MODULAR NEURAL NETWORKS APPLIED TO PATTERN RECOGNITION TASKS

By
Bogdan George Gherman

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
UNIVERSITY OF KENT
CANTERBURY, UNITED KINGDOM
September 9, 2016

© Copyright by Bogdan George Gherman, 2016

UNIVERSITY OF KENT
SCHOOL OF
SCHOOL OF ENGINEERING AND DIGITAL ARTS

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "**Modular Neural Networks Applied to Pattern Recognition Tasks**" by **Bogdan George Gherman** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Dated: September 9, 2016

External Examiner: _____
Name

Research Supervisor: _____
Dr. Konstantinos Sirlantzis

Examing Committee: _____
Name

_____
Name

UNIVERSITY OF KENT

Date: **September 9, 2016**

Author: **Bogdan George Gherman**

Title: **Modular Neural Networks Applied to Pattern Recognition Tasks**

Department: **School of Engineering and Digital Arts**

Degree: **Ph.D.** Convocation: Year: **2016**

Permission is herewith granted to University of Kent to circulate and to have copied for non-commercial purposes, at its discretion, the above title upon the request of individuals or institutions.

Signature of Author

THE AUTHOR RESERVES OTHER PUBLICATION RIGHTS, AND NEITHER THE THESIS NOR EXTENSIVE EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT THE AUTHOR'S WRITTEN PERMISSION.

THE AUTHOR ATTESTS THAT PERMISSION HAS BEEN OBTAINED FOR THE USE OF ANY COPYRIGHTED MATERIAL APPEARING IN THIS THESIS (OTHER THAN BRIEF EXCERPTS REQUIRING ONLY PROPER ACKNOWLEDGEMENT IN SCHOLARLY WRITING) AND THAT ALL SUCH USE IS CLEARLY ACKNOWLEDGED.

To my Mother & Brother

Abstract

Pattern recognition has become an accessible tool in developing advanced adaptive products. The need for such products is not diminishing but on the contrary, requirements for systems that are more and more aware of their environmental circumstances are constantly growing. Feed-forward neural networks are used to learn patterns in their training data without the need to discover by hand the relationships present in the data.

However, the problem of estimating the required size of the neural network is still not solved. If we choose a neural network that is too small for a particular given task, the network is unable to "comprehend" the intricacies of the data. On the other hand if we choose a network size that is too big for the given task, we will observe that there are too many parameters to be tuned for the network, or we can fall in the "Curse of dimensionality" or even worse, the training algorithm can easily be trapped in local minima of the error surface.

Therefore, we choose to investigate possible ways to find the 'Goldilocks' size for a feed-forward neural network (which is just right in some sense), being given a training set.

Furthermore, we used a common paradigm used by the Roman Empire and employed on a wide scale in computer programming, which is the "Divide-et-Impera" approach, to divide a given dataset in multiple sub-datasets, solve the problem for each of the sub-dataset and fuse the results of all the sub-problems to form the result for the initial problem as a whole. To this effect we investigated modular neural networks and their performance.

Acknowledgements

The research for writing this thesis was funded by a departmental scholarship provided by the School of Engineering and Digital Arts at the University of Kent in Canterbury, United Kingdom.

Special thanks are in order for the following people that showed great support for my endevour: Dr. Konstantinos Sirlantzis, Dr. Farzin Deravi, Dr. Michael Fairhurst and Dr. Sanaul Hoque.

Special thanks are due to Dr. Gareth Howells, who supported my paper submission and attendance at the Forth International Conference on Emerging Security Technologies (EST-2013).

I am in debt to my former teachers and lecturers that have guided me on the path of science. I would like to mention only two of them: my general high school math teacher, Mr. Dušan Mitrić and my undergraduate mentor, Conf. Univ. Dr. Ing. Valentina E. Bălaş.

I am also thankful to Prof. Tiberiu Spircu from the Department of Medical Informatics and Biostatistics at "Carol Davila" University of Medicine and Pharmacy, Bucharest, for the great discussions we had about the mathematical formulations.

Some of my friends also need to be mentioned here for their encouragement and great discussions: Dr. Yiqing Liang, Anabela F. Soares, Michael Gillham, Richard K. Bonner, Dr. Philippos Asimakoupulos and many others.

But, the greatest acknowledgement has to go to my dear mother Doina-Elena and my brother Horia. Without whom I would be lost!

I would like to thank everyone who supported me while writing this thesis and making it possible.

THANK YOU ALL!

Bogdan George Gherman (BSc, MSc)

Table of Contents

List of Publications

1. (IN PRINT) Bogdan G. Gherman, Konstantinos Sirlantzis and Farzin Deravi, "Optimizing Neural Network Structures to Match Pattern Recognition Task Complexity", *Book Chapter Submitted to World Scientific Review on 3rd of February 2015*;

2. Bogdan G. Gherman and Konstantinos Sirlantzis, "Polynomial Order Prediction Using a Classifier Trained on Meta-Measurements", *4th International Conference on Emerging Security Technologies*, 9-11 September 2013, Cambridge United Kingdom, pp. 117-120, doi: 10.1109/EST.2013.26;

3. Bogdan G. Gherman and Konstantinos Sirlantzis, "Data Complexity Assessment for Constructing Modular Artificial Neural Networks", *School of Engineering and Digital Arts Conference*, January 2012;

4. Bogdan G. Gherman and Konstantinos Sirlantzis, "Polynomial Order Estimation to Determine the Number of Hidden Nodes in Feed-Forward Neural Networks", *School of Engineering and Digital Arts, Research Group Seminar*, November 2012;

5. Bogdan G. Gherman and Konstantinos Sirlantzis, "Investigation of the Decision Boundaries of Neural Networks", *School of Engineering and Digital Arts, Research Group Seminar*, May 2012.

List of Figures

List of Tables

Glossary

x, n, i, j, k, N	italics are used for scalar variables
$\mathbf{x}, \mathbf{y}, \mathbf{z}$	column vectors are in lowercase bold letters
$\mathbf{X}$	capital boldface letters are used for matrices
$\mathbb{R}^m$	the m dimensional set of real numbers
$\mathcal{D}_{type,index}$	a dataset of particular *type* and given *index*, which groups conceptually $\mathbf{X}$, Ω and the ground truth labels $\omega_k^{(i)}$ if they are available
Ω	the set of class labels, $\Omega = \{\omega_1, \ldots, \omega_c\}$
ω_k	one possible class label of a sample, where $\omega_k \in \Omega$
ω	vector of ground truth labels
$\omega_k^{(i)}$	the ground truth label of sample i, where $\omega_k^{(i)} \in \Omega$
$\mathbf{X}$	input dataset feature values as an $(n \times m)$ matrix
$\mathbf{x}$	generic input vector as an $(m \times 1)$ column vector
$\mathbf{x}^{(i)}$	input sample vector i as an $(m \times 1)$ column vector
$x_j^{(i)}$	scalar input sample value with index i and feature j where $\mathbf{x}^{(i)} = [x_1^{(i)}, x_2^{(i)}, \ldots, x_j^{(i)}]^T$ and $x_j^{(i)} \in \mathbb{R}$
$\mathbf{y}^{(i)}$	output vector corresponding to input i
$y_j^{(i)}$	the j^{th} component of the output vector corresponding to input i
n	total number of samples in $\mathbf{X}$ which is part of $\mathcal{D}_{type,index}$
d	number of features or dimensionality of the dataset $\mathcal{D}_{type,index}$
c	total number of classes in $\mathbf{X}$ which is part of $\mathcal{D}_{type,index}$, or the cardinality of the set Ω, $c = \lvert\Omega\rvert$
i, j, k	indices for enumerating samples, features and classes respectively
$\mathcal{I}(a, b)$	indicator function taking the value 1 if $a = b$ and 0 otherwise
$P(A)$	general notation for the probability of an event A

