GROWING SELF-ORGANIZING TREE-BASED KERNEL SMOOTHER FOR MACHINE LEARNING AND DATA MINING

by

Bohan Zheng

M.Eng., University of New South Wales, Australia, 2013

B.Sc., Yanshan University, P.R.China, 2011

A Thesis

presented to Ryerson University

in partial fulfilment of the requirements for the degree of

Master of Applied Science in the

Program of Electrical and Computer Engineering

Toronto, Ontario, Canada

AUTHOR'S DECLARATION

Abstract

Growing Self-Organizing Tree-Based Kernel Smoother for Machine Learning and Data Mining

© Bohan Zheng, 2016

Master of Applied Science

Electrical and Computer Engineering

Ryerson University

With Internet of Things (IoT) being prevalently adopted in recent years, traditional machine learning and data mining methods can hardly be competent to deal with the complex big data problems if applied alone. However, hybridizing those who have complementary advantages could achieve optimized practical solutions. This work discusses how to solve multivariate regression problems and extract intrinsic knowledge by hybridizing Self-Organizing Maps (SOM) and Regression Trees. A dual-layer SOM map is developed in which the first layer accomplishes unsupervised learning and then regression tree layer performs supervised learning in the second layer to get predictions and extract knowledge. In this framework, SOM neurons serve as kernels with similar training samples mapped so that regression tree could achieve regression locally. In this way, the difficulties of applying and visualizing local regression on high dimensional data are overcome. Further, we provide an automated growing mechanism based on a few stop criteria without adding new parameters. A case study of solving Electrical Vehicle (EV) range anxiety problem is presented and it demonstrates that our proposed hybrid model is quantitatively precise and interpretive.

key words: Multivariate Regression, Big Data, Machine Learning, Data Mining, Self-Organizing Maps (SOM), Regression Tree, Electrical Vehicle (EV), Range Estimation, Internet of Things (IoT)

Acknowledgements

I would like to thank my supervisor Dr. Lian Zhao of the Department of Electrical and Computer Engineering at Ryerson University. The door to Prof. Zhao's office was always open whenever I ran into a trouble spot or had a question about my research or writing.

I would also like to thank my co-supervisor Dr. Peter He for his encouragement, corrections and insightful comments. He consistently allowed this thesis to be my own work, but steered me in the right the direction whenever he thought I needed it.

Also, I must express my very profound gratitude to my wife and daughter for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Finally, the author sincerely acknowledges the support from Ontario Centre of Excellence (OCE) of Canada and RigMaster International Inc. under Grant number OCE # 23975.

Contents

List of Tables

List of Figures

List of Acronyms

AI	Artificial Intelligence
KRR	Knowledge Representation and Reasoning
KB	Knowledge Base
KDD	Knowledge Discovery in Databases
IoT	Internet of Things
SOM	Self-Organizing Maps
CART	Classification and Regression Trees
EV	Electrical Vehicle
SoC	State of Charge
BRC	Battery Residual Capacity
k-NN	k-Nearest Neighbours
RBF	Radial Basis Function
PCA	Principal Components Analysis
PCR	Principal Components Regression
GHSOM	Growing Hierarchical Self-Organizing Maps
BMU	Best Matched Unit
RTSOM	Regression Tree-based Self-Organizing Maps
SS	Sum of Squares
RL	Reinforcement Learning

Chapter 1

Introduction

The central problems (or goals) of Artificial Intelligence (AI) research include knowledge representation, reasoning, planning, learning, perception and the ability to control [1]. Currently popular approaches include statistical methods, computational intelligence and traditional symbolic AI.

Knowledge representation and reasoning (KRR) are central to AI research. Many of the problems to be solved by machines will require extensive knowledge about the world. Among the things that AI needs to represent are: objects, properties, categories and relations between objects [1]; situations, events, states and time; causes and effects; knowledge about knowledge (what we know about what other people know); and many other less well researched domains. Rather than general problem solvers, by the end of last century, AI changed its focus to logical, knowledge-based approach ,*i.e.* expert systems, that could match human competence on a specific task, which is still popular in practice. Expert systems gave us the terminology still in use today where AI systems are divided into a Knowledge Base (KB) with facts about the world and rules and an inference engine that applies the rules to the knowledge base in order to answer questions and solve problems. However, in most expert systems, just as the name indicated, the knowledge base and inference rules are usually provided by domain experts. Therefore the knowledge base tended to be a fairly flat structure, essentially assertions about the values of variables used by the rules.

In recent years, machine learning and data mining have become increasingly popular in the literature. Data mining involves discovering novel, interesting, and potentially useful patterns from large data sets and applying algorithms to the extraction of hidden information. Many other terms are used for data mining, for example, knowledge discovery (mining) in databases (KDD), knowledge extraction, data/pattern analysis, data archeology, data dredging, and information harvesting. The objective of any data mining process is to build an efficient predictive or descriptive model of a large amount of data that not only best fits or explains it, but is also able to generalize to new data. Based on a broad view of data mining functionality, data mining is the process of discovering interesting knowledge from large amounts of data stored in either databases, data warehouses, or other information repositories [2].

1.1 Problem Statement

Thanks to the research of machine learning and data mining methods, the knowledge could be derived and represented from raw information where the knowledge base of expert systems could be automatically built. Although computing statistical learning and data mining could provide some insights of the data, they tend to lose the capacity of predicting accuracy and interpreting when it comes to the big data problem.

Big Data starts with large-volume, heterogeneous, autonomous sources with distributed and decentralized control, and seeks to explore complex and evolving relationships among data. These characteristics make it an extreme challenge for discovering useful knowledge from the Big Data. For example, in Internet of Things (IoT), different sensors or data sources generate only an aspect of the whole view. The knowledge hidden underneath the data might be so multi-disciplinary that experts can hardly aggregate their domains and provide effect decision support or plan for future actions.

Big Data problems offer exciting opportunities that simple data representations are incapable of achieving. One of the fundamental characteristics of the Big Data is the huge volume of data represented by heterogeneous and diverse dimensionalities. Also, while the volume of the Big Data increases, so do the complexity and the relationships underneath the data. The correlations between individuals inherently complicate the whole data representation and any reasoning process on the data. Further, in the circumstances of big data, it is almost certain that most traditional mining algorithms cannot be applied directly to function well and obtain the ideal results, for example predicting accuracy.

1.2 Objectives and Contributions

Motivated by the above mentioned Big Data problems, in this thesis, we attempt to deal with high-dimensional (unstructured) unlabelled data in order to provide future action support (prediction or estimations) and knowledge discovery. In other words, we are trying to build the bridge connecting the state-of-the-art of data mining methods to conventional Knowledge-Based Expert Systems. It is important to note that aimed our attention at dealing with continuous data space, or targeting variables; and for this reason, we focus our study regression analysis.

Our main contribution in this thesis is that we develop a hybrid self-learning model for regression analysis and data mining. Specifically, the contribution of this thesis includes 3 parts: (1) Self-Organizing Maps (SOM) is adopted for data kernelization and preparation of further smoothing; (2) Classification and Regression Trees (CART) serve as the local bin smoother; (3) a growing mechanism is integrated to the hybrid model to select the right parameters involved.

1.3 Case Study: Electrical Vehicle (EV) Range Estimation

Electrical Vehicles (EVs) are regarded as promising replacements of conventional fossil fuel based vehicles in order to reduce carbon emissions and pollution. However, despite the global interest and the investments worldwide, the user acceptance level is still low. Range Anxiety, among others, is considered to be one of the major barriers to large scale adoption of EVs [3], which is defined as the possibility of losing power and being shut down in the middle of a trip.

The main strategies existing to alleviate range anxiety among electric car drivers include the deployment of extensive charging infrastructure (e.g. EV charging network [4]), the development of higher battery capacity [5], battery swapping technology [6], use of range extenders [7]. These solutions, admittedly, are attempting to tackle the problem directly with power electronics technology. However, they could raise the high cost issue instead, which might retard the market growth even more.

The range fear, as a matter of fact, mainly comes from the lack of precise information about power consumption of the trips, especially for new EV drivers. A recent survey in Electrifying Vehicles: Insights From The Canadian Plug-in Electric Vehicle Study [8], suggests that drivers already familiar with electric cars are far more willing than the general public to buy a battery-electric vehicle most likely because they understand how many miles a day they actually drive, and how well a battery car of a given range will meet those needs. However, according to the EV300 Program carried by Fleetwise (funded by Toronto Atmospheric Fund), novice EV drivers consistently left themselves a 50-km (30-mile) buffer of unused battery capacity whether they started the day with 60 or 120 kilometers of range (roughly 38 to 75 miles). Therefore, providing a more accurate prediction of remaining range or power consumption could be an economic and realistic solution to improve the market acceptance.

Most existing technologies that estimate how much longer a battery will last still provide inaccurate measurements such as those proposed in [9, 10, 11, 12, 13] which concentrate on battery

State of Charge (SoC) estimation or Battery Residual Capacity (BRC). Machine Learning solutions come into literature in recent years partially because of the dropping price of embedded systems. Huge amount of sensory data could be collected from the In-Vehicle Networks and transmitted to the Cloud where Machine Learning Algorithms could be applied and provide services including range prediction, influences of the power consumption and recommendations on energy harvesting. However, the majority of the research concerning Machine Learning emphasis on or cut down the problem down to only a limited aspect of the related variable. Methods proposed in [14] and [15] focus on identifying different driving patterns. Approaches introduced in [16] consider more GPS-based and manufacturers-provided data with a simplified EV power train model. Speed data has been analysed as the main variable in [17].

Multiple variables must be comprehensively considered to provide a more accurate prediction of power consumption [18]. For example, driving style matters because quick acceleration and fast driving discharge the battery faster. Aggressive braking might also cheat the EV's regenerative braking system of the chance to recapture some energy and recharge the battery. It has also been mentioned in the literature that factors such as temperature, weight, inclines and weather are also important. For instance, nine factors are considered in an estimation method introduced in [19] including the above mentioned features but it fails to consider the fact that the sensitivity and reliability of the range estimation algorithm changes under different environmental and operating conditions. An online prediction system based on regression analysis methods focusing on time series data including distance, velocity and elevation is introduced in [20]. However, the assumptions of knowing the applicable future information such as energy consumption of a trip beforehand and the future energy consumption is sufficiently similar to the past do not often correspond to reality. Rahimi-Eichi proposed a Big Data framework in [21] considering five types of public data for EV range estimating including route information, weather data, driving behavior data, electrical vehicle modeling data, and battery modeling data. However, the calculation of power consumption is relied on the basic physical mechanisms accordingly in a distributed manner, which is hard to adapt to the complexity of the stochastic nature of how these factors affect the power consumption. Lee and Wu, most recently, proposed a big-data analysis method in [22] concerning Machine Learning Algorithms to process the related data. However, the prediction problem was trimmed to the level that mature clustering and classification methods can be applied, which looks unreliable because normally a regression model is essentially used to predict the continuous types of data while classification is most probably used for categorical ones. Based on the hypothesis that the driving operation and the contextual data can be implied by the speed time series, only speed-energy consumption ratio data is fed to the clustering and all other related variables are considered in the classification.

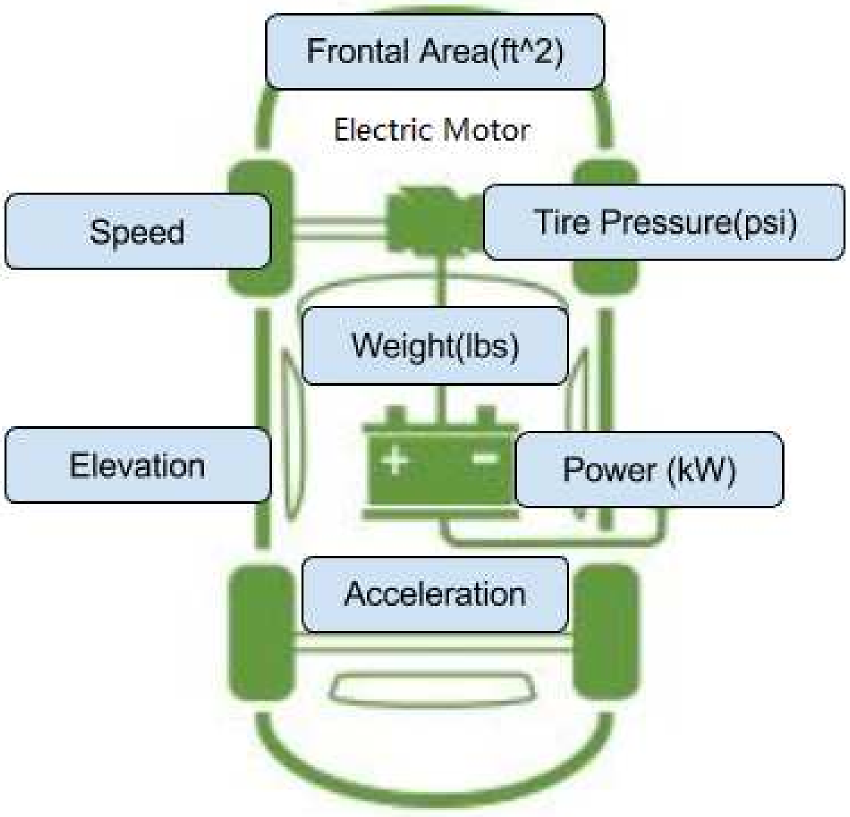


Figure 1.1: Variables to be Considered for EV range Estimation

1.4 Organization of the Thesis

The rest of the thesis is organized as follows.

In Chapter 2, regression problem is formally stated and related literature review is presented;

In Chapter 3, our proposed hybrid machine learning model will be discussed in details. Related mathematical ingredients will be explained before discussing the formal integrated algorithm and the presentation of the automated model selection algorithm.

In Chapter 4, we test and evaluate our proposed hybrid model in this chapter. A minor experiment without any necessary data preparation is presented first with detailed Tree-layer map and the corresponding variable importance ranking plottings of each of the trees. Further, we come back to the EV Range Anxiety case study and apply our proposed algorithm. Details of our testing data and related heuristics of data pre-processing are introduced and plotted followed by the utilization procedure and testing results.

In Chapter 5, we conclude the thesis with a highlight on the important contributions and their potential applications. Also, possible future expansion is presented in this chapter.

Chapter 2

Literature Review

2.1 Regression Problems

Regression analysis, also called Function Approximation (FA), can be loosely defined as the application of methods that investigate the relationship between a dependent (or response) variable and a set of independent (or predictor) variables. This study is usually based on a sample of measurements made on a set of objects. For example, one wishes to find a linear function that best predicts a baby's birth weight on the basis of ultrasound measures of his head circumference, abdominal circumference, and femur length. Here, our domain set $\mathbf{X}$ is some subset of $\mathbb{R}^3$ (the three ultrasound measurements), and the set of "labels", $\mathbf{Y}$, is the the set of real numbers (the weight in grams). In this context, it is more adequate to call $\mathbf{Y}$ the *target set*. Our training data as well as the learner's output are as before (a finite sequence of (x, y) pairs, and a function from $\mathbf{X}$ to $\mathbf{Y}$ respectively).

Formally, the task of a regression method is to obtain a model based on a sample of objects belonging to an unknown regression function. This sample (the training set) consists of pairs of the form (x_i, y_i) where x_i is a vector of the values of the attributes (predictor variables) and y_i is the respective value of the response (output). Let $\mathbf{X}$ be the input matrix whose $i-th$ row is the input vector x_i. If there are n vectors, $\mathbf{X}$ is a matrix with dimension $n \times a$, where a is the number of attributes. We will collect the target values of the input vectors in a $n \times 1$ matrix, $\mathbf{Y}$. We can also represent a data set $\mathcal{D}$ as a $n \times (a+1)$ matrix $\mathcal{D}$. We can look at a *regression learning system* as a function that maps a data set $\mathcal{D}$ into a regression model. A regression model is also a function that maps an input vector $x_i \in \chi$ into a real number $y_i \in \gamma$.

The measure of successful models are different. We may evaluate the quality of a hypothesis function, $h : \mathbf{X} \rightarrow \mathbf{Y}$, by the *expected square difference* between the true labels and their predicted values, namely,

$$L_{\mathcal{D}}(h) \overset{def}{=} \mathbb{E}_{(x,y)\sim\mathcal{D}}(h(x)-y)^2. \tag{2.1}$$

To accommodate a wide range of learning tasks we generalize our formalism of the measure of success with *Generalized Loss Functions* [23].

Given any set $\mathcal{H}$ (that plays the role of our hypotheses, or models) and some domain Z, let l be any function from $\mathcal{H} \times Z$ to the set of nonnegative real numbers, $l : \mathcal{H} \times Z \rightarrow \mathbb{R}_{+}$. We call such functions *loss functions*.

Note that for prediction problems, we have that $Z = X \times Y$. However, our notion of the loss function is generalized beyond prediction tasks, and therefore it allows Z to be any domain of examples.

The *loss functions* used in the preceding examples of regression tasks are *Square Loss*:

$$l_{sq}(h,(x,y)) \overset{def}{=} (h(x)-y)^2. \tag{2.2}$$

2.2 Kernel Smoothing

Kernel smoothing is a class of regression techniques that achieve flexibility in estimating the regression function $f(X)$ over the domain $\mathbb{R}^p$ by fitting a different but simple model separately at each query point x_0. This is done by using only those observations close to the target point x_0 to fit the simple model, and in such a way that the resulting estimated function $\widehat{f}(X)$ is $smooth$ in $\mathbb{R}^p$.

