UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y EN SISTEMAS
INTELIGENCIA ARTIFICIAL

DETERMINATION OF THE STRUCTURE OF POLYNOMIAL MULTIVARIATE APPROXIMATORS USING MACHINE LEARNING

TESIS
QUE PARA OPTAR POR EL GRADO DE
MAESTRO EN EN CIENCIAS E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
SANTIAGO DE JESÚS GONZÁLEZ MEDELLÍN

DIRECTOR DE TESIS:
DR. ÁNGEL FERNANDO KURI MORALES
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

CIUDAD DE MÉXICO, ENERO 2019

Contents

Chapter 1

Introduction

In the machine learning area there are techniques used to fit a mathematical model, not necessarily closed, to a set of data points. This is of particular interest in fields where data is collected experimentally. In this data two kinds of variables may be identified: target and predictor variables, where the target variables are explained and affected by the behavior of the rest of the predictor variables. The models studied in maching learning represent the relationship of a target variable as a function of the predictor variables. These models are calculated through a process known as *learning*. In this process the coefficients of the model are calculated in order to produce the desired output.

There are two types of learning: supervised and unsupervised. The supervised learning models are those in which you have labeled data and want to express patterns within the variables which lead to the labels. In unsupervised learning the data is not labeled and you try to classify it into clusters reflecting similarities of the elements of the clusters. Among these models we find those of the artificial neural networks (ANN). The artificial neural network models are inspired in the understanding of the structure and function of the biological neural networks. These models are proposed to extract relevant features from the input data and perform a pattern recognition task by learning from examples without explicitly stating the rules for performing the task[1].

Among these we can find, for example, multilayer-perceptron networks (MLPN)[2], radial basis function networks[3], support vector machines[4] and the Polynet[5]. When implementing these models one can realize that the main concern is that of the parameters' configuration and the model selection. However, there are some techniques for selecting the parameters to generate the model, such as cross validation[6], grid search, or Akaike Information Criterion[7], for large volumes of data they may require too much computation time. In the MLPNs the number of hidden layers, the number of neurons in each layer, the activation functions selected and the initial weights values affect in a significant way the performance of the learned model. In spite of being universal approximators[8] MLPNs require methods to estimate a suitable set of parameters. However, some advances have been made to estimate the said parameters. In [9], a lower bound to estimate the number of neurons in the hidden layer is found. A dynamic node creation algorithm is proposed in [10]. On the other hand there is the black-box problem in the MLPNs. This means that, in spite of learning high

degree and dimension relations between the variables, MLPNs do not make explicit how they are related to each other. In other words, the neural networks can model the result given an input vector of data but the user can not justify the outcome.

An alternative model inspired on neural networks is given in [11]. This model is a multivariate non linear polynomial in which the relation of each variable in the model is made explicit, making it possible to know the inferring process that was made to produce the outcome. The methodology to generate the polynomial model combines a genetic algorithm with a regression algorithm to find the best set of coefficients for a given configuration of variables in each monomial term within the polynomial to adequately approximate a dataset. There are parameters related to each algorithm mentioned before and must be tuned to find the best approximator. The parameters of the genetic algorithm are the probabilities of crossover and mutation processes, the number of individuals and the generations the algorithm must iterate. On the other hand, since the regression algorithm is iterative, it requires of a convergence criterion to stop. It is also needed to specify the parameters for the structure of the polynomial model such as the number of terms and the maximum degree of the polynomial. These algorithms and the full methodology are explained later in this document. In figure 1.1 a general sketch of the methodology to generate a multivariate polynomial model is shown. It can be seen that the structure of the model must be specified with some parameters as the number of terms and the degree of the polynomial.

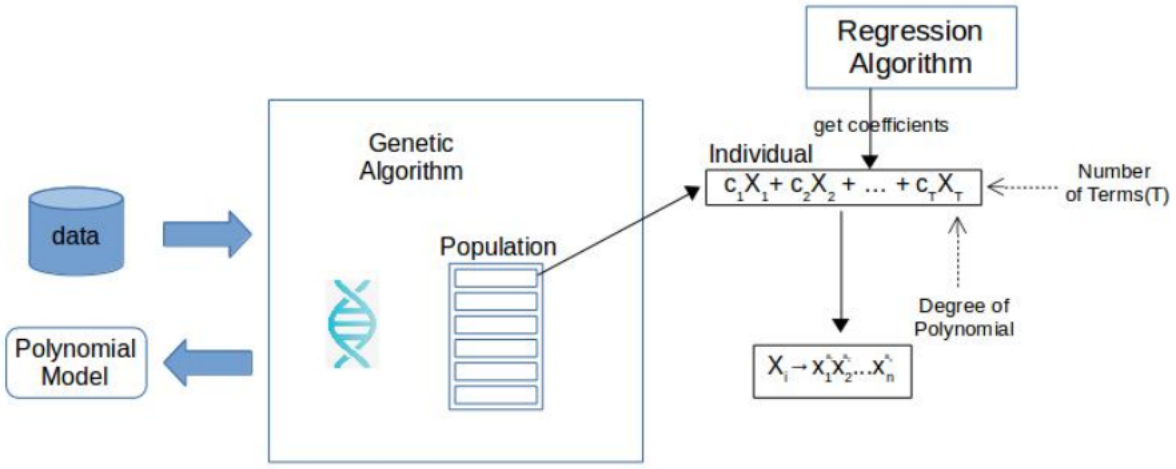

Figure 1.1: General solution diagram.

One of the parameters which can be hard to adjust is the number of terms, since the genetic algorithm iterates through a large number of individuals and for each of them many matrix and vector operations must be done. The time it takes to evaluate a dataset is critical, so it is not advisable to fine tune the parameters testing each value in a given range. In the current work we search for the bounds on the number of terms in the polynomial model. The search space of this parameter may grow exponentially depending on the chosen degree of the polynomial and the number of variables in the data. Let us assume, for example, five independent variables in the data and a max degree of five. The possible number of terms in

a full multivariate polynomial is

$$T = \sum_{a_1=0}^{5} \sum_{a_2=0}^{5} \sum_{a_3=0}^{5} \sum_{a_4=0}^{5} \sum_{a_5=0}^{5} x_1^{a_1} x_2^{a_2} x_3^{a_3} x_4^{a_4} x_5^{a_5} = 6^5 \tag{1.1}$$

which yields a total of 7,776 terms, which is a very large number of terms. We limit this space to a lower and upper bound which statistically ensures that the genetic algorithm will (with a high probability) find a proper approximator for the given problem.

1.1 Justification

The method to generate a multivariate polynomial model has been proven to have a good performance[11]. However, as in other machine learning methods, there is the problem of tunning the parameters to find the best model to fit a dataset. In the case of this method the number of terms is a parameter which is hard to tune for its large space of search. There exists other ways of finding an adequate set of parameters, like cross-validation. Although, in this case, depending on the number of variables of the training dataset and the number of terms this method may take a large amount of computational time to generate the polynomial model, making the tunning process overwhelming. This is why having a method to simplify this search would be advantageous in the implementation of this method.

1.2 Hypothesis

From the tunning process arises the following question: Is there a way to determine an appropriate value for the number of terms in the multivariate polynomial model generated by the studied methodology? There are 2 main hypotheses that we can state: 1) there is a way to obtain an adequate value for the number of terms such that the generated model yields the lowest error measure within a range of values. 2) It is possible to obtain this value before the training process from the characteristics of the dataset of the problem to be solved. To validate them a

1.3 Objectives

The main objective of this work is to find a method to estimate an adequate number of terms of a multivariate polynomial model generated by a methodology which combines a genetic algorithm and a regression method. To achieve this objective the following goals must be achieved.

- Analyze and implement the methodology which generates a multivariate polynomial model.

- Evaluate the methodology variating the number of terms in each case.
- Analyze the results and propose a method to calculate, before the training step, an proper number of terms for the multivariate polynomial model.
- Evaluate the proposed method.
- Apply the method in study cases.

This document is presented in the following order: in Chapter 2 some of the works in the state of the art in multivariate polynomial models are discussed. In Chapter 3 the methodology which generates a multivariate polynomial model and its two main components is described. In Chapter 4 the methodology to analyze the problem is given. In Chapter 5 it is described how a bound is found for the number of terms in the polynomial model. And finally in Chapter 6 some conclusions about the results found are given.

Chapter 2

State of the art

The study of predictive models has been of interest in areas such as mining [12], environment [13], biology[14], military aeronautics [15], etc. The importance of these models lies in how well they can express the relations between the observed variables in each particular problem. The models which approximate a function to a dataset are called regressive models. The polynomial model is commonly used to solve problems for their simplicity of structure. This model has brought a lot of attention because of the power they have to describe complex nonlinear input-output relationships in an effective way. It is tractable (i.e. it can be solved in polynomial execution time) for optimization, sensitivity analysis, and prediction of confidence intervals[16]. Besides it has been proven (in the Stone-Weiestrass theorem) that any function can be uniformly approximated by a polynomial[8]. Consequently, multivariate polynomial regression provides us with an effective way to describe complex nonlinear input-output relationships[17]. A single variable polynomial function with degree k has the form $y = \sum_{k=0}^{n} a_k x^k$ where x is the independent variable and y the dependent variable. Generalizing to a d-dimensional space a multivariate polynomial is defined as follows[18].

Let F be a field and $n \in \mathbb{N}$.

- Each *exponent vector* $e = (e_1, e_2, ..., e_n) \in \mathbb{N}^n$ defines a monomial in $F[x_1, x_2, ..., x_n]$: $x^e = x_1^{e_1}, x_2^{e_2}, ..., x_n^{e_n}$
- A *term* in $F[x_1, x_2, ..., x_n]$ is the product of a non-zero coefficient $c \in F$, i.e. $c \cdot x^e$.
- A *polynomial* $f \in F[x_1, x_2, ..., x_n]]$ is a finite set of terms. It is written $\sharp f$ for the number of terms in f.
- The maximum *degree* of a polynomial is the highest exponent: maxdeg $x^e :=\| e \|_\infty = maxdeg\ _{1 \leq i \leq n}(e_i)$

In the literature there are linear and non-linear polynomial regression models. The simple linear regression models are defined as the linear relationship between a continuous response variable Y and a set of explanatory variables X [19]. On the other hand, the multiple linear regression models express the lineal combination between the explanatory variables, an error variable and the dependent variable[20]. The traditional regression models usually fail to

separate the corresponding effects given the complex structure in the correlation between the variables[21]. In the univariate case almost all problems are well established and solved. In the multivariate case there remain issues to be analyzed[22]. In spite of being simple there are some drawbacks in the polynomial models. Most of the polynomial methods have[5]: 1) non-optimal term generation and storage, 2) generalization error, 3) network complexity (in the case of network structural polynomial systems), 4) high complexity in training, 5) low accuracy.

Current machine learning models are based on linear algebra and probability theory. Some of them search linear relations as in Linear Regression, or Naive Bayes. Unfortunately, in real world problems it is common to have non-linear relations. Various applications of regression models have been explored and in some cases the models can only express linear relationships between variables. For instance in [23] a comparison of simple and multivariate models and logistic regression models is made. In [12] static hand-made models proposed before are studied and compared against MLPNs and reached the conclusion that MLPNs have better accuracy since these models have the capacity to express high order relationships. In [24] a probabilistic learning algorithm is presented based on incremental mixture of Gaussians implementing the maximum expectation algorithm[25]. In [13] an ensemble of MLPNs with other regression models is proposed to measure the air quality. Cortina proposes a multiple regression model adequate to fit the necessities of modeling social interactions[26]. He uses the Moderated Hierarchical Multiple Regression (MHMR) to test the presence of interactions. The model deals with at most 3 variables, with degrees of order ≥ 1. In other cases, the models capable of expressing higher order relationships are usually hand-made to be applied in very specific contexts [27, 12, 15, 28].

There have been other attempts to fit a multivariate polynomial model into a dataset. One of the early attempts to achieve polynomial models similar to NN was the polynomial neural networks which where an implementation of GMDH[29] with the Kolmogorov-Gabor polynomials as basis function. This method offers a very complex polynomial model since the number of terms can grow exponentially. In GMDH the idea is to create a single node function (in the original case a two-input second-order polynomial) which, when cascaded in a multilayer architecture, produces permutations of monomial product terms which comprise an estimation of higher order terms necessary to approximate various functions. However GMDH has some issues: 1) training cannot be straight forward, 2) computes excessive terms which are not necessary in the model, 3) may be over-fitted problems, 4) they are susceptible to local minimum problems.

