amount of memory. In Quickhull, in each iteration, only the furthest point to current convex hull is added to the list of vertices and new necessary facets are generated to keep convexity in each iteration. Unlike Quickhull, in each iteration of ApproxHull, a large number of vertices are added into list of vertices of the current convex hull, and that is why the number of iterations of ApproxHull is lower than that of Quickhull. Moreover, in Quickhull, the current convex hull is described in terms of facets and the corresponding vertices so that, in high dimensions, the number of facets which reflect the whole current convex hull is huge. In contrast, the facets in the fixed size population in

ApproxHull are only used to detect the furthest point as vertices of real convex hull and they do not describe the whole current convex hull.

According to the explanation above, the amount of memory allocated to the facets for Quickhull is much larger, comparing to ApproxHull. The number of facets and the corresponding amount of memory allocated in both algorithms for the last iteration on datasets DS1 to DS4 are given in Table 4.13.

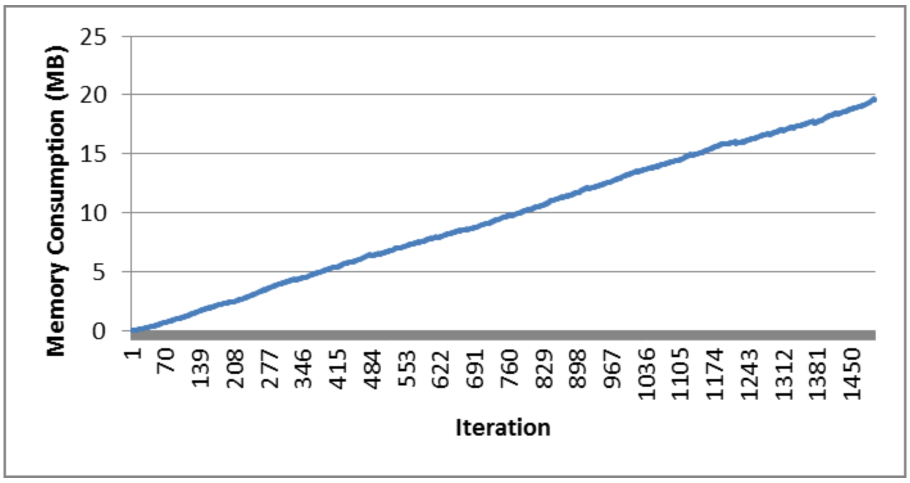


(a)

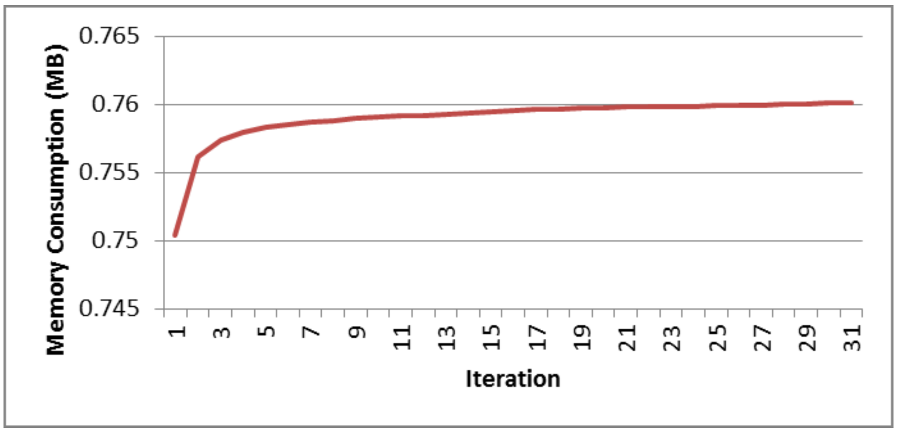


(b)

Fig. 4.17. Trend of memory consumption over iterations on DS1. (a) Quickhull; (b) ApproxHull

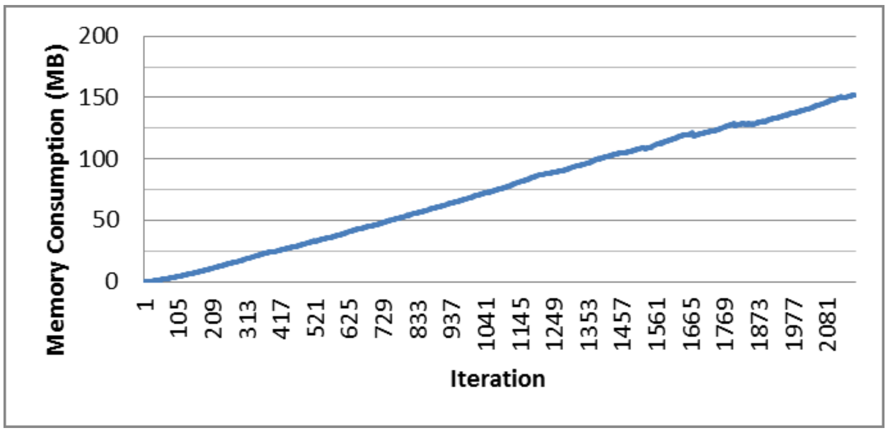


(a)

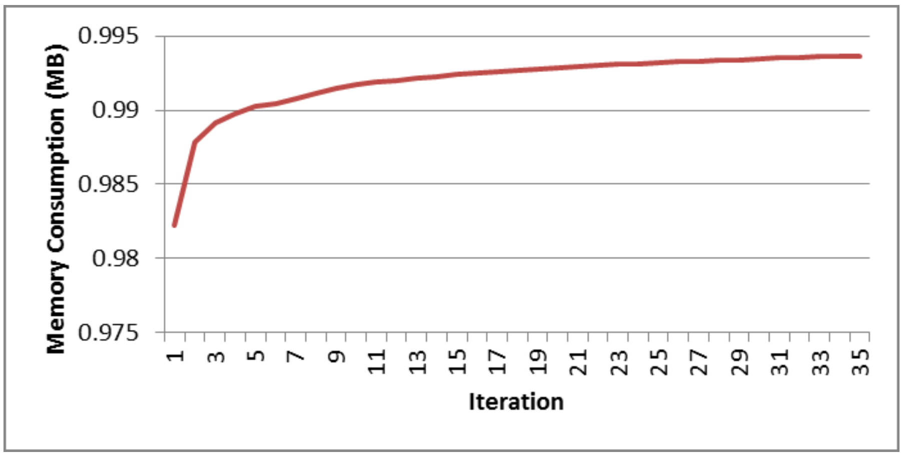


(b)

Fig. 4.18. Trend of memory consumption over iterations on DS2. (a) Quickhull; (b) ApproxHull

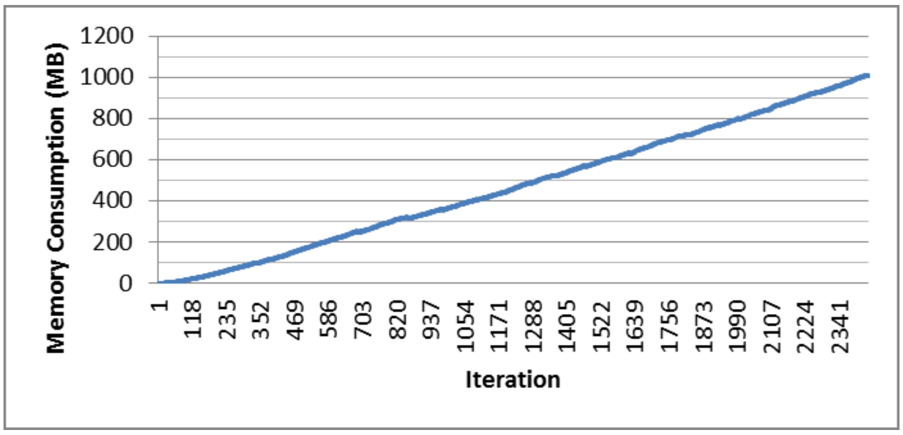


(a)

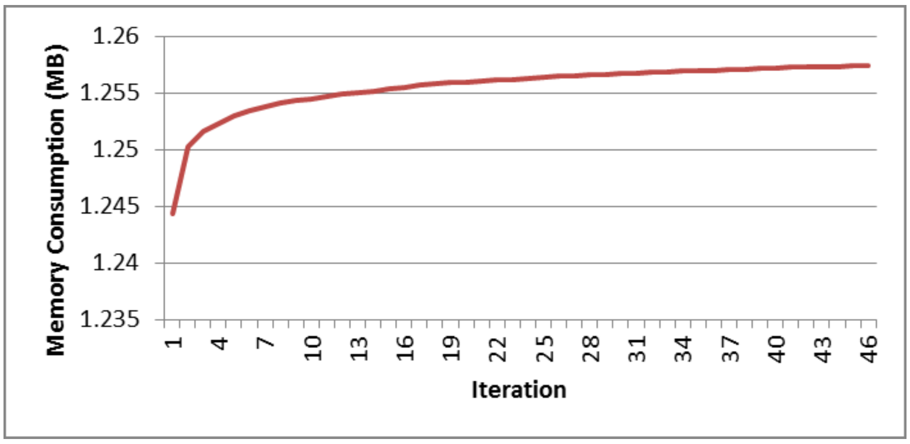


(b)

Fig. 4.19. Trend of memory consumption over iterations on DS3. (a) Quickhull; (b) ApproxHull



(a)



(b)

Fig. 4.20. Trend of memory consumption over iterations on DS4. (a) Quickhull; (b) ApproxHull

TABLE 4.13. NUMBER OF FACETS, TOTAL AND AVERAGE MEMORY SIZE FOR BOTH ALGORITHMS ON DS1 TO DS4 IN THE LAST ITERATION.

		Quikhull			ApproxHull		
		No. of facets	Total memory size (MB)	Average memory size (B)	No. of facets	Total memory size (MB)	Average memory size (B)
Dataset	DS1	12062	2.222557	193	6000	0.556335	97
	DS2	98801	19.62438	208	7000	0.760101	113
	DS3	712234	152.1882	224	8000	0.993637	130
	DS4	4396390	1006.301	240	9000	1.257355	146

4.7. Conclusions

This chapter describes a novel randomized approximation convex hull algorithm for high-dimensional data, to overcome the limiting memory requirements and time complexity problems found in conventional algorithms. ApproxHull is presented with two policies: stochastic policy and GA-based policy. Simulation results indicate that ApproxHull with Stochastic Policy is faster and its performance is better in comparison to the case where GA-based Policy is applied.

According to the simulation results, ApproxHull can find significantly more vertices of the real convex hull in comparison to Wang's algorithm [68]. Moreover, the obtained results in classification and regression problems show that the use of ApproxHull as a data selection method improves the accuracy of the designed models.

Based on the results obtained from employing ApproxHull with stochastic policy on two groups of datasets, it is revealed that dataset size, population size, number of iterations, number of vertices found as vertices of convex hull and the distribution of samples have influence on the run time. Based on a mathematical model obtained by using b-spline networks, the approximated time complexity of ApproxHull is $O(n^2d^3v^3 + i^3p^3)$ where n, d, v, i and p denote number of samples, dimension, number of convex hull vertices found, number of iterations and population size, respectively.

From a memory requirements point of view, simulation results reveal that the memory consumption in ApproxHull is much lower than the Quickhull algorithm, allowing the proposed algorithm to be applied in high-dimensional problems.

5. Applying ApproxHull in MOGA

5.1. Introduction

This chapter addresses the application of ApproxHull, introduced in Section 4.3 as a data selection method, in the model design process carried out by MOGA. In this chapter, the application of ApproxHull in MOGA is analyzed from two points of view. Firstly, the performance of ApproxHull is compared with random data selection method for MOGA. Secondly, the usage of ApproxHull in MOGA is addressed in the two following situations: In the first situation (i.e., hereinafter called common convex hull based data selection method), a common training, testing and validation sets (i.e., which are constructed using ApproxHull) are used to fit the parameters of all models that are generated by MOGA; whereas, in the second situation (i.e., hereinafter called distinct convex hull based data selection method), ApproxHull is used to construct a customized training, testing and validation sets for each generated model. The rest of the chapter is organized as follows: In Section 5.2, two experiments applying ApproxHull and random based data selection methods in MOGA are explained and analyzed. Two alternatives of applying ApproxHull in MOGA are discussed in Section 5.3 and finally some conclusions are given in Section 5.4.

5.2. Comparison of using random and convex hull based data selection methods for MOGA

In order to evaluate the performance of ApproxHull as a data selection method for MOGA, the problem of designing predictive time series models for Inside Air Temperature (IAT) was considered. To address this problem, a non-dominated set of Nonlinear AutoRegressive with eXogenous (NARX) models was designed by MOGA to predict the evolution of IAT over a Prediction Horizon (PH), for rooms in a building at University of Algarve. The data considered to design these models were the subsets of those that used to build the similar models proposed in [3, 119]. The input variables were IAT, Inside Air Humidity (IAH), Outside Air Temperature (OAT), Outside Solar Radiation (OSR), Reference Temperature (RT) and Movement signal (MOV). The corresponding data was collected with a sample rate of 5 minutes. For each variable 12 lags (i.e., one hour before) were considered to design the models. As a result, a data set containing 5062 samples with 73 features was provided.

Two MOGA experiments were carried out to design the models. For both experiments, a common training, testing and validation sets were applied to all models generated by MOGA and throughout MOGA generations, each model is trained and evaluated using a reduced version of the common data sets whose features corresponds to the model's inputs. In the first experiment, the common training, testing and validation sets were generated using common random based data selection method. In this method, the common sets were created by applying random data selection method on the whole data set (i.e., 5062 samples) and then presented to the MOGA. In the second experiment, the common sets were generated using common convex hull based data selection method in such a way that the common sets were produced by employing ApproxHull on the whole data set and presented to the MOGA. In this experiment, 1441 convex hull points were identified from the whole data set and included in the training set. 1596 randomly selected samples from the whole data set were also added to the training set. The size of the data sets is given in Table 5.1. Regarding MOGA parameters, for both experiments, the early stopping method with maximum 100 iterations was applied. The number of generations and the population size were both set to 100. The ranges of number of neurons and features were set to [2, 30] and [0, 30], respectively. For both experiments, the design objectives were typically the RMSE obtained in the training and test data sets, and the model complexity. In both experiments, no restriction on objectives was considered. The number of models in the non-dominated set of first and second experiments was equal to 101 and 173, respectively. The results obtained by the evaluation of models in the non-dominated set of both MOGA experiments are given in Table 5.2. The results are in terms of RMSE (ρ) on the training (tr), testing (te), validation (va) and whole data set (D). rnd and $hull$ in Table 5.2 denote the first and second MOGA experiment, respectively.

TABLE 5.1. THE SIZE OF TRAINING, TESTING AND VALIDATION SETS.

	Training set	Testing set	Validation set
Size	3037×73	1012×73	1013×73

TABLE 5.2. RESULTS OBTAINED FROM THE MOGA EXPERIMENTS.

	ρ_{tr}^{rnd}	ρ_{tr}^{hull}	ρ_{te}^{rnd}	ρ_{te}^{hull}	ρ_{va}^{rnd}	ρ_{va}^{hull}	ρ_{D}^{rnd}	ρ_{D}^{hull}
Min	0.0065	0.0066	0.0085	0.0081	0.0090	0.0085	0.0081	0.0079
Avg	0.0082	0.0085	0.0091	0.0088	0.0231	0.0106	0.0144	0.0092
Max	0.0108	0.0112	0.0107	0.0109	0.5392	0.0669	0.2413	0.0308

As it can be seen in Table 5.2, the performance in the training set using the random approach is slightly better than the convex hull approach. This is expected as the latter includes the convex hull points, using therefore a larger range than the former. This situation changes for the testing set and regarding the validation set, as unseen data, and also the whole data set, the performance of models obtained by applying ApproxHull (the second MOGA experiment) is significantly better than those achieved by using the random selection method (the first MOGA experiment).

5.3. Comparison of the use of the common and distinct convex hull based data selection methods for MOGA

Simulation results obtained in Section 5.2 showed that the performance of the obtained models by convex hull based data selection method is better than that of those achieved by random data selection method.

Another question which remains to be answered is whether or not using the common convex hull based data selection method (i.e., which was applied in the second experiment stated in Section 5.2) brings us a better performance comparing to the situation where distinct convex hull based data selection methods are used in MOGA (i.e., in which ApproxHull is used to construct a customized training, testing and validation sets for each model generated by MOGA). In the case of using distinct convex hull based data selection method, for each model generated by MOGA, ApproxHull is applied on a reduced version of the whole data set whose features corresponds to the model's inputs. Afterwards, the corresponding training, testing and validation sets are generated to train and evaluate the model.

In order to compare these two strategies, we focused on the first generation of MOGA, where a number of models are randomly generated satisfying the restrictions imposed on the number neurons and features (i.e., unlike other generations in which models are generated based on the previous generation by using the crossover and mutation genetic operators).

Like Section 5.2, we considered the IAT models to compare the above two methods. The evaluation results obtained from the two methods are given in Table 5.3. com and dis in Table 5.3 denote the common and distinct convex hull based data selection methods, respectively. ρ_D denotes the RMSE on the whole data set. n_{chv} and t_{chv} indicate the number of convex hull points and ApproxHull run time, respectively.

Table 5.4 shows the total time spent to design all models in the first generation. In Table 5.4, T_{chv} denotes the summation of ApproxHull run time over all models. Since in common convex hull based data selection method, ApproxHull is applied only once on the whole data set, T_{chv} is equal to t_{chv}^{com} in Table 5.3. T_{tr} denotes the summation of training times over all models in the common and distinct convex hull based data selection methods, respectively. In Table 5.4, T denotes the total time including ApproxHull run time and training time over all models in common and distinct convex hull based data selection methods.

TABLE 5.3. EVALUATION RESULTS OBTAINED FROM TWO METHODS.

	ρ_D^{com}	ρ_D^{dis}	n_{chv}^{com}	n_{chv}^{dis}	t_{chv}^{com} (sec)	t_{chv}^{dis} (sec)
Min	0.0152	0.0152	1434	108	283.25	1.70
Avg	0.0294	0.0294	1434	889	283.25	19.21
Max	0.0652	0.0647	1434	1829	283.25	72.16

TABLE 5.4. TOTAL TIME TO DESIGN ALL MODELS IN THE TWO METHODS.

	T_{chv} (sec)	T_{tr} (sec)	T (sec)
Common data sets based strategy	283.25	1550.00	1833.25
Distinct data sets based strategy	1421.32	1544.96	2966.28

According to the first two columns of Table 5.3, there is no significant difference between the performance of models in two cases. Based on the third and fourth columns of Table 5.3, the number of convex hull points obtained using the common convex hull based data selection method is much larger than that of those achieved from the distinct convex hull based data selection method. This result stems from the fact that, in the former, ApproxHull is applied on the whole data set containing 5062 samples with 73 features while in the latter, ApproxHull is employed on reduced data sets containing the same number of samples with at most 30 features (due to forcing MOGA to generate models with at most 30 input features). As it can be seen in Table 5.4, the total time spent to design all models in the common convex hull based data selection method is much less than that in the other method. It comes from the fact that in the former, ApproxHull is applied only once whereas in the latter, it is independently employed for each model.

5.4. Conclusions

This chapter was aimed to evaluate the ApproxHull performance in MOGA. Two groups of MOGA experiments were carried out to design time series models based on RBFNN to predict one-step-ahead IAT for a building at the University of Algarve. The results obtained from the first group of experiments showed that applying ApproxHull as a data selection method can improve the performance of models in comparison with random selection method. Moreover, we were motivated to study applying ApproxHull in MOGA based on two methods; 1- common convex hull based data selection method 2- distinct convex hull based data selection method. The results achieved from the second group of experiments showed that not only the latter is not superior to the former, but also it takes more time in model design, in comparison with the former.

6. Case Studies

6.1. Introduction

To show the feasibility of applying ApproxHull in real applications, this chapter addresses three case studies in which ApproxHull has been employed as a data selection method to create training, testing and validation sets.

The first case study is linked to design a group of predictive RBFNN models, as well as a basic MLP model, which were aimed to forecast the energy consumption of a building at University of Almeria, Spain [120].

The second case study was intended to present an intelligent weather station which not only measures climate variables but also provides a prediction over a predefined prediction horizon [14, 15]. The intelligent weather station was applied to implement a predictive control of HVAC systems [3, 121]. In this case study, a series of predictive RBFNN models were designed to forecast climate variables.

In the third case study, ApproxHull was applied to build a classification model based on RBFNN, as an intelligent support system for automatic diagnosis for CVA, where the model was designed based on the data extracted from CT images of several patients [117].

The rest of this chapter is organized as follows: Section 6.2 details the first case study since it was carried out as a part of this PhD. Section 6.3 and 6.4 present a brief explanation of the second and third case studies, respectively as ApproxHull was used by different researchers involved in other projects. Finally some conclusions are given in Section 6.5.

6.2. Case Study 1: Energy consumption

Due to fast economic development affected by industrialization and globalization, energy consumption has been steadily increasing over the last years [122, 123]. Industry, transportation and buildings are the three main economic sectors which consume a significant amount of energy, with buildings having the biggest proportion. For example in European Union countries, energy consumption in buildings represents about 40% of the total energy consumption [124]. In USA, more than 44% of domestic energy consumption belongs to HVAC systems in buildings [125]. Studies have shown that by following the current energy consumption pattern, the world energy consumption may increase more than 50% before 2030 [126], while most of the energy resources are not renewable in nature. Moreover, the usage of energy causes environmental degradation [123]. Therefore, energy consumption management is a very significant problem not only to tackle the loss resulting from increasing consumption patterns but also to improve the performance of building energy systems. With respect to energy management, a variety of policies have been considered. In recent years, bioclimatic architectures for buildings have been focused to reduce the indoor consumption of energy. In this kind of architecture, buildings are designed based on the local climate conditions. These include wind speed and direction, daily exterior temperature and relative humidity, as well as diverse passive solar technologies where heating and cooling techniques passively absorb solar radiation or protect from it without containing mobile elements [127-129]. Besides environmental variables, physical properties of buildings are considered in bioclimatic architectures, such as shape, buildings' orientation related to the sun and wind, wall thickness and roof construction [127, 130].

Utilizing renewable energy sources such as biomass, hydropower, geothermal, solar, wind and marine energies have been considered as alternatives for conventional energy resources in most developed and developing countries [131, 132]. In the European Union, the use of renewable energies share is 20% of the total energy consumption and 10% of renewable energies will be used in transportation by 2020 [133]. Using renewable energies not only helps keeping the security of non-renewable energy supply in future, but also minimizes environmental degradation [132].

Prediction of energy use in buildings has received a remarkable amount of attention from researchers [122, 124, 134, 135], as an approach to reduce energy consumption, which is intended to conserve energy and reduce environmental impacts [124]. The prediction of energy usage in buildings and modeling the behavior of the corresponding energy system, are complicated tasks due to influential factors such as weather variables, building construction, thermal properties of the physical materials and occupants' activities [124]. Furthermore, there are several nonlinear inter-relationships among the involved variables, often in a noisy environment, which amplify the difficulty in identifying the precise interaction among them [136].

The methods aiming to predict building energy consumption can be categorized mainly into statistical, engineering and artificial intelligence ones. A review on prediction methods can be found in [124, 137].

Engineering methods, which are detailed comprehensive methods, use the structural properties of buildings in the form of physical principles and thermal dynamics equations, as well as environmental information such as climate conditions, occupants, their activities and HVAC equipment parameters. On the one hand, these methods need a high level of details about the structural and thermal parameters of buildings that are not always available and, on the other hand, since engineering methods depend on complex physical principles, a high level of expertise is needed to elaborately develop the corresponding models [54, 124]. To reduce the complexity of the detailed comprehensive engineering methods, simplified methods have been proposed, which can be seen in [138, 139].

Statistical methods use historical data to correlate energy consumption as target with most influential variables as inputs. Hence, the quality and quantity of historical data has a crucial role in developing statistical models [54, 140]. Unlike engineering methods, statistical methods provide models with a smaller number of variables and much less physical understanding. Regression models, CDA (Conditional Demand Analysis), ARMA (Auto Regressive Moving Average), ARIMA (Auto Regressive Integrated Moving Average) and GMM (Gaussian Mixture Models) are some instances of statistical models [140-143].

In recent years, artificial intelligence methods such as neural networks, support vector machines and fuzzy logic have been widely considered in applications of energy consumption. Like statistical methods, artificial intelligence methods use historical data reflecting the behavior of the process to be modeled. Neural networks have shown a high capability to capture complex nonlinear relationships between inputs and outputs. Since the energy consumption process has a nonlinear behavior, neural networks are mostly applied in this domain. In addition, they are quicker and easier to develop than engineering and statistical methods, while being accurate estimators. Some instances of neural network based models may be found in [54, 136, 144-148].

Recently, support vector machines have received much attention as quick methods to build predictive models in applications of energy consumption. They can provide models with a high level of generalization based on number of data. Their application on the prediction of energy utilization can be viewed, for instance, in [149-151].

Besides neural network and support vector machine based models, another kind of models have been considered, which benefit from fuzzy logic. Fuzzy logic deals with imprecise reality and handles the concept of truth value ranging between completely true and completely false (1–0) [152]. Some models of this type can be seen in [153, 154].

As mentioned earlier, both statistical and artificial intelligence methods need sufficient historical data to provide accurate models. In cases where limited amounts of data are available and the information about the process to be modeled is partially known, grey models are suitable alternatives to the prediction of time series associated with processes [155-157].

The objective of this case study is to compare an MLP model obtained in [54] with the RBFNN models obtained by MOGA, to predict the electric power demand of the CIESOL building located at University of Almeria, Spain. Authors in [54] determined the structure and the order of the model by statistical and analytical methods while in this article a non-dominated set of models is generated by a MOGA considering a set of objectives to be optimized. For the sake of completion, the performance of MOGA models is also compared with the results obtained by a Naive Autoregressive Baseline (NAB) approach, introduced in [158].
The following briefly describes the structural properties and power demand profile of CIESOL building. Afterwards the model proposed in [54] and the models generated by MOGA are widely described and finally, experimental results are shown.

6.2.1. Experimental setup: The CIESOL building

The CIESOL building, see Fig. 6.1(a), is a mixed solar energy research center between CIEMAT (Centre for Energy, Environment and Technology – Centro de Investigaciones Energéticas, MedioAmbientales y Tecnológicas (in Spanish)) and the University of Almería, situated in the south-east of Spain. This geographical location is characterized by having a typical semi-desert Mediterranean climate [159]. This building is divided into two floors with a total surface approximately equal to 1100 m2. More specifically, the upper floor is composed by four laboratories, the director's office and a meeting–room. In the lower floor, five offices, four laboratories, two bathrooms and a kitchen are located. Besides these, the machinery of the solar cooling installation is placed into an environment which occupies two floors.
This building has been designed and built within a research project named PSE-ARFRISOL [160], following bioclimatic architecture criteria. Therefore, it makes a beneficial use of natural ventilation and solar energy in order to reduce energy consumption and CO_2 emissions. To do that, it employs a HVAC system based on solar cooling installation, which can be observed in Fig. 6.1(b), composed by a solar collector field, a hot water storage system, a boiler and an absorption machine with its refrigeration tower [160], and a photovoltaic power plant with a peak power of 9 kW which provides electricity to the building (see Fig. 6.1(c) and (d)). Furthermore, a wide network of sensors has been installed in order to monitor the most representative enclosures of the building. Concretely, this network of sensors includes, among others, air temperature, relative humidity, CO_2

concentration, solar radiation, wind velocity and power consumption sensors. Moreover, these sensors are connected to different *Compact FieldPoint* modules from *National Instruments* that are distributed by means of an Industrial Ethernet network all around the building [160]. Data provided by the network of sensors are being stored through a SCADA (Supervisory Control And Data Acquisition) system developed with LabVIEW® [160]. Finally, it is necessary to take into account that this building is a research center which includes chemical, environmental analysis and modeling and control research groups. Hence, the machinery, other electrical devices and experiments performed by these research groups alter the energy use profile of the building in comparison with more common ones, such as residential buildings.

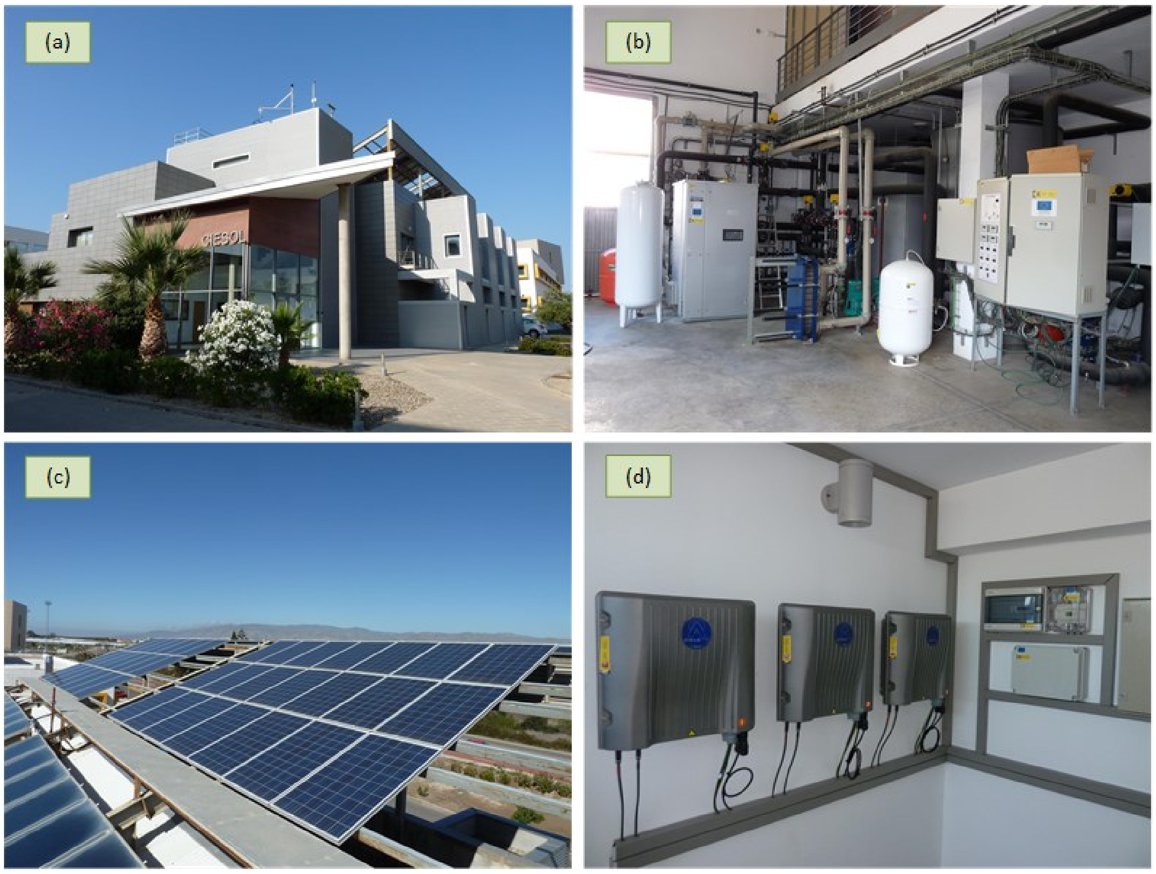


Fig. 6.1. The CIESOL building: (a) Exterior of the CIESOL building; (b) Solar cooling installation; (c) Photovoltaic power plant: PV panels; (d) Photovoltaic power plant: PV inverters.

6.2.1.1. Power demand profiles of the CIESOL building

From a power demand point of view, the CIESOL building has some special characteristics mainly derived from the research tasks which are being developed inside it. Therefore, it is necessary to perform an exhaustive analysis of the different energy demand profiles which can be found at the CIESOL building. Specifically, a statistical characterization involving certain parameters like arithmetic mean ($\bar{x}$), standard deviation (σ), and minimum and maximum values of the power demand (*min* and *max* respectively) under several conditions (different seasons and types of days), has been performed (see Table 6.1).

TABLE 6.1. STATISTICAL ANALYSIS OF THE POWER DEMAND PROFILES (IN KW).

	$\bar{x}$	σ	*min*	*max*
Working day	24.36	6.39	17.39	44.17
Non-working day	19.45	1.83	12.72	23.86
Winter	26.45	4.55	18.93	39.48
Spring	23.91	6.76	12.56	42.79
Autumn	24.23	4.58	15.85	48.14
Summer	28.74	8.67	16.28	63.48

To predict the power demand within a building, it is necessary to consider numerous energy consuming elements, such as illumination, electrical devices, HVAC systems, etc. At the CIESOL building, the element which has the greatest energy consumption is the solar cooling installation. Furthermore, to calculate the total energy demand of the CIESOL building it is necessary to consider both the energy supplied by the electricity company and the energy produced by the photovoltaic power plant which is directly consumed by the building, that is, at this moment it is not possible to store the energy from the photovoltaic power plant.

Firstly, the main differences according to typical power demand profiles between working and non-working days have been studied, as presented in Fig. 6.2. To do that, a typical day for each demand profile, considering working and non-working days, and each season, has been selected as a function of several environmental variables: mean, maximum and minimum temperature, temperature ranges and solar radiation. The methodology consists of selecting the day with the minimum value obtained from the sum of the weighted absolute difference between each parameter (daily) and the mean value of this parameter along the analyzed period. A detailed description of the procedure which has been followed can be found in [161]. It can be observed that power demand in a working day begins to increase around

08:00 *am* and starts to decrease at 05:00 *pm*, reaching a stationary value around 8 *pm*, whereas, in a non-working day it has a stationary value approximately equal to 20 kW, mainly due to the machinery and experimental tests performed inside this building. From the perspective of the statistical analysis shown in Table 6.1, it can be inferred that the mean power demand for a working day is equal to 24.36 kW with a standard deviation of 6.39 kW. On the contrary, for a non-working day, a mean power demand of 19.45 kW and a standard deviation equal to 1.83 kW have been obtained. In addition, working days also present a higher peak power demand, in comparison with non-working days.

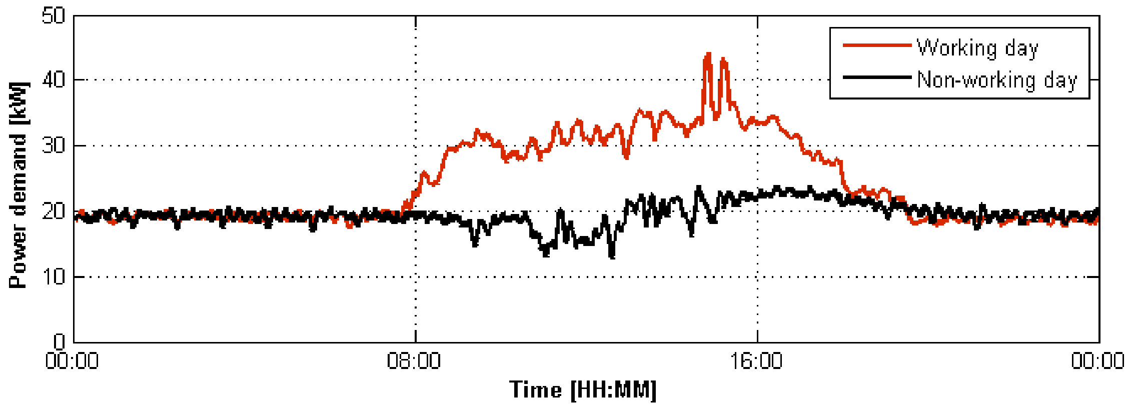


Fig. 6.2. Energy demand profiles for working and non-working days.

Secondly, a detailed examination of the power demand of the CIESOL building through a typical week (from Monday to Sunday), along different environmental conditions has been performed, as shown in Fig. 6.3. The main objectives of this analysis were to determine if there were representative differences among the different seasons of the year and also to identify if there was any characteristic element of the building able to considerably influence its power demand. More specifically, as it can be deduced from Fig. 6.3, the different seasons of the year follow an analogous pattern among working and non-working days. In addition, it can also be inferred that spring and summer seasons present a higher power demand in comparison with winter and autumn. Besides, along the summer season there are several power demand peaks that do not follow any specific pattern associated with the type of day. Therefore, in order to clarify this issue, a detailed analysis of this fact has been performed, and the main conclusions derived from it were that these peaks were associated with the use of a heating pump (for research purposes) and the solar cooling installation. Hence, as the use of both elements is directly associated with the users of the building, it has been decided to take into account the state variables representing these elements within the preliminary list of variables (see Table 6.2). Finally, according to the statistical analysis, it can be concluded that the highest peak power demand and variance is associated with the summer season mainly due to the use the HVAC system for cooling purposes [160].

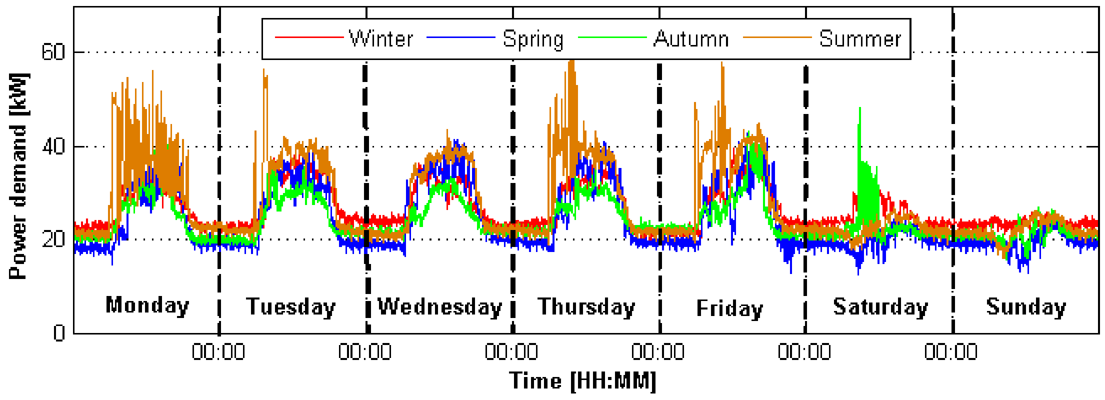


Fig. 6.3. Weekly energy demand profiles for each season.

TABLE 6.2. PRELIMINARY LIST OF VARIABLES [54].

Variable	Unit	Measurement range
Type of the day (Working day/Non-working day)	–	{0, 1}
Hour of the day	h	[0, 23]
Outdoor temperature	[ºC]	[-5, 50]
Outdoor humidity	[%]	[0, ..., 100]
Outdoor solar radiation	[W/m^2]	[0, 1440]
Outdoor wind speed	m/s	[0, 22]
Outdoor wind direction	º	[0, 360]
State of the pump B1.1 (Off/On)	–	{0, 1}
State of the pump B1.2 (Off/On)	–	{0, 1}
State of the pump B2.1 (Off/On)	–	{0, 1}
State of the pump B2.2 (Off/On)	–	{0, 1}
State of the pump B3.1 (Off/On)	–	{0, 1}
State of the pump B3.2 (Off/On)	–	{0, 1}
State of the pump B7 (Off/On)	–	{0, 1}
State of the boiler (Off/On)	–	{0, 1}
State of the absorption machine (Off/On)	–	{0, 1}
State of the refrigeration tower (Off/On)	–	{0, 1}
State of the heat pump (Off/On)	–	{0, 1}
Electric power demand	[kW]	[0, 85]
Electric power injected by the PV plant	[kW]	[0, 9]

Finally, the principal conclusions which have been reached after this precise analysis can be summarized in: a) there is a clear power demand profile within a week and also, the differences among working and non-working days power demand profiles can be undoubtedly established; b) the power demand for summer is higher mainly due to the typical semi-desert

Mediterranean climate of Almería; and c) the use of the solar cooling installation has a considerable influence on the final energy consumption.

6.2.1.2. Data acquisition

As mentioned previously, in this case study, several energy consumption prediction models based on RBFNNs were designed and compared with the corresponding MLP model proposed in [54]. These models were obtained by means of different methodologies. More specifically, groups of non-dominated sets of RBFNN models were designed by MOGA. Afterwards, these groups of models were compared with a basic MLP model presented in [54]. To do that, a historic data set acquired at the CIESOL building was used. Concretely, this data set comprises data from 01/09/2010 to 29/02/2012 with a sample time of 1 minute and it includes a preliminary list of variables which can be observed in Table 6.2. These variables are related with the environmental conditions and the state of the main energy consuming elements of the solar cooling installation.