This localization is achieved via a weighting function or *kernel* $K_\lambda(x_0, x_i)$. For example, the Gaussian kernel has a weight function based on the Gaussian density function:

$$K_\lambda(x_0, x_i) = \frac{1}{\lambda} \exp\left[-\frac{\|x - x_0\|^2}{2\lambda} \right] \tag{2.3}$$

and assigns weights to the points, x_i, that die exponentially with their squared Euclidean distance from x_0. The parameter λ corresponds to the variance of the Gaussian density, and controls the width of the neighborhood.

We focus our study on the *kernel* K_λ which are typically indexed by a parameter λ that dictates the width of the neighbourhood. These *memory-based* methods require in principle little or no training; all the work gets done at evaluation time. The only parameter that needs to be determined from the training data is λ. The simplest form of kernel estimate is the Nadaraya-Watson weighted average

$$\hat{f}(x_0) = \frac{\sum_{i=1}^{N} K_\lambda(x_0, x_i) y_i}{\sum_{i=1}^{N} K_\lambda(x_0, x_i)}. \tag{2.4}$$

In general we can define a local regression estimate of $f(x_0)$ as $f_{\hat{\theta}}(x_0)$,

$$RSS(f_\theta, x_0) = \sum_{i=1}^{N} K_\lambda(x_0, x_i)(y_i - f_\theta(x_i))^2, \tag{2.5}$$

and f_θ is some parameterized function, such as a low-order polynomial [24].

2.2.1 Nearest-Neighbour Methods

Nearest Neighbour algorithms are among the simplest of all machine learning algorithms. The idea is to memorize the training set and then to predict the label of any new instance on the basis of the labels of its closest neighbours in the training set. The rationale behind such a method is based on the assumption that the features that are used to describe the domain points are relevant to their labellings in a way that makes close-by points likely to have the same label. Furthermore, in some situations, even when the training set is immense, finding a nearest neighbour can be done extremely fast (for example, when the training set is the entire Web and distances are based on links) [23].

Assume a metric function ρ that returns the Euclidean distance between any two elements of the instance domain $X = \mathbb{R}^d$, formally $\rho(x, x') = \|x - x'\| = \sqrt{\sum_{d}^{i=1}(x_i - x'_i)^2}$. Let $S = (x_1, y_1), ..., (x_m, y_m)$ be a sequence of training examples. For each $x \in X$, let $\pi_1(x), ..., \pi_m(x)$ be a reordering of $1, ..., m$ according to their distance to x, $\rho(x, x_i)$. That is, for all $i < m$, $\rho(x, x_{\pi_i(x)}) < \rho(x, x_{\pi_{i+1}(x)})$. When $k = 1$, we have the 1-NN rule: $\hat{Y} = y_{\pi_1(x)}$ and the geometric illustration of the 1-NN rule is a $Voronoi\ Tessellation$ of the space, as given in Figure 2.1.

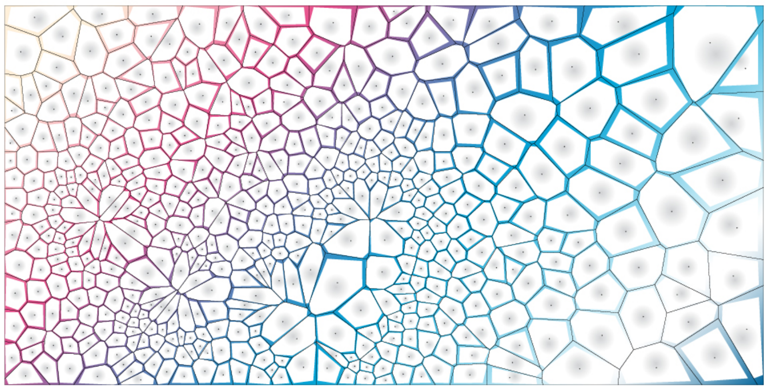


Figure 2.1: An illustration of the decision boundaries of the 1-NN rule. The points depicted are the sample points, and the predicted label of any new point will be the label of the sample point in the center of the cell it belongs to. These cells are called a Voronoi Tessellation of the space.

Concentrating our study on regression problems, namely $Y = \mathbb{R}$, one can define the prediction to be the average target of the k nearest neighbours. That is, $\hat{Y}(x) = \frac{1}{k}\sum_{i=1}^{k} y_{\pi_i(x)}$. More generally, for some function $\phi : (X, Y)^k \to Y$ the k-NN rule with respect to ϕ is:

$$\hat{Y}(x) = \phi((x_{\pi_1(x)}, y_{\pi_1(x)}), ..., (x_{\pi_k(x)}, y_{\pi_k(x)})). \tag{2.6}$$

It is easy to verify that we can cast the prediction by the averaged target (for regression) as in the above Equation by an appropriate choice of ϕ. And the geometric meaning of the Nearest Neighbour regression could be explained by combining the adjacent Voronoi tiles and average among the merged samples. The generality can lead to other kernel smoothing rules such as taking a weighted average of the targets according to the distance from x:

$$\hat{Y}(x) = \sum_{i=1}^{k} \frac{\rho(x, x_{\pi_i(x)})}{\sum_{j=1}^{k} \rho(x, x_{\pi_j(x)})} y_{\pi_i(x)}. \tag{2.7}$$

2.3 Artificial Neural Networks

2.3.1 Multilayer Perceptrons

The term *neural network (NN)* has evolved to encompass a large class of models and learning methods. We discuss here the most widely used single hidden layer back-propagation network, or single layer perceptron.

A neural network is a two-stage regression or classification model. Most neural networks have three layers: the input layer, a hidden layer, and the output layer, typically represented by a network diagram as in Figure 2.2. The hidden layer is so named because it is invisible, with no direct contact to inputs or outputs. This network applies both to regression or classification. And these networks can handle multiple quantitative responses in a seamless fashion. We shall deal with the general case.

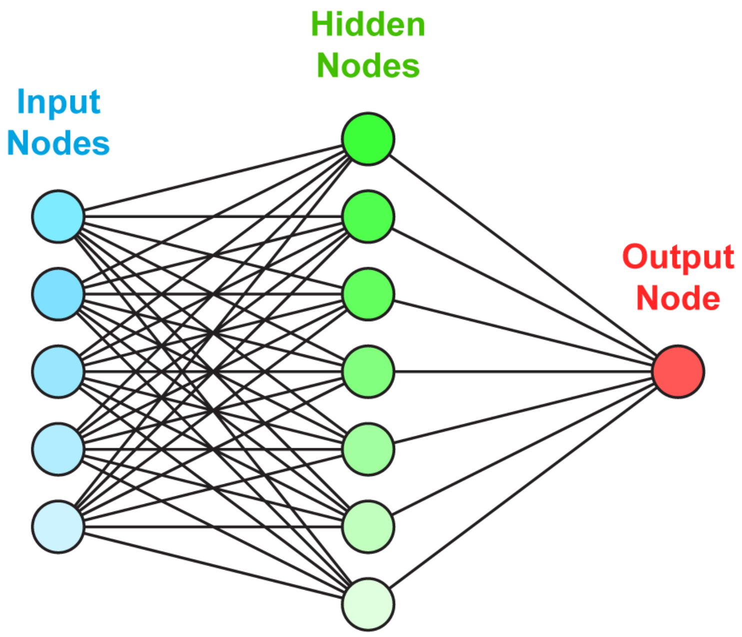


Figure 2.2: ANN Structure

Derived features Z_m are created from linear combinations of the inputs, and then the target Y_k is modelled as a function of linear combinations of the Z_m,

$$\begin{aligned} Z_m &= \sigma(\alpha_{0m} + \alpha_m^T X), m = 1, ..., M, \\ T_k &= \beta_{0k} + \beta_k^T Z, k = 1, ..., K, \\ f_k(X) &= g_k(T), k = 1, ..., K, \end{aligned} \tag{2.8}$$

where $Z = (Z_1, Z_2, ..., Z_M)$ and $T = (T_1, T_2, ..., T_k)$. And the activation function $\sigma(v)$ is usually chosen to be the $sigmoid$ $\sigma(v) = 1/(1 + \mathrm{e}^{-v})$.

The neural network model has unknown parameters, often called $weights$, and we seek values for them that make the model fit the training data well [24]. We denote the complete set of weights by θ, which consists of

$$\begin{aligned} \{\alpha_{0m}, \alpha_m; m = 1, 2, ..., M\} \quad M(p+1) \quad weights, \\ \{\beta_{0k}, \beta_k; k = 1, 2, ..., K\} \quad K(M+1) \quad weights. \end{aligned} \tag{2.9}$$

For regression, we use sum-of-squared errors as our measure of fit (error function)

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2. \tag{2.10}$$

The generic approach to minimizing $R(\theta)$ is by gradient descent, called *back-propagation* in this setting. Because of the compositional form of the model, the gradient can be easily derived using the chain rule for differentiation. This can be computed by a forward and backward sweep over the network, keeping track only of quantities local to each unit. Formally, the *back-propagation* procedure could be described in detail as the following.

Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$ and let $z_i = (z_{1i}, z_{2i}...z_{Mi})$. Then we have

$$\begin{aligned} R(\theta) &= \sum_{i=1}^{N} R_i \\ &= \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2, \end{aligned} \tag{2.11}$$

with derivatives

$$\begin{aligned} \frac{\partial R_i}{\partial \beta_{km}} &= -2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) z_{mi}, \\ \frac{\partial R_i}{\partial \alpha_{ml}} &= -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) \beta_{km} \sigma'(\alpha_m^T x_i) x_{il}. \end{aligned} \tag{2.12}$$

Given these derivatives, a gradient descent updated at the $(r+1)st$ iteration has the form

$$\begin{aligned} \beta_{km}^{(r+1)} &= \beta_{km}^{(r)} - \gamma_r \sum_{i=0}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}, \\ \alpha_{ml}^{(r+1)} &= \alpha_{ml}^{(r)} - \gamma_r \sum_{i=0}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}}, \end{aligned} \tag{2.13}$$

where γ_r is the learning rate, discussed below.

Now, (2.12) can be written as:

$$\begin{aligned} \frac{\partial R_i}{\partial \beta_{km}} &= \delta_{ki} z_{mi}, \\ \frac{\partial R_i}{\partial \alpha_{ml}} &= s_{mi} x_{il}. \end{aligned} \tag{2.14}$$

The quantities δ_{ki} and s_{mi} are "errors" from the current model at the output and hidden layer units, respectively. From their definitions, these errors satisfy

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \gamma_{ki}, \tag{2.15}$$

known as the *back-propagation equations*. Using this, the updates in (2.13) can be implemented with a two-pass algorithm. In the $forward\ pass$, the current weights are fixed and the predicted values $\widehat{f}_k x_i$ are computed from (2.15). In the $backward\ pass$, the errors δ_{ki} are computed, and then back-propagated via (2.15) to give the errors s_{mi}. Both sets of errors are then used to compute the gradients for the updates in (2.13), via (2.14).

The advantages of back-propagation are its simple and local nature. In the back propagation algorithm, each hidden unit passes and receives information only to and from units that share a connection. Hence it can be implemented efficiently on a parallel architecture computer. However, back-propagation can be very slow, and for that reason it is usually not the method of choice.

2.3.2 Radial Basis Function Networks

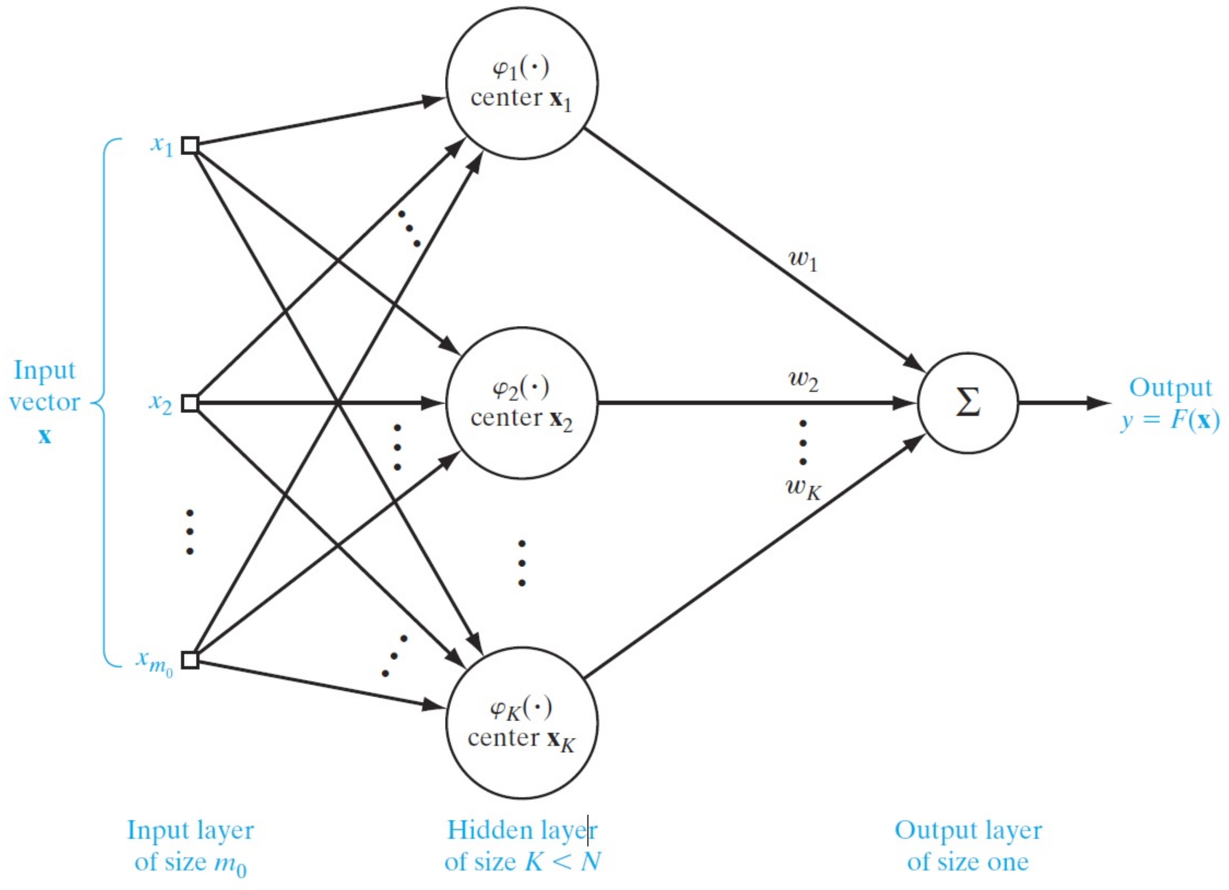
Radial basis function (RBF) networks typically have three layers: an input layer, a hidden layer with a non-linear RBF activation function and a linear output layer [25].

- $Input\ layer$, which consists of m_o source nodes, where m_o is the dimensionality of the input vector $\mathbf{x}$.
- $Hidden\ layer$, which consists of the same number of computation units as the size of the training sample, namely, N; each unit is mathematically described by a radial-basis function

$$\varphi_j(\mathbf{x}) = \varphi(\|\mathbf{x} - \mathbf{x}_j\|), \quad j = 1, 2, ..., N. \tag{2.16}$$

The jth input data point $\mathbf{x}_j$ defines the center of the radial-basis function, and the vector $\mathbf{x}$ is the signal (pattern) applied to the input layer. Thus, unlike a multilayer perceptron, the links connecting the source nodes to the hidden units are direct connections with no weights.

- *Output layer*, which, in the RBF structure of Fig. 2.3, consists of a single computational unit. Clearly, there is no restriction on the size of the output layer, except to say that typically the size of the output layer is much smaller than that of the hidden layer.



Input vector $\mathbf{x}$
x_1
x_2
x_{m_0}
$\varphi_1(\cdot)$ center $\mathbf{x}_1$
$\varphi_2(\cdot)$ center $\mathbf{x}_2$
$\varphi_K(\cdot)$ center $\mathbf{x}_K$
w_1
w_2
w_K
Σ
Output $y = F(\mathbf{x})$
Input layer of size m_0
Hidden layer of size $K < N$
Output layer of size one

Figure 2.3: Structure of a practical RBF network.

Gaussian function is usually used as the radial-basis function, in which case each computational unit in the hidden layer of the network of Fig. 2.3 is defined by

$$\begin{aligned}\varphi_j(\mathbf{x}) &= \varphi(\mathbf{x}-\mathbf{x}_j) \\ &= \exp(-\frac{1}{2\sigma_j^2}\left\|\mathbf{x}-\mathbf{x}_j\right\|^2), \quad j=1,2,...,N\end{aligned} \tag{2.17}$$

where σ_j is a measure of the width of the jth Gaussian function with center x_j. Typically, but not always, all the Gaussian hidden units are assigned a common width σ. In situations of this kind, the parameter that distinguishes one hidden unit from another is the center x_j. The rationale behind the choice of the Gaussian function as the radial-basis function in building RBF networks is that it has many desirable properties, which will become evident as the discussion progresses.

On the choice of the size of the hidden layer, the rudimentary one could be the size of the training set. However, having a hidden layer of the same size as the input layer could be wasteful of computational resources, particularly when dealing with large training samples. When the hidden layer of the RBF network is specified in the manner described in (2.17), we find that any correlation existing between adjacent data points in the training sample is correspondingly transplanted into adjacent units in the hidden layer. Stated in another way, there is redundancy of neurons in the hidden layer when they are chosen in accordance with (2.17) by virtue of the redundancy that may inherently exist in the training sample. In situations of this kind, it is therefore good design practice to make the size of the hidden layer a fraction of the size of the training sample.