Ivakhnenko models input-output relationship of a complex system using a multi-layered perceptron type network[30], each layer implements a non-linear function of its inputs. This function is usually a second-order polynomial of the inputs. The function implemented by each element in one of the layers is the following

$$y = A_2(X) = a_0 + a_1x_1 + a_2x_2 + a_3x_1^2 + a_4x_2^2 + a_5x_1x_2 \tag{2.1}$$

There must be 3 or 4 layers in the architecture so the maximum degree relationships between the variables learned by the model are of 6th degree. The coefficients of each element in the network are determined the following way: consider N input vectors of p

variables $X_n = (x_{n1}, ..., x_{np})$ and the nth output as ϕ_n. Then a set of six coefficients for this element (which has inputs x_{ni} and x_{nj}) must be found such that the mean-squared error between the outputs of this element y_n, and the true output ϕ_n is minimized. The coefficients are obtained from the "Gauss normal equations". In matrix form $\phi = XA$ where matrices ϕ,X and A are of order $N \times 1$,$N \times 6$ and 6×1, (the first element in each row of the X matrix is 1), the normal equations are formed by premultiplying both sides by transpose of X, $X^T = (X^T X)A$. Matrix $X^T X$ is a 6x6 matrix and the solution is found by inverting the matrix $A = (X^T X)^{-1} X^T \phi$. Locally weighted polynomial regression is a popular algorithm for learning continuous non-linear mappings[31]. The regression is calculated according to a point x_{query}. A linear map (in the case of linear regression) is constructed and is influenced by data points that lie close to the query point according to some scaled euclidean distance metric, where, for example, a very close data point to the query point has a weight of 1 and a very far away point a weight of zero. The polynomial of M terms that are being minimized by means of mean-squared error, is of the form:

$$\hat{y}(x) = \beta_1 t_1(x) + \beta_2 t_2(x) + \cdots + \beta_M t_M(x) \tag{2.2}$$

which can be written as $\hat{y}(x) = \beta^T t(x)$ where t_j is a function that produces the *j-th* term in the polynomial

$$\sum_{i=1}^{N} w_k^2 (y_k - \beta^T t_k)^2 \tag{2.3}$$

This does not need gradient descend. It can be inferred from $\beta = (X^T X)^{-1} X^T y$. More succinctly the authors write $X^T X = \sum_{k=1}^{N} w_k^2 t_k t_k^T, X^T y = \sum_{k=1}^{N} w_k^2 y_k t_k$ where N is the number of points to train. Although the model is powerful for fitting multivariate noisy data with non linearities, it requires to scan each point in the dataset. All data points have their weighted contribution added into the $X^T X$ and $X^T y$ matrices. Four ways of preventing to scan all the points are given by[31]: 1) Picking a smaller subset, resulting in detail loss. 2) Build a multivariate spline model. Unfortunately, continuous interpolation above 2 dimensions is too expensive. 3) Range searching with KD-trees: given a query data point the nearest data points within a distance given are returned without searching the entire dataset, which may result too expensive for high dimensions. 4) A modified KD-tree structure to select the subset of more spread points. For high-dimensional and high-order systems, multivariate polynomial regression becomes impractical due to its huge amount of product terms [16]. Nevertheless there have been efforts to overcome this issues and make the polynomial models viable for regression and classification tasks.

In [16] we can observe also a polynomial model which is aimed to be reduced with a series of linear transformations and convert the full polynomial to a few terms. This model is suitable for small features and large number of examples. In this work the authors present 2 ways of working with high dimensionality problems: 1) By means of compact universal basis functions like perceptron and radial basis functions. Another example are Fourier series and wavelets. 2) Dimension reduction techniques, which compromises the approximation capability. The reduced multivariate polynomial model proposed in their work is the following.

$$\hat{f}_{MN}(\alpha, x) = \alpha_0 + \sum_j \alpha_j(x_1^{n_1}, x_2^{n_2}, ..., x_l^{n_l}), j = 1, 2, ..., K \tag{2.4}$$

The full model is formed by $K = 1 + r(l + 1)$ terms where r is the sum of the degrees of the polynomial and l is the number of independent variables. The reduced model has $K = 1+r+l(2r-1)$ terms. The model needs to calculate a gradient from a set of weights, and the matrix operations are made over the full training set. To avoid multilinearity problem they stabilize the weights (coefficients of the polynomial) with a regularization term bL_2 (where b is a regularization constant) which can be added to the least squares minimization by $\alpha = (P^T P + bI)^{-1} P^T y$,also known as ridge regression where I is the identity matrix. The reduced model works with another representation of the full multivariate polynomial model where there are weights in each of variables of each of the monomials.

Shaposhnyk et. al.[32] propose improvements to algorithms made for polynomial model selection (parameter estimation) on multivariate datasets with outliers in response and explanatory variables. These methods were tested in polynomials up to 3rd degree on synthetic functions. A polynomial neural network is trained by calculating weights in each neuron of each layer with least squares regression, they also propose another way (with robust statistics, i.e. without assuming normality on the data) to train this type of artificial neural networks.

In the work of [33] an evolutionary algorithm is used to determine the structure of polynomial fuzzy models. The model implements a multivariate fuzzy function. This fuzzy functions are obtained from a predefined library of multivariate fuzzy polynomials. They test the proposed procedure with 2 independent variables polynomials to 2nd degree (shown to give the best results).

In [17] the authors investigate the frequently encountered under-determined system with ANN estimation formulation based on ridge regression. This model works with the whole expansion of terms (up to 12650 terms). The way they estimate the parameter is similar to least squares.

Lastly in [5] a new algorithm is introduced for implementing a single layer polynomial network that can be rapidly generated and trained to approximate multidimensional nonlinear and continuous-output functions of significant complexity. This algorithm generates only non repeated monomial terms in a single-layer architecture which is easily trained with simple ordinary least squares algorithm. This method still uses the full range in the polynomial, making it not quite visually manageable.

In [34] a methodology is proposed to train a multivariate polynomial model to fit a dataset in which, instead of generating the full range of terms, generates the combination of a preselected number of terms. This is accomplished by means of the ensemble of a breed of a genetic algorithm and a regression algorithm, in which the individuals of the genetic algorithm represent a set of monomials with different combination of variables and their exponents and the regression algorithm computes the coefficients of the terms in such way that the minimax error function (or L_{inf}) is minimized. The current work is based on this methodology. It is well known that the polynomial models may have an exponential number of terms as the input variables and the order of the polynomial grows[17], so it is

not advisable to have more than, let us say, 20 terms. For this reason, in this methodology, an adequate number of terms must be set in order to generate a model which approximates properly the problem. It can represent in a simpler way (with fewer terms) the relationships of higher orders between explanatory variables. As the number of terms grow, and depending on the volume of the training data, evaluating this methodology to find the best tune in the parameters may become an overwhelming task. It is of great interest to find a boundary which reduces the search space for this parameter. Hence there is a way to determine a proper value which brings us closer to the number of terms necessary for best fit. In the current work we search for these boundaries and propose a tool to estimate this value.

Chapter 3

Genetic Multivariate Polynomial Approximation

As mentioned in the previous chapter, polynomial models are important because of their power to express functional relationships between explanatory variables in a simple structural way. However, the number of terms that make up the model may be very high. This is why the model of [11] deals with this issue allowing the user of this methodology to establish a desired number of terms with which the model is going to train. This methodology raises from the Universal Approximation Theorem (UAT) which states the following[8].

Let $\phi(\cdot)$ be a non constant, bounded, and monotonically-increasing continuous function. Let I_{mO} denote the m_O-dimensional unit hypercube $[0,1]$. The space of continuous functions on I_{mO} is denoted by $C(I_{mO})$. Then, given any function $f \in C(I_{mO})$ and $\varepsilon > 0$, there exist an integer M and sets of real constants a_i , b_i and w_{ij}, where $i = 1, 2, \cdots, m_I$ and $j = 1, 2, ..., m_O$ such that we may define:

$$F(x_1, \cdots, x_{m_O}) = \sum_{i=1}^{m_I} \left[a_i \cdot \phi \left(\sum_{j=1}^{m_O} w_{ij} x_j + b_i \right) \right] \tag{3.1}$$

as an approximate realization of a function $f()$, that is,

$$|F(x_1, \cdots, x_{m_O}) - f(x_1, \cdots, x_{m_O})| < \varepsilon \tag{3.2}$$

for all the $x_1, \cdots, x_{m_O}$ in the input space.

This theorem is directly applicable to the multilayer perceptron networks by adding the bias value to the theorem.

$$F(x_1, \cdots, x_{m_O}) = a_0 + \sum_{i=1}^{m_I} \left[a_i \cdot \phi \left(\sum_{j=1}^{m_O} w_{ij} x_j + b_i \right) \right] \tag{3.3}$$

From the UAT we know that a MLPN only needs one hidden layer to accurately approx-

imate a function. Also it is mentioned that the function to be approximated needs to be continuous. This is, in practice, impossible due to the limitations of a computer to represent very small numbers. As continuity is a property that must be satisfied the data may be augmented so there are more points which the algorithm can process, getting closer to a "more continuous" space. Further in this work the topic of augmenting the data to comply this property is treated.

In this methodology $\phi(x) = \text{logistic}(x)$ is selected as the nonlinearity function of the MLPN. It is known[35] that the logistic function may be approximated by a polynomial $P_n(x)$ of degree n with an error $\varepsilon = |P_n(x) - \text{logistic}(x)|$ where $\varepsilon \to 0$ as $n \to \infty$. It is known[36] that a function $y = f(x)$ in $[0, 1)$ may be approximated by a polynomial with Chebyshev orthogonal basis minimizing the least squared error and achieving the minimum maximum absolute value. In [11] it is demonstrated that any function as in equation 3.3 may be approximated with a linear combination of polynomials having only terms of odd degree. Also it is shown that only 20 power combinations for the expansion of the logistic are possible. These combinations are shown in table 3.1

1	11	33	63
3	15	35	77
5	21	45	81
7	25	49	99
9	27	55	121

Table 3.1: Combination of odd powers for the multivariate polynomial model

To accomplish the task of looking through the possible combinations of powers in the variables two main components of this machine learning algorithm work together: the Eclectic Genetic Algorithm (EGA)[34] and the Ascent algorithm[37] which are going to be explained in detail in the following sections.

3.1 Eclectic Genetic Algorithm

Genetic algorithms are a breed of the family of the evolutionary algorithms. The evolutionary algorithms, in general, are inspired in the biological process of evolution in which a population of certain species of individuals is tested in the environment where only the best adapted ones are selected to breed a new generation of individuals which can, with a given probability, be slightly different of their ancestors by means of a mutation process. The new individuals are going to be tested as well. This process is repeated over various generations in order to find the individuals that are best adapted to the provided environment.

The individuals in the population of the evolutionary algorithms are solutions to a specific problem (which may be seen as the environment) codified into a numeric representation. Each individual is evaluated against the problem obtaining a fitness value which indicates how well one single individual is solving the problem. Once all the individuals were tested they are sorted and selected based in a non deterministic way. Then, the selected individuals go

through a process where sections of them are combined to generate new individuals which may have the properties of their parents. This process is called crossover. After the crossover process the new individuals are randomly (with a low probability) altered. This is based in the natural mutation process in which some individuals may come out with new characteristics which previous individuals did not have and improve in the given problem. The algorithm runs until some convergence criterion is achieved, e.g. it may be a maximum number of generations or a threshold fitness value.

Genetic algorithms are one of the most frequently encountered type of evolutionary algorithms. The first genetic algorithm was proposed by Holland [38]. This GA has the following characteristics: the individuals are encoded in binary fixed size strings, the individuals for mating are selected by Roulette Wheel Selection in which each individual has a probability of being selected proportional to their fitness value. The crossover is assumed to be single-point crossover, in which two parent individuals are selected from the mating pool and, with crossing probability p_c, a point is randomly chosen and the strings are swapped with respect the crossover point between the two parents. The mutation operation is executed bit-wise with probability p_m. When it occurs a 0 is changed for a 1 and vice versa. One problem arises from this first approach made by Holland. When the algorithm finds an optimal value the probability of losing the individual in further generations is 1. To overcome this issue the n best individuals are kept so the optimal solutions are never dropped. This is called elitism. With this simple technique the GA convergence to an optimal value is guaranteed[39].

The genetic multivariate polynomial methodology implements the Eclectic Genetic Algorithm (EGA)[40] which was demonstrated to have better performance than other regular types of GA[34]. This algorithm is used to minimize the error in the polynomial model. The EGA incorporates the following characteristics:

1. **Full Elitism**. The best n individuals are kept in a population of n individuals.
2. **Deterministic Selection**. The best individual is crossed with the worst. In general the *i-th* individual is crossed with the *(n-i+1)-th* individual to ensure variety in the next generation's breed.
3. **Annular Crossover**. When mating, the individuals are treated as rings instead of vectors where the leftmost bit is contiguous to the rightmost bit.
4. **Uniform Mutation**. Each bit of each individual is swapped from 1 to 0 and vice versa with a probability p_m.

The individuals of the EGA are encoded to represent polynomial structures. As it is known, the full multivariate polynomial models have an exponential number of terms depending on the order of the model and the number of variables. The number of unique non-repeated terms in a complete polynomial of a particular order is given by number of terms = $\frac{(o+k)!}{o!k!}$ where o is the order of the polynomial and k is the number of inputs.

To explore such huge search space the EGA is an appropriate technique. The polynomial structures that each individual represents are of the following form:

$$y = c_1 X_1 + c_2 X_2 + \cdots + c_m X_m \tag{3.4}$$

where X_i denotes a combination of independent variables and c_i is the coefficient correspondent to the *i-th* monomial. To calculate the coefficient of each monomial in the polynomial structure this methodology implements the Ascent Algorithm (AA).

3.2 Ascent Algorithm

The purpose of this algorithm is to express the behavior of a dependent variable y as a function of a set of n independent variables v, thus

$$\begin{aligned} y &= f(v_1, v_2, \cdots, v_n) \\ y &= f(v) \end{aligned} \tag{3.5}$$

The approximant has the form of equation 3.4. This method requires a sample of size N such that for every set of the independent variables v there is a known value of the dependent variable f. By convention N stands for the number of objects in the sample and $M = m + 1$ where m = number of desired terms of the approximant.

The goal of the Ascent Algorithm (AA) is to find the values of the coefficients in 3.4 such that the approximated values minimize the difference between the known values of the dependent variable f in the subset and those calculated from 3.4 for all the objects in the subset. The approximation error is defined as $\varepsilon_{\text{MAX}} = \max(\varepsilon_1, \cdots, \varepsilon_m)$ where $\varepsilon_i = \text{abs}(f_i - y_i)$. Here f_i stands for the value of the dependent variable of object i and y_i stands for the value which the approximant yields when the X_i are put into 3.4. The AA is based on a two-phase iterative methodology. First, a subset (of size M) is selected (this is called the inner set) and the best approximant (a set of coefficients) in the minimax sense is found. Second, the approximant is tested to see whether $y = f(X)$ satisfies the minimax norm for the remaining $N - M$ objects (this set of cardinality $N - M$ is called the outer set). That is, the y_i are calculated for the external set. If the minimax condition is attained by all the objects the algorithm ends: the coefficients are those of the best possible approximant. If at least one of the objects in the outer set does not comply with the minimax condition then an object of the inner set is swapped with an object in the outer set and the process is repeated. In every step (τ) of the process there are two errors of interest:

(a) The largest absolute approximation error of the internal set (denoted by $\varepsilon_\theta(\tau)$)

(b) The largest absolute approximation error of the external set (denoted by $\varepsilon_\phi(\tau)$)

$\varepsilon_\theta(t)$ is calculated during phase 1; $\varepsilon_\phi(t)$ is calculated during phase 2. The convergence condition is that $\varepsilon_\theta(t) \geq \varepsilon_\phi(t)$. It may be shown [11] that $\varepsilon_\theta(t+1) \geq \varepsilon_\theta(t)$ monotonically and that $\varepsilon_\phi(t+1) \leq \varepsilon_\phi(t)$ non-monotonically. Therefore, they approach each other and the convergence condition is always reached.