$P(A\|B)$	condional probability of event A conditioned by event B
$P(\omega_k)$	the prior probability of class ω_k to occur
$P(\omega_k\|\mathbf{x})$	the posterior probability that the true class is ω_k, given $\mathbf{x} \in \mathbb{R}^n$
$p(\mathbf{x}\|\omega_k)$	the class-conditional probability density function for $\mathbf{x}$ given ω_k
$\mathbf{\Sigma}$	covariance matrix (not to be confused with the summation operator which will have boundary conditions specified like this: $\sum_{i=1}^{n}$)
PCA	Principal Component Analysis

Chapter 1

Challenges in Designing Pattern Recognition Systems

INFORMATION is ubiquitous as the atmosphere we are breathing. The ancient Greek philosophers like Plato [58] and Aristotle [2] have realized this more than two thousand years ago, that we are perpetually surrounded by information, and what is even more interesting is, that they also formulated the fact that information should be grouped into categories. The process of learning is tightly related with grouping similar information so as to form recurring patterns and conversely, dispersing information is used to distinguish and discriminate information.

The early philosophers didn't stop at dreaming just about how to encode information but, they imagined how to construct machines with human-like abilities. These ideas were probably the early sparks that ignited the development of early calculating machines, *Automatons*, computers and which will probably lead to Generalized Artificial Intelligence.

An outstanding account of the history of Artificial Intelligence is to be found in Nils J. Nilsson's book, entitled: "The Quest for Artificial Intelligence" [55], where he shows the timeline of events that have led to the current state of Artificial Intelligence as we know it.

Information is around us, if we care to encode it either consciously or unconsciously, therefore there is a natural labelling of objects with pieces of information that requires two more processes namely, storing and retrieving these labels.

Conversely, recognizing patterns belonging to one or more categories is associated with identification and classification of newly sensed information, which is the main aim of pattern recognition.

The problem of concern for pattern recognition is to build a system which assigns newly acquired measurement examples to one of k categories or classes. This process

is accomplished by first modelling or learning the possible underlying characteristics of a set of known data that has already been categorised correctly into k classes. This type of learning is called supervised learning and the process involved in learning is called training a classifier. A classifier is the realization of a learning algorithm. An intricate classifier may employ several learning algorithms once, such is the case in ensemble learning, but the relationship is of the one-to-many type.

Dietrich [15] makes a clear distinction between a classifier and a learning algorithm which is very important, in the context of evaluating the performance of classifiers which actually is being done, since the assessment of classifiers implies using a learning algorithm. This distinction between classifiers and learning algorithms is analogous to the distinction between an object and its class in Object Oriented Programming. Classifiers are manifestations or incarnations of learning algorithms. Hence, the two notions are closely related.

The overall goal of pattern recognition is to build classifiers with the lowest possible error rates that work well for a wide variety of input data.

This poses a problem straight away, for how can we compare the performance of one learning algorithm with that of another learning algorithm?

The answer to this question is elusive. Since, there is no feasible way of acquiring all of the values of the population from where the data arises. We can only extract samples from that population, which in sometimes cases will have only small number of specimens.

Nonetheless, there are methods of approximating the error a classifier makes on the whole population. To this effect, there are published works that describe the methodology for comparing the performance of two classifiers or learning algorithms one of which is Dietterich's paper [15], which states that none of the statistical tests described in the paper, some of them widely used and evaluated in the same paper, can answer the question whether one algorithm will produce more accurate classifiers when compared against another learning algorithm. Kuncheva [41] re-iterates the same findings in her book "Combining Pattern Classifiers".

There are multiple reasons for the unreliability of the statistical tests to compare the performance difference between two classifiers described by [15],[41], namely:

1. all of the statistical tests require using holdout or re-sampling methods, which reduces the number of available examples in the training set to be smaller than the total number of examples that are labelled and available at the time of assessment, which in turn, is much smaller than all the possible input combinations of the classifier;

2. statistical tests require a number of independence assumptions and some normality assumptions which are often violated;

3. statistical tests that use holdout methods for dividing the existing data into training and testing sets do not measure the variation resulting from the choice of training/testing sets nor do they measure the internal randomness of the algorithm;

4. the choice of the training and testing sets has a major impact;

5. mislabelled ground truth data can be present in the testing sets and this contamination introduces an error constant. The classifier cannot have an error lower than this constant.

As a consequence of these problems the performance comparisons between two classifiers has to be viewed as approximate, heuristic tests, rather than rigorously correct statistical methods, that are not universally correct, regardless of what type of data was used to evaluate the two classifiers.

Nonetheless, some classifiers are better suited than others for one particular arrangement of the input data. For example, linear or quadratic classifiers are very well suited for data that has normally distributed sample values, and more importantly the shape of the decision surface between the k-classes has linear or quadradic shape. Namely, an m-dimensional hyperplane of the linear classifier or a quadratic surface in m-dimensional space of the quadratic classifier (where m is the dimensionality of the data), would be well suited. However, these two examples of classifiers will be hopelessly incapable of efficiently modelling the intricacies of input data that has a spatial organization like the dataset shown in Figure 1.1.

Neural Networks (NNs) have the property that they are Universal Approximators. That is, they can approximate any given function with arbitrary precision given they have a large enough number of neurons at their disposal and most probably an equally large number of training samples and training time. Nevertheless, if NNs do not posses enough neurons, they will not be able to learn the problem that is presented to them.

This is the case shown in Figure 1.1, where we have a given pattern recognition problem, the so called Banana dataset, which contains two sets of unlabelled points that come from two separate distributions that graphically resemble the fruit they were named after. The problem is to estimate or guess from which of the two distribution each point is coming from. For this simple experiment, we trained NNs with varying number of neurons in the, so called, hidden layer. We trained Neural Networks with 1, 2, 3 and 5 neurons and plotted their decision boundary in Figure

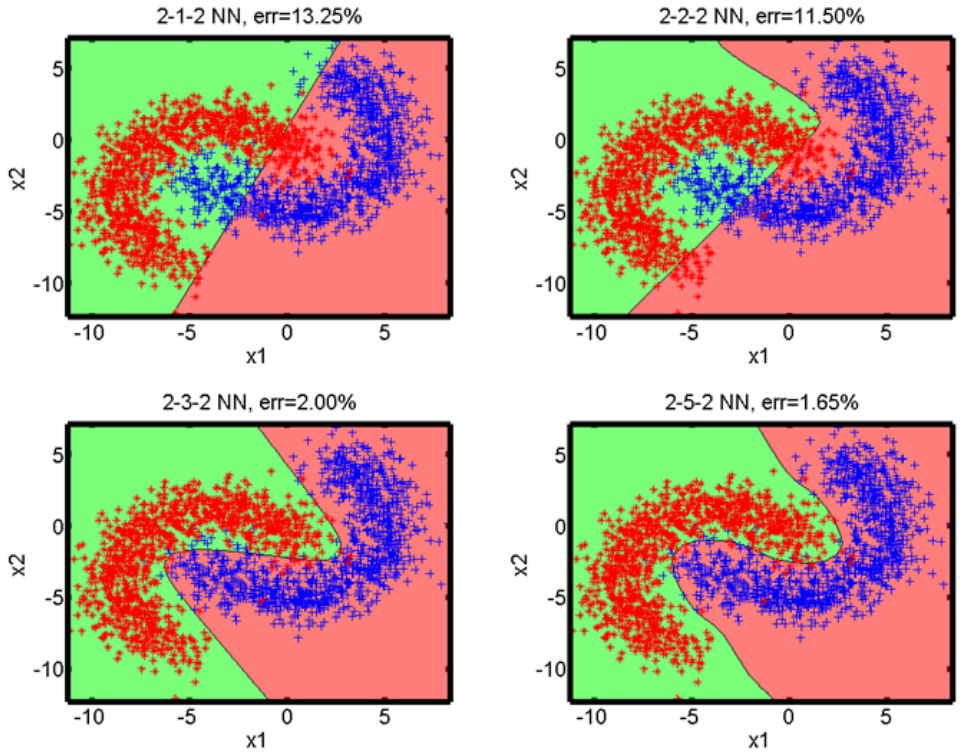


Figure 1.1: Decision boundary produced by four NNs with 1,2,3 and 5 hidden neurons.