In designing the RBF network of Figure 2.3, a key issue that needs to be addressed is how to compute the parameters of the Gaussian units that constitute the hidden layer by using $unlabeled$ data. In other words, the computation is to be performed in an unsupervised manner. The solution to this problem is usually rooted in clustering and the so-called *K-means algorithm* is often applied because it is simple to implement, yet effective in performance.

Moreover, the approximating function realized by both of these two RBF structures has the same mathematical form,

$$F(\mathbf{x}) = \sum_{j=1}^{K} \omega_j \varphi(\mathbf{x}, \mathbf{x}_j). \tag{2.18}$$

where the dimensionality of the input vector $\mathbf{x}$ (and therefore that of the input layer) is m_0 and each hidden unit is characterized by the radial-basis function $\varphi(\mathbf{x}, \mathbf{x}_j)$, where $j = 1, 2, ..., K$, with K being smaller than N. The output layer, assumed to consist of a single unit, is characterized by the weight vector $\mathbf{w}$, whose dimensionality is also K.

2.4 Principal Component Analysis

In many situations we have a large number of inputs, often very correlated. Principal Components Analysis (PCA) is a way of identifying patterns in data, and expressing the data in such a way as to highlight their similarities and differences. Since patterns in data can be hard to find in data of high dimension, where the luxury of graphical representation is not available, PCA is a powerful tool for analysing data by transforming a set of possibly correlated variables into a set of uncorrelated linear re-combinations of those variables called principal components. The other main advantage of PCA is that once you have found these patterns in the data, and you compress the data, *ie.* by reducing the number of dimensions, without much loss of information. Focusing on regression, formally, PCA produces a small number of linear combinations $Z_m, m = 1, ..., M$ of the original inputs X_j, and the Z_m are then used in place of the X_j as inputs in the regression. The methods differ in how the linear combinations are constructed.

Principal component regression (PCR) forms the derived input columns $\mathbf{z}_m = \mathbf{X}_{v_m}$, and then regresses $\mathbf{y}$ on $\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_M$ for some $M \leq p$ [24]. Since the $\mathbf{z}_m$ are orthogonal, this regression is just a sum of univariate regressions:

$$\widehat{\mathbf{y}}^{pcr}_{(M)} = \overline{y}\mathbf{1} + \sum_{m=1}^{M} \widehat{\theta}_m \mathbf{z}_m,$$

where $\widehat{\theta}_m = \langle \mathbf{z}_m, \mathbf{y} \rangle / \langle \mathbf{z}_m, \mathbf{z}_m \rangle$. Since the $\mathbf{z}_m$ are each linear combinations of the original $\mathbf{x}_j$, we can express the solution of the above equation in terms of coefficients of the $\mathbf{x}_j$:

$$\widehat{\beta}^{pcr}(M) = \sum_{m=1}^{M} \widehat{\theta}_m v_m.$$

Principal components depend on the scaling of the inputs, so typically they are first standardized.

2.5 Difficulties

Returning to our EV range anxiety case study, the difficulty states itself from the related work of data-driven estimation solutions discussed in Chapter 1. Multivariate high dimensional data in different types must be analyzed to, ultimately, predict a numerical result.

We have seen that although nearest-neighbor and other local methods focus directly on estimating the function at a point, they face problems in high dimensions – *curse of dimensionality*. They may also be inappropriate even in low dimensions in cases where more structured approaches can make more efficient use of the data. The nearest-neighbor methods discussed so far are based on the assumption that locally the function is constant; close to a target input x_0, the function does not change much, and so close outputs can be averaged to produce $\hat{f}(x_0)$. Other methods such as splines, neural networks and basis-function methods implicitly define neighborhoods of local behavior.

PCA is a classic technique to derive underlying variables, reducing the number of dimensions we need to consider in a dataset. However, PCA makes several assumptions, such as relying on data spread and orthogonality to derive components. Also, the covariance matrix is difficult to be evaluated in an accurate manner [26]. Further, even the simplest invariance could not be captured by the PCA unless the training data explicitly provides this information [27]. The recently proposed method of the Self-Organizing Maps (SOM) is likely to become a complementary or alternative tool of the PCA. Reush et al. compared the pattern extraction capability of SOM and PCA using synthetic example, and showed SOM is capable of extracting pattern in data without superposition of input data.

Neural networks (NN) are relatively successful in applications dealing with subsymbolic raw data; in particular, if the data is noisy or inconsistent. They exhibit the property to produce their structure during learning by the integration (overlay) of many case data. Unfortunately, they have the disadvantage that they cannot be interpreted by looking at the activity or weights of single neurons [28]. It makes them hard to determine how they are solving a problem, because they are opaque (*black box*). Further, there are also several issues in training the Neural Networks because the model is generally over-parametrized, and the optimization problem is nonconvex and unstable unless certain guidelines are followed [29]. For example, it is hard to decide the number of hidden units and layers. With too few hidden units, the model might not have enough flexibility to capture the nonlinearities in the data; with too many hidden units, the extra weights can be shrunk toward zero if appropriate regularization is used. Choice of the number of hidden layers is guided by background knowledge and experimentation. Each layer extracts features of the input for regression or classification. Use of multiple hidden layers allows construction of hierarchical features at different levels of resolution.

Chapter 3

Methodology

In this chapter, we prepare the fundamental theories for our proposed the Hybrid Machine Learning Model. The proposed Hybrid Model can be viewed as a modified version of Self-Organizing Maps (SOM) with one layer deeper combining the Classification and Regression Tree (CART) after the standard training process. The difference with training the conventional SOM is that the data points mapped to the neurons are kept for building the regression trees. Each neuron shall serve as a kernel smoothing neighborhood introduced in Chapter 2 and have a regression tree built as the alternative codebook based on the mapped historical training data.

Local regression is usually considered less useful in dimensions much higher than two or three. It is impossible to simultaneously maintain localness (i.e. low bias) and a sizable sample in the neighborhood (i.e. low variance) as the dimension increases, without the total sample size increasing exponentially in the number of dimensions, p. Visualization of $\hat{f}(\mathbf{X})$ also becomes difficult in higher dimensions, and this is often one of the primary goals of smoothing. The Self-Organizing Map (SOM), also known as the Kohonen network, is a computational method for the visualization, low-dimensional approximation and analysis of high-dimensional data which also preserves the topological structure of that data set.

However, SOM also has some drawbacks. Hybridizing localized regression based on the data partitioning could not only provide more accurate prediction results but also reveal the inherent of the data. Computational complexity is also improved without globally fitting each tester to the training dataset.

3.1 Self-Organizing Maps

Self-organizing map (SOM) is a powerful paradigm that is used for cluster analysis and regression learning thanks to its ability to provide a topological projection of high dimensional non-linear data. Interestingly in SOM, neurons representing closely related information are kept close together during training, mimicking human brains, so that they can interact via short synaptic connections.

In a SOM, the neurons are placed at the nodes of a lattice that is usually one or two dimensional and in the final configuration of the map. Depending on the competitive learning process, the $localized$ feature-sensitive neurons respond to the input patterns in an orderly fashion, as if a curvilinear coordinate system. This reflects some topological order of events in the input space and a SOM is therefore characterized by the formation of a topographic map of the input data vectors, in which the spatial locations (i.e. coordinates) of the neurons in the lattice correspond to intrinsic features of the input patterns.

The SOM defines a mapping from the input data space of k dimensions $\mathbf{x} = [x_1, x_2, ..., x_k]^T$ onto a regular two-dimensional array of nodes. The synaptic reference vectors $\mathbf{m}_j = [m_{j1}, m_{j2}, ..., m_{jk}]^T,\ j = 1, 2, ..., L$ of each neuron j has the same dimension as the input space; L is the total number of neurons.

Before recursive processing, the codebooks, m_i, must be initialized. Random values for the components of m_i may do, but if the initial values of the m_i are selected with care, the convergence of the codebooks can be achieved much faster.

Selecting the neuron with the largest inner product $\mathbf{m}_j^T\mathbf{x}$, is mathematically equivalent to minimizing the Euclidean distance between the input vectors $\mathbf{x}$ and $\mathbf{m}_j$. Thus, the winning neuron n is defined as: $c = \arg\min_{1\leq j\leq l}\{||\mathbf{x}-\mathbf{m}_j||\}$. Essentially this sums up the competition process among the neurons, where the best-matching node locates the center of a topological neighborhood [30].

During the learning, those nodes that are topographically close to a certain distance will activate each other to learn from the same input. Using discrete-time formalism, reference vector at time t is written as $\mathbf{m}_j(t)$, and updated reference vector is defined as:

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + \mathbf{h}_{ci}(t)[\mathbf{x} - \mathbf{m}_i(t)], \tag{3.1}$$

where $\mathbf{h}_{jc}(t)$ is neighbourhood kernel and t is a discrete-time coordinate, m_j is a reference vector, x is a data sample, and h_{jc} is a neighbourhood function, usually in the form of the Gaussian function,

$$\mathbf{h}_{jc}(t) = \alpha(t)\cdot\exp(-\frac{\|\mathbf{r_n} - \mathbf{r_j}\|^2}{2\sigma^2(t)}), \tag{3.2}$$

where $\alpha(t)$ is the training rate function, $\mathbf{r}$ is the location vector in the matrix, $\|\mathbf{r_n} - \mathbf{r_j}\|$ corresponds to the distance between the best-matching node (location $\mathbf{r_n}$) and each of the other nodes (location $\mathbf{r_j}$) in the two-dimensional matrix, and $\sigma(t)$ defines the brand-width of the kernel, often referred to as the radius of training. Both $\alpha(t)$ and $\sigma(t)$ are monotonically decreasing functions over the time, which guarantees convergence and stability of the map.

The learning process involved in the computation of a feature map is stochastic in nature, which means that the accuracy of the map depends on the number of iterations of the SOM algorithm. First, the size of SOM determines the degree of generalization that will be produced by the SOM algorithm - the more nodes, the finer the representation of the details, while the fewer nodes, the broader level of generalization. However, the same broad patterns are revealed at each level of generalization. Another issue regarding the size of the SOM, is that more nodes means longer training time. Second, the $learning\ steps$ must be reasonably large, since the learning is a stochastic process. A "rule of thumb" is that for good statistical accuracy, the number of steps must be at least 500 times the number of network nodes. Furthermore, the $learning\ rate$ parameter α should start with a value close to 1 and then during the first 1000 steps it should be decreasing monotonically but kept above 0.1. The exact form of variation of $\alpha = \alpha(t)$ is not critical, it can be linear, exponential or inversely proportional to t. For instance, $\alpha(t) = 0.9(1 - t/1000)$ may be a reasonable choice. It is during the initial phase of the algorithm that the topological ordering of the m_i occurs. This phase of the learning process is therefore called the ordering phase . The remaining (relatively long) iterations of the algorithm are needed for the fine adjustment of the map; this second phase of the learning process is called the convergence phase. For good statistical accuracy, $\alpha(t)$ should be maintained during the convergence phase at a small value (on the order of 0.01 or less) for a fairly long period of time, which is typically thousands of iterations. Neither is it crucial whether the law for $\alpha(t)$ decreases linearly or exponentially during the convergence phase.

The Neighbourhood function can be chosen to be the simple neighbourhood-set definition of $h_{ci}(t)$ if the lattice is not very large, e.g. a few hundred nodes. For larger lattices, the Gaussian function may do. The size of the Neighbourhood must be chosen wide enough at the start so the map can be ordered globally. If the neighbourhood is too small to start with, various kinds of mosaic-like parcellations of the map are seen, between which the ordering direction changes discontinuously. This phenomena can be avoided by starting with a fairly wide neighbourhood set $N_c = N_c(0)$ and letting it shrink with time. The initial radius can even be more than half the diameter of the network. During the first 1000 steps or so, when the ordering phase takes place, and $\alpha = \alpha(t)$ is fairly large, the radius of N_c can shrink linearly to about one unit, during the convergence phase N_c can still contain the nearest neighbours of node c.

3.2 Classification and Regression Trees (CART)

Tree-based methods recursively partition the space of explanatory variables into a set of rectangles, and then fit a simple model (like a constant) in each one. They are conceptually simple yet powerful. The most famous recursive partitioning method is CART, an acronym for classification and regression trees, see Breiman et al. (1984).

Adopting prediction trees to represent the recursive partition has several attractive advantages.

- Most intuitively, tree structure helps making regression results interpretable.
- CART is non-parametric. Therefore this method does not require specification of any functional form and assumptions.
- CART does not require variables to be selected in advance. The algorithm itself will identify the most significant variables and eliminate non-significant ones.
- CART can easily handle outliers. Outliers can negatively affect the results of some statistical models, like Principal Component Analysis (PCA) and linear regression. But the splitting algorithm of CART will easily handle noisy data because the influence of some variables can be localized to some regions of the domain space and not matter on others.

As the name implies, the tree structure of the recursive partitioning method applies to both regression and classification. However, the techniques used with trees for regression and for classification are quite different owing to the nature of the problem. We focus only on regression trees in this thesis.

Consider the dataset consists of p input variables and a response variable, for each of N observations: that is, (x_i, y_i) for $i = 1, 2, ..., N$, with $x_i = (x_{i1}, x_{i2}, ..., x_{ip})$. The algorithm needs to automatically decide on the splitting variables and split points, and also what topology (shape) the tree should have. Suppose first that we have a partition into M regions $R_1, R_2, ..., R_M$, and we model the response as a constant c_m in each region:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m) \tag{3.3}$$

If we adopt as our criterion minimization of the sum of squares $\sum(y_i - f(x_i))^2$, it is easy to see that the best $\widehat{c}_m$ is just the average of y_i in region R_m:

$$\begin{aligned}
\underset{c_m}{\operatorname{argmin}} \sum_{\widehat{x},\widehat{y} \in D} (f(\widehat{x}) - \widehat{y}) &= \underset{c_m}{\operatorname{argmin}} \sum_{(\widehat{x},\widehat{y})} (\sum_i c_i I[x \in R_i] - \widehat{y})^2 \\
&= \underset{c_m}{\operatorname{argmin}} \sum_{(\widehat{x},\widehat{y}):\widehat{x} \in R_m} (c_m - \widehat{y})^2 \\
&= \underset{(\widehat{x},\widehat{y}):\widehat{x} \in R_m}{\operatorname{mean}} \widehat{y}
\end{aligned} \tag{3.4}$$

Now finding the best binary partition in terms of minimum sum of squares is generally computationally infeasible. Hence we proceed with a greedy algorithm. Starting with all of the data, consider a splitting variable j and split point s, and define the pair of half-planes:

$$R_1(j, s) = \{X | X_j \leqslant s\} \quad \text{and} \quad R_2(j, s) = \{X | X_j > s\}\,. \tag{3.5}$$

Then we seek the splitting variable j and split point s that solve:

$$\min_{j,s} \left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right]. \tag{3.6}$$

For any choice j and s, the inner minimization is solved by:

$$\widehat{c}_1 = mean(y_i | x_i \in R_1(j, s)) \quad \text{and} \quad \widehat{c}_2 = mean(y_i | x_i \in R_2(j, s)) \tag{3.7}$$

For each splitting variable, the determination of the split point s can be done very quickly and hence by scanning through all of the inputs, determination of the best pair (j, s) is feasible. Having found the best split, we partition the data into the two resulting regions and repeat the splitting process on each of the two regions. Then this process is repeated on all of the resulting regions.

Chapter 4

Growing Self-Organizing Kernel Smoother

In this section, we propose our hybrid regression learning model. The hybrid model contains two layers in which the first layer is used for kernelization and input mapping, while the second layer serves as the regression output function (approximator). We also provide the mechanism of the self-growing process of the map. The criteria of building the trees will also partially affect the automated map growth.

For one thing, on the purpose of adopting SOM in our proposed model at the first stage is to cluster the similar cases and eventually the regression could be operated locally. For another, in order to effectively apply Self-Organizing Maps in regression, the output functions play an important role [31]. Therefore, we integrate CART, specifically Regression Tree model, as the output layer of the SOM to achieve a reasonable predictability and mine knowledge from the data. We consider the above architecture depicted in Figure 4.1. As can be seen from the diagram, it is very similar to Radius-Based Function Networks structure illustrated in Figure 2.3 in Section 2.3.2. Indeed, the similarity is that the first layer SOM neurons serve as kernels by clustering the similar historical data points. However, the difference from RBF is that the output layer of RBF is trained as a linear model while ours not by adopting regression trees.