Subsequently a whole data set containing 514762 samples was obtained. To design the basic MLP model proposed in [54], the whole data set was split into three sub-data sets training, testing and validation involving 318340, 107264 and 89158 samples, respectively. This division has been performed by hand since there were some discontinuities in time series. More information about the methodology followed to obtain these data subsets can be found in [54]. On the other hand, to design a group of non-dominated sets of RBFNN models by MOGA, the original whole data set was resampled from 1 minute to 15 minutes to reduce the size of the whole data set due to the presence of limitations in MOGA against large size data sets. Afterwards, along each week period, the corresponding data of three random days were selected. Consequently, a reduced data set consisting of 8640 samples was achieved. To generate the corresponding training, testing and validation sets, ApproxHull algorithm proposed in Section 4.3 was applied as a data selection method. As a result, for all MOGA experiments, the training, testing and validation set including 2592, 864 and 864 samples were generated, respectively. The detailed explanation of the data preparation process for MOGA is given in Section 6.2.3.1.

6.2.2. A Non-linear AutoRegressive with eXogenous inputs Multi-Layer Perceptron Neural Network model

In [54] a prediction model based on MLP for the energy consumption of the CIESOL building was proposed. To do that, the Neural Network Toolbox^{TM} provided by MATLAB$^{®}$ was used. Concretely, the proposed model had a Non-linear AutoRegressive with eXogenous inputs (NARX) architecture, see Eq. (6.1), typified by having a tapped delay line for the input signals set and another one for the output signal, that is, the power demand prediction of the CIESOL building. Moreover, this model was trained using a gradient-descent based algorithm, more specifically the Levenberg-Marquardt algorithm [33].

$$y[k+1]=f\left(u[k],u[k-1],\ldots \text{[illegible]} \ldots \text{[illegible]}\right) \tag{6.1}$$

In Eq. (6.1), $u[k]$and $y[k]$represent the input and output signals at time instant k, $d_u \geq 1$, $d_y \geq 1$ (subjected to $d_y \geq d_u$) are the memory orders for the input and output tapped delay lines, respectively, and f represents a non-linear mapping function which, in this case, was approximated by an MLP network.

The structure of an MLP network is completely defined by indicating: a) the number of hidden layers and the number of neurons in each layer; b) the number of neurons in the output layer; and c) the activation function used in each neuron of the hidden and output layers. More specifically, in the model presented in [54], an MLP with only one hidden layer composed by 10 neurons with tangent hyperbolic activation functions and one neuron with linear activation function at the output layer was considered, since it is a universal approximator [162].

Afterwards, the selection of input variables from the preliminary variables list, see Table 6.2, was performed through analytical methods, since they allow to establish the existing linear and non-linear dependencies. Besides, the scatter-plots and the model tests were used in order to complete the information provided by analytical methods. A detailed description of these methods can be found in [54]. Therefore, after the application of the methods mentioned above, the preliminary variables list was reduced to the following ones: type of the day; hour of the day; outdoor temperature and solar radiation; state variables related to the solar cooling installation; and the total power demand of the CIESOL building.

Finally, it was necessary to select the order of the signal inputs, that is, the embedding delay τ and the embedding dimension d [54]. The former was determined by means of the average mutual information [163], whereas for the latter, optimal values were calculated by the False Neighbors Method [164]. The list of final input variables and their order can be observed in Table 6.3.

TABLE 6.3. FINAL LIST OF VARIABLES WITH THEIR ORDER (EMBEDDING DELAY AND DIMENSION).

Variable	Unit	Measurement range	τ	d
Type of the day (Working day/Non-working day)	–	{0, 1}	1	1
Hour of the day	–	[0, 23]	1	1
Outdoor temperature	[ºC]	[-5, 50]	1	4
Outdoor solar radiation	[W/m^2]	[0, 1440]	1	4
State of the pump B1.1 (Off/On)	–	{0, 1}	1	5
State of the pump B1.2 (Off/On)	–	{0, 1}	1	5
State of the pump B2.1 (Off/On)	–	{0, 1}	1	5
State of the pump B2.2 (Off/On)	–	{0, 1}	1	5
State of the pump B3.1 (Off/On)	–	{0, 1}	1	5
State of the pump B3.2 (Off/On)	–	{0, 1}	1	5
State of the pump B7 (Off/On)	–	{0, 1}	1	5
State of the boiler (Off/On)	–	{0, 1}	1	5
State of the absorption machine (Off/On)	–	{0, 1}	1	5
State of the refrigeration tower (Off/On)	–	{0, 1}	1	5
State of the heat pump (Off/On)	–	{0, 1}	1	5
Electric power demand	[kW]	[0, 100]	1	3

6.2.3. Radial Basis Function Neural Network based models generated by MOGA

MOGA is a design framework which can be applied to determine both the structure and the parameters of ANN based models (i.e., please see Sections 2.6 and 2.7). The models used in this case have a NARX structure as shown in (6.1), with the difference that *f*(.) is now a RBFNN, instead of a MLP. In this approach, instead of one model, a non-dominated set of models are generated. From this set, one solution must be selected. In this section, data preparation for MOGA and related experiments are described.

6.2.3.1. Data preparation

After an analysis of the original data, a new code was considered for the feature “day type”. The new code refers to “special days”. By comparing the amount of energy consumption for working and non-working days, it has been revealed that for some days over the years 2010 and 2011, the amount of energy consumption has an average value between working and non-working days. By comparing these special days with the Spanish calendar for both years, it was found that those days occurred in the early days of the year, or in working days which were located between national/regional holidays and weekends. Based on that, these special days received the code 0.5. Fig. 6.4 shows the distribution of whole data samples in terms of “day type”. Since the original data was obtained with a sampling interval of 1 minute, its size was too large (514762 samples) to be handled by the MOGA framework, and was reduced in several stages. Due to presence of gaps in the data, there were 51 consecutive periods over the whole data. In the first stage, each period was divided into one week length segments. Based on these divisions, those durations whose length was less than two weeks were ignored in this work. This stage resulted into 13 periods containing at least two weeks of data. Table 6.4 shows the periods selected in the first stage.

In the second stage, the data for all periods was reduced by a factor of 15 by averaging every 15 consecutive samples inside each segment. The sampling interval was then increased to 15 minutes.

In the third stage, by starting from the second week within each period, 3 random days along with the last 7 consecutive days were selected as lags for each variable. This way, a data set $\boldsymbol{D}$ with 8640 samples was obtained. Fig. 6.5 shows the distribution of samples of data set $\boldsymbol{D}$ in terms of “day type”.

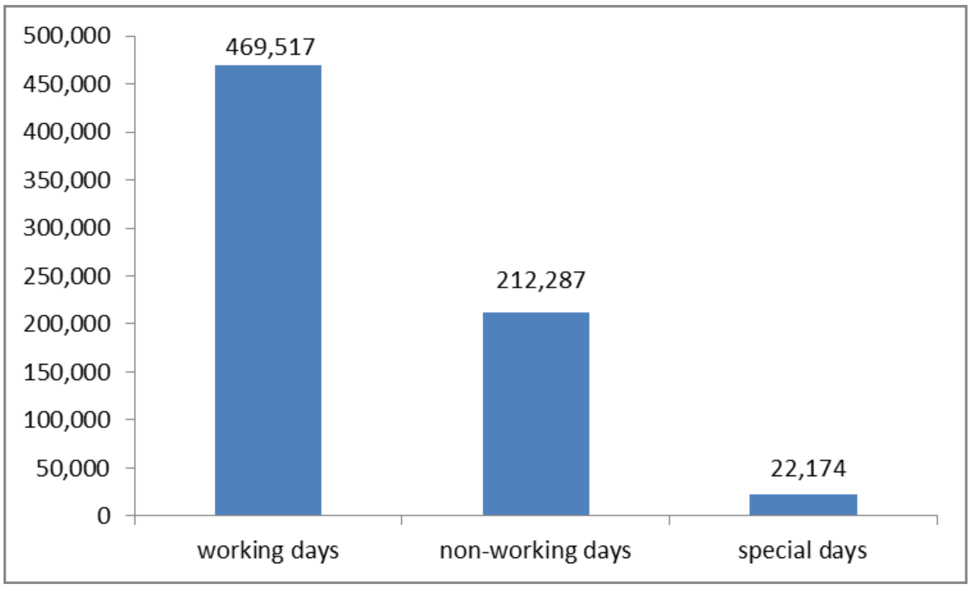


Fig. 6.4. Distribution of original data samples in terms of day type from 01/09/2010 to 29/02/2012.

TABLE 6.4. THE PERIODS SELECTED IN THE FIRST STAGE.

Period number	Start	End
1	02-Sep-2010 00:00:00	15-Sep-2010 23:59:00
2	24-Sep-2010 00:00:00	14-Oct-2010 23:59:00
3	09-Nov-2010 00:00:00	22-Nov-2010 23:59:00
4	27-Dec-2010 00:00:00	09-Jan-2011 23:59:00
5	11-Jan-2011 00:00:00	31-Jan-2011 23:59:00
6	09-Feb-2011 00:00:00	01-Mar-2011 23:59:00
7	11-Mar-2011 00:00:00	31-Mar-2011 23:59:00
8	02-Jun-2011 00:00:00	22-Jun-2011 23:59:00
9	08-Jul-2011 00:00:00	01-Sep-2011 23:59:00
10	14-Oct-2011 00:00:00	27-Oct-2011 23:59:00
11	05-Nov-2011 00:00:00	23-Dec-2011 23:59:00
12	29-Dec-2011 00:00:00	11-Jan-2012 23:59:00
13	19-Jan-2012 00:00:00	08-Feb-2012 23:59:00

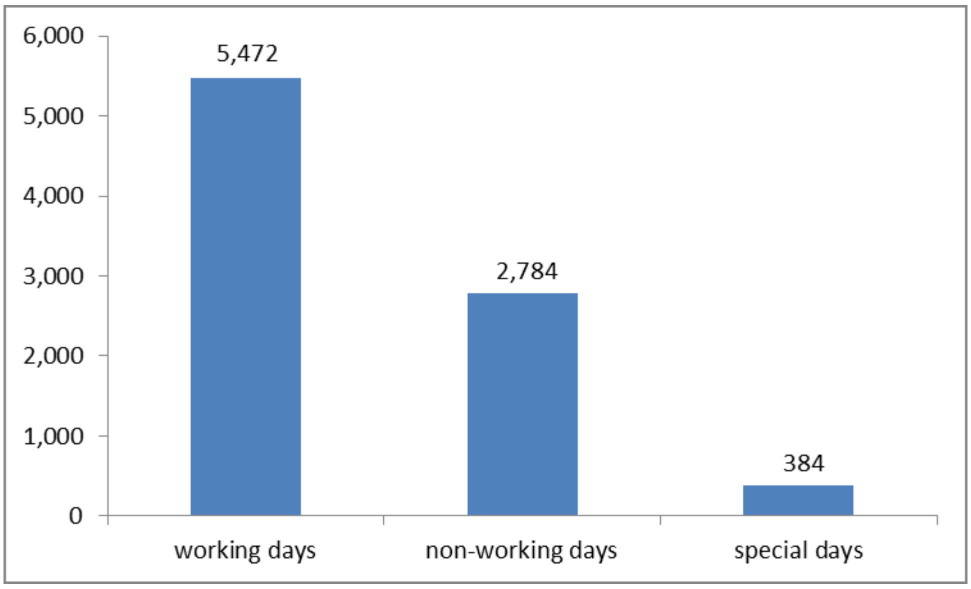


Fig. 6.5. Distribution of samples in data set ***D*** in terms of day type.

6.2.3.2. Design Experiments

Based on the model design cycle described in Section 2.7.4, several designs were conducted in such a way that their results led to the definition of a new design, by redefining variables and their corresponding lag terms, as well as imposing restrictions on objectives.

In a first step, we conducted designs with features requiring lag terms spread over at most 7 days.

After analyzing and comparing the results with those obtained in [54], the spread of lags was reduced to cover at most 2 days, and finally to cover at most one day. Based on that, 4 new designs were carried out.

For all designs, data set ***D***, stated in section 6.2.3.1, containing 8640 samples was used. Since a sampling interval of 15 minutes was used, and the objective was to obtain forecasts of electric power 1 hour-ahead, a prediction horizon of 4 steps was employed. In this work, as in [17], two groups of RBFNN models were considered. The first group contains simple models where only weather variables are used as exogenous variables. The second group considers complete models involving both weather and solar cooling operation variables. The list of candidate variables used and the range of lags for the design experiments are given in Table 6.5 and 6.6, respectively.

TABLE 6.5. LIST OF VARIABLES USED.

Variable	Notation	Unit	Range in D
Electric power demand added up with the electric power supplied by the PV plant	$x1$	kW	$[11.73,74.65]$
Day type (working day/non-working day/semi-holidays)	$x2$	-	$\{0, 0.5, 1\}$
Outdoor temperature	$x3$	℃	$[\,2.73,43.79]$
Outdoor solar radiation	$x4$	W/m^2	$[0, 1127.81]$
State of pump B1.1 (Off/On)	$x5$	-	$\{0,1\}$
State of Pump B1.2 (Off/On)	$x6$	-	$\{0,1\}$
State of Pump B2.1 (Off/On)	$x7$	-	$\{0,1\}$
State of Pump B2.2 (Off/On)	$x8$	-	$\{0,1\}$
State of Pump B7 (Off/On)	$x9$	-	$\{0,1\}$
State of the boiler (Off/On)	$x10$	-	$\{0,1\}$
State of the absorption machine (Off/On)	$x11$	-	$\{0,1\}$
State of the cooling tower (Off/On)	$x12$	-	$\{0,1\}$
State of the heat pump (Off/On)	$x13$	-	$\{0,1\}$

TABLE 6.6. DESCRIPTION OF THE LAGS USED.

Variable	Experiment I	Experiment II	Experiment III	Experiment IV
$x1$	20 lags over 1 day	20 lags over 1 day	20 lags over 1 day	20 lags over 1 day
$x2$	0 lags	0 lags	0 lags	0 lags
$x3$	20 lags over 1 day	20 lags over 1 day	20 lags over 1 day	20 lags over 1 day
$x4$	20 lags over 1 day	20 lags over 1 day	20 lags over 1 day	20 lags over 1 day
$x5$	-	-	1 lag	1 lag
$x6$	-	-	1 lag	1 lag
$x7$	-	-	1 lag	1 lag
$x8$	-	-	1 lag	1 lag
$x9$	-	-	1 lag	1 lag
$x10$	-	-	1 lag	1 lag
$x11$	-	-	1 lag	1 lag
$x12$	-	-	1 lag	1 lag
$x13$	-	-	1 lag	1 lag

As it can be seen in Table 6.6, Experiments I and II correspond to simple models in which only weather variables have been used; Experiments III and IV consider complete models. In Table 6.6, “lag 0” for variable “day type” ($x2$) is translated into the day type of instant $k + 1$

for which the electric power demand is predicted. In fact, weather and electric power demand variables are strongly related to their most recent values and also, to a certain extent, to their values 24 h before. As a result, for x_1, x_3 and x_4 a heuristic, proposed in [24], was used to select 20 lags over one full day, in such a way that more recent values predominate in the set of searchable lags for these variables. Hence, based on this heuristic, the 20 lags used are $[1, 2, 3, 4, 5, 6, 7, 9, 11, 13, 16, 20, 24, 29, 36, 43, 53, 65, 79, 96]$. In this list, and as an example, lags 1 and 2 denote delays of 15 and 30 minutes, respectively. The objectives and the corresponding goals are given in Table 6.7. $\boldsymbol{D}^t$, $\boldsymbol{D}^g$ and $\boldsymbol{D}^s$ denote the training, testing and simulation sets, respectively. $\varepsilon(\boldsymbol{D}^t)$ and $\varepsilon(\boldsymbol{D}^g)$ refer to the RMSE of $\boldsymbol{D}^t$ and $\boldsymbol{D}^g$, respectively. $\varepsilon(\boldsymbol{D}^s, 16)$ is a vector of RMSEs of $\boldsymbol{D}^s$ over a prediction horizon with 16 steps (i.e., one hour) so that the first element of the vector corresponds to the RMSE of 1-step-ahead prediction and the last one corresponds to the RMSE of 16-steps-ahead prediction. $O(\mu)$ denotes the model complexity, which is equal to the number of input features + 1, multiplied by the number of hidden neurons (i.e., for further information about the objectives, please refer to Section 2.7.2.1).

TABLE 6.7. OBJECTIVES AND THEIR CORRESPONDING RESTRICTION OF EXPERIMENTS.

	Experiment I	Experiment II	Experiment III	Experiment IV
$\varepsilon(\boldsymbol{D}^t)$	Minimize	< 0.059	Minimize	< 0.054
$\varepsilon(\boldsymbol{D}^g)$	Minimize	< 0.061	Minimize	< 0.052
$\varepsilon(\boldsymbol{D}^s, 16)$	Minimize	Minimize	Minimize	Minimize
$O(\mu)$	Minimize	< 317	Minimize	< 444

Regarding MOGA's parameters specification, for experiments I and III, the range $[d_m, d_M]$, where d_m and d_M are the minimum and maximum number of features, was set to [1, 30] while for experiments II and IV they were set to [1, 15] and [1, 21], respectively. Similarly, for experiments I and III, the range $[n_m, n_M]$, where n_m and n_M are the minimum and maximum number of neurons, was set to [2, 30] while for experiments II and IV, these ranges were set to $[1,\ 18]$ and $[1, 21]$, repectively. For all designs, the population size and the number of generations were set to 100.

For each experiment, a proper sub dataset $\boldsymbol{D}^W$ was derived from data set $\boldsymbol{D}$ whose features are those columns of $\boldsymbol{D}$ which correspond to the lags defined in the corresponding experiment.

In order to generate training, testing and validation sets for each experiment, firstly the ApproxHull algorithm proposed in Section 4.3 was applied on corresponding $\boldsymbol{D}^W$ to obtain convex points reflecting the whole input-output range in which the model is supposed to be used. Secondly, 50% of whole samples in $\boldsymbol{D}^W$ were used to generate training ($\boldsymbol{D}^t$), testing ($\boldsymbol{D}^g$) and validation ($\boldsymbol{D}^v$) sets with proportions of 60%, 20% and 20%, respectively. In this step all convex points were incorporated in the training set. Afterwards, the remaining samples were shared randomly into the rest of the training set, and the testing and validation sets. Regarding the simulation dataset $\boldsymbol{D}^s$, 1344 consecutive samples from 01-Oct-2010 00:00:00 to 14-Oct-2010 23:59:00 were considered. In this set, the rows correspond to the variables used, whose samples are in each column while, for the other sets, the number of rows correspond to the patterns, and the number of columns to the features. The size of training, testing and validation datasets as well as the simulation dataset of each experiment is given in Table 6.8.

TABLE 6.8. SIZE OF TRAINING, TESTING AND VALIDATION SETS.

	Experiment I	Experiment II	Experiment III	Experiment IV
$\boldsymbol{D}^t$	2592 x 62	2592 x 62	2592 x 71	2592 x 71
$\boldsymbol{D}^g$	864 x 62	864 x 62	864 x 71	864 x 71
$\boldsymbol{D}^v$	864 x 62	864 x 62	864 x 71	864 x 71
$\boldsymbol{D}^s$	4 x 1344	4 x 1344	13 x 1344	13 x 1344

After one run of the MOGA for each experiment, the non-dominated and preferred sets of models were generated. In the case that no restriction is considered on objectives, the non-dominated set is the same as preferred set; otherwise, the preferred set is a subset of the non-dominated set whose solutions satisfy the goals. Please refer to [47] for further information about how the preferred set can be obtained from the non-dominated set by applying the preferably criterion. The number of models in non-dominated and preferred sets for each experiment is given in Table 6.9.

TABLE 6.9. SIZE OF NON-DOMINATED AND PREFERRED SETS.

	Non-dominated set	Preferred set
Experiment I	346	346
Experiment II	238	88
Experiment III	289	289
Experiment IV	366	182

6.2.4. Results and discussion

The models presented in this case study were tested and compared by means of real data acquired at the CIESOL building. To do that, a battery of tests was selected according to certain representative characteristics, such as, the type of day (working and non-working days), the season of the year and the quantity of solar radiation (sunny and cloudy days). A complete description of the battery of tests is shown in Table 6.10. Furthermore, a prediction horizon over 1 hour was set mainly due to the energy price changes and the dynamic behaviour of indoor temperature [54].

Since in MOGA related experiments, the data was used with a sampling interval of 15 minutes, each test in Table 6.10 contains 96 samples. Moreover, the corresponding prediction horizon over 1 hour is equal to 4 steps. For the model proposed in [54], each test includes 1440 samples due to the 1 minute sampling rate. Hence, the corresponding prediction horizon over 1 hour is equal to 60 steps. For convenience, the complete model proposed in [54] and the models obtained by MOGA will be denoted as PREVIOUS and MOGA models, respectively. In order to compare the MOGA models obtained from each experiment with the PREVIOUS model, one model was selected from the non-dominated/preferred set, with a good compromise between performance and complexity.

TABLE 6.10. BATTERY OF TESTS PERFORMED.

Test	Day	Temperature	Radiation	Date (mm/dd/yyyy)
(A)	Working day	Summer	Sunny	06/29/2011
(B)	Non-working day	Summer	Sunny	09/19/2010
(C)	Working day	Winter	Cloudy	02/15/2011
(D)	Non-working day	Winter	Sunny	02/20/2011
(E)	Non-working day	Winter	Cloudy	02/28/2011
(F)	Non-working day	Summer	Cloudy	07/02/2011

In our work, models I, II, III and IV were the selected MOGA models from experiments I, II, III and IV, respectively. Information about the selected MOGA models as well as the PREVIOUS is given in Table 6.11. Using the notation of Table 6.6, the formal description of models I to IV is given by Eqs. (6.2) to (6.5), respectively.

TABLE 6.11. SELECTED MOGA MODELS AND PREVIOUS MODEL.

	Number of features	Number of neurons	Complexity
Model I	18	13	247
Model II	14	18	270
Model III	29	11	330
Model IV	18	20	380
NARX-MLP	67	10	680

$$\begin{aligned}\hat{y}(k+1) = f_1(&x_1(k),\dots,x_1(k-6),x_1(k-8),x_1(k-11),x_1(k-12),x_1(k-19),\\ &x_2(k+1),\\ &x_3(k-2),\ x_3(k-7),x_3(k-10),\\ &x_4(k-4),x_4(k-10),x_4(k-17))\end{aligned} \tag{6.2}$$

$$\begin{aligned}\hat{y}(k+1) = f_2(&x_1(k),\dots,x_1(k-4),x_1(k-6),x_1(k-9),x_1(k-10),x_1(k-15),x_1(k\\ &-18),\\ &x_3(k-9),\\ &x_4(k),x_4(k-8),x_4(k-18))\end{aligned} \tag{6.3}$$

$$\begin{aligned}\hat{y}(k+1) = f_3(&x_1(k-1),x_1(k-3),x_1(k-4),x_1(k-5),x_1(k-7),x_1(k-10),\\ &x_1(k-11),x_1(k-12),x_1(k-14),x_1(k-15),x_1(k-16),\\ &x_2(k+1),\\ &x_3(k),x_3(k-2),x_3(k-3),x_3(k-4),x_3(k-8),x_3(k-12),x_3(k-13),\\ &x_3(k-15),x_3(k-16),\\ &x_4(k-2),x_4(k-3),x_4(k-5),x_4(k-7),x_4(k-12),\\ &x_7(k),x_{11}(k),x_{13}(k))\end{aligned} \tag{6.4}$$

$$\begin{aligned}\hat{y}(k+1) = f_4(&x_1(k),x_1(k-1),x_1(k-2),x_1(k-3),x_1(k-5),x_1(k-17),\\ &x_2(k+1),\\ &x_3(k-18),\\ &x_4(k-3),x_4(k-5),x_4(k-10),x_4(k-14),x_4(k-15),x_4(k-18),\\ &x_9(k),x_{10}(k),x_{11}(k),x_{13}(k))\end{aligned} \tag{6.5}$$

$\hat{y}(k+1)$ in Eqs. (6.2) to (6.5) is the output of the corresponding RBFNN model. Each function f_j, $\{j = 1,2,3,4\}$ has its own set of input terms. These input terms, all together, constitute the input data sample at instant k.

To compare MOGA models with the PREVIOUS model over the battery of tests stated in Table 6.10, five statistical criteria were considered: MAE, MRE, MAPE, MaxAE and σ (introduced in Section 2.5). The evaluations of MOGA and PREVIOUS models over the battery of tests for a prediction horizon of 1 hour are given in Tables 6.12 to 6.17. The best values for each criterion are identified in bold.

TABLE 6.12. RESULTS OBTAINED BY MOGA AND PREVIOUS MODELS OVER TEST A, FOR A PH OF 1 HOUR.

	Model I	Model II	Model III	Model IV	PREVIOUS
MAE(kW)	**1.92**	2.14	2.28	3.55	1.96
MRE(kW)	**0.06**	0.08	0.07	0.12	**0.06**
MAPE(%)	**6.29**	8.11	7.66	12.39	6.38
MaxAE(kW)	12.36	14.22	**10.21**	13.82	10.99
σ (kW)	8.92	7.86	8.91	**6.99**	7.17

TABLE 6.13. RESULTS OBTAINED BY MOGA AND PREVIOUS MODELS OVER TEST B, FOR A PH OF 1 HOUR.

	Model I	Model II	Model III	Model IV	PREVIOUS
MAE(kW)	0.95	1.22	1.29	0.93	**0.84**
MRE(kW)	0.05	0.07	0.07	**0.05**	**0.05**
MAPE(%)	5.60	7.21	7.86	5.80	**5.13**
MaxAE(kW)	3.60	**3.15**	4.83	3.38	3.59
σ (kW)	2.01	2.48	1.78	1.75	**1.52**

TABLE 6.14. RESULTS OBTAINED BY MOGA AND PREVIOUS MODELS OVER TEST C, FOR A PH OF 1 HOUR.

	Model I	Model II	Model III	Model IV	PREVIOUS
MAE(kW)	1.99	3.46	**1.75**	1.95	1.86
MRE(kW)	0.06	0.1	**0.06**	**0.06**	**0.06**
MAPE(%)	6.62	10.55	**6.25**	6.40	6.26
MaxAE(kW)	8.82	16.56	**5.69**	7.04	8.15
σ (kW)	**6.04**	6.94	6.78	7.75	6.70

TABLE 6.15 RESULTS OBTAINED BY MOGA AND PREVIOUS MODELS OVER TEST D, FOR A PH OF 1 HOUR.

	Model I	Model II	Model III	Model IV	PREVIOUS
MAE(kW)	0.94	1.12	**0.82**	0.88	1.08
MRE(kW)	0.04	0.05	**0.03**	0.04	0.05
MAPE(%)	4.21	5.34	**3.81**	4.17	4.86
MaxAE(kW)	4.65	6.35	**5.20**	5.45	6.28
σ (kW)	1.95	1.64	**1.08**	1.72	1.52

TABLE 6.16. RESULTS OBTAINED BY MOGA AND PREVIOUS MODELS OVER TEST E, FOR A PH OF 1 HOUR.

	Model I	Model II	Model III	Model IV	PREVIOUS
MAE(kW)	1.38	1.45	**1.16**	1.30	1.49
MRE(kW)	0.06	0.06	**0.05**	**0.05**	0.06
MAPE(%)	6.00	6.30	**5.06**	5.77	6.38
MaxAE(kW)	**4.44**	5.59	4.81	4.49	6.89
σ (kW)	1.80	1.39	**1.28**	1.65	1.43

TABLE 6.17. RESULTS OBTAINED BY MOGA AND PREVIOUS MODELS OVER TEST F, FOR A PH OF 1 HOUR.

	Model I	Model II	Model III	Model IV	PREVIOUS
MAE(kW)	1.02	**0.80**	1.35	0.89	0.95
MRE(kW)	0.04	**0.03**	0.06	0.04	0.04
MAPE(%)	4.87	**3.70**	6.53	4.28	4.31
MaxAE(kW)	3.43	**2.63**	5.68	3.73	3.75
σ (kW)	1.95	1.99	**1.44**	1.97	1.88

Regarding test A, a working sunny day in summer, Model I, as a simple model, not only has minimum values in terms of MAE, MRE and MAPE among other MOGA models but also has a better performance than PREVIOUS in terms of these criteria. In this test, in overall, simple models I and II have better performance in comparison with complete models III and IV.

With respect to test B, a non-working sunny day in summer, Model IV, as a complete model, has minimum values of MAE, MRE and σ in comparison with other MOGA models; with respect to MaxAE, it has a compromise performance between Model II and PREVIOUS.

In test C, a working cloudy day in winter, and in test D, a non-working sunny day in winter, the complete model III has minimum values in terms of MAE, MAPE and MaxAE among all models. Model I, a simple model, has also a good performance; actually better in four criteria than the complete PREVIOUS model, in test D.

In test E, a non-working cloudy day in winter, both simple and complete MOGA models have lower values in terms of MAE, MAPE and MaxAE than the PREVIOUS model. Model III has better performance in all criteria.

Regarding test F, a non-working cloudy day in summer, simple model II and complete model IV have better performance in terms of MAE, MAPE and MaxAE than PREVIOUS model. In this comparison, model II has minimum values in all criteria, except σ.

According to Tables 6.12 to 6.17, in the group of simple models, model I, in most cases, has better performance than model II. In the group of complete models, model III, in most cases, is better than model IV. Figs. 6.6 to 6.8 show the comparison between measured and predicted value of electric power demand in CIESOL building, over tests A-F for a prediction horizon of 1 hour, for the PREVIOUS model, model I and III, respectively.

Comparing the performance of all MOGA models over the battery of tests, in general complete models III and IV have a better performance in winter than in summer, while simple model I has a compromise performance between summer and winter.

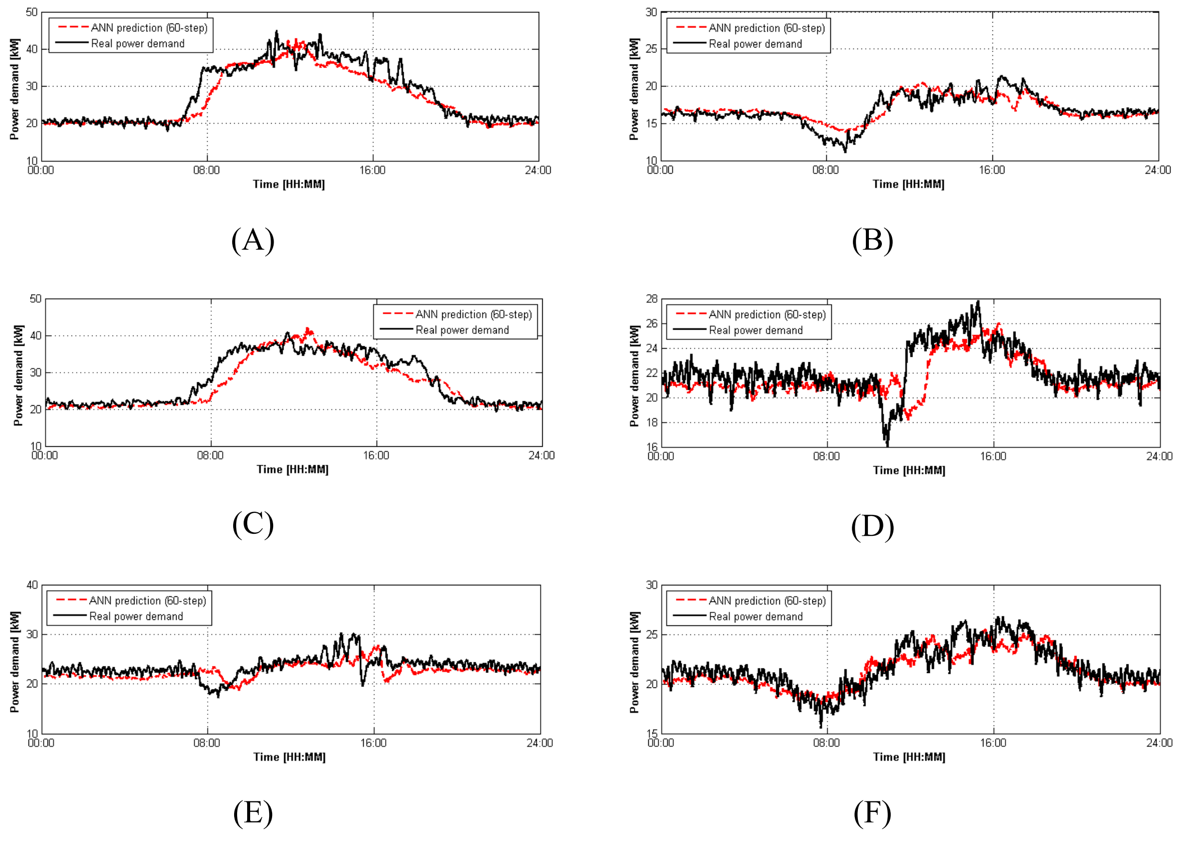
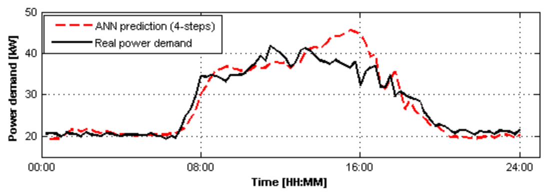
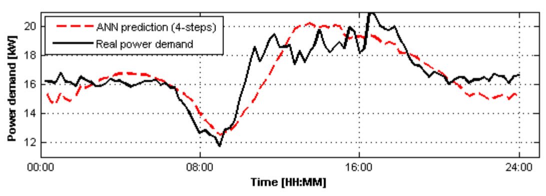


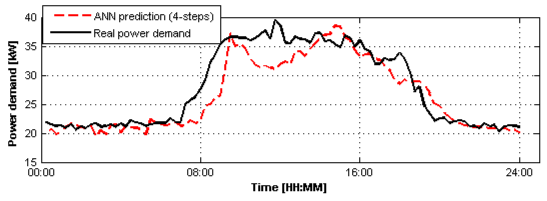
Fig. 6.6. Prediction results for tests A-F using the PREVIOUS model.



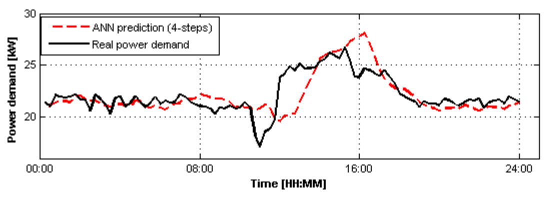
(A)



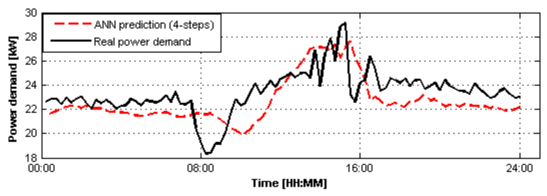
(B)



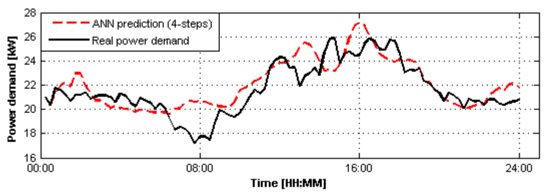
(C)



(D)

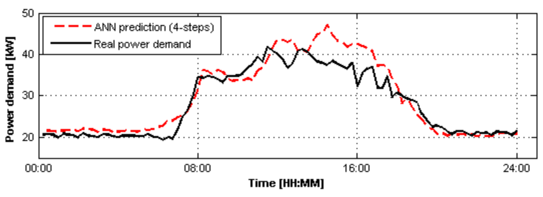


(E)

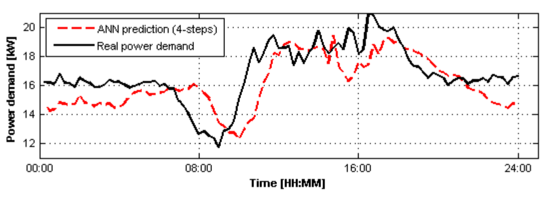


(F)

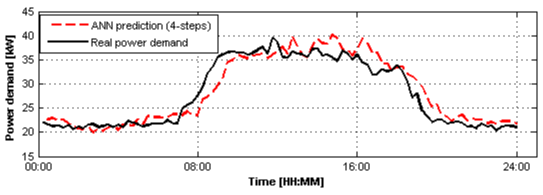
Fig. 6.7. Prediction results for tests A-F using model I.



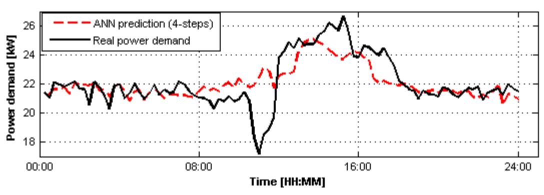
(A)



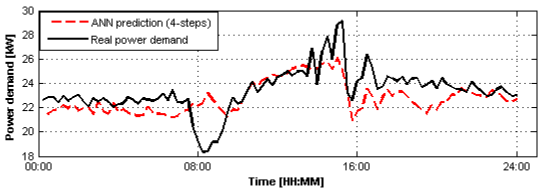
(B)



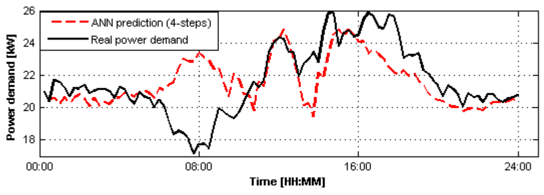
(C)



(D)



(E)



(F)

Fig. 6.8. Prediction results for tests A-F using model III.

6.2.4.1. Comparison of MOGA models with NAB approach

The performance of MOGA models was also compared with a Naive Autoregressive Baseline model, introduced in [158]. The NAB approach considers, as estimate of the electric power demand at instant *k*, the measured value of consumption at the correspond instant of time, in the same day of the previous week. It is therefore a simple model which does not need any computation to predict electric power demand at each time instant k. To apply the NAB approach to tests A-F, consecutive data corresponding to the previous week would be needed. Since there were several gaps in the whole dataset among tests A-F, only for tests D and E, corresponding to special days in winter, consecutive data exist to implement this method. In order to evaluate the NAB model in summer, we considered another special day in summer, corresponding to 06-Aug-2011, hereinafter called test G. For convenience, the description of the tests D, E and G is given in Table 6.18.

TABLE 6.18. BATTERY OF TESTS PERFORMED TO COMPARE THE NAB MODEL WITH THE NEURAL NETWORKS MODELS.

Test	Day	Temperature	Radiation	Date (mm/dd/yyyy)
(D)	Non-working day	Winter	Sunny	02/20/2011
(E)	Non-working day	Winter	Cloudy	02/28/2011
(G)	Non-working day	Summer	Sunny	08/06/2011

In order to compare the performance of NAB model with MOGA models and PREVIOUS model, the three models were evaluated over the battery of tests stated in Table 6.18. The results obtained over tests D, E and G are given in Tables 6.19 to 6.21. Please note that the results of MOGA models and PREVIOUS model, for tests D and E, are obtained from Tables 6.15 and 6.16, respectively, and are reproduced here for easy of comparison with the NAB approach.

TABLE 6.19. RESULTS OBTAINED BY NEURAL NETWORK AND NAB MODELS OVER TEST D, FOR A PH OF 1 HOUR.

	Model I	Model II	Model III	Model IV	PREVIOUS	NAB
MAE (kW)	0.94	1.12	**0.82**	0.88	1.08	1.9439
MRE (kW)	0.04	0.05	**0.03**	0.04	0.05	0.0856
MAPE (%)	4.21	5.34	**3.81**	4.17	4.86	8.5575
MaxAE (kW)	**4.65**	6.35	5.20	5.45	6.28	6.8341
σ (kW)	1.95	1.64	**1.08**	1.72	1.52	1.8933

TABLE 6.20. RESULTS OBTAINED BY NEURAL NETWORK AND NAB MODELS OVER TEST E, FOR A PH OF 1 HOUR.

