A GENERIC FRAMEWORK FOR AUTOMATIC CONFIGURATION OF ARTIFICIAL NEURAL NETWORKS FOR DATA MODELING

Morten Gill Wollsen

The Maersk Mc-Kinney Moller Institute
The Technical Faculty
University of Southern Denmark

SDU
ENERGY
INFORMATICS

AUGUST 2017

Committee

Colophon

© Morten Gill Wollsen 2017
Typeset by the author in Utopia 12pt
using LaTeX and Ti*k*Z with the *memoir* class.
Printed by Print&Sign, SDU.

Abstract

More and more sensor data is becoming available due to the increasing digitalization of society. Especially the advent of Internet of Things and developments in Big Data contribute to this trend. Many tools exist for applying Artificial Neural Networks (ANNs) to the growing amount of data to create different system models. These tools are excellent at allowing a developer to experiment with different ANNs and configurations of such, but they require extensive expert knowledge. A user-friendly and highly automated framework or tool is currently missing and is therefore highly sought after.

The contribution of this doctoral project is the software framework AGFACAND: a framework for creating system models without the necessary expertise of ANNs. The framework is fully automatic and requires only limited configuration.

The framework contains a variety of machine learning based methods for system modelling and automatic feature selection. The variety of methods ensures that a method to match the requirements of the specific system model can be found. The selected methods are well-proven in literature and state of the art.

The inclusion of automatic feature selection ensures the machine learning methods are not encumbered by non-contributing input features and improves the computational speed and accuracy of the methods. This doctoral project presents a novel automatic feature selection algorithm, the Ranked Distinct Elitism Selection Filter, which automatically chooses a subset of the data with high correlation to the output.

The presented framework automates the choice of which machine learning method and automatic feature selection algorithm to use. Furthermore, the framework automatically configures both through the use of Bayesian Optimization, a technique that is well used in expensive and high dimensional search spaces similar to the problem addressed within this doctoral project. By using Bayesian Optimization, the framework will automatically find the optimal combination of algorithms for the given problem.

The framework has been developed through real-life applications that range from modeling of weather forecasts, greenhouses, office buildings and ventilation systems. The same applications have been revisited with the final version of the framework to emphasize its improved effectiveness on already state of the art results. Finally, the framework allows for uncomplicated export of the system models to be used in other systems for control, fault detection or similar applications.

Resumé

Voksende mængder af sensor data bliver tilgængeligt som del af den forøgede digitialisering af samfundet. Særligt indtoget af Internet of Things og udviklingen indenfor Big Data bidrager til denne trend. Der findes mange værktøjer til at bruge kunstige neurale netværk (ANN'er) på den voksende mængde af data til at konstruere forskellige system modeller. Disse værktøjer er fremragende til at lade en udvikler eksperimentere med forskellige typer af ANN'er og konfigurationen af disse, men værktøjerne kræver omfattende ekspertviden. Et brugervenligt og kraftigt automatiseret framework eller værktøj mangler.

Afhandlingen bidrager med software frameworket AGFACAND: Et framework til at konstruere system modeller uden den nødvendige ekspertise inden for ANN'er. Frameworket er fuldt automatiseret og kræver kun begrænset konfiguration. Frameworket indeholder en række af forskellige maskin læring metoder til modellering af systemer og indeholder også automatisk feature selection.

De forskellige maskin læring metoder sikrer et match der kan imødekomme kravet fra den specifikke system model. De udvalgte metoder er velafprøvede i litteraturen og state of the art.

Inkluderingen af automatisk feature selection sikrer at maskin læring metoderne ikke er belastet med ikke-bidragende input features og vil forøge udregningshastigheden og præcisionen af metoderne. Afhandlingen præsenterer også en ny automatisk feature selection algoritme, Ranked Distinct Elitism Selection Filter, som automatisk udvælger en delmængde af dataen med høj korrelation til outputtet.

Det præsenterede framework automatiserer valget af maskin læring metoden og hvilken automatisk feature selection algoritme der skal bruges. Ydermere, frameworket konfigurer automatisk begge algoritmer gennem Bayesian Optimization. Bayesian Optimization er en teknik der er gennemtestet i resourcekrævende og høj dimensionelle søge problemer ligesom problemet i afhandlingen. Ved at bruge Bayesian Optimization vil frameworket automatisk finde den bedste kombination af algoritmer til det givne problem.

Frameworket er udviklet gennem scenarier fra den virkelige verden der strækker sig fra modellering af vejrforudsigelser og drivhuse til kontorbygninger og ventilationssystemer. De samme scenarier er blevet besøgt igen med den endelige version af frameworket for at understrege den forøgede effektivitet af frameworket overfor allerede state of the art resultater. Frameworket muliggør eksport af system modellerne til brug i andre system som kontrolsystemer, fejl detektering eller lignende scenarier.

Publications

Main author publications

Wollsen, M. G., J. Hallam, and B. N. Jørgensen (2016). "Novel Automatic Filter-Class Feature Selection for Machine Learning Regression". In *Advances in Big Data: Proceedings of the 2nd INNS Conference on Big Data, October 23-25, 2016, Thessaloniki, Greece*, pp. 71–80. Springer International Publishing.

Wollsen, M. G. and B. N. Jørgensen (2015). Improved Local Weather Forecasts Using Artificial Neural Networks. In *Distributed Computing and Artificial Intelligence (DCAI), 12th International Conference*, pp. 75–86. Springer.

Wollsen, M. G., M. B. Kjærgaard, and B. N. Jørgensen (2016). Influential Factors for Accurate Load Prediction in a Demand Response Context. In *2016 IEEE Conference on Technologies for Sustainability (SusTech)*, pp. 9–13. IEEE.

Co-author publications

Kjaergaard, M., K. Arendt, A. Clausen, A. Johansen, M. Jradi, B. N. Jørgensen, P. Nellemann, F. Sangogboye, C. Veje, and M. G. Wollsen (2016). Demand Response in Commercial Buildings with an Assessable Impact on Occupant Comfort. In *2016 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE.

Tang, M. M., M. G. Wollsen, L. Aagaard, et al. (2016). Pain Monitoring and Medication Assessment in Elderly Nursing Home Residents with Dementia. *Journal of Research in Pharmacy Practice 5*(2), 126–131.

ACKNOWLEDGMENTS

I would like to thank my supervisor Professor Bo Nørregaard Jørgensen and co-supervisor Professor John Hallam for their persistent guidance, patience and support throughout this doctoral project.

I would also like to thank Rodney Martin at NASA's Ames Research Center and the team at Universities Space Research Association for allowing me to visit and do research along the scientists at NASA's Ames Research Center.

A special thanks to Augustinus Fonden, Oticon Fonden and Knud Højgaards Fond for their scholarships to support my trip to California.

I would also like to thank my co-workers at the Center for Energy Informatics for invaluable discussions, good ideas and critical reviews.

Finally, I would like to thank my family and friends for their support and comfort.

A year spent in artificial intelligence is enough to make one believe in God.

Alan Perlis

Contents

List of Figures

List of Tables

Listings

Acronyms

AGFACAND	A Generic Framework for Automatic Configuration of Artificial Neural Networks for Data Modeling
AICc	corrected Akaike Information Criteria
ANN	Artificial Neural Network
BO	Bayesian Optimization
CBR	Case-Based Reasoning
DNN	Deep Neural Network
DR	Demand Response
ELM	Extreme Learning Machine
ESN	Echo State Network
GP	Gaussian Process
GRNN	General Regression Neural Network
HDF	Hierarchical Data Format
IoT	Internet of Things
MI	Mutual Information
MLP	Multilayer Perceptron

PCA	Principal Component Analysis
RDESF	Ranked Distinct Elitism Selection Filter
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RP	Random Projection
RPROP	Resilient Backpropagation
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
SVR	Support Vector Regression

CHAPTER

1 INTRODUCTION

1.1 Motivation

With the advent of the Internet of Things (IoT) and developments in Big Data, more and more sensors are placed everywhere from buildings to vehicles to aircraft. All these sensors produce data and the possibilities of using this data are nearly endless.

Artificial Neural Networks (ANNs) are a brain-inspired method to learn patterns in historical data and applications include control schemes, fault detection or prediction of future values for time-series data. ANNs are a well-proven methodology and are often applied to the growing amount of data to create different system models (Kalogirou, 2006).

Currently, a dozen frameworks exist for applying ANNs, such as Google's TensorFlow[1], the Neural Network toolbox in MATLAB, scikit-learn in Python, Apache Spark[2] and many more. These frameworks or tools are excellent for research purposes and for experimenting with different ANNs. They are excellent at allowing a developer to experiment with different ANNs and configurations; however, the short-coming of these frameworks is that in order to use the them they require extensive expert knowledge of ANNs. Furthermore, the system models created in a framework rarely leave the framework itself because of lacking support for exporting and integrating them into other systems.

A highly automated framework or tool that requires only a data source or data set and not also expert configuration would allow ANNs to be applied in more applications and get a more wide-spread use. The automation should cover all aspects of ANNs that would otherwise require expert knowledge, such that the ANNs can be applied by a developer who does not possess the knowledge of or have the time to become an ANN specialist. The automation should choose a suitable type of ANN for the problem and automatically configure it for optimal performance. The framework should furthermore allow for system models to be exported for use in other applications such as control systems.

The ANNs of the framework should solve regression problems where the output variable represent the values of a continous variable (Bishop, 1995). However, it will be possible to use the framework for classification by manually converting the continous output values to a class label. All problems of this doctoral project are regression problems.

[1]http://www.tensorflow.org

[2]http://spark.apache.org

1.2 Research challenges

The lack of an automatic and user-friendly framework or tool for applying ANNs on data is an interesting research problem — because in order to automate the framework, a combination of several individual technologies is required. These technologies include ANNs for the regression part, feature selection to select relevant input features and model selection to configure the ANN. The framework should also be open to addition of new machine learning and feature selection algorithms and allow the resulting system models to be exported.

Based on the problems above, this doctoral research project considers the problem of creating a framework that allows automated creation of ANN-based system models without the need of extensive ANN knowledge. More specifically, this doctoral research project addresses the following question:

> To what extent can a user-friendly framework or tool be constructed for applying ANNs to the rapidly growing amount of sensor data with a high degree of automation that does not require extensive expertise knowledge within ANNs?

1.3 Evaluation Criteria

The answer to the research question is evaluated against the following criteria.

1. **The framework must enable integration of many types of ANNs.** The framework should not be limited to a single type of ANN. Different types of ANNs have different benefits, and the framework should enable multiple ANNs to be implemented in the framework. The framework should also enable future addition of further types of ANNs.

2. **The framework must automatically configure the ANNs for the best possible performance.** Configuration of a ANN is one of the most important factors determining its performance. The framework should automatically configure the ANN for optimal performance regardless of which type of ANN is being used.

3. **The framework must support multiple time resolutions (sampling rates) in the data**. Sensor data can have different time resolutions (5-minute intervals, 15-minute interval, 1-hour intervals etc) and the framework should support any time resolution in the data as well as different time resolutions in the same data set.

4. **The framework must support varying prediction horizons.** The framework should support time series data as well as non-time series data. In the case of time series data, the framework should support a prediction horizon of multiple steps. This will allow the framework to produce models that can predict short-term as well as long-term. For non-time series data a varying prediction horizon is irrelevant.

5. **The framework must be able to export the trained model for use in other systems.** In order for the trained system model to be useful outside the framework itself, the model should be exportable and integrable. Although some standards exist, such as the FMI format used by MATLAB and other modeling software (Thiele and Henriksson, 2011), all functionality in the framework would have to be exported as C-functions in order to follow these standards. A custom format gives more freedom and an FMI wrapper could always be created.

6. **The framework must be usable by developers without ANN expertise.** This criteria separates this framework from the tools mentioned in the first section. Only if the framework can be operated by users without expert knowledge can the framework be called user friendly.

1.4 Research Method

This doctoral project uses the constructive research approach. The constructive research approach deduces theoretical contributions from applying constructs on practical and relevant research problems. The approach comprises the following steps (Kasanen et al., 1993):

- Selection of a practical and relevant research problem.
- Literature study to achieve a general and comprehensive understanding of the topic.
- Construction of models or theories that address the challenge of the research problem.
- Demonstration of the models or theories on the research problem to demonstrate that they work.
- Deduce theoretical contributions from the solution to the research problem.
- Examination of the scope of the solution to identify consequences and limitations of the solution.

The constructive research approach has been applied iteratively throughout the different sections of this doctoral project.

1.5 Contributions

A summarized list of contributions of this doctoral research project:

- An automatic feature selection algorithm of the filter-class named Ranked Distinct Elitism Selection Filter.
- A user-friendly framework for applying ANNs on data modeling problems with automatic configuration.

- Definition of interfaces to allow for further expansion of the framework with new algorithms.
- Experimental results from applying standard ANNs on various real-life applications.
- Experimental results from applying the presented A Generic Framework for Automatic Configuration of Artificial Neural Networks for Data Modeling (AGFACAND) on the same real-life applications.

1.6 Thesis Organization

The thesis is organized similarly to the framework which is presented in the following chapter. Following is a chapter on background information that will outline the contents of the framework and will also contain a literature review for the individual parts of the framework. Chapters 3 to 5 present the theoretical underpinnings of the individual parts of the framework. Chapter 6 will cover the framework design, the Java specific implementation of the framework as well as a how-to guide on extending the framework with new algorithms. Chapter 7 will be a step-by-step tutorial on applying the framework to a real-life scenario. Chapter 8 will review and revisit the real-life applications from which the framework has been built. Chapter 9 evaluate the framework with respect to the criteria set forth in this chapter. Finally, chapters 10 and 11 finalize this thesis with a discussion on future research tasks and a conclusion.

CHAPTER

2 BACKGROUND

This chapter introduces the conceptual workflow of applying ANNs to a data modeling problem commonly used in the state of art literature. The presentation of state of the art will follow the structure of this workflow and systematically present state of the art for each part of the workflow. A section on loading data and data assumptions will follow.

The conceptual workflow of applying ANNs to a data modeling problem is depicted in figure 2.1. The first step is to load the data and do any preprocessing required for the modeling, including normalization (Bishop, 1995). Once the data is loaded a feature selection step might be required in order to remove irrelevant features from the data set (Whiteson et al., 2005). After the feature selection, the actual system modeling can be executed. System modeling using ANNs is a regression problem where the model is defined by the relationship between input and output data. In order to select the optimal type of ANN or to optimize the already selected ANN the process of model selection is applied (Bishop, 2006). Model selection is an iterative process that will repeat parts of the workflow.

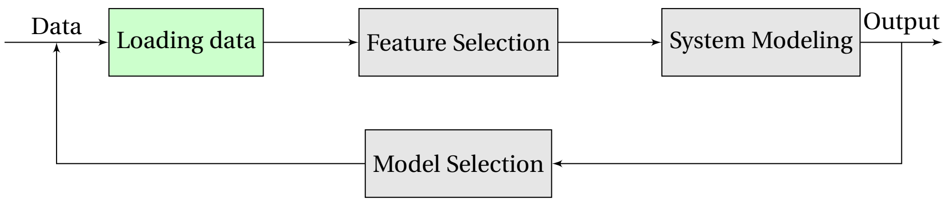


Figure 2.1: The conceptual workflow of system modeling using ANNs.

The workflow of figure 2.1 will be used throughout this thesis to emphasize which part of AGFACAND is currently being covered. Loading of data and data assumptions will be covered at the end of this chapter, which is why it is highlighted in figure 2.1. A darker shade of green means that the topic has already been covered.

2.1 State of the art

This section will describe state of the art methodology of the individual parts of the conceptual workflow as presented in figure 2.1. The order will be by importance in relation to the functionality. This means that the system modeling part will be covered before the feature selection even though feature selection is performed before the system modeling. This is because the system modeling can be applied without feature selection, but without a functioning system modeling part, the output of the feature selection process will not be used.

System Modeling

Modeling in AGFACAND will be performed using ANNs. ANNs are a data-driven method that learn relationships from data (Zhang and He, 2007). The theoretical background for all types of ANN in AGFACAND will be given in chapter 3. The selected literature presents a wide range of different types of ANNs, however, the configuration of the ANN is performed manually. Both the choice of ANN type and the configuration of the chosen ANN will be performed using model selection in AGFACAND.

Powell et al. (2014) applies a NARX-network to the problem of modeling heating, cooling and electrical load forecasting with a 24 hour prediction horizon. A NARX-network is a Multilayer Perceptron (MLP) (Rosenblatt, 1961) with delayed input and delayed output as input (Chen et al., 1990). The delayed inputs makes the NARX-network a non-linear version of an autoregressive moving average (ARMA) model (Whitle, 1951) which have shown in literature to be well suited for time-series forecasting. The non-linear version will allow the modeling of more complex relations. Powell et al. (2014) also adds time variables (month, hour and day) as an input to the modeling. Both the size of the delay as well as the number of hidden nodes in the ANN are chosen manually and thus time consuming as every possible combination needs to be statistically evaluated. The delaying of data, MLP and addition of time variables are all supported in AGFACAND.

Maqsood et al. (2004) presents a neural network ensemble of multiple types of ANNs for weather forecasting. The networks included are MLP, Elman, Radial Basis Function network and Hopfield model. Once again the selection of the number of hidden nodes is performed manually. The chosen numbers are based on a compromise of convergance rate and computational speed. Based on the work by Maqsood et al. (2004), the Elman network as well as neural network ensembles are also supported by AGFACAND.

Şenkal (2010) presents a General Regression Neural Network (GRNN) (Specht, 1991) for the modeling of solar radiation with excellent results. GRNN is significantly different from MLP and does not require any configuration. Based on the work by Şenkal (2010), GRNNs are supported by AGFACAND to get a variety of models to choose from.

Wan et al. (2014) presents the use of an Extreme Learning Machine (ELM) (Huang et al., 2006) for prediction of wind power generation. ELM is identical in structure to an MLP but applies a random initialization of weights and only modifies the output weights on training. A manual cross-validation approach is applied in order to select the number of nodes in the hidden layer. ELM has unique properties and for this reason, ELMs are supported in AGFACAND.

Morando et al. (2014) performs forecasting on remaining lifetime for fuel cells using a Echo State Network (ESN) (Jaeger, 2001). The work performs a manual parameter search to select the number of nodes in the hidden layer. ESN is a recurrent neural network like the Elman network. A recurrent neural network has a circular connection to itself which gives the network information on the previous state and thereby a sense of memory or time understanding. For this reason both Elman and ESNs are supported in AGFACAND.

Jain et al. (2014) presents the prediction of energy consumption using Support Vector Regression (SVR) (Cortes and Vapnik, 1995). SVR is unique because the learning of the network is a convex problem, which means that the solution found is the global minimum. This is a unique feature, which is why SVR is supported in AGFACAND.

Feature Selection

The choice of a feature selection algorithm in AGFACAND is also handled by the model selection. This means that multiple feature selection algorithms need to be included in AGFACAND. Robert May and Maier (2011) present a comprehensive overview of the different types of feature selection algorithm. This section will cover more exclusive or exotic feature selection algorithms that have the potential to be supported in AGFACAND. The feature selection part of AGFACAND will be covered in chapter 4.

Estévez et al. (2009) presents the Normalized Mutual Information feature selection algorithm. The standard Mutual Information requires the selection of a threshold but, through a greedy selection, the Normalized Mutual Information has removed this requirement. Estévez et al. (2009) furthermore combine the algorithm with a genetic algorithm to form a hybrid algorithm. A drawback of the approach is that although the algorithm does not require a selection of a threshold, it does require the maximum number of selected features to be specified.

Song et al. (2013) present a feature selection algorithm, FAST, that performs an unsupervised clustering of the data. The best representative features from each cluster form the feature subset. FAST is compared with other feature selection algorithms applied to a classification problem and obtained best proportion of selected features, best runtime and best classification accuracy for 3 data sets and second best for a 4th data set. Although FAST is used for classification it might be relevant for regression problems as well.