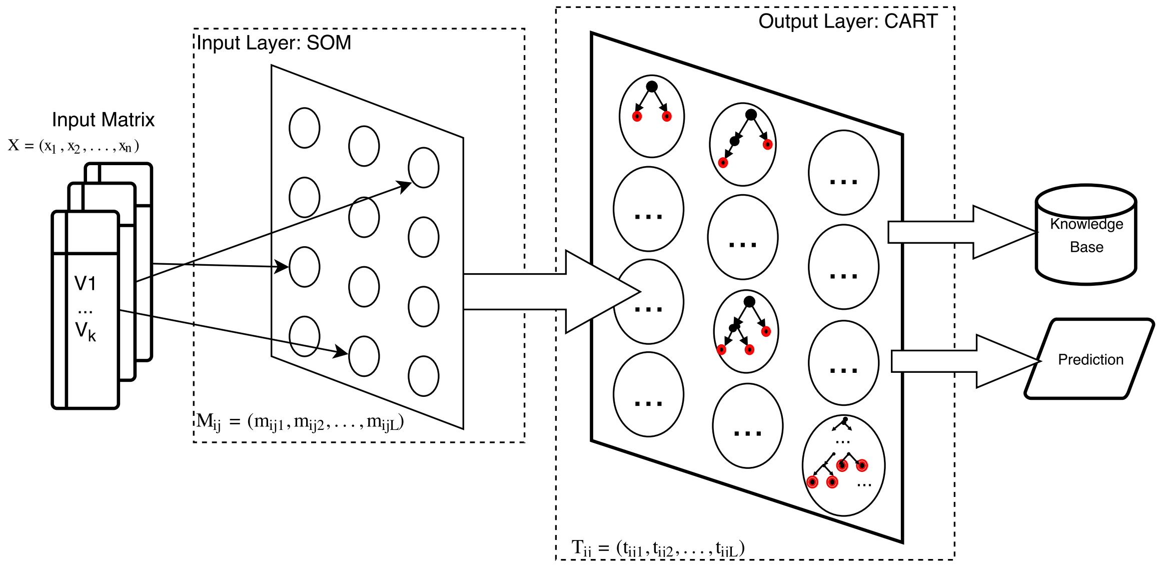


Figure 4.1: System Architecture

4.1 Self-Organizing Tree-Based Kernel Smoother

4.1.1 Self-Organizing Kernels

The SOM can be said to be a ”nonlinear projection” of the probability density function $p(x)$ of the high-dimensional input data vector $\mathbf{x}$ onto the two-dimensional array. The central result in self-organization is that if the input patterns have a well-defined probability density function $p(x)$, then the weight vectors associated with the models m_i should try to imitate it [30]. This will result in a local relaxation or smoothing effect on the weight vectors of neurons in this neighbourhood, which in continued learning leads to global ordering.

However, the self-organizing map algorithm suffers from several limitations. From a theoretical and applicational perspective, there are four shortcomings [25, 32, 30]:

1. The estimate of the probability density function of the input space provided by the algorithm

lacks accuracy.

2. The formulation of the algorithm has no objective function that could be optimized, especially in the high-dimensional case.

3. Difficult to represent very many variables in two dimensional plane

4. SOM uses a fixed network architecture in terms of number and arrangement of neural processing elements, which has to be defined prior to training.

5. Hierarchical relations between the input data are not mirrored in a straightforward fashion.

To cope the first two obstacles, modified versions of SOM have been applied with kernelization in various ways [33, 34, 35]. F. Mulier and V. Cherkassky have already proved in [36] that SOM can be considered in a statistical way in which SOM algorithm can be directly interpret as a kernel smoothing problem.

Following the statistical SOM Kernel Smoothing in [36], in this section, we provide the mathematical explanations of using the SOM as the Kernel Smoother. Note that the terminology we use in our model is ”Self-Organizing Kernels” rather than ”kernel SOM”. In spite of the connections between them, the technique of ”kernel smoothing” should not be confused with those associated with the more recent usage of the phrase ”kernel tricks”. The kernels for smoothing are mostly used as a device for $localization$. In those contexts of kernel tricks, however, the kernel computes an inner product in a high-dimensional (implicit) feature space, and is used for regularized nonlinear modeling. Fortunately, there introduced connections between this two methodologies in Chapter 6.7 of [24] and Chapter 5.10 of [25], since after all the essential operation of these ideas is utilizing the regional neighbourhood data samples.

Assuming that the convergence to some stable state of the SOM is true [37], we require that the expectation values of $\mathbf{m}_i(t+1)$ and $\mathbf{m}_i(t)$ for $t \to \infty$ must be equal, while h_{ci} is nonzero, where $c = c(\mathbf{x}(t))$ is the index of the winner node for input $\mathbf{x}(t)$. In other words we must have

$\forall i, E_t \left\{ h_{ci}(\mathbf{x}(t) - \mathbf{m}_i(t)) \right\} = 0$ where E_t is the mathematical expectation value operator over t. In the assumed asymptotic state, for $t \to \infty$, the $\mathbf{m}_i(t)$ are independent of t and are denoted by m_i^*. If the expectation values $E_t(.)$ are written, for $t \to \infty$, as $(1/t) \sum_t (.)$, we can write

$$m_i^* = \frac{\sum_t h_{ci} \mathbf{x}(t)}{\sum_t h_{ci}} \tag{4.1}$$

Then it is useful to notice that for the different nodes i, the same addends occur a great number of times. Therefore it is advisable, especially with very large SOMs, to first compute the mean $x_{m,j}$ of all of the $x(t)$ that are closest to model m_j in the data space, and then weight it by the number n_j of the samples in this subset and by h_{ji}. Then we obtain

$$m_i^* = \frac{\sum_j n_j h_{ji} \mathbf{x}_{m,j}}{\sum_j n_j h_{ji}} \tag{4.2}$$

where the notation $x_{m,j}$ is used to denote the mean of the inputs that are closest to the model m_j, and n_j is the number of those inputs.

With the shrinkage of the neighbourhood function h_{ci} and convergence of the map in the training process, each neuron of the map represents a $Voronoi\ Partitioning$ of the input data and can be used as a smoothing kernel. Mimicking the work in [36], we utilize the SOM as a kernel estimate as in [36], batch SOM is adopted for getting rid of choosing the learning rate. Using the interpretation given by (4.2), it is possible to view the SOM algorithm as a statistical nonparametric regression problem. And the form of (4.2) is very similar to the Nadaraya-Watson kernel estimator introduced in Chapter 2:

$$\hat{y}(x) = \frac{\sum_{k=1}^{K} H(x - x_k) y_k}{\sum_{k=1}^{K} H(x - x_k)}$$

where $x, y \in \mathbb{R}^1$. Due to the similarities, (4.2) can be interpreted as a kernel estimate. The SOM plays the role of kernel function and achieves spherical clusters [38] $\mathcal{L}^{(c)}$ (the local set of mapped data in neuron c).

Due to the high dimensionality, if local weight average is applied just as the traditional local regression methods, the spatial resolution of each kernel $\mathcal{L}^{(c)}$ is rather coarse to achieve a satisfactory result, which is why the kernels are later to be further microscopically analysed in the second layer by locally generated regression trees $\mathcal{T}^{(c)}$. This could meantime provide more interpretative codebooks and reveal hierarchical relations of the data, solving the third and fifth shortcomings mentioned above. The first layer of SOM will be used for locating the new case to its nearest neighbours. And because of the second layer, $\mathcal{T}^{(c)} - layer$, will ultimately be applied for regression output functions, the criterion of selecting the radius of c should be based on the goodness of fitting of the $\mathcal{T}^{(c)} - layer$. This will be discussed in detail in later verses.

4.1.2 Local Tree Settings

In conventional kernel/local regression methods, there include usually four components – bandwidth, local polynomial degree, weight function and fitting criterion. The local polynomial degree is selected to affect the bias-variance trade-off with the bandwidth and a weight function is selected to provide a smoother estimate. Since our proposed kernel smoothing method is non-parametric, selecting the local polynomial degree and weight function is substituted by building Classification and Regression Tree (CART) locally instead. In regression, CART acts as a smart bin-smoother that performs automatic variable selection. Bin smoothers might be the simplest non-parametric estimators of a regression function.

CART is suitable for multivariate scenario because they are non-parametric, have naturally define subgroups, scale with the complexity of the data, and are not limited by the number of predictor variables. Also, single decision trees are highly interpretable. The entire model can be completely represented by a simple two-dimensional graphic (binary tree) that is easily visualized. However, CART suffers from its inaccuracy and recently, bagging and boosting trees are proposed as remedy in which CART is usually considered as the base (weak) learner. On the other hand, linear combinations of trees lose this important feature, and must therefore be interpreted in a different way. Further, if applying the CART globally, a huge bushy tree shall be generated, which is not efficient for regression prediction and drops the goal of knowledge extraction. Another well known disadvantage of trees is that they can be unstable. Small changes in the data can result in completely different tree. One reason for this is that any change at an upper level of the tree is propagated down the tree and affects all other splits. This is because if a particular split changes then all the other splits that are under it change as well.

These dilemmas motivate us adopting CART locally on the rudimentary clusters trained by SOM. First, it could gain a more sophisticated partitioning among local data cases and also achieve a concrete and concise knowledge extraction, preserving its advantages. In addition, since there is only a small portion of the dataset mapped to a single kernel (neuron), the cost of re-constructing a new tree is fairly acceptable. Last but not least, there are very few parameters to be decided when modelling.

Formally, for a fitting point x (trained as the centroid of SOM), define a bandwidth $\lambda(x)$ and a smoothing window $(x-\lambda(x), x+\lambda(x))$. To estimate $\hat{f}(x)$, only observations (local observations) within this window are used to form a regression tree, formally:

$$\mathcal{T}_i^{(c)} = \mathcal{T}(\frac{x_i - x}{\lambda(x)}, \theta)$$

where $\mathcal{T}(\cdot, \theta)$ is the regression tree as the smoothing estimator taking global standard θ which is the parameter set of building the tree. This is also important when it comes to the growing process to be discussed in the section afterwards.

The only two parameters (eventually one) that we take into consideration in θ to build the local regression trees are the $minsplit$ and the $minbucket$. $minsplit$ represents the minimum number of observations that must exist in a node in order for a split to be attempted and this parameter can save computation time, since smaller nodes are almost always pruned away by cross validation. $minbucket$ represents the minimum number of observations in any terminal (leaf) node. Setting $minbucket$ to 1 is meaningless, since each leaf node will (by definition) have at least one observation on it. If we set it to a higher value, say 3, then it would mean that every leaf node would have at least 3 observations in that bucket. The smaller the value of $minbucket$, the more precise our CART model will be. By setting $minbucket$ to too small a value, such as 1, we may run the risk of overfitting the model. If only one of $minsplit$ or $minbucket$ is specified, the code either sets

$minsplit$ to $minbucket \times 3$ or $minbucket$ to $minsplit/3$, as appropriate. For example, by default, if we set the $minbucket$ to 2, $minsplit$ would be $2 \times 3 = 6$.

4.1.3 Define and Select the Bandwidth (λ) of the Kernel

In each of the kernels K_λ, λ is a parameter that controls its width. For example, for the Epanechnikov or tri-cube kernel with metric width, λ is the radius of the support region and for the Gaussian kernel, λ is the standard deviation. λ is the number k of nearest neighbours in k-nearest neighbourhoods, often expressed as a fraction or span $\frac{k}{N}$ of the total training sample.

The bandwidth $\lambda(x)$ has a critical effect on the local regression fit. The goal in choosing the bandwidth is to produce an estimate that is as smooth as possible without distorting the underlying pattern of dependence of the response on the independent variables. In other words, we want $\hat{f}$ to have as little bias as possible and as small a variance as possible. There is a natural *bias-variance* trade-off as we change the width of the (averaging) window, which is most explicit for local averages:

- If the window is narrow, $\hat{f}(x_0)$ is an average of a small number of y_i close to x_0, and its variance will be relatively *large-close* to that of an individual y_i. The bias will tend to be small, again because each of the $E(y_i) = f(x_i)$ should be close to $f(x_0)$.
- If the window is wide, the variance of $\hat{f}(x_0)$ will be small relative to the variance of any y_i, because of the effects of averaging. The bias will be higher, because we are now using observations x_i further from x_0, and there is no guarantee that $f(x_i)$ will be close to $f(x_0)$.

In this work, we are mimicking k-NN kernels using Self-Organizing Maps (SOM) as our kernel trainer in the first stage of unsupervised learning and the λ in our model indicates the size of the SOM or the ideal number of samples mapped on each neuron.

Selecting the size of the SOM is usually an observational result. The goodness of fit is usually based on the node count, as recorded with the number of mapped training samples, and mapping quality, as estimated by the average distance of an object to its corresponding codebook vector. If a proper size of the map has been chosen, the sample distribution is relatively uniform. However, as we discussed in Chapter 1, when multivariate data is encountered, experts' empirical observation might be ineffectual. More importantly, as mentioned above, there lack object functions to be optimized with conventional SOM and the two dimensional plane is not interpretable to observers. There are some modified SOM trying to automatically grow the SOM in the literature. However, each one has different settings in particular to the targeting the problems. These obstacles encourage us to explore new standard of bandwidth selection.

Since the modified SOM we proposed generates another layer of output functions composed with regression trees, the goodness of fit of that layer should be considered as the criterion of selecting the bandwidth. We develop our growing mechanism in Section 4.2 in details.

4.1.4 RTSOM Algorithm

In this section, we conclude the theories that have been discussed into a whole algorithm, *RTSOM*. As can be seen in Algorithm 1, there are three inputs for the *RTSOM* model. They are $\mathbf{X}$, the data matrix in which each row vector is a training case and each column represents a variable, n, is the number of neurons in the SOM (size) and θ include the parameters needed for deciding whether to build a tree in the second stage of the algorithm.

At the beginning, the SOM codebooks will be initialized randomly. When training the first stage SOM, we choose the *learning rate* decreasing linearly from 0.05 and to stop at 0.01, which determines the size of the adjustments during training. And we choose the initial size of the neighbourhood function *radius* in such a way that two-thirds of all distances of the map units fall inside this number. The size of the neighbourhood decreases linearly during training; after one-third of the iterations only the winning unit is being adapted and the algorithm corresponds to *k-means*. At the end of the algorithm, the object will be returned consisted of two layers of SOM network information and they will be used later in the automated growing algorithm.

Algorithm 1 Dual-layer RT-SOM

```
1:  procedure RT-SOM($\mathbf{X}, n, \theta$)
2:      for each node $m_i \in SOMGrid(n)$ do                          ▷ Coodbook initialization
3:          $m_i \leftarrow rand()$
4:      end for
5:      $count \leftarrow nrow(\mathbf{X})$                                 ▷ Get the size of the dataset
6:      while $count \neq 0$ do                                          ▷ First Layer Training
7:          $x_i \leftarrow \mathbf{X}[randperm(count, 1)]$                 ▷ Random sampling without replacement
8:          BMU $i(x_i) \leftarrow \arg\min_{1 \leq j \leq l}\{||\mathbf{x_i} - \mathbf{m}_j||\}$    ▷ Similarity matching
9:          $\mathbf{m}_j(t+1) \leftarrow \mathbf{m}_j(t) + \mathbf{h}_{jc}(t)[\mathbf{x} - \mathbf{m}_j(t)]$   ▷ Updating
10:         $\mathcal{L}[i] \leftarrow append(x_i)$                         ▷ Mapping Records
11:         $count \leftarrow count - 1$
12:     end while
13:     for each Kernel $\kappa_i \in \mathcal{L}$ do                       ▷ Second Layer Training
14:         $\mathcal{T}_i \leftarrow TreeBuilder(\kappa_i, \theta(minbucket), \theta(minsplit))$   ▷ Tree Building Criterion
15:         if $\mathcal{T}_i \neq Null$ then
16:             $TreeGrid \leftarrow \mathcal{T}_i$
17:         else                                                         ▷ If cannot build a tree in this neuron
18:             $TreeGrid \leftarrow m_i$                                ▷ use the codebook as result instead
19:         end if
20:     end for
21:     $retObj \leftarrow list("SOM" : SOMGrid, "RTMap" : TreeGrid)$
22:     return $retObj$                                                  ▷ return the Dual-Layer map
23: end procedure
```

4.1.5 The Fitting Criterion

When the model has been trained, we could now use it for prediction. When , x_{new}, a new data case comes, it will first search in the first layer for mapping the BMU. Then, the BMU will be the position locating the regression tree, $\mathcal{T}$, in the second layer. Finally, the new data will be fitted to the retrieved $\mathcal{T}$ to get the prediction of the targeting variable, $\hat{y}$.

Algorithm 2 Model Utilization: Fitting the RT-SOM

```
1: procedure FIT-RTSOM($x_{new}, RTSOM$)
2:     for each node $m_i \in RTSOM.SOMGrid(n)$ do
3:         BMU $i(x_{new}) = \arg\min_{1 \le j \le l}\{||\mathbf{x_{new}} - \mathbf{m}_j||\}$
4:     end for
5:     $\mathcal{T} \leftarrow RTSOM.TreeGrid[i(x_{new})]$
6:     $\hat{y} \leftarrow \mathcal{T}(x_{new})$
7:     return $\hat{y}$
8: end procedure
```

4.2 Automated Growing Mechanism

As discussed in 3.3.1, while the SOM has proven to be a very suitable tool for detecting structure in high-dimensional data and organizing it accordingly on a 2-D output space, at least two limitations of SOM have to be noted, which are related to the static architecture of this model as well as to the limited capabilities for the representation of hierarchical relations of the data. Obviously, in case of largely unknown input data characteristics, it is far from trivial to determine the network architecture that provides satisfying results. Thus, it certainly is worth considering SOM models that determine the number and arrangement of units during their training process. A growing self-organizing map (GSOM) is a growing variant of the popular self-organizing map (SOM). The GSOM was developed to address the issue of identifying a suitable map size in the

SOM. It starts with a minimal number of nodes (usually 4) and grows new nodes on the boundary based on a heuristic [39]. Further, hierarchical relations may be observed in a wide spectrum of application domains. Thus, their proper identification remains a highly important data mining task that cannot be addressed conveniently within the framework of the SOM [32].