1.1. The top two decision boundaries show that the NNs could not learn the problem at hand. They will always generate a decision boundary similar to the ones shown in Figure 1.1, this is a fact that we will prove in Chapter 6 when we talk about weight adaptation.

It remarkable to see that even with 3 neurons (bottom left graph of Figure 1.1) the drop in classification error rate has started to flatten out. The selection of 3 neurons in the hidden layer of the NN is the crucial number of neurons needed to learn the Banana dataset. This is a characteristic of the dataset or the layout of the points which we consider to be correct.

Increasing the number of hidden neurons shows diminishing returns, even though the error rate will continue to drop, the training time will increase with the addition of extra neurons and the network will have to much variability that in extreme cases it will be able to learn each data point. Which is categorically undesirable because it will not generalize well unseen new data-points, which is one of the main targets of pattern classifiers.

Therefore, it is imperative to be able to tailor the number of hidden neurons that make up the Neural Network to the pattern recognition problem that is to be solved,

this was the main idea that inspired this present work.

1.1 Research Problem and Motivation

The present thesis will investigate the issues related in designing a Modular Artificial Neural Network (MANN) for pattern recognition. The goal that we persuade is two fold.

Firstly, we desire to tackle the problem of selecting the architecture for a neural network, by seeking the answer to the following question:

> *"What is the number of hidden nodes a Neural Network architecture requires in order to classify a given dataset (in some sense) optimally?"*

Secondly, we address the problem of module selection in a greater scenario of building a modular classifier based on individual Neural Networks, whose architecture was determined by the process that we stated as our first goal.

Let us elaborate our first goal a bit more in the following paragraphs.

The question of *"how many hidden nodes are required in a Neural Network to classify a given dataset optimally?"*, does not have a straight-forward answer. Several issues arise straight away from this single question: What are the hidden nodes? What is the architecture of Neural Network? but more importantly: What is optimal classification? We shall try to provide an answer to some of these questions.

In order to try to tackle the question of *"What is optimal classification?"* we first have to define a measure of optimality. There are several choices, we have decided to use the classification accuracy for this purpose. However, we will also investigate the Mean Squared Error (MSE, equation (2.2)) between the outputs of the Neural Network and the desired target outputs of the Neural Network and the gradient of the improvement at each training step or epoch.

We are going to use Feed-Forward Neural Networks that have all the nodes fully connected to each other.

Protagonists:

- Neural Network
- Decision Boundary
- The link between the decision boundary and the architecture of the Neural Network

In Figure 1.2 you can find the block diagram of flow chart of the elements that will be presented in more detail in the following chapters. In the above mentioned

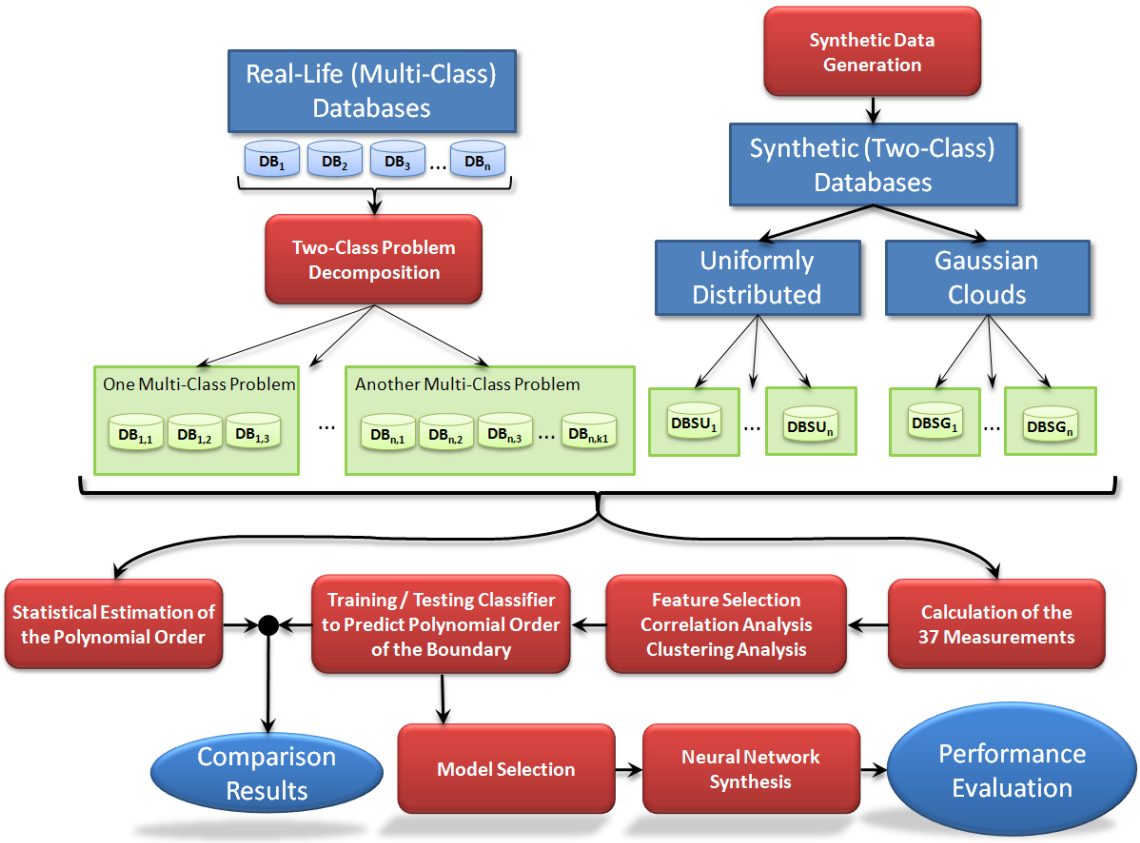


Figure 1.2: Overall system view of the work presented in this thesis.

figure, the rectangles with rounded corners stand for a conceptional task that was investigated, whereas the rectangles with sharp corners are denoting information in the form of collection of databases and datasets.

1.2 Aims and Objectives

Our aim in this present research is to develop a methodology for suggesting the number of hidden nodes required for a Feed-Forward Back-Propagation Neural Network (FFBPNN) to have in its hidden layer in order to have the capacity to learn the given pattern recognition problem that it is being trained on.

With this aim in mind we are proposing several immediate objectives in order to reach our goal:

1. Firstly, obtain pattern recognition datasets that have an arrangement of the sample points in recognizable prototypes. We focus on data arrangements which have polynomial functions as the decision boundary formed between pairs of classes.

2. Relate the complexity of the datasets to the order of the polynomial that can be fitted onto the decision boundary present between the pairs of classes present in the dataset.

3. Implement algorithms to synthetically generate datasets that have a decision boundary of a given polynomial order, ranging from 1 to 10.