3.3 The Minimax Polynomial for the Internal Set

In what follows we use $|X|$ to denote $\det(X)$ as opposed to $\text{abs}(X)$. The proper interpretation of this notation will be apparent from the context. We know that $\varepsilon_\theta = \max(\varepsilon_1, \cdots, \varepsilon_M)$, also that $\varepsilon_i = \text{abs}(f_i - y_i)$ and $y_i = c_1X_{i1} + c_2X_{i2} + \cdots + c_mX_{im}$. Therefore, $\varepsilon_i = \text{abs}(f_i - (c_1X_{i1} + c_2X_{i2} + \cdots + c_mX_{im}))$,$i = 1, 2, \ldots, M$.

Without loss of generality we may take the positive value of ε_i and then $\varepsilon_i + c_1X_{i1} + c_2X_{i2} + \cdots + c_mX_{im} = f_i$. We can write

$$\begin{aligned} &\varepsilon_1 + c_1X_{11} + c_2X_{12} + \cdots + c_mX_{1m} = f_1 \\ &\varepsilon_2 + c_1X_{21} + c_2X_{22} + \cdots + c_mX_{2m} = f_2 \\ &\cdots \\ &\varepsilon_M + c_1X_{M1} + c_2X_{M2} + \cdots + c_mX_{Mm} = f_M \end{aligned} \tag{3.6}$$

. Making $\varepsilon_i = s_i\varepsilon_\theta$, where s_i are constants to be determined, we have

$$\begin{aligned} &s_1\varepsilon_\theta + c_1X_{11} + c_2X_{12} + \cdots + c_mX_{1m} = f_1 \\ &s_2\varepsilon_\theta + c_1X_{21} + c_2X_{22} + \cdots + c_mX_{2m} = f_2 \\ &\cdots \\ &s_m\varepsilon_\theta + c_1X_{M1} + c_2X_{M2} + \cdots + c_mX_{Mm} = f_M \end{aligned} \tag{3.7}$$

Applying Cramer's rule to system (3.7) we have

$$\varepsilon_\theta = \frac{\begin{vmatrix} f_1 & X_{11} & \cdots & X_{1m} \\ \cdots & \cdots & \cdots & \cdots \\ f_M & X_{M1} & \cdots & X_{Mm} \end{vmatrix}}{s_1\begin{vmatrix} X_{21} & \cdots & X_{2m} \\ \cdots & \cdots & \cdots \\ X_{M1} & \cdots & X_{Mm} \end{vmatrix} - \cdots + s_M\begin{vmatrix} X_{11} & \cdots & X_{1m} \\ \cdots & \cdots & \cdots \\ X_{m1} & \cdots & X_{mm} \end{vmatrix}} \tag{3.8}$$

Using X_{i*} to mean that the $i-th$ row and the first column of the determinants in the denominator of (3.8) have been deleted we put

$$\varepsilon_\theta = \frac{\Delta}{s_1|X_{1^*}| - s_2|X_{2^*}| + s_3|X_{3^*}| - \cdots} \tag{3.9}$$

or,

$$\varepsilon_\theta = \frac{\Delta}{s_1\Delta_{1^*} - s_2\Delta_{2^*} + s_3\Delta_{3^*} - \cdots} \tag{3.10}$$

Where Δ is the determinant of the numerator of (3.8) and $\Delta_{i*} = |X_{i*}|$. In more compact notation:

$$\varepsilon_\theta = \frac{\Delta}{\sum_{i=1}^{M} s_i(-1)^{i-1}\Delta_{i*}} \tag{3.11}$$

It can be seen that for ε_θ to be minimized the denominator of (3.11) has to be maximized. This implies that:

(a) All its products must be maximized, and

(b) All its addends must be of the same sign

This observation leads to an algorithm which allows us to solve (3.11) minimizing ε_θ and thus finding the coefficients of (3.4) in the minimax sense. Since $\varepsilon_i = s_i\varepsilon_\theta$ it follows that $s_i \leq 1$ and $s_\theta = 1$. Hence, for the $s_i\Delta_i$ to be maximized, we have to

(a) Take the largest values of the s_i

(b) Set the values of the s_i such that $s_i = \text{sgn}[(-1)^{i-1}\Delta_{i*}]$

The first condition is fulfilled iff $s_i = 1\ \forall i$. To fulfill the second condition we must determine the signs of the Δ_{i*}. The best way to do this is by resorting to the following cofactor property: "*if the cofactors of one column of a determinant are multiplied by the elements of a different column and summed, the result is zero*". To illustrate consider the next determinant:

$$A = \begin{vmatrix} \sigma_1 & X_1 & Y_1 \\ \sigma_2 & X_2 & Y_2 \\ \sigma_3 & X_3 & Y_3 \end{vmatrix} \tag{3.12}$$

Denoting the cofactors of column 1 with σ_i, from the property we know that

$$\begin{aligned} \sigma_1 X_1 + \sigma_2 X_2 + \sigma_3 X_3 = 0 \\ \sigma_1 Y_1 + \sigma_2 Y_2 + \sigma_3 Y_3 = 0 \end{aligned} \tag{3.13}$$

which is a system lacking one condition to be solved. However, we may assign the value of one of the determinants arbitrarily, since we only care for its sign and not its magnitude. Notice that its sign is irrelevant since we are minimizing the absolute value of the error. For simplicity, we may set σ_3, from which

$$\begin{aligned} \sigma_1 X_1 + \sigma_2 X_2 = X_3 \\ \sigma_1 Y_1 + \sigma_2 Y_2 = Y_3 \end{aligned} \tag{3.14}$$

We know that $\text{sgn}(\Delta_{i*}) = \text{sgn}(\sigma_i)$ and, since their absolute value is 1, we may easily solve

$$\begin{vmatrix} \sigma_1 & X_{11} & X_{12} & \cdots & X_{1m} \\ \sigma_2 & X_{21} & X_{22} & \cdots & X_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sigma_M & X_{M1} & X_{M2} & \cdots & X_{Mm} \end{vmatrix} \begin{vmatrix} \varepsilon_\theta \\ c_1 \\ c_2 \\ \vdots \\ c_m \end{vmatrix} = \begin{vmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{vmatrix} \tag{3.15}$$

We remarked earlier that X_i in (3.4) denotes a combination of the independent variables. A common choice is, as mentioned, some linear combination of the monomials of the independent variables. To illustrate, assume that our choice has been the following:

$$w(x,y) = \sum_{i=0}^{1}\sum_{j=0}^{2} c_{ij}x^i y^j \quad = c_{00} + c_{01}y + c_{02}y^2 + c_{10}x + c_{11}xy + c_{12}xy^2 \tag{3.16}$$

This determines that $X_1 = (1, y_1, y_1^2, x_1, x_1y_1, x_1y_1^2)$, ... , $X_N = (1, y_N, y_N^2, x_N, x_Ny_N, x_Ny_N^2)$. In this case $m = 6$, $M = 7$ and, from our experiments, $N = 150$. Before outlining the algorithm which will allow us to do so, there is still one issue to take into account.

3.4 Perturbation and Stability

It may be seen that it is easy to find (from the above procedure) data where a set of rows or columns may be linearly dependent. In such case the system of equations which will be formulated may become numerically unstable or simply have no solution. For instance, if $y_i \approx 1 \forall i$ the first three columns of (3.15) will be very nearly 1. Therefore, we replace the elements in the data matrix X thus:

$$X_i^* = \begin{cases} X_i \times (1 + \rho_u * \delta_H) & \text{if } X_i \neq 0 \\ \rho_u * \delta_H & \text{if } X_i = 0 \end{cases} \tag{3.17}$$

where ρ_u denotes a uniformly distributed random variable $0 \leq \rho_u < 1$ and $\delta_H = O(10^{-6})$. This tends to replace linearly dependent vectors by linearly independent ones. The approximation coefficients are very closely correct and no further modifications are needed. It may be shown [11] that the relative approximation error is of $O(\delta_H)$. More precisely:

$$\frac{|F_\tau^* - F_\tau|}{F_\tau} < \delta_H \sum_i d_i \tag{3.18}$$

where F_τ^* is the value of the function approximating the disturbed vectors in (the convergence) step τ of the algorithm, F_τ is the value of function approximating the original undisturbed vectors, δ_H is the size of the perturbation constant and d_i denotes the highest degree of the *i-th* variable.

The ascent algorithm works as follows[11]

Algorithm 1 Ascent Algorithm

1: Input data vectors (call them D)
2: Input the degrees of each of the variables of the approximating polynomial.
3: Map the original data vectors into the powers of the selected monomials (call them P).
4: Stabilize the vectors of P by randomly disturbing the original values as above (call the resulting data S).
5: Select a subset of size M from S. Call it I. Call the remaining vectors E.
6: Obtain the minimax signs (call the matrix incorporating the σ's A).
7: $B \leftarrow A^{-1}$
8: **loop**
9: Calculate the coefficients $C \leftarrow fB$. The maximum internal error ε_δ is also calculated.
10: Calculate the maximum external error ε_ϕ by evaluating C in E. Call its index I_E.
11: **if** $\varepsilon_\theta \geq \varepsilon_\phi$ **then**
12: Stop, the coefficients of C are those of the minimax polynomial for the D vectors.
13: **else**
14: Calculate λ vector $\lambda = A^{I_E} B$
15: Calculate β vector which maximizes $\sigma_{I_E \frac{\lambda_j}{B_j}}$. Call its index I_I.
16: Interchange vectors I_E and I_I.
17: Calculate the new inverse $\hat{B}$. Make $B \leftarrow \hat{B}$
18: **end if**
19: **end loop**

Chapter 4

Experimentation Methodology

To find out whether there are boundaries to search the value for the number of terms in the multivariate polynomial model a Multi Layer Perceptron Network approach was taken, since we know from [8] that they are universal approximators. For this we collected 46 datasets from the University of California Machine Learning dataset repository [41] and the Knowledge Extraction Evolutionary Learning dataset repository [42]. From the 46 datasets 32 were chosen to evaluate the multinomial genetic model to find the best fit RMS error varying the number of terms $t \in [3, 15]$. From the results a lower bound is statistically inferred. Some characteristics of the datasets are obtained (such as the number of attributes, the number of rows in the dataset, the total information, etc.) and the lower bound of the number of terms obtained before is put as function of these characteristics in order to train a MLPN. The rest of the datasets are used as recall data for the trained MLPN model. The results obtained show that there is a delimited range of search. As a result of the process to find these boundaries a tool to calculate the proper number of terms in the polynomial model was created. This is statistically proven and some case studies are made to test the elaborated tool.

4.1 Data collection

The data collected was aimed to be of heterogeneous nature. In the consulted repositories there is a large number of different datasets from different areas of knowledge: from specific industrial problems to classic ones which have been well studied (as the Iris setosa or the wine classification datasets). The objective of collecting a wide variation of data is to validate that the current methodology works in any of them, yielding a good performance.

4.1.1 Data Preprocessing

There are certain properties the data must satisfy for the model to work properly and ensure a good performance. It must be taken into account that most of the collected data is not cleaned (e.g. it has missing values), the data which the model works with is all numerical and

scaled into the range $[0, 1)$. Hence the datasets are preprocessed before training the model. The preprocessing phases in this methodology are (not necessarily in the following order) the following: 1. Feature selection, 2. Categorical encoding, 3. Fill missing values, 4. Scaling, 5. Stabilization, 6. Sub Sampling, 7. Data Augmentation.

Feature selection. In many datasets there are features which provide no relevant information to the learning algorithm. For example, an Id attribute whose values are all different and have no pattern in them. Another case could be a feature which has many missing values (i.e. $> 70\%$). It may be difficult to fill those missing values, so the field is removed. If that is the case, then the whole column is removed. It's important to note that more predictor variables do not make a better model, but identify the variables which may contribute in a better way to the learning process of the model (i.e. those with more variating values). Also a correlation analysis is made to check linear relations between variables. In the case there exists a high correlation index between two or more variables (i.e. $> 85\%$) one variable is chosen to be kept and the others are removed.

Categorical encoding. As most of the machine learning strategies, the polynomial model in this work deals only with numerical values. Because of this, there must be a way in which the values from the categorical variables are transformed into numbers. There are many ways to encode categorical variables, although is not clear yet which one is the best. This is due to the difficulty of preserving numerically the meaning of the instances of the categories. Some ways (such as assigning an integer number to each value) may induce an order which the original value did not have. For example a variable which represents a color (colors do not have order). Other ways such as one hot encoding may increase the dimensionality of the data set making it difficult to work with. In this methodology one-hot encoding is used in the case the categorical attribute has only two values (like sex) so the dimensionality does not increase. In the case of multiple values an alternative method called CESAMO (Categorical Encoding by Statistical Applied Modeling)[43] is applied. The aim of CESAMO is to find numerical codes such that the patterns in the original categorical variable are preserved. In this method, random numerical codes are assigned to each instance of a categorical variable. With the assigned codes the mean of a function f representing the relationship between the categorical variable x_i and $x_{j!=i}$ for $i = 1, \cdots, n$ where n is the number of variables. This process is made until the distribution of the means of the samples becomes Gaussian to achieve statistical stability (this means that further samples will not make a significant difference in the characterization of the population). For the current work the f =*Pearson's correlation*. The compliance of the Gaussian distribution is calculated using χ^2 goodness-of-fit test, establishing the maximum correlation as best coding criterion over 3600 means per categorical variable.