Our research is inspired by GHSOM introduced in [32] in which GHSOM layers the cluster structure of the data. However, the cluster structure of the data could be revealed by the U-Matrix proposed by Ultsch and the related methods [40]. We, on the other hand, focus on the structure of the variables and feature selection from the dataset by integrating the Regression Tree model as the second layer discussed above. In other words, GHSOM tries to build a tree of SOM neuron clusters while we use the tree structure locally in the SOM neurons.

To solve the self-growing issue, our ideology is the same with GHSOM because for exploratory data analysis, a homogeneous distribution of data samples across the map space is desired, allowing to capture finer differences between clusters in more densely populated areas of the data space. However, when it comes to the *global termination criterion*, other detailed parameters must be considered such as the minimum quality of data representation of each unit, denoted as τ_2. In our work, we would like to keep the growing process nonparametric as the category that kernel smoothing and tree-based structure fall under, rather than include other parameters needing manual settings.

As in GHSOM, our growing two-layer grids, starting from a very small map such as 2×2 with a random initialization, tentatively add rows or columns of units during the training process. During each iteration, we evaluate the current built experimental model with some goodness of fit criteria serving as the *stopping criteria* as well. It is important to state that in the description here, we would like to keep the stopping criteria as general and simple as possible because different heuristics might be considered depending on the dataset or problem domain.

4.2.1 Objective Function

In general we can define a local regression estimate of $f(x_0)$ as $f_{\hat{\theta}}(x_0)$, where x_0 is a centroid and $f_{\hat{\theta}}(x_0)$ minimizes:

$$RSS(f_\theta, x_0) = \sum_{i=1}^{N} K_\lambda(x_0, x_i)(y_i - f_\theta(x_i))^2 \tag{4.3}$$

and f_θ is some function with a parameter set θ [24]. For example, Nearest-neighbor methods can be thought of as kernel methods having a more data-dependent metric:

$$K_k(x, x_0) = I \left\| x - x_0 \right\| \leq \left\| x_{(k)} - x_0 \right\|,$$

where $x_{(k)}$ is the training observation ranked the *k-th* in distance from x_0, and $I(S)$ is the indicator of the set S.

To avoid the curse of dimensionality, we modify the objective function to gain the stable state of the learning process for our proposed hybrid model. On the whole, during the growing and training process, in addition to the tree building criterion discussed in earlier sections, it must also ensure that there ought to be no empty neurons in the tree-level and the current trial is better than those tried before.

Formally in the objective function (4.4), a kernel (local) regression solves a separate least squares problem at each centroid, in our case each neuron, $\mathbf{m_j}$:

$$\begin{aligned} \min_{\lambda_t, \theta_{mb}} \quad & \sum_{i=1}^{N} K_{\lambda_t}(\mathbf{m_j}, \mathbf{X_i^{(m_j)}}) \left[\sum_{\mathbf{p=1}}^{\mathbf{P}} \sum_{\mathbf{i \in R_p}} (\mathbf{y_i} - \widehat{\mathbf{f}}_{\mathbf{R_p}}^{(\theta_{\mathbf{mb}}, \mathbf{ms})}(\mathbf{m_j}))^{\mathbf{2}} \right] \\ \text{subject to} \quad & (1)\ \theta_{mb} \geq 2, \\ & (2)\ \theta_{ms} = \theta_{mb} \times 3, \\ & (3)\ \lambda_t \in [2 \times 2, row_{max} \times col_{max}], \end{aligned} \tag{4.4}$$

where K_{λ_t}, the kernel, is each of the SOM neurons with projected training data (introduced above) and $\widehat{f}$ serves as the fitting criterion by the Regression Tree built by the projected kernel data points,

$X_i^{(\mathbf{m_j})}$. For each of the kernels K_{λ_t}, λ_t is the tuning parameter, at the *t-th*, that controls the current trying width of the kernel, in other words, the number of neurons in the SOM, D_t. It also implicitly indicates the visionary node counts which are ideally uniformly distributed. The tuning of λ_t starts from the smallest possible SOM, 2×2, with 2 rows and 2 columns and grows till the row and column size reach the maximum, which are determined by the stopping criteria to be discussed later.

Moreover, $\theta_{mb} \geq 2$, announces that the $minibucket$ when building the trees must be at least 2, which has been explained in Section 4.1.2 along with its relationship with θ_{ms}, the $minsplit$ indicated in constraint (2). For simplicity and accuracy, by default, we set this parameter to 2, which means the model will try its best find the finest partition in each kernel. Given a kernel is partitioned into P regions, $R_1, R_2, ..., R_P$, the kernel fitting function on each region R_p provided by the regression tree model is formally $\widehat{f}_{R_p}(x) = \sum_{p=1}^{P} c_p I(x \in R_p)$ [24]. As discussed in Section 3.2, the optimized estimation $\widehat{c}_p$ in each region R_p is generated by the mean value of the responses y_i to the corresponding training instances x_i, formally $\widehat{c}_p = ave(y_i|x_i \in R_p)$.

4.2.2 Goodness of Fit

In studies of linear regression, one often focuses on the regression coefficients. One assumes the model being fitted is correct and asks questions such as how well the estimated coefficients estimate the true coefficients. For example, one might compute variances and confidence intervals for the regression coefficients, test significance of the coefficients or use model selection criteria, such as stepwise selection, to decide what coefficients to include in the model. The fitted curve itself often receives relatively little attention.

In local regression, we have to change our focus. Instead of concentrating on the coefficients, we focus on the fitted curve. A basic question that can be asked is "how well does $\hat{f}(x)$ estimate the true mean $\hat{f}(x)$?". When variance estimates and confidence intervals are computed, they will be computed for the curve estimate $\hat{f}(x)$. Model selection criteria can still be used to select variables for the local model. But they also have a second use, addressing whether an estimate $\hat{f}(x)$ is satisfactory or whether alternative local regression estimates, for example, with different bandwidths, produce better results.

In traditional local regression studies, one is faced with several model selection issues: Variable selection, choice of local polynomial degree and smoothing parameters [41]. Since we are adopting Tree based local leaner, variable selection has been implied in tree generation and we do not have local polynomial degrees to select. Thus, we must have a goodness of fit to the regression trees in order to evaluate the quality of the localization and the stopping mechanism of the growing.

It is important to remember that no one diagnostic technique will explain the whole story of a dataset. Rather, using a combination of diagnostic tools and looking at these in conjunction with both the fitted curves and the original data provide insights into the data.

After testing with some regression dataset and our case study, to be discussed in Chapter 4, we are considering the following three factors: full tree-layer, R-Square and homogeneity. It is intuitive to explain that a local regression tree would provide a better prediction value and insight knowledge from the neighbourhood. Therefore, there must be a regression tree built in each neuron based on the local tree degree discussed above. Homogeneity in SOM training means that ideally, the sample distribution of the training data mapped to the map is relatively uniform. Large values in some map areas suggests that a larger map would be beneficial. Empty nodes indicate that the map size is too large for the number of samples.

The coefficient of determination, R^2, is the proportion of the total (corrected) sum of squares (SS) of the dependent variable "explained" by the independent variables in the model:

$$R^2 = \frac{SS(Regr)}{SS(Total)} = \frac{SS(Regr)}{\sum y_i^2},$$

where $SS(Total)$ is used to denote the corrected sum of squares of the dependent variable and $SS(Regr)$ measures the additional information provided by the independent variable [42].

The objective is to select a model that accounts for as much of the variation in Y as is practical. Since R^2 cannot decrease as independent variables are added to the model, the model that gives the maximum R^2 will necessarily be the model that contains all independent variables. The typical plot of R^2 against the number of variables in the model starts as a steeply upward sloping curve, then levels off near the maximum R^2 once the more important variables have been included. Thus, the use of the R^2 criterion for model building requires a judgement as to whether the increase in R^2 from additional variables justifies the increased complexity of the model. The subset size is chosen near the bend where the curve tends to flatten.

4.2.3 Stopping Criteria

In our previous work [43], imitating many other SOM applications, we manually set the λ to $\frac{N}{50}$ assuming that the sample space N is large enough so that, averagely speaking, each neuron should be mapped with 50 training samples. Apparently, this is a rudimentary selection based on experiments and observations and does not guarantee to be a good choice with another layer of regression trees attached.

In this work, we introduce two stopping criteria, 4.5 and 4.6, considering more combinational sampling possibilities to be explored (spreading and mapping) while letting the regression trees play a role as much as possible. Note that these two are all-purpose factors and more heuristics can absolutely be included according to different applications.

$$\forall \mathbf{m_j} \in SOMGrid_t, \qquad \exists \widehat{f}_{\theta_{mb}} \in TreeGrid_t. \tag{4.5}$$

$$\overline{R^2}_{t_{cur}}(\sum_{d_{t_{cur}}=1}^{D_{t_{cur}}} \widehat{f}_{\theta_{mb}}) \geq median \left\{ \forall \overline{R^2}_t(\sum_{d_t=1}^{D_t} \widehat{f}) | t \in (0, t_{cur}-1) \right\}. \tag{4.6}$$

The first stopping criterion, 4.5, indicates that the growing procedure attempts ensure that each neuron could have a tree trained based on its historical mapped training data. In the previously demonstrated single trial of Algorithm 1, when a neuron does not have the eligibility of building a local regression tree, that particular neuron would yield to the original codebook $\mathbf{m_j}$. However, an empty neuron implies that there might be too many neurons for mapping the data and it is the cause of overfitting.

The second stopping criterion, 4.6, indicates the rule of applying the goodness of fit, R-squares, as the stopping criterion. A compromised scheme would be, in the current trial, t_{cur}, if the $average$ R-squares of all the trees ($D_{t_{cur}}$ in total) is less than the median one of average R-squares of those (D_t in total) in the previous tried models (not getting any better), the growing process stops and the preceding model is retrieved as the final result. We choose the median value because, on one hand, we would like to keep exploring more kernels with larger maps because it is obvious that with the growing size of the map, local trees can achieve increasingly better local predictivity. On the other hand, there is no point to continue searching if the global effect is not getting better.

It is important to point out that the stop criteria are logically conjunctional, which means that the process stops when neither of them happens at the same time. This setting ensures the trade-off between *exploration* (trying bigger maps) and *exploitation* (local regression trees). Therefore, when the growing process stops, it is possible that the previous investigated model (the one we are ultimately using) include neurons failed to train their regression trees and substituted with the codebook from the first layer.

4.2.4 Growing RTSOM Algorithm

To sum up, the growing process is presented in the Algorithm 3 below. Algorithm 1 is called inside of Algorithm 3. The input parameters of Algorithm 3 are the same as Algorithm 3 except that the map size is dynamically fetched by its row and column. This also means that there is no extra parameter added to the model to be decided before and during the training process.

Algorithm 3 Self-Growing RT-SOM

```
 1: procedure GROW-RTSOM($\mathbf{X}, r_{init} = 2, c_{init} = 2, \theta$)
 2:     do
 3:         $model_{cur} \leftarrow$ RT-SOM($\mathbf{X}, r_{init} \times c_{init}, \theta$)
 4:         for each node $\mathcal{T}_i \in RTSOM.TreeGrid$ do
 5:             $R^2_{cur}[i] \leftarrow R^2(\mathcal{T}_i)$                ▷ Calc the $R^2$ for each tree
 6:             if $\mathcal{T}_i = Null$ then                 ▷ Count empty neurons in Tree-layer
 7:                 $emptyCount++$
 8:             end if
 9:         end for
10:         $\overline{R^2}_{cur} \leftarrow ave(R^2_{cur})$
11:         if $r = c$ then                                  ▷ add one more row
12:             $r \leftarrow r + 1$
13:         else                       ▷ add one more column to match the row
14:             $c \leftarrow c + 1$
15:         end if
16:     while criterion 4.5 AND 4.6 are NOT satisfied
17: end procedure
```

Chapter 5

Experimental Results and Analysis

We evaluate our proposed Growing SOM Kernel Smoother in this chapter with two test cases. Both of them are multivariate regression problems. The first case, Building Energy Efficiency, is rather small comparing to our EV range anxiety case study. In this test, we aim to visualize the SOM tree-layer and the variable importance ranking as a whole. Therefore, the interpretative power of hybridizing the tree-layer could be revealed.

We come back to our EV range anxiety case study in the second test case. Through this test, we aim to demonstrate that our proposed hybrid model could resolve the complex power consumption estimation problem practically. Applying our Growing RT-SOM testifies that hybridizing localized regression based on the data partitioning could not only provide more accurate predictions but also reveal the insights of the data, preserving a global trained bushy tree from losing meaningful knowledge extraction. Numerical evaluations tested on various terrestrial trip profiles are presented to demonstrate the feasibility of the proposed scheme.

5.1 Experiment 1: Building Energy Efficiency

Energy analysis is performed using 12 different building shapes simulated in Ecotect. The buildings differ with respect to the glazing area, the glazing area distribution, and the orientation, amongst other parameters. The author simulated various settings as functions of the aforementioned characteristics to obtain 768 building shapes. The dataset comprises 768 samples and 8 features, aiming to predict two real valued responses. We focus only on regression in our work but it can also be used as a multi-class classification problem if the response is rounded to the nearest integer [44]. Each of the 768 simulated buildings can be characterized by eight building parameters (to conform to standard mathematical notation and facilitate the analysis in this work, henceforth these building parameters will be called input variables and will be represented with X) which we are interested in exploring further. Also, for each of the 768 buildings we recorded Heating Load (HL) and Cooling Load (CL) which are called output variables and will be represented with y. Table 1 summarizes the input variables and the output variables in this study, introduces the mathematical representation for each variable, and indicates the number of possible values.

The whole growing process of the map stops very fast from 2×2 to 3×4. It makes sense since only 600 samples of the total 768 were used for training and ideally, each neuron should be mapped with around 50 data in average, which is a good empirical choice used in many SOM applications such as [43].

As can be seen from the regression tree map Fig. 5.1, the leaf value only represents the possible predictive heating load, $\hat{y_9}$. An overall measure of variable importance is the sum of the goodness of split measures for each split for which it was the primary variable. In each neuron, these are scaled to sum to 100 and the rounded values are shown, omitting any variable whose proportion is less than 1%. It can also be noted that a variable may appear in the tree many times. This does not

Table 5.1: Mathematical representation of the input and output variables to facilitate the presentation of the subsequent analysis and results.

Math representation	Input or output variable	Number of possible values
X1	Relative compactness	12
X2	Surface area	12
X3	Wall area	7
X4	Roof area	4
X5	Overall height	2
X6	Orientation	5
X7	Glazing area	8
X8	Glazing area distribution	6
Y1	Heating load	586
Y2	Cooling load	636

necessary mean that if a variable appears more than another, it is then more important. The reason for this is that the trees in Fig. 5.1 only demonstrate the splitting decisions based on the regression objectives described in Chapter 3. For example in neuron 6, x_2 only appeared once comparing to x_1, x_3 and x_7 twice, but x_2 turned to be very important amongst the others according to Fig. 5.2.