	Model I	Model II	Model III	Model IV	PREVIOUS	NAB
MAE (kW)	1.38	1.45	**1.16**	1.30	1.49	4.8314
MRE (kW)	0.06	0.06	**0.05**	**0.05**	0.06	0.2086
MAPE (%)	6.00	6.30	**5.06**	5.77	6.38	20.8610
MaxAE (kW)	**4.44**	5.59	4.81	4.49	6.89	13.0946
σ (kW)	1.80	1.39	**1.28**	1.65	1.43	5.6539

TABLE 6.21. RESULTS OBTAINED BY NEURAL NETWORK AND NAB MODELS OVER TEST G, FOR A PH OF 1 HOUR.

	Model I	Model II	Model III	Model IV	PREVIOUS	NAB
MAE (kW)	0.8297	1.2684	0.8089	**0.7598**	0.7787	3.2966
MRE (kW)	0.0472	0.0745	0.0465	0.0434	**0.0432**	0.1909
MAPE (%)	4.7154	7.4521	4.648	4.3363	**4.3154**	19.0867
MaxAE (kW)	3.7347	7.4701	4.7188	3.8647	**2.9473**	13.6549
σ (kW)	2.08	2.216	1.5135	**1.2575**	1.822	3.8805

Regarding these tests, the NAB model has the worst performance (by a large difference) in comparison to MOGA and PREVIOUS models, in terms of all criteria.

Regarding test G, a new test corresponding to a non-working sunny day in summer, Model IV, a complete model, has minimum values in terms of MAE and σ. In terms of MRE and MAPE, Model IV has approximately the same performance as PREVIOUS model. In the same way as in tests D and E, the NAB model has the worst performance.

To sum up, comparing the performance of MOGA models and the PREVIOUS, despite the fact that MOGA models were trained with a small training set of 2592 samples compared to the 318340 samples used to train the PREVIOUS model, they have obtained better results, except in Test B. Moreover, as it can be seen in Table 6.11, the complexity of models obtained from MOGA is lower than the PREVIOUS model.

According to tests D, E and G reflecting special days in winter and summer, both MOGA and PREVIOUS models have much better performance than the NAB model in terms of all criteria.

6.3. Case Study 2: An Intelligent Weather Station

Since accurate measurements of global solar radiation, atmospheric temperature and relative humidity as well as the ability of evaluating their predictions over time, are important for different areas of applications, an intelligent weather station was developed by the University of Algarve. For implementing the predictions, two groups of models were proposed. The first group involved predictive models based on nearest-neighbors (NEN) algorithm whereas the second group included NAR RBFNN models designed by MOGA.

The NEN models use pattern matching to compute the predictions. They need two parameters d and k where d denotes the number of full days used to search the best matching patterns and n corresponds to the number of closest neighbors that are be averaged to compute one-step-ahead prediction. To design the models, data was collected by sampling 5 minutes between 22-Feb-2015 and 7-Apr-2015. Totally, 12,800 samples were obtained. The first 35 days were used to compute predictions by NEN models over prediction horizon with 48 steps (4 hours). For each climate variable, the first 10,000 samples were considered to design the corresponding RBFNN models by MOGA. In this case, firstly, ApproxHull was applied on the whole data set and then training set containing convex hull points and random samples as well as testing and validation sets were generated. The size of training, testing and validation set for each climate variable is given in Table 6.22.

The last 1350 samples were considered to evaluate both groups of models over prediction horizon with 48 steps. The evaluation results of predictive models of each climate variable are given in Tables 6.23 to 6.25.

TABLE 6.22. SIZE OF TRAINING, TESTING AND VALIDATION SETS FOR THE ATMOSPHERIC CLIMATE MODELS.

	Training	Testing	Validation	Convex hull points
Atmospheric Air temperature	2888 x 74	962 x 74	964 x 74	696 x 74
Atmospheric Relative Humidity	2888 x 74	962 x 74	964 x 74	696 x 74
Global Solar Radiation	2895 x 74	965 x 74	966 x 74	659 x 74

TABLE 6.23. ATMOSPHERIC TEMPERATURE.

	$RMSE_1$	$\sum_{i=1}^{48} RMSE_i$
NEN(2,2)	2.17	111.06
NEN(7,4)	1.93	101.52
NEN(35,4)	1.39	84.79
RBFNN	0.30	65.46

TABLE 6.24. ATMOSPHERIC RELATIVE HUMIDITY.

	$RMSE_1$	$\sum_{i=1}^{48} RMSE_i$
NEN(2,2)	14.34	742.17
NEN(7,4)	11.32	632.72
NEN(21,4)	8.52	497.36
RBFNN	0.99	409.43

TABLE 6.25. GLOBAL SOLAR RADIATION.

	$RMSE_1$	$\sum_{i=1}^{48} RMSE_i$
NEN(2,2)	132.22	12109
NEN(7,4)	122.26	12173
NEN(14,4)	154.67	11951
RBFNN	29.49	7850

As it can be seen in Tables 6.23 to 6.25, for all climate variables, RBFNN model is superior to its corresponding NEN models in terms of one-step-ahead RMSE and of the summation of RMSE over the prediction horizon (48 steps).

Correspondingly, Figs. 6.9-6.11 show the one-step-ahead predictions of climate variables obtained by the NEN algorithm and the RBFNN for the last 1350 samples, as well as the evolution of the RMSE along the prediction horizon with 48 steps. As it can be seen in Figs 6.9-6.11, it is clear that the best performance is obtained by the RBFNN model.

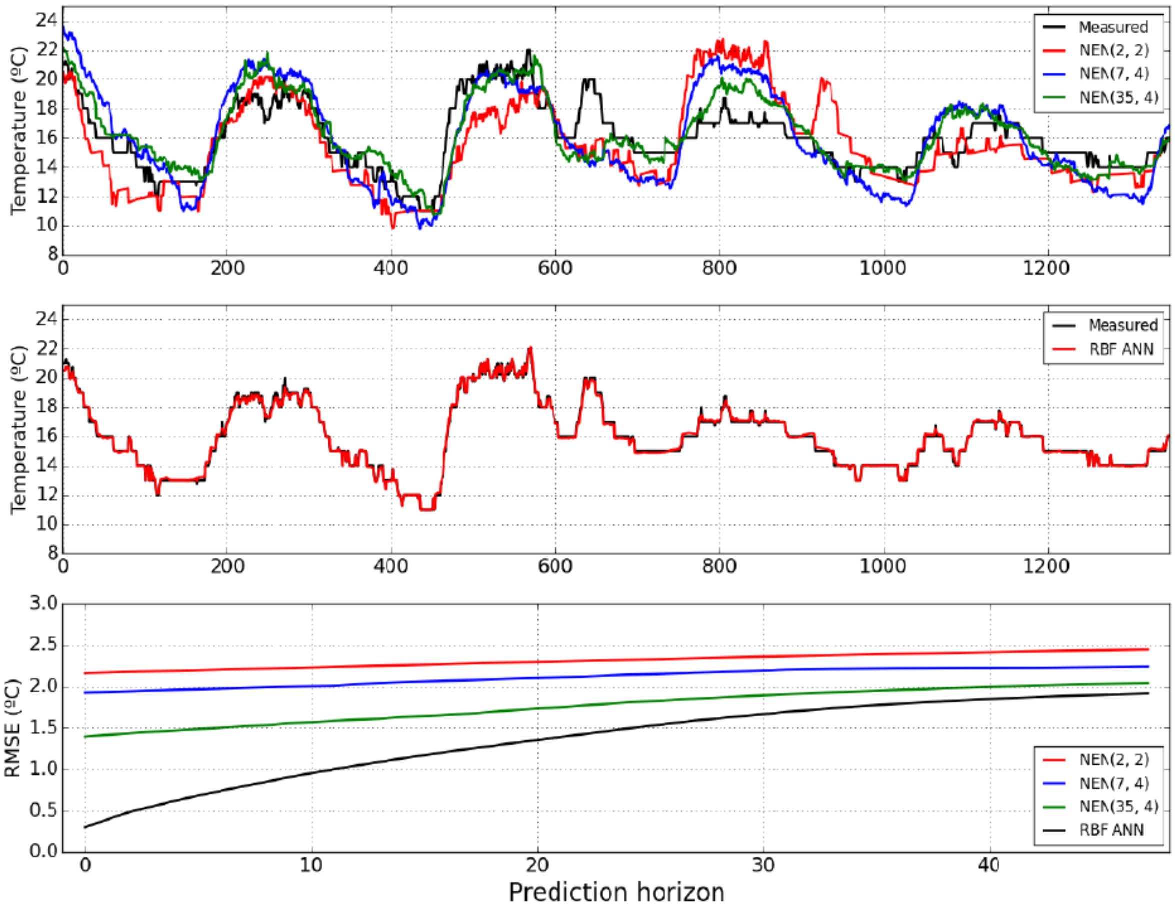


Fig. 6.9. One-step-ahead prediction for the NEN algorithm and the RBFNN over the last 1350 samples as well as the evolution of the RMSE along the prediction horizon with 48 steps for atmospheric temperature.

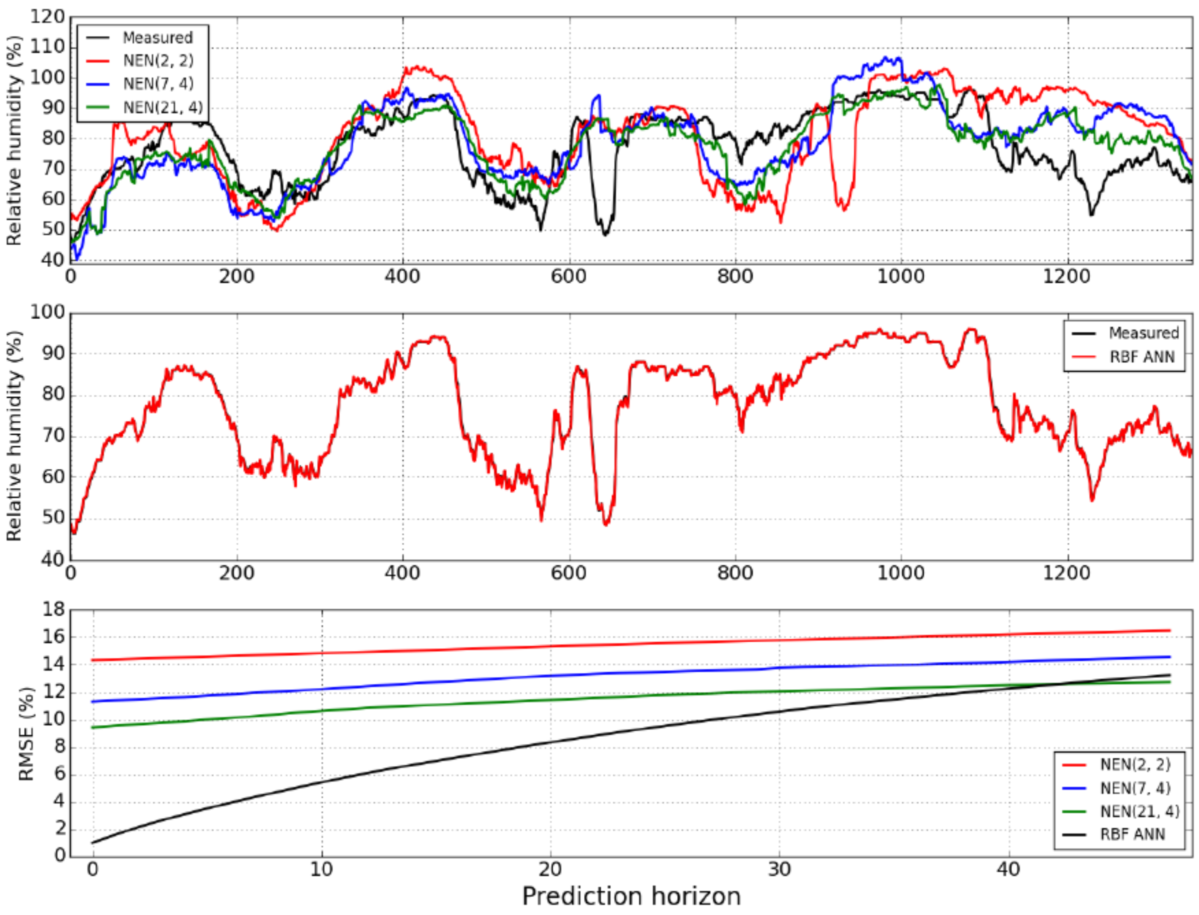


Fig. 6.10. One-step-ahead prediction for the NEN algorithm and the RBFNN over the last 1350 samples as well as the evolution of the RMSE along the prediction horizon with 48 steps for atmospheric relative humidity.

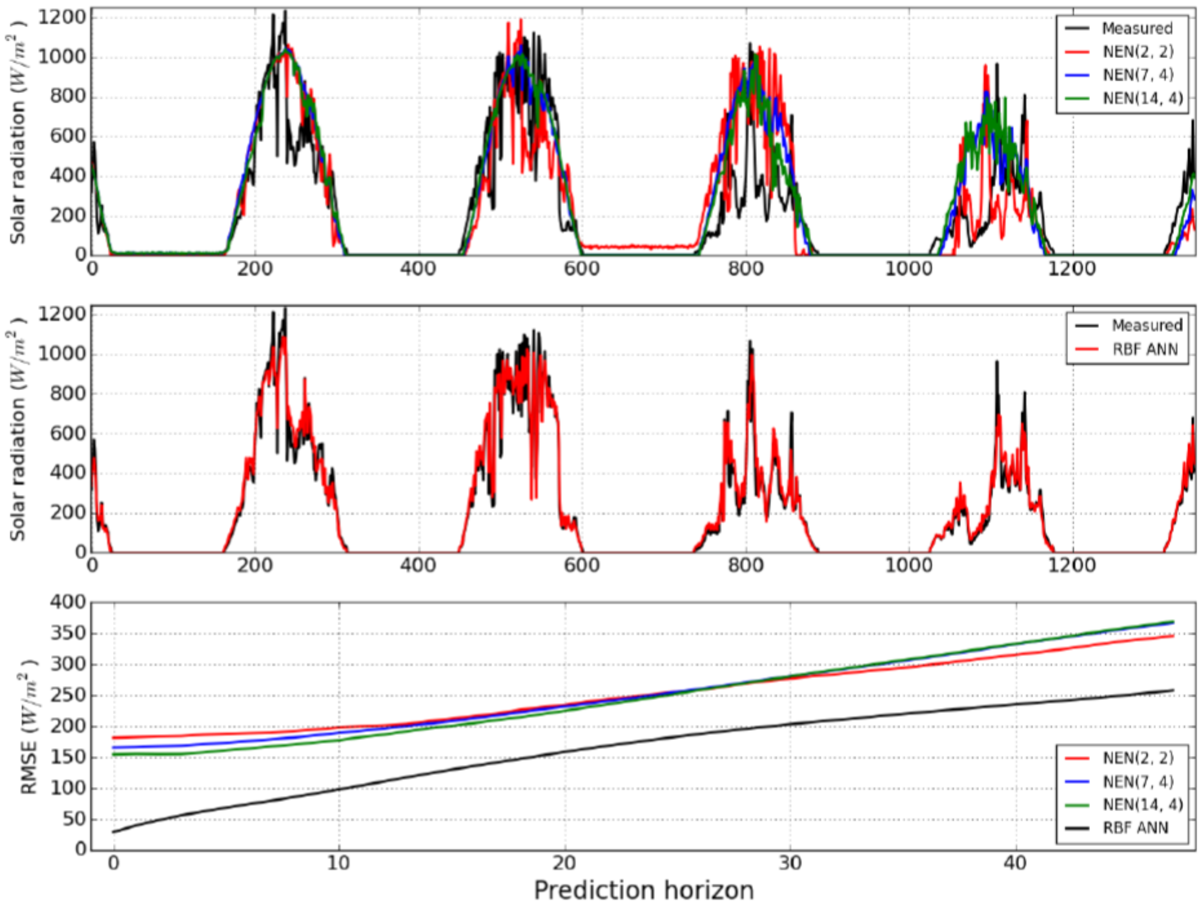


Fig. 6.11. One-step-ahead prediction for the NEN algorithm and the RBFNN over the last 1350 samples as well as the evolution of the RMSE along the prediction horizon with 48 steps for global solar radiation.

6.4. Case Study 3: An Intelligent Support System for Automatic Diagnosis of Cerebral Vascular Accidents from Brain CT Images

In this case study, a RBFNN based diagnosis system for automatic identification of CVA through analysis of CT images was considered. Totally 1,867,602 samples with 52 features were extracted from 150 CT images, for which a collaborating Neuroradialogist registered his opinions. To design RBFNN models, MOGA was applied. Two experiments, Exp.1 and Exp.2, were carried out. In Exp.1 no restriction was imposed on MOGA objectives , while in Exp.2, restrictions on two objectives, FP (False Positive) and FN (False Negative) labels on the training set were imposed based on the results obtained from Exp.1. To design both experiments, ApproxHull was employed on the whole set (i.e., 1,867,602 samples) and resulted in 13,023 convex hull points. Afterwards a training set with 20,000 samples was created so that it included the convex hull points and 6,977 random samples. For testing and validation sets, each of them involved 6,666 random samples.

Exp.1 resulted in a non-dominated set of 406 RBFNN models whereas from Exp.2 a non-dominated set of 281 RBFNN models was obtained, from where 69 models were in the preferred set. To compare Exp.1 and Exp.2, the best model from each of them was selected, using a threshold on FP and FN in the whole set (i.e., 1,867,602 samples). Exp.1 and Exp.2 were additionally compared with an ensemble of Exp.2 preferable models, where the classification output was obtained based on the majority of models outputs in the preferred set of Exp.2.

Table 6.26 shows the evaluation results of Exp.1 and Exp2 as well as the ensemble of Exp.2 in terms of specificity and sensitivity (Please refer to Section 2.5) on the whole set. As it can be observed in Table 6.26, Exp.2 has better performance in comparison with Exp.1. The ensemble of Exp.2 resulted in the best performance in comparison with Exp.1 and Exp.2.

TABLE 6.26. EVALUATION RESULTS

	Specificity	Sensitivity
Best model in EXP.1	97.04	97.12
Best model in EXP.2	97.60	97.66
Ensemble of EXP.2	98.01	98.22

6.5. Conclusions

The experimental results obtained from these case studies demonstrated the applicability of ApproxHull, as a data selection method, on real data in high dimensions where for the first, second and third case studies, Approxhull was applied on the corresponding whole data sets with the maximum of 71, 145 and 52 features, respectively. In all cases, MOGA was used for model design, employing the data partitions given by ApproxHull. According to the results obtained from the first case study, MOGA models achieved better performance than MLP models, designed using a much larger training set. The results achieved from the second case study showed that MOGA achieved an excellent predictive performance, much better, for the weather models, than the NEN approach. The third case study proved that the ApproxHull can be applied successfully on very large size data sets (nearly 2 million samples).

7. Comparing four data selection methods for off-line model design

7.1. Introduction

This chapter aims to compare four data selection methods including the Random Data Selection (RDS) method, the Convex hull Based Data Selection (CBDS) method, the Entropy Based Data Selection (EBDS) method and the Hybrid Data Selection (HDA) method. In this study, the ApproxHull method introduced in Section 4.3 was considered as the CBDS method. Regarding the EBDS method, the method proposed in [13] was applied which is one of the latest efforts of using information theory in data selection. The methods were applied on eight benchmarks: four binary class classification problems and, the other four related to regression problems. The experiments were organized in three groups.

In the first group, for one classification problem (named *Breast Cancer*) and one regression problem (named *Bank*), five runs of the MOGA were executed for each of the data selection methods. Each MOGA execution resulted in a non-dominated set of RBFNN models. In this case the MOGA selects the number of neurons and the inputs of the models in order to minimize the objectives described in (please see Sections 2.7.1 and 2.7.2)

In the second and third groups of experiments MLPs were considered for all regression problems and SVMs for all classification problems, respectively. In these cases, for each benchmark problem the four data selection methods were applied, repeating the execution 10 times in each case. In this case the structure of the models was fixed beforehand and was the same for the 10 executions.

The rest of this chapter is organized as follows: In Section 7.2, the entropy based data selection method is briefly introduced. The procedure for constructing the training, testing and validation sets for experiments is described in Section 7.3. Regarding the MOGA experiments, the performance of the methods was analyzed based on two scenarios; the best model scenario and the ensemble scenario. The experiments are detailed in Section 7.4. The simulation results obtained from the evaluation of the methods in all experiments are discussed in Section 7.5. Finally, some conclusions are given in Section 7.6.

7.2. An entropy based unsupervised data selection method

The main idea behind the EBDS method proposed in [13] is selecting k samples of a given data set $\boldsymbol{D}$ for training set so that the information content and the diversity of data in the training set used to adjust the model parameters is maximized. This method benefits from the information entropy of any random variable $\boldsymbol{X}$ given in Eq. (7.1).

$$H(\boldsymbol{X}) = \sum_{i=1}^{N} p(x_i) I(x_i) \tag{7.1}$$

where N is the number of all possible observations of $\boldsymbol{X}$. $p(x_i)$ denotes the probability that $\boldsymbol{X}$ takes value x_i and $I(x_i)$ denotes the information content (also called self-information or surprisal) that $\boldsymbol{X}$ represents when it takes value x_i. $I(x)$ is defined as:

$$I(x) = -log_2\, p(x) \tag{7.2}$$

Suppose data set $\boldsymbol{D} = [\boldsymbol{X}|\boldsymbol{y}]$ consists of an input pattern matrix $\boldsymbol{X}$ of size $N \times d$ and a target vector $\boldsymbol{y}$ of size $N \times 1$. Each row of $\boldsymbol{D}$ is a point of dimension $(1\ \times (d+1))$, and assume that $\boldsymbol{z}_i$ refers to the ith point in $\boldsymbol{D}$. Since data set $\boldsymbol{D}$ represents a set of values of a multidimensional random variable $\boldsymbol{Z}$, $P(\boldsymbol{z}_i)$ is translated into the probability that $\boldsymbol{Z}$ takes $\boldsymbol{z}_i$. In this method, $P(\boldsymbol{z}_i)$ is estimated by Eq. (7.3) [165].

$$\hat{p}(\boldsymbol{z}_i) = \frac{1}{N}\sum_{j=1}^{N}[\prod_{l=1}^{d+1} k_{h_l}(\boldsymbol{z}_i[l] - \boldsymbol{z}_j[l])] \tag{7.3}$$

where $k_{h_l}(.)$ is a Gaussian kernel function whose bandwidth is h_l which is obtained by [165]:

$$h_l = \widehat{\sigma_l} N^{\frac{-1}{(d+1+4)}} \tag{7.4}$$

where $\widehat{\sigma_l}$ is the sample standard deviation along dimension l of the data.
By using Eq. (7.3) for each point in $\boldsymbol{D}$, vector $\hat{\boldsymbol{p}}$ is obtained as (7.5).

$$\hat{\boldsymbol{p}} = [\hat{p}(\boldsymbol{z}_1), \hat{p}(\boldsymbol{z}_2), \cdots \hat{p}(\boldsymbol{z}_N)] \tag{7.5}$$

Using Eq. (7.2) for each point in $\boldsymbol{D}$, vector $\hat{\boldsymbol{I}}$ is obtained as,

$$\hat{\boldsymbol{I}} = [\hat{I}(\boldsymbol{z}_1), \hat{I}(\boldsymbol{z}_2), \cdots \hat{I}(\boldsymbol{z}_N)] \tag{7.6}$$

where $\hat{I}(\boldsymbol{z}_i)$ denotes the self-information estimate which is presented by point $\boldsymbol{z}_i$.

Having $\hat{\boldsymbol{p}}$ and $\hat{\boldsymbol{I}}$ at hand, vector $\widehat{\boldsymbol{H}}$ is obtained as (7.7) by taking the Hadamard product of $\hat{\boldsymbol{p}}$ by $\hat{\boldsymbol{I}}$.

$$\widehat{\boldsymbol{H}} = [\hat{p}(\boldsymbol{z}_1)\hat{I}(\boldsymbol{z}_1), \hat{p}(\boldsymbol{z}_2)\hat{I}(\boldsymbol{z}_2), \cdots \hat{p}(\boldsymbol{z}_N)\hat{I}(\boldsymbol{z}_N)] \tag{7.7}$$

where $\hat{p}(\boldsymbol{z}_i)\hat{I}(\boldsymbol{z}_i)$ is considered as the information based fitness of point $\boldsymbol{z}_i$ reflecting the contribution of point $\boldsymbol{z}_i$ to the entropy obtained by Eq. (7.1).

Once vector $\widehat{\boldsymbol{H}}$ is obtained, k points are selected from $\boldsymbol{D}$ proportionally to their information fitness, by means of the Stochastic Universal Sampling (SUS) method. For the additional details please consult [13].

7.3. Construction of data sets for the experiments

To fairly compare the data selection methods, the existence of a common validation data set, $\boldsymbol{V}$, which does not have any contribution in model design, is needed. Notice, however, that in a practical case, each data selection method should be applied to the whole data set, $\boldsymbol{D}$. This is particularly relevant for the methods relying in convex hull (CBDS and HDS methods), as their rational is incorporating in the training set the convex hull points obtained from the whole data set.

In this chapter, as we aim to compare the performance of the data selection models in a common validation set, the procedure for constructing the data sets for all groups of experiments of each model type is as follows. First, for each group of experiments, a common validation set $\boldsymbol{V}$ containing N_v samples is randomly extracted from the whole data set $\boldsymbol{D}$; the remaining samples will constitute the set $\boldsymbol{DSG}$, from where in a second step training set $\boldsymbol{T}$ and testing set $\boldsymbol{G}$ containing N_t and $N_g$samples, respectively, will be extracted.

In RDS method, firstly, N_t samples are extracted randomly from $\boldsymbol{DSG}$ (i.e., resulting in a reduced set $\boldsymbol{D'}$) to construct $\boldsymbol{T}$. Subsequently**,** N_g samples are randomly extracted from $\boldsymbol{D'}$ to form $\boldsymbol{G}$.

Regarding the CBDS method, first *ApproxHull* is applied on the data set $\boldsymbol{DSG}$ to obtain the convex hull points. Afterwards, the convex hull points as well as some random samples are extracted from $\boldsymbol{DSG}$ (i.e., resulting in a reduced set $\boldsymbol{D'}$) to form $\boldsymbol{T}$ so that $N_t = N_{chv} + N_{rnd}$ where N_{chv} and N_{rnd} denote the number of convex hull points and random samples, respectively. Subsequently, as in the RDS method, N_g samples are randomly extracted from $\boldsymbol{D'}$ to form $\boldsymbol{G}$.

In the EBDS method, N_t samples are selected from $\boldsymbol{DSG}$ using the entropy based method mentioned in Section 4.2 to form $\boldsymbol{T}$. Then $\boldsymbol{T}$ is extracted from $\boldsymbol{DSG}$ resulting in a reduced set $\boldsymbol{D'}$ and set $\boldsymbol{G}$ is constructed in the same way as in the RDS method.

Finally, the idea behind the HDS method is combining the two previous data selection methods, CBDS and EBDS. In the first step of the HDS method, *ApproxHull* is applied on $\boldsymbol{DSG}$ to obtain the convex hull points which are extracted from $\boldsymbol{DSG}$ (i.e, resulting in a reduced set $\boldsymbol{D'}$) and included in $\boldsymbol{T}$. In the next step, $N_t - N_{chv}$ samples are extracted from $\boldsymbol{D'}$ using the EBDS method and included in $\boldsymbol{T}$**.** $\boldsymbol{G}$ is obtained from the rest of the samples in the same way as in the RDS method.

7.4. Experiments

In the first group of experiments, for the RBFNN models designed by the MOGA, two scenarios are considered: the *best model* and *ensemble* scenarios. As in the end of each MOGA run we have access to a set of non-dominated models, typically one model is selected out of this set. This scenario will be called *best model*.

The criterion for selecting the best model out of the non-dominated set for the regression problem is the minimum RMSE on the common validation set $\boldsymbol{V}$. In the case of classification problems the best model is selected on the basis of the Classification Rate (please see Section 2.5) in a procedure composed of the following three steps: first, the model which has the maximum $CR(\boldsymbol{V})$ is selected. In case of tie, the one with the maximum $CR(\boldsymbol{G})$ is chosen. Similarly, in case of tie, the one with the maximum $CR(\boldsymbol{T})$ is selected. Finally if more than one model is remained, one of them is randomly selected as the best model.

The second scenario, called *ensemble*, involves using all non-dominated solutions. In this scenario, for the regression problem, the output of the ensemble scheme is the average of all non-dominated models' outputs, whereas for the classification, the output of the ensemble scheme is determined based on the majority of all models' outputs in the non-dominated set. In this case, the class of an input pattern is the one in which the majority of the models in the non-dominated set are unanimous.

The second and third groups of experiments do not involve the MOGA and employ different kinds of models, MLPs for the four regression problems and SVMs for the four classification problems, respectively. From the second group, the MLP models obtained in the regression problem named *Bank*, trained with the modified LM algorithm introduced in [34, 166], are compared to the RBFNN MOGA generated models from the first group of experiments. Similarly, from the third group of experiments, the SVM models in the classification problem named *Breast Cancer*, trained using the Matlab implementation, are compared to the RBFNN MOGA generated models from the first group of experiments. For all models and experiments, the four data selection methods were used. The datasets were taken from the UCI repository [115]. Their names, number of samples (*N*) and inputs (*d*) are given in Table 7.1.

TABLE 7.1. DETAILS OF THE DATA SETS.

	Problem	*N*	*d*
Bank	Regression	8192	32
Puma	Regression	8192	32
Concrete	Regression	1030	8
Wine Quality	Regression	4898	11
Breast Cancer	Classification	569	30
Parkinson	Classification	1040	26
Satellite	Classification	2033	36
Letter	Classification	1555	16

The number of samples of $\boldsymbol{T}$, $\boldsymbol{G}$, and $\boldsymbol{V}$ sets, and the average number of convex hull points ($\overline{N}_{chv}$) obtained over all executions in the experiments of each problem, are given in Table 7.2.

TABLE 7.2. NUMBER OF SAMPLES OF $\boldsymbol{T}$, $\boldsymbol{G}$ AND $\boldsymbol{V}$ AND THE AVERAGE NUMBER OF CONVEX HULL POINTS.

		N_t	N_g	N_v	$\overline{N}_{chv}$
Regression problems	Bank	4195	1638	1639	3437
	Concrete	618	206	206	307
	Puma	4915	1638	1639	3686
	Wine Quality	3134	784	980	599
Classification problems	Breast Cancer	300	76	193	183
	Parkinson	550	136	354	280
	Satellite	1074	268	691	711
	Letter	822	204	529	564

Regarding the MOGA experiments parameterization, the same parameters were used in all experiments. The number of generations and the population size were both set to 100 and no restriction on objectives was considered. The range of the number of neurons was set to [2, 30] and the range of the number of features for Bank and Breast Cancer was set to [1, 32] and [1, 30], respectively. The early-stopping termination criteria within a maximum of 100 iterations were considered.

In terms of model structure, the MLP models in the second group of experiments had 2 hidden layers and used all features in the data sets as inputs. The number of neurons for each hidden layer for Bank and Puma problems was 10, while for the other problems was 5. For all MLP models, a maximum of 100 training iterations with early stopping method was considered. Regarding the SVM models for the binary class classification problems, all input features were used. The SVM hyper-parameters γ and C were set as stated in [68]. These are shown in Table 7.3.

TABLE 7.3. HYPER PARAMETERS OF SVM MODELS FOR THE CLASSIFICATION PROBLEMS.

	γ	**C**
Breast Cancer	0.05	1
Parkinson	0.1	200
Satellite	0.1142	500
Letter	0.6576	1

7.5. Experimental results

Considering the regression problem Bank in the first and second groups of experiments, the average of the RMSEs obtained on the common dataset $\boldsymbol{V}$ over the experiments, for the two MOGA result scenarios and for the MLP model, are given in Table 7.4.

TABLE 7.4. AVERAGE RMSES OBTAINED FOR DATASET BANK.

	RDS	CBDS	EBDS	HDS
Best model	0.1908	**0.1901**	0.1907	0.1903
Ensemble	0.1870	0.1872	**0.1869**	0.1878
MLP	0.1969	**0.1963**	0.1979	**0.1963**

As shown in Table 7.4, independently of the data selection method, MOGA optimized models are always better than MLP models, despite the latter being much more complex. In fact, MLPs have a model complexity (number of nonlinear parameters) of 440 while the MOGA generated RBFNNs have on average 100 (using the average number of input features and neurons shown in Table 7.5). Another conclusion that can be taken from Table 7.4 is that the ensemble scenario provides better performance than the best model.

TABLE 7.5. AVERAGE NUMBER OF FEATURES AND NEURONS OF THE BEST MOGA MODELS FOR DATASET BANK.

Method	Number of features	Number of neurons
RDS	24	4
CBDS	20	5
EBDS	25	4
HDS	25	4

Regarding all regression problems in the second group of experiments, where MLP models were considered, Table 7.6 shows the average RMSE obtained over the 10 executions.

TABLE 7.6. AVERAGE RMSE FOR THE REGRESSION PROBLEMS.

	RDS	CBDS	EBDS	HDS
Bank	0.1969	**0.1963**	0.1979	**0.1963**
Concrete	**0.1408**	0.1417	0.1458	**0.1408**
Puma	0.0687	**0.0671**	0.0676	0.0687
Wine Quality	0.2361	**0.2349**	0.2370	0.2370

Regarding the best data selection method, the bold values in Tables 7.4 and 7.6 denote the best performance, for each model type/problem. Although it seems to indicate that CBDS and HDS should be chosen as best, with a slightly advantage of the former, the average RMSEs might not be the only criterion for that selection.

To analyze the statistical validity of the results, two tests are used: a sign test, and a Wilcoxon signed-ranks test [59] (as presented in Sections 2.9.1 and 2.9.2). For the former, we counted, for each problem or group of problems, the number of times (C) that a data selection method (say j) had a better performance than another method (i), for each model type. For the latter test, the test value T is obtained using a rank based approach and then it is compared with its corresponding critical value (please see Section 2.9.2).

Tables 7.7 shows the $C(i,j)$ and T values, considering the Best and the Ensemble RBFNN models, for dataset Bank.

TABLE 7.7. *C*(*I*,*J*) /*T* FOR BANK – BEST AND ENSEMBLE MODELS.

C(i,j)/T	RDS	CBDS	EBDS	HDS
RDS		8/19	4/26.5	6/27
CBDS	2/19		4/21	4/20
EBDS	5/26.5	6/21		4/23
HDS	4/27	6/20	6/23	

Analyzing the results of Tables 7.4 and 7.7 shows the CBDS method is the best one. Statistically, however, according to the Wilcoxon test, no method can be considered better than the others, while according to the sign test (weaker than the Wilcoxon test), we can only say CBDS outperforms RDS method, with a level of significance of 10%.
Table 7.8 shows the $C(i,j)$ and T values for the 40 MLP regression experiments in the second group.

TABLE 7.8. C(I,J) /T FOR ALL MLP MODELS

C(i,j)/T	RDS	CBDS	EBDS	HDS
RDS		25/307	17/308.5	22/386.5
CBDS	13/307		12/238.5	16/306
EBDS	23/308.5	27/238.5		24/305.5
HDS	18/386.5	23/306	15/305.5	

Analyzing this table, CBDS should also be the chosen data selection method, which has, according to both tests, statistical validity, with a level of significance of 5%.
Considering now the classification problems, the average CR values for dataset Breast Cancer are shown in Table 7.9.

TABLE 7.9. AVERAGE CRS FOR BREAST CANCER.

	RDS	CBDS	EBDS	HDS
Best model	0.9762	**0.9803**	0.9762	0.9783
Ensemble	0.9689	0.9689	**0.9700**	0.9679
SVM models	0.9601	**0.9668**	0.9611	0.9653

As it can be seen, MOGA models achieve better performance than SVM models, despite the huge difference in complexity. The average number of features (#F) and neurons for the MOGA models (#N) as well as the average number of support vectors for SVMs (#S) are given in Table 7.10. We can say that the largest complexity of RBFNN MOGA models is 42, while the smallest complexity of SVMs is 4691.

TABLE 7.10. AVERAGE NUMBER OF FEATURES, NEURONS OF THE BEST MOGA MODELS, AND SUPPORT VECTORS, FOR BREAST CANCER.

Method	#F	#N	#S
RDS	8	3	159
CBDS	10	3	160
EBDS	13	3	156
HDS	6	3	159

In contrast with the results found for Bank, here the performance of the ensemble is inferior to the best model.

Analyzing the performance of the four data selection models in Table 7.9, CBDS seems again to be the method to apply. Regarding all classification problems with SVM models, Table 7.11 shows the averages CRs.

TABLE 7.11. AVERAGE CRS FOR THE CLASSIFICATION PROBLEMS.

	RDS	CBDS	EBDS	HDS
Breast Cancer	0.9600	**0.9668**	0.9611	0.9653
Parkinson	0.6588	0.6692	**0.6732**	0.6689
Satellite	0.9900	**0.9903**	0.9881	**0.9903**
Letter	0.9968	**0.9985**	0.9964	**0.9985**

The bold values denote the best performance for each data selection/problem combination. As it can be seen, for all classification problems except Parkinson, CBDS is superior to the others. For Satellite and Letter problems, HDS has the same performance as CBDS.

In the same way as in the regression cases, Table 7.12 illustrates the $C(i,j)$ and T values for the MOGA models, and Table 7.13 for all the 40 SVM models.

TABLE 7.12. *C*(*I*,*J*) /*T* FOR BREAST CANCER – BEST AND ENSEMBLE.

C(i,j)/T	RDS	CBDS	EBDS	HDS
RDS		4/14.5	3/25	3/19.5
CBDS	2/14.5		3/22.5	3/23
EBDS	4/25	4/22.5		5/25
HDS	3/19.5	5/23	4/25	

TABLE 7.13. *C*(*I*,*J*) /*T* FOR ALL SVM MODELS.

C(i,j)/T	RDS	CBDS	EBDS	HDS
RDS		20/222.5	16/399.5	20/215
CBDS	8/222.5		9/251.5	9/391
EBDS	16/339.5	23/251.5		21/292
HDS	9/215	10/391.5	9/292	

In the case of MOGA models, the indication found in Table 7.9 seems to be confirmed, although without statistical validity. For the SVM models, we can say that, with a level of significance of 5%, CBDS is better than RDS and EBDS, and HDS is better than EBDS, according to the sign test.; based on the Wilcoxon test, HDS and CBDS are better than RDS, and HDS is better than EBDS.

7.6. Conclusions

In this chapter we have compared the performance obtained by RBFNN models designed by a MOGA to that obtained by MLPs (for regression) and by SVMs (for classification). It was shown that the former obtain much better performance, despite the much smaller complexity of MOGA models. Another conclusion that can be taken is that the naïve versions of the ensemble of non-dominated MOGA models proposed here, in some cases perform better, while in other cases worse than the selected best model. In relation with the best data selection methods, we can say that the CBDS and HDS should be used for SVM and MLP models. For the RBFNN MOGA models, the same conclusion can be taken although without any statistical validity. This can be explained by the small number of experiments conducted, which was due to the high computational time, and also to the much better performance obtained by these models, compared with MLPs and SVMs, which reduces the range of differences between the data selection methods.

8. A Convex hull, sliding-window based online adaptation method

8.1. Introduction

Principally, the online adaptation process is considered in two situations in the domain of data-driven models. The first case is when only a small number of training samples is available offline and it is impossible to collect additional informative data samples reflecting the whole operating region(s) of the process to be modeled. The second case is when the behavior of the process is time-varying (i.e., its dynamics and operating regions change over time). In both cases, data-driven models need to be updated to cover new dynamics and operating regions of the underlying process.