Xue et al. (2013) presents a multi-objective particle swarm optimization algorithm for feature selection. The algorithm generates a Pareto front of non-dominated feature subsets. The algorithm is compared with conventional feature selection algorithms as well as other evolutionary multi-objective algorithms. The algorithm outperforms all the other algorithms in classification performance. The algorithm even produces better results with fewer selected features and a faster computation time. The algorithm requires parameters to be configured for the evolutationary part. Once again, this algorithm was developed on a classification problem, but could be relevant for regression problems as well if the configuration could be automated.

Model Selection

Model selection should handle the choice of both feature selection algorithm, type of ANN for the modeling and the configuration of the chosen ANN. The model selection of AGFACAND will be covered in chapter 5. This section will cover different options for applying model selection in AGFACAND.

Oneto et al. (2015) present two strategies for model selection: Fully-empirical Algorithmic Stability (FAS) and Bag of Little Bootstraps (BLB). FAS works by measuring the ability of an algorithm to select similar models which ensures that the algorithm is learning without overfitting. BLB performs the bootstrap procedure which means that the generalization performance of the possible model configuration is estimated with a smaller data set. The main focus of the article is on the deployment of the algorithms on GPUs and the possible time savings compared to CPUs. The author states however, that both model selection algorithms scale sub-linearly with respect to the data, and therefore are well suited for Big Data.

Gibert et al. (2010) suggest using a Case-Based Reasoning (CBR) system for choosing which model to apply to a problem. The data set of the problem is analyzed and compared with existing cases. The model of the most similar case is chosen and information from applying that model on the new data set is added to the system. Using a CBR means that the model selection is based purely on meta-data from the data set. A CBR system requires a large database of data sets in order to properly recommend an algorithm to use.

Aras and Kocakoç (2016) present a methodology that combines feature selection and model selection. The Input, Hidden and Trial Selection (IHTS) strategy initially classifies the potential models in groups based on their input units. The group with the smallest mid-mean is divided into smaller groups based on the number of hidden units. The models from this subgroup is then evaluated and the results are saved in a matrix. The Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) (Hwang and Yoon, 1981) is then applied to the matrix to find the final model. TOPSIS is a multi-criteria decision making method. Although IHTS combines feature selection and model selection, there is not enough applications of IHTS in literature to support using IHTS in AGFACAND.

Snoek et al. (2012) present the application of Bayesian Optimization (BO) to machine learning algorithms. BO works using Bayes' theorem and applies a Gaussian Process (GP) to the known data. Based on the GP, one can estimate which model and configuration of said model will be best. Because of the estimation, a true optimal model and configuration can not be guaranteed. However, the number of evaluations in BO is limited which will decrease the computational time and BO have been applied as a model technique in a large body of literature. For this reason, BO has been chosen as the model selection algorithm to use in AGFACAND.

2.2 Loading data

The data used to create the models in AGFACAND will come from many different sources and in different formats. The data can be both human created values, sensor readings, actuator positions, status variables or time based variables. Unfortunately, sensor data is never perfect and often suffers from missing or overflowing values. AGFACAND assumes that the data is perfect, which means that it is up to the user to remove any overflowing or similarly faulty values. Missing values will be handled by the data loading mechanism in AGFACAND. Dozens of methods for dealing with missing data exists and the topic is a research field itself, but simply ignoring them is a fast and unbiased approach (Little and Rubin, 2014; Jordanov and Petrov, 2014). The method applied is listwise deletion in which the entire data sample is deleted if a parameter within the sample has a missing value. This will reduce the amount of data that can be used for the training, but ensures that all samples are complete which is considered more important.

Another matter to handle in terms of data is time series. Time-variant systems are systems where the variables change over time, such systems will respond differently to changes at different times. Examples as such could be an office where the temperature in the office reacts differently to airconditioning based on the outdoor temperature. Data recorded in time-variant systems are time series where the data must be in a specific sequence in order to keep the information. ANN structures such as Time-Delay Neural Networks (TDNN) or Input-Delay Neural Network (IDNN) (Waibel et al., 1989) show how time series data can be handled by a stateless method such as the MLP by delaying the input data. Similarly, delaying the output and providing it as an input can also give a MLP the abilities to model time series problems (Powell et al., 2014). The latter is through a ANN structure called nonlinear autoregressive model with exogenous inputs (NARX) (Chen et al., 1990). These two options are both supported by AGFACAND through a delay parameter. This parameter will delay both the input and output for the given number of samples. The configuration of the parameter is performed by the automatic model selection. A completely different approach is to use an ANN structure that has a built in memory such as recurrent neural networks like Elman and ESN which will also be supported by AGFACAND. By supporting both approaches, all models in AGFACAND can handle both time series data as well as data from time-invariant systems.

When removing entire samples using listwise deletion, gaps will occur which needs to be handled before delaying the data. This is performed by splitting the data into separate sections and then delaying each section separately.

2.3 Summary

This chapter presented the conceptual workflow of applying ANNs on data modeling problems and reviewed state of the art for every part of the workflow. Model selection will release the developer from having the required knowledge of ANNs, but for the framework to be user-friendly, the model selection needs to be automated. When the model selection process is automated, the feature selection process also needs to be automated in order to have an automated and user-friendly framework. Automating both the model selection and the feature selection processes sets forth requirements in terms of algorithms and implementation that AGFACAND needs to handle. Finally, how AGFACAND deals with faulty data and how AGFACAND handles time series were presented.

CHAPTER

3

DYNAMIC SYSTEM MODELING USING ARTIFICIAL NEURAL NETWORKS

Even before Kurt Hornik in 1991 (Hornik, 1991) showed that the multilayer feedforward architecture of Artificial Neural Networks (ANNs) is a universal approximator, ANNs were used as a tool for modeling of dynamic systems (Carpenter, 1989). Universal approximation means that the ANN can approximate any function, allowing the ANN to model the underlying relationship in the data gathered from the dynamic system.

This chapter describes the body of knowledge of dynamic system modelling using ANNs. The chapter starts with background information on how ANNs work and information on different types of ANNs. Following that, it gives an overview of the types of ANNs included in AGFACAND and the chapter is summarized.

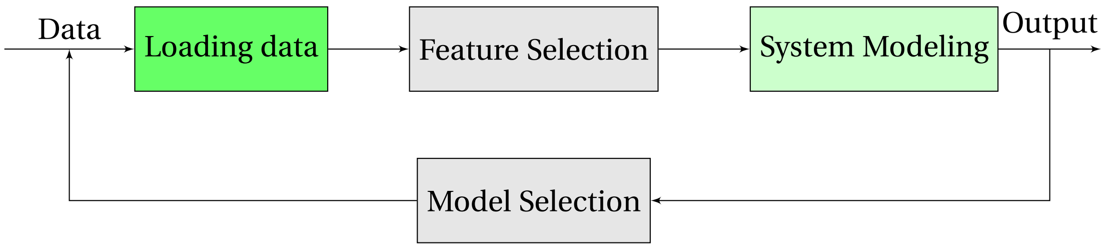


Figure 3.1: The System Modeling part of the conceptual workflow is the focus of this chapter.

3.1 Introduction to Artificial Neural Networks

An ANN is made up of a large collection of “artificial neurons”, each of which is a mathematical model of the brain’s neurons first presented by McCullogh and Pitts in 1943 (Bishop, 1995). Just like in the brain, these artificial neurons are connected with many other neurons which allows signals to travel between them. The inputs to the artificial neurons are combined linearly by summation and transformed through a typically non-linear activation function. This transformation allows the ANN to model complex problems.

A schematical drawing of an artificial neuron can be seen in figure 3.2. The out-

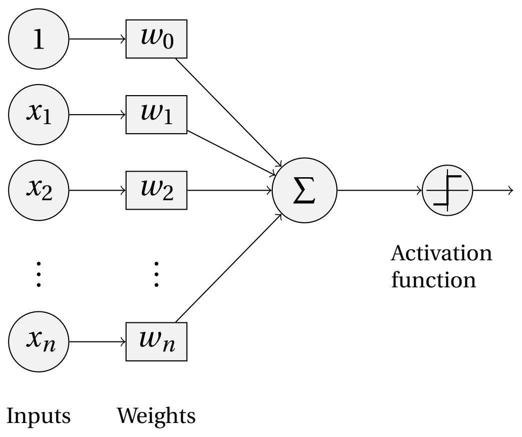


Figure 3.2: A single artificial neuron with inputs x and input weights w. w_0 is used for the bias node that always has a value of 1.

put of an artificial neuron is expressed mathematically as:

$$y = \phi\left(\sum_{i=0}^{n} w_i x_i\right) \tag{3.1}$$

where ϕ is the activation function. A common choice for an activation function is a sigmoid (Bishop, 1995).

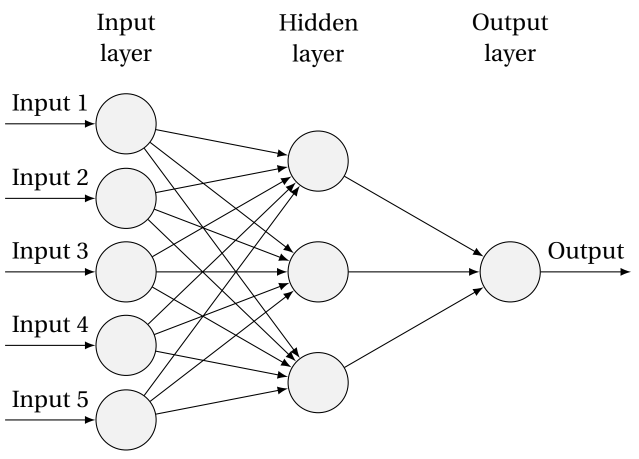


Figure 3.3: A feedforward Artificial Neural Network or a MLP

A schematical drawing of a complete feedforward ANN can be seen in figure 3.3. Note however that the neurons in the input layer do not include a bias nor any activation function. Feedforward means that the signal only goes forward (i.e. from left to right in figure 3.3), as compared to recurrent neural networks, which will be reviewed shortly. The depicted ANN takes 5 inputs, contains 3 nodes in its hidden layer and produces a single output. The feedforward ANN is also called a Multilayer Perceptron (MLP).

In order to perform regression with an ANN the network needs to be trained. This chapter will not go into great depths with the training algorithms, because the different included network types use different training algorithms. Rumelhart et al. (1988) presented the backpropagation algorithm which is commonly used in training ANNs. When an input is fed to the ANN it is propagated forwards using equation 3.1 until it reaches the output layer. The output of the ANN is then compared to the desired output and an error value is calculated. The error value is then propagated backwards starting from the output layer, and the weights between neurons are updated according to their contribution to the error. The training is performed on a data set containing multiple samples. The error is the total error summarized from every sample in the data set. This is iterated multiple times, meaning that the data set is run through the network multiple times. Stopping criteria include having achieved a maximum number of iterations or having achieved a sufficiently low error.

The mentioned error is called the training error and is a measure of how well the ANN performs on the training data. Generalization performance is the crucial measurement for AGFACAND, and therefore the data is split into two sections: training data and test data. By running the trained ANN with the test data, the generalization performance can be measured as the test data is unseen data (Zhang et al., 1998). If the training error is low and the test error is significantly higher, overfitting is present (Zhang et al., 1998). Overfitting means that the ANN has been fitted exactly to the training data and any noise that may be present instead of to the underlying relationship between the inputs and the outputs. Measures for avoiding overfitting include regularization, pruning or hyperparameter optimization (Zhang et al., 1998). The latter is included in AGFACAND and is reviewed in chapter 6.

Other Artificial Neural Network types

Besides the feedforward neural network, other types of ANNs exist. A brief introduction will be given to these other types of ANNs with explanations of how they are different from the feedforward neural network.

Recurrent Neural Network (RNN)

RNNs are characterized by a circular connection of the neurons in the network. The hidden layer or the output layer has connections to the hidden layer which gives RNNs temporal information by knowing the state of the hidden layer in the previous time step. Consequently, the training process is more complicated with Backpropagation Through Time (BPTT) as a commonly used algorithm (Boden, 2002).

Deep Learning

Deep Learning refers to very large ANNs with many neurons and multiple hidden layers (Schmidhuber, 2015). Deep Learning is in literature mainly used for classification and therefore Deep Learning methods are not in the scope of this doctoral research project.

Neural Network Ensembles

Neural Network Ensembles are defined as sets of trained networks whose output is combined with an appropriate strategy (Hansen and Salamon, 1990). Ensembles have shown, through an extensive body of literature, that a combination of ANNs outperforms a single optimized ANN (Kourentzes et al., 2014). The obvious choice for a combining strategy, or operator, is the mean. Kourentzes et al. (2014) present a comparison of the mean, the median and their novel mode operator. The mode operator is defined as the most frequent value in a set of data which is calculated by a kernel density estimation. The mode operator is insensitive to outliers (Kourentzes et al., 2014).

Network ensembles with the mode operator is an attractive approach to overcome randomness and intensive tuning of a single ANN. For this reason, ensembles are included in AGFACAND and is also used by default for any model. This means that when a model is used in AGFACAND, it is in fact an ensemble of 50 individual models of the specific type. This number is selected based on the results in (Kourentzes et al., 2014).

3.2 Chosen models

The types of ANNs that are included in AGFACAND are chosen based on variety and proven reliability in literature. From this point forward, a regression method will be called a model. This will allow the regression to be performed by other means than ANNs as well as match with literature and the terminology in this area.

By having variety in the included models, model selection, which will be discussed in chapter 6, has the best possibility of finding an optimal model. Furthermore, simpler problems characterized by linear relations between input and output might achieve better results using linear regression methods compared to complex methods such as ANNs. Many models support multiple outputs, but AGFACAND only supports a single output. This simplifies the implementation of a model and configuration of a data set. AGFACAND can still be used to produce multiple outputs by creating two parallel models. However, multi-output models can learn the relationship between multiple outputs in case of strong correlation and might produce better models (Borchani et al., 2015). On the other hand, if no correlation exist, the model might need to have a larger size in order to fully grasp the relationships for all outputs (Borchani et al., 2015).

The initial models chosen to be included in AGFACAND are:

1. Multilayer Perceptron (MLP)
2. ADALINE network
3. Echo State Network (ESN)
4. Elman network
5. Extreme Learning Machine (ELM)
6. General Regression Neural Network (GRNN)
7. Ordinary Least Squares Linear Regression
8. Support Vector Regression (SVR)

Table 3.1: The desired characteristics and the chosen models in AGFACAND.

Characteristic	Model							
	1	2	3	4	5	6	7	8
Linear problems	X	X	X	X	X	X	X	X
Non-linear problems	X		X	X	X	X		X
No configuration		X				X	X	X
Global optimum							X	X
Recurrent			X	X				

These models have characteristics that separate them which is visualized in table 3.1

Linear and non-linear problems in table 3.1 refer to the models' ability to solve these type of problems. No configuration means that the model does not have a size or similar parameter that needs to be configured. Global optimum means that the model will always return the same solution regardless of how many times it is trained. Finally, recurrent means that the structure of the model is recurrent which gives the model memory abilities. Table 3.1 confirms that the chosen models in AGFACAND provide a variety in the different characteristics that the models provide.

3.3 Configuration of models

Each of the included models will be briefly introduced, including how the model is trained and any configuration necessary for using the model. The number of neurons in the hidden layer(s) will be configured by the model selection explained in chapter 6. All models but the Ordinary Least Squares Linear Regression apply data normalization to compress the values to fit within the working interval of the activation function. The tanh activiation function will be used for all ANNs. The tanh function operates within an interval from −1.0 to 1.0. The normalization will put every value in the interval from −0.9 to 0.9, which will allow unseen data to go beyond the ranges of the training data without being saturated by the activation function.

Multilayer Perceptron

Multilayer Perceptrons were explained at the beginning of this chapter. The output of the (single hidden layer) MLP can be stated mathematically as:

$$\mathbf{Y} = \mathbf{W}_{\text{out}}\phi\left(\mathbf{W}_{\text{in}}\mathbf{x}\right) \tag{3.2}$$

where $\mathbf{W}_{\text{in}}$ is the randomly initialised weight matrix between the input layer and the hidden layer, $\mathbf{W}_{\text{out}}$ is the randomly initialised weight matrix between the hidden layer and the output layer and $\mathbf{x}$ and $\mathbf{Y}$ are inputs and outputs respectively. The MLP is trained using the Resilient Backpropagation (RPROP) algorithm (Riedmiller and Braun, 1993).

As MLPs show very good performance in a large body of literature, it has become a de facto standard in ANN tools. It is therefore natural to include in AGFACAND.

ADALINE network

The ADAptive LINear Element (Widrow and Lehr, 1990) is an early type of ANN which was very useful before the backpropagation algorithm was introduced. The ADALINE network will work well for simpler problems such as linear relations. The training is performed iteratively with a Stochastic Gradient Descent (SGD) (Robbins and Monro, 1951). The learning rate used is 0.01.

Echo State Network

Echo State Networks are recurrent neural networks where only the weights between the hidden layer (or the "reservoir") and the outputs are trained (Lukoševičius et al., 2012). This is possible as long as certain properties are present. This property is called the "echo state" property: the reservoir state should not depend on the initial conditions that were before the input (Lukoševičius, 2012). Figure 3.4 presents a graphical overview of how ESNs function.

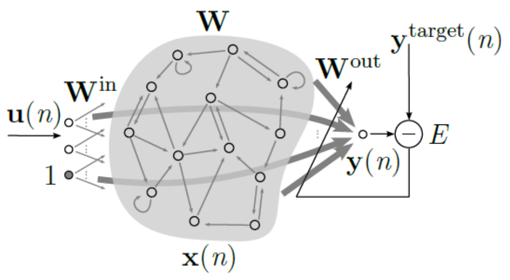


Figure 3.4: An Echo State Network (Lukoševičius, 2012)

The output weights are trained with linear least squares which in practice is performed through the calculation of the Moore-Penrose pseudoinverse:

$$\mathbf{W}^{\text{out}} = \mathbf{Y}^{\text{target}}\mathbf{X}^{+} \tag{3.3}$$

where $\mathbf{X}^{+}$ is the Moore-Penrose pseudoinverse of the state matrix. The pseudoinverse is calculated using the Apache Commons Math Library[1] that employs a Singular Value Decomposition. The internal weights are divided by the maximum eigenvalue to obtain a spectral radius below 1.0 which ensures the "echo state" property in most cases (Lukoševičius, 2012). A noise level of 0.0001 is added to avoid matrix singularities as well as overfitting. The ESN uses the leaky neuron with a leaking rate of 0.5. The leaky neuron is defined as $x(t) = (1-\alpha)x(t-1) + \mathcal{F}$ where α is the leaking rate and $\mathcal{F}$ is the function that returns the new value for $x(t)$. By using the leaky neuron, the memory abilities of the ESN can be increased (Lukoševičius, 2012) by including the previous value for x. For further information on the parameters, the reader is refered to Lukoševičius (2012).

Although the time series issue has been neutralized by adding delayed data as described in chapter 2, having a recurrent neural network can still prove to be advantageous and also adds a variety to AGFACAND which improves the model selection described in chapter 6.

[1]http://commons.apache.org/proper/commons-math/

Elman network

An Elman network is a simple recurrent neural network with a second set of hidden neurons that only are connected to the original hidden neurons. This second set is called the context layer and the weights to and from this layer are fixed at 1 and are not modified by the training process (Cruse, 2006). The context layer will have the value of the hidden layer at the previous time step, thereby giving the Elman network a short term memory. For more information, the reader is refered to Boden (2002) and Cruse (2006). The Elman network is trained using the RPROP algorithm (Riedmiller and Braun, 1993) and a greedy max iteration strategy. The greedy max iterations continuously saves the network with the lowest error until a maximum number of iterations is performed. The error might increase in some iterations and never achieve a new lowest error, which is why the best network is saved. The training runs for 500 iterations.