4. Identify and apply metrics that can be calculated from the training set of a pattern recognition problem, that can estimate the complexity inherent in the dataset.

5. Evaluate the prediction power of the complexity metrics.

6. Decompose any pattern recognition problem that has c number of classes into a p number of 2-class problems, using either a "1-versus-ALL" or "1-versus-1" splitting method. Where p is the given by the binomial coefficient:

$$p = \binom{c}{2} = \frac{c!}{2(c-2)!}$$

and also,
p is the number of 2-class problems, and
c is the initial number of classes in the dataset.

7. Evaluate the classification performance of Single Neural Networks that have the architecture parameter suggested by the complexity measurement. This evaluation will be carried out on synthetic datasets, while comparing the performance of the Neural Networks with other baseline approaches.

8. Improve the performance of Single Neural Networks by choosing the initial weight parameters of the Neural Networks to approximate the polynomial decision boundary present in the dataset.

9. Evaluate the classification performance of Modular Neural Networks that have the architecture parameter suggested by the complexity measurement for each module independently. This evaluation shall be conducted on realistic datasets. Also, the evaluation will have to include comparisons to control experiments and baseline performance experiments.

1.3 Thesis Outline

The thesis is structured in the following way, after this current introductory chapter we are going to present in Chapter 2 the background to Neural Networks that will include a brief historical overview, literature review and implementation details of Neural Networks and Modular Neural Networks.

The third chapter will present the Meta-Measurement metrics devised and calculated to describe the complexity required to classify a given dataset.

In Chapter 4 we will describe the data that was used in experiments throughout the rest of the thesis.

The fifth chapter will present the experimental setup and results from employing the complexity measures described in Chapter 3 for approximating decision boundaries with several different methods that are being compared.

The sixth chapter will show how to select initial parameters for Neural Network training based on approximating the rough decision boundary.

The seventh chapter will present the experimental setup and results achieved by creating Modular Neural Networks trained on synthetic and realistic datasets.

Finally, in Chapter 8 we shall present the final conclusions resulting from our research and a few future development suggestions.

Chapter 2

Neural Networks

The following chapter will give a short introduction to Feed-Forward Artificial Neural Networks (ANNs or simply NNs) and modular realization of such networks (MANNs or MNNs). After a short historical review, we shall discuss the following issues related to Artificial Neural Networks:

- the building blocks of Neural Networks: the neurons;
- the connectionist topology of the neurons in a layered organization;
- the back-propagation training algorithm of the NNs.
- and finally the structure of Modular Neural Networks (MNNs).

2.1 Historical Background

The roots of the development of neural networks can be traced back to ideas from philosophy, psychology and neurology, when people wanted to build autonomous entities long before the present day computers, which, opened the gates to unimagined possibilities. But only the advent of recent advances and emergence of the following domains contributed to the development of neural networks according to Mitchell [51]: philosophy, psychology, neurobiology, Bayesian theory, computational complexity theory, information technology, control theory, cybernetics, information theory and last but not least: statistics.

Artificial Neural Networks (ANNs) were created by drawing inspiration from biological brains in order to mathematically model their behaviour and produce massively parallel computational schemes.

The first formulation of basic Artificial Neural Network Principles were presented by McCulloch and Pits in 1943 [48], where they assumed the neuron is a binary element. Later, Donald Hebb introduced the *Hebbian Learning Law* in his influential

book *The Organization of Behaviour* [34] where he stated the following: "repeated activation of one neuron by another, across a particular synapse, increases its conductance", which is not necessarily directly used in ANN designs however, it is employed by Kohonen Self Organizing Maps, Cognitron, NeoCognitron and Large Scale Memory Storage and Rertrieval (LAMSTAR) Networks.

The earliest ANN, *The Perceptron*, was proposed by the psychologist Frank Rosenblatt that appeared in the Psychological Review of 1958 [66], introducing a learning method for the McCulloch and Pitss neuron model. Frank Rosenblatt's widely influential contribution to the field of artificial intelligence was the introduction of the perceptron, a "hypothetical nervous system" designed to mimic some of the organizational systems used in the brain. Rosenblatt and his followers called their approach *connectionist* to emphasize the importance in learning of the creation and modification of connections between neurons.

Widrow and Hoff [78] introduced in 1960 the *"Adaline"* (ADaptive LInear NEuron), a single neuron trained by gradient descent rule to minimize the squared error.

In 1969 Minsky and Papert demonstrated the limits of the simple perceptrons, proving they are not computationally universal, which resulted in a drastic reduction in research interest in neural networks. However, this hurdle was overcome by the discovery of the backpropagation training of Multi Layered Perceptrons (MLPs) by Rumelhart et al. in 1986 [67] which has been since proven that multi-layered perceptrons with non-linear activation functions are indeed universal approximators.

Simon Haykin [33] provides the following definition for a Neural Network, which he adapted from Aleksander and Morton in 1990 [1]:

> *"A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experimental knowledge and making it available for use. It resembles the brain in two respects:*
>
> 1. *Knowledge is acquired by the network from its environment through a learning process.*
> 2. *Inter-neuron connection strengths, known as synaptic weights, are used to store the acquired knowledge."*

The literature covering neural networks is vast and growing. Amongst the large number of textbooks and treaties we can suggest the works of Bishop[8], Mitchell [51], Haykin [33].

Neural networks have applications in: business(financial forecasting, insurance policy evaluation), aerospace, automotive (manufacturing control), defence, healthcare and others.

Neural networks are part of the broad field of *Computational Intelligence* which was first originated in 1990 by the IEEE Neural Networks Council but was first stated by around the years 1993-1994.

Marks in 1993 [47] made a clear distinction between Computational Intelligence and Artificial Intelligence, although both seek similar goals, however Bezdek [7] argues that Computational Intelligence is a subset of Artificial Intelligence.

According to Bezdek [7], a system is called Computationally Intelligent if all the following are true:

1. it deals with the low level representation of the data (e.g. numeric representation),
2. has a pattern recognition component which does not employ knowledge in the Artificial Intelligence sense but, exhibits computational adaptivity, fault tolerance, and finally;
3. it approaches the speed and accuracy of human performance.

The IEEE Computational Intelligence Society (formerly known as IEEE Neural Networks Council) defines its subject of interest as Neural Networks, Fuzzy Systems and Evolutionary Algorithms [17], and is one of the biggest publishers of journal papers on the subject.

Engelbrecht [21] considers the following six basic approaches on how to achieve Computational Intelligence, through an individual or a combination of the following:

1. Fuzzy Logic;
2. Neural Networks;
3. Evolutionary Computing;
4. Learning Theory;
5. Probabilistic methods;
6. Swarm Intelligence.

obviously Neural Networks are among them and they are the main focus of this thesis.

The construction of a Computational Intelligent system has the purpose of modelling a system or a process which is not tractable to mathematical or traditional modelling techniques because:

1. the processes are too complex to represent mathematically;

2. the models are difficult and/or expensive to evaluate;

3. there are uncertainties in the process' operation;

4. the process is non-linear, distributed, incomplete and stochastic.

Nils J. Nilsson [55] summarized the field of Artificial Intelligence by dividing the ideas and achievements of AI research into:

1. Complete AI Systems;

2. Architectures;

3. Processes; and

4. Representations.

Since we are concerned with Neural Networks, which fall into the Computational Intelligence field according to [7] while according to Nilsson, NNs would fall into the "Architectures" subset of Artificial Intelligence.

The most important, inherent, properties of Artificial Neural Networks, which makes them an attractive tool for computational intelligence and arguably artificial intelligence tasks, according to [33] are:

- generalization;
- graceful degradation;
- adaptation and learning;
- inherent parallelism.