Fill missing values. In datasets there are often missing values caused by the way they were collected. To ensure the data is complete one may fill these values in ways which can make sense accordingly to the other values (the ones which are not missing). For this we use a technique called cubic spline interpolation. The idea is to generate a series of cubic

polynomials which fit between each of the data points in each feature column of the form[44]

$$S(x) = \begin{cases} s_1(x) & \text{if } x_1 \leq x < x_2 \\ s_2(x) & \text{if } x_2 \leq x < x_3 \\ \vdots & \\ s_{n-1}(x) & \text{if } x_{n-1} \leq x < x_n \end{cases} \tag{4.1}$$

where s_i is a third degree polynomial defined by

$$s_i = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i \tag{4.2}$$

Using natural splines the missing data is calculated and the values remain coherent in the feature column.

Scaling. As mentioned before, to comply with the assumptions of the UAT[1] [8], the data vectors must lay within the range $[0, 1]$, the data is scaled in the following form $x_{new} = (x - x_{min})/(x_{max} - x_{min})$.

Stabilizing. The current method implement matrix operations such as the inverse of a matrix and the solutions of a system of equations we must ensure that the feature columns in our dataset are not linearly dependent on other feature column. As there may exist a dataset with this property we replace the original data X for X^* as follows:

$$X_i^* = \begin{cases} X_i \times (1 + \rho_u * \delta_H) & \text{if } X_i \neq 0 \\ \rho_u * \delta_H & \text{if } X_i = 0 \end{cases} \tag{4.3}$$

where ρ_H denotes a uniformly distributed random variable such that $0 \leq \rho < 1$ and $\delta = 10^{-6}$. The resulting vectors are no longer unstable. It is possible to do the desired operations on them.

Sampling. Some of the datasets may have a large amount of data which can seriously impact on the performance of the algorithm in the use of memory and the processing time. Even more, there may exist redundant data which do not contribute with extra information which was already learned so it is advisable to work with a subset of the original dataset. A subset which contains the most possible non-redundant data for this the minimum sample size is calculated based on the entropy of each feature vector.

Data Augmentation. There may be datasets with very few instances (i.e. < 600) which can lower the learning power of the algorithm. As assumed in the UAT, the MLPN approximating capabilities refer to continuous functions. It is difficult to ensure continuity from a small finite set of data. As in [9], to ensure this continuity natural splines (as in the step where the missing data is filled) are used to generate data from the cubic polynomial functions calculated between each value in each feature column. The resulting spline will exhibit the minimum curvature of all possible collocating functions if the following conditions are fulfilled[45]: Let $S(x)$ be a natural spline, $S_i(x)$ its *i-th* element and $f(x)$ the unknown

[1] Universal Approximation Theory

function corresponding to the observed data, then a) $S(x_i) = f(x_i) i = 0, \cdots, n$, b) $S'_i(x_i) = S'_{i+1}(x_i) i = 1, \cdots, n-2$, c) $S''_i(x_i) = S''_{i+1}(x_i) i = 1, \cdots, n-2$, and d) $S''(x_0) = S''(x_n) = 0$. This is the only way (mathematically) which minimizes the curvature in the structure of the values in each feature. In the cases data contained few values it was augmented up to 1000 or 1200 records.

Compression analysis. In this step the data is compressed with the PPMZ2 algorithm [46] in order to obtain the compression ratio and the compressed file size which are going to be used in later analysis of the dataset. This values represent the actual effective information within the file, which may be an important parameter which affects in some way the number of terms in the polynomial model.

4.2 Genetic Multivariate Polynomial Model Evaluation

The genetic polynomial model is implemented in a specialized software. This software runs the algorithms in the methodology to generate a multivariate polynomial model from a dataset. The evaluation of the 46 experiments (32 for training and 14 for testing) was made with the original software. Each problem took about 2 hours to get evaluated. This software lasts too much for large parameter values (i.e. terms in the polynomial > 10) and relatively large volumes of data (let us say tuples > 1500 and attributes > 10). To improve the performance of the methodology this software is replicated using programing language c++. The use of pointers made this new version about 4 times faster (and in some cases generating more accurate models than the original one) and was used to generate the multivariate polynomial models for the experiments shown in the section of cases of study.

In the newly developed software the data files must be defined and some parameters adjusted as seen in figure 4.1.

```
--- FILE DEFINITION

TRAIN FILE = DB43-mv/TRAIN.TXT
TEST FILE  = DB43-mv/TEST.TXT
RECALL DATA  = TEST
NUMBER OF FIELDS = 11
ROWS IN TRAIN FILE = 1040
ROWS IN TEST FILE = 260
-------------------------------

STABILIZATION FACTOR = 0.000001
QUASI MINIMAX CONDITION = true
QUASI MINIMAX VALUE = 0.050000
OPTIMIZE = rms
-------------------------------

CROSS PROBABILITY = 1.000000
MUTATION PROBABILITY = 0.050000
GENERATIONS = 100
INDIVIDUALS = 20
MAXIMUM DEGREE = 21
-------------------------------

NUMBER OF TERMS = 10
```

Figure 4.1: Parameters configuration.

In the box:

1. *Number of Terms* is user specified and sets the number of terms the polynomial will have.

2. *Quasi-Minimax* indicates that the Ascent Algorithm will not stop when the convergence condition is reached but rather when $1 - \varepsilon_\phi/\varepsilon_\theta$ is less or equal than the value specified in the quasi-minimax box.

3. *Stabilization Factor* indicates the factor δ by which the data is going to be stabilized as mention before.

4. *Cross Probability* stands for the cross probability of the EGA and is the probability of two individuals to perform the cross over procedure.

5. *Mutation Probability* is the mutation probability of th EGA which indicates the probability of every bit in the individuals to change.

6. *Individuals and Generations* is the number of individuals in the EGA and generations is the number of iterations the algorithm will run.

7. *Optimize* indicates which error function will be the objective function of the EGA to be minimized.

8. *Generations* establishes the number of generations of the EGA.

9. *Individuals* is the number of individuals in each generation of the EGA.

10. *Maximum Degree* is the degree of the polynomial model. This value should be one of the possible values shown in table 3.1.

To test the first 32 datasets the parameters were set as follows: the quasi-minimax value was set to 5%, the regularization factor value is 10^{-6}, the cross probability was established to 1 to ensure the individuals always do the crossover procedure, the mutation probability is set to 0.05, the number of individuals was set to 50 and the number of generations to 100 and the minimizing error function is set to RMS in all the cases. The number of terms varies from 3 to 15 taking the values 3,6,9,12 and 15 to test the performance of the algorithm in each case (Table 4.1). To train the model 80% of the whole data was used as training set and the remaining 20% as test set. The evaluation of each problem in each of the term values took up to 1 hour depending on various characteristics of the dataset as the number of fields, the number of records and the number of terms in the polynomial model. In some cases the software outcome had repeated terms (i.e. it has the same variables with the same exponents) and the number of terms is reduced.

From the five experimental values obtained (one for each term value tested) we find an approximated relationship $RMS_{error} = f(T)$, where T is the number of terms. To do this, a regression tool (shown in figure 4.2) is used to find the equation which better the five points given by the pairs (RMS error, number of terms) obtained from the evaluation of the algorithm.

Number of Terms	Resulting RMS
3	0.264978493
6	0.257745103
9	0.294190045
10	0.328746804
13	0.312208313

Table 4.1: RMS error is calculated for each value of T

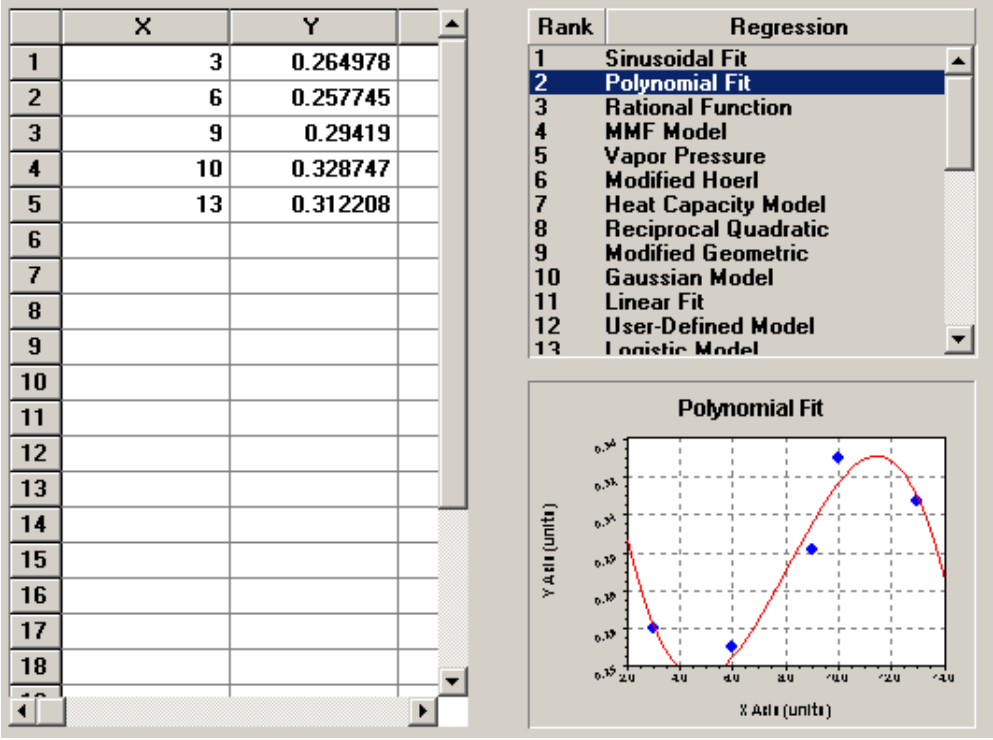

Figure 4.2: Regression tool[47] to find the error as function of the number of terms.

Then we obtain the full range of values of the RMS errors evaluating the calculated equation with missing number of terms in $[3, 15] \subset \mathbb{Z}$. In the case of the example of the figure 4.2 the calculated equation which approximates the five points is $y = a + b\cos(cx + d)$ where $a = 0.29941952, b = 0.03878902, c = 2.36955707, d = 1.95233267$. With the full range in hand now we can calculate the average RMS error and the standard deviation.

Number of Terms	**Resulting RMS**
3	0.264978493
4	0.315765152
5	0.31224719
6	0.257745103
7	0.33635578
8	0.281217922
9	0.294190045
10	0.328746804
11	0.261903206
12	0.31942511
13	0.312208313

Table 4.2: The full range of values is calculated from the approximating equation

The minimum error is calculated from this range of values. Recalling the Chebyshev theorem[48]:

$$P(\mu_T - k\sigma_T \leq T \leq \mu_T + k\sigma_T) > 1 - \frac{1}{k^2} \tag{4.4}$$

and making $k = 3$, the probability of finding the minimum expected error 3 standard deviations below the average value is $\approx 88\%$. For the example of the table 4.2, $\sigma = 0.028110234$, $\mu = 0.298616647$ and the minimum expected error $= \mu - 3\sigma = 0.214285945$. Afterwards, the number of terms is put as function of the RMS error and a new approximation equation is calculated from this values. The approximation equation calculated for the values in table 4.2 is of the form $y = \frac{(a+bx)}{1+cx+dx^2}$ with coefficients $a = 3.99, b = -15.1, c = -5.48, d = 6.41$. Evaluating the minimum expected error in the equation we find the expected value of terms to minimize the error in the data set.

In table 4.3 it is shown the name of the evaluated datasets and the information which led to the expected number of terms(T) which yields the minimum RMS error when generating a multivariate polynomial model. As it is seen, the information which was extracted from each dataset are: the number of attributes (or fields), the number of tuples, the size in bytes of the dataset, the compressed size of the dataset in bytes and the ratio of the compression of the file.

ID	Dataset Name	# Attributes	# Tuples	Size	Comp Size	Comp Ratio	Expected T
1	Breast Cancer wisconsin	10	364	38121	1003	38.0070	8
2	Protein localization sites	7	336	28140	7548	3.7281	7
3	Servomechanism	13	167	22610	771	29.3256	6
4	Yeast	9	1484	140066	32021	4.3742	7
5	Abalone	11	3133	360864	39891	9.0463	6
6	Car Evaluation	22	1728	216384	34636	6.2474	10
7	CPU	36	209	28398	5348	5.3100	9
8	Hepatitis	16	125	30384	5906	5.1446	7
9	Wine	14	125	25986	8876	2.9277	13
10	IRIS	4	150	9120	2685	3.3966	5
11	Facebook	21	500	55200	14521	3.8014	6
12	Whole Sale	10	440	46112	13649	3.3784	8
13	3D road network	2	871	53261	15941	3.3400	10
14	Air quality	8	1200	217522	52252	4.1600	6
15	Air foil self noise	5	1503	182089	43915	4.1500	4
16	Concrete strength	8	1030	18752	52062	3.5900	8
17	Auto mpg	6	398	64368	14798	4.3500	5
18	Credit approval	15	690	215937	42663	5.0600	6
19	Gas turbine propulsion	2	1000	966816	259129	3.7300	5
20	Energy efficiency	11	768	185506	36553	5.0700	9
21	Fertility diagnosis	24	100	50934	8950	5.6900	6
22	Student knowledge	5	403	48989	11119	4.4100	5
23	Ecoli	7	336	54386	12223	4.4500	10
24	Glass type	5	214	43368	11014	3.9400	1
25	Indian liver	10	583	129229	28143	4.5900	10
26	Steel plates	22	1000	461770	114179	4.0400	6
27	Vertebral column	7	310	50200	13628	3.6800	11
28	Bank note authentication	4	500	138766	38100	3.6400	6
29	Leaves	9	340	68694	19654	3.4951	9
30	Mammographic masses	5	835	101261	18568	5.4535	9
31	Indian woman diabetes	8	768	104448	30897	3.3805	9
32	Seeds	4	210	15960	5836	2.7347	7

Table 4.3: Resulting number of terms which yielded the minimum RMS error from the evaluation of the methodology

To find some relation between the number of terms and the features which were extracted from the datasets we put the expected number of terms in function of each one of them (Attributes, Tuples, Size, Compression Size and Compression Ratio). As we can see from the Table 4.3 there are some outliers (as the abalone dataset which presents very large values in comparison to the other datasets). These outliers were omitted in this first analysis to focus on the other data. In the scatter plots shown in figures from 4.3 to 4.7 we can observe that the number of terms behaves in accordance to the features. The plots suggest that when the values of the features are small the number of terms seem to agglomerate in a different way than the case when the values are large. However, it is not clear that there exists a linear relation between each one of these features separately and the number of terms. This is fine since the aim of this work is to find a relationship of this features of the integrated data and a suitable number of terms. To find out whether there is a higher order relationship between the data collected a MLPN is used and the results are analyzed.