Extreme Learning Machine

Extreme Learning Machines are feedforward neural networks like MLPs. However, only the output weights are trained similarly to the ESN. The output of the ELM is:

$$\mathbf{Y} = \mathbf{W}_{\text{out}}\phi(\mathbf{W}_{\text{in}}x) \tag{3.4}$$

where $\mathbf{W}_{\text{in}}$ is the weight matrix between the input layer and the hidden layer, $\mathbf{W}_{\text{out}}$ is the weight matrix between the hidden layer and the output layer and x and $\mathbf{Y}$ are inputs and outputs respectively. Both weight matrices are randomly populated. Although the output weights could be calculated using the Moore-Penrose pseudoinverse as with the ESN, the author achieved results significantly faster using the SGD method with only insignificant precision loss. The SGD uses an initial learning rate of $\frac{1}{n}$ where n is the number of samples in the data set. For every iteration the learning rate is reduced by 10 % to ensure convergence. The training runs for 50 iterations. For further information on ELMs, the reader is refered to Huang et al. (2006).

General Regression Neural Network

The General Regression Neural Network is a feedforward neural network with two hidden layers and a variation of the popular radial basis neural network. The first hidden layer contains neurons called pattern units that each are dedicated to one pattern or cluster. The second layer is a summation layer that sums the distances between a sample and the clusters in the first hidden layer. For further information on GRNNs the reader is referred to Specht (1991).

The GRNN uses a Gaussian kernel and applies the Brent method for global minimum search (Brent, 1973) of the smoothing factor σ. The search space is between 0.0001 and 10.0 and runs for 20 iterations.

GRNNs are often used for function approximation or regression and will converge to the underlying regression surface. Having GRNNs in AGFACAND contributes to the variety of neural network types.

Ordinary Least Squares Multiple Linear Regression

Multiple linear regression models the relationship between a dependent variable y and independent variables X in the form $\mathbf{y} = \mathbf{X}\beta + \epsilon$. The regression is performed using the Ordinary Least Squares algorithm that minimizes the sum of squared residuals. The QR decomposition is used to estimate the parameters of the regression. The Ordinary Least Squares Multiple Linear Regression is implemented using the Apache Commons Math library [2].

The reason a simple linear regression is included in AGFACAND is due to both having a variety of models but also that a linear model might result in a better generalization instead of a complex fitted model, as illustrated in figure 3.5. Although the complex fit is a near-perfect fit, the linear fit can be expected to generalize better. If the two functions were used to extrapolate the data beyond the fitted data, the linear fitting would make better predictions. Generalization is desirable as it will increase the performance on unseen data.

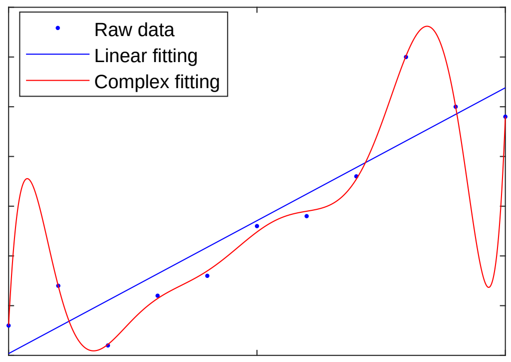


Figure 3.5: The same data fitted with a complex function and a linear function. The linear function is a better generalization of the data.

Support Vector Regression

Support Vector Regression is based on Support Vector Machines (SVMs) and the names are used interchangeably when SVMs are applied to regression. SVR is a kernel-based algorithm with a unique property that the determination of model parameters corresponds to a convex optimization problem, meaning that any local solution is also the global minimum (Bishop, 2006). This also makes SVR deterministic and the use of network ensembles is irrelevant for SVRs. The type of SVR implemented is the ϵ-SVR with a Radial Basis Function kernel. The training process performs a cross-validated grid-search of the two parameters C and ϵ.

The global minimum and deterministic properties of SVRs are desirable. SVRs also have a large body of literature.

3.4 Summary

This chapter briefly introduced artificial neurons, ANNs and how ANNs are used to perform regression. Afterwards, this chapter presented the models that are included

[2]http://commons.apache.org/proper/commons-math/

in AGFACAND and briefly explained their inner workings and how they differentiate from each other.

CHAPTER

4 Automatic Feature Selection

Chapter 3 covered the different machine learning algorithms (models) used to perform regression in AGFACAND. Implementing a successful machine learning algorithm requires choosing a representation of the solution, selecting relevant input features, or input variables, and setting parameters associated with the model (Whiteson et al., 2005).

Selecting the relevant input features, or *feature selection*, is the process of determining which subset of the combined available input data should be included to give the best performance. Feature selection is a critical task because excluding important input features means the model will not be able to correctly reflect its respective system. On the other hand, including unnecessary features complicates and slows down the training. Any input that is added increases the search space by at least one dimension (Whiteson et al., 2005). Feature selection can be performed by humans experts with the necessary application domain expertise. However, in some cases these experts do not exist or the work itself can be expensive and time consuming (Whiteson et al., 2005). A reduced feature set for the model also reduces training time and overfitting (Guyon and Elisseeff, 2003). By automating the feature selection process, the time and expertise required is reduced and the feature selection process can be integrated into an automated system (Whiteson et al., 2005) like AGFACAND.

This chapter describes the body of knowledge of automatic feature selection. The chapter starts with an introduction to feature selection. Following is a section on which automatic feature selection algorithms are included in AGFACAND. Then follows a description of the novel automatic feature selection algorithm, Ranked Distinct Elitism Selection Filter (RDESF), developed as part of the doctoral research project work. The chapter ends with a summary.

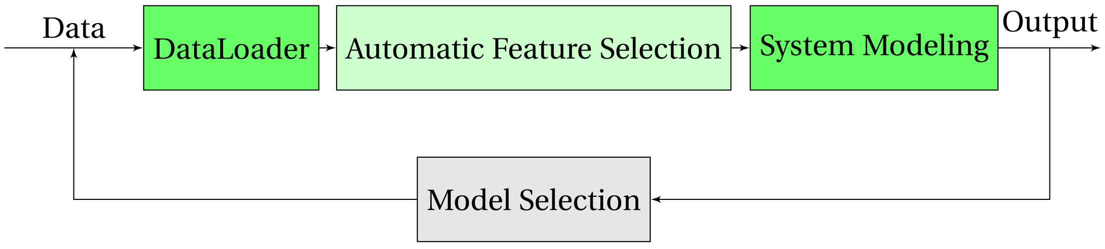
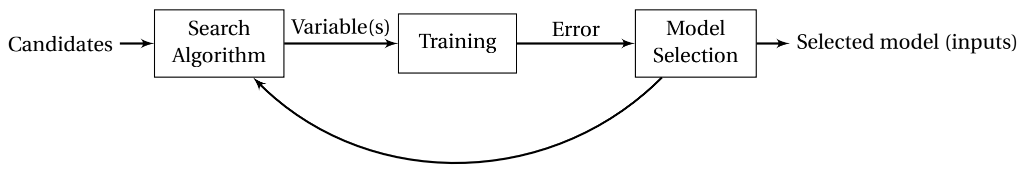


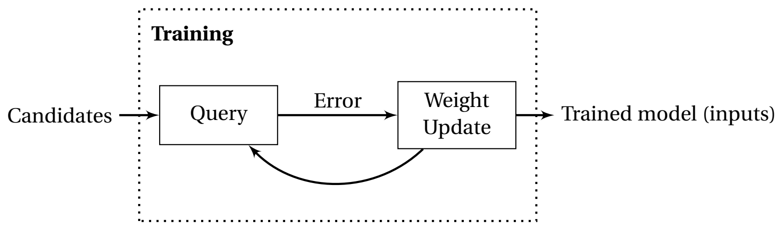
Figure 4.1: Automatic Feature Selection is the focus of this chapter.

4.1 Introduction to Feature Selection

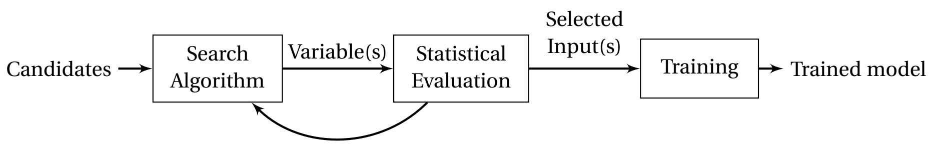
Feature selection is broadly split into three categories: *wrapper*, *embedded* and *filter* algorithms (Robert May and Maier, 2011). Common to all algorithms is that they select a subset of the input features. Another strategy is to reduce the dimensions of the original feature set; such methods are named *dimensionality reduction* algorithms (Robert May and Maier, 2011).



(a) Wrapper



(b) Embedded



(c) Filter

Figure 4.2: The three types of feature selection, after Robert May and Maier (2011).

The three types of feature selection are depicted in figure 4.2. The wrapper method (4.2a) includes the feature selection as an optimization component by training the model for every possible feature. The optimization component searches through all possible combinations of features or subsets thereof, and selects the set with the best generalization performance (Robert May and Maier, 2011). The wrapper method is computationally exhausting since every combination of features is trained with the data set resulting in $2^n - 1$ complete training passes for n candidate features.

The embedded method (4.2b) incorporates the feature selection into the model's training process. An example of such a model is evolutionary ANNs such as NeuroEvolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen, 2002). NEAT applies a genetic algorithm to build a neural network by altering both the weights as well as the structure of the network (Stanley and Miikkulainen, 2002). As listed in chapter 3, no models in AGFACAND include an embedded feature selection process.

The filter method (4.2c) separates the feature selection process from the model training. The filter method performs statistical analysis on individual or combinations of features to measure relevance to the output feature (Robert May and Maier, 2011). Finally, dimensionality reduction is a class of algorithms that reduce the number of features in a data set by extracting features by transforming the data to a space of fewer dimensions (Robert May and Maier, 2011).

4.2 Feature Selection Algorithms included in AGFACAND

Only completely automatic feature selection algorithms can be included in AGFACAND in order for AGFACAND to be completely automatic as well. The algorithms included in AGFACAND are Principal Component Analysis (PCA) and a novel filtering algorithm, Ranked Distinct Elitism Selection Filter (RDESF). These two algorithms provide fast options for applying automatic feature selection in AGFACAND, and chapter 6 will cover how further algorithms can be included in AGFACAND. For comparison during the development of RDESF, the following algorithms have also been developed: Non-Linear Principal Component Analysis (Kramer, 1991), Forward and Backward Selection (Robert May and Maier, 2011; Cavanaugh, 1997). Those algorithms are not included in AGFACAND because of poor performance or high time consumption. The comparison can be found in (Wollsen, Hallam and Jørgensen, 2016). Additionally, an option for no feature selection is also included in AGFACAND. More specific implementation details will be given in chapter 6.

Principal Component Analysis

PCA is a linear feature dimensionality reduction technique. The features are mapped into a smaller dimensional space to hopefully reveal structures in the underlying data (Pearson, 1901; Shlens, 2014). The mapping converts the features into a set of linearly uncorrelated variables or "principal components", which can be calculated using Singular Value Decomposition implemented in the Apache Commons Math Library[1]. A subset of the components is selected by removing components whose standard deviations are close to zero with respect to machine precision. PCA is included in AGFACAND because of its extensive use in literature (Wei, 2012; Beccali et al., 2004; Yang et al., 2005; Zhang and He, 2007).

Ranked Distinct Elitism Selection Filter

Filtering algorithms are model-free, which makes them very fast. Model-free is illustrated in figure 4.2c by having the training component separated from the feature selection process. Unfortunately they require a threshold that decides which features are selected. This presents a problem because the same threshold cannot be used for all algorithms and selecting a threshold requires expert knowledge. RDESF is a novel filter feature selection algorithm developed during this doctoral project. It is a meta-filter that combines commonly used filtering algorithms to combine their strengths and to create a broad-ranging generic filter that can be used in many applications. The included filters return a ranked score of the input features based on their individual measurement. This still means that a threshold is required to select

[1]http://commons.apache.org/proper/commons-math/

the relevant features. To overcome this issue, an elitism selection is used inspired by Genetic Algorithms: the top 10 % highest ranked features are selected from every included filter. Experiments on adjusting this elitism rate follows later in this chapter. From the combined features a distinct union selection based on set theory is performed to remove duplicate features. The process of RDESF is:

1. Let every included filter score the features based on its respective measurement;
2. Rank the scores and select the best 10 % from each filter;
3. Perform a union selection on the combined selected features from the filters.

The included filters are Shannon entropy, Granger Causality, Mutual Information (MI), Pearson and Spearman Correlations. Common for all description of the filters is that **X** is a single feature (a column in the data set), **Y** is the output vector and n is the number of samples in the data set.

Shannon entropy

The Shannon entropy is a measure of unpredictability, which means that features with more uniform probability density over value will be ranked higher. The entropy of a random variable, $H(\mathbf{X})$ is defined as (Shannon, 2001):

$$H(\mathbf{X}) = -\sum_i P(x_i)\log P(x_i) \tag{4.1}$$

For every feature a histogram is produced and $P(x_i)$ returns the probability of the sample x_i from the histogram. $H(\mathbf{X})$ is then the ranking for that feature. The process is repeated for every feature in the data set.

Granger Casuality

A variable X "Granger-causes" Y if Y can be predicted better using the histories of both X and Y than it can using the history of Y alone (Granger, 1988). The Granger causality score is calculated by creating two linear regressions; one that only contains the samples from Y and one that also includes the samples from X. The F statistics is calculated from the residuals:

$$F = \frac{\frac{\mathrm{RSS}_1 - \mathrm{RSS}_2}{1}}{\frac{\mathrm{RSS}_2}{n-2}} \tag{4.2}$$

An F(1, n-2) distribution is used to calculate the cumulative probability of the F statistic. This cumulative probability is the ranking of the feature X. The process is repeated for every feature in the data set.

Mutual Information

Mutual Information (MI) is a measure of the distance between the distributions of two variables and hence of the dependence of the variables (Cover and Thomas, 1991). Choosing features with high dependence to the output could indicate that the feature is good for predicting the output. The MI of an input X to the output Y is defined as (Cover and Thomas, 1991):

$$I(X;Y) = \sum_i \sum_j p(x,y) \log \frac{p(x_i, y_j)}{p(x_i)p(y_j)} \tag{4.3}$$

with $p(x_i, y_j)$ being the joint probability density function and $p(x_i)$ and $p(y_j)$ being marginal probability distribution functions of samples of X and Y respectively. For every feature in the data set, the joint and marginal probabilities are approximated using histograms. With the histograms, the formula can be applied by calculating the probabilities from the histograms. The process is repeated for every feature in the data set.

Pearson Correlation

The Pearson correlation coefficient is a measure of the linear dependence between two variables (Pearson, 1895; Shanmugan and Breipohl, 1988). Shanmugan and Breipohl (1988) defines the Pearson correlation coefficient as:

$$\rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y} \tag{4.4}$$

with cov() as the covariance function and σ_X and σ_Y are the standard deviations of X and Y respectively. Once again, the formula is applied for every feature in the data set.

Spearman Correlation

Spearman's rank correlation coefficient is simply the Pearson correlation coefficient applied to ranked data (Spearman, 1904; Shanmugan and Breipohl, 1988). Ranking the data will be more resistant to outliers (Shanmugan and Breipohl, 1988), and does not assume the data is numerical. The ranking used is the normal ordering of the input. Once again the process is applied for every feature in the data set.

Configuration of Ranked Distinct Elitism Selection Filter

The elitism rate, the selection mechanism and the included filters of RDESF have been investigated. All parameters were tested in a series of experiments on two data sets and in both short-term and long-term prediction horizons. The best elitism rate is dependent on the data set to which RDESF is applied, but is fairly stable between 5 % - 10 %. The results from one of the data sets can be found in figure 4.3. An elitism rate of 10 % is chosen in the implementation in AGFACAND, but this is configurable.

The union selection mechanism was benchmarked against the selection mechanisms intersection, complement and difference and tested with a one-way ANOVA test. Union outperformed or equaled the best selection mechanism in all experiments. Figure 4.4 clearly shows how the union selection outperforms the other selection mechanisms for one of the data sets.

Finally the included filters were excluded in a leave-one-out approach and tested with a one-way ANOVA to uncover whether a single filter decreases the performance of the prediction with the reduced data set. Figure 4.5 shows the error with all filters

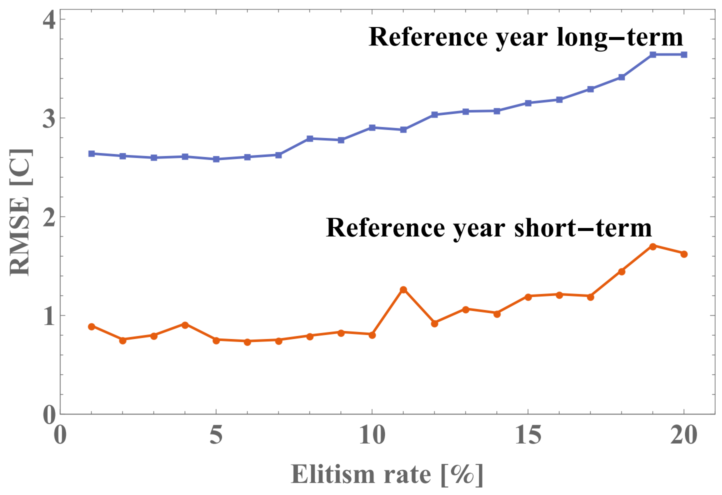


Figure 4.3: The RMSE of changing the elitism rate for both short-term and long-term prediction horizons. Reference year refers to the data set, for which the reader is refered to Wang et al. (2013)

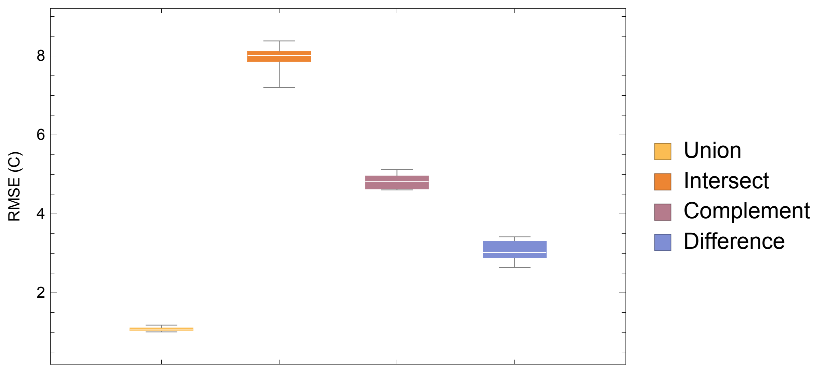


Figure 4.4: Boxplots for the different selection mechanisms for short-term prediction of the reference year data set (Wang et al., 2013). The y-axis is the RMSE of the resulting model with RDESF applied.

included and the individual filters excluded. Excluding the Shannon Entropy sub-filter lowered the mean of the error in the experiment, but there wasn't statistical confidence to draw any conclusions.

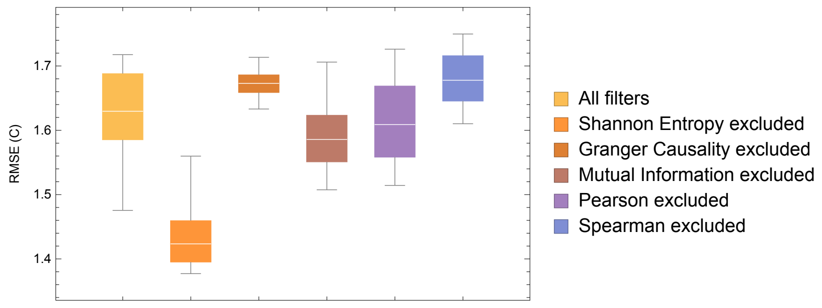


Figure 4.5: Boxplots for the different selection mechanism for short-term prediction of the reference year data set Wang et al. (2013). Once again, the y-axis is the RMSE of the resulting model with RDESF applied.