Specifically, for Feedforward Neural Networks such as RBFNN models, online adaptation process can be considered from the structure, parameter and data points of view. In the structure aspect, the number of hidden nodes may be changed or kept constant over the online adaptation process. As pointed out earlier, the RBFNNs have two groups of parameters; 1- linear parameters 2- nonlinear parameters. In online adaptation either only the former group or both are updated online. Regarding the data, a specific RBFNN model can be adapted in several ways. Based on how much data is available/used and how to manage those data, a variety of online learning methods have been proposed.

This chapter is organized as follows: Section 8.2 gives an overview of related works in online adaptation. A new online adaptation method based on convex hull and sliding-window is introduced in Section 8.3. Experimental results are given in Section 8.4. In this section, two case studies are considered to evaluate the proposed method. The comparison between the performances of both case studies is given in Section 8.5. The proposed method is compared with others in Section 8.6. Finally, conclusions are given in Section 8.7.

8.2. A brief overview of online adaptation methods

In order to update models online, sequential learning methods, also called online learning methods, are applied. Regarding the model structure, the online learning methods are categorized into two main classes. In the first class, the structure of the model, translated into the number of hidden neurons is constant over the adaptation process and only the parameters are adjusted; In the second class, the hidden neurons are inserted or removed from the model structure using specific growing and pruning criteria, respectively.

From a parameter point of view, online learning methods can be categorized into two groups. The first group only updates the linear parameters while the nonlinear parameters are kept unchanged, while in the second group both linear and nonlinear parameters are updated.
Online learning methods can also be categorized according to the amount of data that they use [167, 168]. The first class uses the information of the new observation at each time instant. Regarding only linear parameters, in case that nonlinear parameters have been determined offline and are kept unchanged throughout online adaptation process, first order methods such as Least Means Square (LMS) [169] and Normalized Least Means Square (NLMS), and second order methods such as Recursive Least Squares (RLS) [170] and Kalman Filter and its variants [171] can be used to update only linear parameters. Regarding all parameters, recursive version of offline algorithms such as Stochastic Gradient Descent Back Propagation (SGBP) [172] as the first order method, and Recursive Least Squares (RLS) [170] and recursive Levenberg-Marquardt [173, 174] as the second order methods can be applied to update both linear and nonlinear parameters.
The second class of online learning method from data point of data uses a sliding-window of past observations to update the parameters. Moving average of LMS/NLMS search directions [175], maximum error method and Gram-Schmidt orthogonalization [176] are methods used to updated only linear parameters while any gradient descent based methods mentioned in Section 2.4.1 with a window management policy can be employed to update both linear and nonlinear parameters.
Fig 8.1 briefly illustrate the classification of online methods from the three different points of view.
In the following sections, we summarize important contributions on RBFNN online model adaptation, regarding models with a fixed structure and models with an adaptive structure, whose structure varies through time.

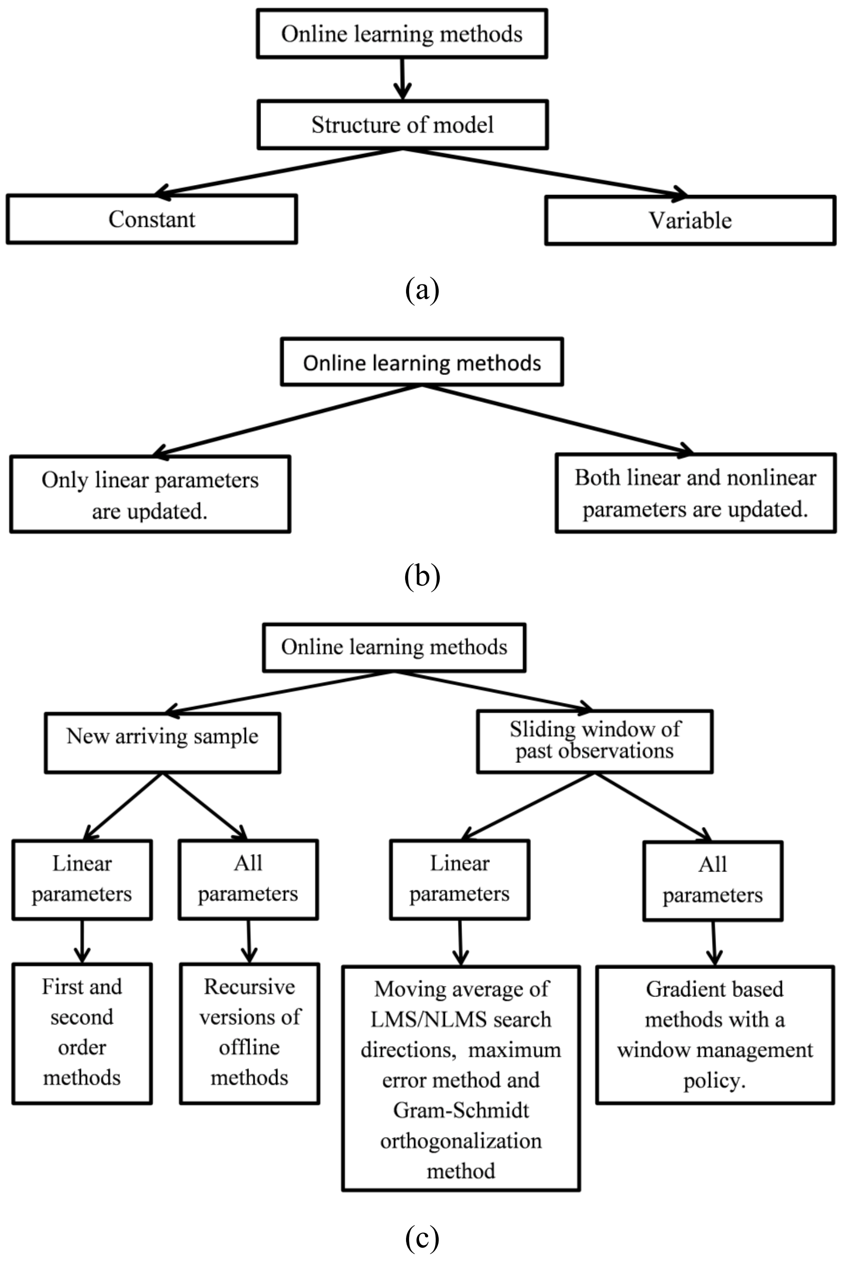

Online learning methods
Structure of model
Constant
Variable
(a)
Online learning methods
Only linear parameters are updated.
Both linear and nonlinear parameters are updated.
(b)
Online learning methods
New arriving sample
Sliding window of past observations
Linear parameters
All parameters
Linear parameters
All parameters
First and second order methods
Recursive versions of offline methods
Moving average of LMS/NLMS search directions, maximum error method and Gram-Schmidt orthogonalization method
Gradient based methods with a window management policy.
(c)

Fig. 8.1. Classification of online learning methods. (a) From the model structure point of view; (b) From the model's parameters point of view; (c) From the data point of view.

8.2.1. Online learning methods for RBFNNs with fixed structure

As mentioned earlier, from the model structure point of view, there is a group of methods which keep the structure fixed throughout the online adaptation process and only updates the model parameters. Authors in [1] presented an online adaptation method to update a fixed-structure RBFNN model, designed offline by MOGA [44, 45, 47]. Subsequently, both linear and nonlinear parameter are updated using the Levenberg-Marquardt method [32, 33], working on a sliding-window of the past observations, employing a FIFO management policy.

The same authors in [168] improved their method, using the Akaike information criterion [177] for off-line model design. In this method, a new sliding-window management policy, based on a dissimilarity measure, was proposed to overcome the problem of gradually forgetting previous mappings over the online adaption process, typically found using a FIFO policy.

Authors in [178] presented a new method to tune a fixed-structure RBFNN model.In this method, the contribution of each hidden neuron to the overall network performance is measured based on the increment of the error variance. The neuron with the smallest increment of is considered as an insignificant neuron and is replaced with the information of the new arriving sample. The linear weights are updated using the Multi-Innovative Recursive Least Square (MRLS) method over a sliding-window of p past observations, while the nonlinear parameters (centers and spreads) of the new node are adjusted using Quantum Particle Swarm Optimization method (QPSO).

An efficient sequential algorithm was proposed in [179] as well as an online version of the Extreme Learning Machine (ELM) method. In this method, the centers and spreads are arbitrarily chosen and only the weights, as linear parameters, are updated. The update is done using the proposed online version of ELM based on the new observation or a chunk of new observations over the online adaptation process.

8.2.2. Online learning methods for RBFNNs with adaptive structure

Regarding RBFNNs with adaptive structure, the first approach known as Resource Allocation Network (RAN) was proposed by Platt [180]. This method starts with a RBFNN with no hidden neurons. For each new observation at time instant k, if a new arriving sample has enough novelty, a new hidden neuron containing the information of the sample is added to the existing network, so that the updated network not only preserves the accuracy of the mapping for the previous samples which have been received so far, but also reflects a new mapping for the new sample. The novelty of the new sample is computed based on a prediction error and a distance criteria which are compared with user-defined thresholds. In the case that the new observation does not reflect a desired level of novelty, the model structure is kept unchanged and only the model parameters are updated. Both weights as linear parameters and, centers and spreads as nonlinear parameters of the existing network are updated by LMS method. This update is only done based on the new observation. An enhanced version of RAN known as RAN-EKF was subsequently proposed by

Kadirkamanathan and Niranjan [171], where the Extended Kalman Filter (EKF) as a sequential method was applied in place of the LMS method to improve the convergence rate of RAN.

For both RAN and RAN-EKF, no pruning strategy is considered. Thus, a large size network is obtained which is not suitable to be applied in real time applications, due to the high computational run time. To deal with this drawback of both RAN methods, a considerable improvement to RAN-EKF was made by Lu Yingwei *et al.* [181]. This version of RAN, which is known as M-RAN, presents a pruning strategy to remove insignificant hidden neurons with the aim of making the underlying network parsimonious and compact. In other words, the hidden neurons for which the relative contribution to the overall network output is less than a user-defined threshold are removed.

In the method proposed in [182], a new hidden neuron is added using a growing criterion based on the normalized error reduction. Beside the proposed growing strategy, a pruning strategy was also proposed to remove those hidden neurons which have had small contribution to the model output over l consecutive observations. Both linear and nonlinear parameters are updated using pseudo-inverse method based on a fixed-size sliding-window with FIFO management policy.

In [183], a criterion called "Active Firing Rate" is used to present new neurons to the hidden layers. In this method, the hidden neurons whose Active Firing is larger than the user-defined threshold AF_o (i.e., $0.05 < AF_o < 0.3$) are divided into N_{new} (i.e., $N_{new} < 10$) new neurons where N_{new} is determined based on the Active Firing Rate of the neuron. This criterion reflects the contribution of the neuron to the overall model output. Moreover, a pruning criterion based on the mutual information between hidden neurons and the output neuron is used to remove those hidden neurons which have a low connectivity strength with the output neuron. Both linear and nonlinear parameters are updated using a gradient based method based on the new observation.

The method proposed in [184] applies three criteria to add a new hidden neuron centered with the new sample. Those criteria are: the distance of the new sample to the nearest center, the output error of the new sample and the neuron's significance. Only the parameters (i.e., linear and nonlinear) of the new neuron are determined using the EKF method based on the new sample. If after update, the new hidden neuron is identified as an insignificant one, it will be removed from the model structure.

The authors in [185] proposed a method called EOS-ELM. The proposed method applies the growing and pruning criteria introduced in MRAN to adapt the structure. The weights, as linear parameters, are updated using an online version of ELM) method proposed in [179] where the linear parameters can be updated based on either the new observation or a chunk of new observations. On the other hand, the centers and spreads, as nonlinear parameters, are updated using EKF based on the new observation. Other efforts in online adaptation of RBFNNs with a flexible structure can be seen in [186-190].

8.3. A convex hull, sliding-window-based online adaptation method

In this section, we introduce a new online adaptation method based on convex hull and a sliding-window technique to update RBFNN models. This method starts with a RBFNN model which has been offline designed by MOGA based on a limited number of training, testing and validation datasets corresponding to an earlier period of time. In this method, the structure of the underlying model (i.e., hidden neurons) kept unchanged through the online adaptation process and only the parameters (both linear and nonlinear) are updated. As it can be realized from the title, the proposed method relies on two concepts; convex hull and sliding-window. The basic idea behind the proposed method is updating the model if a new arriving sample reflects a new range of input-output spaces. As we mentioned in Section 4.2.2.4, it is very important that the model is trained based on a set of data covering the whole range of input-output space in which the process is intended to be modeled. Moreover, convex hull algorithms can help us to select data samples reflecting the whole range of all existing data samples. Since at the beginning of online adaptation process there exists a model which is trained offline based on a limited number of samples and the corresponding convex hull vertices may only reflect a local range of the existing data, the initial convex hull might need to be updated with new samples changing the samples ranges. After updating the current convex hull at each time instant, the model should be trained based on the updated convex hull vertices as well as some inner points so that it can cover the whole range of the input-output space over time.

In this method, the model is updated by the LM training method operating on a fixed-size sliding-window. As mentioned in Section 8.2.2, applying sliding-window with FIFO policy leads to parameter interference phenomenon reflecting the situation in which the model forgets the mappings which have been constructed by the previous samples over time. Hence, in this method, two management policies are applied. One is a management policy proposed in [168] and the other is a convex hull based policy.

Mainly, the proposed method consists of three phases; evaluation of the arriving sample, sliding-window update and parameters update. In the following we will describe these phases.

8.3.1. Evaluation of the arriving sample

At each time instant, a new arriving sample is evaluated to see whether it leads to a new range of input-output space, or not. The new sample is compared with the current convex hull. The new sample is considered as an informative sample when it is located outside the current convex hull, meaning that a new range of input-output space must be determined, including the new point. To determine whether the new sample is located outside the current convex hull or not, a convex hull algorithm is applied on a set containing the vertices of current convex hull and the new sample. If the new sample is marked as a new vertex of the convex hull, it is definitely located outside the current convex hull; otherwise, it is considered as an inner point. Since, in practice, the input space of the underlying model can have high dimensions and standard real convex hull algorithms in high dimensions suffer from high time complexity and memory requirements (i.e., please see Section 3.4), it is not feasible to apply standard real convex hull algorithms in the online adaptation process.

To deal with this challenge, a heuristic is used to identify the location of a new sample with respect to the current convex hull. The idea behind the heuristic stems from the basic property of convex hull vertices. A point of a given set is a vertex of the corresponding convex hull if and only if there is a hyperplane passing through the point and all remaining points are located in the same side of the hyperplane. Figs. 8.2 and 8.3 illustrate a difference between a vertex of convex hull and an inner point in terms of the hyperplanes passing through them.

If a point is a vertex of the convex hull, there is an infinite number of hyperplanes passing through the point, so that all remaining points are located in the same side of each hyperplane. Hence examining all possible hyperplanes passing through the point is not possible. In our work, only the hyperplane whose direction of its normal vector is the same as that of the vector from the center of the current convex hull to the new point is formed. After forming the hyperplane, all vertices of the current convex hull are examined. If all vertices are located below the hyperplane, the new arriving point is definitely an outer point; otherwise, if the maximum distance to the hyperplane among those vertices which are located above the hyperplane is very small, it can be interpreted as follows:

- Either the new point is located outside the convex hull but very close to the convex hull
- Or the new point is an inner point which is very close to the convex hull.

If the maximum distance is large, it is very likely that the new point is located inside the convex hull. Figs. 8.4 and 8.5 illustrate how intuitively the heuristic works.

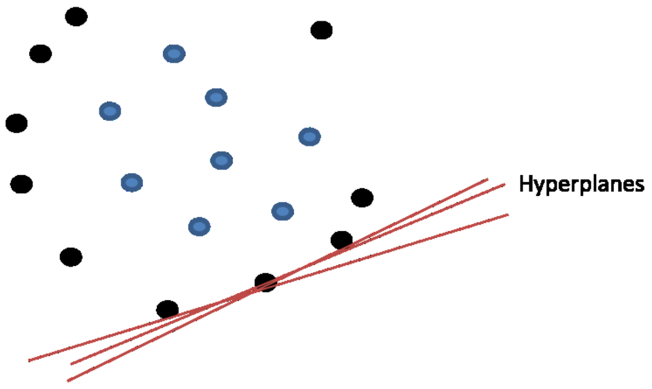


Fig. 8.2. A vertex of the convex hull. Black and blue circles are convex hull vertices and inner points, respectively.

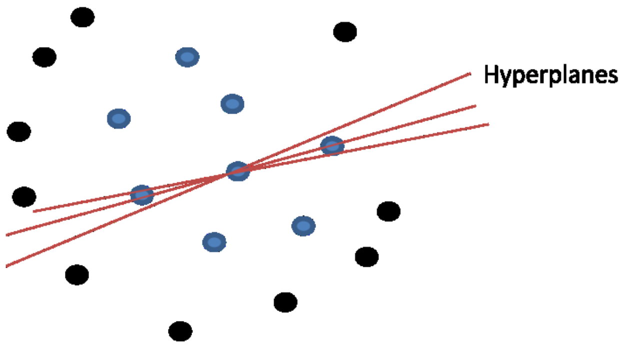


Fig. 8.3. A point is located inside the convex hull. Black and blue circles are convex hull vertices and inner points, respectively.

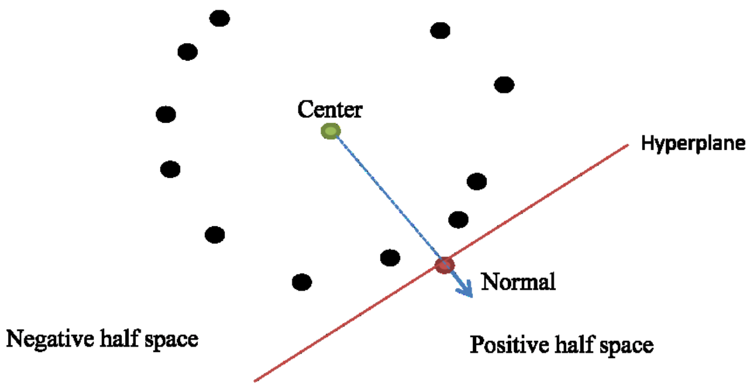


Fig. 8.4. A new point is located outside the convex hull. Black and red circles denote the convex hull vertices and the new point, respectively.

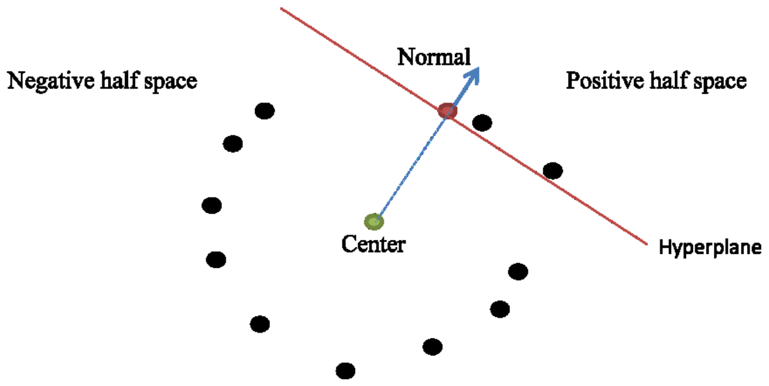


Fig. 8.5. A new point is located outside the convex hull but very close to the convex hull. Black and red circles denote the convex hull vertices and the new point, respectively.

Since there is uncertainty in classifying the new point in the case that some vertices are located above the hyperplane (i.e., the new point can be an outer or an inner point), we use a threshold for the maximum distance to the hyperplane for those vertices. If the maximum distance exceeds the threshold, the new point is marked as an inner point and rejected for inserting into the sliding-window; otherwise, the new point is examined in the next step by applying ApproxHull (i.e., please see Section 4.3) on all vertices of the current convex hull, including the new point. If the new point is identified as a vertex of the convex hull, it is accepted to be inserted into the sliding-window. In the following, we will explain how to compute a hyperplane and the distance of a point to the hyperplane.

8.3.1.1. Hyperplane computation

Let $\boldsymbol{c} = (c_1, c_2, \dots, c_d)$ be the center of the current convex hull and $\boldsymbol{p} = (p_1, p_2, \dots, p_d)$ a new arriving sample in a d-dimensional Euclidean space. The vector from $\boldsymbol{c}$ to $\boldsymbol{p}$ is defined as Eq. (8.1).

$$\overrightarrow{\boldsymbol{cp}} =< p_1 - c_1, p_2 - c_2, \dots, p_d - c_d > \tag{8.1}$$

The normal vector whose direction is the same as that of $\overrightarrow{\boldsymbol{cp}}$ can be obtained as Eq. (8.2).

$$\overrightarrow{\boldsymbol{A}} = \frac{\overrightarrow{\boldsymbol{cp}}}{|\overrightarrow{\boldsymbol{cp}}|} \tag{8.2}$$

where $\overrightarrow{\boldsymbol{A}} =< a_1, a_2, \dots, a_d >$ is the normal vector and $|\overrightarrow{\boldsymbol{cp}}|$ denotes the length of vector $\overrightarrow{\boldsymbol{cp}}$. The general form of hyperplane equation in d-dimensional Euclidean space is given as Eq. (8.3).

$$a_1x_1 + a_2x_2 + \cdots + a_dx_d = b \tag{8.3}$$

where b is the offset of hyperplane denoting the distance of hyperplane from the origin.

Since based on the heuristic, we are interested to have a hyperplane with a normal vector $\overrightarrow{\boldsymbol{A}}$ passing through point $\boldsymbol{p}$, the offset b is computed as Eq. (8.4).

$$b = a_1p_1 + a_2p_2 + \cdots + a_dp_d \tag{8.4}$$

Having the equation of hyperplane H in hand, the distance of any point $\boldsymbol{q}$ to H is computed as Eq. (8.5).

$$dist(H, \boldsymbol{q}) = a_1q_1 + a_2q_2 + \cdots + a_dq_d - b \tag{8.5}$$

$\overrightarrow{\boldsymbol{A}}$ and b, as the normal vector and the offset, are computed with time complexity $O(d)$. The distance of a point to the hyperplane is also computed with time complexity $O(d)$.

Assuming that the current convex hull contains v vertices at time instant k, the time complexity of the heuristic is $O(vd)$.

8.3.2. Sliding-windows update

In the proposed online adaptation method, two sliding-windows are considered; training sliding-window and an additional sliding-window (described later in Section 8.3.2.2) which can be updated using two proposed management policies, rather than FIFO policy: one is the policy proposed in [168], hereinafter called $F - R\, policy$ and the other is a convex hull based policy. The following describes each of these two policies.

8.3.2.1. $\boldsymbol{F - R\, policy}$

The $F - R\, policy$ was introduced in [168] and for the sake of completion, is summarized hereThe idea behind $F - R\, policy$ is updating the sliding-window with the new point; hopefully it can bring new information to the sliding-window and keep the sliding-window in a desirable level of diversity. To achieve this goal, a dissimilarity measure based on Euclidean distance is used in such a way that, in each iteration, a similarity vector rather than a similarity matrix is updated in an efficient way. In the case of the similarity matrix, in each iteration, $(N(N-2))/2$ similarities should be computed while in the case of similarity vectors only $N-1$ elements are removed from the vector and $N-1$ new elements are appended into it, where N is the size of the involving sliding-window.

Suppose $\boldsymbol{X} = [\boldsymbol{x}(1), \boldsymbol{x}(2), \cdots, \boldsymbol{x}(N)]^T$ is the input matrix and $\boldsymbol{y}$ is the corresponding output vector for a NARX model as $\hat{\boldsymbol{y}} = f(\boldsymbol{X})$ where the regressor vector $\boldsymbol{x}(k)$ is given as Eq. (8.6).

$$\boldsymbol{x}(k) = [\boldsymbol{u}(k), \boldsymbol{v}_1(k), \cdots, \boldsymbol{v}_m(k)\ ]^T \tag{8.6}$$

Vector $\boldsymbol{u}(k)$ consists of n_u lags of $\boldsymbol{y}$ and each vector $\boldsymbol{v}_i$ denoting the i^{th} exogenous variable includes nv_i lags for $i = 1,2,\cdots m$. Examples of $\boldsymbol{u}$ and $\boldsymbol{v}_i$ in the time instant k are shown in Eqs. (8.7) and (8.8).

$$\boldsymbol{u}(k) = [y(k - l_1^u), y(k - l_2^u), \cdots, y(k - l_{n_u}^u)] \tag{8.7}$$

$$\boldsymbol{v}_i(k) = [v_i(k - l_1^{v_i}), v_i(k - l_2^{v_i}), \cdots, v_i(k - l_{nv_i}^{v_i})] \tag{8.8}$$

where vectors $\boldsymbol{l}^u$ and $\boldsymbol{l}^{v_i}$ denote the order of lags for output variable $\boldsymbol{y}$ and the input variable $\boldsymbol{v}_i$, respectively. The number of elements in $\boldsymbol{x}(k)$ is d where $d = n_u + \sum_{i=1}^{m} nv_i$. In the case of Nonlinear AutoRegressive (NAR) model where only the lags of output variable $\boldsymbol{y}$ are considered, $d = n_u$. Hence $\boldsymbol{X}$ is a matrix $N \times d$. Since in our problem, we suppose that a

NAR or NARX model will compute a one-step-ahead prediction, the output vector $\boldsymbol{y}$ corresponding to the input matrix $\boldsymbol{X}$ is:

$$\boldsymbol{y} = [y(2), y(3), \cdots, y(N+1)]^T \tag{8.9}$$

The underlying sliding-window $\boldsymbol{T}$ is defined as Eq. (8.10).

$$\boldsymbol{T} = \{\boldsymbol{X}, \boldsymbol{y}\} \tag{8.10}$$

where $\boldsymbol{T}(i) = \{\boldsymbol{X}(i), \boldsymbol{y}(i)\}$ denotes the ith input-output pattern of $\boldsymbol{T}$. When a new point $\boldsymbol{p} = \{\boldsymbol{X}(k-1), \boldsymbol{y}(k)\}$ is presented to the model, two steps should be performed to update the sliding-window. The first one is whether $\boldsymbol{p}$ can be inserted into the sliding window. If so, the second one is which sample of $\boldsymbol{T}$ should be replaced with $\boldsymbol{p}$,since the size of sliding-window is assumed to be constant throughout the online adaptation process. For the first and second point, two criteria called *Include* and *Exclude* are considered, respectively. The following describes the criteria.

- ***Include criterion***

This criterion checks whether $\boldsymbol{p}$ has enough dissimilarity to all points of $\boldsymbol{T}$. To do this, the Euclidean distances between $\boldsymbol{p}$ and all points in $\boldsymbol{T}$ are considered. If all distances are greater than a user-defined threshold, point $\boldsymbol{p}$ is inserted into $\boldsymbol{T}$. Let $\delta(n, m)$ be the Euclidean distance between the nth and mth points of $\boldsymbol{X}$ (the nth and mth points will be called origin and destination points, respectively). For any two points $\boldsymbol{X}(m_1)$ and $\boldsymbol{X}(m_2)$, we say point $\boldsymbol{X}(n)$ is more similar to $\boldsymbol{X}(m_1)$ than $\boldsymbol{X}(m_2)$ if $\delta(n, m_1) < \delta(n, m_2)$. For any point $\boldsymbol{X}(n)$ in $\boldsymbol{T}$, a vector of distances between $\boldsymbol{X}(n)$ and its predecessors denoted by $\boldsymbol{\Delta}(n)$ is defined as Eq. (8.11).

$$\boldsymbol{\Delta}(n) = [\delta(1, n), \delta(2, n), \cdots, \delta(n-1, n)] \tag{8.11}$$

The vector of distances between each pair of points in $\boldsymbol{X}$ is defined as Eq. (8.12).

$$\mathbf{D} = [\boldsymbol{\Delta}(2), \boldsymbol{\Delta}(3), \cdots, \boldsymbol{\Delta}(N)\ ]^T \tag{8.12}$$

Suppose that new point $\boldsymbol{p}$ is the nth arriving point. Based on the definitions above, $\boldsymbol{p}$ is inserted into $\boldsymbol{T}$ if all distances in $\boldsymbol{\Delta}(n)$ is greater than a user-defined threshold η. Each pattern in $\boldsymbol{X}$ is organized into several components so that each component corresponds to the number of lags of a particular variable. Since the scales and dynamics may be different from variable to variable, it motivates us to consider a separate analysis of the distance between $\boldsymbol{p}$ and all points in $\boldsymbol{X}$. Based on this idea, $\boldsymbol{\Delta}(n)$can be divided into several vectors as Eqs. (8.13) and (8.14).

$$\boldsymbol{\Delta}_u(n) = [\delta_u(1,n), \delta_u(2,n), \cdots, \delta_u(n-1,n)] \tag{8.13}$$

$$\{\boldsymbol{\Delta}_{v_i}(n) = [\delta_{v_i}(1,n), \delta_{v_i}(2,n), \cdots, \delta_{v_i}(n-1,n)]\}_{i=1}^{m} \tag{8.14}$$

According to this idea, instead of considering one user-defined threshold η, a set of thresholds should be considered as Eq. (8.15).

$$\boldsymbol{\eta} = \{\eta_u, \{\eta_{v_i}\}_{i=1}^{m}\} \tag{8.15}$$

Therefore, point $\boldsymbol{p}$ is inserted into $\boldsymbol{T}$ if there is at least a distance vector in $\boldsymbol{\Delta}(n)$ so that all distances in $\boldsymbol{\Delta}(n)$ are greater than the corresponding threshold. The $Include\ criterion$ denoted by *I* is defined as Eq. (8.16).

$$I\left(\left\{\eta, \eta_u, \{\eta_{v_i}\}_{i=1}^{m}\right\}, \left\{\boldsymbol{\Delta}(n), \boldsymbol{\Delta}_u(n), \{\boldsymbol{\Delta}_{v_i}(n)\}_{i=1}^{m}\right\}\right) = (\boldsymbol{\Delta}(n) > \eta)\ or\ (\boldsymbol{\Delta}_u(n) > \eta_u)\ or \\ (\exists\ i\ \in [1,2,\cdots,m]: \boldsymbol{\Delta}_{v_i}(n) > \eta_{v_i}) \tag{8.16}$$

- ***Exclude criterion***

If *Include* criterion is *True* for the new point $\boldsymbol{p}$, to keep the size of $\boldsymbol{T}$ fixed, first one point is removed from $\boldsymbol{T}$ and then $\boldsymbol{p}$ is inserted. The main idea behind the *Exclude* criterion is to randomly remove one of two points in $\boldsymbol{T}$ which have the largest similarity (i.e., the minimum Euclidean distance) between each other. As in each iteration, there is a correspondence between $\boldsymbol{T}$ and $\boldsymbol{D}$, by updating $\boldsymbol{T}$, $\mathbf{D}$ should be updated. For each point $\boldsymbol{X}(n)$ in $\boldsymbol{X}$, there are exactly $N-1$ occurrences in $\boldsymbol{D}$ so that for $n-1$ consecutive occurrences, $\boldsymbol{X}(n)$ is a destination point whereas for $N-n$ nonconsecutive occurrences, it is an origin point, where *N* is the size of $\boldsymbol{T}$. Suppose $\boldsymbol{X}(n)$ is a point that should be removed from $\boldsymbol{T}$. To do this, their corresponding occurrences in $\boldsymbol{D}$ should be identified and then discarded from $\boldsymbol{D}$. To efficiently find the index of the corresponding occurrences of $\boldsymbol{X}(n)$ in $\boldsymbol{D}$, a sequence of functions is needed. The following introduces such functions.

Given $\boldsymbol{\Delta}(n)$ and $\boldsymbol{D}$, function $si(n)$ defined in Eq. (8.17) computes the starting index of $\boldsymbol{\Delta}(n)$ in $\boldsymbol{D}$.

$$si(n) = 1 + \sum_{i=2}^{n-1}(i-1) = \frac{n^2 - 3n + 4}{2}, \qquad n \geq 1 \tag{8.17}$$

Suppose i is an arbitrary index on $\boldsymbol{D}$. By the solutions obtained from $si(n) = i$, function $pn(i)$ defined in Eq. (8.18) computes the index of the point in $\boldsymbol{X}$ which is the destination point in the ith position of vector $\boldsymbol{D}$.

$$pn(i) = \left\lfloor \frac{3 + \sqrt{-7 + 8i}}{2} \right\rfloor, \qquad i \geq 1 \tag{8.18}$$

where $\lfloor a \rfloor$ is the largest integer smaller than a. Using Eqs. (8.17) and (8.18), the function $pm(i)$ is defined in Eq. (8.19) which computes the index of the point in $\boldsymbol{X}$ that is the origin point in the ith position of vector $\boldsymbol{D}$.

$$pm(i) = i - si\big(pn(i)\big) + 1, \qquad i \geq 1 \tag{8.19}$$

The indices in $\boldsymbol{D}$ where $\boldsymbol{X(n)}$ is a destination point are obtained by the function $dp(n,j)$ defined in Eq. (8.20).

$$dp(n,j) = si(n) + (j-1) = \frac{n^2 - 3n + 2(j+1)}{2}, \qquad 1 \leq j \leq n-1 \tag{8.20}$$

The indices in $\boldsymbol{D}$ where $\boldsymbol{X(n)}$ is an origin point are obtained by the function $op(n,j)$ defined in Eq. (8.21).

$$\begin{aligned} op(n,j) &= si(n) + 2(n-1) + n(j-n) + S(j-n) \\ &= \frac{-n^2 + n(2j+1) + 2S(j-n)}{2}, \qquad n-1 < j \leq N-1 \end{aligned} \tag{8.21}$$

where

$$S(v) = \begin{cases} 0, & v < 2 \\ \displaystyle\sum_{i=1}^{v-1} i, & v \geq 2 \end{cases}$$

By means of Eqs. (8.20) and (8.21), the index of jth occurrence of a distance in $\boldsymbol{D}$ involving point $\boldsymbol{X}(n)$ is obtained by the function $ni(n,j)$ defined in Eq. (8.22).

$$ni(n,j) = \begin{cases} dp(n,j), & 1 \leq j \leq n-1 \\ op(n,j), & n-1 < j \leq N-1 \end{cases} \tag{8.22}$$

In terms of $ni(n,j)$, the index vector of the occurrences of the distances in $\boldsymbol{D}$ involving point $\boldsymbol{X}(n)$ is as:

$$\boldsymbol{OnD} = [ni(n,1), ni(n,2), \cdots, ni(n,N-1)]^T \tag{8.23}$$

Assume the notation $< a, \boldsymbol{S} >$ denotes the index of element a in vector $\boldsymbol{S}$ and suppose also that function $\varrho(\ldots)$ randomly returns one of its arguments. Hence by means of Eqs. (8.18) and (8.19), the *Exclude* criterion denoted by O is defined as Eq. (8.24).

$$O(\boldsymbol{D}) = \varrho(pm(< \min(\boldsymbol{D}), \boldsymbol{D} >), pn(< \min(\boldsymbol{D}), \boldsymbol{D} >)) \tag{8.24}$$

After the index of the point that should be removed from $\boldsymbol{T}$ is determined by the *Exclude* criterion, the point is removed from $\boldsymbol{T}$ and then all its corresponding indices in $\boldsymbol{D}$ given by (8.23) are discarded from $\boldsymbol{D}$. Afterwards, the new point is inserted into $\boldsymbol{T}$ and then its corresponding $\boldsymbol{\Delta}(n)$ is appended into $\boldsymbol{D}$.

8.3.2.2. A proposed convex hull based policy

In the proposed online adaptation method, both the training and the additional sliding-window are updated based on $F - R\ policy$ as well as a convex hull based policy. If the new arriving sample is accepted as an outer point with respect to the current convex hull, the current convex hull is updated considering the new sample as a new vertex of the convex hull. In this step, if some vertices of the current convex hull are marked as inner points by ApproxHull, they are replaced with points from the additional sliding-window. Since the training sliding-window should contain the vertices of convex hull as well as some inner points, these points are selected from the additional set in such a way that those selected are located inside the convex hull and are dissimilar enough from the convex hull vertices. To do this, r points which have the largest minimum distance to all convex hull vertices are selected from the additional sliding-window where r denotes the number of inner points.

To compute the largest minimum distances to the convex hull vertices, a distance matrix denoted by $\boldsymbol{DIS}$ of size $s \times v$ is employed, where s and v refer to the size of the additional sliding-window and the number of convex hull vertices, respectively. Finally, the selected points and inner points are swapped between the training and the additional sliding-window. Throughout this process, $\boldsymbol{DIS}$ is updated by removing r rows corresponding to the r selected points from the additional sliding-window and appending r new rows into $\boldsymbol{DIS}$ where each new row corresponds to a distance vector including the distances between an inner point to all convex hull vertices. Besides $\boldsymbol{DIS}$, two other vectors are updated, $\boldsymbol{D}_{tr}$ and $\boldsymbol{D}_{add}$ denoting the distance vector used in the $F - R\ policy$ for managing the training and the additional sliding-window, respectively. Afterwards, the new sample is inserted into the training-sliding window by $F - R\, policy$ but with the difference that the $Include$ criterion of $F - R\, policy$ is not needed to be checked (i.e., the $F - R\, policy$ is forced to insert the new point into the training sliding-window) since the new point has been accepted as an outer point with the current convex hull. If the new arriving sample is rejected from the convex-hull approach, it is tried to be inserted into the additional sliding-window using the $F - R\, policy$.

8.3.3. Parameters update

The idea behind the procedure of parameters update in this work is the same as that mentioned in [168]. As explained before, we assume that the change of dynamics of most processes is gradual over a period of time. Hence, the underlying model does not need to be updated whenever a new sample arrives and is inserted into the training sliding-window. Additionally, frequently updating parameters over a period of time not only imposes an extra computational cost but also may cause overfitting. In order to avoid unnecessary parameter updates, two standard termination criteria (8.25) and (8.26) of the Levenberg-Marquardt method are evaluated at time instant k, when a new sample is accepted and inserted into the training sliding-window.

$$\Phi_{ku} - \Phi_k < \theta_k \tag{8.25}$$

$$\|\boldsymbol{g}_k\| \leq \sqrt[3]{\tau_f}(1 + |\Phi_k|) \tag{8.26}$$

$$\theta_k = \tau_f(1 + \Phi_k) \tag{8.27}$$

where Φ_{ku} and Φ_k denote the value of the cost function obtained based on the current parameters update and the previous parameters in time instant k, respectively. $\boldsymbol{g}_k$ is the gradient vector of the cost function and τ_f as the resolution parameter denoting a measure of the desired correct number of digits in the cost function. $\|.\|$ and $|.|$ denote the 2-norm and absolute operators.

When both criteria (8.25) and (8.26) are met, the model parameters are updated. The LM method starts with the parameters found in the last update. In order to prevent overfitting, the early-stopping method can be applied in the learning process using the additional sliding-window as the test set.

8.3.4. Analysis of the proposed method

In this section, we address the analysis of time complexity of the new sample evaluation and the sliding-window update. In this analysis, we consider the worst case scenario in terms of run time, which is where a new arriving sample is determined as an outer point, inserted into the training sliding-window, and additionally some inner points are identified due to the updating the current convex hull by the new sample. In the following, we detail the analysis of each phase.

As mentioned in Section 8.3.1, a new sample is accepted to be inserted into the training-sliding window, if it meets two conditions. Firstly, the maximum distance of those vertices located the positive half space to the corresponding hyperplane is less than or equal to a user-defined threshold and secondly, ApproxHull marks the new sample as a new convex hull vertex. As stated in Section 8.3.1.1, the time complexity for computing the maximum distance is $O(vd)$. According to Section 4.5, the time complexity of ApproxHull is $O(n^2d^3v^3 + i^3p^3)$ where n, d, v, i and p denote the number of total data samples, dimension, the number of convex hull vertices, the number of iterations and the population size, respectively. In this case, the number of samples on which the ApproxHull is applied, is equal to $v + 1$. Therefore, the time complexity of ApproxHull in this situation takes $O(v^2d^3u^3 + i^3p^3)$ where u denotes the number of vertices of updated convex hull.