The conclusion from this first analysis is that there may be a higher degree relationship (which is why we used a MLPN) for the gathered data and the results may then be analyzed and tested as illustrated in figures from 4.3 to 4.7.

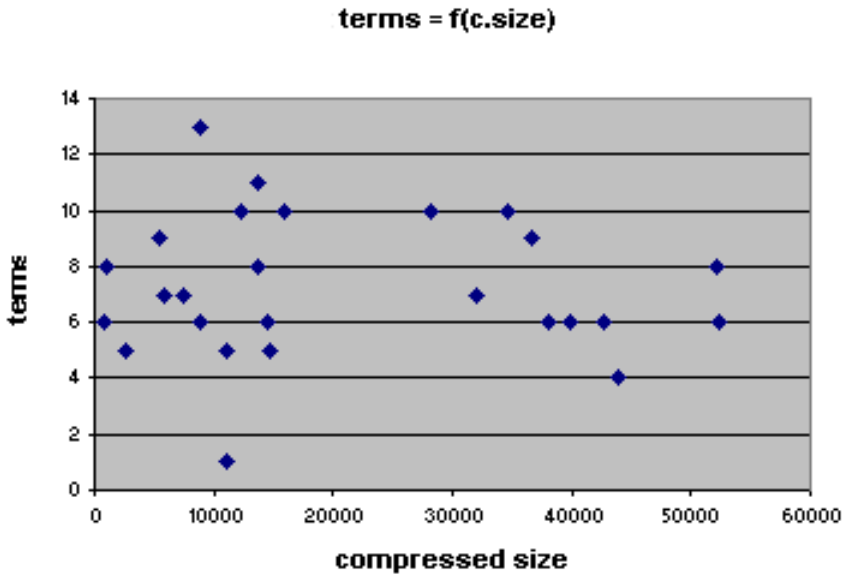

Figure 4.3: Terms as function of compressed size

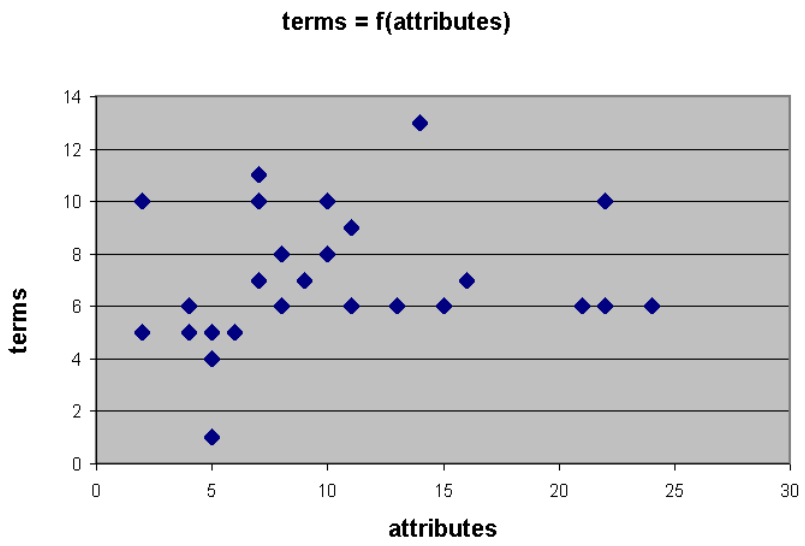

Figure 4.4: Terms as function of number of attributes

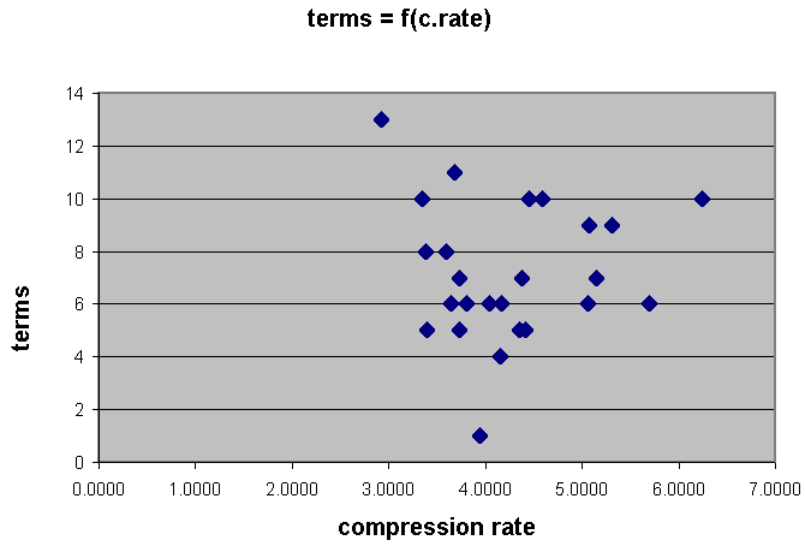

Figure 4.5: Terms as function of compression rate

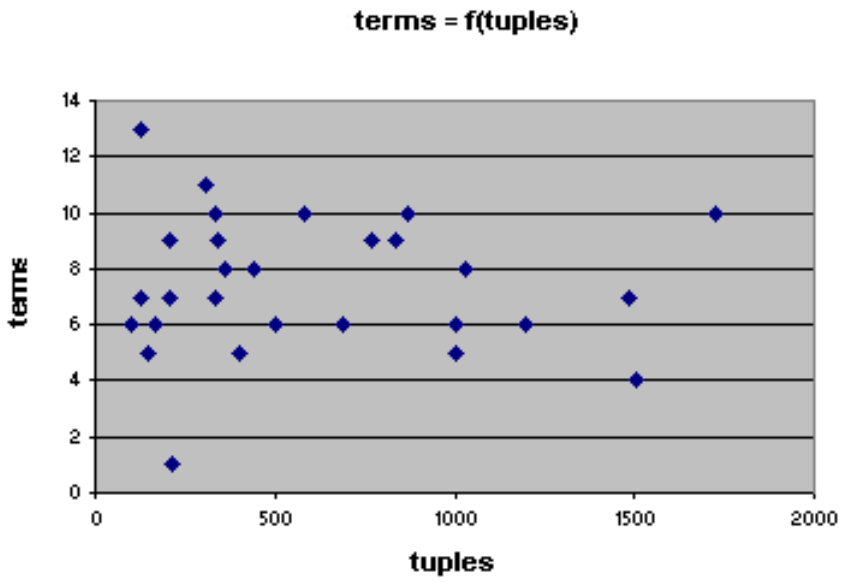

Figure 4.6: Terms as function of number of tuples

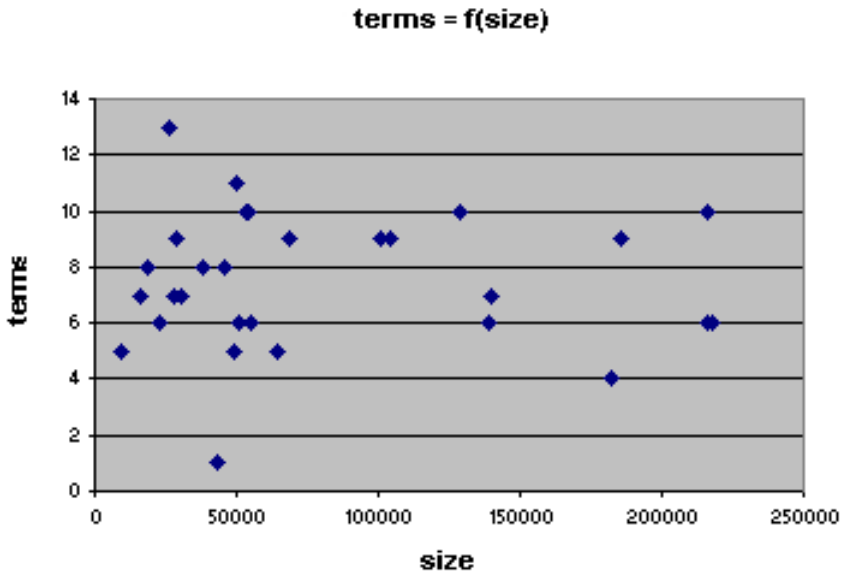

Figure 4.7: Terms as function of size in bytes

Chapter 5

Determining the Number of Terms

As mentioned before, a MLPN was used to find the relationships between the features of a dataset and a desired number of terms in the polynomial model. The data must be scaled to the range $[0, 1)$. Therefore, before training a MLPN the datasets must be preprocessed. To do so the same methodology described in the previous chapter is applied to the data collected shown in table 4.3. The preprocess is described next.

1. More data is created using natural splines. During this process the data is stabilized. The data is augmented up to 500 tuples.
2. The data is scaled in the range $[0, 1)$
3. A correlation analysis is made, and the results show that the variables *size of the file* and *compressed size* are correlated with a $> 90\%$ of relevance so the *size of the file* variable is removed from the final dataset.
4. The dataset is compressed to obtain the compression ratio ($cr \approx 2.578893$) with which we can calculate the effective sample size, i.e. the effective number of tuples which may contain the necessary information to be representative of the population.
5. The number of tuples for the effective sample size is $s = 500/2.578893 \approx 194$ tuples, with this piece of information we calculate the number of hidden neurons as done in [9] in the architecture of the neural network (figure 5.1).

The dataset is divided in train and test sets (80% and 20% respectively) the architecture of the MLPN was tuned varying the values of the parameters to find the best configuration of parameters which are described next.

The four input variables correspond to: 1) the number of tuples in the dataset, 2) the number of attributes, 3) the compressed size of the data set and 4) the compression rate. The MLPN has one hidden layer as the UAT suggests, the number of hidden neurons are tested with values from 1 to 3 neurons to ensure that 2 hidden neurons are enough to train the MLPN. The activation function in the hidden layer was tested with hyperbolic tangent

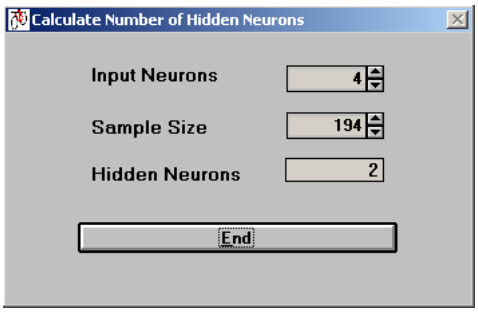

Figure 5.1: Calculation of hidden neurons in MLPN

function, logistic function and linear function. The learning strategy used is the well known backpropagation training algorithm. To improve convergence the learning rate is set in the hidden and output layers. The learning rate was tested in the hidden layer with the values $.1, .2, .3$ and $.25$, on the other hand the learning rate in the output layer was tested for the values $.1, .01$ and $.001$. In the learning process a momentum term was added. Adding momentum allows the adjustments on the weights to depend also on the previous changes. If the momentum is equal to 0, the change depends only on the gradient, while a value of 1 means that the change will only depend on the last change. The value of the momentum in the hidden and output layers was tested for values $0.3, 0.5$ and $.8$. The weights were initialized with uniform random numbers in the range $[-0.1, 0.1] \in \mathbb{R}$. The learning strategy used was stochastic in which each input creates a weight adjustment. The MLPN is trained for 10,000 epochs with a test every 100 epochs.

To implement and train the MLPN architecture the software Data Engine[49] was used. After testing several configurations of the architecture of the MLPN, the best results were obtained with the following values.

- **Architecture**
  - *Input Layer*
    - Number of Neurons = 4
    - Activation Function = Linear
  - *Hidden Layer*
    - Number of Neurons = 2
    - Activation Function = Logistic
  - *Output Layer*
    - Number of Neurons = 1
    - Activation Function = Linear
  - Number of connections = 10
- **Learning Method**
  - Learning = Backpropagation

  - Learning Strategy = Stochastic Learning
  -

- **Learning Parameters**
  - *Hidden Layer*
    - Learning Rate = 0.25
    - Momentum = 0.5
  - *Output Layer*
    - Learning Rate = 0.001
    - Momentum = 0.8

In figure 5.2 we can see the architecture of the trained MLPN. There were in total 10 connections in the hidden layer denoted with w_{ij} where $i = 0, ...4$ for the input neurons (and the bias input) and $j = 1, 2$ for the hidden neurons and the bias. In the output layer there are 3 connections denoted with w'_{j1}. The output of the MLPN is a function represented by the trained model which yields the number of terms(T) that induces the minimum RMS error in the model.