Neural networks are used for the following tasks:

- classification;
- function estimation;
- regression tasks.

The variety of fields of application of neural networks, in their various forms, is limited only by the ability to measure, quantize and digitize the desired inputs that will be applied to the neural network. However neural networks do have some limitations and drawbacks. Among the drawbacks of neural networks we can mention: training data over-fitting tendency, entrapment in local minima of the error surface and long training time. The main goal of research in the field of artificial

neural networks is to understand and emulate the working principles of biological neural systems [5]. Biological neural systems consist of billions of individual biological neurons, interconnected via tens of thousands of synaptic weights to other neurons. Recent advances in neurobiological sciences have given more insight into the structure and the workings of the brain which inspired researches in the field of Machine Learning, Computational and Artificial Intelligence.

However, it is surprising to find that Machine Learning does not only borrow ideas and solutions from the biological world but also helps understand how biological brains function on a neuro-physiological level by providing a theoretical framework for how animals learn. This was achieved through the research and development of reinforcement learning conducted by Donahue & Seo [16] and earlier work of [68].

2.2 Number of Published Articles

Neural networks have enjoyed a great amount of attention only in the past 20 years, due to their wide applicability and desirable features. Figure 2.1 shows the increasing trend in published papers related to the application and advances in neural networks according to ISI Web of Knowledge (WOK) [64]. The number of articles has been queried in 2009 and revised in 2014 to show the difference which occurred within roughly 5 years.

The graph shown in Figure 2.1, shows the bars of the number of articles corresponding to the query done 2009 and 2014 in three annotations:

1. trend line of the number of articles reported by WOK to be published in 2009 (dashed blue line);

2. trend line of the number of articles reported by WOK to be published in 2014 (dashed red line);

3. ratio of the number of articles reported by WOK to be published in 2014 over the number of articles reported to be published in 2009 (dashed and full green line) with the scale shown on the left-hand side of the figure.

The trend lines for the number of articles reported by WOK to be published in 2009 and 2014 have been analysed by fitting a linear function on the two sets of reported number of published articles from the year 1989 onwards, since before 1989 the number of articles returned by the two queries is small. In the case of the year 1988 there were reported by WOK in 2014 to have been published only 556 articles related to neural networks respectively the 2009 query of the WOK database returned only 101 articles. Before the year 1988 the number of articles

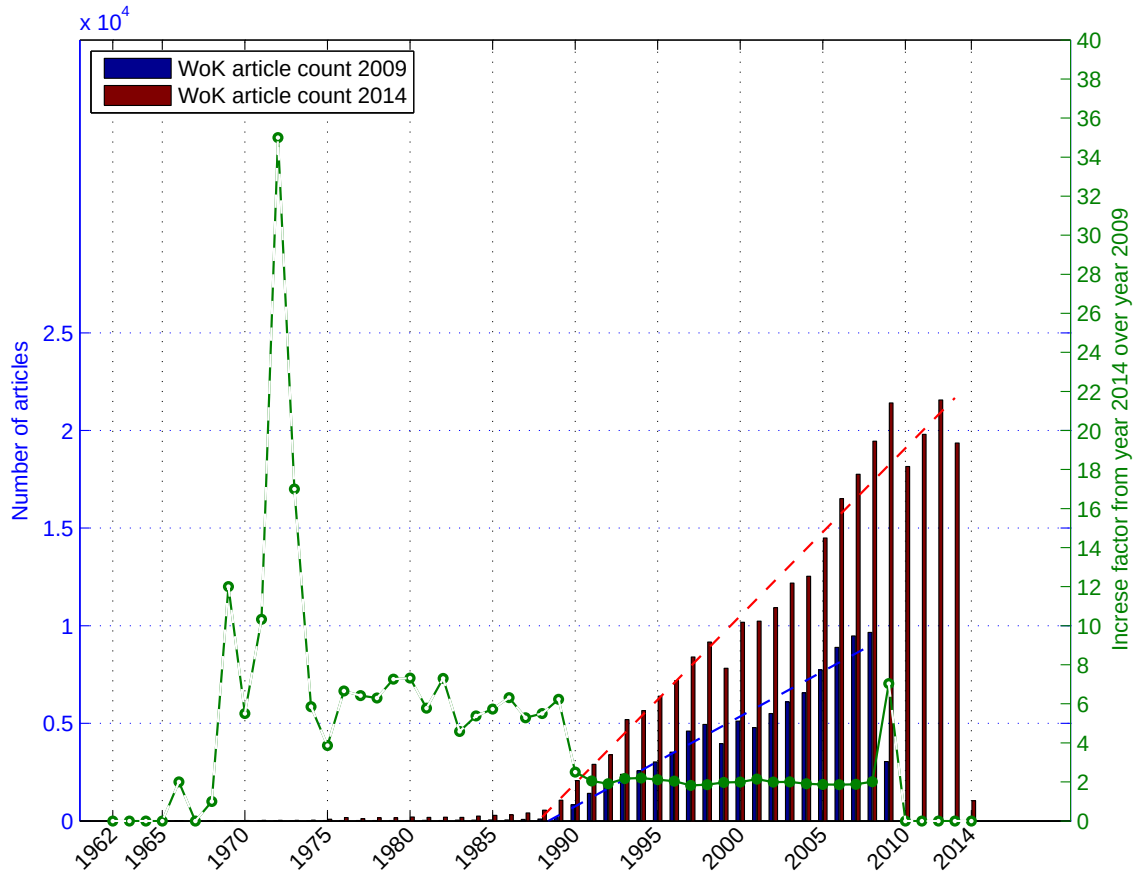


Figure 2.1: Number of publications per year related to neural networks based on ISI Web of Knowledge in 2009 and 2014 [64] along with the increase factor from one year to the other.

indexed by WOK are steadily decreasing the further we look back in time. We are not implying anything about the impact or quality of any of the articles, merely doing a quantitative assessment.

The first trend line, shown with the dashed blue line, in Figure 2.1, which corresponds to the number of articles indexed by the WOK in 2014, has the following parameters:

$$y_{2009}(x - 1988) = 460.82x - 633.87$$

The second trend line, shown with the dashed red line, in Figure 2.1, which corresponds to the number of articles indexed by the WOK in 2014, has the following parameters:

$$y_{2014}(x - 1988) = 858.66x - 655.67$$

From the coefficients of the two trend-lines we can see that the average growth in the indexed number of articles relating to neural networks queried in 2009 was about 460 articles per year in the period 1988 to 2009 and respectively in the query

done in 2014 the number of articles per year was 858, an almost two fold increase in the number of articles searchable in the Web Of Knowledge database.

This two fold increase is also evident when we calculate the ratio between the number of articles returned from the query done in 2014 over the number of articles returned by the query done in 2009. After removing the values that produced division by zeros, we have plotted this ratio also in Figure 2.1, with the green colour and using the scale on the right side of the graph. Values are plotted with a dashed green line for the time period between the years 1962 and 1990, respectively for the period between the years 2009 and 2014 where the ratio is meaningless either because there are very few number of articles or there is no information in the query done in 2009 about publication done between 2009 and 2014.

From this analysis we can see that the development and application of neural networks has began with an explosion of publication around the year 1990 and it is a field that is benefiting from an increased interest, by a growing number of publications indexed by the Web Of Knowledge indexing service [64].

An important source of bibliographic and citation information was the British Library's searchable catalogue, which includes conference proceedings, journals and books. Among the reported 56 million items the British Library has on record, there were a total of hits 111,587 for the term "neural networks".