The parameters update phase consists of 8 steps. As stated before, for both training and additional sliding-windows, the *F_R policy* is applied. Per the *F_R policy*, a vector of distances between each two points of the sliding-window is formed and it is updated whenever the sliding-window is updated. The time complexity of inserting a sample into the sliding-window is $O(Nd)$ due to computing the distances between the new sample and its predecessors where N denotes the sliding-window size. Removing a sample from the sliding window takes $O(N)$ due to the computation of the indices of the distance vector, where the sample is a either destination or origin point. Here we suppose that $\boldsymbol{D}_{tr}$ and $\boldsymbol{D}_{add}$ denote the distance vector of the training and the additional sliding-window, respectively. In the proposed method, matrix $\boldsymbol{DIS}$ with size $s \times v$ of distances between each point of the additional sliding-window and the vertices of current convex hull is formed, where s is the size of additional sliding-window.

Step 1 involves computing the distances between the new arriving sample and all points of the additional sliding-window and appending the distances as a new column into $\boldsymbol{DIS}$. The time complexity of Step 1 is $O(sd)$.

In Step 2, the vertices marked as inner points during the convex hull update are removed from the training-sliding window. Step 2 takes $O(rm)$ operations due to updating $\boldsymbol{D}_{tr}$ where r and m denote the number of inner points and the training-sliding-window size, respectively.

In Step 3, matrix $\boldsymbol{DIS}$ is updated by removing the corresponding columns of inner points. Step 3 takes $O(r)$ operations.

In Step 4, r points which have the largest minimum distance to all vertices of the updated convex hull are selected from the additional sliding-window. Based on the matrix $\boldsymbol{DIS}$, computing the minimum distance for each point of the additional sliding-window takes $O(su)$ operations. Selecting r samples from the additional sliding-window which have the largest minimum distance needs sorting these distances in descending order, and choosing the first r corresponding points. Hence, sorting takes $O(s \log s)$ operations. In practice u is larger than $\log s$. Therefore, the maximum time complexity of Step 4 is $O(su)$.

Step 5 involves removing the selected points from the additional sliding-window. This step leads to update both $\boldsymbol{DIS}$ and $\boldsymbol{D}_{add}$. The time complexity for updating $\boldsymbol{D}_{add}$ is $O(rs)$ while updating $\boldsymbol{DIS}$ takes $O(r)$ operations. Hence the time complexity of Step 5 is $O(rs)$.

Step 6 corresponds to adding r selected points from the additional sliding-window into the training sliding-window. This step leads to update $\boldsymbol{D}_{tr}$ which takes $O(rmd)$ operations.

Step 7 corresponds to add the inner points into the additional sliding-window which leads to update $\boldsymbol{DIS}$ and $\boldsymbol{D}_{add}$. Updating $\boldsymbol{DIS}$ due to adding rows takes $O(rud)$ operations. The time complexity of updating $\boldsymbol{D}_{add}$ is $O(rsd)$. Therefore, the time complexity of Step 7 is $O(rud + rsd)$.

Finally, In Step 8, the new arriving sample is added into the training sliding-window and $\boldsymbol{D}_{tr}$ is updated. In this step, if the point which has been replaced with the new sample using the *F-R policy* is a vertex of the convex hull, it will be removed from the vertices of convex hull and $\boldsymbol{DIS}$ will also be updated. The time complexity of Step 8 is therefore $O(m + md) = O(md)$.

The total time complexity of the proposed method, in the worst case scenario at time k is equal to $O(v^2d^3u^3 + i^3p^3 + sd + rm + r + su + rs + rmd + rud + rsd + md) = O(v^2d^3u^3 + i^3p^3 + su + rmd + rud + rsd) = O(v^2d^3u^3 + i^3p^3 + su + rmd)$.

Concisely, the sliding-window update algorithm is presented in Algorithm 8.1.

Algorithm 8.1: Sliding-windows update

Inputs: $\boldsymbol{T}$ as the training sliding-window, $\boldsymbol{D}_{tr}$ as the distance vector obtained from $\boldsymbol{T}$ using the $F - R\ policy$, $\boldsymbol{A}$ as the additional sliding-window, $\boldsymbol{D}_{add}$ as the distance vector obtained from $\boldsymbol{A}$ using the $F - R\ policy$, $\boldsymbol{V}$ as the vertices of current convex hull where $\boldsymbol{V} \subset \boldsymbol{T}$, $\boldsymbol{DIS}$ as a matrix of distances between each point of $\boldsymbol{A}$ to all points of $\boldsymbol{V}$, β as a user-defined threshold, $\boldsymbol{p} = (\boldsymbol{x}^k, y^k)$ as the new arriving sample at time instant k where x^k and y^k denote the input and output pattern.

```
1:  Let $\boldsymbol{c}$ be the center of the current convex hull.
2:  Let $H$ be the hyperplane passing by the new arriving sample so that the direction of its normal
    vector is the same as that of the vector from $\boldsymbol{c}$ to $\boldsymbol{p}$.
3:  Let $m_d$ be the maximum positive distance of points in $\boldsymbol{V}$ to $H$.
4:  Let $flag = \boldsymbol{True}$
5:  If ($m_d \leq \beta$ ) then
6:      Let $\boldsymbol{S} = \boldsymbol{V} \cup \{\boldsymbol{p}\}$
7:      Let $\boldsymbol{U}$ be the vertices of convex hull obtained by Apply ApproxHull on $\boldsymbol{S}$.
8:      If ($m_d = 0$ and $\boldsymbol{p}$ not in $\boldsymbol{U}$) then
9:          Let $\boldsymbol{U} = \boldsymbol{U} \cup \{\boldsymbol{p}\}$
10:     If ($\boldsymbol{p}$ in $\boldsymbol{U}$) then
11:         Let $\boldsymbol{I} = \boldsymbol{V} - \boldsymbol{U}$ be the set of inner points
12:         Let $\boldsymbol{V} = \boldsymbol{U}$
13:         Add the corresponding column of $\boldsymbol{p}$ into $\boldsymbol{DIS}$.
14:         If ($\boldsymbol{I} \neq \emptyset$) then
15:             Remove the inner points from $\boldsymbol{T}$ and update $\boldsymbol{D}_{tr}$ using the $F - R\ policy$.
16:             Remove the corresponding columns of inner points from $\boldsymbol{DIS}$.
17:             Let $r = |\boldsymbol{I}|$
18:             Let $\boldsymbol{W}$ contains $r$ points of $\boldsymbol{A}$ which have the largest minimum distance
                to all vertices of $\boldsymbol{V}$.
19:             Let $\boldsymbol{A} = \boldsymbol{A} \backslash \boldsymbol{W}$ and removing the corresponding elements of the points in
                $\boldsymbol{W}$ from $\boldsymbol{D}_{add}$ using the $F - R\ policy$.
20:             Remove the corresponding rows of points in $\boldsymbol{W}$ from $\boldsymbol{DIS}$.
21:             Let $\boldsymbol{T} = \boldsymbol{T} \cup \boldsymbol{W}$ and update $\boldsymbol{D}_{tr}$ using the $F - R\ policy$.
22:             Let $\boldsymbol{A} = \boldsymbol{A} \cup \boldsymbol{I}$ and update $\boldsymbol{D}_{add}$ using the $F - R\ policy$.
```

```
23:                     Add the corresponding row of each point in $\boldsymbol{I}$ into $\boldsymbol{DIS}$.
24:                 Add $\boldsymbol{p}$ into $\boldsymbol{T}$ and update $\boldsymbol{D}_{tr}$ using $F - R\ policy$.
25:                 Let $\boldsymbol{q}$ be the point which has been replaced with $\boldsymbol{p}$ using $F - R\ policy$.
26:                 **If** ($\boldsymbol{q}$ in $\boldsymbol{V}$) **then**
27:                     Let $\boldsymbol{V} = \boldsymbol{V} \backslash \boldsymbol{q}$
28:                     Remove the corresponding column of $\boldsymbol{q}$ from $\boldsymbol{DIS}$.
29:             **else**
30:                 Let $flag = \boldsymbol{False}$
31:     **else**
32:             Let $flag = \boldsymbol{False}$
33:     **If** (**not** $flag$) **then**
34:             Add $\boldsymbol{p}$ into $\boldsymbol{A}$ and update $\boldsymbol{D}_{add}$ using the $F - R\ policy$.
35:             Let $\boldsymbol{z}$ be the point which has been replaced with $\boldsymbol{p}$ using $F - R\ policy$.
36:             Remove the corresponding row of $\boldsymbol{z}$ from $\boldsymbol{DIS}$.
37:             Add the corresponding row of $\boldsymbol{p}$ into $\boldsymbol{DIS}$.
**Outputs:** $\boldsymbol{T}$, $\boldsymbol{A}$, $\boldsymbol{V}$, $\boldsymbol{DIS}$, $\boldsymbol{D}_{tr}$ and $\boldsymbol{D}_{add}$.
```

8.4. Experimental results

To evaluate the performance of the proposed online adaptation method, two case studies were considered. In both case studies, a time series NAR model was chosen to compute the one-step ahead value of Outside Air Temperature. The first case study explained in Section 8.4.1 uses the data collected at the University of Algarve, Portugal while the second one discussed in Section 8.4.2 is linked to the data collected at the University of Almeria, Spain. For both case studies, the corresponding models were designed offline using one run of MOGA. The design objectives were the RMSE obtained in the training and test data sets as well as the summation of RMSE over the prediction horizon with 48 steps obtained in the simulation data set, and the model complexity. On the objectives, no restriction was considered (please refer to Section 2.7.2.1). Regarding MOGA's parameters, both the maximum number of generation and the population size were set to 100. The early-stopping method was applied with a maximum of 100 iterations. After one complete run of MOGA, one model was selected from the non-dominated set for the case study. The following explains each case study along with the analysis of the corresponding evaluation results.

8.4.1. Case Study 1: OAT model for the University of Algarve

The data provided by the University of Algarve has been collected over the years 2015 and 2016. In the design process, the data in the range 12-Nov2015 to 28-Nov-2015 (i.e., approximately 17 days) with a sample rate of 5 minutes was used to create the training, testing and validation sets with 2538, 846 and 846 points, respectively. Data in the range 29-Nov-2015 to 30-Nov-2015 (i.e, 2 days) was used as the simulation set, to evaluate the offline models over 48-steps-ahead prediction (i.e., a 4 hours ahead prediction). ApproxHull was applied on the whole data which resulted in 1356 convex hull points, that were included in the training set. In this process, the range of features considered by MOGA comprised the first 48 lags (i.e., corresponding to the first 4 previous hours), together 25 lags centered on the sample corresponding to one day before (1 hour before and 1 hour after). Therefore, 73 features were considered by MOGA, and the formal description of the selected OAT model is given in Eq. (8.28).

$$\begin{aligned}\widehat{T_o}(k+1) = f(&T_o(k), T_o(k-1), T_o(k-2), T_o(k-8), T_o(k-10), T_o(k-27), T_o(k\\ &-32), T_o(k-42), T_o(k-44), T_o(k-277), T_o(k-280), T_o(k\\ &-282), T_o(k-284), T_o(k-298)\ )\end{aligned} \tag{8.28}$$

According to Eq. (8.28), the selected model has 14 inputs which are all lags of OAT. The corresponding RBFNN model has 3 hidden neurons and one output neuron. To simulate the online adaptation process, 17 periods were considered. The periods are given in Table 8.1. The simulation samples of each period were normalized in the range $[-1,1]$. Since the model has only used 14 lags out of 73 lags in the design process, the initial convex hull of the model should be obtained from the reduced version of the whole data which was supplied to MOGA. ApproxHull was hence applied to the reduced dataset with 15 dimensions (inputs and target pattern), which resulted in 875 convex hull points that were included in the initial training-sliding window.

For all experiments, the online adaptation process starts with the parameters' values obtained in the offline training in the design process. The model is subsequently updated over the periods based on the order stated in Table 8.1. In this procedure, at the beginning of each period, the online adaptation process continues with the last update of the model over the previous period. After each model update within a period, the model is evaluated based on its 48-steps-ahead prediction (i.e., 4 hours ahead prediction) over the period.

TABLE 8.1. PERIODS OVER THE YEARS 2015 AND 2016 IN CASE 1.

Period Name	Range
01-Dec-2015	01-Dec-2015 00:00:00 to 13-Dec-2015 06:20:00
16-Dec-2015	16-Dec-2015 09:29:00 to 20-Dec-2015 09:39:00
20-Dec-2015	20-Dec-2015 10:22:00 to 29-Dec-2015 05:42:00
01-Jan-2016	01-Jan-2016 00:00:00 to 15-Jan-2016 09:45:00
21-Jan-2016	21-Jan-2016 19:07:00 to 31-Jan-2016 23:57:00
01-Feb-2016	01-Feb-2016 00:00:00 to 29-Feb-2016 23:55:00
01-Mar-2016	01-Mar-2016 00:00:00 to 31-Mar-2016 23:55:00
01-Apr-2016	01-Apr-2016 00:00:00 to 30-Apr-2016 23:55:00
01-May-2016	01-May-2016 00:00:00 to 11-May-2016 08:20:00
11-May-2016	11-May-2016 09:26:00 to 31-May 2016 23:56:00
01-Jun-2016	01-Jun-2016 00:00:00 to 30-Jun-2016 23:55:00
01-Jul-2016	01-Jul-2016 00:00:00 to 06-Jul-2016 02:45:00
04-Aug-2016	04-Aug-2016 22:26:00 to 31-Aug-2016 23:56:00
01-Sep-2016	01-Sep-2016 00:00:00 to 30-Sep-2016 23:55:00
01-Oct-2016	01-Oct-2016 00:00:00 to 31-Oct-2016 23:55:00
01-Nov-2016	01-Nov-2016 00:00:00 to 19-Nov-2016 17:05:00
19-Nov-2016	19-Nov-2016 18:01:00 to 26-Nov-2016 19:26:00

In this study, 6 different experiments were carried out. For all experiments, the training and the additional sliding-window size were set to 2538 and 846, respectively. The maximum number of iterations of the Levenberg-Marquardt method was set to 100 for all experiments. Moreover, two user-defined thresholds β (the hyperplane distance threshold which is used in the sliding-window management policy) and $\boldsymbol{\tau_f}$ (the desired resolution in the LM termination criteria) were considered as parameters. η (the dissimilarity threshold which is used in the *F-R policy)* had a fixed value of 0.005.

The experiments' specification is given in Table 8.2. As we can see in Table 8.2, two groups of experiments were carried out. For the first group $\boldsymbol{\tau_f}$ was set to 0.001 while for the second one it was set to 0.0001. The aim was to see if an increase of the number of iterations in each update process, could result in a better performance. For both groups, β was set to the constant value 0.005 while η varies from 0.0 to 0.5.

TABLE 8.2. EXPERIMENT'S SPECIFICATION IN CASE 1.

		$\boldsymbol{\tau_f}$	$\boldsymbol{\beta}$	$\boldsymbol{\eta}$
First group of experiments	Exp.1	0.001	0.0	0.005
	Exp.2	0.001	0.1	0.005
	Exp.3	0.001	0.5	0.005
Second group of experiments	Exp.4	0.0001	0.0	0.005
	Exp.5	0.0001	0.1	0.005
	Exp.6	0.0001	0.5	0.005

In order to compare the experiments, the criteria stated in Table 8.3 were considered. Please note that the number of samples in each period is obtained as 12*24=288*number of days.

TABLE 8.3. LIST OF CRITERIA USED TO COMPARE THE EXPERIMENTS IN CASE 1.

$\boldsymbol{n_T}$	Number of samples which have been inserted into the training sliding-window over all periods.
$\boldsymbol{n_A}$	Number of samples which have been inserted into the additional sliding-window over all periods.
$\boldsymbol{n_R}$	Number of samples which have been rejected from inserting into both training and additional sliding-window over all periods.
$\boldsymbol{n_U}$	Number of parameter updates over all periods.
$\boldsymbol{n_I}$	Average number of iterations of training process per each update over all periods.
$\boldsymbol{n_{CH}}$	Number of convex hull points at end of the last period.
$\boldsymbol{\rho_1^i}$	Scaled one-step-ahead $RMSE$ associated with the initial model.
$\boldsymbol{\rho_1^u}$	Scaled one-step-ahead $RMSE$ associated with the updated model at end of the period.
$\boldsymbol{\rho_{48}^i}$	Scaled 48-steps-ahead $RMSE$ associated with the initial model.
$\boldsymbol{\rho_{48}^u}$	Scaled 48-steps-ahead $RMSE$ associated with the updated model at the end of the period.
$\boldsymbol{S^i}$	Summation of scaled $RMSE$s over the 48 steps of prediction associated with the initial model.
$\boldsymbol{S^u}$	Summation of scaled $RMSE$s over the 48 steps of prediction associated with the updated model at end of the period.

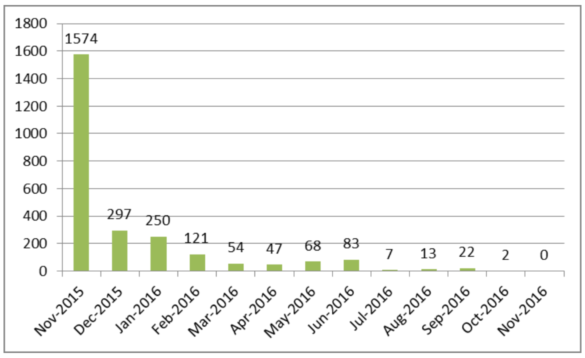
The statistical results obtained from the three groups of experiments are given in Table 8.4.

TABLE 8.4. STATISTICAL RESULTS OF THE EXPERIMENTS IN CASE 1.

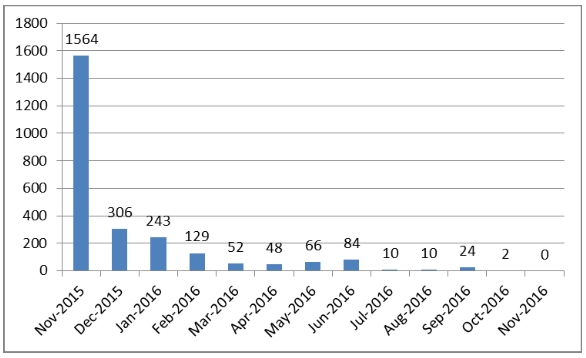
		n_T	n_A	n_R	n_U	n_I	n_{CH}
First group of experiments	Exp.1	456	86422	73	63	2.16	253
	Exp.2	1232	85645	74	142	2.11	243
	Exp.3	3304	83574	73	289	2	162
Second group of experiments	Exp.4	464	86414	73	241	2.23	256
	Exp.5	1212	85666	73	606	2.14	225
	Exp.6	3282	83597	72	1516	2.04	158

According to this Table for each group of experiments, increasing β causes an increase in n_T due to the fact that the new arriving samples have more chance to be inserted into the training sliding-window. This, in turn, causes an increase in n_U due to an increase of training sliding-window updates. Moreover, we can see that in all experiments, n_U is much smaller than n_T. This result reveals the fact that, the proposed method can prevent unnecessary parameter updates whenever the training sliding-window is updated due to the insertion of the new arriving sample.

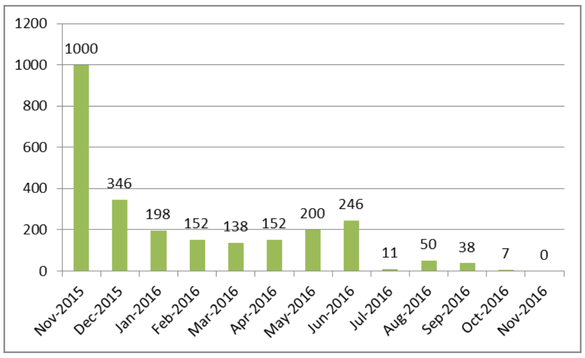
Fig. 8.6 shows the number of samples of each period in the last training sliding window at the end of the online adaptation process for both groups of experiments. As it can be seen in Fig. 8.6, each pair of experiments for which the same β has been used, (Exp.1, Exp.4), (Exp.2, Exp.5) and (Exp.3, Exp.6), the pattern of training sliding window update is the same resulting in somehow the same sliding window at the end of online adaptation process. The presence of small variations between two experiments in each pair stems from the stochastic behavior of ApproxHull. Furthermore, as it can be observed in Fig. 8.6, by increasing β, the update rate of the initial training sliding window containing samples of Nov-2015 is raising where gradually, the samples of Nov-2015 are being replaced with the new arriving samples of the other periods.



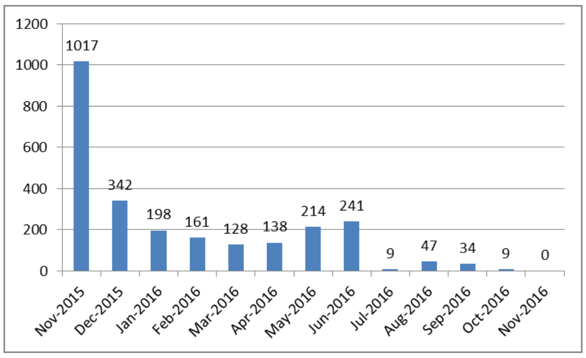
(a)



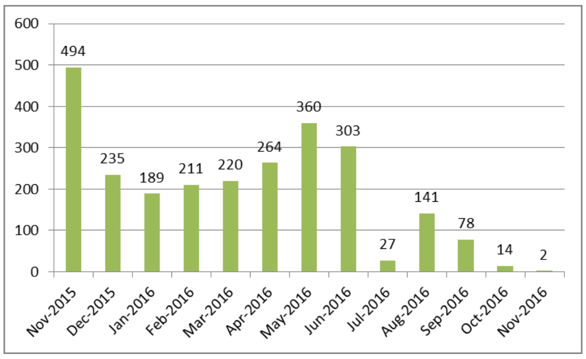
(b)



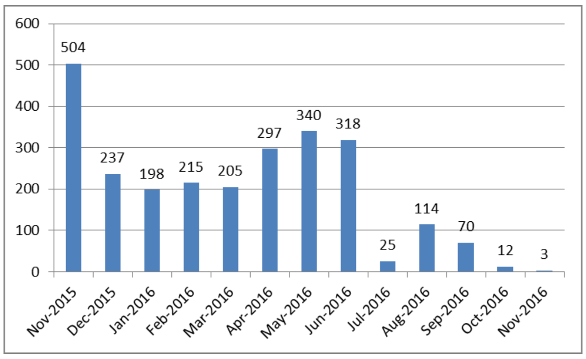
(c)



(d)



(e)



(f)

Fig.8.6. Number of samples of each period in the last training sliding window. (a), (c) and (e) correspond to the Exps.1-3 of the first group of experiments of the case study 1, respectively. (b), (d) and (f) denote Exps.4-6 of the second group of experiments of case study 1, respectively.

For all experiments, the model has been evaluated in terms of the RMSE over each period in two different situations: In the former, the initial model which was trained offline has been evaluated over each period, while in the second one, in each period, after each update at time instant k, the updated model has been evaluated over the corresponding period. The evaluation results of three groups of experiments are given in Tables 8.5 to 8.7. In these tables, the bold values indicate the best results over each period.

TABLE 8.5. ONE-STEP-AHEAD PREDICTION IN CASE 1.

		$\boldsymbol{\rho_1^u}$					
		First group of experiments			Second group of experiments		
	$\boldsymbol{\rho_1^i}$	Exp.1	Exp.2	Exp.3	Exp.4	Exp.5	Exp.6
01-Dec-2015	0.008	0.008	0.008	0.008	0.008	0.008	0.008
16-Dec-2015	0.007	0.007	0.007	0.007	0.007	0.007	0.007
20-Dec-2015	0.007	0.007	0.007	0.007	0.007	0.007	0.007
01-Jan-2016	0.007	0.007	0.007	0.007	0.007	0.007	0.007
21-Jan-2016	0.008	0.008	0.008	0.008	0.008	0.008	0.008
01-Feb-2016	0.010	0.009	0.008	0.009	0.008	0.008	0.008
01-Mar-2016	0.010	0.010	0.010	0.010	0.010	0.010	0.010
01-Apr-2016	0.010	0.010	0.010	0.010	0.010	0.010	0.010
01-May-2016	0.011	0.011	0.011	0.011	0.010	0.010	0.010
11-May-2016	0.014	0.012	0.012	0.012	0.011	0.011	0.011
01-Jun-2016	0.050	0.019	0.015	0.015	0.013	**0.012**	0.013
01-Jul-2016	0.041	0.020	0.015	0.014	0.013	**0.012**	0.013
04-Aug-2016	0.073	0.022	0.015	0.014	**0.012**	**0.012**	**0.012**
01-Sep-2016	0.069	0.019	0.013	0.013	0.012	**0.011**	**0.011**
01-Oct-2016	0.011	0.009	0.009	0.008	0.008	0.008	0.008
01-Nov-2016	0.009	0.009	0.009	0.009	0.009	0.009	0.009
19-Nov-2016	0.009	0.008	0.009	0.009	0.008	0.008	0.008

As it can be inferred from Table 8.5, the performance of the updated models in terms of the one-step-ahead prediction (i.e., 5 minutes ahead prediction) for all experiments over the periods Dec-2015 to May-2016 and Oct-2016 to Nov-2016 is, to some extent, similar to the initial model (i.e., the offline model). In contrast, the updated models for all experiments outperform significantly the initial model in the periods Jun-216 to Sep-2016. The updated models for the second group of experiments have a slightly better performance than their correspondents in the first group.

TABLE 8.6. 48-STEPS-AHEAD PREDICTION IN CASE 1.

		$\boldsymbol{\rho^{u}_{48}}$					
		First group of experiments			Second group of experiments		
	$\boldsymbol{\rho^{i}_{48}}$	Exp.1	Exp.2	Exp.3	Exp.4	Exp.5	Exp.6
01-Dec-2015	0.082	0.082	0.086	**0.079**	0.082	0.081	0.080
16-Dec-2015	0.102	0.102	0.104	**0.099**	0.102	**0.099**	0.101
20-Dec-2015	0.087	0.087	0.090	**0.085**	0.087	**0.085**	0.087
01-Jan-2016	0.129	0.123	0.124	0.131	**0.120**	**0.120**	0.127
21-Jan-2016	0.076	0.081	0.080	**0.071**	0.078	0.076	0.076
01-Feb-2016	0.104	0.131	0.130	0.108	0.091	**0.087**	0.096
01-Mar-2016	0.080	0.100	0.099	0.085	0.087	**0.077**	0.092
01-Apr-2016	0.094	0.123	0.099	0.103	0.092	**0.088**	0.096
01-May-2016	**0.104**	0.144	0.123	0.121	0.111	0.114	0.119
11-May-2016	**0.108**	0.139	0.128	0.129	0.111	0.110	0.116
01-Jun-2016	0.160	0.238	0.169	0.191	**0.137**	**0.137**	0.178
01-Jul-2016	0.157	0.228	0.175	0.198	**0.155**	0.162	0.214
04-Aug-2016	0.184	0.195	0.149	0.154	**0.112**	0.125	0.127
01-Sep-2016	0.159	0.206	0.138	0.156	**0.121**	0.132	0.129
01-Oct-2016	0.098	0.149	0.133	0.109	0.091	**0.090**	0.095
01-Nov-2016	0.088	0.209	0.320	0.115	**0.079**	0.086	0.093
19-Nov-2016	0.123	0.281	0.410	0.142	**0.105**	0.112	0.127

As it can be concluded from Table 8.6, in terms of the 48-steps-ahead prediction, the performance of the updated model for all experiments over the periods 01-Dec-2015, 16-Dec-2015, 20-Dec-2015, 01-Jan-2016 and 21-Jan-2016 is somehow the same as that of the initial model.

The updated model in Exps.4 and 5 is slightly superior to the initial model. Regarding the period 01-Feb-2016, the updated model in the second group of experiments is superior to the initial model while that in the first group has worse performance in comparison with the initial model. With respect to the period 01-Mar-2016, only the updated model in Exp.5 has better performance than the initial model whereas that in Exp.1 has the worst performance. Regarding the period 01-Apr-2016, the updated model in both Exp.4 and Exp.5 performs better than the initial model. With respect to the periods 01-Jun-2016 to 19-Nov-2016, for the most cases, the updated model in both Exp.4 and Exp.5 is significantly superior to the initial model while the updated model in Exp.1 has the worst performance.

Over the periods 01-May-2016 and 11-May-2016, the initial model has slightly better performance in comparison with the others.

TABLE 8.7. SUMMATION OVER PH IN CASE 1.

		S^u					
		First group of experiments			Second group of experiments		
	S^i	Exp.1	Exp.2	Exp.3	Exp.4	Exp.5	Exp.6
01-Dec-2015	2.703	2.703	2.785	**2.621**	2.703	2.668	2.623
16-Dec-2015	3.239	3.239	3.250	3.157	3.239	**3.143**	3.173
20-Dec-2015	2.744	2.744	2.824	2.693	2.744	2.671	**2.646**
01-Jan-2016	3.824	3.617	3.694	3.894	**3.542**	3.571	3.745
21-Jan-2016	2.426	2.528	2.528	**2.342**	2.464	2.420	2.436
01-Feb-2016	3.699	3.980	3.851	3.456	2.800	**2.703**	3.022
01-Mar-2016	2.887	3.250	3.174	2.858	2.853	**2.632**	3.008
01-Apr-2016	3.111	3.834	3.306	3.330	3.008	**2.942**	3.132
01-May-2016	3.561	4.373	3.973	3.906	3.583	**3.538**	3.687
11-May-2016	3.639	4.198	3.892	3.818	3.504	**3.383**	3.465
01-Jun-2016	6.480	6.333	4.917	5.373	**4.414**	4.519	4.915
01-Jul-2016	6.361	6.778	5.172	5.617	5.096	**4.993**	5.492
04-Aug-2016	8.021	6.129	4.599	4.664	**3.813**	4.148	3.834
01-Sep-2016	6.795	6.196	4.317	4.675	3.934	4.256	**3.898**
01-Oct-2016	3.312	4.142	3.787	3.311	2.880	**2.879**	2.937
01-Nov-2016	2.911	6.040	8.919	3.767	**2.667**	2.892	3.002
19-Nov-2016	3.854	9.682	14.027	4.680	**3.282**	3.609	3.867

As it can be seen in Table 8.7, in terms of the summation of RMSE over the prediction horizon of 48 steps, the updated models in the second group of experiments significantly outperform the initial model over the periods 01-Jun-2016 to 01-Oct-2016; over the other periods, no considerable difference can be seen between the performance of the updated model in the second group of experiments and that of the initial model. To conclude, we can say that the updated model in the second group of experiment has better performance than that of the first one and is superior to the initial model. In relation with the parameter Beta, the value of 0.1 seems to be the best one.

In order to graphically compare the performance of the updated model throughout every period, with the initial model, the corresponding updated model of Exp.5 was selected as an alternative for the initial model since according to the Tables 8.5 to 8.7, the updated model in Exp.5 is superior to the others. Figs. 8.7 to 8.23 illustrate the real values of OAT (blue line), the one-step-ahead predictions over each period for the initial (red) and the updated model (green) of Exp.5.

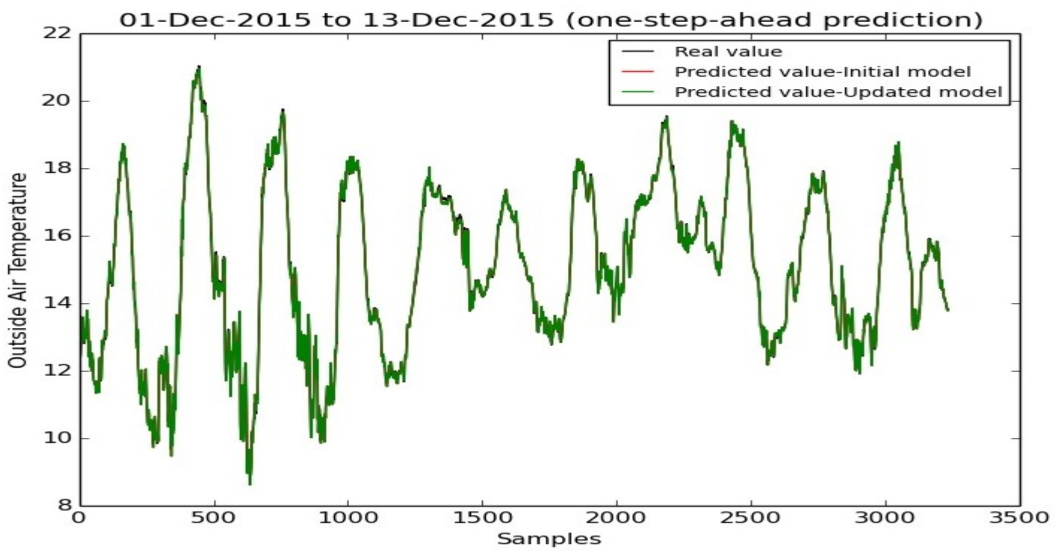


Fig. 8.7. One-step-ahead prediction over the 01-Dec-2015 period in case 1.

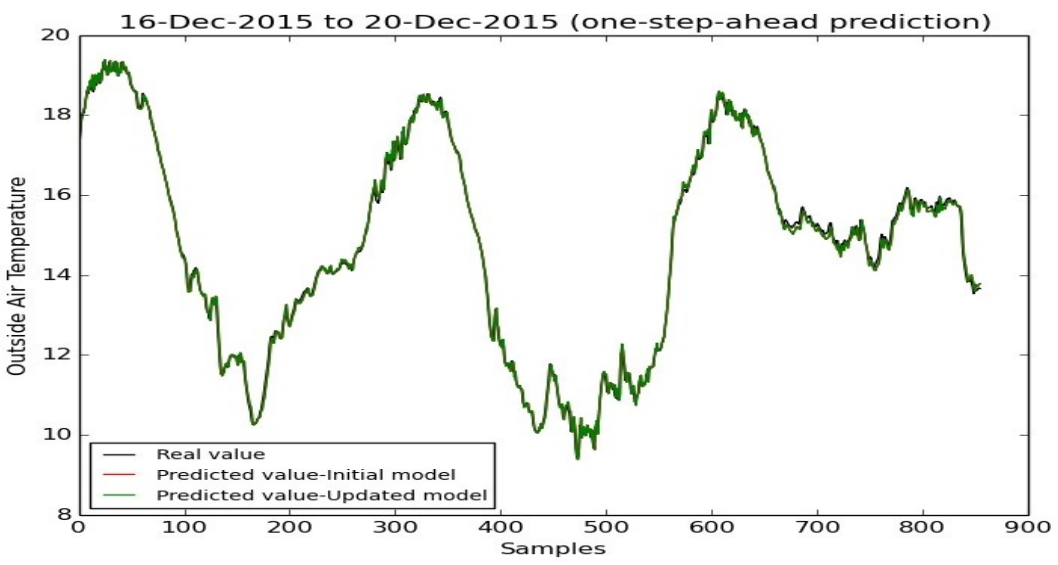


Fig. 8.8. One-step-ahead prediction over the 16-Dec-2015 period in case 1.

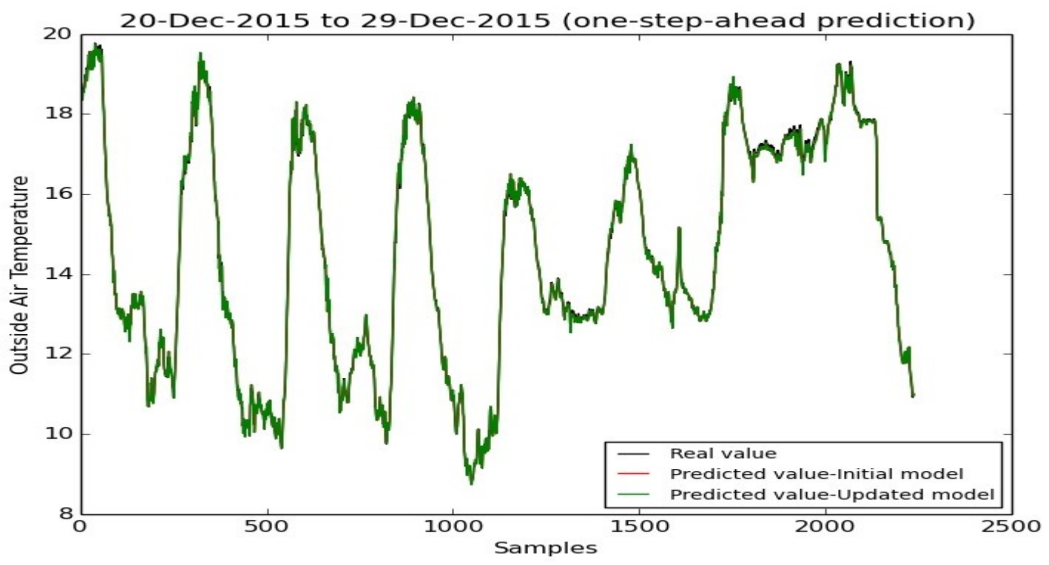


Fig. 8.9. One-step-ahead prediction over the 20-Dec-2015 period in case 1.

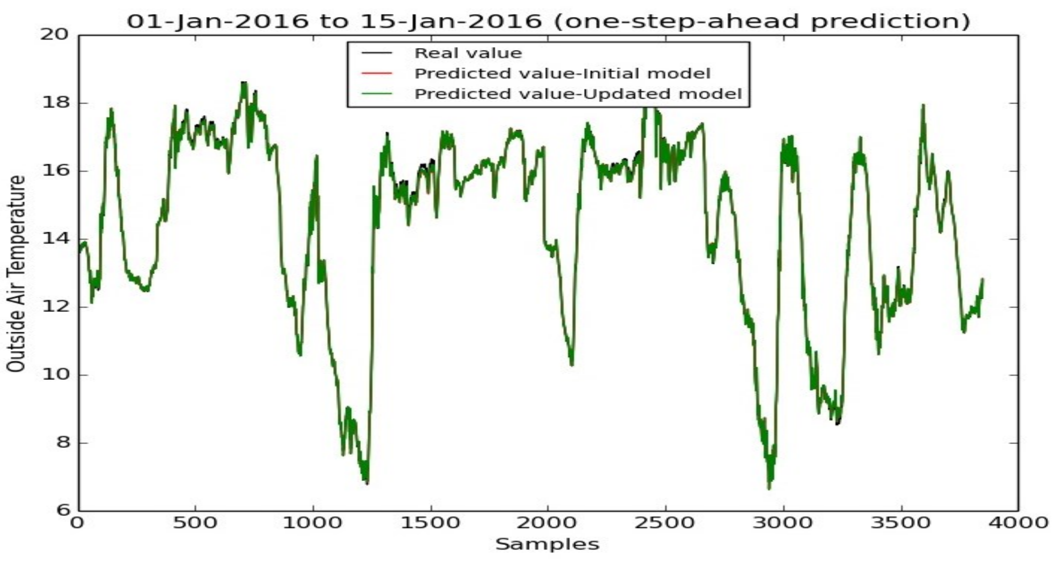


Fig. 8.10. One-step-ahead prediction over the 01-Jan-2016 period in case 1.

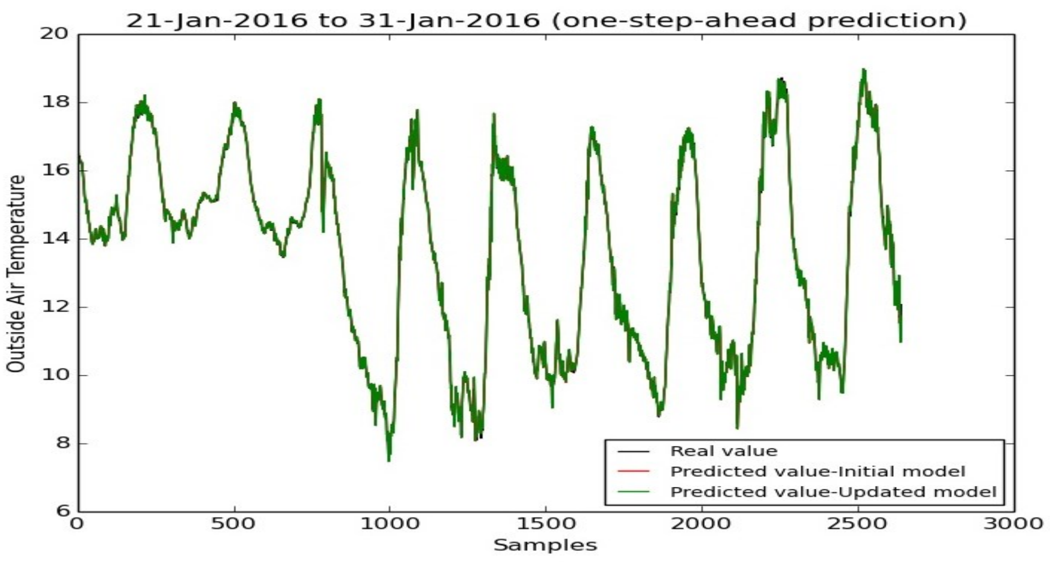


Fig. 8.11. One-step-ahead prediction over the 21-Jan-2016 period in case 1.