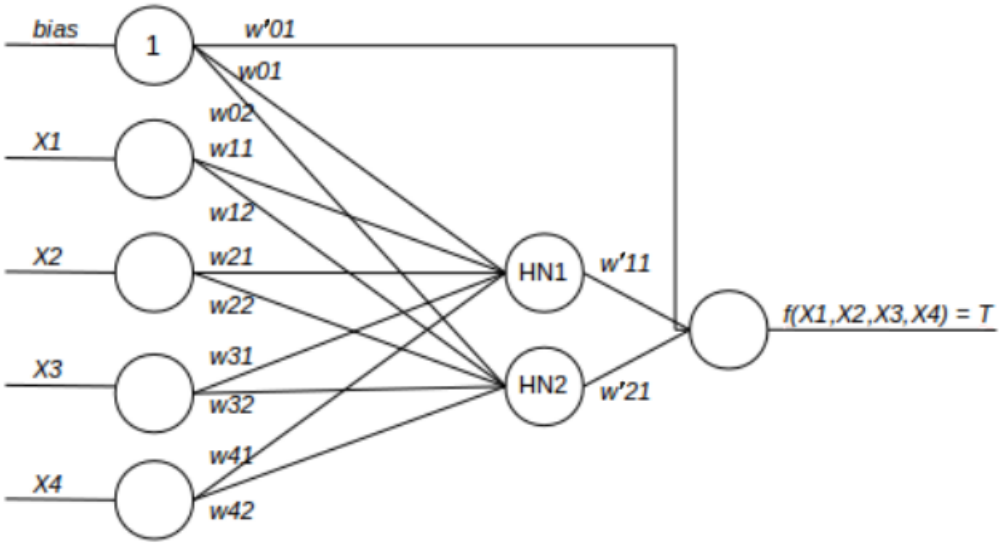

Figure 5.2: Architecture and connections of the trained MLPN

The model was trained and the learning curve presented during the epochs is shown in figure 5.3. It is see that after a few epochs the RMS error does not change significantly.

After the MLPN was trained with the best configuration of parameters the RMS errors shown in table 5.1 indicate that, in fact, there exists a relationship of the number of terms as function of the characteristics gathered. This value is enough to try to establish a search bound for the number of terms. Moreover the trained MLPN can be used as a tool to calculate the terms in a polynomial model which leads to the minimum error given the characteristics

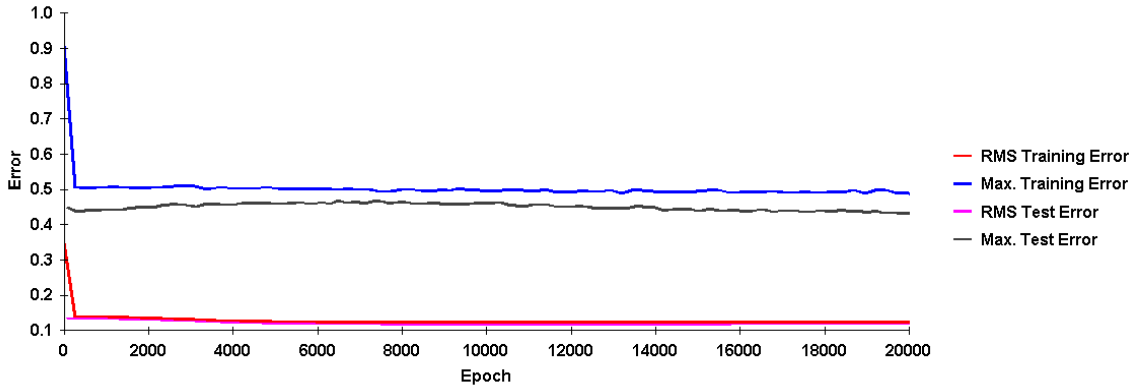

Figure 5.3: MLPN learning curve.

Max. Training Error	0.49867762
RMS Training Error	0.12281852
Max. Test Error	0.45594774
RMS Test Error	0.11719234

Table 5.1: Best training and test errors of the MLPN

with which the MLPN was trained (number of attributes, number of tuples, compressed file size and compression ratio).

The weights for the connections of the MLPN are shown in table 5.2. These values plus the activation functions depicted above express the relationship of the extracted characteristics of a dataset and the optimum number of terms such that the RMS error of a multivariate polynomial model generated with the methodology studied in this work is, with high probability, the minimum within a range of possible values. These weights determine the architecture of the MLPN to allow us to build a program that can calculate this parameter (the number of terms) in an automated way.

The next step is to validate it against the remaining experiments (14 experiments). The data of the experiments were preprocessed the same way as the first 32 experiments. The information obtained from the preprocessing step is shown in table 5.3. This information was fed to the MLPN to get the expected number of terms in the polynomial model which yields the best fit error. In table 5.4 we can observe the number of terms (*Estimated* T) which are expected to be near the optimum fit. In the table it is shown near values for the number of terms are evaluated ($RMS(T \pm 2)$) to confirm that the calculated value is, or is near, to the suitable one. In each row of the table 5.4 the lower RMS is highlighted to make notice of the number of terms which resulted in that error.

In figure 5.4 we can see that in most cases the optimum value of the number of terms is within $T \pm 2$. As mentioned before, the experiments are of heterogeneous nature. In table 5.4 the actual values of the RMS errors given by the tested terms are shown. Also it is seen the estimated number of terms given by the MLPN which is expected to yield the optimum error value. These values (the optimum number of terms) are obtained from the information

First Layer	
Weights	Values
$w_{0,1}$	9.93878
$w_{0,2}$	−1.277238
$w_{1,1}$	12.794103
$w_{1,2}$	2.149855
$w_{2,1}$	−32.754846
$w_{2,2}$	9.336392
$w_{3,1}$	1.290772
$w_{3,2}$	−8.214893
$w_{4,1}$	8.711606
$w_{4,2}$	−21.087847

Second Layer	
Weights	Values
$w'_{0,1}$	−0.018833
$w'_{1,1}$	0.452167
$w'_{2,1}$	0.531239

Table 5.2: Weights of the MLPN which calculates the optimum number of terms of a multivariate polynomial model

ID	DB Name	# Attributes	# Tuples	compressed size	compression ratio
1	pima indian diabetes database	8	768	32,360	3.227688
2	yacht hydrodynamics	6	308	9,174	3.558752
3	image segmentation	20	2310	213,221	3.42
4	Analizing categorical data	7	900	25,627	4.249424
5	appendicitis	7	106	3,024	3.189815
6	balance	4	625	10,881	4.365407
7	baseball salaries	13	337	21,125	3.366011
8	hayes-roth	4	160	298	22.013422
9	house prices	15	1300	111,601	2.807322
10	laser	4	993	21,357	3.533642
11	mv synthetic	10	1300	75,388	2.862524
12	phonems	5	1200	38,677	2.823383
13	treasury	5	1049	36,651	2.60454
14	weather izmir	7	1461	53,229	3.32114

Table 5.3: Validation experiments

ID	Estimated T	Rounded T	RMS(T-2)	RMS(T-1)	RMS(T)	RMS(T+1)	RMS(T+2)
1	8.6634990936	9	0.412718594	**0.406120106**	0.4200713649	0.410491696	0.418528407
2	7.4266991038	7	0.037526053	0.061839319	0.03250066	**0.031067453**	0.032766126
3	2.8167562596	3	**0.207598173**	0.214679407	0.229129966	0.224681547	0.225307872
4	8.477513871	8	0.067469116	0.050167953	**0.04093897**	0.043811914	0.04796777
5	7.2928672806	7	0.337443754	0.319763624	0.318348383	0.313419036	**0.313209334**
6	7.7389574277	8	0.340522415	0.325661129	0.334526005	**0.308926934**	0.324996133
7	7.7052644686	8	0.119055026	0.11559727	**0.108739684**	0.124787914	0.117380777
8	6.2126030641	6	0.258552437	0.245556743	**0.241290077**	0.26038423	0.269854381
9	8.0014918908	8	0.101553883	**0.096687173**	0.101330919	0.102006909	0.100162057
10	8.5616749213	9	0.086605404	0.0839230232	0.082046364	0.078676946	**0.076810137**
11	7.5255321089	8	0.074918077	0.063573936	0.076393697	**0.056997275**	0.06215833
12	7.0866960532	7	0.395806995	0.400738082	0.405261065	**0.392372343**	0.399877438
13	8.6697108438	9	0.068657151	0.068666017	0.062620414	**0.061057446**	0.062685869
14	5.7585208346	6	0.14429961	0.137041337	**0.130248103**	0.140362715	0.135891573

Table 5.4: Terms calculated by MLPN on validation experiments

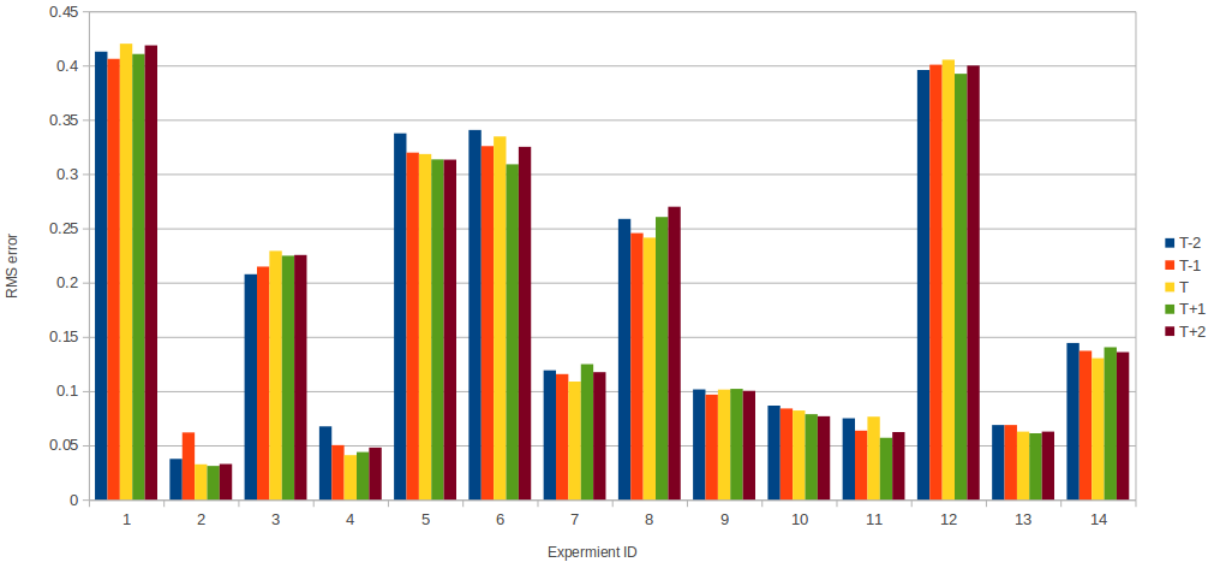

Figure 5.4: RMS errors calculated with near values of T.

shown in the table 5.3. Even when trained with different kind of experiments the objective of estimating a proper number of terms in this polynomial model is achieved. With the calculated number of terms of the test problems it is possible to set an appropriate lower bound of T calculating the mean (μ_T) and the standard deviation (σ_T) of the resulting number of terms given by the trained MLPN. This is derived from Chebyshev's theorem (equation 4.4) which does not assume that the form of the distribution of T, making these values general. Thus, given for example $k = 2.5$, and having $\mu_T = 7.2812705158$ and $\sigma_T = 1.55237372$.

$$P(3.4003362157 \leq T \leq 11.1622048159) > .84 \tag{5.1}$$

Without loss of generality if $T_{min} = \mu_T - 2.5\sigma_T$ then $P(T > T_{min}) > .84$ which is a good interval to start with. To start looking for a good value for T it is enough (given the previous value of k) to search in the interval $[3, 11] \in \mathbb{Z}$.

5.1 Cases of Study

In this section 5 problems are selected to illustrate the performance of the methodology studied and replicated in this work. The resulting multivariate polynomial model was generated with the new version of the specialized software with which the first experiments were evaluated. The datasets were preprocessed the same way the training data was and is described shortly in each case. The description of the datasets is provided in the source repositories. The number of terms selected in these experiments are those of the minimum RMS error shown in figure 5.4. It can be noticed that in some of the cases the RMS error was lower than the values obtained from the old version of the software. The selected experiments are: 1) Yatch Hydrodynamics with T=8, 2) Analyzing Categorical Data with T=8, 3) Laser with T=11, 4) mv synthetic with T=9 and 5) Treasury with T=8. In some of the cases the results obtained were better than the ones in the validation process. The descriptions of the datasets are extracted from UCI Machine Learning Repository[41] and KEEL repository[42].

5.1.1 Yacht Hydrodynamics

Dataset description: Variations concern hull geometry coefficients and the Froude number, the measured variable is the residuary resistance per unit weight of displacement (number 7):

1. Longitudinal position of the center of buoyancy
2. Prismatic coefficient
3. Length-displacement ratio
4. Beam-draught ratio
5. Length-beam ratio

6. Froude number

7. Residuary resistance per unit weight of displacement

Preprocessing: The data was stabilized with a factor of 0.000001 and then scaled into the range $[0, 1]$.

The polynomial model generated with 8 terms with the following coefficients and combination of variables is shown in table 5.5

Resulting Model:

Coefficient	Variables
0.0131043094	$x_4 x_6^{10}$
0.016130096	x_1
0.0137365978	$x_3^5 x_5^2$
-0.0087693701	-
-0.0304941186	x_3^3
0.9395045761	x_6^5
-0.2292907978	$x_2 x_6^4$
0.0581267797	x_2^7

Table 5.5: Polynomial model for Yacht Dataset

The test RMS error is 0.0263822472 and the performance of the model is shown in the graphic in figure 5.5.

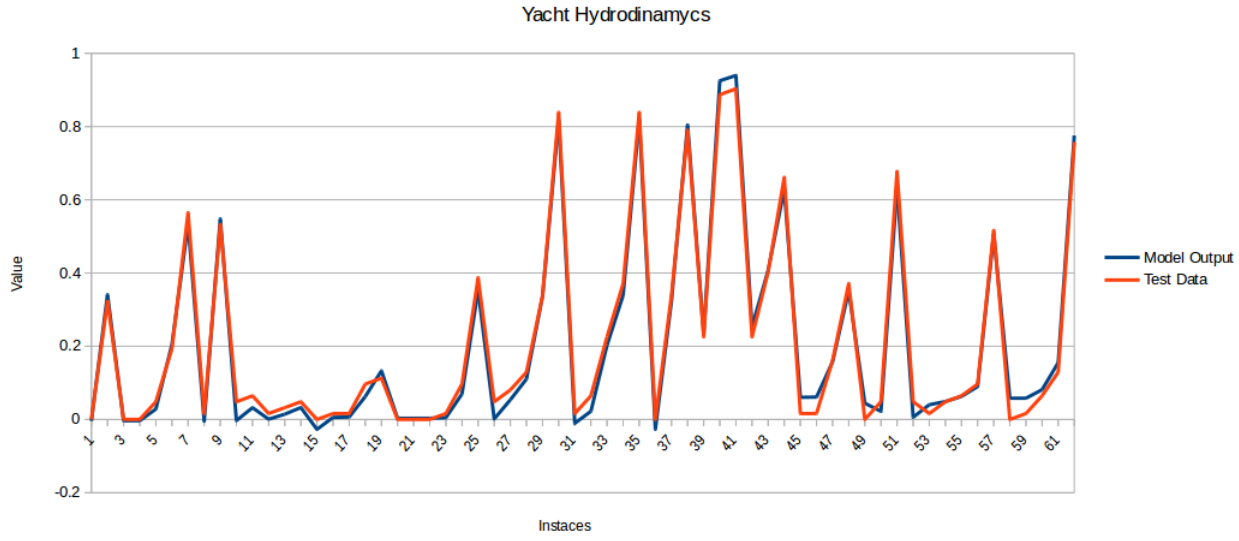

Figure 5.5: Recall graph for yacht dataset.