We conducted a search to find the most influential journals relating to neural networks. The result of this search is shown in Table 2.1.

Table 2.1: Journals with most publications on neural networks according to British Library

Name of journal	Number of publications
Lecture Notes in Computer Science Journal on Data Semantics	16,080
IEEE International Conference on Neural Networks	4,198
Proceedings of SPIE, The International Society for Optical Engineering	3,747
Neural Networks	3,085
Intelligent Engineering Systems through Artificial Neural Networks	2,158
Neurocomputing: An International Journal	1,705
Proceedings of the International Joint Conference on Neural Networks	1,674
World Congress on Neural Networks	1,246
Neural Networks for Signal Processing	772
International Conference on Artificial Neural Networks (ICANN)	712
IEE Conference Publication	611
European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN)	558
Proceedings of the Australian Conference on Neural Networks	423
Cellular Neural Networks and their Applications (CNNA)	398
Communications in Computer and Information Science	388
International Conference on Fuzzy Logic and Neural Networks	267
Brazilian Symposium on Neural Networks	176
International Symposium on Artificial Neural Networks	122
Proceedings of the International Workshop on Applications of Neural Networks to Telecommunications	80
TOTAL:	**38,400**

2.3 Artificial Neural Networks for Pattern Classification Applications

Neural Networks (NNs) are part of the machine learning scientific discipline, which is concerned with the design and development of algorithms that allow computers to learn based on data, such as from sensor data or off-line databases. They are suitable for performing the following types of general tasks: pattern classification, regression, and clustering [33].

There are several paradigms of learning which apply to pattern recognition [33],[18]:

1. supervised learning;
2. unsupervised learning;
3. reinforcement learning and active learning.

We are going to employ in our research the first paradigm of learning, namely the supervised learning paradigm.

The goal of NNs is not to learn an exact representation of the training data itself, but rather build a model of the process that generates the data. This problems is said to be well posed if an input always generates a unique output and the mapping is continuous.

Supervised learning is an ill posed problem, given the training or input examples that are a subset of the Input-Output relationship, an approximate mapping is needed to be estimated. However, the input data might be noisy, imprecise and it might be insufficient to uniquely construct the mapping.

Regularization techniques can transform an ill-posed problem into a well posed one, in order to stabilize the solution by adding some auxiliary, non-negative functional of constrains [59], [73].

In this research we are concerned with solving pattern classification tasks using Neural Networks (NNs). The NNs are adapting their internal parameters (network connection weights) to model their output according to the input from the environment they are taking the information from. This process is generally called "learning" of the NN.

The parameters can be learnt in three different paradigms:

1. Using prescribed outputs for some of the inputs (desired outputs) which are evaluated as correct or incorrect by a third party, this is called Supervised Learning, and the input/output data is said to be labelled;

2. Using a task independent measure of quality that will require the network to behave in a self organizing manner, which is called Unsupervised Learning;

3. The Input to Output mapping of the network is performed continuously in order to improve on scalar performance measure, this is called Reinforcement Learning. The data in the latter two paradigms is said to be unlabeled and in contrast with the labelled data, this kind of data is readily available and inexpensive to obtain.

The manner in which the neurons are organized is closely related with the learning algorithm used to train the network. According to Mehrotra et al. [49], Neural Networks can be categorized according to their topology, i.e. the way neurons are organized as the following types:

1. Fully connected Networks

2. Single Layer Networks feed-forward Networks

3. Multilayer feed-forward Networks

4. Acyclic Networks

5. Modular Neural Networks

We will need to add that feedback links can exist between the nodes, so that the outputs of a particular node are connected to nodes from which the particular node receives a connection. According to Haykin [33] these types of networks are called Recurrent Networks. In the next two sections we will deal with two types of NNs, namely Feed-forward and modular networks, which are the main types of networks in our research.

2.4 Neurons as Processing Units

The Neural Network consists of processing units called nodes or neurons, that take the summation of the inputs and map it through a non-linear function to produce an output. An example of a single neuron can be seen in Figure 2.2, where a neuron with three inbound inputs x_1, x_2, x_3 are pictured, along with the biasing connection w_1 and also the summation output, which is then passed through the *tanh* function is labelled as y_1. The neurons or nodes (as they are also called) are arranged in layers and connect the inputs of the Neural Network to the outputs of the network, see Figure 2.4 for an example of a Neural Network with 2 inputs,

3 hidden nodes and 2 outputs. The neural network presented in Figure 2.4 has 3 layers.

The non-linear activation function in Figure 2.4 for each of the neurons is the hyperbolic tangent, this activation function was also in the rest of our work. Examples of other common activation functions to be employed by the neurons can be seen in Figure 2.3.

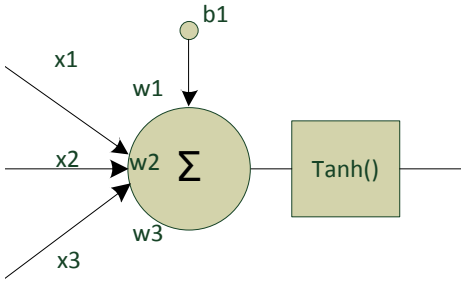


Figure 2.2: Single neuron with the summation of the inputs, the non linear activation functions ($tanh$) and the resulting output.

A hidden node or neuron is the processing unit in a Neural Network that is located between the input layer and output layer of a Neural Network.

$$y_k = tanh\left(\sum_{j=0}^{n_k} w_{j,k} \cdot x_k\right) \tag{2.1}$$

where

y_k = the output of k^{th} layer node

$w_{j,k}$ = is the weight associated with input k into layer j

x_k = the input into the k^{th} layer

n_k = the number of nodes in the k^{th} layer

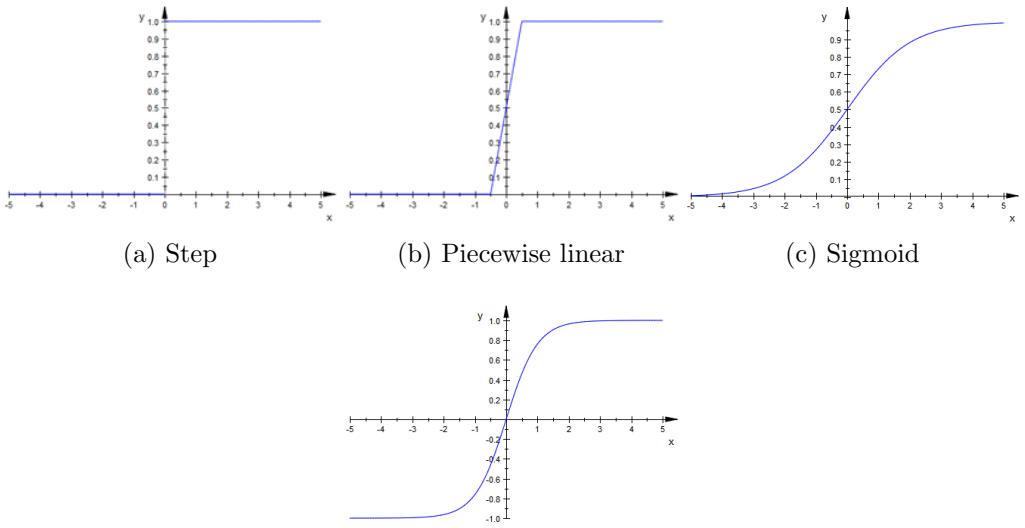
$$MSE = \frac{1}{N}\sum_{i=0}^{N}(t_i - y_i)^2 \tag{2.2}$$

where:

N = the number of training samples

t_i = the i^{th} desired output of the network

y_i = the i^{th} output produced by the network at a given training state



(a) Step (b) Piecewise linear (c) Sigmoid

(d) Tanh

Figure 2.3: Examples of common non-linear transfer functions.