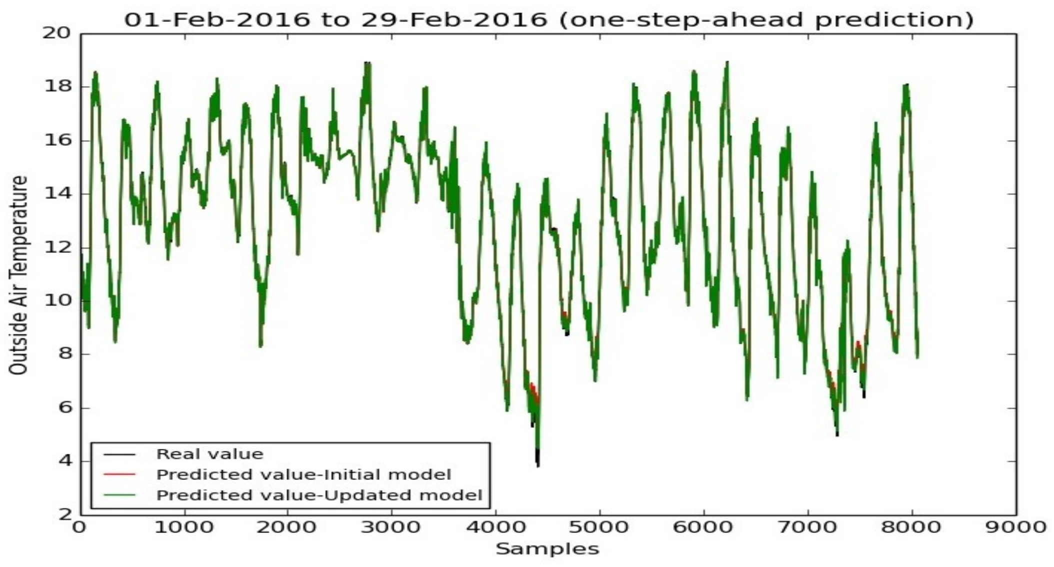


Fig. 8.12. One-step-ahead prediction over the 01-Feb-2016 period in case 1.

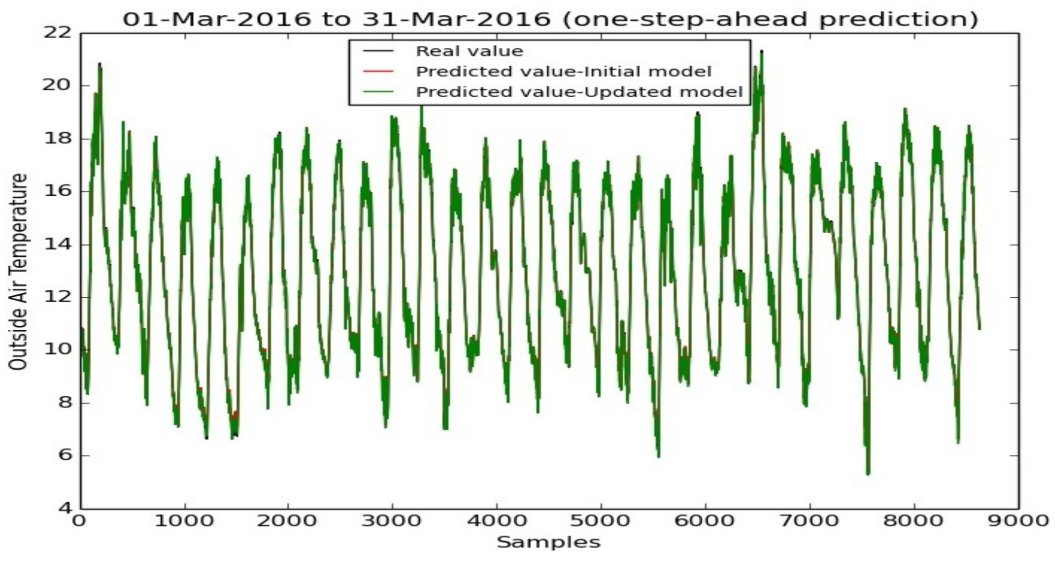


Fig. 8.13. One-step-ahead prediction over the 01-Mar-2016 period in case 1.

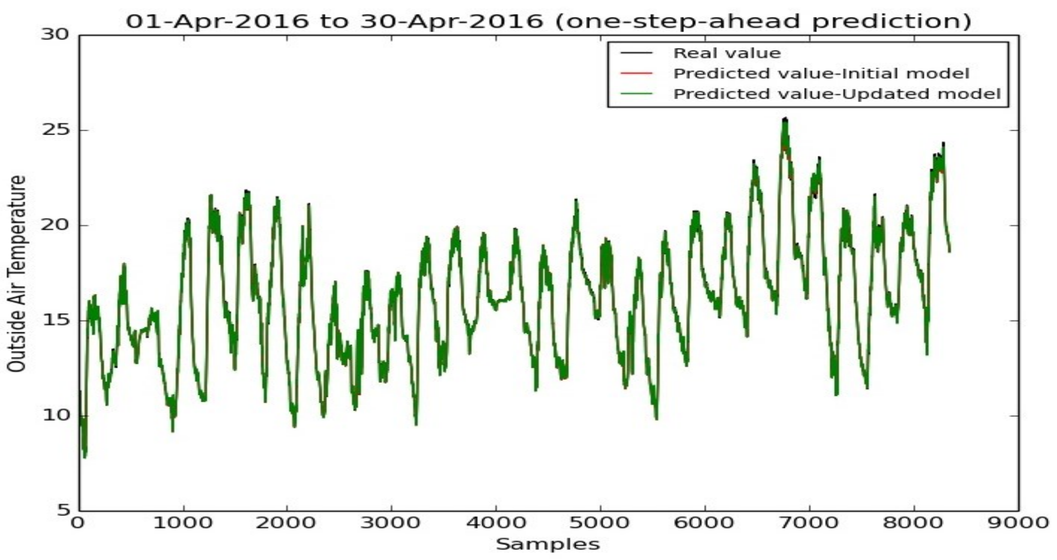


Fig. 8.14. One-step-ahead prediction over the 01-Apr-2016 period in case 1.

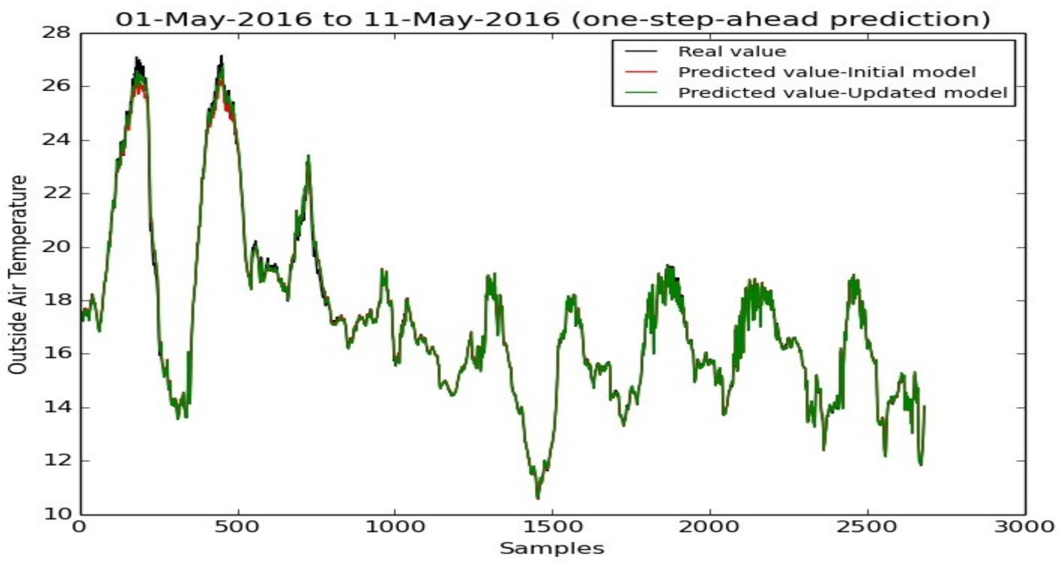


Fig. 8.15. One-step-ahead prediction over the 01-May-2016 period in case 1.

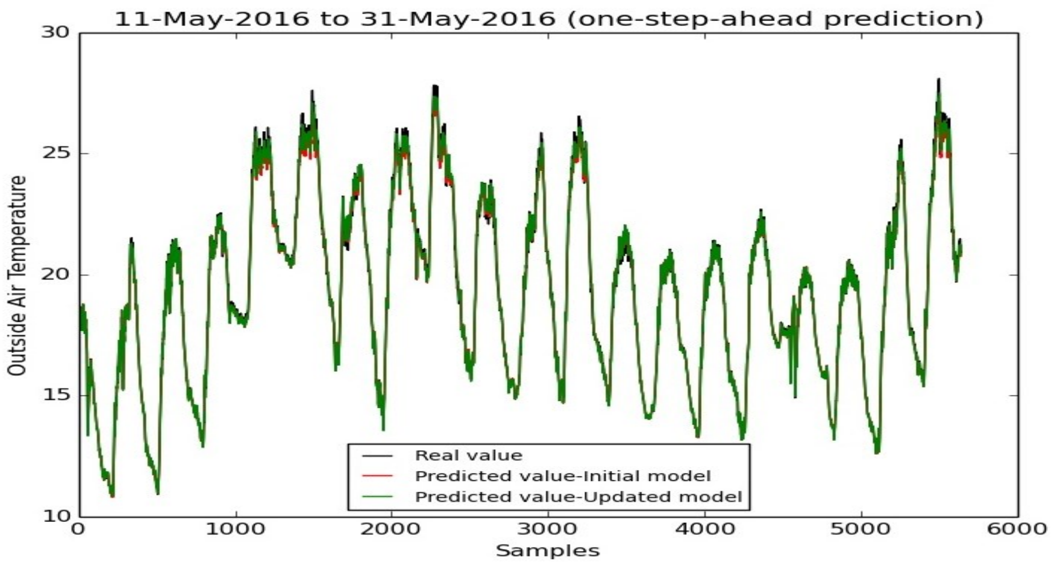


Fig. 8.16. One-step-ahead prediction over the 11-May-2016 period in case 1.

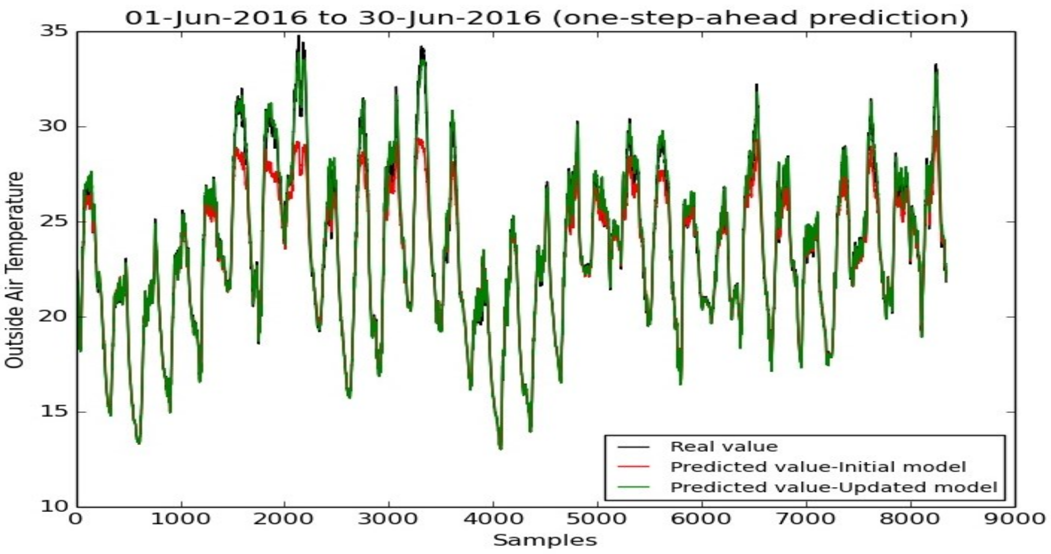


Fig. 8.17. One-step-ahead prediction over the 01-Jun-2016 period in case 1.

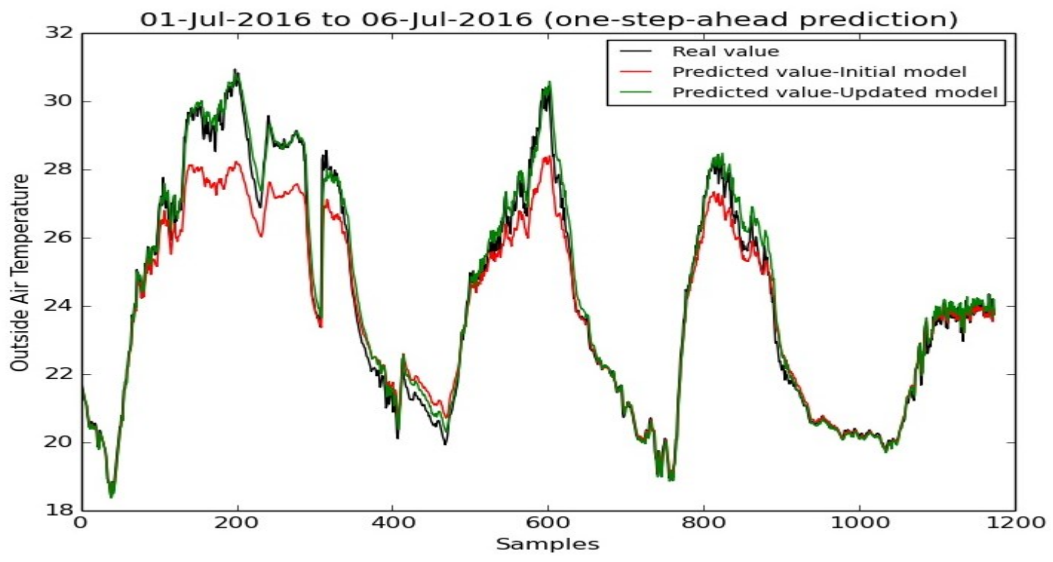


Fig. 8.18. One-step-ahead prediction over the 01-Jul-2016 period in case 1.

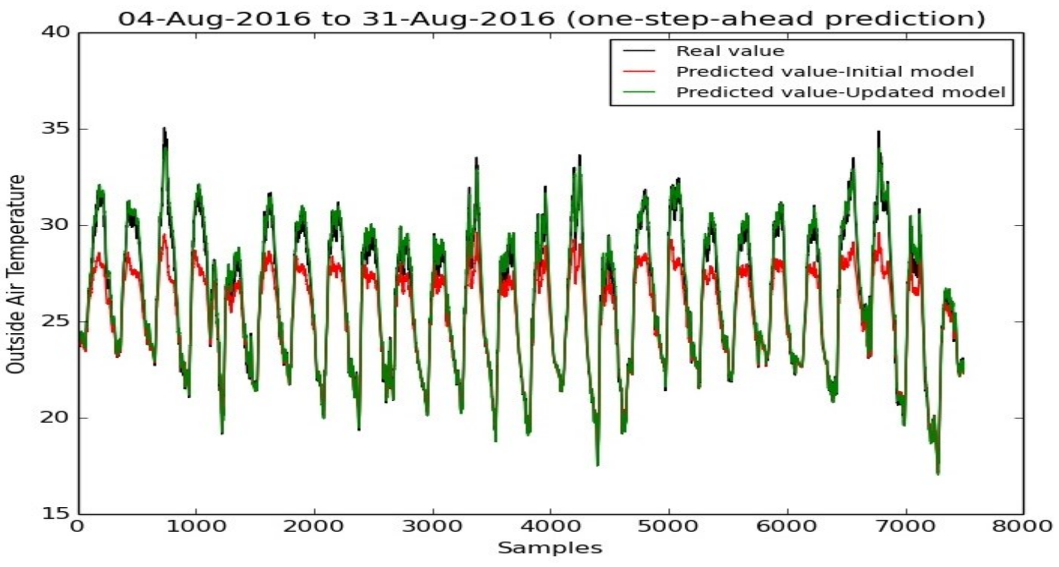


Fig. 8.19. One-step-ahead prediction over the 04-Aug-2016 period in case 1.

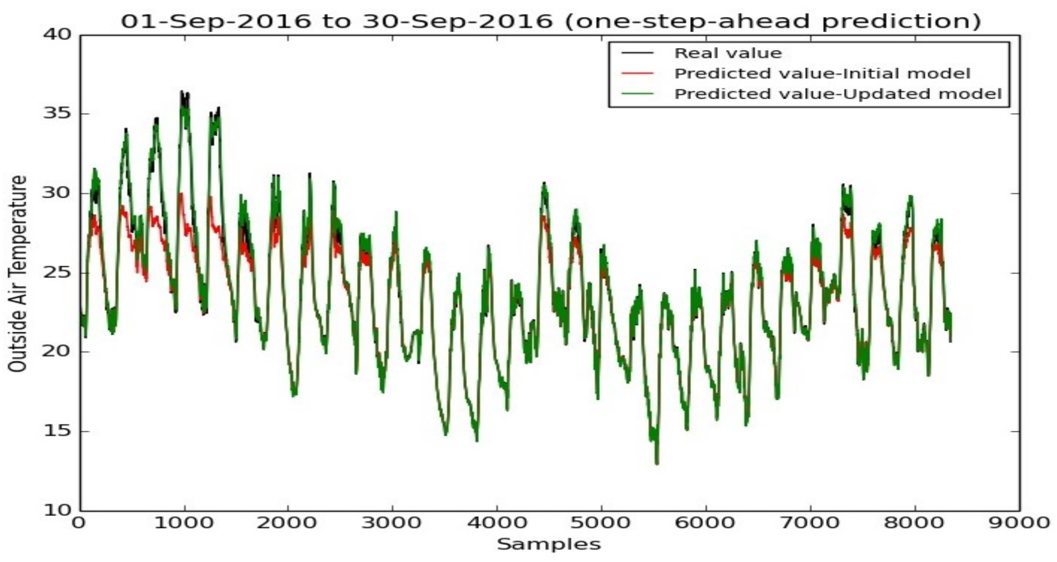


Fig. 8.20. One-step-ahead prediction over the 01-Sep-2016 period in case 1.

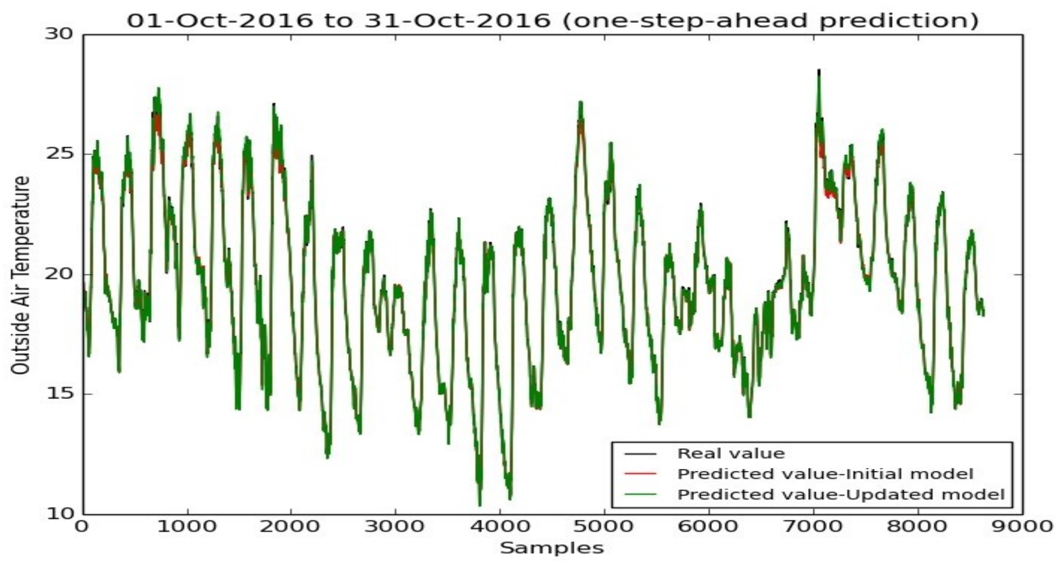


Fig. 8.21. One-step-ahead prediction over the 01-Oct-2016 period in case 1.

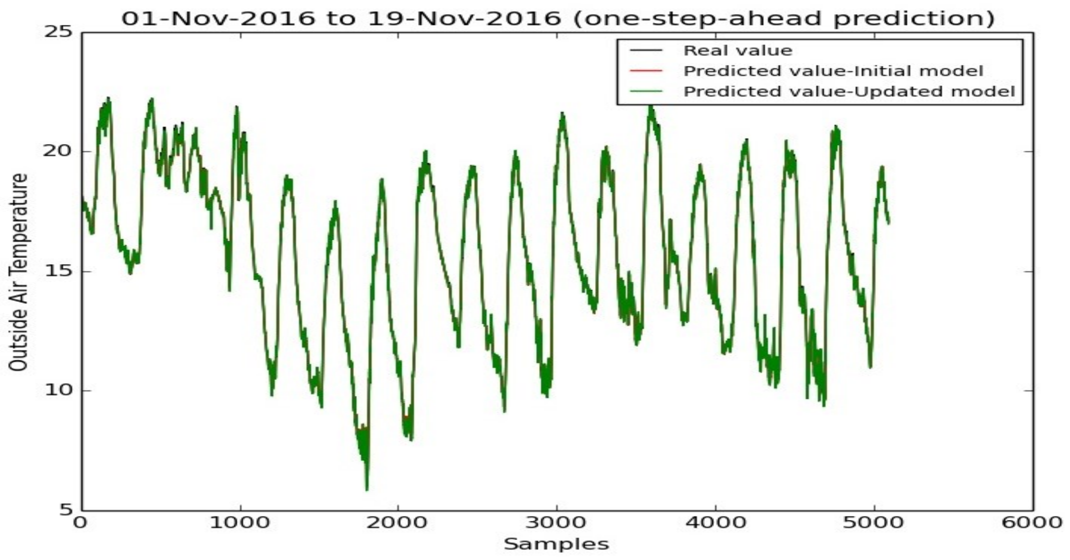


Fig. 8.22. One-step-ahead prediction over the 01-Nov-2016 period in case 1.

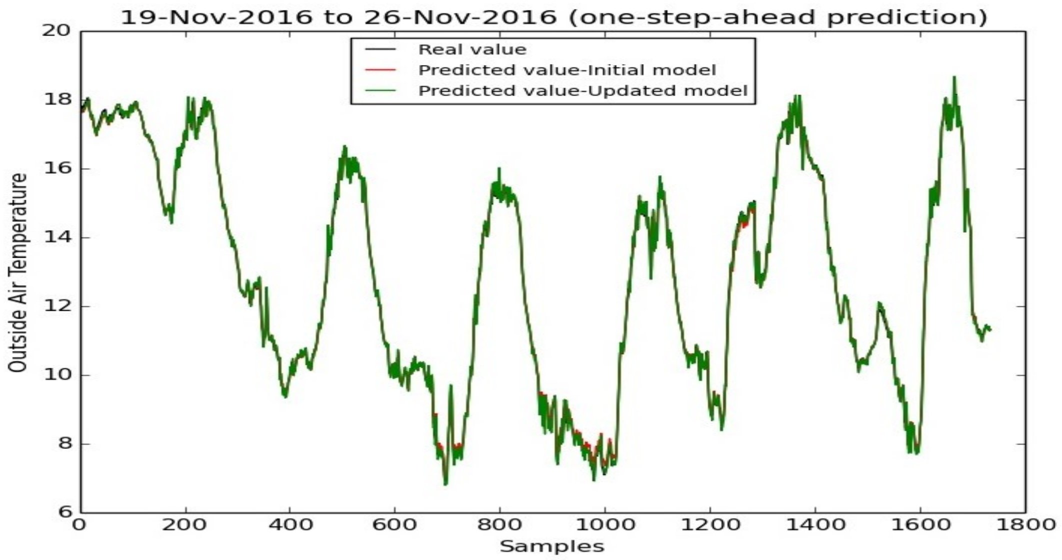


Fig. 8.23. One-step-ahead prediction over the 19-Nov-2016 period in case 1.

8.4.2. Case Study 2: OAT model for the University of Almeria

The data provided by the University of Almeria has been collected over the years 2010 to 2012, including climate variables such as outside air temperature, outside air humidity, outside solar radiation, etc. In the design process, the data in range 02-Sep-2010 to 11-Sep-2010 (i.e., 10 days) with a sample rate of 5 minutes was used to create the training, testing and validation sets with 1548, 516 and 516 points, respectively. ApproxHull was applied on the whole data which resulted in 880 convex hull points, that were included in the training set. Like the previous case study in Section 8.4.1, 73 lags out of the available 300 lags (i.e., corresponding to one day and one hour) were considered by MOGA.. The formal description of the selected OAT model is given in Eq. (8.29).

$$\widehat{T_o}(k+1) = f(T_o(k), T_o(k-1), T_o(k-10), T_o(k-25), T_o(k-30), T_o(k-38), T_o(k-44), T_o(k-276), T_o(k-296)) \tag{8.29}$$

According to Eq. (8.29), the selected model has 9 inputs which are lags of OAT. The corresponding RBFNN model has 14 hidden neurons. To simulate the online adaptation process, 12 periods over the years 2010 and 2011 were considered, shown in Table 8.8. The samples of each period were normalized in the range $[-1,1]$. Since the model has only used 9 lags out of the 73 lags in the design process, the initial convex hull of the model should be obtained from the reduced version of the whole data which was supplied to MOGA. ApproxHull was hence applied to the reduced dataset with 10 dimensions (i.e., 9 inputs and the target pattern) which resulted in 544 convex hull points that were included in the initial training-sliding window.

For all experiments, the scenario of model update throughout the online adaptation process is the same as that for the previous case study. In this case study, 9 different experiments corresponding to 9 different combinations of τ_f, β and η values were carried out. For all experiments, the sizes of the training and the additional sliding-window size were set to 1548 and 500, respectively. The maximum number of iterations of the Levenberg-Marquardt method was set to 100 for all experiments.

The experiments' specification is given in Table 8.9. "ES" in Table 8.9 stands for Early-Stopping method, using the additional sliding window as a test set. As we can see in Table 8.9, three groups of experiments were carried out. For the first group of experiments, the model is updated without applying the early-stopping method. In this case, for each update, the training process ends when the three standard termination criteria are met, without considering the early stopping method. For the second and third groups of experiments, the early-stopping method was applied. In this situation, for the first five iterations of the training process, only the three standard termination criteria are checked. After the initial five iterations, training stops when the three termination criteria (2.40 – 2.42), or early-stopping is met. Early stopping method uses the last 4 iterations.

TABLE 8.8. PERIODS OVER THE YEARS 2010 AND 2011 IN CASE 2.

Period name	Range
Oct	01-Oct-2010 to 19-Oct-2010 (19 days)
Nov	09-Nov-2010 to 28-Nov-2010 (20 days)
Dec	04-Dec-2010 to 15-Dec-2010 (12 days)
Jan	11-Jan-2011 to 31-Jan-2011 (21 days)
Feb	09-Feb-2011 to 28-Feb-2011 (21 days)
Mar	11-Mar-2011 to 31-Mar-2011 (21 days)
Apr	07-Apr-2011 to 12-Apr-2011 (6 days)
May	20-May-2011 to 31-May-2011 (12 days)
Jun	02-Jun-2011 to 23-Jun-2011(22 days)
Jul	08-Jul-2011 to 31-Jul-2011 (24 days)
Aug	01-Aug-2011 to 31-Aug-2011 (31 days)
Sept	02-Sept-2010 to 11-Sept-2010 (10 days)

TABLE 8.9. EXPERIMENT'S SPECIFICATION IN CASE 2.

		τ_f	β	η	ES
First group of experiments	Exp.1	0.001	0.0	0.005	No
	Exp.2	0.001	0.1	0.005	No
	Exp.3	0.001	0.5	0.005	No
Second group of experiments	Exp.4	0.001	0.0	0.005	Yes
	Exp.5	0.001	0.1	0.005	Yes
	Exp.6	0.001	0.5	0.005	Yes
Third group of experiments	Exp.7	0.0001	0.0	0.005	Yes
	Exp.8	0.0001	0.1	0.005	Yes
	Exp.9	0.0001	0.5	0.005	Yes

Similarly, to compare the experiments, the criteria stated in Table 8.3 were used. Since in this case study, as in the previous one, a sample rate of 5 minutes were considered, the number of samples in each period is obtained as 12*24=288*number of days. The statistical results obtained from the three groups of experiments are given in Table 8.10.

TABLE 8.10. STATISTICAL RESULTS OF EXPERIMENTS IN CASE 2.

		n_T	n_A	n_R	n_U	n_I	n_{CH}
First group of experiments	Exp.1	414	58393	377	24	4.25	184
	Exp.2	2134	56684	366	38	3.68	128
	Exp.3	7376	51447	361	61	3.11	191
Second group of experiments	Exp.4	419	58390	375	23	4.35	176
	Exp.5	2103	56712	369	37	3.57	130
	Exp.6	7214	51611	359	72	3	180
Third group of experiments	Exp.7	420	58385	379	23	10.65	182
	Exp.8	2088	56727	369	46	6.52	127
	Exp.9	7404	51419	361	106	4.65	164

According to this Table for each group of experiments, increasing β causes an increase in n_T , due to the fact that the new arriving sampls has more chance to be inserted into the training sliding-window. This, in turn, causes an increase in n_U due to an increase of training sliding-window updates. Moreover, we can see that in all experiments, n_U is much smaller than n_T. This result reveals the fact that, the proposed method can prevent unnecessary parameter updates whenever the training sliding-window is updated due to the insertion of the new arriving sample.

For all experiments, the initial and updated models have been evaluated over each period in the same way used in Section 8.4.1. The evaluation results of the three groups of experiments are given in Tables 8.11 to 8.13.

TABLE 8.11. ONE-STEP-AHEAD PREDICTION IN CASE 2.

		$\boldsymbol{\rho_1^u}$								
		First group of experiments			Second group of experiments			Third group of experiments		
	$\boldsymbol{\rho_1^i}$	Exp.1	Exp.2	Exp.3	Exp.4	Exp.5	Exp.6	Exp.7	Exp.8	Exp.9
Oct	0.075	0.007	0.007	**0.006**	0.007	0.007	**0.006**	0.007	**0.006**	**0.006**
Nov	0.387	0.008	0.008	**0.007**	0.009	0.009	**0.007**	0.008	0.008	**0.007**
Dec	0.290	0.008	0.008	**0.007**	0.009	**0.007**	0.008	**0.007**	**0.007**	**0.007**
Jan	0.480	0.009	**0.005**	**0.005**	0.009	0.006	**0.005**	0.006	**0.005**	**0.005**
Feb	0.409	0.009	**0.007**	0.008	0.009	0.009	**0.007**	**0.007**	**0.007**	**0.007**
Mar	0.293	0.008	0.007	**0.006**	0.007	0.007	0.007	0.007	0.007	**0.006**
Apr	0.152	0.007	**0.006**	**0.006**	**0.006**	**0.006**	0.008	**0.006**	**0.006**	**0.006**
May	0.046	**0.007**	**0.007**	**0.007**	**0.007**	**0.007**	**0.007**	**0.007**	0.008	0.008
Jun	0.024	**0.006**	**0.006**	**0.006**	**0.006**	**0.006**	0.007	0.010	0.007	**0.006**
Jul	0.007	**0.006**	**0.006**	**0.006**	**0.006**	**0.006**	0.007	0.417	0.039	0.008
Aug	0.017	0.007	**0.006**	**0.006**	0.007	**0.006**	**0.006**	2.271	0.022	0.008
Sept	**0.005**	**0.005**	**0.005**	**0.005**	**0.005**	**0.005**	**0.005**	0.006	**0.005**	0.006

As it can be concluded from Table 8.11, the performance of the updated model for the one-step-ahead prediction (i.e., 5 minutes ahead prediction) for all experiments over the periods Oct to Jun is much better than that of the initial model. Regarding the period Aug, the performance of the updated model for all experiments except Exps.7 and 8 is also better than that of the initial model. Moreover, for all experiments except Exps.7 and 8, the performance of the updated model over the period Jul is somehow the same as that of the initial model. Furthermore, over the period Sept, for all experiments, similar performances of the updated model and the initial model can be observed, which is due to the fact that the range of data, in those months, is similar to the range used in the offline design (12 days in September of the last year).

TABLE 8.12. 48-STEPS-AHEAD PREDICTION IN CASE 2.

		ρ_{48}^{u}								
		First group of experiments			**Second group of experiments**			**Third group of experiments**		
	ρ_{48}^{i}	**Exp.1**	**Exp.2**	**Exp.3**	**Exp.4**	**Exp.5**	**Exp.6**	**Exp.7**	**Exp.8**	**Exp.9**
Oct	0.161	0.119	0.124	0.098	0.122	0.120	0.106	0.134	0.125	**0.085**
Nov	0.387	0.126	0.158	**0.107**	0.159	0.159	0.137	0.307	0.112	0.134
Dec	0.314	**0.140**	0.150	0.148	0.173	0.149	0.162	0.202	0.146	0.160
Jan	0.469	0.117	**0.113**	0.181	0.168	0.180	0.147	0.212	0.151	0.146
Feb	0.407	0.147	0.138	0.165	0.150	0.166	0.148	0.212	**0.135**	0.144
Mar	0.307	0.149	0.159	0.127	0.156	**0.126**	0.150	0.193	0.127	0.158
Apr	0.216	0.144	0.192	**0.110**	0.148	0.116	0.134	0.139	0.143	0.127
May	0.115	0.135	0.136	0.115	0.135	**0.099**	0.119	0.112	0.141	0.165
Jun	0.103	0.116	0.133	0.087	0.115	**0.072**	0.105	0.083	0.108	0.936
Jul	**0.079**	0.084	0.120	0.135	0.084	0.090	0.117	0.099	0.095	0.127
Aug	0.081	0.097	0.109	0.108	**0.080**	0.097	0.100	0.102	0.101	0.120
Sept	**0.051**	0.068	0.087	0.098	0.063	0.083	0.083	0.070	0.107	0.102

With respect to the 48-steps-ahead prediction (i.e., 4 hours ahead prediction), in all experiments, the updated model considerably outperforms the initial model over the periods Nov to Apr. Regarding the periods May and Jun, for Exp.5, the updated model has the best performance and is superior to the initial model. For Exp.4, the updated model has the best performance over the period Aug in comparison with the others and has the same performance as the initial model. Moreover, the initial model over the periods Jul and Sept has the best performance, in comparison with the corresponding updated model of each experiment.

TABLE 8.13. SUMMATION OVER PH IN CASE 2.

		S^u								
		First group of experiments			**Second group of experiments**			**Third group of experiments**		
	S^i	**Exp.1**	**Exp.2**	**Exp.3**	**Exp.4**	**Exp.5**	**Exp.6**	**Exp.7**	**Exp.8**	**Exp.9**
Oct	7.442	4.077	4.990	3.740	4.423	4.373	3.869	4.092	4.033	**2.916**
Nov	19.98	5.640	6.124	**4.283**	5.528	6.475	4.736	8.756	4.532	4.543
Dec	16.15	5.853	6.004	5.225	6.156	5.280	5.019	6.662	5.306	**4.877**
Jan	23.79	5.245	**4.117**	5.456	5.346	5.457	4.343	6.294	4.396	4.672
Feb	20.77	7.208	5.021	5.645	5.503	5.782	4.853	6.732	**4.481**	4.973
Mar	16.17	7.880	5.745	4.252	6.130	5.017	4.598	6.453	**4.218**	5.292
Apr	10.80	5.639	5.533	4.073	4.883	4.255	4.308	4.827	**3.893**	3.955
May	5.119	4.267	4.298	4.299	4.291	**3.349**	3.963	3.709	4.222	4.739
Jun	4.239	3.629	4.008	3.033	3.790	**2.540**	3.646	2.785	3.341	7.373
Jul	2.716	2.733	3.855	3.939	**2.699**	3.009	4.139	6.107	3.831	4.147
Aug	3.026	3.179	3.444	3.730	**2.718**	3.325	3.495	9.037	3.501	3.836
Sept	**1.684**	2.210	2.765	3.300	2.082	2.889	3.130	2.482	3.202	3.297

In order to analyze the performance of the initial and updated model over the whole prediction horizon within each period, we compared the initial model with the updated model in terms of the summation of RMSEs over the prediction horizon of 48 steps, within each period. Similarly, the bold values in Table 8.13 denote the best result over each period. In all experiments, the updated model performs much better than the initial model over the prediction horizon for the periods Oct to Apr. For the remaining periods, in all experiments except in Exp.7 for the periods Jul and Aug and in Exp.9 for the period Jun, the behavior of the updated model over the prediction horizon is, to some extent, similar with that of the initial model.

To sum up, based on the evaluation results shown in Tables 8.11 to 8.13, we can say that in all experiments, the performance of the updated model within the periods of autumn and winter is much better than that of the initial model, which has been trained based on September data. On the other hand, in all experiments except Exp.7 and Exp.9, the updated model within the periods of spring and summer can keep the mappings which have been obtained for the previous periods. Hence, the behavior of the updated models, for all periods, is better or similar to that of the initial model, the latter obtained when the range of the considered period is similar to the one used for off-line model design.

In order to graphically compare the performance of the updated model with the initial model, the updated model of Exp.1 was selected. Figs. 8.24 to 8.35 illustrate the one-step-ahead prediction over each period for both initial and updated model of Exp.1.

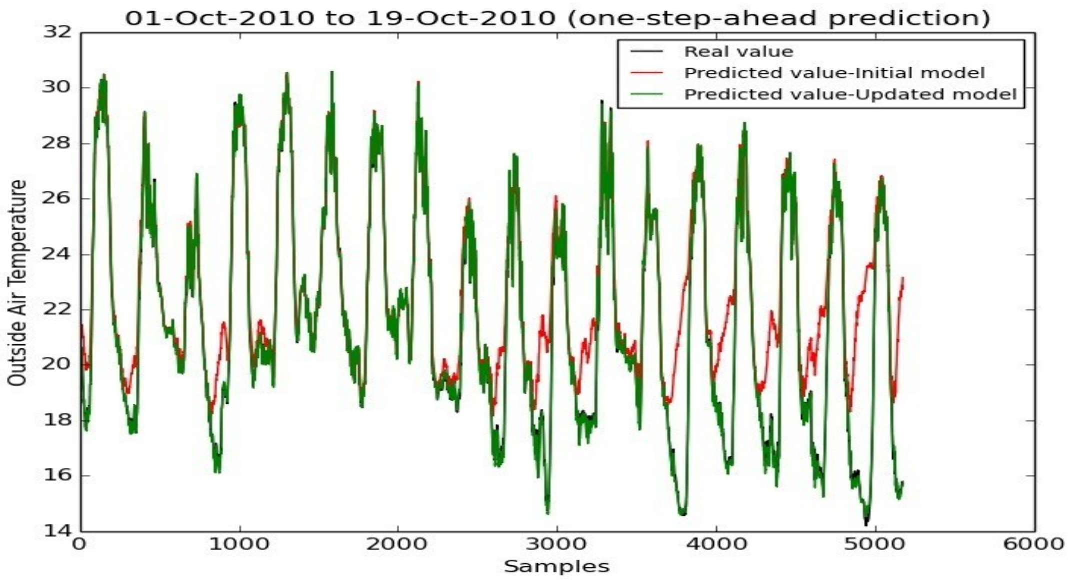


Fig. 8.24. One-step-ahead prediction over the Oct period in case 2.