4.3 Summary

This chapter introduced the concept of feature selection, why it is important and how it performed. Algorithms that allow for automatic feature selection were presented, including a novel algorithm, RDESF, that was developed as part of this doctoral research project.

CHAPTER

5

AUTOMATIC MODEL SELECTION USING BAYESIAN OPTIMIZATION

In chapters 3 and 4, the process from data set to final model was reviewed. The process consists of loading data, performing feature selection and finally producing a model. The process of selecting the optimal feature selection algorithm, the optimal modeling algorithm and configuring any model hyperparameters is called model selection.

The chapter presents the body of knowledge of model selection. The chapter starts with an introduction to model selection. Following that is a section on how model selection has been automated in AGFACAND using Bayesian Optimization (BO). A section on other considered approaches for model selection will follow and the chapter is finally summarized.

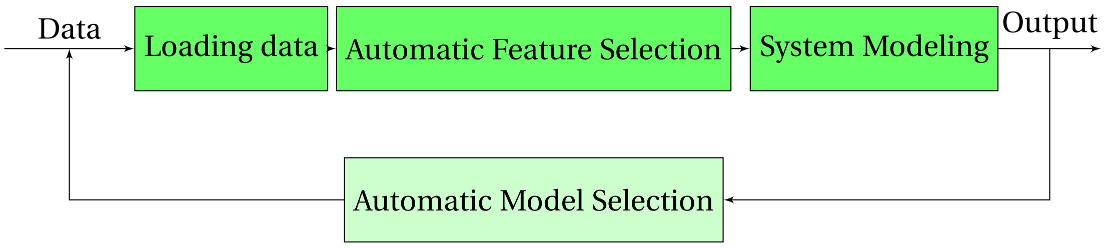


Figure 5.1: Automatic model selection using Bayesian Optimization is the main focus of this chapter.

5.1 Model selection

Model selection is the process of finding the best performing model with respect to a relevant measurement from a set of models that are created with different hyperparameters. As covered in chapter 3, the data is split into two sections: training data and test data. This splitting allows the measurement of generalization performance of the produced model, by presenting it with unseen data. To avoid noisy performance measures, cross-validation is performed, where the generalization performance is measured multiple times for the same data set (Bishop, 2006). Cross-validation is illustrated in figure 5.2.

The training set can be further divided into a training set and a validation set. The validation set is used for model selection. The model's performance on the validation set is used for selecting the best model. Cross-validation suffers from an increase in

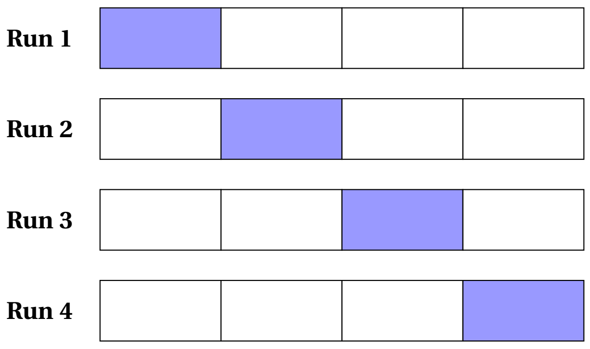


Figure 5.2: k-fold cross-validation. The data set is split into k folds, 4 in this case, and the $k-1$ folds are used for training and evaluated on the remaining fold (blue). This procedure is repeated for all k folds and the performance measures are averaged from the k runs.

training runs by a factor of k, which is problematic for models where the training itself is computationally expensive (Bishop, 2006).

Considering only the configuration of hyperparameters, model selection can be considered an optimization problem. This optimization can be formulated as:

$$\boldsymbol{\lambda}^* = \underset{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}}{\operatorname{argmin}} \mathcal{L}(M_{\boldsymbol{\lambda}}) \tag{5.1}$$

where $\boldsymbol{\lambda}^*$ is the optimal hyperparameter of a hyperparameter $\boldsymbol{\lambda}$ in the hyperparameter search space $\boldsymbol{\Lambda}$. $\mathcal{L}$ is the error function of applying model M with hyperparameters $\boldsymbol{\lambda}$ on the data set. The error function in AGFACAND is the RMSE which is defined as (Shanmugan and Breipohl, 1988, p.497):

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n}(\hat{y}_t - y_t)^2}{n}} \tag{5.2}$$

where $\hat{y}_t$ is the predicted value and y_t is the actual value. The number of samples is denoted n.

Selecting the optimal feature selection algorithm and model can be transformed into a hyperparameter optimization as a combined problem by extending equation 5.1 (Thornton et al., 2013):

$$M^*_{F^*,\boldsymbol{\lambda}^*} = \underset{M^j \in \mathcal{M}, F^k \in \mathcal{F}, \boldsymbol{\lambda} \in \boldsymbol{\Lambda}}{\operatorname{argmin}} \mathcal{L}\left(M^j_{\boldsymbol{\lambda}}, F^k\right) \tag{5.3}$$

where $\mathcal{M}$ is a set of models $\mathcal{M} = \{M^1, \ldots, M^j\}$ with associated hyperparameter spaces $\Lambda^1, \ldots, \Lambda^j$. Similarly, $\mathcal{F}$ is a set of feature selection algorithms $\mathcal{F} = \{F^1, \ldots, F^k\}$. The error function $\mathcal{L}$ is expanded to also take the feature selection algorithm F^k as a parameter.

Bayesian Optimization (BO) is a method for finding global maxima, or minima by applying a transformed function $g(x) = -f(x)$, of expensive cost functions (Brochu et al., 2010) that has shown to outperform methods such as an exhaustive grid-search or random search (Snoek et al., 2012; Bergstra et al., 2013; Thornton et al., 2013). The training of models is computationally expensive, which makes BO ideal for this optimization problem.

5.2 Bayesian Optimization

Bayesian Optimization employs the Bayesian technique of setting a prior over the objective function and combining it with evidence to get a posterior function (Brochu et al., 2010). The objective function is the generalization performance achieved by applying the selected model and configuration on the data set. Bayes theorem states (simplified) that the posterior probability of a model M given evidence E is proportional to the likelihood of E given M multiplied by the prior probability of M (Brochu et al., 2010):

$$P(M|E) \propto P(E|M)P(M) \tag{5.4}$$

Let $\mathbf{x}_i$ be the ith sample in the hyperparameter search space and let $f(\mathbf{x}_i)$ be the observation of the objective function that needs to be minimized. In the case of model selection, the objective function is the error function $\mathcal{L}$ With the accumulated observations $\mathcal{D} = \{\{\mathbf{x}_1, f(\mathbf{x}_1)\}, \ldots, \{\mathbf{x}_n, f(\mathbf{x}_n)\}\}$, the prior distribution is combined with the likelihood function $P(\mathcal{D}|f)$. The posterior distribution is then:

$$P\left(f|\mathcal{D}\right) \propto P\left(\mathcal{D}|f\right) P\left(f\right) \tag{5.5}$$

The prior distribution $P\left(f\right)$ is estimated with a Gaussian Process (GP). A GP is a stochastic process that represents the evolution of a system over time (Rasmussen and Williams, 2006). It is defined as:

$$f \sim \mathcal{GP}(m, k) \tag{5.6}$$

with m and k being the mean and the covariance functions respectively. The mean function is $m(\mathbf{x}) = 0$ and the covariance function is the Matérn 5/2 kernel (Rasmussen, 2006):

$$k(\mathbf{x}_i, \mathbf{x}_j) = \left(1 + \sqrt{5}r + \frac{5}{3}r^2\right) e^{-\sqrt{5}r} \tag{5.7}$$

where

$$r = \sqrt{\sum_{m=1}^{d} \left(\mathbf{x}_{i,m} - \mathbf{x}_{j,m}\right)^2} \tag{5.8}$$

where d is the size of $\mathcal{D}$ resulting in the kernel matrix:

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \ldots & k(\mathbf{x}_1, \mathbf{x}_t) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_t, \mathbf{x}_1) & \ldots & k(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix} \tag{5.9}$$

The kernel matrix is used to draw samples from the prior by the distribution $\mathcal{N}(0, \mathbf{K})$. For a future version of AGFACAND, the Matérn 5/2 kernel should be replaced by its automatic relevance determination (ARD) equivalent that automatically scales each hyperparameter independently. Other options for a kernel include the squared exponential function and the rational quadratic kernel.

The likelihood $P(\mathcal{D}|f)$ is a measure of how likely the data that has been seen is, based on the prior (Brochu et al., 2010). The acquisition function of BO determintes the next location of the hyperparameter search space $\mathbf{x}_{t+1}$. The objective of the acquisition function is a trade-off between exploration of areas where the prior is uncertain and exploitation of areas where the values of the prior is expected to be low. Given the acquisition function $\mathcal{U}$, the optimal location for sampling f is at $\operatorname{argmax}_{\mathbf{x}} \mathcal{U}(x|\mathcal{D})$. The acquisition function used in AGFACAND is the expected improvement (Jones et al., 1998) with an added variable (ξ) to control the exploration/-exploitation trade-off (Lizotte, 2008):

$$\mathcal{U} = \mathrm{EI} = \begin{cases} \left(\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi\right)\Phi(Z) + \sigma(x)\phi(Z) & \text{if } \sigma(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma(\mathbf{x}) = 0 \end{cases} \tag{5.10}$$

where

$$Z = \begin{cases} \frac{\mu(\mathbf{x}) - f(\mathbf{x}^+) - \xi}{\sigma(\mathbf{x})} & \text{if } \sigma(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma(\mathbf{x}) = 0 \end{cases} \tag{5.11}$$

and $\phi(\cdot)$ and $\Phi(\cdot)$ denote the PDF and CDF of the standard normal distribution respectively. $f(\mathbf{x}^+)$ is the minimum of sampled values of f in $\mathcal{D}$ and $u(\mathbf{x})$ is the mean function sampled from the prior distribution, $P(f)$, using the Cholesky decomposition (Gentle, 2009) of the kernel matrix in equation 5.9. Lizotte (2008) suggests that $\xi = 0.01$ works well in almost all cases.

Using equations 5.6 and 5.10, the procedure of BO is:

Listing 5.1: Bayesian Optimization

```
for t=1,2,... do
        Find $\mathbf{x}_{t+1}$ by optimizing the acquisition function over the GP:
            $\mathbf{x}_{t+1} = \operatorname{argmax}_{\mathbf{x}} \mathcal{U}(\mathbf{x}|\mathcal{D})$
        Sample the objective function: $y_{t+1} = f(\mathbf{x}_{t+1})$
        Augment the data $\mathcal{D} = \{\mathcal{D}, \{\mathbf{x}_{t+1}, y_{t+1}\}\}$ and update the GP.
end for
```

Optimized hyperparameters in AGFACAND

To summarize, the hyperparameters that are optimized in AGFACAND are: which model to apply, the size of the model if applicable, which feature selection algorithm to apply and the size of the delay when loading the data. The models included in AGFACAND were reviewed in chapter 3. Because of how they function, model size is not applicable for Linear regression, ADALINE, GRNN and SVR. The feature selection algorithms included in AGFACAND were reviewed in chapter 4. This means that $\mathbf{x}_i$ in equation 5.7 is a vector $\mathbf{x}_i = \{M^j, F^k, D, S\}$ with D and S being delay size and model size respectively. The delay size and model size form the hyperparameter vector $\boldsymbol{\lambda}$ in equation 5.1. The search space for $\mathbf{x}_i$ is created by calculating the cartesian product of the individual search spaces. The cartesian product is the set of all ordered pairs, meaning that every possible combination of the parameters in the vector is in the set.

By default the model size is divided linearly 11 times between 100 and 1000. The delay is divided similarly in 13 times between 0 and 24. This default division for model size is suitable for most cases. The delay size is very dependent on the prediction horizon, and therefore the model size and the delay size can both be overridden as part of customizing AGFACAND.

In order to get a wide distribution of the initial random samples in $\mathcal{D}$, the samples are guaranteed to be from different models. By sampling from the different models, unsuitable models for the data set will only be sampled a few times by the exploitation trade-off in the acquisition function. Unsuitable models in this case means models that perform poorly on the given data set.

The BO iterates by default 50 times, but this is also configurable.

5.3 Other approaches for model selection

Besides the related literature reviewed in chapter 2, other methods for model selection were considered. An interesting approach has been proposed by (Gibert et al., 2010) where the meta-data of a data set is analyzed in order for a decision support system based on Case-Based Reasoning (CBR) to suggest the optimal data mining technique. CBR works by re-using the solutions to similar problems which are stored in a case base. The meta-data they use are number of features, ratio of symbolic features and relative probability of missing values and document promising results. Their experimental setup is very limited, but the approach of analyzing meta-data is interesting.

Based on the ideas from (Gibert et al., 2010), the author performed some experiments on similar approaches. Model selection based on meta-data analysis is very desirable as it would be unnecessary to evaluate the possible models against the data set and thus save a lot of computational time. However, the meta-data analysis is not feasible due to incomplete information by not knowing the result of evaluating another model other than the selected. Furthermore, biases can be present that can give false selections and the approaches might also suffer from requiring large training data sets. Two approaches are reviewed here:

Informed meta-data

The first approach is also based around meta-data but uses a classification methodology. The meta-data would consist of subset of the data set reduced by Random Projection (RP). RP is a dimensionality reduction technique that is computationally fast compared to PCA and is widely used in literature (Bingham and Mannila, 2001). Given a data set as a matrix $\mathbf{D}$ with size NxM, the reduced data set $\mathbf{D_R}$ is then:

$$\mathbf{D_R} = \mathcal{R}_{K,N}\left(\mathbf{D}\mathcal{R}_{M,L}\right) \tag{5.12}$$

where $\mathcal{R}_{K,N}$ is a normally distributed random matrix of size KxN. K and L are hyperparameters of this approach and should be small (< 10). It is then up to a classification algorithm to match the meta-data with an error from running the a model $\mathcal{M}$ on the data set.

A big problem with this approach is that a generic modeling technique such as ANNs will result in low errors for many data sets, although maybe not the lowest. This results in a biased classification, where the classification algorithm will guess on that technique for all test cases. Additionally, similarly to the approach by (Gibert et al., 2010), this approach suffers from the need of a large training set.

Complete knowledge

The second approach is based on similarity scores between data sets. Given a data set as a matrix $\mathbf{D}$ with size NxM the generalization performance from a model is saved. The generalization performance could be the RMSE or other similar measures. Given a new data set $\mathbf{D}_x$, a similarity score S could be measured between existing data sets and the new data set:

$$S_{x_i} = \mathcal{S}(\mathbf{D}_x, \mathbf{D}_i) \tag{5.13}$$

where $\mathcal{S}$ could be Dynamic Time Warping (DTW) (Berndt and Clifford, 1994), kernel distances (Phillips and Venkatasubramanian, 2011) or similar methods. The similarity score S would have to be calculated against every data set in the database in order to find the most similar data set. The model used for that specific data set would be the recommendation for the new data set $\mathbf{D}_x$. This is similar to k-nearest neighbor with $k = 1$.

Obviously this method suffers from the problem that every single data set would have to be saved in a database which takes up a lot of space. Another big problem is that the similarity score has to be measured against every other data set, making the method computationally heavy.

5.4 Summary

In this chapter, the process of Bayesian Optimization has been reviewed and its use in AGFACAND has been explained. By applying BO in AGFACAND, the model selection process is automated and does not require human interaction besides initial setup. This minimizes the risk of errors and also allows users without the necessary expert knowledge to use AGFACAND. Two decision support systems were suggested as an alternative to model selection, but it was also explained why they are not applicable to AGFACAND.

CHAPTER

6 DESIGN AND IMPLEMENTATION

In the previous three chapters the theoretical background of the fundamental parts in AGFACAND were covered. These parts are respectively: Machine learning regression, feature selection and model selection. Each of these parts has its own requirements for the implementation in the framework and must support the interaction and functionality between those parts.

This chapter covers the framework design of AGFACAND and how the three fundamental parts are implemented. Examples will be given of how AGFACAND can be extended with new models through its extensible framework design. Additionally, implementation of other framework functionality that is needed to support the essential parts of AGFACAND will also be discussed, in order to give a complete overview of the functionality included in AGFACAND. How AGFACAND exports models and integrates them into other systems is also covered. Finally, the chapter ends with a summary.

6.1 Framework design

AGFACAND is designed with extensiblity in mind, such that multiple types of models and feature selection algorithms can be implemented and added in the future. The package structure of AGFACAND is presented in figure 6.1.

The modeling part of AGFACAND as presented in chapter 3 is placed in the *core* package of figure 6.1. The package contains the *Model* interface which will be described in the next section. The specific model types is implemented in the *networks* subpackage.

The automatic feature selection of AGFACAND as described in chapter 4 is placed in the *featureselection* package that is also a subpackage of the *data* package. The *featureselection* package contains the *FeatureSelector* interface which will also be described later in this chapter. This package also contains a subpackage of every type of feature selection algorithm for a clear separation of the different types. The *data* package also contains the *normalization* package that holds all the different types of data normalization techniques. Cross validation, a matrix wrapper and the *ModelContainer* class are also all placed in the *data* package. All of them will be explained in this chapter. This package also contains the *DataLoader* class that loads the data as described in chapter 2.

The BO is placed in the *optimization* package and is the only class in this package. The *tools* package contains a plotting tool which is used in the tutorial of chapter

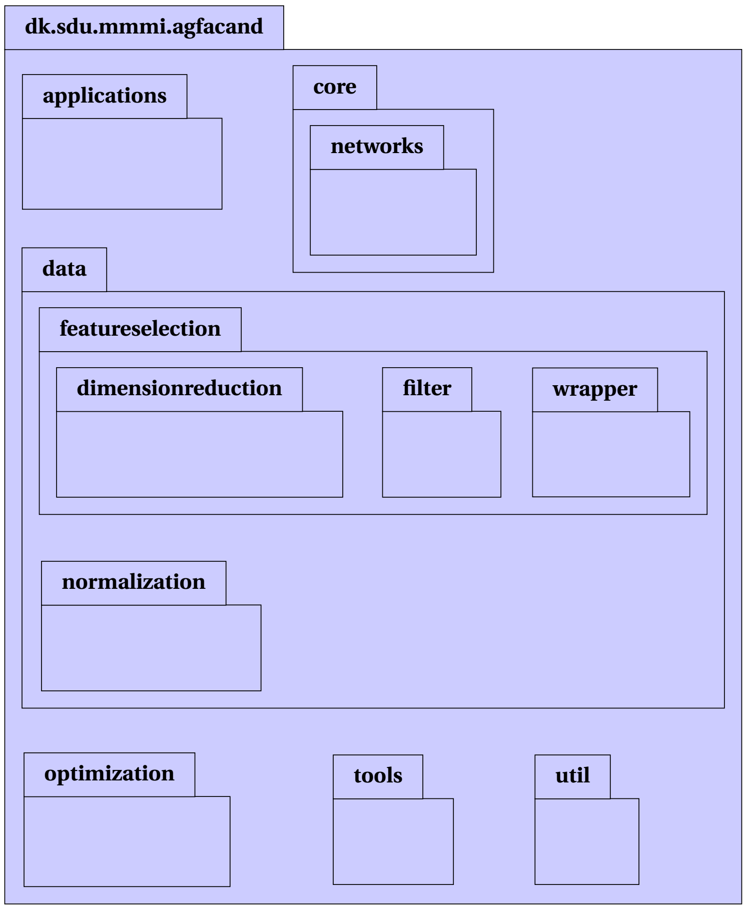


Figure 6.1: Package diagram of AGFACAND

7. The *util* package contains utility classes for arrays, matrices and statistics and include support functionality for the fundamental parts of AGFACAND.