2.5 Feedforward Backpropagation Networks

Backpropagation, or propagation of error, is a common method of teaching artificial neural networks how to perform a given task. It was first described by Paul Werbos in 1974, but it wasn't until 1986, through the work of David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams, that it gained recognition, and it led to revival of the research of artificial neural networks.

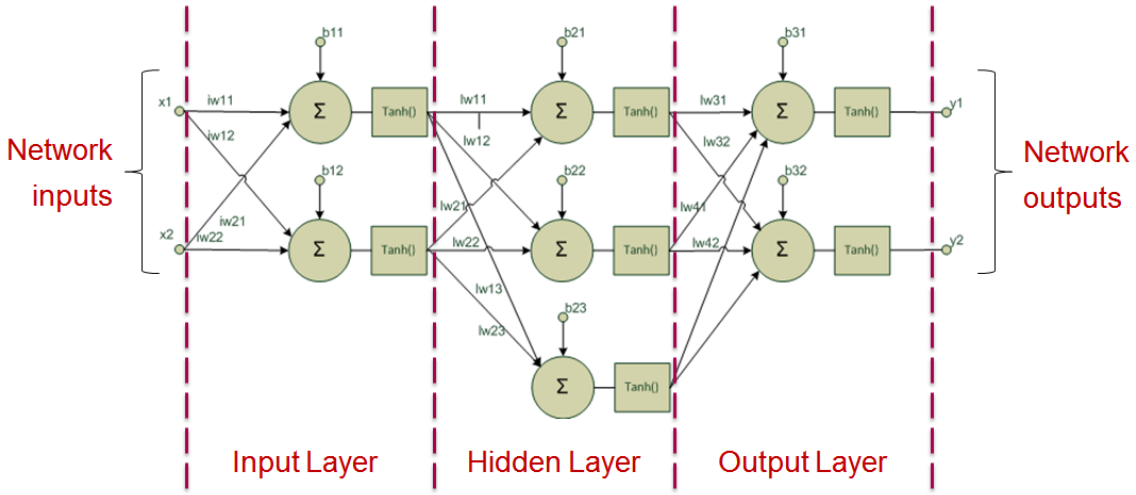


Figure 2.4: Example of a multi layered, fully connected, Feed-Forward Neural Network architecture with 3 layers, 2 inputs, 3 hidden nodes and 2 outputs, generally called a 2-3-2 network.

According to Simon Haykin [33], the back-propagation algorithm has emerged as

the workhorse for the design of feedforward networks known as multilayer perceptrons (MLP).

As shown in Figure 2.4, a multilayer perceptron has an input layer of source nodes and an output layer of neurons (i.e., computation nodes); these two layers connect the network to the outside world. In addition to these two layers, the multilayer perceptron usually has one or more layers of hidden neurons, which are so called because these neurons are not directly accessible. The hidden neurons extract important features contained in the input data.

The back-propagation learning algorithm is simple to implement and somewhat computationally efficient in that its complexity is linear in the connection weights of the network. However, a major limitation of the algorithm is that it does not always converge or it can be extremely slow, particularly when we have to deal with a difficult learning task that requires the use of a large network.

Despite these shortcomings of the MLP architecture and training algorithms, these networks are used in solving problems where the dimensionality is small and the amount of training data is sufficient so that the "Curse of dimensionality" is avoided, as for example in the recent work published by: Foody [24] in 1995, Blamire [9] in 1996 and more recently in 2003 the work of Pal and Mather [56]. The performance of the MLP is dependent on the quality of the training data, a fact that was neglected a bit by previous studies but Taskin Kavzoglu [39] in 2009 has improved the classification accuracy by 2-3 percent by eliminating outliers, using training set refinement, supports the premise that MLPs are still viable to solve current problems.

For a good generalization the number of examples in the training set N, has to be several times larger than the neural network's capacity [79]:

$$N \gg \frac{N_w}{N_y}$$

where N_w is the total number of weights or free parameters and N_y is the total number of output units.

2.6 Training Algorithm

Neural Networks (NN) have two distinct modes of operation:

1. Learning mode, parameter estimation or training of the NN;

2. Running mode or testing of the NN.

2.7 Modular Neural Networks

The best description of the Modular Neural Networks (MNN) has been given by Ronco and Gawthrop in their technical report from 1995 [65], where they define the MNN in comparison with "global" back-propagation and clustering neural networks. Here they state that a modular network is the most likely to combine the desirable features of the two aforementioned classes of networks. It is pointed out that a MNN has to have its modules assigned problem specific sub tasks, and not just any sub-task, resulting from an arbitrary decomposition scheme that might not have any physical meaning. The reason for dividing the problem, that we want to solve, into sub tasks that are physically meaningful, is desirable from at least two stand points:

1. the number of variable parameters is reduced to a number large enough to provide a good solution;

2. the network becomes tractable, so that the internal workings of the network have some meaning attached and are not just black boxes that cannot be inspected easily.

Further in this report and in the survey of Auda and Kamel [4], we find the steps needed to be accomplished by the designer of a MNN:

1. Decomposition of the main task into sub-tasks;

2. Organization of the modular architecture;

3. Communication between the modules and decision fusion.

Modular neural networks possess the conceptual means to overcome some of the shortcomings of back-propagation multi-layered networks (BP-NNs) and the benefits of clustering networks. Here are some of the problems BP-NNs face:

1. Flat area or local minima in the error surface, leading to slow convergence to a solution if it is not trapped in local minima when it will fail to converge at all.

2. Interference in the input data, i.e. the so called "spatial crosstalk" that is best seen in the "what and where" type of problems where a BPNN has problems retaining both of the aspects of the problem in a single architecture. Imagine breaking down the task into "what" and "where", now it is subjectively obvious that the resulting system will perform better than the system that tries to solve the whole problem by itself. This way of task decomposition is often denoted

in the literature as "Divide and conquer" (from the Latin: "Divide et Impera" which was a very successful way of the Roman Empire to triumph over its enemies). It must be pointed out that the term "Divide and conquer" refers to any task decomposition and there exists a learning rule in modular networks with the same name described by Fu, Lee and Pao [27], that splits the training data in two hopefully easier to learn regions with very desirable outcome.

3. Closely related to the previous problem of BP-NNs is the problem of "temporal crosstalk", where the network is trained to do one task and afterwards it is trained to do another task. Usually the network will tend to forget the first task it has learned as it learns the second one.

The idea of using multiple modules or committees to realize a complex task can be traced back to the published work of Nilsson [54] in 1965, where he considered a network having a layer of elementary perceptrons followed by a vote taking perceptron in the second layer. This approach is based on the same common engineering principle "divide and conquer" stated previously. The combination of experts is said to constitute a committee machine, these machines can be classified in two major categories [33]:

1. Having Static Structure, where the responses of several experts are combined by means that do not involve the input signal or pattern. Major approaches in this class are: ensemble averaging and boosting of weak learning algorithms (e.g. Adaboost);

2. Having a Dynamic Structure, where several modular networks are used to learn the whole input space and the output of each module is mediated by an integrating unit to produce the final output. Major architectures in this class are: the Mixture of Experts (ME), gated experts and hierarchical mixture of experts.