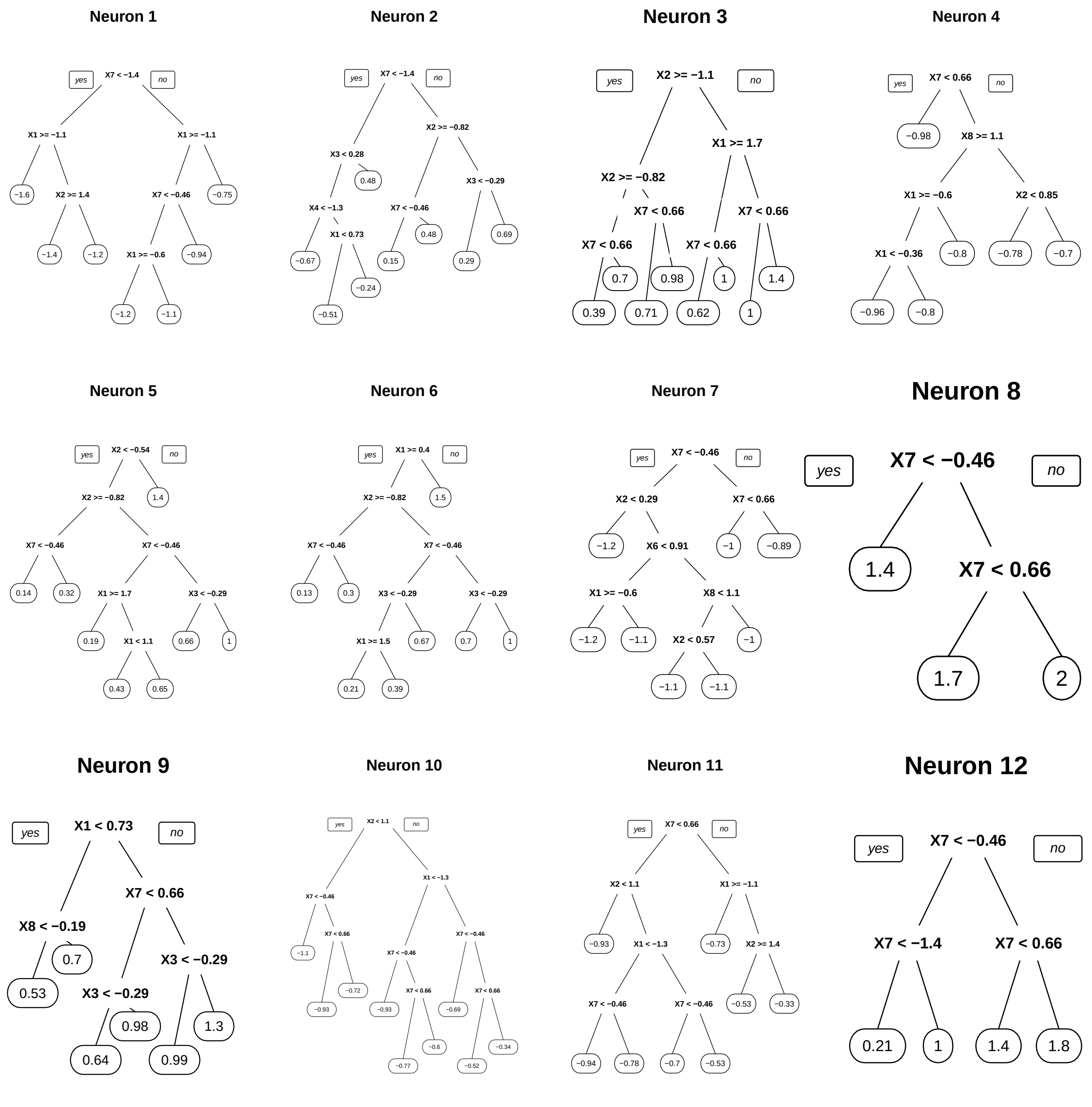


Figure 5.1: Self-Organizing Regression Tree Map

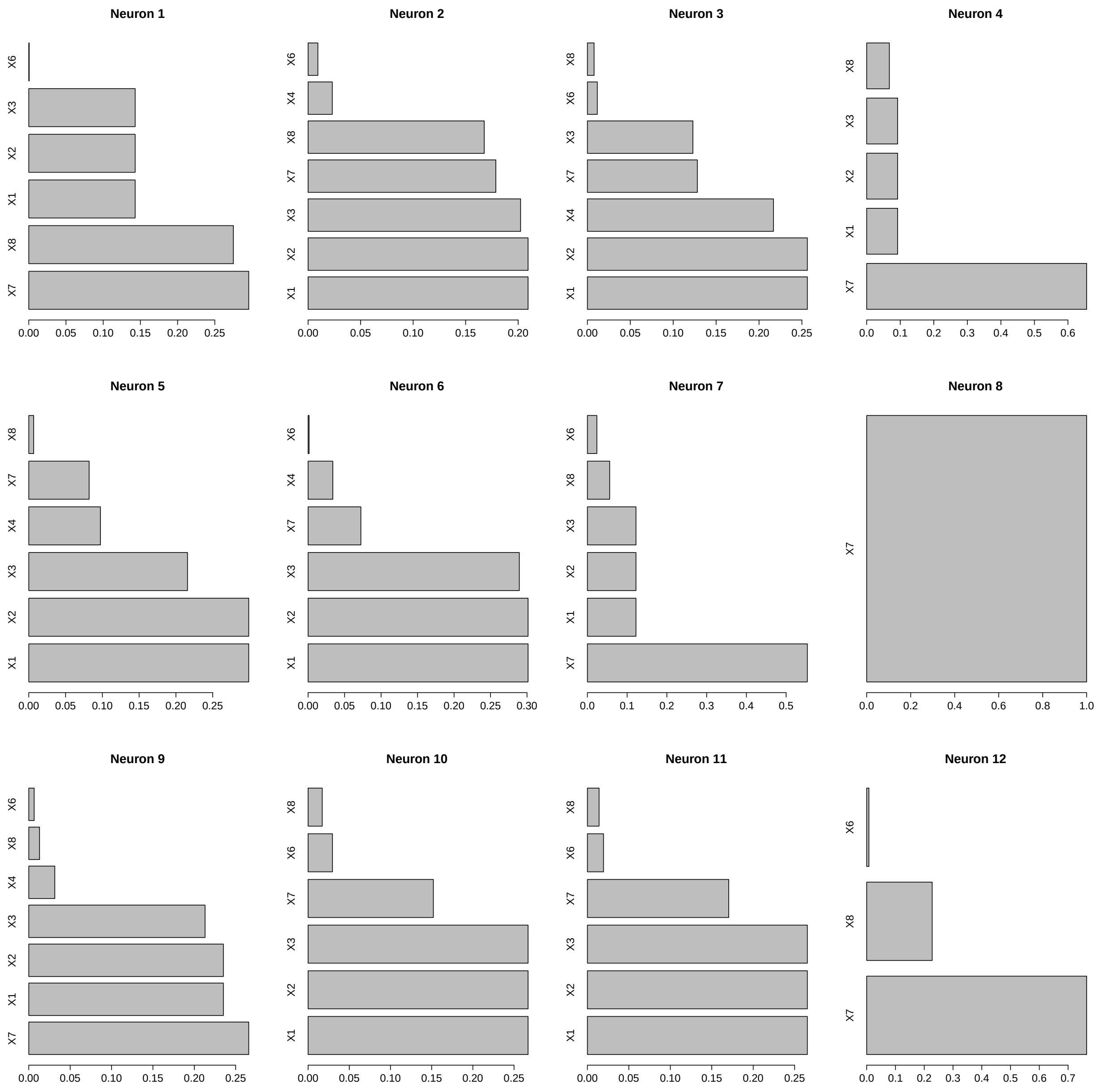


Figure 5.2: Variable Importance of Each Neuron

5.2 Case Study: EV Range Anxiety

5.2.1 Data Description

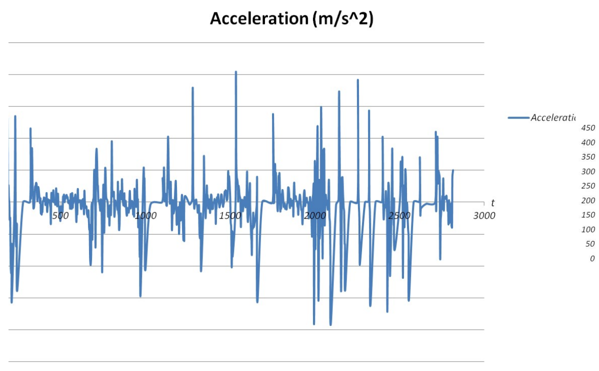
We test our Hybrid Model using the data collected from the ChargeCar project of the CREATE Lab at Carnegie Mellon University [45]. The dataset is crowd-sourced by actual EV drivers in North America and Europe with real-world commutes in terms of speed, distance, traffic conditions, hills and driving behaviour.

The dataset includes only 421 EV trips and in order to evaluating the proposed model with flexibility, sensitivity and robustness, we generate a larger dataset of time-series data by applying three levels of randomness among the original commutes. A trip profile is firstly randomly selected from the 421 trips, a window of portional random size then cuts out a period of the time series starting from a random time point appending the original static features of the selected trip. We generate 5000 new trips for training our Hybrid Model off-line and 500 new trips for testing. It must be explained that knowing the exact time series before hand the prediction is unrealistic but we assume that the statistical features can be estimated as the predictive inputs such as the traffic status marked as color scheme on Google maps. And as can be seen from the system architecture in Fig. 4.1, it is the trip-feature vectors of new trips that are fed for prediction rather than raw trip profiles.

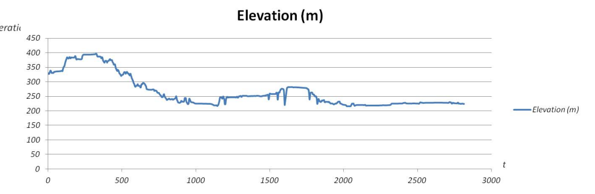
5.2.2 Data Fusion and Normalization

Since some sensory data might be sampled as time-series data, they need to be properly represented and merged with other static sensory data sources so that (1) all related sensory variates could be considered in model training together; and (2) common similarity measurements such as Euclidian Distance can still function well enough to find the similar historical cases and form clusters.

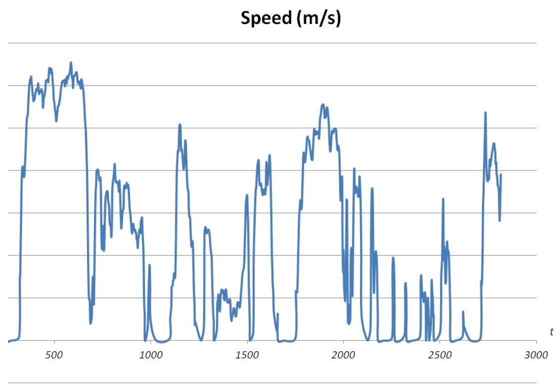
The time series data in a trip profile is illustrated in Fig. 5.3a, Fig. 5.3b, Fig. 5.3c and Fig. 5.3d.



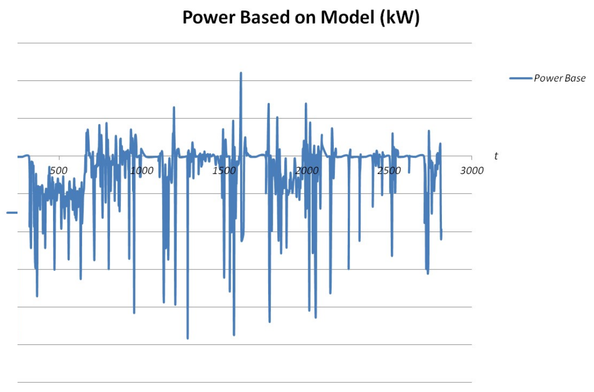
(a) Acceleration time series data (m/s^2)



(b) Elevation time series data (m)



(c) Speed time series data (m/s)



(d) Power time series data (kW)

Figure 5.3: EV trip time series data

The most straightforward representation of time-series data is the time domain form, then distances between time series relate to differences between the time-ordered measurements themselves. However, taking the actual values of the time series as the model training input, also referred in [46] as instance based representation, presents several drawbacks such as it is sensitive to the noise and length of time-series data [47]. Clustering becomes even more challenging when tackling multivariate high dimensional data [48, 49]. Fortunately, feature based time series data mining is a beneficial alternative in many ways and it thereby could transform the temporal problem to a static one.

In this work, four most common statistical features, mean μ, standard deviation σ, skewness SKEW and kurtosis KURT, are considered following the previous work in [47] to represent an univariate time-series data. The mathematical equations for these features are:

$$\mu_{T_i} = \frac{\sum_{t=1}^{N}(y_i(t))}{N}, \tag{5.1}$$

$$\sigma_{T_i} = \sqrt{\frac{\sum_{t=1}^{N}(y_i(t) - \mu)^2}{N}}, \tag{5.2}$$

$$SKEW_{T_i} = \frac{\sum_{t=1}^{N}(y_i(t) - \mu)^3}{N\sigma^3}, \tag{5.3}$$

$$KURT_{T_i} = \frac{\sum_{t=1}^{N}(y_i(t) - \mu)^4}{N\sigma^4}, \tag{5.4}$$

where skewness and kurtosis contain information on the shape of the distribution of the time-series values. Skewness is a measure of symmetry, or more precisely, the lack of symmetry. A distribution, or data set, is symmetric if it looks the same to the left and right of the center point. Kurtosis is a measure of whether the data are peaked or flat relative to a normal distribution.

It is assumed in this paper that EVs are capable of collecting data from their own internal communications bus such as Controller Area Network (CAN) and Local Interconnect Network (LIN) and also able to penetrate Internet data such as temperature, GPS and traffic into the power consumption estimation. The small variation of static data sensors is ignored for a single trip. Namely, for example, temperature during a trip is assumed to be static.

Sensors measuring elevation, speed, acceleration and power are generating time-series data. And the Area Under the power (in kW) Curve (AUC in kWh) is utilized as the dependent variable $\widehat{y}$. As discussed above, four statistical features are extracted as regressors from each of the other three time-series variates. Therefore, twelve features consist the first part of the Trip-Feature Vector. Other static factors considered include distance, trip run time, temperature, total weight of loads, tire pressure, frontal area. The above mentioned factors related to EV power is listed in table 5.3.

Note that the V_i index of each variable table is corresponding to the codebook plotting in Fig. 5.6 and tree-layer plottings in Fig. 5.7.

Table 5.2: Numerical Accuracy Evaluation

Source	Trip-Features(V_i)			
$TS_{Elevation}$	$Mean_{T_{Elev}}(V_1)$	$SD_{T_{Elev}}(V_2)$	$Skew_{T_{Elev}}(V_3)$	$Kurt_{T_{Elev}}(V_4)$
TS_{Speed}	$Mean_{T_{sp}}(V_5)$	$SD_{T_{sp}}(V_6)$	$Skew_{T_{sp}}(V_7)$	$Kurt_{T_{sp}}(V_8)$
$TS_{Acceleration}$	$Mean_{T_{Acc}}(V_9)$	$SD_{T_{Acc}}(V_{10})$	$Skew_{T_{Acc}}(V_{11})$	$Kurt_{T_{Acc}}(V_{12})$
Static Variables	distance(V_{13})	run time(V_{14})	Temperature(V_{15})	total loads(V_{16})
Static Variables	Tire Pressure(V_{17})		Frontal Area(V_{18})	
Targets	Power Consumption(V_{19})		Power Conservation(V_{20})	

Table 5.3: Specification of EV trip factors

Before feeding the training data into the model, it is very important to normalize the features. There can be instances found in data frame where values for one feature could range between 1 and 10 and values for other feature could range from 1 to 1000. In scenarios like these, owing to mere greater numeric range, the impact on response variables by the feature having greater numeric range could be more than the one having less numeric range, and this could, in turn, impact prediction accuracy. The objective of normalization is to improve predictive accuracy and not allow a particular feature impact the prediction due to large numeric value range. Thus, we must scale the values under different features such that they fall under common range. Because the physics and mechanisms are very complex when including many variables, we do not make any assumptions on the relative importance or relationships of the variables and we choose the standard score, $\frac{X-\mu}{\sigma}$, for normalization.

Eventually, the Trip-Feature Vector is constructed as follows:

$$\begin{array}{c} Trip-Feature \\ Vector \end{array} = \begin{bmatrix} Mean_{T_{Elev}} \\ SD_{T_{Elev}} \\ Skew_{T_{Elev}} \\ Kurt_{T_{Elev}} \\ \dots \\ Distance \\ RunTime \\ \dots \\ Pow_{Con} \\ Pow_{Save} \end{bmatrix} \begin{array}{l} \left.\begin{array}{l} \\ \\ \\ \\ \\ \end{array}\right\} \begin{array}{c} Features\ Extracted \\ from\ TSs \end{array} \\ \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} Static\ Features \\ \left.\begin{array}{l} \\ \\ \end{array}\right\} Predicting\ Targets. \end{array}$$

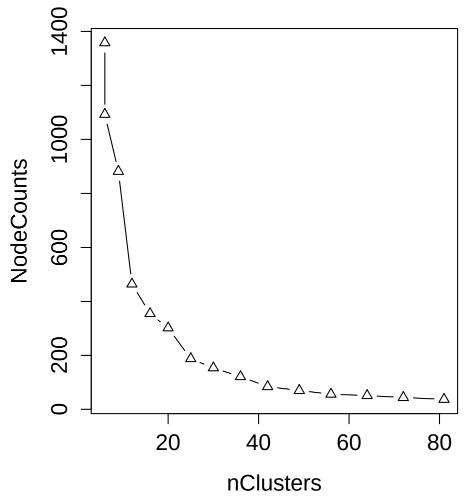
Trip profiles are now represented by he Trip-Feature Vectors and they are fed into the proposed Hybrid Model for training, prediction and further knowledge extraction.

5.2.3 Training Process and Resulted Model

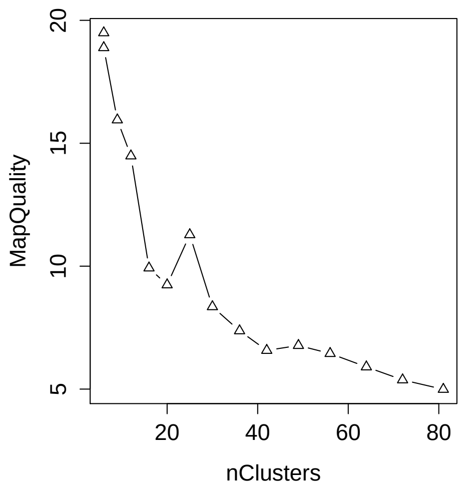
In this section, we demonstrate the growing process for model selection using our proposed Algorithm 3. The training process and results of our proposed hybrid model, *RTSOM*, are also shown and explained in details.

In our previous work in [43], we chose the empirical setting of the map size 10×10 and it demonstrated an relatively uniform sample distribution, which was ideal. From the Building Energy Efficiency experiment, we have also seen that averagely speaking, a neuron mapped by 50 samples could generate a good result. In this thesis, our growing algorithm gives a 9×9 map. As can be seen in Fig. 5.4, the ultimate model consist 81 neurons. This is better because a smaller map means less complexity of computation and memory. Further, a bigger map have more chance of overfitting and being less interpretative because with less local mapped samples, there might exist more *stumps* (a tree with only one splitting) in the tree-layer. And furthermore, this 9×9

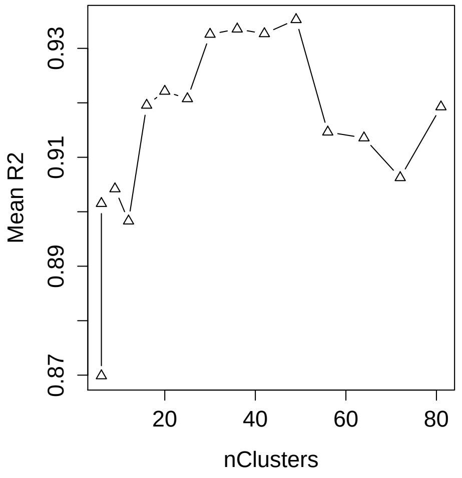
model is ideal because (1)shown in Fig. 5.4a and Fig. 5.4b, the standard deviations (SD) of the *NodeCounts* and *MapQuality* have been decreased and converged, which means that homogeneity has been achieved among neurons; (2)plus, the average R^2 of the 9×9 model is around 0.92 in Fig. 5.4c, which is also the evidence that this is a good model to use.