Finally, the *applications* package contains all the runner classes for the specific applications on which AGFACAND is applied. A runner class is a class named and created solely for the specific application. A runner class will contain the code to initialize the functionlity of AGFACAND. The applications are reviewed in chapter 8.

The clear separation of the parts of AGFACAND in packages ensures that related classes are located together. The packages *core*, *featureselection* and *normalization* are only accessed through their respective interfaces. These interfaces are the extensibility points of AGFACAND that allows AGFACAND to be expanded with additional algorithms of the different types without requiring any changes to the existing functionality of AGFACAND.

6.2 Implementation of models

The Encog Machine Learning Library for Java (Heaton, 2015*a*) has been used a base for AGFACAND. Encog already implements some of the included models mentioned in chapter 3. Furthermore, Encog includes a Matrix implementation to hold data. Encog is well-tested, optimized, has support for multithreading and has an active community.

The models in AGFACAND are separated into two categories: those that are implemented in Encog and those that are implemented by the author. The models that are already implemented into Encog are: Multilayer Perceptron, ADALINE, Elman, General Regression Neural Network and Support Vector Regression. The models implemented by the author include: Echo State Network, Extreme Learning Machine and Linear. All models needs to implement a common interface that defines any interaction with a model. These interactions include functionality such as training, running a trained model, feature selection and configuration of parameters (normalization and model parameter). Additionally, all Encog models have almost identical code for training and running that has been moved to a higher abstraction layer via an abstract class.

Abstract class for Encog models

The abstract class *EncogNetwork* is the super class for all wrapper classes of Encog models. Each Encog model is wrapped in a class by AGFACAND and the *EncogNetwork* class contains the code to perform training and running of these models as the code is identical. The process of training and running a model in AGFACAND is: Perform feature selection, normalize data, train/run and denormalize results. For the Encog models, the training and running step contains a data transfer step as Encog uses another class for data used with ANNs. This class (*MLDataSet*) holds every sample of data as a data pair of input data and desired output data, and is thereby a combination of two matrices.

In Encog every type of ANN extends an abstract class called *BasicML*. The training algorithms for the ANNs implement an interface called *MLTrain*. This means that *EncogNetwork* class must hold a reference to both the *BasicML* class and the *MLTrain* interface. Another abstraction level is however put into the *BasicML* class depending on what type of ANN that is being implemented. This lower abstraction level divides the networks into three types: *BasicPNN*, *BasicNetwork* and *SVM*. The *BasicPNN* class is for GRNN, the *SVM* class is for SVR and the *BasicNetwork* class is for ADALINE, Elman and MLP. One way of dealing with this additional abstract layer is through the use of generic types. This results in the following signature for the *EncogNetwork* class:

Listing 6.1: Class signature for the *EncogNetwork* class

```java
public abstract class EncogNetwork<T extends BasicML> implements Model { ... }
```

and an abstract method to get the training algorithm:

Listing 6.2: Method signature to get the training algorithm for an Encog model

```java
public abstract MLTrain getTrainer(T network, MLDataSet trainingSet);
```

In listing 6.1 the generic type must extend the *BasicML* class which limits the subclasses to classes that contain Encog functionality. This also limits the risks of implementation errors. Further more, the *EncogNetwork* class implements the Model interface which will be described in the next section. The code to prepare the data for training the model is identical across all Encog networks. This can thereby be implemented in the *EncogNetwork* class as it is an abstract class. The training itself is performed in the training algorithm. The *MLTrain* interface defines an *iterate* method which trains the network when called. This method may have to be called multiple times depending on the specific type of ANN, but this is handled in the *EncogNetwork* class. In order to get the training algorithm for the specific type of ANN, the template method design pattern is used by defining the abstract method in listing 6.2.

Although the code to run the models are identical for all Encog models, the *compute* method is not defined in the *BasicML* class, but rather in the next level of classes: *BasicPNN*, *BasicNetwork* and *SVM*. The *compute* method runs the data through the network to produce a result. Therefore does the *EncogNetwork* class define an abstract method named *computeNetwork* to run the models as shown in listing 6.3

Listing 6.3: Method signature to run a trained model for Encog models

```java
protected abstract MLData computeNetwork(MLData mlData);
```

Listing 6.4 is the full implementation of GRNN which is included in Encog and therefore uses the *EncogNetwork* abstract class. The constructor calls the constructor of the *EncogNetwork* class as well, that sets an iteration threshold for the training loop. Although almost all models in AGFACAND uses the range normalization, it is up to the individual model to configure this. This will ensure a strict format in the code and improves readability. More information on normalizers will be given in section 6.7. The *initializeNetwork* method is called in the beginning of training method which will be covered in the next section. The *initializeNetwork* method creates the specific type of ANN, in this case a GRNN. Finally, the *getTrainer* method returns the exact training algorithm that is used to train the GRNN model. By having the abstract class *EncogNetwork*, the implementation of Encog models is simplified and the resulting code is relatively small.

Listing 6.4: Java code for the implementation of GRNN with the use of the abstract class *EncogNetwork*

```java
public class GeneralRegressionNetwork extends EncogNetwork<BasicPNN> {
  public GeneralRegressionNetwork() {
    super();
    this.normalizer = new RangeNormalizer(-0.9, 0.9);
  }
  @Override
  public void initializeNetwork(int numberOfInputs, int numberOfOutputs) {
    this.network = new BasicPNN(PNNKernelType.Gaussian,
        PNNOutputMode.Regression, numberOfInputs, numberOfOutputs);
  }
  @Override
  protected MLData computeNetwork(MLData mlData) {
    return this.network.compute(mlData);
  }
  @Override
  protected MLTrain getTrainer(BasicPNN network, MLDataSet trainingSet) {
    return new TrainBasicPNN(network, trainingSet);
  }
}
```

The Model Interface

The common interface for all models in AGFACAND is simply called the Model interface. This interface defines methods for training a model, running a trained model and configuration of feature selection, normalizer and the model parameter.

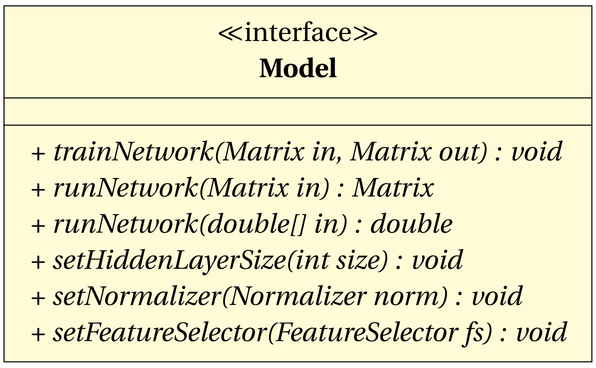

≪interface≫
Model
+ trainNetwork(Matrix in, Matrix out) : void
+ runNetwork(Matrix in) : Matrix
+ runNetwork(double[] in) : double
+ setHiddenLayerSize(int size) : void
+ setNormalizer(Normalizer norm) : void
+ setFeatureSelector(FeatureSelector fs) : void

Figure 6.2: The Model interface and its methods

The model interface is depicted in figure 6.2 and shows the methods it defines. The second *Matrix* in the *trainNetwork* method contains the desired output, which is required to train the model. The interface contains two methods for running the trained model, one that takes multiple samples and one that takes only a single sample. The method with multiple samples (the matrix) is only used internally for evaluating the network and calculating its generalization performance. The second method is used in external applications. It can be argued that only one method would be sufficient, however this approach simplifies the use of the model and for the single sample method, the sample can be wrapped in the *Matrix* class and call the other method.

The three configuration methods (*setHiddenLayerSize()*, *setNormalizer()* and *setFeatureSelector()*) configures the model and must be used before the network is trained. It is these methods that are used in the BO. The *Normalizer* and the *FeatureSelector* argument types are two interfaces and covered in section 6.7 and the following section respectively.

Network ensembles are implemented with a class that also implements the Model interface. The ensemble is simply a list of the Model interface and for all methods from the Model interface, the ensemble calls the respective method in every model in the list. By having network ensembles implement the Model interface, there is no difference between a single network or an ensemble of multiple networks. This allows ensembles to be used as a single model in all of AGFACAND with the added benefits that ensembles provide. The training method in the ensemble uses multi-threading to increase the computational speed.

The entire structure of the models in AGFACAND is depicted in figure 6.3.

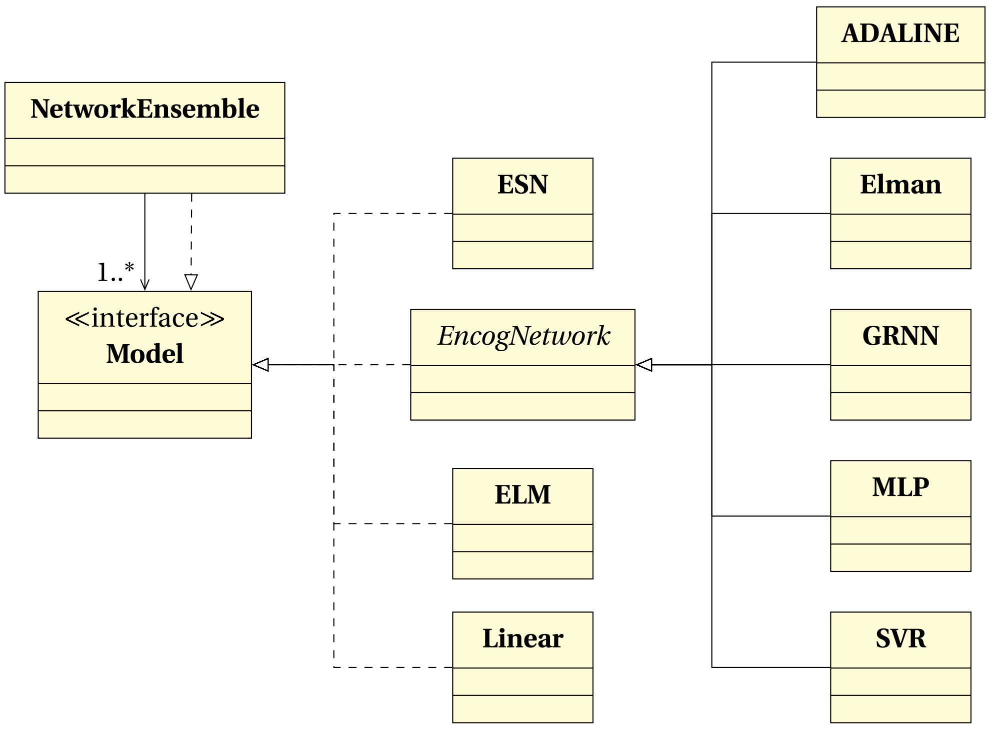


Figure 6.3: The Model classes and interface visualized with UML. The class names are abbreviated for simplicity.

6.3 Implementation of feature selection

Feature selection has been implemented by the use of interfaces and abstract classes. Remember, from the previous section that the Model interface has a method for registering a feature selector to the model using a feature selector interface. The entire structure of the feature selection hierarchy is presented in figure 6.4 and this section will cover each abstract class or interface in further detail, with *DimensionReductor* being the only exception as no functionality is defined in this class. The class is kept here to keep the clear separation between the different types of feature selection algorithms. The classes that were used for the comparison in (Wollsen, Hallam and Jørgensen, 2016) are also included in the visualization to give a complete overview, however these classes are grayed out.

This hierarchical structure not only clearly separates the different types of feature selection algorithms for better overview, but it also makes the implementation of new algorithms easier.

FeatureSelector interface

The *FeatureSelector* interface is the top-level interface for all feature selection algorithms and defines a single method, as depicted in figure 6.5. The *getInputSubset* method is used to return a subset of the input data, depending on which feature selection algorithm is used. The desired output of the model is also given as a parameter, as some of the feature selection algorithms calculate the correlation between input and output.

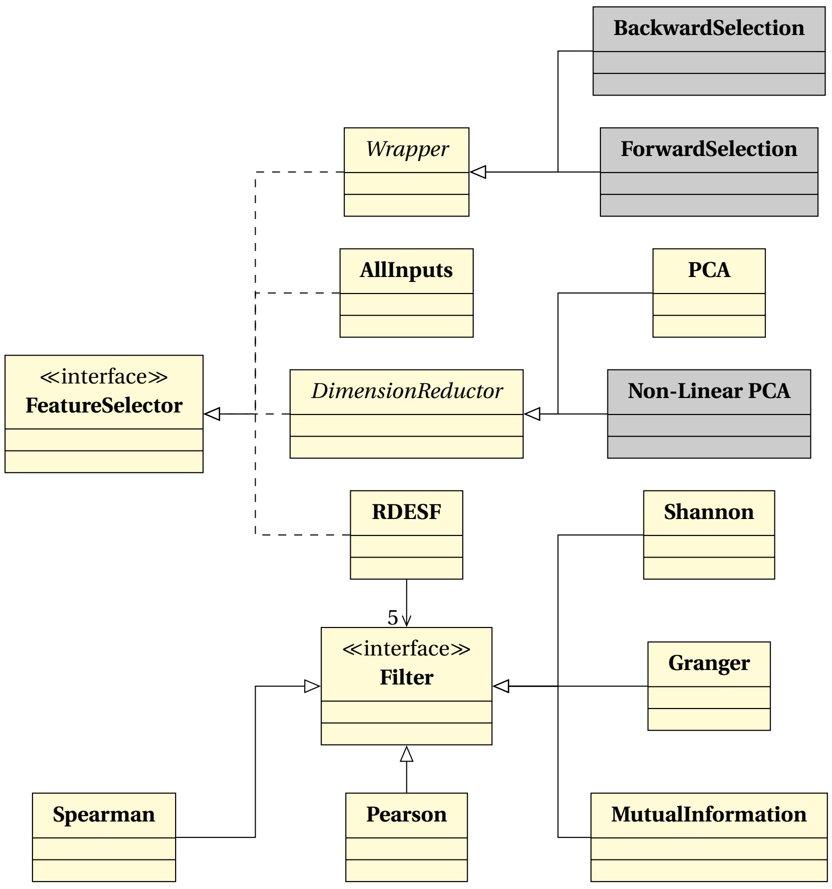


Figure 6.4: Classes and interfaces that supports the implementation of feature selection algorithms in AGFACAND. Some class names are abbreviated for simplicity and gray classes are not part of the automatic model selection.

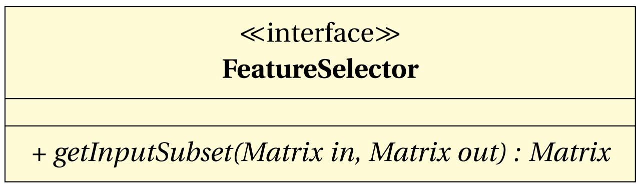


Figure 6.5: The FeatureSelector interface and its method.

The *Wrapper* abstract class

The *Wrapper* abstract class contains the functionality for the two Wrapper-type feature selection algorithms that are included in AGFACAND: backward selection and forward selection. The algorithms have been covered in chapter 4 and are not included as part of the automatic model selection because of their slow computational speed. The *Wrapper* abstract class is depicted in figure 6.6.

The wrapper-type process is as follows: The initial columns to test is picked by the subclass with the *initializeData* method where backward selection adds all columns and forward selection adds a single randomly. In a loop then, the picked columns is used to train a model and run it to get a generalization performance. With the generalization performance the corrected Akaike Information Criteria (AICc)

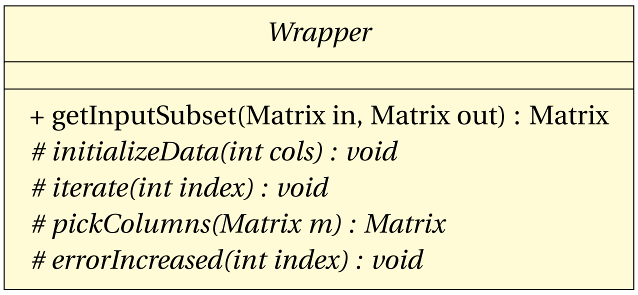

Wrapper
+ getInputSubset(Matrix in, Matrix out) : Matrix
# initializeData(int cols) : void
# iterate(int index) : void
# pickColumns(Matrix m) : Matrix
# errorIncreased(int index) : void

Figure 6.6: The abstract class Wrapper and its methods

(Cavanaugh, 1997) is calculated. If the AICc is lower than the previous AICc it is saved, otherwise the *errorIncreased* method is called in the subclass. This method will remove the chosen index from the final indices, as that index resulted in a decrease in generalization performance. Finally the *iterate* method is called for the subclasses to add a new column to be tested. At the end of the loop the *pickColumns* method is called for the subclass to pick the columns from the data that was selected during the loop.

The two wrapper-type algorithms have a lot of common functionality and only distinct themselves in a few steps. This makes it obvious to implement the common functionality in the abstract class and to call abstract methods for the distinct steps. The separation of common code to an abstract class and having the subclasses implement the specific details is also called the template method design pattern.

The *Filter* interface

The *Filter* interface does not extend the *FeatureSelector* interface, as a single filter cannot function as feature selection algorithm. This is because the filter only returns a rank of the features and a threshold is required to select a subset of the features. The *Filter* interface therefore defines a method to return the rank of the features, which is used by RDESF as described in chapter 4. The *Filter* interface is depicted in figure 6.7.

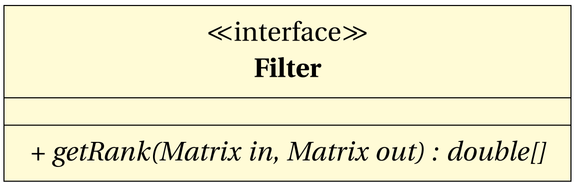


Figure 6.7: The Filter interface and its methods

The individual filters that implement this interface return a value for every feature in the data set with lower values as the most related or correlated feature.

6.4 Implementation of Bayesian Optimization

The methods in the Model interface is used by the automatic model selection. A sequence diagram of BO is depicted in figure 6.8. The figure shows how the BO process iterates and how the methods defined in the interfaces described earlier are used. BO also uses the DataLoader class to delay the data with the delay chosen from the BO process.