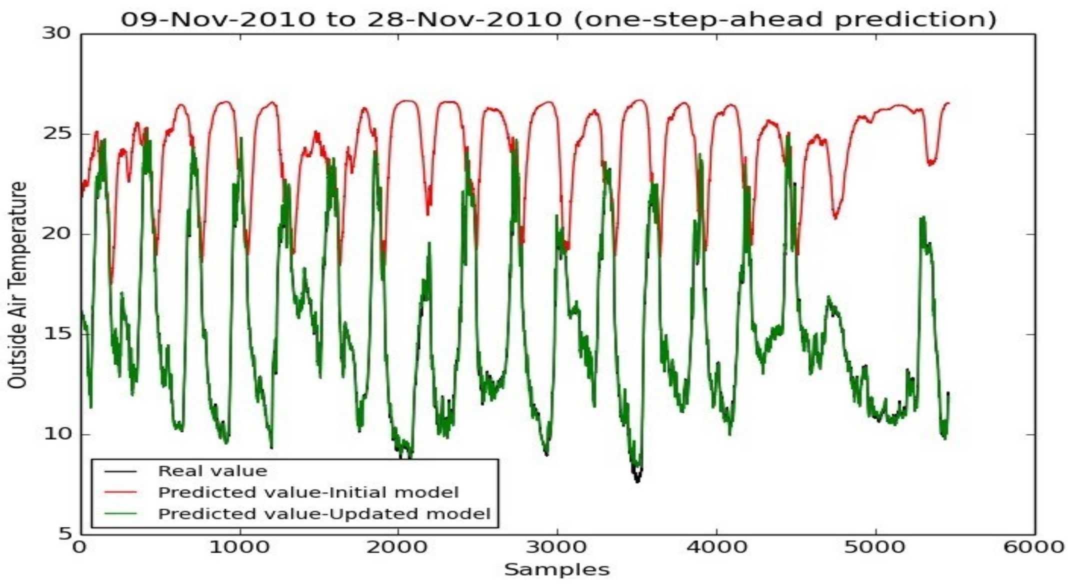


Fig. 8.25. One-step-ahead prediction over the Nov period in case 2.

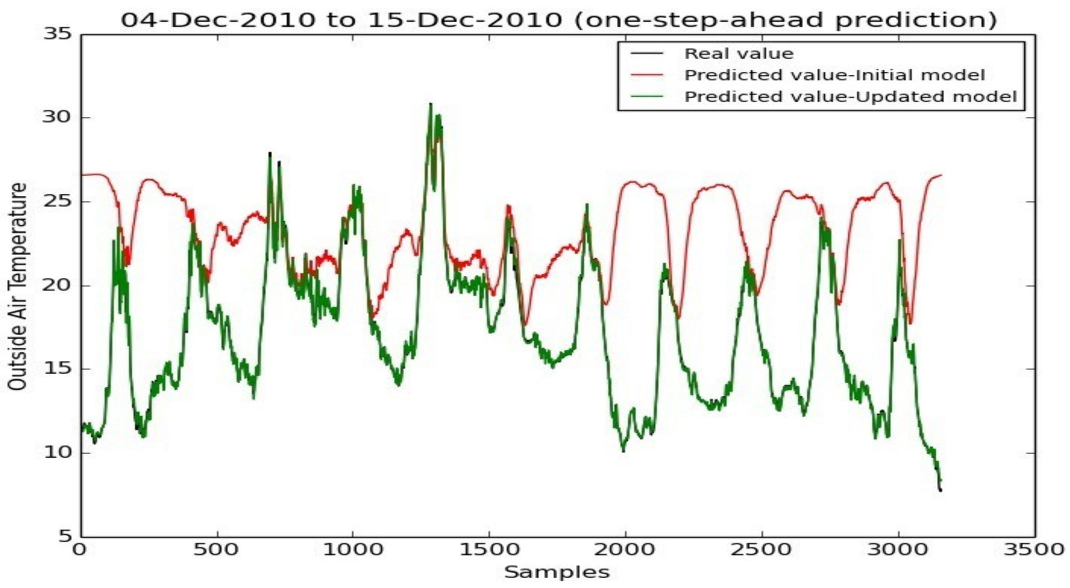


Fig. 8.26. One-step-ahead prediction over the Dec period in case 2.

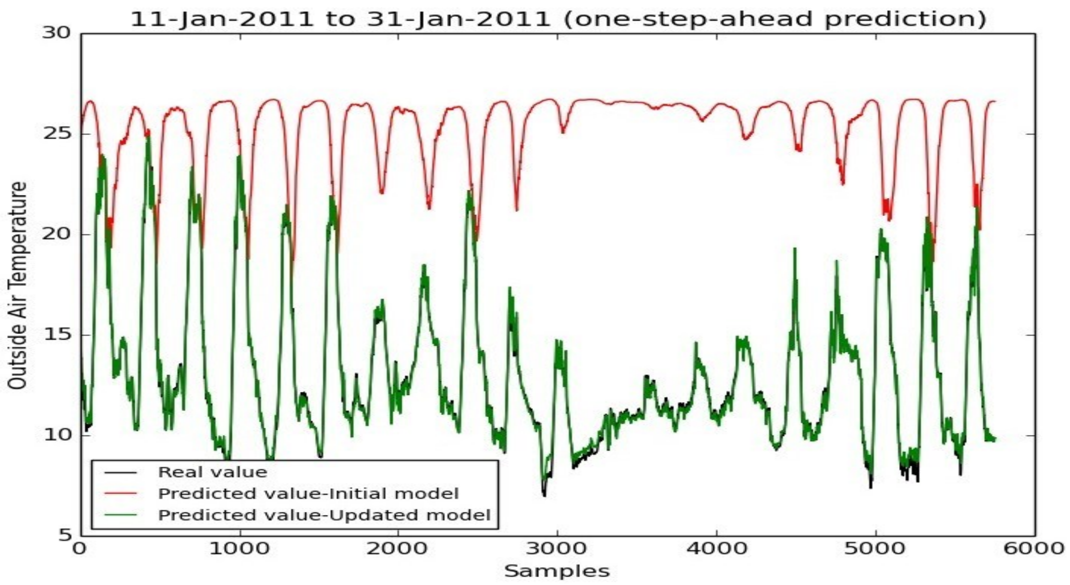


Fig. 8.27. One-step-ahead prediction over the Jan period in case 2.

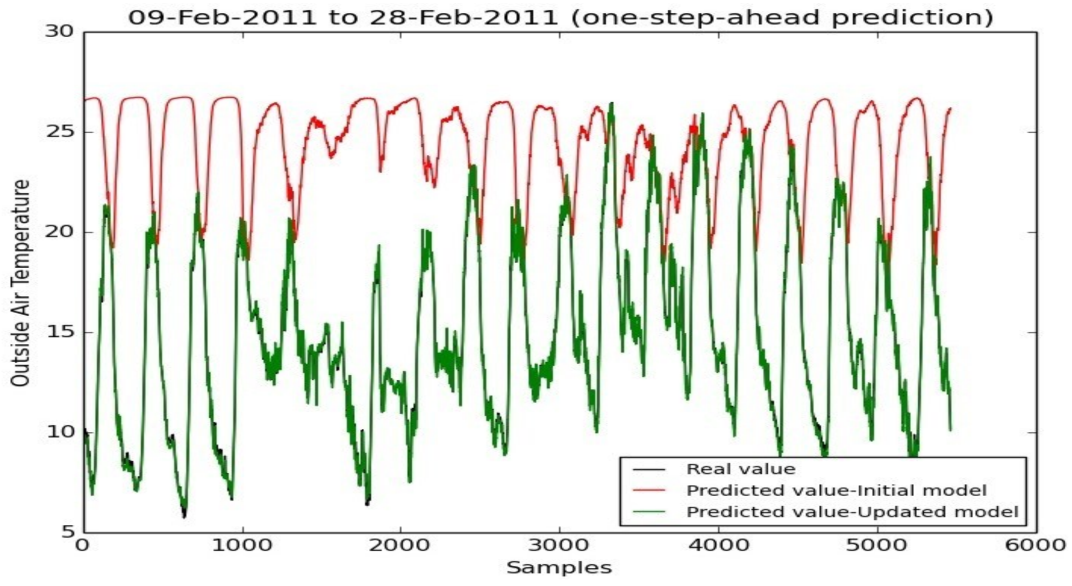


Fig. 8.28. One-step-ahead prediction over the Feb period in case 2.

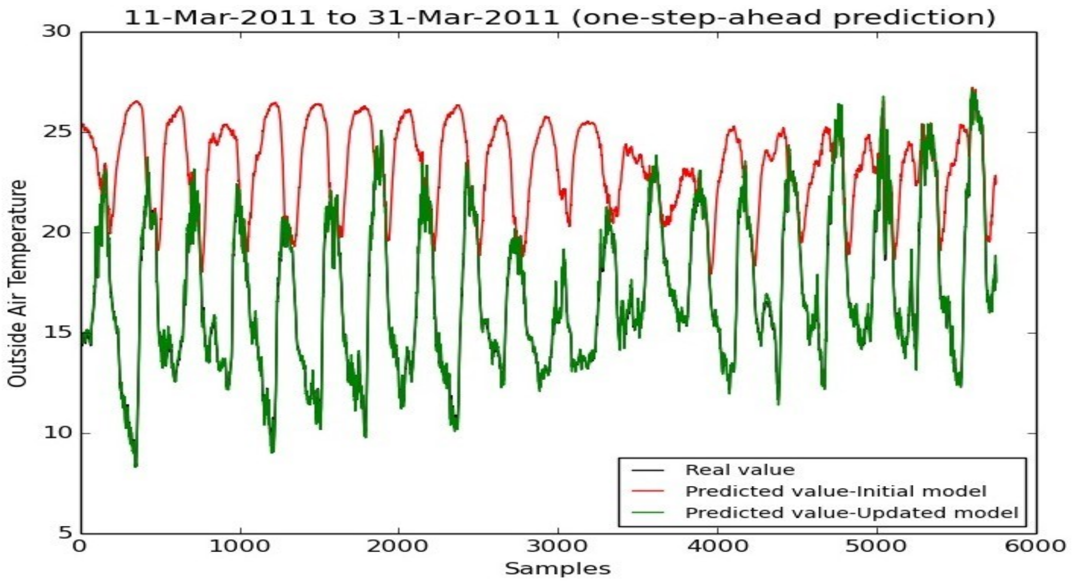


Fig. 8.29. One-step-ahead prediction over the Mar period in case 2.

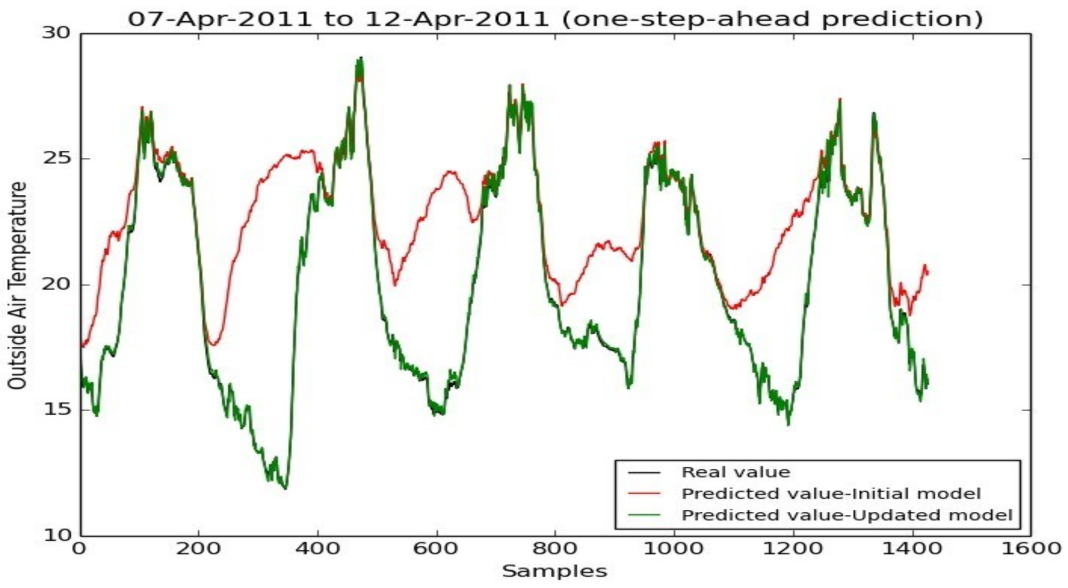


Fig. 8.30. One-step-ahead prediction over the Apr period in case 2.

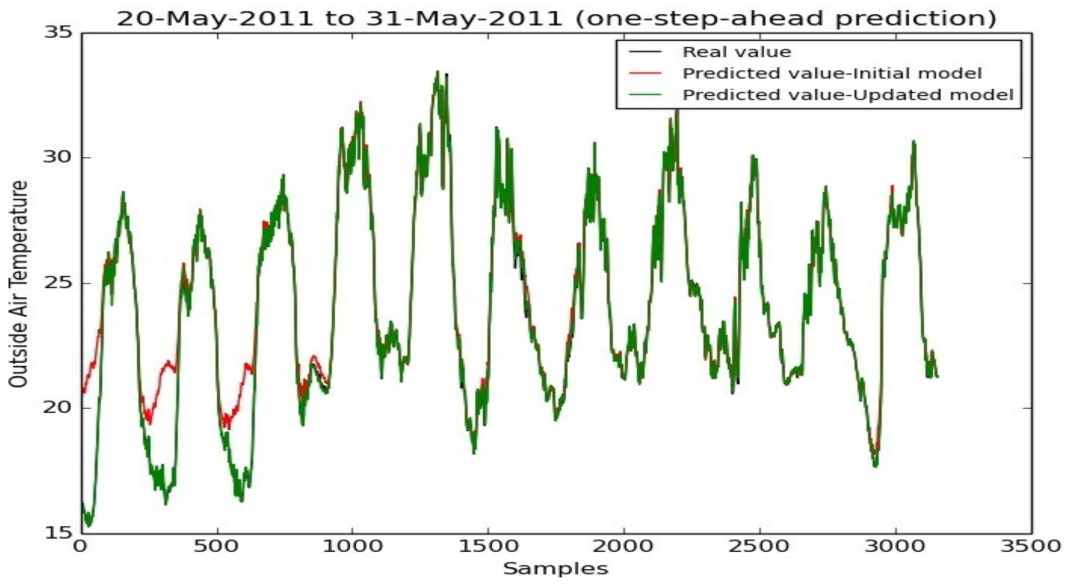


Fig. 8.31. One-step-ahead prediction over the May period in case 2.

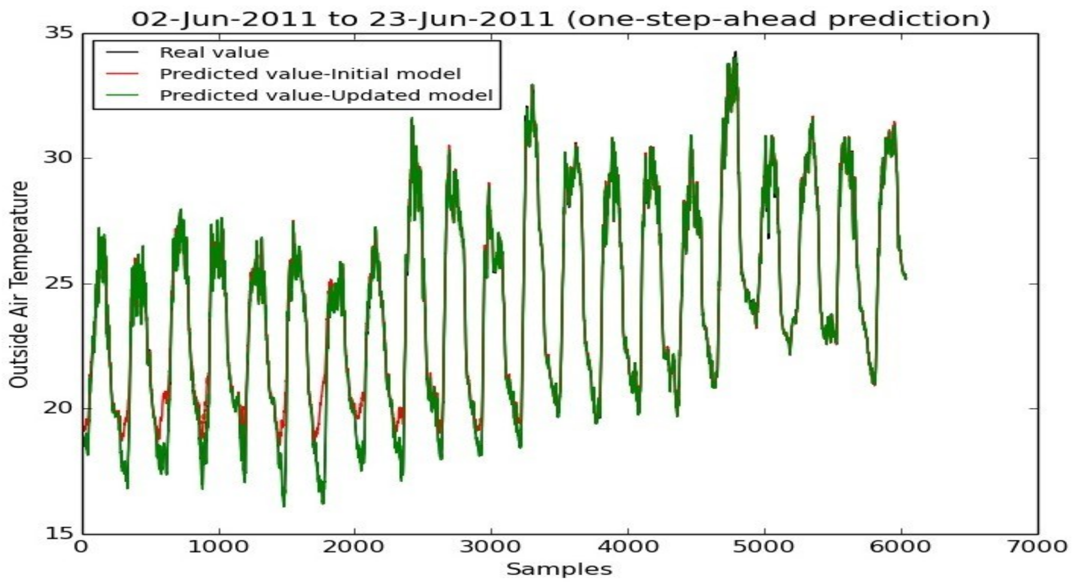


Fig. 8.32. One-step-ahead prediction over the Jun period in case 2.

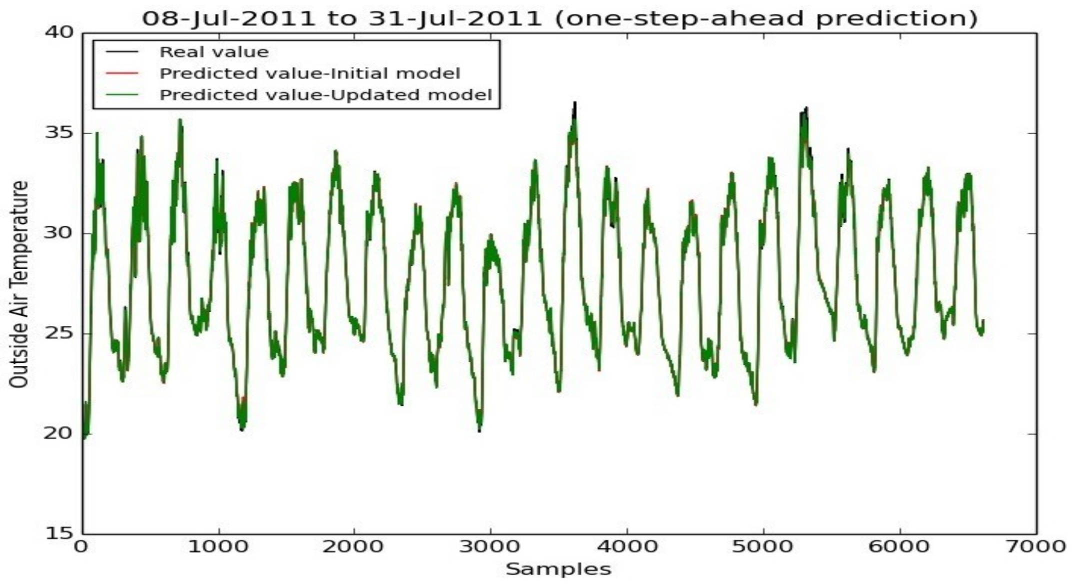


Fig. 8.33. One-step-ahead prediction over the Jul period in case 2.

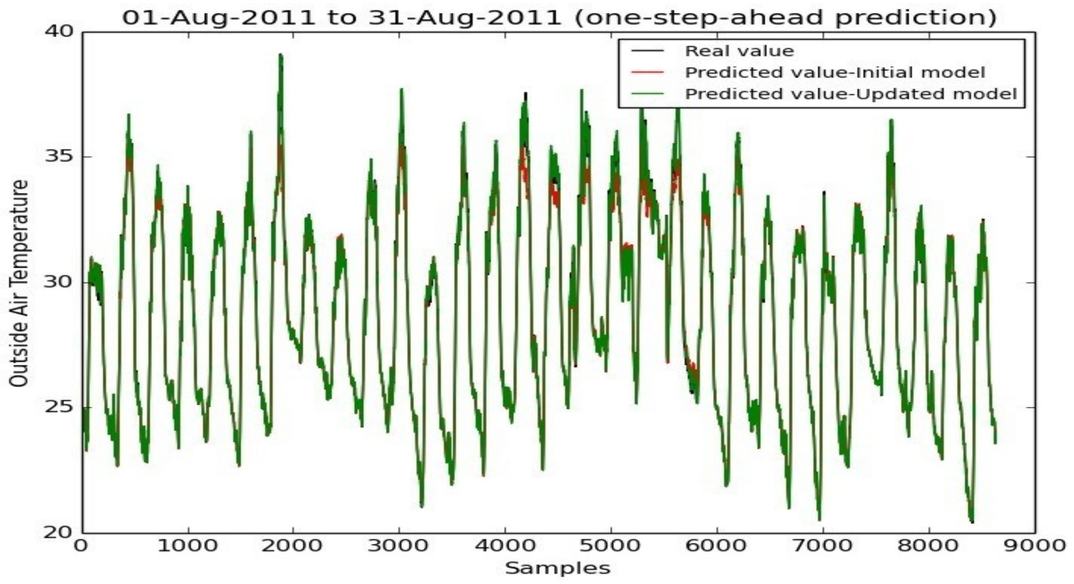


Fig. 8.34. One-step-ahead prediction over the Aug period in case 2.

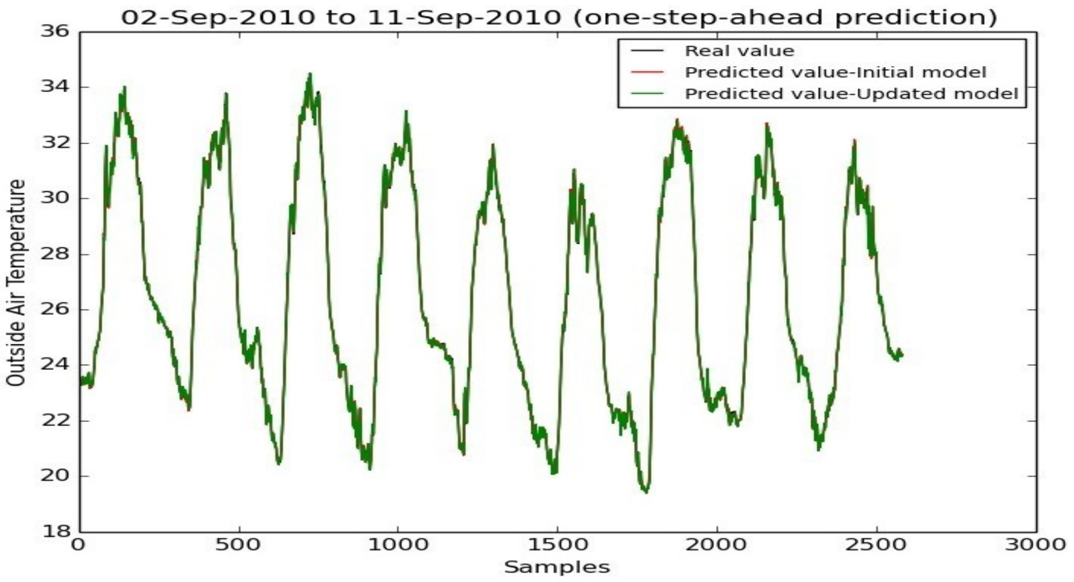
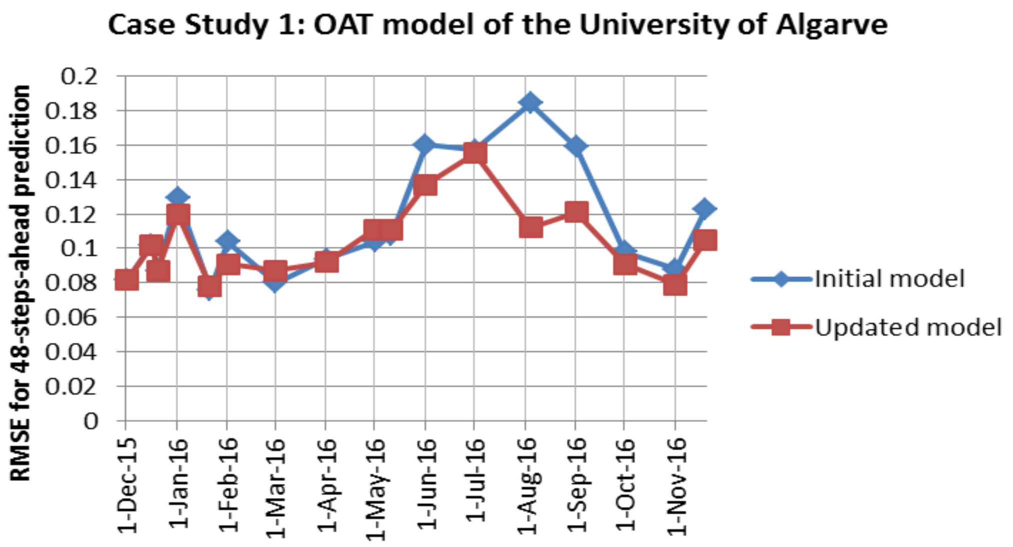


Fig. 8.35. One-step-ahead prediction over the Sept period in case 2.

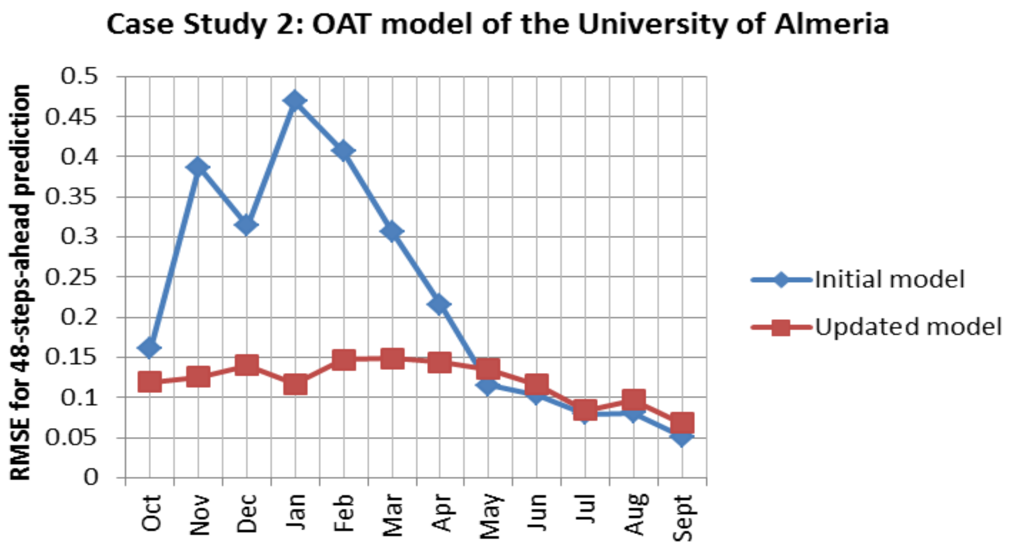
As we can see in Figs. 8.24 to 8.35, there is a significant difference between the initial and the updated models at the end of Oct to May periods. For these periods, comparing the predicted value (the green curve) with the corresponding real value (the black curve), the updated model has a much higher level of accuracy at the end of each period. As it can be seen, in those months, the output range of the initial model is similar to the one obtained in September, where the off-line design was done Moving to summer months, we can see that the difference between the updated model and the initial model is decreasing, but with the former performing better. It reflects the fact that the updated model not only keeps the mappings which have been constructed over the periods of winter but also adapts itself with new samples arriving during the summer periods.

8.5. Comparison between the two case studies

In order to compare the first case study (i.e., OAT model of the University of Algarve) with the second one (i.e., OAT model of the University of Almeria), Exp.4 from the first case study and Exp.1 from the second one were selected (i.e., the corresponding graphs of Exp.4 and Exp.1 over each period were shown in Section 8.4.1 and 8.4.2, respectively.). Fig 8.36 shows the comparison of the updated models at the end of each period in Exp.4 and Exp.1 with their corresponding initial model in terms of RMSE for the 48-steps-ahead prediction.
As it can be seen in Fig. 8.36, the difference between the performance of the updated model in Exp.1 (i.e., Fig 8.36(b)) and that of its corresponding initial model is significantly larger than that in Exp.4 (i.e., 8.36(a)). Regarding the initial model of case study 1, we can say that the initial model could somehow cover the operating regions of all periods except 1-Aug-2016 and 1-Sep-2016, where model update was necessary. This stems from the fact that the range of data used to design the initial model (i.e., November data) covers, to some extent, the range of most periods.
In contrast, in case 2, the initial model has a considerably worse performance than the updated model, for the majority of the periods. As it can be seen in Fig. 8.36(b), the performance difference between over periods Nov to Apr is considerable. This is explained by the observation that in September the temperature ranges from 20^{o} to 34^{o}, roughly (please see Fig. 8.35), and in several months the minimum temperature is much lower, while in Summer months the maximum is higher than 34^{o}.



(a)



(b)

Fig. 8.36. Comparison of the updated model with its corresponding initial model. (a) Exp.4 in case study 1; (b) Exp.1 in case study 2.

8.6. Comparison with other methods

This section addresses the comparison of the proposed online adaptation method, herein after called CHSWNLM, with others.

As it has been referred, in [168] two methods using a sliding window strategy, called SWNLM and SAWNLM were proposed, and served as the basis of the method introduced in this thesis. Recalling, in SWNLM, the sliding window is managed using FIFO policy, while in the SAWNLM, the sliding-window management policy is based on a dissimilarity measure. In order to compare the CHSWNLM with the SWNLM and SAWNLM methods, data from case study 2 (i.e., the OAT model for the University of Almeria) was used with the same scenario mentioned in Section 8.4.2. The statistical and evaluation results are shown in Table 8.14 and 8.15, respectively. In Table 8.14, $\boldsymbol{n}$ denotes the number of new arriving samples over all periods. The other statistics in this table are ones used in Table 8.3. In Table 8.15, $\boldsymbol{\rho_1}$ and $\boldsymbol{\rho_{48}}$ denote the scaled one-step-ahead and the 48-steps-ahead $RMSEs$ associated with the updated model at the end of each period, respectively. The bold values in these tables refer to the best results.

TABLE 8.14. COMPARISON OF STATISTICAL RESULTS OBTAINED BY EXP.1, SWNLM AND SAWNLM.

	$\boldsymbol{n}$	$\boldsymbol{n_T}$	$\boldsymbol{n_A}$	$\boldsymbol{n_R}$	$\boldsymbol{n_U}$	$\boldsymbol{n_I}$
CHSWNLM	59184	**414**	58393	377	**24**	4.25
SWNLM	59184	59184	-	0	270	2.77
SAWNLM	59184	58794	-	390	52	3.21

As it can be seen in Table 8.14, the total number of new arriving samples which have been inserted into the training sliding window ($\boldsymbol{n_T}$), the total number of updates ($\boldsymbol{n_U}$) and the total number of iterations ($\boldsymbol{n_U} \times \boldsymbol{n_I}$) in CHSWNLM are much smaller than in the other methods.

TABLE 8.15. COMPARISON OF CHSWNLM WITH SWNLM AND SAWNLM.

	$\boldsymbol{\rho_1}$			$\boldsymbol{\rho_{48}}$		
	CHSWNLM	SWNLM	SAWNLM	CHSWNLM	SWLNM	SAWNLM
Oct	0.030	**0.015**	0.022	0.115	0.115	**0.114**
Nov	0.045	0.042	**0.040**	0.148	0.854	**0.144**
Dec	**0.008**	6.977	**0.008**	**0.140**	2.855	0.162
Jan	0.009	143.44	**0.007**	**0.118**	1.278	0.133
Feb	0.010	8.353	**0.007**	**0.149**	1.006	0.158
Mar	0.008	0.024	**0.007**	0.150	**0.104**	0.168
Apr	**0.007**	0.150	**0.007**	0.145	0.160	**0.142**
May	**0.007**	0.017	**0.007**	0.136	0.112	**0.131**
Jun	**0.006**	0.554	**0.006**	0.116	0.149	**0.112**
Jul	**0.007**	0.010	**0.007**	**0.084**	0.111	0.096
Aug	**0.008**	0.013	**0.008**	0.094	**0.089**	0.127
Sept	**0.006**	0.013	**0.006**	0.068	**0.067**	0.120

According to Table 8.14, SWNLM is clearly the worst method. Regarding CHSWNLM and SAWNLM the mean values of $\boldsymbol{\rho_1}$ are 0.0126 and 0.0110, respectively, while the corresponding values for $\boldsymbol{\rho_{48}}$ are 0.1219 and 0.1339. Therefore, for this data, SAWNLM is better than CHSWNLM for small prediction horizons, but worse for large ones.

As stated in Section 8.2.2, the method proposed in [179] is an efficient online version of offline method ELM [191, 192]. In this method, called OS-ELM, the centers and spreads are arbitrarily chosen and only the weights as linear parameters are updated. This method was evaluated in several benchmarks in classification, regression and time series problems. It was also evaluated using two types of feedforward networks: MLPs and RBFNNs. Moreover in [179], the proposed method was compared with other online methods including RAN [180], RAN-EKF [171], MRAN [181] and GGAP-RBF [193]. In this study, the CHSWNLM was applied on Mackey-Glass time series stated in [179] where a RBFNN model was considered. The time series problem is generated from the following delay differential equation as (8.30).

$$\frac{dx(t)}{d(t)} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t) \tag{8.30}$$

By integrating Eq. (8.30) over the time interval $[t, t+1]$, the equation for one-step-ahead prediction is obtained as Eq. (8.31).

$$x(t+1) = \frac{2-b}{3}x(t) + \frac{a}{2+b}\left[\frac{x(t+1-\tau)}{1+x^{10}(t+1-\tau)} + \frac{x(t-\tau)}{1+x^{10}(t-\tau)}\right] \tag{8.31}$$

The time series used is generated under the condition $x(t-\tau) = 0.3$ for $0 \leq t \leq \tau$, $a = 0.2$, $b = 0.1$ and $\tau = 17$ and predicted using the four past samples s_{k-50}, s_{k-44}, s_{k-38} and s_{k-32} for each time instant k. Therefore, the time series predictive model can be described as Eq. (8.32).

$$y(k) = F(s_{k-50}, s_{k-56}, s_{k-62}, s_{k-68}) \tag{8.32}$$

In the phase of performance evaluation in [179], the weights as linear parameters of the corresponding RBFNN model are adjusted using the proposed online adaptation method based on the training set of size 4000 samples; then the model is evaluated based on the one-step-ahead prediction RMSE in the training and in a testing set of size 500 samples. All samples were scaled in the range [0, 1]. In our work, in order to compare the CHSWNLM with the others in [179], a fixed-structure RBFNN model with 120 hidden neurons was used as the one that was selected in the OS-ELM method. To keep further consistency, in our study, an initial sliding window of the first 1620 samples (i.e., number of hidden neurons + 1500) was considered. Moreover, the next 650 samples were selected to initialize the additional sliding window and then the next 1730 samples were considered as new arriving samples throughout online adaptation process. Finally, the last 500 samples were constituted the testing set. In this study, $\boldsymbol{\tau_f}$, $\boldsymbol{\eta}$ and $\boldsymbol{\beta}$ were set to 0.001, 0.005 and 0.5, respectively.

The comparison of evaluation results obtained by CHSWNLM and those achieved by the others methods in [179] is given in Table 8.16. In this Table, ρ_{tr} and ρ_{te} denote the average of RMSE on the training and testing sets over 50 trials, respectively. In addition, n_n refers to the number of hidden neurons in the corresponding RBFNN model.

As it can be observed in Table 8.16, CHSWNLM is much superior to the other methods.

TABLE 8.16. COMPARISON BETWEEN THE CHSWNLM METHOD AND OTHER METHODS DESCRIBED IN [179].

	ρ_{tr}	ρ_{te}	n_n
OS-ELM	0.0184	0.0186	120
GGAP-RBF	0.0700	0.0368	13
MRAN	0.1101	0.0337	16
RAN-EKF	0.0726	0.0240	23
RAN	0.1006	0.0466	39
CHSWNLM	**0.0016**	**0.0016**	120

8.7. Conclusions

In this chapter, a sliding-window based online adaptation method was proposed to update a RBFNN model, previously designed offline. The proposed method is an extension of the ones proposed in [168], where the convex hull concept is employed, incorporating the current sample in the training sliding window if it lies outside the current convex hull.

Experimental results showed that the proposed method can considerably improve the performance of offline designed models for time-varying processes. In addition, it presents a performance similar to SAWNLM, and much better performance than other methods.

9. Conclusions and future work

9.1. Conclusions

This PhD was intended to address two important problems in the model design process which are very important for an HVAC MPC application: data selection and online model adaptation.

In a first step, a sequence of predictive RBFNN models were designed offline with the aim of intelligently control HVAC systems to save energy and provide thermal comfort. Since RBFNN models are data-driven models, data has a critical role in the model's performance. Inclusion of the input range boundary data samples in the training set is vital as they indicate the input-output range of system/process. To identify such samples, convex hull algorithms are applied. Due to the inefficiency of standard convex hull algorithms in terms of time and space in high dimensions (they take $O(n^{\lfloor \frac{d}{2} \rfloor})$ time and space where n and d denote the number of samples and dimensions, respectively), as the first phase of this PhD thesis a new randomized approximation convex hull algorithm in high dimensions called ApproxHull was proposed, to cope with the limitations of standard convex hull algorithms in high dimensions. ApproxHull takes $O(n^2d^3v^3 + i^3p^3)$ time where v denotes the number of convex hull vertices found, and i and p denote the number of iterations and population size, respectively.

ApproxHull was evaluated (Chapter 4) by comparing it to Quickhull [8], a known efficient standard real convex hull algorithm, and also to Wang's algorithm [68], a known approximation algorithm in high dimensions, where the Quickhull algorithm was considered a baseline for the comparisons. The simulation results obtained by applying them on a number of artificial data sets showed that all vertices identified by ApproxHull belong to the set of vertices of the real convex hull obtained by Quickhull, indicating a 100% precision, and also demonstrated that ApproxHull could identify a higher percentage of vertices of the real convex hull in comparison to the percentage identified by Wang's algorithm, indicating a higher recall. The ApproxHull's performance was also evaluated in classification and regression problems by applying it as a data selection method to create a proper training set for designing models. For classification problems, SVM models were used while for regression problems, MLP models were employed. In this evaluation, ApproxHull was compared to a common random data selection method. The simulation results showed that ApproxHull had better performance than random selection method for all classification and regression problems except for one classification and one regression problem in which both selection methods presented the same results.

As in this work the MOGA was used to design RBFNN models, the influence of applying ApproxHull in the MOGA model design framework was studied (Chapter 5) by comparing it to the random selection method. The results demonstrated that ApproxHull had better performance than random selection method in the context of the MOGA model design framework. Two strategies were followed to design the models by the MOGA. In the first, ApproxHull was employed on the whole data set to select fixed training, testing and validation data sets to fit the parameters of all models in all generations of the MOGA. In the second strategy, ApproxHull was independently applied for each single model to create distinct training, testing and validation sets. The results showed that not only, the fixed and distinct data sets strategies presented the same performance, but also that the run time of the first strategy was much smaller than that of the second one.

To demonstrate the use of ApproxHull in real applications, three case studies were introduced (Chapter 6). The two first corresponded to the estimation of the electricity consumption of a building and to the application of MPC to the HVAC system in several rooms in order to save energy and maintain thermal comfort. They demonstrated that the models designed by benefiting from ApproxHull and the MOGA framework are comparable to those obtained by other methods, but with much less complexity. In the third case study, which was intended to develop an intelligent support system for automatic diagnosis of CVAs, a set of RBFNN classification models were designed using ApproxHull and MOGA. This case study proved the capability of ApproxHull to be applied on large size data sets in high dimensions. To provide a more in-depth analysis of ApproxHull's performance, it was compared (Chapter 7) to other three methods, including random data selection, an entropy based unsupervised data selection method proposed in [13] and a hybrid method involving ApproxHull and the entropy based data selection method. Based on the experimental results, in most cases, the ApproxHull and the hybrid method were superior to the others.

In the second phase of the work (Chapter 8), a convex-hull-based sliding window online adaptation method was proposed. The goal was to update the models training data by capturing newly arrived points that are out of the known input-output range, and hence being able to adapt the models over time. The basic idea behind the method consists in comparing newly arrived points to the known convex hull obtained by ApproxHull. If the new point is outside the known convex hull (and sufficiently far) it is considered to update the model. To evaluate the proposed method, two case studies were considered so that in both cases, a

RBFNN predictive model was considered to forecast the one-step-ahead outside air temperature, where the corresponding model was gradually updated over a number of periods in one year. The results showed that the proposed method could prevent unnecessary updates while keeping the model in an acceptable level of accuracy, and also comparable to, or better than to other online adaptation methods.