5.1.2 Analyzing Categorical Data

Dataset description: This section describes main characteristics of the ANACAT data set and its attributes. This is one of the data sets used in the book Analyzing Categorical Data by Jeffrey S. Simonoff, Springer-Verlag, New York, 2003. The data contains information about the decisions taken by a supreme court. The attribute description and domain are presented next.

1. Actions taken, domain values: $[0.0, 11.0]$
2. Liberal, domain values: $[0.0, 1.0]$
3. Unconstitutional, domain values: $[0.0, 1.0]$
4. Precedent alteration, domain values: $[0.0, 1.0]$
5. Unanimous, domain values: $[0.0, 1.0]$
6. Year of decision, domain values: $[1953.0, 1988.0]$
7. Lower court disagreement, domain values: $[0.0, 1.0]$
8. Log exposure, domain values: $[0.0, 2.3]$

Preprocessing: This data was stabilized with a factor of 0.000001 and then scaled into the range $[0, 1]$.

The polynomial model generated with 8 terms with the following coefficients and combination of variables is shown in table 5.6.

Resulting Model:

Coefficient	Variables
0.139010414487	$x_4x_6^4$
-0.629731670578	x_1x2^2
-0.916572823218	x_6^9
0.003401718461	$x_2x_4^4$
0.755120351943	$x_1^3x_2^6$
0.025789136673	x_6
0.006570008701	$x_2x_3^2$
0.990483840766	$-$

Table 5.6: Polynomial model for ANACAT Dataset

The test RMS error is 0.0429532553 and the performance of the model is shown in the graphic in figure 5.6.

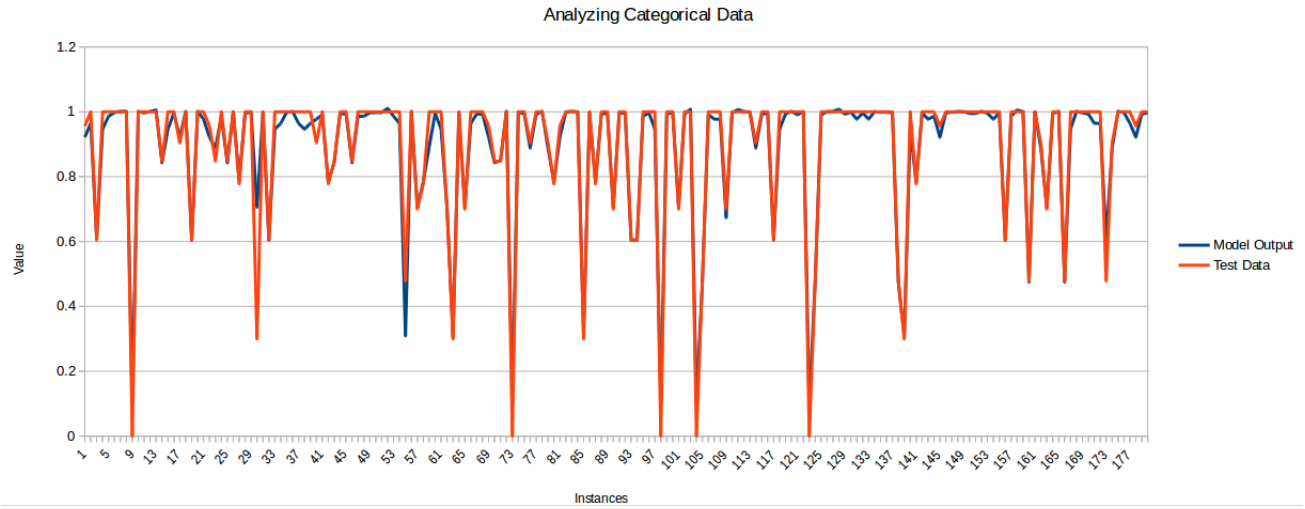

Figure 5.6: Recall graph for ANACAT dataset.

5.1.3 Laser

Dataset description: This data set was originally a univariate time record of a single observed quantity, recorded from a Far-Infrared-Laser in a chaotic state. The original set 1000 points has been adapted for regression by considering every set of four consecutive values as inputs, and the next one as output. Duplicated instances has been removed. The dataset is meant to be for regression tasks. It was generated in a laboratory, it contains 4 features, 993 instances and has no missing values. The attributes description and domain are presented next.

1. Input1, domain values: $[2.0, 255.0]$
2. Input2, domain values: $[2.0, 255.0]$
3. Input3, domain values: $[2.0, 255.0]$
4. Input4, domain values: $[2.0, 255.0]$
5. Output, domain values: $[0.0, 255.0]$

Preprocessing: The data was stabilized with a factor of 0.000005 and then scaled into the range $[0, 1]$.

The polynomial model generated with 11 terms with the following coefficients and combination of variables is shown in table 5.7.

Resulting Model:

The test RMS error is 0.0735489535 and the performance of the model is shown in the graphic in figure 5.7.

Coefficient	Variables
-6.5820194996	$x_1^2x_4^3$
-21.5876022399	$x_2x_4^2$
0.471648252	x_1
3.3484027071	$x_1^2x_4$
1.0830207183	x_4
-6.1327683209	$x_3^5x_4^4$
-0.1930334406	x_1^3
0.0237993135	$-$
26165.9848278958	$x_1^2x_2^3x_4^4$
-18.9788413155	$x_1^4x_2$
-2.249784998	$x_1x_3x_4$

Table 5.7: Polynomial model for Laser Dataset

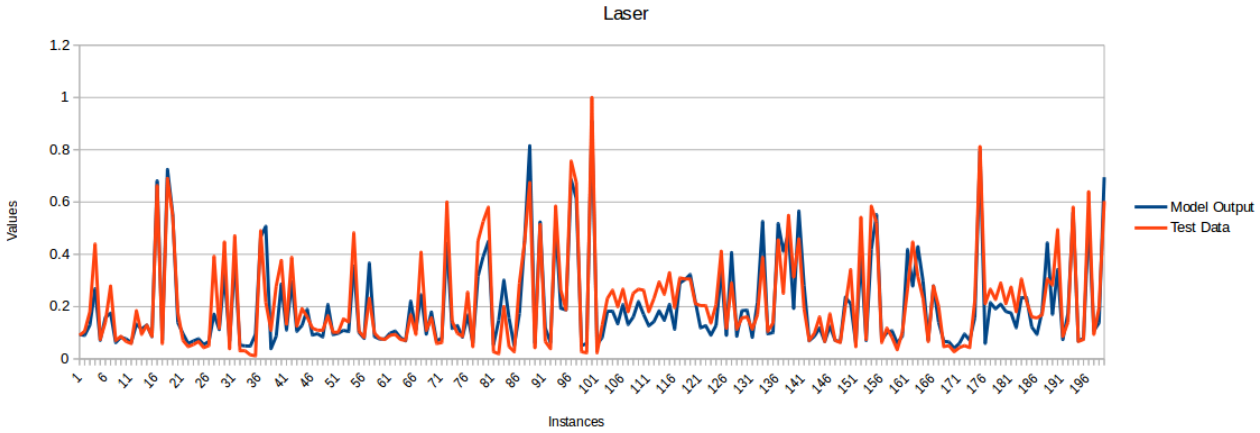

Figure 5.7: Recall graph for Laser dataset.

5.1.4 Mv Synthetic

Dataset description: This is an artificial data set with dependencies between the attribute values. The cases are generated using a fixed method (see KEEL repository reference for more information). The variables and their domain are described next.

1. X1, domain values: $[-5.0, 5.0]$
2. X2, domain values: $[-15.0, -10.0]$
3. X3, domain values: $[0, 2]$
4. X4, domain values: $[-7.5, 2.5]$
5. X5, domain values: $[-1.0, 1.0]$

6. X6, domain values: $[-37.5, 12.5]$
7. X7, domain values: $[0, 1]$
8. X8, domain values: $[0, 1]$
9. X9, domain values: $[100.0, 500.0]$
10. X10, domain values: $[1000.0, 1200.0]$
11. Y, domain values: $[-41.8222, 2.49978]$

Preprocessing: The data was stabilized with a factor of 0.000001 and scaled into range $[0, 1]$. After a correlation analysis no strong correlations were found. The polynomial model generated with 9 terms with the following coefficients and combination of variables is shown in table 5.8.

Resulting Model:

Coefficient	Variables
$6.133907E-06$	$x_2x_5^2$
0.8220042186	x_8^3
$-2.940906E-06$	$x_1x_5x_7$
-0.0119412039	$-$
-0.0003841958	x_8
0.225612967	x_1
-1.024745422	$x_6x_8^2$
1.1220534102	x_6
-0.1217373969	$x_1^3x_8^2$

Table 5.8: Polynomial model for Mv Synthetic Dataset

The test RMS error is 0.0402891528 and the performance of the model is shown in the graphic in figure 5.8.

5.1.5 Treasury

Dataset description: This file contains the Economic data information of USA from 01/04/1980 to 02/04/2000 on a weekly basis. From given features, the goal is to predict 1 Month CD Rate. The dataset was meant to be a regression problem, it was obtained from the real world, contains 15 features with 1049 instances and no missing values. The attributes description and domain are presented next.

1. (v1)1Y-CMaturityRate, domain values: $[77.055, 142.645]$
2. (v2)DemandDeposits, domain values: $[105.6, 533.0]$

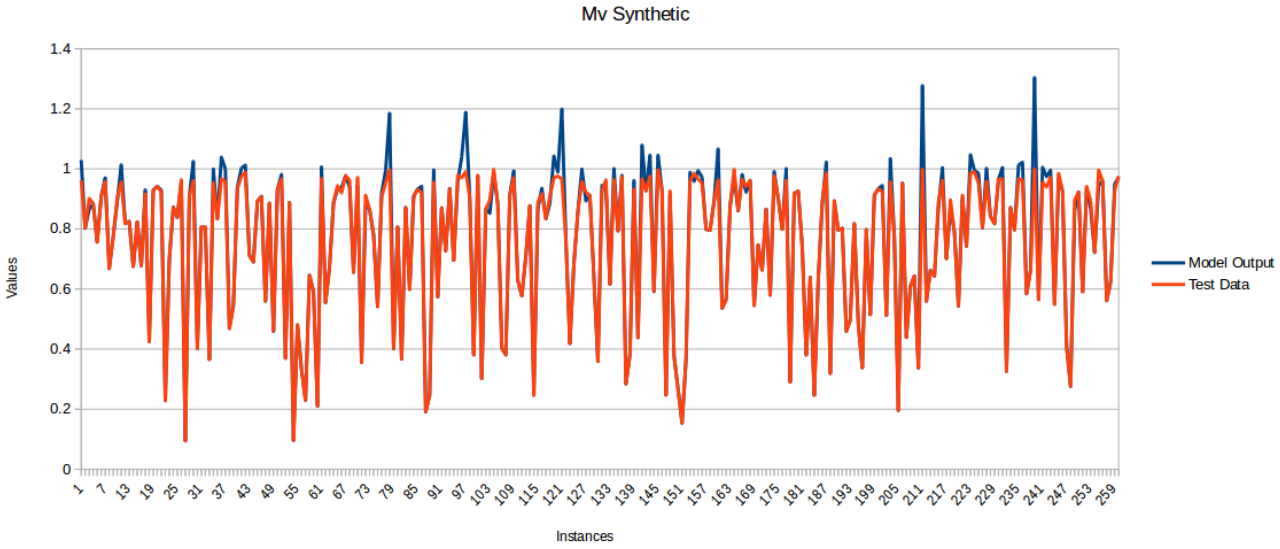

Figure 5.8: Recall graph for Mv Synthetic Dataset

3. (v3)30Y-CMortgageRate, domain values: $[3.02, 17.15]$
4. (v4)FederalFunds, domain values: $[225.8, 412.1]$
5. (v5)3M-Rate-AuctionAverage, domain values: $[6.49, 18.63]$
6. (v6)MoneyStock, domain values: $[2.86, 20.06]$
7. (v7)3M-Rate-SecondaryMarket, domain values: $[2.67, 16.75]$
8. (v8)CheckableDeposits , domain values: $[381.1, 1154.1]$
9. (v9)3Y-CMaturityRate, domain values: $[2.69, 16.76]$
10. (v10)LoansLeases, domain values: $[269.9, 803.4]$
11. (v11)5Y-CMaturityRate , domain values: $[4.09, 16.47]$
12. (v12)SavingsDeposits, domain values: $[868.1, 3550.3]$
13. (v13)BankCredit, domain values: $[4.17, 16.13]$
14. (v14)TradeCurrencies, domain values: $[175.6, 1758.1]$
15. (v15)Currency, domain values: $[1130.9, 4809.2]$
16. (f1)1MonthCDRate, domain values: $[3.02, 20.76]$

Preprocessing: This dataset was also stabilized with a factor of 0.000001 and scaled into range $[0, 1]$. When a correlation analysis was made many of the variables were correlated (with a correlation rate $> .95$). In the figure 5.9 the correlated variables are shown. The

variables which names have a black mark on the left side are removed from the dataset as they may contribute to the learning algorithm the same way as the other variables they are correlated to.