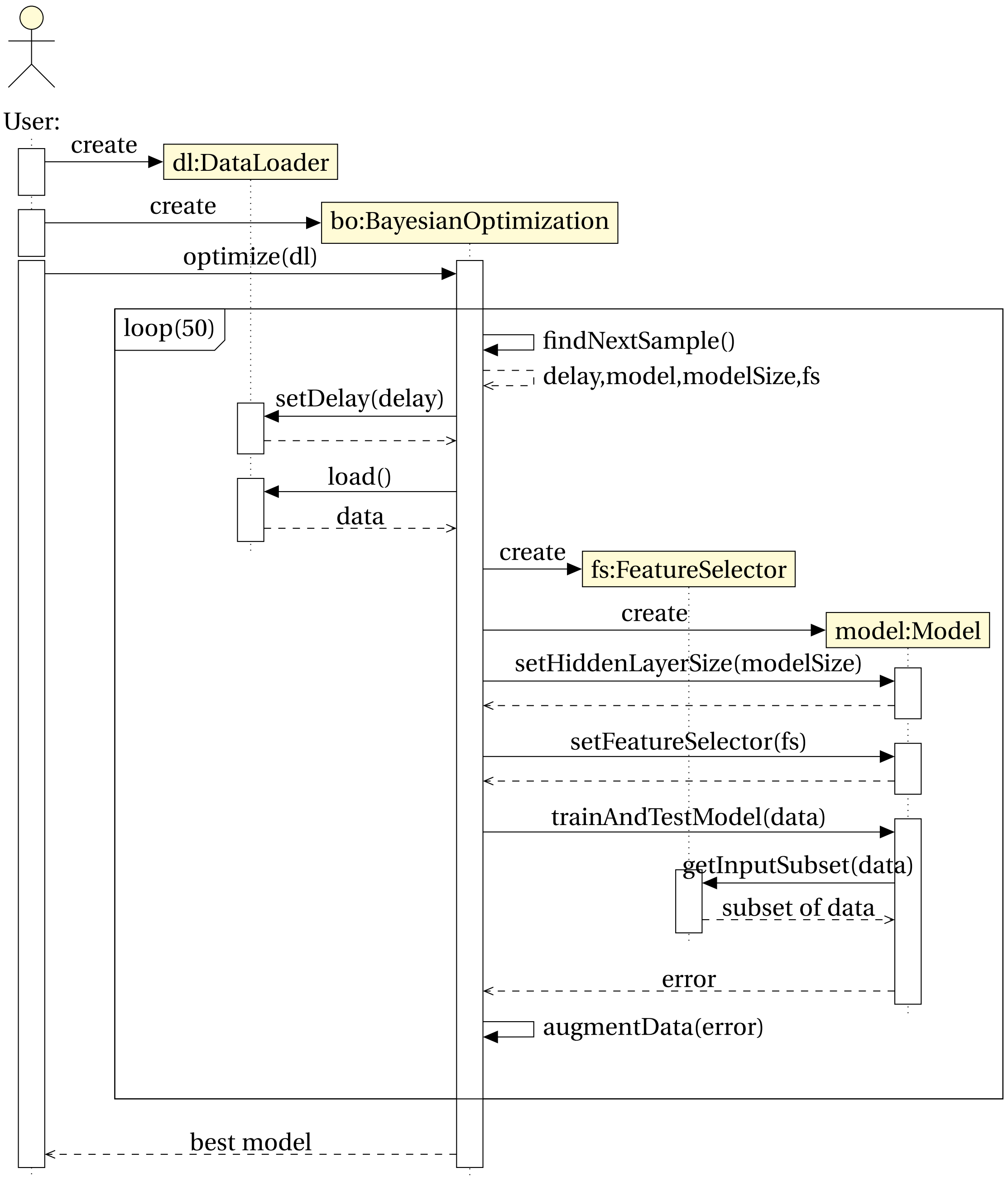


Figure 6.8: Sequence diagram for BO and the use of the interfaces in AGFACAND

The *findNextSample* method in the diagram is the process of fitting the GP and calculating the optimal location for next sample using the Expected Improvement, which was all covered in chapter 5. The *trainAndTestModel* method is a simplified way to show the calls to both *trainNetwork* and *runNetwork* from the Model interface. The *augmentData* method adds the new knowledge gained from running the chosen model which is used in the next iteration of BO.

6.5 Export and integration of models

In order to use the trained models in other systems or applications, the trained model needs to be exported. AGFACAND uses the XStream library [1] which serializes to XML and back. The advantages of using XML serialization over the default object serialization in Java are twofold: information about model such as model type, feature selection and delay size can be read directly in the file and the serialization is less sensitive to changes in the source code as long as the same variable names are used as all referencing is name based. XStream respects the "transient" keyword, which allows specific variables to be ignored in the serialization. The ability to view the contents of the serialized file such as which data set is used, the type of model used or even which type of feature selection algorithm is applied makes the XML format more informational compared to a binary serialization format.

In order to reduce the size of the serialized object and thereby the exported model, all weight matrices of the models are saved in a base64 encoded binary format. XStream supports custom converters between objects and XML. This allows the creation of a custom converter of the Matrix class in AGFACAND in order to serialize it in base64 encoded binary format. Although a binary format of matrices disables changes in the source code of the matrix class, the saved space makes up for this. Any additional functionality required in the matrix class can always be added in auxiliary classes, which thereby will not change the source code of the matrix class, and older serialized models will still be usable.

The trained model is wrapped in a *ModelContainer* class that can hold a model for every step of the prediction horizon and information on which data set was used to create the model as well as a user-defined description. This allows the creation of a model for every sample of the desired prediction horizon and all of these are contained in the *ModelContainer* class.

Along with the XML file of the model a jar file of AGFACAND is required in order to run the trained model in other applications.

Example of integration

In chapter 8, an application from the GreenTech House will be covered. The trained model from this application had to be used in another control system. The runner class and the code to load and run the trained model is all presented in listing 6.5.

Listing 6.5: Java code for the GreenTech House application for loading and running of a trained model

```java
public class GTCVentilationPrediction {
    public static double[] runModel(double[] input) {
        int predictionAhead = 24 * 4;
        ModelContainer models = ModelContainer.load("path/to/model.xml");
        double[] result = new double[predictionAhead];
        for (int i = 0; i < predictionAhead; i++) {
                result[i] = models.getNetwork(i).runNetwork(input);
        }
        return result;
    }
}
```

[1] http://x-stream.github.io/

A *predictionAhead* variable is set to $24 * 4$ as this model is trained on data with a 15-minute resolution and the prediction horizon is 24 hours. An array is created to save a value for every step towards the prediction horizon and the models in the *ModelContainer* class is then iterated. Every model in the *ModelContainer* class is presented with the same data and returns a value using the *runNetwork* method of the *Model* interface as depicted in figure 6.2. The array of the results is returned in the end.

6.6 Examples of addition of new algorithms

Example of adding a new model to AGFACAND

A new model should be placed in the *core.networks* package of figure 6.1. For the sake of this example, a model will be created that returns the difference in averages between the training data and the currently presented data. This model is by no means applicable to system modeling, but will serve nicely as an example of adding new models to AGFACAND. The model will learn the average of the training data, and once presented with new data, the model will return the difference between the average of the training data and the newly presented data. The configurable model parameter will simply be added to the average from the training data. The model will be implemented in a class called *AveragingModel*:

Four variables are created in lines 6-9 that are used to save the normalizer, the feature selection algorithm, the configurable model parameter and the average calculated from the training data. The *trainNetwork* method starts by performing feature selection and normalization of the data. Then, the *average* method from the *StatisticsUtility* class is called on every column and then on every row. Those two calls result in a single 1x1 matrix, which is why the *getData* method is called. The result is the average of the input data, and this average is saved and then the configurable model parameter is added.

The *runNetwork* method also performs feature selection and normalization first (line 21). Then the average of each row of the input is calculated (line 22) and the difference to the trained average is returned. The *runNetwork* method for a single sample just wraps the sample in a matrix and calls the previous method.

The final 3 methods are for setting the parameters which are called during the model selection as reviewed earlier.

Finally, the new model needs to be added to the BO loop. This is performed in the *BayesianOptimization* class in the *optimization* package. In the private method *buildDomainSet* the list of possible networks is saved in the variable *networkTypes*, and here a new line should be added to add the new network.

Example of adding a new feature selection algorithm to AGFACAND

A new feature selection algorithm should be added to the responding subpackage of *data.featureselection*. This example will be the addition of random projection, which is a dimension reduction algorithm that has shown to perform well as a feature selection algorithm (Bingham and Mannila, 2001). Random projection is not included

Listing 6.6: Java code for adding the new model *AveragingModel* to AGFACAND. Imports have been omited for better readability

```java
package dk.sdu.mmmi.agfacand.core.networks;

//Imports

public class AveragingModel implements Model {
  private Normalizer normalizer = new PassThroughNormalizer();
  private FeatureSelector featureSelector = new AllInputs();
  private int modelParameter = 0;
  private double trainedAverage = 0;

  @Override
  public void trainNetwork(Matrix in, Matrix out) {
        in = this.featureSelector.getInputSubset(in,out);
        this.normalizer.prepare(in,out);
        in = this.normalizer.inputNormalize(in);
    this.trainedAverage = in.mapColumnToVector(col ->
         StatisticsUtility.average(col)).mapRowToVector(row ->
         StatisticsUtility.average(row)).getData(0,0);
    this.trainedAverage += this.modelParameter;
  }
  @Override
  public Matrix runNetwork(Matrix in) {
    in = this.normalizer.inputNormalize(this.featureSelector.getInputSubset(in, null));
    Matrix average = in.mapRowToVector(row -> StatisticsUtility.average(row));
    return average.subtract(this.trainedAverage);
  }
  @Override
  public double runNetwork(double[] in) {
        Matrix matIn = new Matrix(new double[][] { in });
        return runNetwork(matIn).getData(0,0);
  }
  @Override
  public void setHiddenLayerSize(int size) {
    this.modelParameter = size;
  }
  @Override
  public void setNormalizer(Normalizer norm) {
    this.normalizer = norm;
  }
  @Override
  public void setFeatureSelector(FeatureSelector fs) {
    this.featureSelector = fs;
  }
}
```

in AGFACAND in favor of PCA. The algorithm will project the columns to half the size. The algorithm will be implemented in a class called *RandomProjection*:

The class extends the *DimensionReductor* interface which does nothing except implementing the *FeatureSelector* interface as explained earlier. The interface defines the single method *getInputSubset* method. The projection matrix is saved such that the algorithm returns the same output for the identical inputs. The projection matrix is created using one of the creation methods of the *Matrix* class which fills the matrix with normal distributed random values between 0 and 1. Finally, the subset is returned which is the input matrix multiplied with the projection matrix.

Listing 6.7: Java code for adding the new feature selection algorithm, RandomProjection, to AGFACAND.

```java
package dk.sdu.mmmi.agfacand.data.featureselection.dimensionreduction;

import dk.sdu.mmmi.agfacand.data.Matrix;

public class RandomProjection extends DimensionReductor {
  private Matrix projectionMatrix = null;

  @Override
  public Matrix getInputSubset(Matrix in, Matrix out) {
        if (out != null) {
                this.projectionMatrix = Matrix.randomNormal(in.getCols(), in.getCols() /
                    2);
        }
        return in.times(this.projectionMatrix);
  }
}
```

6.7 Other implementations

This section will shortly cover some implementations in AGFACAND that provide crucial support for the functionality of AGFACAND. This is done in order to complete the documentation of AGFACAND.

Matrix

The Matrix class in Encog has been wrapped in a new Matrix class that supports timestamps for every row, which is very useful when plotting and interpreting the results. Furthermore the new Matrix class supports additional basic algebraic operations, concatenation operations as well as commonly used methods for constructing new matrices. These methods include matrices filled with 0's, 1's, uniformly random values and normal distributed random values. Additionally, methods for calculating the Moore-Penrose pseudoinverse and linear least squares using stochastic gradient descent are also added. Finally, methods for mapping a function either row-wise, column-wise or element-wise are added. These methods can for example be used to average every row of a matrix.

Cross-validation

k-fold cross-validation is implemented as a independent class in AGFACAND. The class allows separation of the data set into an arbitrary number of folds and has methods to access the different training and test portions of a specific bin. The size of the last bin might be a bit smaller or larger compared to the rest, as the size of the bins is rounded. Cross-validation is used in BO to get a better estimate of the generalization performance of the model that is being tested.

Statistics

A utility class to do simple statistics is also implemented in AGFACAND. This class contains methods for calculating the average, the standard deviation, RMSE and normalized RMSE. Statistics such as probabilistic distributions are calculated using Apache Commons.

Normalization

As shown earlier, an interface for normalization is included in AGFACAND. This interface supports 3 different normalization techniques, however only 2 are used in AGFACAND. These techniques are: pass-through normalization, range normalization and shift normalization. Pass-through normalization is used only by linear regression, and simply does nothing. It has still been implemented by a class in order for the interface for be used. Range-normalization is used by all the other models and normalizes the data into an interval, typically -0.9 to 0.9. Finally, shift-normalization was implemented during the attempted implementation of Deep Neural Networks (DNNs) and shifts the data above 0 to have the data within the softmax units' operational interval.

6.8 Summary

This chapter covered how the three sections of AGFACAND are implemented through the use of interfaces and abstract classes. The interfaces provide a contract from AGFACAND towards the algorithms and allow for easy addition of additional algorithms in the future. By having used interfaces in the implementation of models and feature selection, the BO process was simplified. Additionally, the chapter covered the implementation of support functionality that is required in AGFACAND, such as the matrix class, cross-validation and normalization.

CHAPTER

7 TUTORIAL ON AGFACAND

This chapter will be a tutorial on applying AGFACAND on a data set. This includes loading the data set, configuration of the automatic model selection, calculating the final model and finally exporting the model.

The data set will be the SML2010 data set from the UCI Machine Learning repository. This data set is also used in one of the applications in the next chapter.

7.1 Loading the data

The data set can be found at the following url `https://archive.ics.uci.edu/ml/datasets/SML2010`. The data set is split up into two files, where only the first will be used for this tutorial. The raw file is placed in the data folder at the root of AGFACAND and renamed "sml.txt". The first three characters of the file has to be removed manually, as the file will not otherwise stick to a proper format. The data set is collected from a house and contains parameters such as indoor temperature, carbon dioxide level, relative humidity and outdoor parameters such as sun light and temperature. For the sake of this tutorial, the indoor temperature in one of the two provided rooms will be the target output for the model.

A class is created in the *applications* package, and named after the application. As this tutorial aims at predicting the temperature, the class is named *SMLTemperature*. In the *main* method the data of the SML data set is loaded as follows:

Listing 7.1: Java code to load the data in the SML application

```java
package dk.sdu.mmmi.agfacand.applications;

import dk.sdu.mmmi.agfacand.data.DataLoader;

public class SMLTemperature {
  public static void main(String[] args) {
    DataLoader dl = new DataLoader("data/sml.txt")
          .setSeparator('␣')
          .hasHeader()
          .ignoreColumn(0,1)
          .setOutputIndex(2);
  }
}
```

A *DataLoader* object is initialized with the path to the data set as an argument. Using method chaining, the separator of the data set file is set to a space, the *DataLoader* object is told that the data set file contains a header, that the first two columns should be ignored and finally that the column index of the output is 2. Remember that the first index in Java is 0. The third column will be the output which is the temperature of the dining room ("comedor" in the data set). By telling the *DataLoader* object that the file contains a header, the first line in the data set file is removed when loading the data, but saved for when the model is exported.

7.2 Configuration of AGFACAND

The next task is to configure AGFACAND and the automatic model selection. The following code is added to the *main* method:

Listing 7.2: Java code to configure the automatic model selection in AGFACAND

```
        BayesianOptimization bo = new BayesianOptimization(dl);
        bo.setDataShift(11);
        Model bestModel = bo.optimize();
```

A *BayesianOptimization* object is initialized with the *DataLoader* object as an argument. The *BayesianOptimization* object is configured to use a data shift of 11 which means that the predicted value will be 12 samples into the future. By default the data shift will be 0 which equals a single sample into the future. In this tutorial it is assumed that the model found by BO will be the best model for 1 sample ahead, 2 samples ahead etc all the way to 12 samples ahead. The other values for configuration of BO will be kept at default values, which means a delay value between 0 and 24, all possible models, all possible feature selection algorithm and hidden layer sizes between 100 and 1000. The possible delay value could be limited to a maximum of 12, but will be kept at default for this tutorial.

Finally, the model found by BO by calling the *optimize* method and saving the return value.

7.3 Calculating the final model and exporting

The output of BO is the configured model, but the model has not yet been trained. During the process of BO the model was evaluated using cross validation. This ensures that the returned model is the model with the best generalization performance. However, it also means that the model needs to be trained again as AGFACAND does not know if the model needs to be evaluated or exported. The *DataLoader* object is also configured with the optimal delay, so the only step is to train the model with all the data. The following code is added to the *main* method:

Listing 7.3: Java code to calculate the final model and export it

```
        dl.load();
    ModelContainer mc = new ModelContainer(dl);
    for (int i = 0; i < 11; i++) {
      Matrix in = dl.getInput().removeRowsEnd(i);
      Matrix out = dl.getOutput().removeRowsBeginning(i);
      bestModel.trainNetwork(in, out);
      mc.saveNetwork(i, bestModel);
    }
    mc.setDescription("SML temperature prediction");
    mc.save();
```

The data is loaded through the *load* method call. A *ModelContainer* object is initialized with the *DataLoader* object as an argument. The *ModelContainer* object is used to save the 11 individual models, one for each sample of the prediction horizon. The *DataLoader* is given as an argument such that the information from the header of the data file is saved along with the model. In a loop the data is shifted a single sample per loop in order to create a model for every sample of the prediction horizon. Once the data has been shifted the model is trained using the *trainNetwork* method from the *Model* interface as depicted in figure 6.2 and finally saved in the *ModelContainer* object. Before saving everything, a description of the exported model is added. Finally the *save* method is called in the *ModelContainer* object which saves all the models in the root of AGFACAND. The file's name will be the current UNIX timestamp and have an .xml extension.

A summary of the process so far is that the data has been loaded, the BO has been configured and performed an automatic model selection. The optimal model is then trained with all data to create 11 independent models, one for every sample of the prediction horizon from 1 sample into the future to 12 samples into the future. Finally the completed models are exported such that they can be used in another application. On a server with an Intel Xeon E5-2630 v4 @ 2.20 GHz, 64 GB of ram and Windows 2016 server the method completes in 294 minutes.

The final code of the *SMLTemperature* class is as follows:

Listing 7.4: Java code of the complete *SMLTemperature* class

```java
package dk.sdu.mmmi.agfacand.applications;

import dk.sdu.mmmi.agfacand.core.Model;
import dk.sdu.mmmi.agfacand.data.DataLoader;
import dk.sdu.mmmi.agfacand.data.Matrix;
import dk.sdu.mmmi.agfacand.data.ModelContainer;
import dk.sdu.mmmi.agfacand.optimization.BayesianOptimization;

public class SMLTemperature {
  public static void main(String[] args) {
    DataLoader dl = new DataLoader("data/sml.txt")
          .setSeparator('␣')
          .hasHeader()
          .ignoreColumn(0,1)
          .setOutputIndex(2);
    BayesianOptimization bo = new BayesianOptimization(dl);
    bo.setDataShift(11);
    Model bestModel = bo.optimize();
    dl.load();
    ModelContainer mc = new ModelContainer(dl);
    for (int i = 0; i < 11; i++) {
      Matrix in = dl.getInput().removeRowsEnd(i);
      Matrix out = dl.getOutput().removeRowsBeginning(i);
      bestModel.trainNetwork(in, out);
      mc.saveNetwork(i, bestModel);
    }
    mc.setDescription("SML␣temperature␣prediction");
    mc.save();
  }
}
```

7.4 Using and testing an exported model

After the model has been trained it can be used in other applications or it can be tested to document its generalization performance. Both cases start by loading the saved model. To keep it separate from the previous method, a new method named *testModel* is created in the *SMLTempetaure* class:

Listing 7.5: Java code to load the exported model

```java
public static void testModel() {
    ModelContainer mc = ModelContainer.load("sml.xml");
}
```

Again a *ModelContainer* object is initialized, but this time using a static method which is given the path to the xml file of the model. As the final model was saved with a timestamp as its name, it has been renamed to "sml.xml".

Using the exported model

To test the model, the *runNetwork* method is used which was described in chapter 6. The model from this example needs 22 parameters as inputs with the delay being 0, but this can be different when following the tutorial because of the random initial evaluations of the hyperparameter search space which can result in a different chosen delay when following this tutorial. In this example the model is run using a single sample, and therefore, AGFACAND can not delay the data. If the model is applied in another system, it is up to the developer of that system to provide delayed data for the model. For the sake of this example, an empty array is provided. The following code is added to the *testModel* method.

Listing 7.6: Java code to run the trained model with new data

```java
    for (int i = 0; i < 11; i++) {
      System.out.println(mc.getNetwork(i).runNetwork(new double[22]));
    }
```

Every sample of the prediction horizon is iterated through a loop, and the *runNetwork* method from the *Model* interface as depicted in figure 6.2 is called with the empty array. This means that 11 values will be printed out which are the predictions for the next 11 hours of the prediction horizon. Remember that the values will not hold any meaning, as the model is presented with an empty array.