(a) SD of node counts through growing

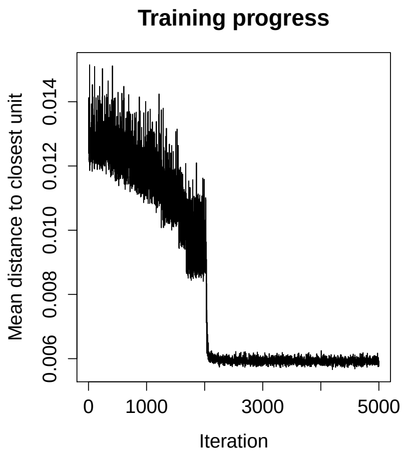


(b) SD of map quality through growing

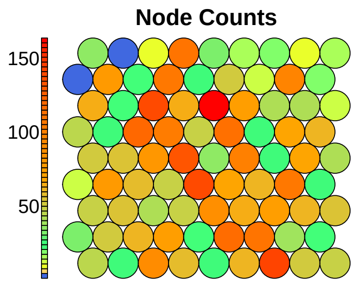


(c) Average value of R^2 through growing

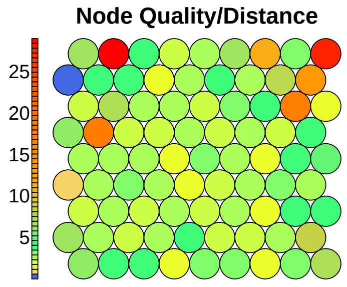
Figure 5.4: Growing Process



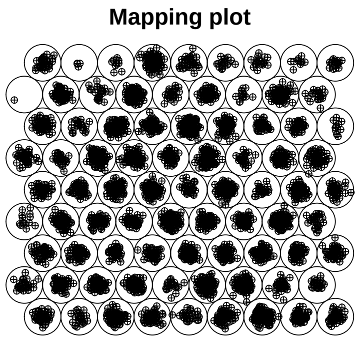
(a) Convergence of SOM training



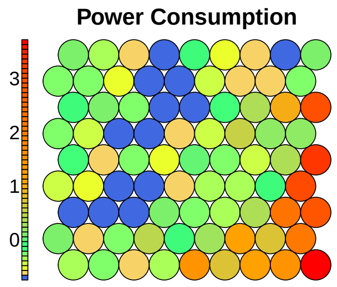
(b) Color Scheme for 1st-layer Node Count



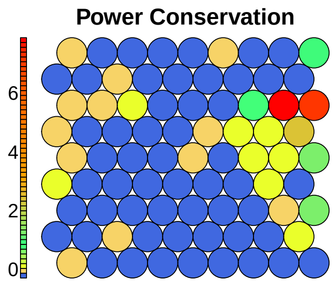
(c) Color Scheme for 1st-layer Node Quality



(d) Mapping density for 1st-layer



(e) Heatmap of power consumption property



(f) Heatmap of power conservation property

Figure 5.5: Final selected model

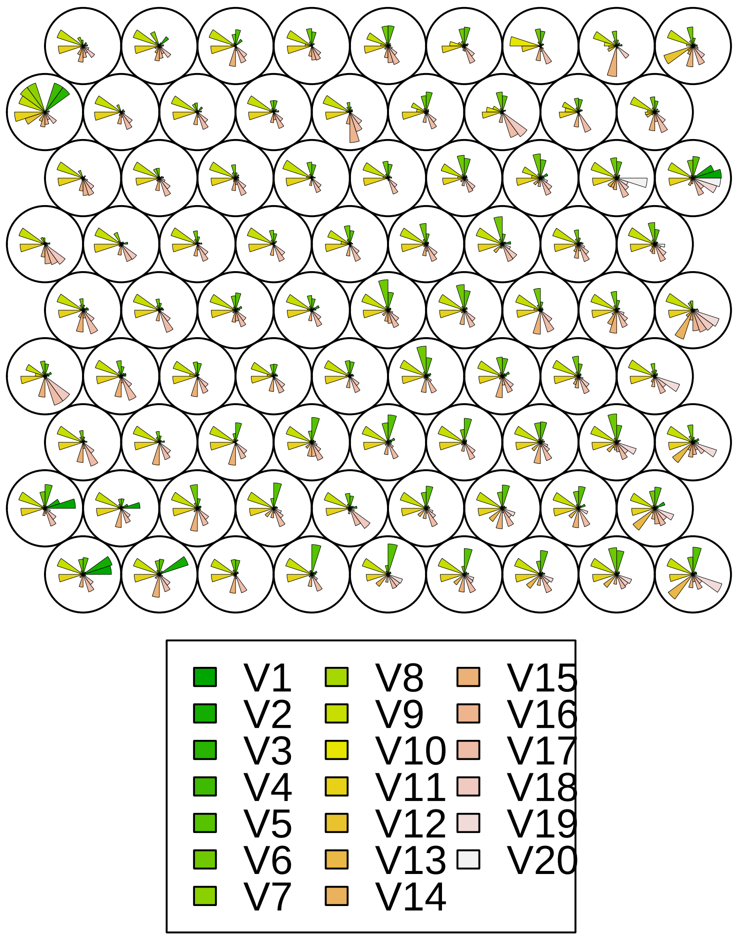
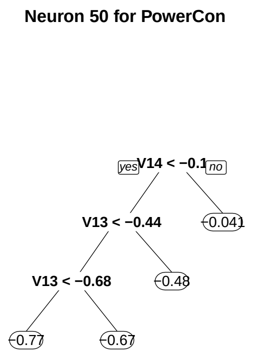


Figure 5.6: SOM Training

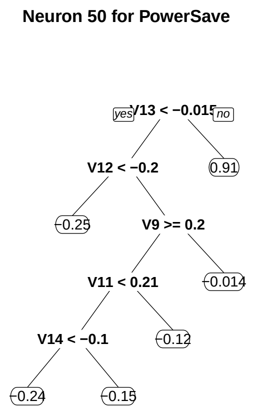
We now take a close look at the selected 9×9 model. The training iterations progress of the final selected 9×9 SOM model can be seen in Fig. 5.5a. The distance from each neuron codebook to the samples represented by that node has been continually reduced, which means that this distance has converged to a minimum plateau and no more iterations are required. The quality of the training is plotted in Fig. 5.5c. It shows that the mean distances of objects mapped to a unit to the codebook vector of that unit are generally small enough so that the codebook vectors represent the objects sufficiently well.

In Fig. 5.5b, the background color of a unit corresponds to the number of samples mapped to that particular unit; they are reasonably spread out over the map. All neurons have some samples mapped (empty nodes indicate that the map size is too big for the number of samples). Intuitively, the sample mapping distribution is illustrated in Fig. 5.5d. Take a close look at the first neuron in the second row in Fig. 5.5d, it is a neuron without a regression tree trained in the map since obviously, there was only one historical data mapped to it. As pointed out in Chapter 3, this is because after training the 9×9 model, the stopping constraints in our objective function are not fully satisfied. In this case, the codebook of this neuron from the first layer will be used for prediction instead.

Property *heatmap* is a very important visualisation tool for SOM. The use of a weight space view as in Fig. 5.6 that tries to view all dimensions on the one diagram is unsuitable for a high-dimensional SOM. A SOM heatmap allows the visualisation of the distribution of a single variable across the map. Typically, a SOM investigative process involves the creation of multiple heatmaps, and then the comparison of these heatmaps to identify interesting areas on the map. However, our regression tree layer could provide a more descriptive and revealing illustration. We plot Fig. 5.5e and Fig. 5.5f here in order to briefly understand the distribution of our two predicting targets. Similar to the Building Energy Efficiency experiment, we could achieve the detailed information of the tree-layer neurons. Nonetheless, avoiding an unclear huge map of trees, only two randomly chosen trees are plotted in Fig. 5.7 for illustration who are from the tree-map targeting Pow_{Con} and that targeting Pow_{Save} respectively. Note that the values of the branch variables and leaves are still scaled because the model is under training at this stage. The leaves in Fig. 5.7a represents V_{19} and Fig. 5.7b represents V_{20} referring to Fig. 5.3. By comparison, we could tell that completely two different trees are generated because they are fitting different targets. In addition, concise structural knowledge is extracted from the kernels accordingly, which is also an important role regression models should play.



(a) Neuron 50 of Pow_{Con} Tree



(b) Neuron 50 of Pow_{Save} Tree

Figure 5.7: Comparing the Tree-layer of Two Different Targets.

It is worth pointing out that, as illustrated in the trip-feature vector, there are two targets fed into the first layer of SOM, Power Consumption (Pow_{Con}) and Power Conservation (Pow_{save}). Indeed, since the first layer performs unsupervised learning, no targets variables need to be assigned. However, regression has to be achieved in the second layer by supervised learning process (CART). Therefore, the whole trip-feature vector (20 variables in total) would be fed for training the unsupervised layer while only the regressors (18 variables) and one target are used for building the supervised layer. In this case study, it also has to be clarify that the growing process shown in Fig. 5.4 is based on evaluation of trees targeting Pow_{Con} because our original goal is to solve the range anxiety problem by estimating the power consumption of EV trips. For simplicity, we just use the trained 9×9 SOM as the first layer to train the tree-layer of Pow_{save}. And the results shown in Fig. 5.9 proves that the prediction accuracy is fulfilling using the 9×9 RTSOM model.

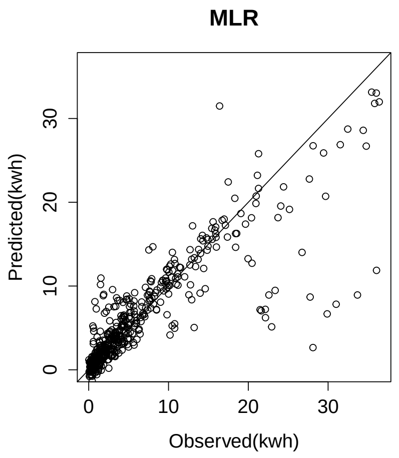
5.2.4 Model Assessment

We have already demonstrated the interpretive ability of our proposed hybrid model. In this section, we evaluate the prediction accuracy numerically still on the EV case study.

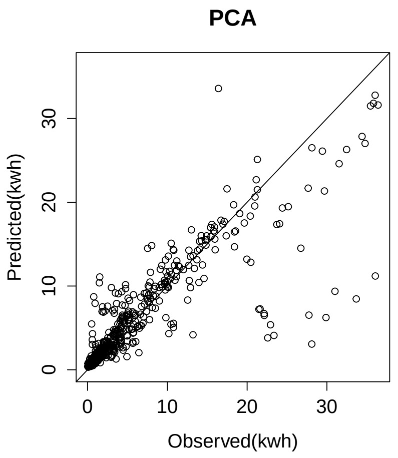
Cross-validation is used when tough decisions have to made for partitioning the dataset when there are limited data available. However, we are in a data-rich situation since a random data generation approach has been introduced in earlier Section 5.2.1. The best approach is hence the traditional validation and as discussed also in Section 5.2.1, 500 generated testing trip profiles are used for evaluation. That is 10% of the dataset, namely, with 5000 training samples.

Intuitively and typically, aiming to predict continuous response variables by multiple predictor variables, Multiple Linear Regression (MLR) [50] and Principal Component Analysis/Regression (PCA/PCR) [51] are adopted globally on the whole dataset. They are therefore picked in this work as comparison to local regression methods. We also tried different possibilities of adopting SOM including applying the SOM directly as regression model [30] and applying 1st-NN [52] to the local clustered data instead of regression trees as comparisons.

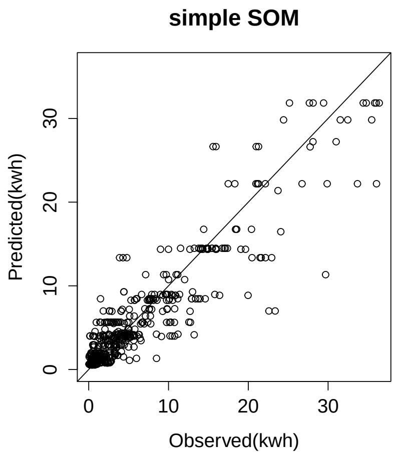
Scatter graphs are used to provide a perceptual intuition of the predicting accuracy. In our scatter plottings, abscissa axes represent the actual measured values and the vertical axes represent the predicted values adopting different methods. Illustrated by the validation results for predicting Power Consumption (V_{19}) in the scatter graphs in Fig. 5.8, it can easily tell that the global methods, MLR and PCA, perform acceptably on short trips while they generate intolerable results on relatively longer trips. Local regression methods based on SOM also demonstrate agreeable accuracy on short trips and achieve better but mediocre outcomes when predicting longer trips. It is obvious that the proposed RTSOM Hybrid Model beats the others on all ranges. Favourable results are produced not only for short trips but more importantly, the prediction accuracy is satisfactory for long trips as well. Similar validation results for predicting Power Conservation (V_{20}) can be found in Fig. 5.9. Again, our proposed algorithm outperforms the other methods.



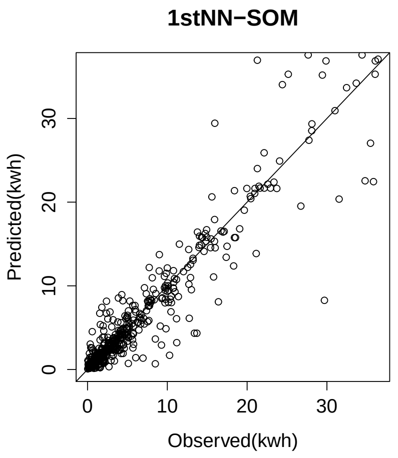
(a) Accuracy of MLR



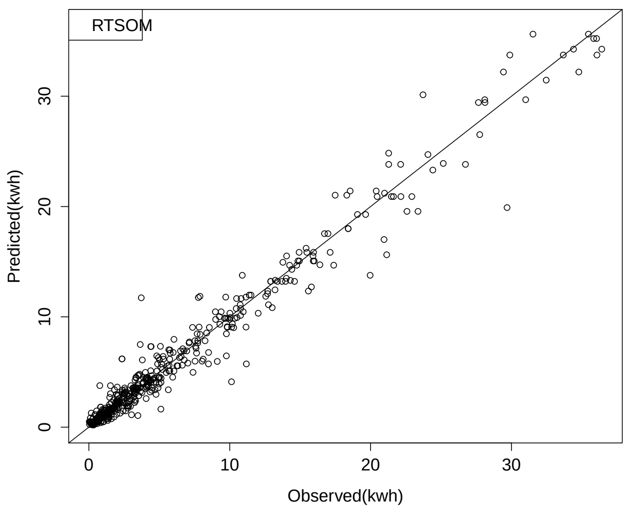
(b) Accuracy of PCA



(c) Accuracy of simpleSOM

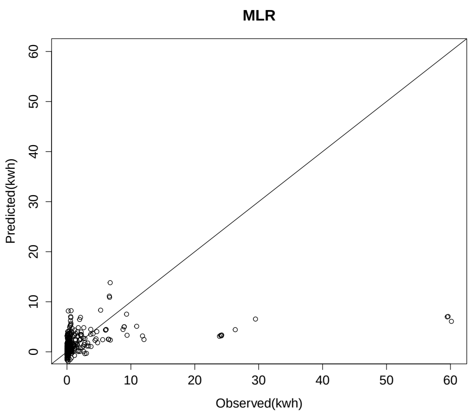


(d) Accuracy of 1stNN-SOM

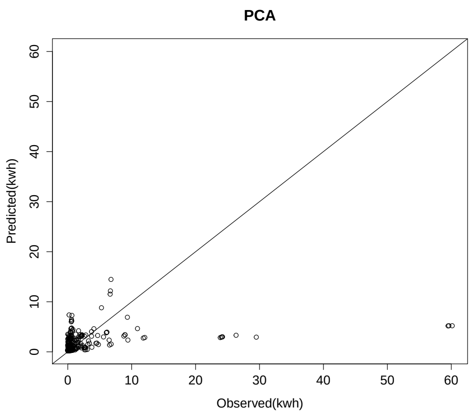


(e) Accuracy of RT-SOM

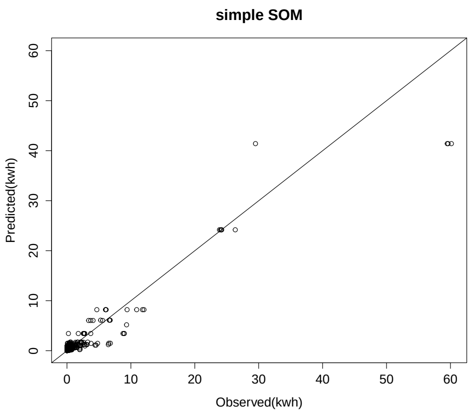
Figure 5.8: Cross Validation for Power Consumption