9.2. Future works

Experimental results showed that the hybrid data selection method involving ApproxHull and the entropy based data selection method proposed in [13], in most cases was comparable to ApproxHull and superior to the other methods. This means that the combination of ApproxHull to other filtering methods (e.g., unsupervised methods) should be studied. For example, a clustering based method could be a proper alternative for random data selection method. Based on such studies, a data selection tool could be provided allowing the user to create training, testing and validation sets using different methods to hybridize with ApproxHull.

Regarding the ApproxHull method, one of the termination criteria is the maximum approximation distance of the furthest points to the current convex hull. The evaluation of this criterion needs finding $2 \times d$ nearest neighbors of each furthest point and then solving a quadratic optimization problem. Replacing this criterion with the heuristic applied in the new proposed online adaptation method could be studied in terms of the run time and the performance. In this work the proposed online adaptation method was evaluated based on only one time series problem (i.e., the outside air temperature model). Applying the method for a variety of case studies in different situations and comparing it with other online adaptation methods could be considered.

References

[1] A. E. Ruano, E. M. Crispim, E. Z. E. Conceicao, and M. Lucio, "Prediction of building's temperature using neural networks models," *Energy and Buildings,* vol. 38, no. 6, pp. 682-694, Jun 2006.

[2] P. M. Ferreira, A. E. Ruano, S. Silva, and E. Z. E. Conceicao, "Neural Networks based predictive control for thermal comfort and energy savings in public buildings," *Energy and Buildings,* vol. 55, pp. 238-251, 2012.

[3] A. E. Ruano *et al.*, "The IMBPC HVAC system: A complete MBPC solution for existing HVAC systems," *Energy and Buildings,* vol. 120, pp. 145-158, May 2016.

[4] P. M. Ferreira and A. E. Ruano, "Evolutionary multiobjective neural network models identification: evolving task-optimised models," *New Advances in Intelligent Signal Processing,* vol. 372, pp. 21-53, 2011 2011.

[5] V. Bayer, "Survey of algorithms for the convex hull problem," Department of Computer Science; Oregon State University, 1999.

[6] R. L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," *Inf. Process. Lett.,* vol. 1, no. 4, p. 2, 1972.

[7] R. A. Jarvis, "On the identification of the convex hull of a finite set of points in the plane," *Information Processing Letters,* vol. 2, no. 1, pp. 18-21, 1973/03/01 1973.

[8] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The Quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software,* vol. 22, no. 4, pp. 469-483, Dec 1996.

[9] F. P. Preparata and S. J. Hong, "Convex hulls of finite sets of points in two and three dimensions," *Commun. ACM,* vol. 20, no. 2, pp. 87-93, 1977.

[10] J. L. Bentley, F. P. Preparata, and M. G. Faust, "Approximation algorithms for convex hulls," *Commun. ACM,* vol. 25, no. 1, pp. 64-68, 1982.

[11] F. P. Preparata and M. I. Shamos, *Computational geometry: an introduction*. Springer-Verlag New York, Inc., 1985, p. 390.

[12] K. L. Clarkson and P. W. Shor, "Applications of random sampling in computational geometry," *Discrete & Computational Geometry,* vol. 4, no. 5, pp. 387-421, 1989 1989.

[13] P. M. Ferreira, "Entropy based unsupervised selection of data sets for improved model fitting," in *Proceedings of the 2016 International Joint Conference on Neural Networks (World Congress on Computational Intelligence)*, Vancouver, Canada, 2016: IEEE.

[14] G. Mestre *et al.*, "An intelligent weather station," *Sensors,* vol. 15, no. 12, p. 29841, 2015.

[15] A. E. Ruano *et al.*, "A neural-network based intelligent weather station," in *Intelligent Signal Processing (WISP), 2015 IEEE 9th International Symposium on*, 2015, pp. 1-6.

[16] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics,* journal article vol. 5, no. 4, pp. 115-133, 1943.

[17] S. Haykin, *Neural networks: A comprehensive foundation*, 2nd ed. Prentice Hall, 1999.

[18] A. Ruano, *Artificial neural networks*. Faro, Portugal: University of Algarve, Centre for Intelligent Systems.

[19] M. W. Gardner and S. R. Dorling, "Artificial neural networks (the multilayer perceptron) - A review of applications in the atmospheric sciences," *Atmospheric Environment,* vol. 32, no. 14-15, pp. 2627-2636, Aug 1998.

[20] D. S. Broomhead and D. Lowe, "Multivariable Functional Interpolation and Adaptive Networks," *Complex Systems,* vol. 2, pp. 321--355, 1988.

[21] O. Kaynak, "Artificial neural networks and neural information processing," in *Joint International Conference Icann/Icinip*, Istanbul, Turkey, 2003: Springer Berlin Heidelberg.

[22] A. E. Ruano, C. Cabrita, J. V. Oliveira, and L. T. Koczy, "Supervised training algorithms for B-Spline neural networks and neuro-fuzzy systems," *International Journal of Systems Science,* vol. 33, no. 8, pp. 689-711, Jun 2002.

[23] A. E. Ruano, P. J. Fleming, C. Teixeira, K. Rodriguez-Vazquez, and C. M. Fonseca, "Nonlinear identification of aircraft gas-turbine dynamics," *Neurocomputing,* vol. 55, no. 3-4, pp. 551-579, Oct 2003.

[24] A. E. Ruano, P. M. Ferreira, and C. M. Fonseca, "An overview of nonlinear identification and control with neural networks," in *Intelligent Control Systems using Computational Intelligence Techniques*, A. E. Ruano, Ed. (IEEE Control Engineering Series, no. 70): Institution of Electrical Engineers, 2005, pp. 37-87.

[25] T. Kavli and E. Weyer, "ASMOD (Adaptive Spline Modelling of Observation Data): some theoretical and experimental results," presented at the Advances in Neural Networks for Control and Systems, IEEE Colloquium on, Berlin, 1994.

[26] T. Kavli, "ASMOD - An algorithm for adaptive spline modeling of observation data," *International Journal of Control,* vol. 58, no. 4, pp. 947-967, Oct 1993.

[27] V. Vapnik, *The nature of statistical learning theory*, 2 ed. (Information Science and Statistics). Springer-Verlag New York, 2000, p. 314.

[28] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning,* vol. 20, no. 3, pp. 273-297, Sep 1995.

[29] T. Thomas Frie, N. Cristianini, and C. Campbell, "The kernel-adatron algorithm: A fast and simple learning procedure for support vector machines," presented at the Proceedings of the Fifteenth International Conference on Machine Learning, 1998.

[30] W. Sun, Ya-Xiang, *Optimization Theory and Methods*, 1 ed. Springer US, 2006.

[31] D. Tsegay and A. Mebrahtu, "Multidimensional and multi-parameter fortran-based curve fitting tools," *MEJS,* vol. 1, no. 1, p. 20, 2009.

[32] K. Levenberg, "A method for the solution of certain problems in least squares," *Quart. Applied Math.,* vol. 2, pp. 164-168, 1944.

[33] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial and Applied Mathematics,* vol. 11, no. 2, pp. 431-441, 1963.

[34] A. E. B. Ruano, D. I. Jones, and P. J. Fleming, "A new formulation of the learning problem for a neural network controller," in *30th IEEE Conference on Decision and Control*, Brighton, UK, 1991, vol. 1, pp. 865-866.

[35] Y. Wang, M. Chang, H. Chen, and M. Q. Wang, "Application of RBF neural network in intelligent fault diagnosis system," in *International Conference on Soft Computing Techniques and Engineering Application*, Icsctea, 2013, vol. 250, pp. 561-566.

[36] K. B. Kim and C. K. Kim, "Performance improvement of RBF network using ART2 algorithm and fuzzy logic system," in *Advances in Artificial Intelligence*, 2004, vol. 3339, pp. 853-860.

[37] S. Papadimitriou, S. Mavroudi, L. Vladutu, and A. Bezerianos, "Generalized radial basis function networks trained with instance based learning for data mining of symbolic data," *Applied Intelligence,* vol. 16, no. 3, pp. 223-234, May-Jun 2002.

[38] J. Moody and C. J. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Comput.,* vol. 1, no. 2, pp. 281-294, 1989.

[39] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal least-squares learning algorithm for radial basis function networks," *Ieee Transactions on Neural Networks,* vol. 2, no. 2, pp. 302-309, Mar 1991.

[40] S. Chen, P. M. Grant, and C. F. N. Cowan, "Orthogonal least-squares algorithm for training multioutput radial basis function networks," *Iee Proceedings-F Radar and Signal Processing,* vol. 139, no. 6, pp. 378-384, Dec 1992.

[41] C.-M. Huang and F.-L. Wang, "An RBF network with OLS and EPSO algorithms for real-time power dispatch," *Ieee Transactions on Power Systems,* vol. 22, no. 1, pp. 96-104, Feb 2007.

[42] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing,* vol. 4, no. 2, pp. 65-85, Jun 1994.

[43] C. M. M. d. Fonseca, "Multiobjective genetic algorithms with application to control engineering problems," PhD, Department of Automatic Control and Systems Engineering, University of Sheffield, 1995.

[44] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm and Evolutionary Computation,* vol. 1, no. 1, pp. 32-49, Mar 2011.

[45] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary Computation,* vol. 8, no. 2, pp. 173-195, 2000.

[46] C. M. Fonseca and P. J. Fleming, "An overview of evolutionary algorithms in multiobjective optimization," *Evolutionary Computation,* vol. 3, no. 1, pp. 1-16, 1995.

[47] C. M. Fonseca and P. J. Fleming, "Multiobjective optimization and multiple constraint handling with evolutionary algorithms. I. A unified formulation," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on,* vol. 28, no. 1, pp. 26-37, 1998.

[48] C. M. Fonseca and P. J. Fleming, "Multiobjective genetic algorithms made easy: Selection, sharing and mating restriction," presented at the Genetic Algorithms in Engineering Systems: Innovations and Applications, UK, 1995.

[49] C. A. Teixeira, M. G. Ruano, A. E. Ruano, and W. C. A. Pereira, "A soft-computing methodology for noninvasive time-spatial temperature estimation," *IEEE Transactions on Biomedical Engineering,* vol. 55, no. 2, pp. 572-580, Feb 2008.

[50] P. M. Ferreira, E. A. Faria, and A. E. Ruano, "Neural network models in greenhouse air temperature prediction," *Neurocomputing,* vol. 43, pp. 51-75, Mar 2002.

[51] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal,* vol. 27, pp. 379-423, 623-656, 1948.

[52] W. Li, "Mutual information functions versus correlation functions," *Journal of Statistical Physics,* journal article vol. 60, no. 5, pp. 823-837, 1990.

[53] M. B. Stojanovic, M. M. Bozic, M. M. Stankovic, and Z. P. Stajic, "A methodology for training set instance selection using mutual information in time series prediction," *Neurocomputing,* vol. 141, pp. 236-245, Oct 2 2014.

[54] R. Mena, F. Rodríguez, M. Castilla, and M. R. Arahal, "A prediction model based on neural networks for the energy consumption of a bioclimatic building," *Energy and Buildings,* vol. 82, pp. 142-155, 10// 2014.

[55] M. Rosenblatt, "Remarks on some nonparametric estimates of a density function " in *The Annals of Mathematical Statistics*, 1956, vol. 27, pp. 832-837: Institute of Mathematical Statistics.

[56] E. Parzen, "On estimation of a probability density function and mode," *The Annals of Mathematical Statistics,* vol. 33, no. 3, pp. 1065-1076, 1962.

[57] R. Moddemeijer, "On estimation of entropy and mutual information of continuous distributions," *Signal Processing,* vol. 16, no. 3, pp. 233-248, 1989/03/01 1989.

[58] Y.-I. Moon, B. Rajagopalan, and U. Lall, "Estimation of mutual information using kernel density estimators," *Physical Review E,* vol. 52, no. 3, pp. 2318-2321, 09/01/ 1995.

[59] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in Statistics: Methodology and Distribution*, S. Kotz and N. L. Johnson, Eds. New York, NY: Springer-Verlag New York, 1992, pp. 196-202.

[60] J. Demsar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research,* vol. 7, pp. 1-30, Jan 2006.

[61] O. R. Joseph, *Computational geometry in C*, 2 ed. Cambridge University Press, 1998.

[62] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational geometry: Algorithms and applications*, 3rd ed. Berlin Heidelberg: Springer-Verlag 2008, p. 386.

[63] H. Edelsbrunner, W. Brauer, G. Rozenberg, and A. Salomaa, Eds. *Algorithms in combinatorial geometry* (EATCS Monographs in Theoretical Computer Science, no. 10). Springer Berlin Heidelberg, 1987, p. 423.

[64] H. R. Khosravani, A. E. Ruano, and P. M. Ferreira, "A simple algorithm for convex hull determination in high dimensions," in *Intelligent Signal Processing (WISP), 2013 IEEE 8th International Symposium on*, 2013, pp. 109-114.

[65] B. Grünbaum, G. M. Ziegler, Ed. *Convex polytopes*, 2 ed. (Graduate Texts in Mathematics, no. 221). New York: Springer-Verlag, 2003.

[66] R. Seidel, "Constructing higher-dimensional convex hulls at logarithmic cost per face," presented at the Proceedings of the eighteenth annual ACM symposium on theory of computing, Berkeley, California, USA, 1986.

[67] B. Chazelle, "An optimal convex hull algorithm in any fixed dimension," *Discrete & Computational Geometry,* journal article vol. 10, no. 4, pp. 377-409, 1993.

[68] D. Wang, H. Qiao, B. Zhang, and M. Wang, "Online support vector machine based on convex hull vertices selection," *IEEE Transactions on Neural Networks and Learning Systems,* vol. 24, no. 4, pp. 593-609, Apr 2013.

[69] J. Arturo Olvera-Lopez, J. Ariel Carrasco-Ochoa, J. Francisco Martinez-Trinidad, and J. Kittler, "A review of instance selection methods," *Artificial Intelligence Review,* vol. 34, no. 2, pp. 133-143, Aug 2010.

[70] D. R. Wilson and T. R. Martinez, "Reduction techniques for instance-based learning algorithms," *Machine Learning,* vol. 38, no. 3, pp. 257-286, Mar 2000.

[71] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory,* vol. 13, no. 1, pp. 21-+, 1967 1967.

[72] P. Hart, "The condensed nearest neighbor rule (Corresp.)," *IEEE Transactions on Information Theory,* vol. 14, no. 3, pp. 515-516, 1968.

[73] H. Liu and H. Motoda, *Instance selection and construction for data mining*. Springer US, 2013.

[74] G. Ritter, H. Woodruff, S. Lowry, and T. Isenhour, "An algorithm for a selective nearest neighbor decision rule (Corresp.)," *IEEE Transactions on Information Theory,* vol. 21, no. 6, pp. 665-669, 1975.

[75] K. Gowda and G. Krishna, "The condensed nearest neighbor rule using the concept of mutual nearest neighborhood (Corresp.)," *IEEE Transactions on Information Theory,* vol. 25, no. 4, pp. 488-490, 1979.

[76] C.-H. Chou, B.-H. Kuo, and F. Chang, "The generalized condensed nearest neighbor rule as a data reduction method," in *18th International Conference on Pattern Recognition (ICPR 2006)*, Hong Kong, PEOPLES R CHINA, 2006, pp. 556-559, 2006.

[77] D. L. Wilson, "Asymptotic properties of nearest neighbor rules using edited data," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. SMC-2, no. 3, pp. 408-421, 1972.

[78] I. Tomek, "An experiment with the edited nearest-neighbor rule," *IEEE Transactions on Systems, Man, and Cybernetics,* vol. SMC-6, no. 6, pp. 448-452, 1976.

[79] H. Brighton and C. Mellish, "Advances in instance selection for instance-based learning algorithms," *Data Mining and Knowledge Discovery,* vol. 6, no. 2, pp. 153-172, Apr 2002.

[80] J. R. Cano, F. Herrera, and M. Lozano, "Using evolutionary algorithms as instance selection for data reduction in KDD: An experimental study," *IEEE Transactions on Evolutionary Computation,* vol. 7, no. 6, pp. 561-575, Dec 2003.

[81] P.-Y. Yin, *Modeling, analysis, and applications in metaheuristic computing: Advancements and trends*. IGI Publishing, 2012, p. 463.

[82] L. I. Kuncheva, "Editing for the k-nearest neighbors rule by a genetic algorithm," *Pattern Recognition Letters,* vol. 16, no. 8, pp. 809-814, 8// 1995.

[83] L. I. Kuncheva, "Fitness functions in editing k-NN reference set by genetic algorithms," *Pattern Recognition,* vol. 30, no. 6, pp. 1041-1049, 6// 1997.

[84] L. I. Kuncheva and J. C. Bezdek, "Nearest prototype classification: Clustering, genetic algorithms, or random search?," *IEEE Transactions on Systems Man and Cybernetics Part C-Applications and Reviews,* vol. 28, no. 1, pp. 160-164, Feb 1998.

[85] V. Cerveron and F. J. Ferri, "Another move toward the minimum consistent subset: A tabu search approach to the condensed nearest neighbor rule," *IEEE Transactions on Systems Man and Cybernetics Part B-Cybernetics,* vol. 31, no. 3, pp. 408-413, Jun 2001.

[86] H. B. Zhang and G. Y. Sun, "Optimal reference subset selection for nearest neighbor classification by tabu search," *Pattern Recognition,* vol. 35, no. 7, pp. 1481-1490, Jul 2002, Art. no. Pii s0031-3203(01)00137-6.

[87] F. Glover and M. Laguna, "Tabu Search∗," in *Handbook of Combinatorial Optimization*, M. P. Pardalos, D.-Z. Du, and L. R. Graham, Eds. New York, NY: Springer New York, 2013, pp. 3261-3362.

[88] J. A. Olvera-Lopez, J. A. Carrasco-Ochoa, and J. F. Martinez-Trinidad, "Sequential search for decremental edition," *Intelligent Data Engineering and Automated Learning Ideal 2005, Proceedings,* vol. 3578, pp. 280-285, 2005 2005.

[89] J. Kittler, "Feature selection and extraction," in *Handbook of pattern recognition and image processing*, T. Young and K. Fu, Eds. New York: Academic Press, 1986, pp. 203-217.

[90] J. A. Olvera-Lopez, J. F. Martinez-Trinidad, and J. A. Carrasco-Ochoa, "Restricted sequential floating search applied to object selection," *Machine Learning and Data Mining in Pattern Recognition, Proceedings,* vol. 4571, pp. 694-702, 2007 2007.

[91] P. Pudil, F. Ferri, J. Novovicová, and K. J, "Floating search methods for feature selection with nonmonotonic criterion functions," in *Proceedings of the 12th international conference on pattern recognition*, 1994, pp. 279-283: IEEE Computer Society Press.

[92] Y. G. Li, Z. H. Hu, Y. Z. Cai, and W. D. Zhang, "Support vector based prototype selection method for nearest neighbor rules," *Advances in Natural Computation, Pt 1, Proceedings,* vol. 3610, pp. 528-535, 2005 2005.

[93] A. Srisawat, T. Phienthrakul, and B. Kijsirikul, "SV-kNNC: An algorithm for improving the efficiency of k-nearest neighbor," *Pricai 2006: Trends in Artificial Intelligence, Proceedings,* vol. 4099, pp. 975-979, 2006 2006.

[94] R. A. Mollineda, F. J. Ferri, and E. Vidal, "An efficient prototype merging strategy for the condensed 1-NN rule through class-conditional hierarchical clustering," *Pattern Recognition,* vol. 35, no. 12, pp. 2771-2782, Dec 2002, Art. no. Pii s0031-3203(01)00208-4.

[95] C. J. Veenman and M. J. T. Reinders, "The nearest subclass classifier: A compromise between the nearest mean and nearest neighbor classifier," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 27, no. 9, pp. 1417-1429, Sep 2005.

[96] C. J. Veenman, M. J. T. Reinders, and E. Backer, "A maximum variance cluster algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 24, no. 9, pp. 1273-1280, Sep 2002.

[97] J. Arturo Olvera-López, J. Ariel Carrasco-Ochoa, and J. Francisco Martínez-Trinidad, "Object selection based on clustering and border objects," in *Computer Recognition Systems 2*, M. Kurzynski, E. Puchala, M. Wozniak, and A. Zolnierek, Eds. Berlin, Heidelberg: Springer, 2007, pp. 27-34.

[98] R. Paredes and E. Vidal, "Weighting prototypes. A new editing approach," in *15th International Conference on Pattern Recognition (ICPR-2000)*, Barcelona, Spain, 2000, pp. 25-28, 2000.

[99] J. A. Olvera-Lopez, J. A. Carrasco-Ochoa, and J. F. Martinez-Trinidad, "Prototype selection via prototype relevance," *Progress in Pattern Recognition, Image Analysis and Applications, Proceedings,* vol. 5197, pp. 153-160, 2008 2008.

[100] D. R. Wilson and T. R. Martinez, "Improved heterogeneous distance functions," *Journal of Artificial Intelligence Research,* vol. 6, pp. 1-34, 1997 1997.

[101] A. Guillen, L. J. Herrera, G. Rubio, H. Pomares, A. Lendasse, and I. Rojas, "New method for instance or prototype selection using mutual information in time series prediction," *Neurocomputing,* vol. 73, no. 10–12, pp. 2030-2038, 6// 2010.

[102] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[103] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proceedings of the Second International Conference on Genetic Algorithms*, 1987, pp. 14-21.

[104] J. C. Riquelme, J. S. Aguilar-Ruiz, and M. Toro, "Finding representative patterns with ordered projections," *Pattern Recognition,* vol. 36, no. 4, pp. 1009-1018, Apr 2003, Art. no. Pii s0031-3203(02)00119-x.

[105] B. L. Narayan, C. A. Murthy, and S. K. Pal, "Maxdiff kd-trees for data condensation," *Pattern Recognition Letters,* vol. 27, no. 3, pp. 187-200, Feb 2006.

[106] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.,* vol. 3, no. 3, pp. 209-226, 1977.

[107] P. Malosek and V. Stopjakova, "PCA data preprocessing for neural network-based detection of parametric defects in analog IC," in *IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*, 2006, pp. 131-135.

[108] A. Lopez-Chau, L. L. Garcia, J. Cervantes, X. Li, and W. Yu, "Data selection using decision tree for SVM classification," presented at the 24th International Conference on Tools with Artificial Intelligence (ICTAI), 2012.

[109] J. Nalepa and M. Kawulok, "A memetic algorithm to select training data for support vector machines," presented at the Proceedings of the 2014 conference on genetic and evolutionary computation, Vancouver, BC, Canada, 2014.

[110] A. Lopez Chau, X. Li, and W. Yu, "Large data sets classification using convex-concave hull and support vector machine," *Soft Computing,* vol. 17, no. 5, pp. 793-804, May 2013.

[111] E. W. Weisstein. (2014). *Plane*. Available: http://mathworld.wolfram.com/Plane.html
[112] E. W. Weisstein. (2014). *Point-Plane Distance*. Available: http://mathworld.wolfram.com/Point-PlaneDistance.html
[113] C. B. Barber. (1995). *Imprecision in Qhull*. Available: http://www.qhull.org/html/qh-impre.htm
[114] K. Sastry and D. Goldberg, "Search methodologies," in *Introductory Tutorials in Optimization and Decision Support Techniques*, E. K. K. Burke, Graham, Ed.: Springer US, 2005.
[115] A. Frank and A. Asuncion, "UCI machine learning repository," ed, 2013.
[116] E. Hajimani, M. G. Ruano, and A. E. Ruano, "MOGA design for neural networks based system for automatic diagnosis of cerebral vascular accidents," in *Intelligent Signal Processing (WISP), 2015 IEEE 9th International Symposium on*, 2015, pp. 1-6.
[117] M. G. Ruano, E. Hajimani, and A. E. Ruano, "A radial basis function classifier for the automatic diagnosis of cerebral vascular accidents," presented at the GMEPE/PAHCE, Madrid, Spain, 2016.
[118] C. E. Rasmussen *et al.* (1996). *Delve datasets*. Available: http://www.cs.toronto.edu/~delve/data/datasets.html
[119] A. E. Ruano *et al.*, "Improving a neural networks based HVAC predictive control approach," in *Intelligent Signal Processing (WISP), 2015 IEEE 9th International Symposium on*, 2015, pp. 1-6.
[120] H. Khosravani, M. Castilla, M. Berenguel, A. Ruano, and P. Ferreira, "A comparison of energy consumption prediction models based on neural networks of a bioclimatic building," *Energies,* vol. 9, no. 1, p. 57, 2016.
[121] A. Ruano *et al.*, "PVM-based intelligent predictive control of HVAC systems," *IFAC-PapersOnLine,* vol. 49, no. 5, pp. 371-376, 2016.
[122] L. Perez-Lombard, J. Ortiz, and C. Pout, "A review on buildings energy consumption information," *Energy and Buildings,* vol. 40, no. 3, pp. 394-398, 2008 2008.
[123] P. Nejat, F. Jomehzadeh, M. M. Taheri, M. Gohari, and M. Z. Abd. Majid, "A global review of energy consumption, CO2 emissions and policy in the residential sector (with an overview of the top ten CO2 emitting countries)," *Renewable & Sustainable Energy Reviews,* vol. 43, pp. 843-862, Mar 2015.
[124] H.-x. Zhao and F. Magoules, "A review on the prediction of building energy consumption," *Renewable & Sustainable Energy Reviews,* vol. 16, no. 6, pp. 3586-3592, Aug 2012.
[125] J. Liang and R. Du, "Model-based fault detection and diagnosis of HVAC systems using support vector machine method," *International Journal of Refrigeration-Revue Internationale Du Froid,* vol. 30, no. 6, pp. 1104-1114, Sep 2007.
[126] L. Suganthi and A. A. Samuel, "Energy models for demand forecasting-A review," *Renewable & Sustainable Energy Reviews,* vol. 16, no. 2, pp. 1223-1240, Feb 2012.
[127] F. Manzano-Agugliaro, F. G. Montoya, A. Sabio-Ortega, and A. Garcia-Cruz, "Review of bioclimatic architecture strategies for achieving thermal comfort," *Renewable & Sustainable Energy Reviews,* vol. 49, pp. 736-755, Sep 2015.
[128] C. Gallo, "Bioclimatic architecture," *Renewable Energy,* vol. 5, no. 5-8, pp. 1021-1027, Aug 1994.
[129] A. F. Tzikopoulos, M. C. Karatza, and J. A. Paravantis, "Modeling energy efficiency of bioclimatic buildings," *Energy and Buildings,* vol. 37, no. 5, pp. 529-544, May 2005.
[130] R. Albatici and F. Passerini, "Bioclimatic design of buildings considering heating requirements in Italian climatic conditions. A simplified approach," *Building and Environment,* vol. 46, no. 8, pp. 1624-1631, Aug 2011.

[131] N. L. Panwar, S. C. Kaushik, and S. Kothari, "Role of renewable energy sources in environmental protection: A review," *Renewable & Sustainable Energy Reviews,* vol. 15, no. 3, pp. 1513-1524, Apr 2011.

[132] Y. Hua, M. Oliphant, and E. Jing Hu, "Development of renewable energy in Australia and China: A comparison of policies and status," *Renewable Energy,* vol. 85, pp. 1044 - 1051, 2016.

[133] N. Scarlat, J.-F. Dallemand, F. Monforti-Ferrario, M. Banja, and V. Motola, "Renewable energy policy framework and bioenergy contribution in the European Union – An overview from national renewable energy action plans and progress reports," *Renewable and Sustainable Energy Reviews,* vol. 51, pp. 969 - 985, 2015.

[134] A. S. Ahmad *et al.*, "A review on applications of ANN and SVM for building electrical energy consumption forecasting," *Renewable & Sustainable Energy Reviews,* vol. 33, pp. 102-109, May 2014.

[135] N. Fumo, "A review on the basics of building energy estimation," *Renewable & Sustainable Energy Reviews,* vol. 31, pp. 53-60, Mar 2014.

[136] S. A. Kalogirou, "Artificial neural networks in energy applications in buildings," *International Journal of Low Carbon Technologies,* vol. 1, no. 3, p. 15, 2006.

[137] A. Foucquier, S. Robert, F. Suard, L. Stephan, and A. Jay, "State of the art in building modelling and energy performances prediction: A review," *Renewable & Sustainable Energy Reviews,* vol. 23, pp. 272-288, Jul 2013.

[138] M. S. Al-Homoud, "Computer-aided building energy analysis techniques," *Building and Environment,* vol. 36, no. 4, pp. 421-433, May 2001.

[139] S. W. Wang and X. H. Xu, "Simplified building model for transient thermal performance estimation using GA-based parameter identification," *International Journal of Thermal Sciences,* vol. 45, no. 4, pp. 419-432, Apr 2006.

[140] X. Lu, T. Lu, C. J. Kibert, and M. Viljanen, "Modeling and forecasting energy consumption for heterogeneous buildings using a physical-statistical approach," *Applied Energy,* vol. 144, pp. 261-275, Apr 15 2015.

[141] A. Lomet, F. Suard, and D. Cheze, "Statistical modeling for real domestic hot water consumption forecasting," *International Conference on Solar Heating and Cooling for Buildings and Industry, Shc 2014,* vol. 70, pp. 379-387, 2015 2015.

[142] Z. Ma, H. Li, Q. Sun, C. Wang, A. Yan, and F. Starfelt, "Statistical analysis of energy consumption patterns on the heat demand of buildings in district heating systems," *Energy and Buildings,* vol. 85, pp. 464-472, Dec 2014.

[143] N. Fumo and M. A. R. Biswas, "Regression analysis for prediction of residential energy consumption," *Renewable & Sustainable Energy Reviews,* vol. 47, pp. 332-343, Jul 2015.

[144] A. Hernandez Neto and F. A. Sanzovo Fiorelli, "Comparison between detailed model simulation and artificial neural network for forecasting building energy consumption," *Energy and Buildings,* vol. 40, no. 12, pp. 2169-2176, 2008 2008.

[145] M. Aydinalp-Koksal and V. I. Ugursal, "Comparison of neural network, conditional demand analysis, and engineering approaches for modeling end-use energy consumption in the residential sector," *Applied Energy,* vol. 85, no. 4, pp. 271-296, Apr 2008.

[146] P. M. Ferreira, A. E. Ruano, R. Pestana, and L. T. Kóczy, "Evolving RBF predictive models to forecast the Portuguese electricity consumption " presented at the 2nd IFAC International Conference on Intelligent Control Systems and Signal Processing, Turkey, 2009.

[147] K. Li, H. Su, and J. Chu, "Forecasting building energy consumption using neural networks and hybrid neuro-fuzzy system: A comparative study," *Energy and Buildings,* vol. 43, no. 10, pp. 2893-2899, Oct 2011.
[148] S. Karatasou, M. Santamouris, and V. Geros, "Modeling and predicting building's energy use with artificial neural networks: Methods and results," *Energy and Buildings,* vol. 38, no. 8, pp. 949-958, Aug 2006.
[149] F. Kaytez, M. C. Taplamacioglu, E. Cam, and F. Hardalac, "Forecasting electricity consumption: A comparison of regression analysis, neural networks and least squares support vector machines," *International Journal of Electrical Power & Energy Systems,* vol. 67, pp. 431-438, May 2015.
[150] B. Dong, C. Cao, and S. E. Lee, "Applying support vector machines to predict building energy consumption in tropical region," *Energy and Buildings,* vol. 37, no. 5, pp. 545-553, May 2005.
[151] H. C. Jung, J. S. Kim, and H. Heo, "Prediction of building energy consumption using an improved real coded genetic algorithm based least squares support vector machine approach," *Energy and Buildings,* vol. 90, pp. 76-84, Mar 1 2015.
[152] L. Suganthi, S. Iniyan, and A. A. Samuel, "Applications of fuzzy logic in renewable energy systems - A review," *Renewable & Sustainable Energy Reviews,* vol. 48, pp. 585-607, Aug 2015.
[153] L. Ciabattoni, M. Grisostomi, G. Ippoliti, and S. Longhi, "Fuzzy logic home energy consumption modeling for residential photovoltaic plant sizing in the new Italian scenario," *Energy,* vol. 74, pp. 359-367, Sep 1 2014.
[154] K. Li and H. Su, "Forecasting building energy consumption with hybrid genetic algorithm-hierarchical adaptive network-based fuzzy inference system," *Energy and Buildings,* vol. 42, no. 11, pp. 2070-2076, Nov 2010.
[155] C. Hamzacebi and H. A. Es, "Forecasting the annual electricity consumption of Turkey using an optimized grey model," *Energy,* vol. 70, pp. 165-171, Jun 1 2014.
[156] Y.-S. Lee and L.-I. Tong, "Forecasting energy consumption using a grey model improved by incorporating genetic programming," *Energy Conversion and Management,* vol. 52, no. 1, pp. 147-152, Jan 2011.
[157] J. J. Guo, J. Y. Wu, and R. Z. Wang, "A new approach to energy consumption prediction of domestic heat pump water heater based on grey system theory," *Energy and Buildings,* vol. 43, no. 6, pp. 1273-1279, Jun 2011.
[158] P. Dagnely, T. Ruette, and T. Tourwe, "Predicting hourly energy consumption. Can you beat an autoregressive model?," in *24th Annual Machine Learning Conference of Belgium and the Netherlands, Benelearn*, 2015.
[159] Estación Experimental "Las Palmerillas", C. R. d. Almería, Ed. *Datos meteorológicos. Campañas agrícolas 76/77 - 94/95*. 1997, p. 54.
[160] M. Á. Castilla, José Domingo Rodriguez, Francisco de Asis Berenguel, Manuel, *Comfort control in buildings*, 1 ed. (Advances in Industrial Control). London, England: Springer London, 2014, pp. XXIV, 237.
[161] M. R. Heras, M. J. Jimenez, M. J. San Isidro, L. F. Zarzalejo, and M. Perez, "Energetic analysis of a passive solar design, incorporated in a courtyard after refurbishment, using an innovative cover component based in a sawtooth roof concept," *Solar Energy,* vol. 78, no. 1, pp. 85-96, 2005 2005.
[162] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks,* vol. 2, no. 5, pp. 359-366, 1989 1989.
[163] A. M. Fraser and H. L. Swinney, "Independent coordinates for strange attractors from mutual information," *Physical Review A,* vol. 33, no. 2, pp. 1134-1140, Feb 1986.

[164] M. B. Kennel, R. Brown, and H. D. I. Abarbanel, "Determining embedding dimension for phase-spece reconstruction using a geometrical construction," *Physical Review A,* vol. 45, no. 6, pp. 3403-3411, Mar 15 1992.

[165] D. W. Scott and S. R. Sain, "Multidimensional density estimation," in *Handbook of Statistics: Data Mining and Data Visualization*: Elsevier, 2005.

[166] P. M. Ferreira, A. E. Ruano, and I. Ieee, "Exploiting the separability of linear and nonlinear parameters in radial basis function networks," in *Adaptive Systems for Signal Processing, Communications and Control Symposium*, 2000, pp. 321-326: IEEE.

[167] P. M. Ferreira and A. E. Ruano, "On-line sliding-window Levenberg-Marquardt methods for neural network models," in *International Symposium on Intelligent Signal Processing*, 2007, pp. 163-168: IEEE.

[168] P. M. Ferreira and A. E. Ruano, "Online sliding-window methods for process model adaptation," *IEEE Transactions on Instrumentation and Measurement,* vol. 58, pp. 3012-3020, 2009.

[169] B. Widrow and M. A. Lehr, "30 years of adaptive neural networks - perceptron, madaline and backpropagation," 1990, vol. 78, no. 9, pp. 1415-1442: IEEE.

[170] O. Nelles, *Nonlinear system identification*, 1 ed. Springer-Verlag Berlin Heidelberg, 2001, p. 786.

[171] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks," *Neural Computation,* vol. 5, no. 6, pp. 954-975, Nov 1993.

[172] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Muller, "Efficient backprop," *Neural Networks: Tricks of the Trade,* vol. 1524, pp. 9-50, 1998 1998.

[173] L. S. H. Ngia, J. Sjoberg, and M. Viberg, "Adaptive neural nets filter using a recursive Levenberg-Marquardt search direction," in *32nd Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, Ca, 1998, pp. 697-701, 1998.

[174] V. S. Asirvadam, S. F. McLoone, and G. W. Irwin, "Parallel and separable recursive Levenberg-Marquardt training algorithm," *Neural Networks for Signal Processing Xii, Proceedings,* pp. 129-138, 2002 2002.

[175] J. Nagumo and A. Noda, "A learning method for system identification," *IEEE Transactions on Automatic Control,* vol. 12, no. 3, pp. 282-287, 1967.

[176] P. C. Parks and J. Militzer, "A comparison of five algorithms for the training of CMAC memories for learning control systems," *Automatica,* vol. 28, no. 5, pp. 1027-1035, 1992.

[177] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control,* vol. 19, no. 6, pp. 716-723, 1974.

[178] H. Chen, Y. Gong, and X. Hong, "Online modeling with tunable RBF network," *IEEE Transactions on Cybernetics,* vol. 43, no. 3, pp. 935-947, Jun 2013.

[179] N.-Y. Liang, G.-B. 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, Nov 2006.

[180] J. Platt, "A resource-allocating network for function interpolation," *Unsupervised Learning: Foundations of Neural Computation,* pp. 341-353, 1999 1999.

[181] Y. W. Lu, N. Sundararajan, and P. Saratchandran, "A sequential learning scheme for function approximation using minimal radial basis function neural networks," *Neural Computation,* vol. 9, no. 2, pp. 461-478, Feb 15 1997.

[182] J.-c. Yin, Z.-j. Zou, and F. Xu, "Sequential learning radial basis function network for real-time tidal level predictions," *Ocean Engineering,* vol. 57, pp. 49-55, Jan 1 2013.

[183] H.-G. Han, Q.-l. Chen, and J.-F. Qiao, "An efficient self-organizing RBF neural network for water quality prediction," *Neural Networks,* vol. 24, no. 7, pp. 717-725, Sep 2011.

[184] G. B. Huang, P. Saratchandran, and N. Sundararajan, "An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks," *IEEE Transactions on Systems Man and Cybernetics Part B-Cybernetics,* vol. 34, no. 6, pp. 2284-2292, Dec 2004.

[185] Y. Jun and M. J. Er, "An enhanced online sequential extreme learning machine algorithm," in *Chinese Control and Decision Conference*, 2008, vol. 1-11, pp. 2902-2907.

[186] D. K. S. Tok, D.-L. Yu, C. Mathews, D.-Y. Zhao, and Q.-M. Zhu, "Adaptive structure radial basis function network model for processes with operating region migration," *Neurocomputing,* vol. 155, pp. 186-193, May 1 2015.

[187] A. Alexandridis, H. Sarimveis, and G. Bafas, "A new algorithm for online structure and parameter adaptation of RBF networks," *Neural Networks,* vol. 16, no. 7, pp. 1003-1017, Sep 2003.

[188] A. Alexandridis, "Evolving RBF neural networks for adaptive soft-sensor design," *International Journal of Neural Systems,* vol. 23, no. 6, Dec 2013, Art. no. 1350029.

[189] X. P. Lai and B. Li, "An efficient learning algorithm generating small RBF neural networks," *Neural Network World,* vol. 15, no. 6, pp. 525-533, 2005 2005.

[190] V. S. Asirvadam, S. F. McLoone, and R. Palaniappan, "Bio-signal identification using simple growing RBF-network (OLACA)," in *International Conference on Intelligent & Advanced Systems*, 2007, vol. 1-3, pp. 263-267.

[191] G. B. Huang, Q. Y. Zhu, and C. K. Siew, "Extreme learning machine: A new learning scheme of feedforward neural networks," in *IEEE International Joint Conference on Neural Networks (IJCNN)*, Budapest, HUNGARY, 2004, pp. 985-990, 2004.

[192] G. B. Huang and C. K. Siew, "Extreme learning machine: RBF network case," in *8th International Conference on Control, Automation, Robotics and Vision*, 2004, vol. 1-3, pp. 1029-1036.

[193] G. B. Huang, P. Saratchandran, and N. Sundararajan, "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," *IEEE Transactions on Neural Networks,* vol. 16, no. 1, pp. 57-67, Jan 2005.