Testing the exported model

In case of measuring the generalization, the previously trained model cannot be used as it has been trained with all the available data. Instead the model's configuration will be loaded from the xml file and the model will be trained again. This time the cross validation technique will be used such that the performance of the model is evaluated on unseen data. This means that the data set needs to be loaded again, as the data set is not saved in the exported file. The following code is added to the *testModel* method:

As the model needs to be trained again, the first model from the *ModelContainer* object is fetched. Then a *DataLoader* object is once again initialized, but this time using the *ModelContainer* object as an argument. This will automatically set all the

Listing 7.7: Java code to train, test and plot the performance of the model

```java
    Model model = mc.getNetwork(0);
    DataLoader dl = mc.getDataLoader().load();
    CrossValidationContainer cvc = new CrossValidationContainer(10,
        dl.getInput(), dl.getOutput());
    double sum = 0;
    for (int i = 0; i < cvc.getNumberOfFolds(); i++) {
      model.trainNetwork(cvc.getInputTraining(i), cvc.getOutputTraining(i));
      Matrix result = model.runNetwork(cvc.getInputTest(i));
      sum += StatisticsUtility.calculateRMSE(result.toPackedArray(),
          cvc.getOutputTest(i).toPackedArray());
      mc.saveNetworkOutput(0, new Matrix[] { result, cvc.getOutputTest(i) });
    }
    System.out.println(sum / ((double) cvc.getNumberOfFolds()));
    PlotterEngine pe = new PlotterEngine(mc);
    pe.createOutputPlot(new Function<Integer, String>() {
      public String apply(Integer t) {
        return t.toString();
      }}, 0);
```

required configurations for the *DataLoader* object. A *CrossValidationContainer* object is initialized with 10 bins and with the data from the *DataLoader* object. Only the first sample of the prediction horizon is investigated for this example. In order to investigate other samples, the data would have to be shifted as demonstrated earlier.

The cross validation bins are iterated in a loop, and for each iteration the model is trained with the training portion of the data for that cross validation bin. After the model has been trained using the *trainNetwork* from the *Model* interface it is run using the test portion with the *runNetwork* method from the *Model* interface and the RMSE is calculated and added to the *sum* variable. The last part of the loop saves the results in the *ModelContainer* object as it will enable plotting of the results. The results are overwritten for each iteration of the loop, which means that the results from the last bin will be plotted, but this will serve fine for this example.

After the loop the average RMSE is printed to the console. Then a *PlotterEngine* object is initialized with the *ModelContainer* object as an argument. The *PlotterEngine* object can make a variety of plots, but only the "output plot" will used for this example. The *createOutputPlot* method requires a function as an argument which is used to name the plot in the legend of the plot. The second argument is the index to use from the *ModelContainer* object, and is also the index that is given to the function of the first parameter. This allows naming each plot independently. On the same server as before, this code takes 40 minutes to run and returns the generalization performance of the model as well as a plot that shows both the model's output as well as the ideal output.

7.5 Summary

This chapter has demonstrated how to use AGFACAND on a real-life data set by going through the process from the raw data file to a final model. Every step of the process has been carefully explained line by line of the code. The final product of the tutorial is an exported model that can be used in other systems, as well as code to document the generalization performance of the model and to create plots of the model's performance. This tutorial allows a user to create a system model of the temperature in the dining room of the SML2010 data set. No ANN knowledge was required as the configuration and selection of the ANN was automatic.

CHAPTER

8 APPLICATIONS

The previous four chapters were about the theoretical background for the different parts in AGFACAND and the design and implementation of AGFACAND. As described in chapter 1, this doctoral project has employed the constructive research method. A practical research problem is a fundamental principle of that method and this chapter will review the research problems that have been faced as part of this doctoral project and what the outcome of these problems has been. The problems presented in this chapter has been solved without AGFACAND and have contributed to the development of AGFACAND. The sub-sections below are named after the published articles in which the results from the problems are presented. In each section a paragraph will cover how the specific problem has contributed to the development of AGFACAND. To emphasize the effectiveness of AGFACAND, the same problems will be attempted again using AGFACAND in its complete version.

8.1 Local Weather Forecast

Weather forecasts are not only used by people to dress appropriately; they are also used in dynamic control systems such as intelligent climate control of greenhouses. Furthermore, current weather data is used as input data in many cases of building modeling (Kalogirou, 2006). From these building models the future state of a building's indoor climate can be predicted by using weather forecasts. A general tendency observed at a greenhouse research facility in Aarslev, Denmark, is that a commercial weather forecast is pessimistic during the summer and optimistic during the winter, specifically with regards to illuminance prediction which is used to plan supplementary light in the greenhouse. For this reason a local and more precise weather forecast is desirable.

Improved Local Weather Forecast

Local weather stations with multiple sensors located at the greenhouses in Aarslev can provide data for a machine learning based approach to weather forecasting by predicting the weather parameters in the future. The local weather forecast needed is short term ($<$ 25 hours). The approach was to use a MLP with only previous values to predict future values. The results of this problem are presented in (Wollsen and Jørgensen, 2015).

The work on this problem laid the groundwork for AGFACAND by implementing the MLP which is described in chapter 3. To support the modeling, an early version of the Model interface were required, which is described in chapter 6. Inspired by MATLAB, the model in this preliminary version of the framework used a closed loop setup in order to achieve a longer prediction horizon than a single time step. In this closed loop setup there input to the model is the previous output and the output of the model is connected back to it self as an input. This means however, that any error in the prediction is propagated through the network again resulting in poor performance for longer prediction horizons. Additionally, the model used an online-mode of training where 14 days of training data was used to predict the 15th day. The entire data set is used by applying a sliding window technique. This approach might have resulted in an improved prediction performance as the training and test periods are close to each other in time and therefore potentially similar. The online-mode also requires constantly retraining the model which is computationally expensive.

Applying AGFACAND to the same problem reveals improved results. Due to the online-mode used in the paper, the results are not directly comparable, so for that reason the results have been recalculated with the same model configuration. The results are presented in table 8.1 and is the generalization performance measured by the RMSE.

Table 8.1: Comparison of results from the paper and AGFACAND for local weather forecasting. The values are RMSE values and 95 % confidence intervals.

	Paper results	**AGFACAND**
Short-term solar irradiance	$134.0\,\mathrm{W/m^2} \pm 21.5\,\mathrm{W/m^2}$	$88.5\,\mathrm{W/m^2} \pm 20.1\,\mathrm{W/m^2}$
Long-term solar irradiance	$157.8\,\mathrm{W/m^2} \pm 29.3\,\mathrm{W/m^2}$	$135.9\,\mathrm{W/m^2} \pm 26.1\,\mathrm{W/m^2}$
Short-term temperature	$3.09° \pm 1.03°$	$0.80° \pm 0.32°$
Long-term temperature	$3.89° \pm 0.59°$	$3.58° \pm 0.65°$

AGFACAND found an ESN with 505 hidden nodes and a delay of 24 to be optimal for the short-term solar irradiance forecast. For the long-term forecast an ESN network with 100 hidden nodes, RDESF and a delay of 0 was found to be optimal. For the temperature the optimal model was found to be GRNN with a delay of 6 for short-term prediction and an ESN with 100 hidden nodes, no feature selection and a delay of 0 for long-term prediction. That AGFACAND outperforms the paper's approach of the online-mode training is noteworthy as the approach used with AGFACAND uses more data for both training and testing. The increased amount data complicates both the learning process and the generalization effort but the model selection still finds a better model than what was used originally in the paper.

Automatic Feature selection

Following the work in (Wollsen and Jørgensen, 2015), more time was invested into weather forecasts. A reference year data set, which is a weather data set comprising 10 years of data, was used as a baseline comparison in (Wollsen and Jørgensen, 2015). However, only a single parameter was used for input and output in the paper. The data set contains parameters for cloud cover, solar radiation, illuminance, atmospheric pressure, relative humidity, temperature, wind speed and wind direction (Wang et al., 2013). It was discovered that the prediction performance greatly depended on which input features were chosen, and that a feature subset which felt intuitive was not actually the optimal. From this, further investigation suggested feature selection to be a critical functionality that should be implemented in AGFACAND. Further investigation into the field of feature selection revealed that filter-class feature selection algorithms were not automatic. Filter-class algorithms are nevertheless interesting because they typically rank the features based on correlation. If there is a high correlation between input features and the output, then the system can easily be modeled. This led to the development of the Ranked Distinct Elitism Selection Filter (RDESF) which was covered in chapter 4 and presented in (Wollsen, Hallam and Jørgensen, 2016). As described in chapter 6, the feature selection process is integrated into the Model interface and is interchangeable, similar to the modeling algorithm itself.

Revisiting the same two data sets from the paper with the full framework and comparing with backward selection shows the effectiveness of AGFACAND. Backward selection was the feature selection algorithm that performed best in the paper, but it is too slow for practical applications. The model was an MLP with 20 hidden nodes. The two data sets are the reference year data set (Wang et al., 2013) and the SML2010 data set (Zamora-Martínez et al., 2014) from the UCI Machine Learning Repository. Both data sets apply a prediction horizon up to 24 hours. Only the first and last predictions are used for the comparison as the error increases in relation to the prediction horizon, which was witnessed in (Wollsen and Jørgensen, 2015). The results of the comparison is presented in table 8.2. Once again, the presented values are RMSE values.

Table 8.2: Comparison of results from the paper and AGFACAND for the feature selection benchmark data sets. The values are RMSE values and 95 % confidence intervals

	Paper results	**AGFACAND**
SML short-term	$1.79° \pm 0.31°$	$0.03° \pm 0.01°$
SML long-term	$3.11° \pm 0.92°$	$2.49° \pm 0.73°$
Reference year short-term	$4.72° \pm 1.21°$	$0.55° \pm 0.18°$
Reference year long-term	$5.22° \pm 1.75°$	$3.29° \pm 0.96°$

The optimal model configuration for the SML data set was found to be a MLP with RDESF feature selection, 100 hidden nodes and a delay of 19 for the short-term prediction. For the long term prediction the optimal model was found to be an ESN with no feature selection, 100 hidden nodes and a delay of 0. Similarly for the reference year data set the optimal models were found to be linear with RDESF as feature selection and a delay of 24 and a ESN with 100 hidden nodes, no feature selection and no delay for the short-term and long-term respectively. The number of nodes is in the lower end of the spectrum which suggests a lower number increases the generalization performance by not overfitting the problem. Interestingly, ESNs are chosen for many of the long term prediction tasks but without any delays. Remember from chapter 3 that an ESN is a recurrent neural network which has memory abilities in its structure, which explains why ESNs do not need a delayed input. The final framework outperforms the original models in all data sets used for the feature selection paper which confirms the effectiveness of AGFACAND.

Summary

Modeling of weather patterns is a very challenging field for any kind of modeling and thus an interesting measure of performance of any modeling technique. Additionally, the knowledge gained from modeling weather patterns is applicable to other kinds of dynamical systems. The two papers that were published as part of the local weather forecast investigation and laid the foundation for large parts of the final framework. Additionally they opened up relevant research areas that added functionality to AGFACAND which will be covered in the next section.

8.2 Office Buildings

Office buildings are often modeled by both machine learning and mathematical methods. Office buildings are responsible for 19.3 % of the electrical consumption in Denmark, only surpassed by agricultural buildings (Dubois et al., 2016) and therefore are an area with great possibilities for adjusting energy consumption in Demand Response (DR) events. DR is the change of consumer electricity usage in response to changes in electricity price or production (Albadi and El-Saadany, 2008). With increased focus on DR it is necessary to have a model of the building in order to accommodate DR events properly. The model can either provide a baseline of the existing consumption or estimate the future consumption given a DR signal.

Demand Response in Commercial Buildings with an Assessable Impact on Occupant Comfort

As part of the research project Demand-Response Capacity Management in Commercial Buildings, a baseline ventilation power consumption model was needed for the GreenTech House in Vejle, Denmark. The GreenTech House is a working environment that houses a range of organizations and companies. It is a three-story building with 50 rooms comprising office spaces, a cafeteria, meeting rooms and restrooms. The baseline power consumption allows the system in the project to predict the amount of power consumption that is reduced or the "load shed" during a DR event. The ventilation system is responsible for a large part of the consumption but also has a high impact on occupant comfort, which was the context of the project. The project and its results are presented in (Kjærgaard et al., 2016).

The model applied to this problem is an MLP with RDESF applied and with 20 hidden nodes. Revisiting the same problem with the complete framework will be presented in the next section, as this and the next paper share the same data set. For the results of (Kjærgaard et al., 2016), a MLP was used with 46 hidden nodes which was decided using the rule of thumb that says to take the average of the number of inputs and the number of outputs of the MLP.

Influential Factors for Accurate Prediction in a Demand Response Context

Based on the work in (Kjærgaard et al., 2016), additional time was invested in investigating the factors that influenced the prediction performance of the ventilation power consumption. Three experiments were conducted that tested the influence of data, time and algorithms chosen. The results are presented in (Wollsen, Kjærgaard and Jørgensen, 2016).

From this work, the framework was extended with additional models: ADALINE, ESN, ELM, GRNN, Linear and SVR, all presented in chapter 3. The results showed that different algorithms provide different results, but also that the different algorithms have different complexities. The inclusion of additional models allows the framework to model a wider variety of problems. Eventually, this also led to the implementation of BO which was covered in chapter 5.

Applying AGFACAND on the data from the GreenTech House gives the results presented in table 8.3. The input data for the problem is the previous ventilation power consumption and weather station data such as relative humidity, temperature, binary rain indicator, illuminance, wind direction and wind speed.

Table 8.3: Comparison of results from the papers and AGFACAND for the GreenTech House data set. The values are RMSE values and 95 % confidence intervals.

	GreenTech House data set
DR paper	0.40 kWh ± 0.11 kWh
AGFACAND (DR)	0.16 kWh ± 0.07 kWh
Influential factor papers	2.44 kWh ± 0.84 kWh
AGFACAND	1.65 kWh ± 0.51 kWh

The two applications are different in terms of resolution. (Kjærgaard et al., 2016) used a time resolution of 5 minutes whereas (Wollsen, Kjærgaard and Jørgensen, 2016) used a time resolution of 1 hour. This difference is the reason for AGFACAND appearing twice in table 8.3. The prediction horizon was 3 hours for both cases. For comparison with (Kjærgaard et al., 2016), the optimal model was found to be an ESN with 100 hidden nodes. For comparison with (Wollsen, Kjærgaard and Jørgensen, 2016), the optimal model was found to be a MLP with 100 hidden nodes. Once again AGFACAND outperforms the models that were used in the respective papers to once again confirm its effectiveness. As with the feature selection benchmark data sets, small numbers of nodes in the hidden layer are selected as optimal, which suggests that overfitting occurs at higher numbers of hidden nodes and that the relation between input and output can be modeled satisfactorily by a simpler model.

Summary

Modeling of buildings has a wide range of applications from control schemes to DR. In the case of AGFACAND, the modeling has helped discover the energy saved in a DR situation. With the added functionality to AGFACAND from these cases, AGFACAND is essentially complete. Only additional finished touches and optimizations were added to AGFACAND afterwards.

CHAPTER

9 EVALUATION

This chapter contains an evaluation of the work in this doctoral project. The framework is evaluated against the criteria set forth in chapter 1 as well as choices and assumptions made with regards to the implementation of AGFACAND. Finally, the generalization of the framework is evaluated against the applications for which AGFACAND has been developed and tested.

9.1 Evaluation of framework criteria

The framework is evaluated against the criteria set forth in chapter 1. By individually evaluating the criteria, areas that require further focus for future work will be revealed.

Integration of many types of ANNs

Multiple types of ANNs are supported in AGFACAND through the Model interface described in chapter 6. The interface describes common behavior for any type of model (machine learning or mathematical) and enables configuration of a single model parameter (nodes in the hidden layer). This however poses a risk as other models might require more configurable parameters. Random initialization is used in many types of ANNs but the effect of randomization has been greatly reduced through the use of network ensembles as described in chapter 3. The implementation of AGFACAND is based upon Encog (Heaton, 2015*a*), which is a community-driven neural network library. Any additional ANNs that are implemented in AGFACAND are validated through example data sets such as linear relations or a tunable sine wave. It is possible, however, that implementation errors exist even if the ANNs pass the validation tests. Other implementations necessary to support AGFACAND such as those of a matrix, data normalization, statistics etc, are all validated through unit tests which contain hard-coded data that are externally validated through applications like MATLAB and Mathematica. This criteria is therefore considered met.

Automatic configuration of ANNs

The models in AGFACAND are automatically configured through the optimization performed in BO, which was covered in chapter 5. The configuration covers both the internal model parameter (number of nodes in the hidden layer), a feature selection step, how much to delay input data and which model to use. By automating the choice of which model to use, any user preference or user experience is put aside and the BO automatically finds the optimal model for the given data set. BO is a well-known technique for optimization of expensive functions. BO was also just recently (September 2016) added to MATLAB.

The main problem of the optimization is that the parameters that need to be optimized are heavily dependent on each other. This means that any change in one parameter, for example the model type, will heavily influence the optimal value of the other parameters. The dependence between the parameters has been dealt with using a cartesian product of the parameter values, and thereby creating a hyperparameter search space of every possible combination of every parameter. One concern is the validity of the implementation of BO in the framework, although the implementation has been validated by examples in both Python and MATLAB. This criteria is considered met.

Support for multiple time resolutions (sampling rates)

Multiple time resolutions are supported in AGFACAND only by multiple data sets. No assumptions are made about the time resolution in a data set and the Matrix class holds the data and time information blindly. The time information of the data set is only used to add time parameters such as time of day, time of year etc and for plotting. In other words, AGFACAND ignores the time resolution in the data set and assumes that the data set is a sequence. Although this assumption is made clear, it means that preprocessing by the user might be required for a data set to live up to these assumptions. Supporting multiple time resolutions in a single data set is not planned in AGFACAND as tools already exists for time series data. One example of such is Panda[1] for Python that allows for fast data preprocessing of time series. Because AGFACAND does not make any assumptions towards a specific time resolution and supports multiple time resolutions via multiple data sets, this criteria is considered met.

Support for varying prediction horizons

Varying prediction horizons are well supported in AGFACAND and are one of the only necessary configurations to AGFACAND. By shifting the output a sample in relation to the input, the same input can be used to create a prediction a sample further into the future. Repeating this process allows AGFACAND to create both short-term and long-term prediction models. The ModelContainer described in chapter 6 can save independent models for every sample of the prediction horizon, such that when the models are executed, all models are executed in parallel and thereby producing a sequence of output values — one for every sample of the prediction horizon. Therefore this criteria is considered met.

Export of a trained model

The trained models are exported in an XML format which is a good compromise between readability and size. For a model trained on a year of data in an hourly

[1]http://pandas.pydata.org/

resolution, the exported file size is around 200 MB. This exported file contains 50 independently trained ANNs from the ensemble with their respective normalization and feature selection. Furthermore, the chosen export format has better support for changes in the source code compared to the standard Java serialization. The library that completes the XML serialization is used by a large community and considered bug free. Therefore this criteria is considered met.

Usable by developers without ANN expertise

The only configuration requirements for applying AGFACAND on a data set are the path to the data set and the prediction horizon; any other configuration is optional. Therefore AGFACAND requires absolutely no expert knowledge with regards to ANNs or feature selection. However, it does require basic programming skills in Java. The tutorial in chapter 7 confirms that no ANN expertise is required to produce a system model. However, user testing is required before this criteria can be considered met.