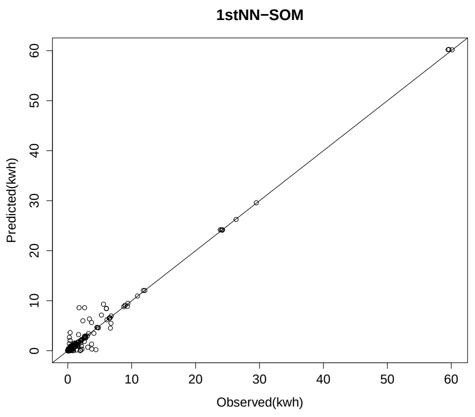
(a) Accuracy of MLR



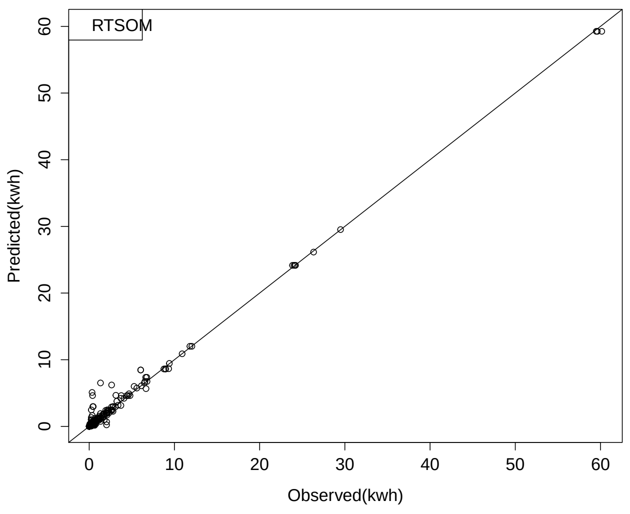
(b) Accuracy of PCA



(c) Accuracy of simpleSOM



(d) Accuracy of 1stNN-SOM



(e) Accuracy of RT-SOM

Figure 5.9: Validation for Power Conservation

We also evaluate the prediction accuracy numerically using metrics such as the Mean Absolute Error (MAE), the Root Mean Square Error (RMSE) and the Mean Absolute Percentage Error (MAPE). Formally, assuming modelled with N testing samples and predictions $P_i(i = 1, 2, ..., N)$ and pairwise matched observations $O_i(i = 1, 2, ..., N)$, they are calculated as as follows:

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |e_i| \,, \tag{5.5}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} e_i^2}, \tag{5.6}$$

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}(\frac{|e_i|}{|O_i|}), \tag{5.7}$$

where $e_i = P_i - O_i$ indicating each of the prediction errors. MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. MAE is a linear score which means that all the individual differences are weighted equally in the average. RMSE is a quadratic scoring rule which measures the average magnitude of the error. Since the errors are squared before they are averaged, RMSE gives a relatively high weight to large errors. This means RMSE is most useful when large errors are particularly undesirable. MAPE expresses accuracy as a percentage of the error and it is easier to be understood than the other statistics to the ultimate EV users.

As can be seen from the numerical performance comparison in table 5.4, global methods including MLR and PCA, in terms of MAE and RSME, are generally less accurate on both targets than the other three kernel regression models measured. Especially when predicting the Pow_{Save} using MLR and PCA, RMSE index highlights the large errors from predicting trips that consume more energy corresponding to Fig. 5.9a and Fig. 5.9b. Applying 1st-nearest-neighbour on SOM kernels (1stNN-SOM) performs slightly better than utilizing SOM as regression directly (simple SOM) measured by MAE and RSME, but when it comes to MAPE, the prediction results of simple SOM and 1stNN-SOM, 88.341% and 47.003% are far beyond acceptable. These are caused by the

Table 5.4: Numerical Accuracy Evaluation

Accuracy Measurements					
Methods	MAE (Pow_{Con})	RMSE (Pow_{Con})	MAE (Pow_{Save})	RMSE (Pow_{Save})	MAPE (Pow_{Con})
MLR	1.95368	3.95865	1.74305	5.16654	77.473%
PCA	2.06659	4.28323	1.63533	5.47941	75.736%
Simple SOM	1.91543	3.01570	0.56985	1.81187	88.341%
1stNN-SOM	1.35387	2.65164	0.23387	0.70062	47.003%
RTSOM	0.76857	1.33579	0.17264	0.53064	26.112%

large margin of errors when using simple SOM and 1stNN-SOM to predict large trips.

The proposed Hybrid Model, RTSOM, outperforms all the other methods in all measurements. Specifically speaking, the MAE shows that there is only averagely around 0.7 kWh prediction error. The RMSE demonstrates that the power consumption of longer trips are estimated significantly more accurate than the two global models and the other two kernel methods. Last but not least, with the MAPE of 26.112%, our RT-SOM could eventually provide the EV users with a tolerable power consumption estimation, which beats their conservative EV utility.

Chapter 6

Conclusion and Future Expansion

In this thesis, we aim to tackle multivariate regression and data mining problems. Classical methods such as kernel smoothing and PCA have their limitations and assumptions. Therefore, each of them alone cannot effectively solve both of the problems facing the big data problems. We propose a novel Kernel Smoothing method hybridizing SOM and CART into a dual-layer network in which unsupervised learning is done to form kernels and regression is locally applied to achieve the prediction on the targeting variables and extract knowledge from the neighborhoods.

6.1 Conclusion

We investigated the advantages and the disadvantages of SOM and CART and found that these two have complementary advantages. We then demonstrated the proof that SOM neurons can be used as kernel smoothers and it overcomes the limitation of conventional methods on high dimensional data. Mimicking the typical kernel smoothing methods, important components including local setting of building trees and bandwidth selections are discussed. After that, pieces are then assembled to form our RTSOM hybrid algorithm.

Further, we developed the automated growing mechanism as our model selection. A goodness of fit index, R^2, and intuitively, the attempt of keeping a regression tree in each neuron are combined as our stopping criteria for the self growing process.

To conclude, our proposed Growing Self-Organizing Kernel Smoother for data mining has several merits. First, aiming to the regression goals, it is highly quantitatively precise and interpretive. Examples in Chapter 4 could testify this points. More importantly, we successfully keep the original advantages intact when hybridizing these two. For example, CART is popular not only because its structure is illustrative but also CART is a non-parametric methods without the difficulty of choosing suitable parameters such as NN. After hybridization, the model remains non-parametric and no new tough decisions have to make on model selection.

However, we have found two drawbacks during the research and development of our proposed Growing RTSOM. For one thing, also the hardest problem encountered, the growing process is based on heuristics and it does not guarantee to give the reproducible model selection. In the case study we have presented in Chapter 4, 9×9 map size was a repeated result after many trials. However, occasionally, different decisions are made such as 8×9 or 9×10 maps. They also provide similar good predicting results, though. For another, our proposed model can only predict static real number values. It is not built for forecasting time series data such as their trends.

6.2 Future Work

Based on the conclusion, this work could be expanded to serve some interesting open research areas.

In order to provide a more sophisticated framework for self-growing, Reinforcement Learning (RL) could serve an intuitive mechanism because it is most suited to problems where an optimal input-output mapping is unavailable a priori, but where a method for evaluating any given input-output pair is available instead [53]. As long as a reward function and state-action mapping could be defined reasonably, more legitimate constrains could be integrated into our growing algorithm to get reliable results.

As a result of adopting reinforcement learning, our hybrid model could also be applied to many other potential research directions such as robotics control and subchannel allocation in multicell OFDMA networks [54]. For example, similar dual-layer SOM models have been proposed in [55] to deal with continuous *state/action* space in the reinforcement learning (RL) problem.

Bibliography

[1] S. J. Russell, P. Norvig, and E. Davis, *Artificial intelligence: a modern approach*. Upper Saddle River, NJ: Prentice Hall, 3 ed., 2010.

[2] F. Chen, P. Deng, J. Wan, D. Zhang, A. V. Vasilakos, and X. Rong, "Data mining for the internet of things: Literature review and challenges," *International Journal of Distributed Sensor Networks*, vol. 11, no. 8, 2015.

[3] R. A. Daziano, "Conditional-logit bayes estimators for consumer valuation of electric vehicle driving range," *Resource and Energy Economics*, vol. 35, pp. 429–450, 2013.

[4] KredEx, "Estonia becomes the first in the world to open a nationwide electric vehicle fast-charging network," *Estonian World*, Dec. 2013.

[5] J. Gomez-Ibane, P. H. Bennett, D. J. Haigh, A. Wiederer, and R. Philip., "Policy options for electric vehicle charging infrastructure in c40 cities," *Clinton Climate Initiative*, July 2010.

[6] B. Place, "How we give the electric car unlimited range," *Better Place*, Apr. 2014.

[7] M. Roland and E. Ulrich, "The voltec system energy storage and electric propulsion," *Better Place*, May 2014.

[8] J. Axsen, S. Goldberg, J. Bailey, G. Kamiya, B. Langman, J. Cairns, M. Wolinetz, and A. Miele, "Electrifying vehicles: Insights from the canadian plug-in electric vehicle study," report, Simon Fraser University, Vancouver, Canada., May 2015.

[9] M. Ceraolo and G. Pede, “Techniques for estimating the residual range of an electric vehicle,” *IEEE Trans. Vehicular Tech.*, vol. 50, pp. 109–115, May 2001.

[10] C. Unterrieder, R. Priewasser, S. Marsili, and M. Huemer, “Battery state estimation using mixed kalman/hinfinity, adaptive luenberger and sliding mode observer,” pp. 71–76, 2013.

[11] T. Wang, M. Yang, K. Shyu, and C. Lai, “Design fuzzy soc estimation for sealed lead-acid batteries of electric vehicles in reflextm,” *IEEE International Symposium on Industrial Electronics*, pp. 95–99, 2007.

[12] M. Cheng, Y. Lee, M. Liu, and C. Sun, “State-of-charge estimation with aging effect and correction for lithium-ion battery,” *Electrical Systems in Transportation*, vol. 5, pp. 70–76, June 2015.

[13] B. Sun and L. Wang, “The soc estimation of nimh battery pack for hev based on bp neural network,” *Intelligent Systems and Applications (ISA 2009)*, pp. 1–4, May 2009.

[14] H. Yu, F. Tseng, and R. McGee, “Driving pattern identification for ev range estimation,” *Electric Vehicle Conference (IEVC)*, pp. 1–7, May 2012.

[15] H. He, C. Sun, and X. Zhang, “A method for identification of driving patterns in hybrid electric vehicles based on a lvq neural network,” *Energies*, vol. 5, pp. 3363–3380, May 2012.

[16] J. G. Hayes, R. P. R. de Oliveira, S. Vaughan, , and M. G. Egan, “Simplified electric vehicle power train models and range estimation,” *Vehicle Power and Propulsion Conference (VPPC)*, pp. 1–5, 2011.

[17] Y. Kondo, H. Kato, R. Ando, and T. Suzuki, “To what extent can speed management alleviate the range anxiety of ev?,” *Electric Vehicle Symposium and Exhibition (EVS27)*, pp. 1–8, 2013.

[18] S. Heath, P. Sant, and B. Allen, “Do you feel lucky? why current range estimation methods are holding back ev adoption.,” *Hybrid and Electric Vehicles Conference 2013 (HEVC 2013)*, pp. 1–6, 2013.

[19] Y. Zhang, W. Wang, Y. Kobayashi, and K. Shirai, "Remaining driving range estimation of electric vehicle," *in Proc. IEEE Int. Electr. Vehicle Conf. (IEVC)*, Mar. 2012.

[20] A. Bolovinou, I. Bakas, A. Amditis, F. Mastrandrea, and W. Vinciotti, "Online prediction of an electric vehicle remaining range based on regression analysis," *in Electric Vehicle Conference (IEVC)*, 2014.

[21] H. Rahimi-Eichi and M.-Y. Chow, "Big-data framework for electric vehicle range estimation," *in Proc. 40th Annu. Conf. IEEE Ind. Electron. Soc. (IECON)*, pp. 5628–5634, 2014.

[22] C. H. Lee and C. H. Wu, "A novel big data modeling method for improving driving range estimation of evs," *IEEE Access*, vol. 3, pp. 1980–1993, 2015.

[23] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning*. Cambridge University Press, 2014. Cambridge Books Online.

[24] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Verlag, New York, 2011.

[25] T. Hastie, R. Tibshirani, and J. Friedman, *Neural Networks and Learning Machines (3rd Edition)*. Pearson, 2008.

[26] P. J. Phillips, P. J. Flynn, T. Scruggs, K. W. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min, and W. Worek, "Overview of the face recognition grand challenge," *Computer vision and pattern recognition*, pp. 947–954, 2005.

[27] C. Li, Y. Diao, H. Ma, and Y. Li, "A statistical pca method for face recognition," *Intelligent Information Technology Application*, pp. 376–380, 2008.

[28] A. Ultsch and D. Korus, "Integration of neural networks with knowledge-based systems," *Neural Networks, 1995. Proceedings., IEEE International Conference*, vol. 4, pp. 1828–1833, 1995.

[29] J. V. Tu, “Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes,” *Journal of Clinical Epidemiology*, vol. 49(11), pp. 1225–1231, 1996.

[30] T. Kohonen, *Self-Organizing Maps.* Springer, 3rd edition, 2001.

[31] T. Hecht, M. Lefort, and A. Gepperth, “Using self-organizing maps for regression: the importance of the output function,” *European Symposium on Artificial Neural Networks (ESANN)*, 2015.

[32] A. Rauber, D. Merkl, and M. Dittenbach, “The growing hierarchical self-organizing map: Exploratory analysis of high-dimensional data,” *IEEE Transactions on Neural Networks*, vol. 13, pp. 1331–1341, 2002.

[33] P. Andras, “Kernel-kohonen networks,” *International Journal of Neural Systems*, vol. 12, pp. 117–135, 2002.

[34] D. MacDonald and C. Fyfe, “The kernel self organising map,” *In Proc. 4th int. conf. on knowledge-based intelligence engineering systems and applied technologies*, pp. 317–320, 2000.

[35] Z. S. Pan, S. C. Chen, and D. Q. Zhang, “A kernel-base som classifier in input space,” *Acta Electronica Sinica*, vol. 32, pp. 227–231, 2004.

[36] F. Mulier and V. Cherkassky, “Self organization as a kernel smoothing process,” *Neural Computation*, vol. 7, pp. 1165–1177, 1995.

[37] T. Kohonen, “Essentials of the self-organizing map,” *Neural Networks*, vol. 37, pp. 52–65, 2013.

[38] H. Bock, *Clustering and Self-Organizing Networks: Regression-Type Models and Optimization Algorithms*, pp. 39–48. Heidelberg: Physica-Verlag HD, 1999.

[39] D. Alahakoon, S. K. Halgamuge, and B. Srinivasan, "Dynamic self-organizing maps with controlled growth for knowledge discovery," *IEEE Transactions on Neural Networks*, vol. 11, pp. 601–614, May 2000.

[40] J. Lötsch and A. Ultsch, *Exploiting the Structures of the U-Matrix*, pp. 249–257. Cham: Springer International Publishing, 2014.

[41] C. Loader, *Local Regression and Likelihood*. Springer-Verlag New York, 1999.

[42] J. O. Rawlings, S. G. Pantula, and D. A. Dickey, *Applied Regression Analysis: A Research Tool*. Springer, 1998.

[43] B. Zheng, P. He, L. Zhao, and H. Li, "A hybrid machine learning model for range estimation of electric vehicles," *In Proc. GlobeComm 2016*, 2016.

[44] A. Tsanas and A. Xifara, "UCI machine learning repository," 2012.

[45] C. Lab, "Charge car project."

[46] B. Fulcher and N. Jones, "Highly comparative feature-based time series classification," *IEEE Trans. Knowl. Data Eng.*, vol. 26, pp. 3026–3037, 2014.

[47] A. Nanopoulos, R. Alcock, and Y. Manolopoulos, "Feature-based classification of time-series data," *Information Processing and Technology*, p. 4961, 2001.

[48] X. Wang, A. Wirth, and L. Wang, "Structure-based statistical features and multivariate time-series clustering," *in Proc. IEEE Int. Conf. Data Mining*, pp. 351–360, 2007.

[49] H. Deng, G. Runger, E. Tuv, and M. Vladimir, "A time series forest for classification and feature extraction," *Inf. Sci.*, vol. 239, pp. 142–153, 2013.

[50] S. O. Haykin, *Applied Regression Analysis*. Wiley, 3rd edition, 1998.

[51] E. Bair, T. Hastie, D. Paul, and R. Tibshirani, "Prediction by supervised principal components," *American Statistical Association*, vol. 101, pp. 119–137, 2006.

[52] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, pp. 175–185, 1992.

[53] A. J. Smith, "Applications of the self-organising map to reinforcement learning," *Neural Networks*, vol. 15, no. 8-9, pp. 1107–1124, 2002.

[54] J. R. Millán, D. Posenato, and E. Dedieu, "Continuous-action q-learning," *Machine Learning*, vol. 49, no. 2, pp. 247–265, 2002.

[55] H. Montazeri, S. Moradi, and R. Safabakhsh, "Continuous state/action reinforcement learning: A growing self-organizing map approach," *Neurocomputing*, vol. 74, no. 7, pp. 1069 – 1082, 2011.