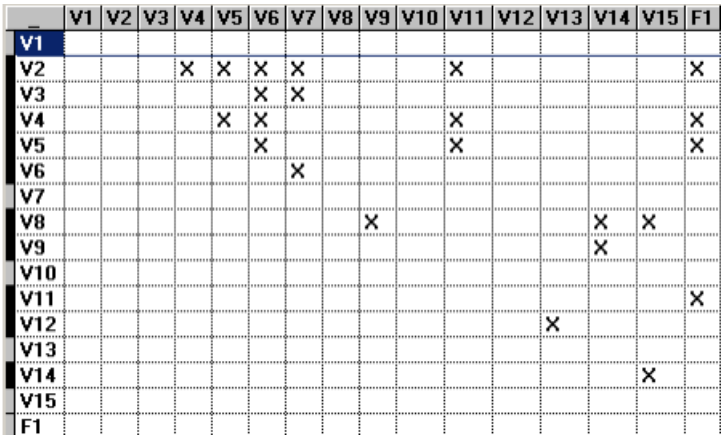

Figure 5.9: Correlated variables of treasury dataset.

The polynomial model generated with 8 terms with the following coefficients and combination of variables is shown in table 5.9.

Resulting Model:

Coefficient	Variables
0.7196892288	x_5
76.5866688178	$x_1x_2^3x_4^3$
0.1710131977	$x_1^4x_2$
−6.6544156776	$x_1x_2x_4$
1.5053294251	$x_1^2x_3$
−0.4845971345	$x_4x_5^2$
1.5111485565	x_2
−4.9529806175	$x_2^4x_4$
−0.4229015954	−
0.2365384053	x_1

Table 5.9: Polynomial model for Treasury Dataset

The test RMS error is 0.0610590967 and the performance of the model is shown in the graphic in figure 5.10.

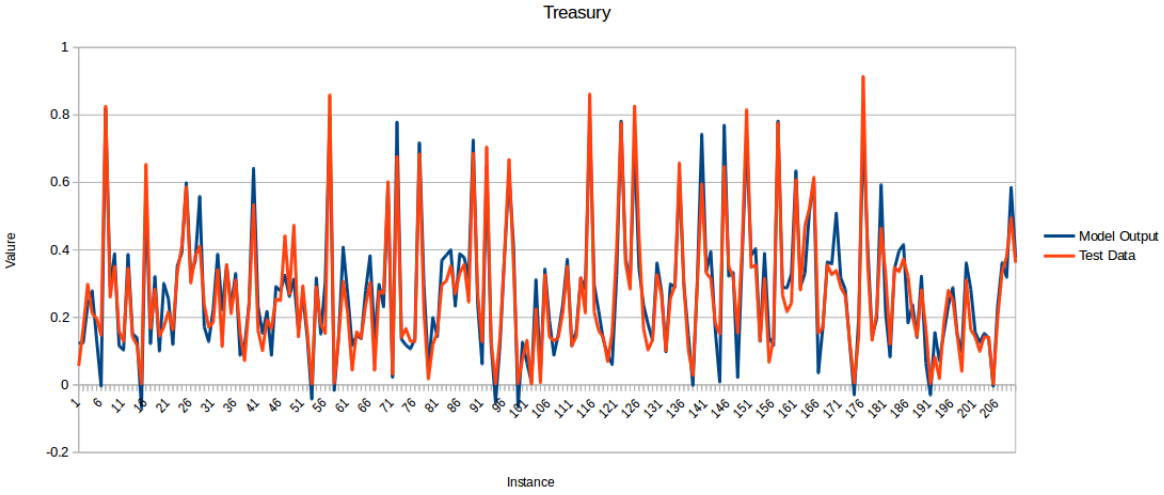

Figure 5.10: Recall graph for Treasury dataset.

Chapter 6

Conclusions

In this work a methodology to generate a multivariate polynomial model was studied, evaluated and replicated. As discussed, one of the parameters which is difficult to tune is the number of terms that will conform the model. In order to improve the parameter tunning of this methodology a set of experimental data was evaluated with a specialized software that implements the algorithms to generate the model. The experimental data took about 80 hours to get evaluated. To overcome this issue a new version of this software was developed. The new version is implemented in C^{++} with the use of pointers to optimize vector operations. It also was able to run up to 4 times faster and, in some cases, generate more accurate models. Some case studies were presented to show the performance of this new version. For completeness of the methodology, the resulting weights of the trained MLPN were implemented into a tool capable of calculating the appropriate number of terms in the multivariate polynomial model. Functionality to apply some preprocessing job to data (as scaling, stabilizing and compressing analysis) is added to this software.

As stated in the hypothesis we could find close bounds in which we can find a suitable value for the number of terms in the genetic polynomial model that was studied in this work. The search was restricted to the interval $[1, 15] \in \mathbb{Z}$ since we wanted to have a manageable polynomial which can be further analyzed. The preprocessing phase was proposed to establish a standard in the format of the data to be used for the methodology to generate the model. Therefore no matter how complex the structure of the data could be (it may have large amount of variables or data rows) after preprocessing it will be numerical data scaled in $[0, 1]$. In the case there is few the data it will be augmented, in the contrary case, a representative subset will be selected to work with.

The MLPN tool allows us to avoid unnecessary search through the range of the number of terms, making the use of this method more efficient. As seen in the case studies, the proper number of terms obtained from the recall experiments may present a better performance. An interesting thing that can be noticed is that in spite of the heterogeneous nature of the datasets used to train and test the MLPN, there was a relationship found in the structure of the datasets and a proper value of the number of terms in the generated multivariate polynomial model. This allows the MLPN to work with any kind of problem.

For future works it would be interesting to analyze the relationships of the resulting

multivariate polynomial models. It would be possible to extend the methodology to work not only with polynomials but enable the genetic algorithm to somehow encode other non-linear functions into the individuals and test the methodology with this new characteristic.

Bibliography

[1] B. Yegnanarayana, *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

[2] S. S. Haykin, S. S. Haykin, S. S. Haykin, and S. S. Haykin, *Neural networks and learning machines*. Pearson Upper Saddle River, NJ, USA:, 2009, vol. 3.

[3] M. D. Buhmann, *Radial basis functions: theory and implementations*. Cambridge university press, 2003, vol. 12.

[4] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[5] M. S. Pukish, P. Różycki, and B. M. Wilamowski, "Polynet: A polynomial-based learning machine for universal approximation," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 3, pp. 708–716, 2015.

[6] R. Kohavi *et al.*, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Ijcai*, vol. 14, no. 2. Montreal, Canada, 1995, pp. 1137–1145.

[7] Y. Sakamoto, M. Ishiguro, and G. Kitagawa, "Akaike information criterion statistics," *Dordrecht, The Netherlands: D. Reidel*, vol. 81, 1986.

[8] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[9] A. Kuri-Morales, "Closed determination of the number of neurons in the hidden layer of a multi-layered perceptron network," *Soft Computing*, vol. 21, no. 3, pp. 597–609, 2017.

[10] T. Ash, "Dynamic node creation in backpropagation networks," *Connection science*, vol. 1, no. 4, pp. 365–375, 1989.

[11] A. Kuri-Morales and A. Cartas-Ayala, "Polynomial multivariate approximation with genetic algorithms," in *Canadian Conference on Artificial Intelligence*. Springer, 2014, pp. 307–312.

[12] M. Darbor, L. Faramarzi, and M. Sharifzadeh, "Performance assessment of rotary drilling using non-linear multiple regression analysis and multilayer perceptron neural network," *Bulletin of Engineering Geology and the Environment*, pp. 1–13, 2017.

[13] S. S. Ganesh, P. Arulmozhivarman, and R. Tatavarti, "Forecasting air quality index using an ensemble of artificial neural networks and regression models," *Journal of Intelligent Systems*, 2017.

[14] H. B. Bashir, S. S. Ahmad, and M. N. Nawaz, "Modeling some plant species distribution against environmental gradients using multivariate regression models," *Kuwait Journal of Science*, vol. 44, no. 4, 2017.

[15] A. Nizamitdinov, Y. Şöhret, A. Shamilov, and T. H. Karakoç, "Statistical model development for military aircraft engine exhaust emissions data," in *Advances in Sustainable Aviation*. Springer, 2018, pp. 177–187.

[16] K.-A. Toh, Q.-L. Tran, and D. Srinivasan, "Benchmarking a reduced multivariate polynomial pattern classifier," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 26, no. 6, pp. 740–755, 2004.

[17] K.-A. Toh, "Pattern classification adopting multivariate polynomials," in *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*. IEEE, 2014, pp. 1–6.

[18] D. Roche, "Introduction to symbolic computation," 2009.

[19] L. Moutinho and G. D. Hutcheson, *The SAGE dictionary of quantitative management research*. Sage, 2011.

[20] C. M. Dayton, "Logistic regression analysis," *Stat*, pp. 474–574, 1992.

[21] H. Thaden, "Effect separation in regression models with multiple scales," Ph.D. dissertation, Georg-August-Universität Göttingen, 2017.

[22] S. D. Marchi, "Lectures on multivariate polynomial interpolation," 2015.

[23] T. A.-R. A.-M. Al *et al.*, "A comparison study of linear and nonlinear regression models," *Journal of Progressive Research in Mathematics*, vol. 12, no. 4, pp. 2039–2056, 2017.

[24] J. M. Acevedo Valle, K. A. Trejo Ramírez, and C. Angulo Bahón, "Multivariate regression with incremental learning of gaussian mixture models," in *Recent Advances in Artificial Intelligence Research and Development: Proceedings of the 20th International Conference of the Catalan Association for Artificial Intelligence, Deltebre, Terres de l'Ebre, Spain, October 25–27, 2017*. IOS Press, 2017, pp. 196–205.

[25] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977.

[26] J. M. Cortina, "Interaction, nonlinearity, and multicollinearity: Implications for multiple regression," *Journal of Management*, vol. 19, no. 4, pp. 915–922, 1993.

[27] M. Tian and Y. Su, "Multiple nonlinear regression model for predicting the optical performances of dielectric crossed compound parabolic concentrator (dccpc)," *Solar Energy*, vol. 159, pp. 212–225, 2018.

[28] P. Somarathna, B. Minasny, and B. P. Malone, "More data or a better model? figuring out what matters most for the spatial prediction of soil carbon," *Soil Science Society of America Journal*, 2017.

[29] A. G. Ivakhnenko, "The group method of data of handling ; a rival of the method of stochastic approximation," *Soviet Automatic Control*, vol. 13, pp. 43–55, 1968. [Online]. Available: https://ci.nii.ac.jp/naid/10004319713/en/

[30] ——, "Polynomial theory of complex systems," *IEEE transactions on Systems, Man, and Cybernetics*, no. 4, pp. 364–378, 1971.

[31] A. W. Moore, J. Schneider, and K. Deng, "E cient locally weighted polynomial regression predictions," in *Proceedings of the 1997 International Machine Learning Conference. Morgan Kaufmann*. Citeseer, 1997.

[32] V. Shaposhnyk, A. E. Villa, and T. Aksenova, "Advances in structural modeling robust to outliers in explanatory and response variables," in *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE, 2010, pp. 1–8.

[33] J. J. Buckley, T. Feuring, and Y. Hayashi, "Multivariate non-linear fuzzy regression," in *Fuzzy Systems Conference Proceedings, 1999. FUZZ-IEEE'99. 1999 IEEE International*, vol. 2. IEEE, 1999, pp. 714–718.

[34] A. F. Kuri-Morales, E. Aldana-Bobadilla, and I. López-Peña, "The best genetic algorithm II," in *Mexican International Conference on Artificial Intelligence*. Springer, 2013, pp. 16–29.

[35] E. Bishop, "A generalization of the stone-weierstrass theorem," *Pacific Journal of Mathematics*, vol. 11, no. 3, pp. 777–783, 1961.

[36] K. R. Koornwinder T, ong R and S. R, *NIST Handbook of Mathematical Functions*. Cambridge University Press, 2010.

[37] H. UGOWSKI, "Remarks on the ascent algorithm for the linear minimax problem," *COMPEL - The international journal for computation and mathematics in electrical and electronic engineering*, vol. 8, no. 3, pp. 181–184, 1989. [Online]. Available: https://doi.org/10.1108/eb010058

[38] J. H. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[39] G. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE transactions on neural networks*, vol. 5, no. 1, pp. 96–101, 1994.

[40] A. K. Morales and C. V. Quezada, "A universal eclectic genetic algorithm for constrained optimization," in *Proceedings of the 6th European congress on intelligent techniques and soft computing*, vol. 1, 1998, pp. 518–522.

[41] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[42] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework." *Journal of Multiple-Valued Logic & Soft Computing*, vol. 17, 2011.

[43] A. Kuri-Morales, "Transforming mixed data bases for machine learning: A case study," visited on October 16, 2018. [Online]. Available: http://www.micai.org/2018/

[44] S. McKinley and M. Levine, "Cubic spline interpolation," *College of the Redwoods*, vol. 45, no. 1, pp. 1049–1060, 1998.

[45] L. F. Shampine and R. C. Allen, *Numerical computing: an introduction*. Harcourt Brace College Publishers, 1973.

[46] C. Bloom, "Ppmz–high compression markov predictive coder," *http://www.cbloom.com/src/ppmz.html acce[accessed 18 January 2009]*, 1999.

[47] D. G. Hyams, "Curve expert," 2017. [Online]. Available: https://www.curveexpert.net/

[48] J. G. Saw, M. C. Yang, and T. C. Mo, "Chebyshev inequality with estimated mean and variance," *The American Statistician*, vol. 38, no. 2, pp. 130–132, 1984.

[49] J. Angstenberger, R. Weber, and W. Meier, "Dataengine: a software tool for intelligent data analysis," in *Proceedings of WESCON '94*, Sept 1994, pp. 348–350.