9.2 Generalization of results

This doctoral project presents a generic framework for applying ANNs and similar techniques on data sets with automatic configuration. The external validity of AGFACAND has been demonstrated by both developing AGFACAND through real-life applications, but also revisiting the same applications with the completed framework. Since the applications vary in data origin, time resolution of the data and output parameter, AGFACAND can be considered generic. The applications presented in chapter 8 confirms that the automated workflow by AGFACAND outperforms the manual workflow of a researcher with the necessary expert knowledge. It is very possible that tedious tuning of a specific ANN will achieve a better result on a specific data set in literature than that of AGFACAND, but in such cases it will always be possible to implement that ANN, override the configuration in AGFACAND and still receive the other benefits from AGFACAND such as export and testing of the model.

Optimality of automatic model selection

The optimal model selection by BO is not the true optimal model of the problem. Strict optimality can only be guaranteed by performing a grid-search where every possible combination of model and configuration is tested. A grid-search is however not feasible as the computation time would be far too big. BO offers a satisfactory trade-off between optimality and computation time, which is also further confirmed by its extensive use in literature. The fact that AGFACAND does not produce the true optimal model might mean that an ANN expert within a single type of ANN can produce a better performing model for a given problem, but for the average ANN expert or non-ANN expert, the model produced by AGFACAND will be sufficient. This has been confirmed by the results presented in chapter 8.

Programming language

AGFACAND is implemented in Java as this is the main programming language by the author as well as the language used by the other systems that the models from

AGFACAND was produced for. Java is not a commonly used for neural networks because of the computational speed compared to other hardware-near languages such as C or C++. AGFACAND and the models produced by AGFACAND can be used in other language through the Java Native Interface (JNI).

Chapter

10 Future Research

This doctoral project covers the development of a generic framework for applying ANNs on data sets with automatic configuration. This chapter on future research will cover specific functionality that should be supported by a future version of AGFACAND as well as areas that might be interesting to investigate.

Inclusion of additional ANNs could further improve the generality of AGFACAND. DNNs are becoming more and more used and are showing groundbreaking results (Schmidhuber, 2015). DNNs are defined as ANNs with multiple hidden layers. It was attempted to include DNNs in AGFACAND; however, neither backpropagation, Levenberg-Marquardt algorithm nor RPROP could produce a stable training, which is why DNNs were not included in AGFACAND. Additional attempts to include DNNs in AGFACAND should be made part of future work.

Big Data is the big buzz-word these years, and is classified as data with high volume/high dimensionality, high variety, high velocity, high variability and high veracity (Hilbert, 2016). Big Data complicates the learning task in many Ways, as some ANNs simply are not suited for Big Data. Feature selection can reduce the dimensionality of Big Data, but feature selection is still an optional step in AGFACAND. The data loading mechanism in AGFACAND has been prepared for Big Data as it is buffered, but the entire data set is still kept in memory and presented to the ANN. Whether or not AGFACAND can handle Big Data is unknown and remains to be studied as part of future work.

Data storage or data management also need to be considered. For now, AGFACAND only supports data coming from local text based files. This should be expanded with data management sources such as SQL or data management systems such as sMAP[1] that includes actuation in systems where the model would be used for control. Other formats such as Hierarchical Data Format (HDF)[2] are also viable options.

As mentioned in chapter 5, the covariance kernel should be replaced with its automatic relevance determination (ARD) equivalent. The ARD version supports multiple scales for each variable in the hyperparameter vector, such that the variance of each variable is considered individually. As mentioned in chapter 9, AGFACAND should be expanded to support multiple parameters in a model, which also requires a change in the implementation of BO.

Finally, AGFACAND needs user testing to finally confirm if AGFACAND truly is user-friendly and if it usable by developers without ANN expertise.

[1]https://people.eecs.berkeley.edu/s̃tevedh/smap2/

[2]https://www.hdfgroup.org/

CHAPTER

11 CONCLUSION

This doctoral project deals with the lack of a generic framework for applying Artificial Neural Networks (ANNs) on data sets without requiring expert knowledge within ANNs.

The generic framework AGFACAND that has been developed through this doctoral project combines existing approaches and methodologies in order to create a solution to this problem. AGFACAND includes 8 different modeling algorithms, 3 feature selection algorithms and automatic configuration on top of that. The modeling algorithms will perform machine learning regression which results in system models or a prediction system. The feature selection algorithms can reduce the dimensionality of the data set and thereby its training complexity to enable faster computation. Additionally, the feature selection algorithms can remove data parameters that only act as noise to the modeling process, which in the end results in better performing models. One of the feature selection algorithms is Ranked Distinct Elitism Selection Filter (RDESF) which is a novel filter-class algorithm developed as part of this doctoral project. The automatic configuration is performed using Bayesian Optimization (BO) which is an optimization method for expensive function evaluations like ANNs.

AGFACAND is implemented in Java with core functionality coming from Encog (Heaton, 2015*b*). The modeling algorithms in AGFACAND implement an interface that allows the algorithms to be interchanged and automatically supports feature selection, configuration of parameters and export of models. The feature selection algorithms also implement an interface so they can be interchanged as well. The interfaces in AGFACAND provide a strict contract towards the algorithms and allow for further expansion of AGFACAND.

Real-life applications have been the core of the development of AGFACAND. The individual parts of AGFACAND has been developed towards real-life applications as described in 8. This ensures that the individual parts of AGFACAND are well-tested and that AGFACAND is generic as the applications vary in data origin, time resolution of the data and output parameter. The completed AGFACAND has been evaluated on the same applications to confirm its effectiveness. AGFACAND outperformed the results that were published as part of the work on these applications in all cases. The results from the publications followed the traditional manual workflow of applying ANN as presented in chapter 2.

Future research will further improve the functionality of AGFACAND with new ANNs and with additional functionality to deal with Big Data. AGFACAND will also continuously be evaluated on new applications.

BIBLIOGRAPHY

Albadi, M. H. and El-Saadany, E. (2008), 'A summary of demand response in electricity markets', *Electric power systems research* **78**(11), 1989–1996.

Aras, S. and Kocakoç, İ. D. (2016), 'A new model selection strategy in time series forecasting with artificial neural networks: Ihts', *Neurocomputing* **174**, 974–987.

Beccali, M., Cellura, M., Brano, V. L. and Marvuglia, A. (2004), 'Forecasting daily urban electric load profiles using artificial neural networks', *Energy conversion and management* **45**(18), 2879–2900.

Bergstra, J., Yamins, D. and Cox, D. (2013), Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures, *in* 'International Conference on Machine Learning', pp. 115–123.

Berndt, D. J. and Clifford, J. (1994), Using dynamic time warping to find patterns in time series., *in* 'KDD workshop', Vol. 10, Seattle, WA, pp. 359–370.

Bingham, E. and Mannila, H. (2001), Random projection in dimensionality reduction: applications to image and text data, *in* 'Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining', ACM, pp. 245–250.

Bishop, C. M. (1995), *Neural networks for pattern recognition*, Oxford university press.

Bishop, C. M. (2006), 'Pattern recognition', *Machine Learning* **128**, 1–58.

Boden, M. (2002), 'A guide to recurrent neural networks and backpropagation', *the Dallas project* .

Borchani, H., Varando, G., Bielza, C. and Larrañaga, P. (2015), 'A survey on multi-output regression', *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **5**(5), 216–233.

Brent, R. P. (1973), *Algorithms for minimization without derivatives*, Prentice-Hall.

Brochu, E., Cora, V. M. and De Freitas, N. (2010), 'A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning', *arXiv preprint arXiv:1012.2599* .

Carpenter, G. A. (1989), 'Neural network models for pattern recognition and associative memory', *Neural networks* **2**(4), 243–257.

Cavanaugh, J. E. (1997), 'Unifying the derivations for the akaike and corrected akaike information criteria', *Statistics & Probability Letters* **33**(2), 201 – 208.

Chen, S., Billings, S. and Grant, P. (1990), 'Non-linear system identification using neural networks', *International journal of control* **51**(6), 1191–1214.

Cortes, C. and Vapnik, V. (1995), 'Support-vector networks', *Machine learning* **20**(3), 273–297.

Cover, T. M. and Thomas, J. A. (1991), *Elements of Information Theory*, John Wiley & Sons.

Cruse, H. (2006), 'Neural networks as cybernetic systems', *Brain, minds, and media. See http://www. brains-minds-media. org/archive/289* .

Dubois, M.-C., de Boer, J., Deneyer, A., Fuhrmann, P., Geisler-Moroder, D., Hoier, A., Jakobiak, R., Knoop, M., Koga, Y., Osterhaus, W. et al. (2016), Building stock distribution and electricity use for lighting, *in* 'A Technical Report of Subtask D (case Studies), T50. d1', Fraunhofer-Institut für Bauphysik.

Estévez, P. A., Tesmer, M., Perez, C. A. and Zurada, J. M. (2009), 'Normalized mutual information feature selection', *IEEE Transactions on Neural Networks* **20**(2), 189–201.

Gentle, J. E. (2009), *Computational statistics*, Vol. 308, Springer.

Gibert, K., Sànchez-Marrè, M. and Codina, V. (2010), 'Choosing the right data mining technique: classification of methods and intelligent recommendation', *Iemss.Org* .

Granger, C. (1988), 'Some recent development in a concept of causality', *Journal of Econometrics* **39**, 199 – 211.

Guyon, I. and Elisseeff, A. (2003), 'An Introduction to Variable and Feature Selection', *The Journal of Machine Learning Research* **3**, 1157–1182.

Hansen, L. K. and Salamon, P. (1990), 'Neural network ensembles', *IEEE transactions on pattern analysis and machine intelligence* **12**(10), 993–1001.

Heaton, J. (2015*a*), 'Encog: Library of interchangeable machine learning models for java and c#', *Journal of Machine Learning Research* **16**, 1243–1247.

Heaton, J. (2015*b*), 'Encog: Library of interchangeable machine learning models for java and c#', *Journal of Machine Learning Research* **16**, 1243–1247.

Hilbert, M. (2016), 'Big data for development: A review of promises and challenges', *Development Policy Review* **34**(1), 135–174.

Hornik, K. (1991), 'Approximation capabilities of multilayer feedforward networks', *Neural networks* **4**(2), 251–257.

Huang, G.-B., Zhu, Q.-Y. and Siew, C.-K. (2006), 'Extreme learning machine: theory and applications', *Neurocomputing* **70**(1), 489–501.

Hwang, C.-L. and Yoon, K. (1981), *Multiple attribute decision making: methods and applications*, Springer.

Jaeger, H. (2001), 'The "echo state" approach to analysing and training recurrent neural networks-with an erratum note', *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* **148**(34), 13.

Jain, R. K., Smith, K. M., Culligan, P. J. and Taylor, J. E. (2014), 'Forecasting energy consumption of multi-family residential buildings using support vector regression: Investigating the impact of temporal and spatial monitoring granularity on performance accuracy', *Applied Energy* **123**, 168–178.

Jones, D. R., Schonlau, M. and Welch, W. J. (1998), 'Efficient global optimization of expensive black-box functions', *Journal of Global optimization* **13**(4), 455–492.

Jordanov, I. and Petrov, N. (2014), Sets with incomplete and missing data—nn radar signal classification, *in* 'Neural Networks (IJCNN), 2014 International Joint Conference on', IEEE, pp. 218–224.

Kalogirou, S. A. (2006), 'Artificial neural networks in energy applications in buildings', *International Journal of Low-Carbon Technologies* **1**(3), 201–216.

Kasanen, E., Lukka, K. and Siitonen, A. (1993), 'The constructive approach in management accounting research', *Journal of management accounting research* **5**, 243.

Kjærgaard, M. B., Arendt, K., Clausen, A., Johansen, A., Jradi, M., Jørgensen, B. N., Nelleman, P., Sangogboye, F. C., Veje, C. and Wollsen, M. G. (2016), Demand response in commercial buildings with an assessable impact on occupant comfort, *in* 'Smart Grid Communications (SmartGridComm), 2016 IEEE International Conference on', IEEE, pp. 447–452.

Kourentzes, N., Barrow, D. K. and Crone, S. F. (2014), 'Neural network ensemble operators for time series forecasting', *Expert Systems with Applications* **41**(9), 4235–4244.

Kramer, M. a. (1991), 'Nonlinear Principal Component Analysis Using Autoassociative Neural Networks', *AIChE Journal* **37**(2), 233–243.

Little, R. J. and Rubin, D. B. (2014), *Statistical analysis with missing data*, John Wiley & Sons.

Lizotte, D. J. (2008), *Practical bayesian optimization*, University of Alberta.

Lukoševičius, M. (2012), A practical guide to applying echo state networks, *in* 'Neural networks: Tricks of the trade', Springer, pp. 659–686.

Lukoševičius, M., Jaeger, H. and Schrauwen, B. (2012), 'Reservoir computing trends', *KI-Künstliche Intelligenz* **26**(4), 365–371.

Maqsood, I., Khan, M. R. and Abraham, A. (2004), 'An ensemble of neural networks for weather forecasting', *Neural Computing & Applications* **13**(2), 112–122.

Morando, S., Jemei, S., Gouriveau, R., Zerhouni, N. and Hissel, D. (2014), Fuel cells remaining useful lifetime forecasting using echo state network, *in* ‘Vehicle Power and Propulsion Conference (VPPC), 2014 IEEE’, IEEE, pp. 1–6.

Oneto, L., Pilarz, B., Ghio, A. and Anguita, D. (2015), Model selection for big data: Algorithmic stability and bag of little bootstraps on gpus, *in* ‘Proceedings’, Presses universitaires de Louvain, p. 261.

Pearson, K. (1895), ‘Note on regression and inheritance in the case of two parents’, *Proceedings of the Royal Society of London* **58**, 240–242.

Pearson, K. (1901), ‘Liii. on lines and planes of closest fit to systems of points in space’, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **2**(11), 559–572.

Phillips, J. M. and Venkatasubramanian, S. (2011), ‘A gentle introduction to the kernel distance’, *arXiv preprint arXiv:1103.1625* .

Powell, K. M., Sriprasad, A., Cole, W. J. and Edgar, T. F. (2014), ‘Heating, cooling, and electrical load forecasting for a large-scale district energy system’, *Energy* **74**, 877–885.

Rasmussen, C. E. (2006), Gaussian processes covariance functions and classification, Technical report, Technical report.

Rasmussen, C. E. and Williams, C. K. (2006), *Gaussian processes for machine learning*, Vol. 1, MIT press Cambridge.

Riedmiller, M. and Braun, H. (1993), A direct adaptive method for faster backpropagation learning: The rprop algorithm, *in* ‘Neural Networks, 1993., IEEE International Conference on’, IEEE, pp. 586–591.

Robbins, H. and Monro, S. (1951), ‘A stochastic approximation method’, *The annals of mathematical statistics* pp. 400–407.

Robert May, G. D. and Maier, H. (2011), *Review of Input Variable Selection Methods for Artificial Neural Networks*, InTech.

Rosenblatt, F. (1961), Principles of neurodynamics. perceptrons and the theory of brain mechanisms, Technical report, CORNELL AERONAUTICAL LAB INC BUFFALO NY.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1988), ‘Learning representations by back-propagating errors’, *Cognitive modeling* .

Schmidhuber, J. (2015), ‘Deep learning in neural networks: An overview’, *Neural networks* **61**, 85–117.

Şenkal, O. (2010), ‘Modeling of solar radiation using remote sensing and artificial neural network in turkey’, *Energy* **35**(12), 4795–4801.

Shanmugan, K. S. and Breipohl, A. M. (1988), *Random signals: detection, estimation, and data analysis*, Wiley.

Shannon, C. E. (2001), 'A mathematical theory of communication', *SIGMOBILE Mob. Comput. Commun. Rev.* **5**(1), 3–55.

Shlens, J. (2014), 'A tutorial on principal component analysis', *arXiv preprint arXiv:1404.1100* .

Snoek, J., Larochelle, H. and Adams, R. P. (2012), Practical bayesian optimization of machine learning algorithms, *in* 'Advances in neural information processing systems', pp. 2951–2959.

Song, Q., Ni, J. and Wang, G. (2013), 'A fast clustering-based feature subset selection algorithm for high-dimensional data', *IEEE transactions on knowledge and data engineering* **25**(1), 1–14.

Spearman, C. (1904), 'The proof and measurement of association between two things', *The American journal of psychology* **15**(1), 72–101.

Specht, D. F. (1991), 'A general regression neural network', *IEEE transactions on neural networks* **2**(6), 568–576.

Stanley, K. O. and Miikkulainen, R. (2002), 'Evolving neural networks through augmenting topologies', *Evolutionary computation* **10**(2), 99–127.

Thiele, B. and Henriksson, D. (2011), Using the functional mockup interface as an intermediate format in autosar software component development, *in* '8th Modelica Conference'.

Thornton, C., Hutter, F., Hoos, H. H. and Leyton-Brown, K. (2013), Auto-weka: Combined selection and hyperparameter optimization of classification algorithms, *in* 'Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining', ACM, pp. 847–855.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K. and Lang, K. J. (1989), 'Phoneme recognition using time-delay neural networks', *IEEE transactions on acoustics, speech, and signal processing* **37**(3), 328–339.

Wan, C., Xu, Z., Pinson, P., Dong, Z. Y. and Wong, K. P. (2014), 'Probabilistic forecasting of wind power generation using extreme learning machine', *IEEE Transactions on Power Systems* **29**(3), 1033–1044.

Wang, P. G., Mikael, S., Nielsen, K. P., Wittchen, K. B. and Kern-Hansen, C. (2013), 'Reference climate dataset for technical dimensioning in building, construction and other sectors', *DMI Technical reports* .

Wei, C.-C. (2012), 'Rbf neural networks combined with principal component analysis applied to quantitative precipitation forecast for a reservoir watershed during typhoon periods', *Journal of Hydrometeorology* **13**(2), 722–734.

Whiteson, S., Stone, P., Stanley, K. O., Miikkulainen, R. and Kohl, N. (2005), Automatic feature selection in neuroevolution, *in* 'GECCO'.

Whitle, P. (1951), *Hypothesis testing in time series analysis*, Vol. 4, Almqvist & Wiksells.

Widrow, B. and Lehr, M. A. (1990), '30 years of adaptive neural networks: perceptron, madaline, and backpropagation', *Proceedings of the IEEE* **78**(9), 1415–1442.

Wollsen, M. G., Hallam, J. and Jørgensen, B. N. (2016), Novel automatic filter-class feature selection for machine learning regression, *in* 'INNS Conference on Big Data', Springer, pp. 71–80.

Wollsen, M. G. and Jørgensen, B. N. (2015), Improved local weather forecasts using artificial neural networks, *in* 'Distributed Computing and Artificial Intelligence, 12th International Conference', Springer, pp. 75–86.

Wollsen, M. G., Kjærgaard, M. B. and Jørgensen, B. N. (2016), Influential factors for accurate load prediction in a demand response context, *in* 'Technologies for Sustainability (SusTech), 2016 IEEE Conference on', IEEE, pp. 9–13.

Xue, B., Zhang, M. and Browne, W. N. (2013), 'Particle swarm optimization for feature selection in classification: A multi-objective approach', *IEEE transactions on cybernetics* **43**(6), 1656–1671.

Yang, J., Rivard, H. and Zmeureanu, R. (2005), 'On-line building energy prediction using adaptive artificial neural networks', *Energy and buildings* **37**(12), 1250–1259.

Zamora-Martínez, F., Romeu, P., Botella-Rocamora, P. and Pardo, J. (2014), 'On-line learning of indoor temperature forecasting models towards energy efficiency', *Energy and Buildings* **83**, 162 – 172.

Zhang, G., Eddy Patuwo, B. and Y. Hu, M. (1998), 'Forecasting with artificial neural networks: The state of the art', *International Journal of Forecasting* **14**(1), 35–62.

Zhang, X.-l. and He, G.-g. (2007), 'Forecasting approach for short-term traffic flow based on principal component analysis and combined neural network', *Systems Engineering-Theory & Practice* **27**(8), 167–171.