The previous research on MNNs is assessed in a very good and concise manner in the referenced papers [65] [4] and the references therein. Some problems still remain to be solved so the modular networks can be efficiently implemented for various applications, namely:

1. How to split the input space properly, such that the decomposition can be beneficial to both learning and generalization;

2. How to decide the proper number of experts in a committee machine for a particular task.

However, the problem of task decomposition was studied in the paper produced by Lu and Ito [44], where they consider that task decomposition can be roughly divided into three classes as follows:

1. Explicit Decomposition, before learning the problem is divided into a set of sub-problems by the designer of the system using domain and a priori knowledge about the problem. The difficulty lies with the fact that a large amount of prior knowledge is required and also this knowledge has to be properly translated to the system;

2. Class Decomposition, again, before learning the problem is broken down into a set of sub-problems so that a problem having K classes is divided into K, two class problems; An example is presented in reference [44] where a significant gain in performance is achieved compared with classical global neural networks;

3. Automatic Decomposition, where a problem is decomposed into a set of sub-problems by the process of learning. The former two methods are more efficient because the task is decomposed before the learning, but this latter method is more general since it does not require any prior knowledge. Most of the automatic methods fall into this category, for instance: the mixture of experts [52] and the multi-sieving network.

The multi classifier system presented here [52] describes the usage of a MNN to recognize shapes in an industrial robotic vision project. Another example of the classification discriminatory power of MNNs is presented in [60] where the results suggest that MNNs achieved comparable to Support Vector Machines while the proposed method was slightly surpassed by Exponential Radial Basis Function kernels, the method did prove to be better than the SVM using Gaussian radial basis function.

2.8 Chapter Conclusions

In this chapter we have given an introduction to Neural Networks and Modular Neural Networks along with a review of the literature on the topic.

We have found that Neural Network research is still very active and productive, thus, our research is novel and has a valid place within the field.

Chapter 3

Data Complexity

This chapter describes the problem related to the definition of data complexity for pattern recognition datasets and describes the method that we employed to tackle this problem by calculating additional measurements on the input training dataset in order to obtain a uniform complexity measure of the dataset. The order of the polynomial which is associated to the complexity of the dataset will be used later on to suggest the number of hidden nodes to be used in the architecture of the neural networks or modules of neural networks.

3.1 Introduction

Complexity is an illusive term, we seem to understand straight away what is meant when the word arises in a conversation, yet there is no useful quantitative definition of the word.

Here is how Random House Webster's Electronic Dictionary defines the term *complex*, the second definition is the most appropriate for our endeavour:

> **com-plex** (adj., *kuhm pleks'*)
> characterized by a complicated or involved arrangement of parts, units, etc.: complex machinery.

This definition does not hint of a usable quantitative insight into what *complexity* is! We turn our attention to information theory where in turn we find in-computable or impractical measures of complexity in the form of the Kolmogorov complexity and the Universal Distribution [43].

Following the work of Tin Kam Ho and Mitra Basu in 2002 [35] and their latter book of Ho, Basu and Law [6] we are describing the complexity of the datasets based on measurements calculated on the training data itself.

In order to define the complexity of an input dataset, we have resorted to using several measurements that characterize the intricacies of the dataset, which we associated further on, with the order of a polynomial that can be fitted onto the decision boundary between the two classes within the input dataset. We have named these additional measurements, meta-measurements, because they are aiding in defining the complexity of the data.

The mapping between the meta-measurements and the polynomial order of the decision boundary is predicted by a machine learning classifier. Thus, we are introducing another abstraction layer between the data and the actual classifier that is supposed to do the actual classification task. This layer of abstraction is supposed to evaluate the input training data and select a suitable classifier or in the case of modular neural networks make suggestions to alter the architecture of the modules in a neural network.

We set out to investigate methods of finding a suitable classifier for the training data that is available to estimate its parameters, this was hinted by previous published work of Cano in 2013 [12], Cavalcanti 2012 [13] and Sotoca in 2006 [71].

However, we did not find a reference to an automated system that would select a suitable classifier based on measurements obtained from the training data.

In the following section, we shall describe the definition of the meta-measurements to be calculated on the input data, which can be seen as complexity measurement, since they will be able to distinguish, for example in the simplest case, between a linearly separable dataset and a dataset which has a decision boundary of a polynomial of second order. The latter dataset will obviously have a higher complexity than the former.

The examination pertaining the potential validity of the proposed method is deferred to section 4.3 of chapter 4 when we will have discussed also the datasets used in the evaluation.

3.2 Meta-Measurements

Meta-measurements are measurements taken on the input training data of a pattern recognition problem, that can be associated with the complexity of the dataset on which they are calculated. We have chosen the meta-measurements to investigate from the literature and we have proposed some new meta-measurements.

The chosen meta-measurements were selected because of their descriptive ability for the spatial distribution of the sample points belonging to the two classes present in the classification dataset.

A concise list of all the meta-measurements can be found in Table 3.1 at the end of this section. In this table the meta-measurements which have a star in parenthesis (*) besides their shorthand name have been inspired from Ho and Basu's paper from 2002 [35] with slight modifications which will be mentioned when they are detailed. The other meta-measurements are grouped together by the first one or two letters in their name. These have a capital "E" to denote entropy measurements, the capital letters "FE" for feature evaluation measurements, "S" for statistical measurements, "G" for geometrical measurements and finally "A" for the angle profile measurements. The vector correlation measurement stands alone in its own group named "VC".

An important note about all of employed meta-measurements is that, they all work with datasets having at most two classes, they are not adapted to work with more than two classes. However, by splitting the k-class problem either in a "1 versus ALL" or "1 versus 1" fashion they are adapted to work even for this type of datasets.

F1: ***Multi-dimensional Fisher's discriminant ratio***

The first meta-measurement is the Fisher's discriminant ratio (F1) that is adapted to work with multi dimensional data.

It is defined as:

$$\mathrm{F1} = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2}$$

where μ_1 and μ_2 are the means of the feature values in vectorial form, respectively σ_1 and σ_2 are the variances of the two classes.

Ho, Basu and Law [6] use the maximum value of the Fisher discriminant over all the features of the dataset, however we employ the definition of the discriminant that takes into account all features as suggested by Xu and Lu [81].

The employed multi-dimensional Fisher criterion is used to measure the linear separability of the feature space obtained using Fisher Discriminant Analysis.

F2: ***Volume of overlap***

The volume of overlap (F2) meta-measurement is used to calculate the area or generally, in higher dimensions, the volume, of the overlap between the two classes in the dataset.

It calculates the area (or volume) of the bounding box of the samples belonging to each of the two classes and it finds the region which is overlapping and it normalizes it by the area/volume of the largest bounding box of all the samples from both classes together.

If the two bounding boxes are disjoined and do not have any common points between them, then the value of this measurement is the area/volume of the smallest region between the two bounding boxes. This is yielded by the negative sign of the measurement.

The F2 measurement is defined as:

$$\text{F2} = \prod_i \frac{MINMAX_i - MAXMIN_i}{MAXMAX_i - MINMIN_i}$$

$$MINMAX_i = \min\left(\max_i(\forall x_i \in \omega_1), \max_i(\forall x_i \in \omega_2)\right)$$

$$MAXMIN_i = \max\left(\min_i(\forall x_i \in \omega_1), \min_i(\forall x_i \in \omega_2)\right)$$

$$MAXMAX_i = \max\left(\max_i(\forall x_i \in \omega_1), \max_i(\forall x_i \in \omega_2)\right)$$

$$MINMIN_i = \min\left(\min_i(\forall x_i \in \omega_1), \min_i(\forall x_i \in \omega_2)\right)$$

where $i = 1 \ldots m$ is the index of the features in the dataset and $\max_i(\forall x_i \in \omega_1)$ should be read as the maximum of all feature values with index i that belong to class ω_1, with the other operators having the similar meaning.

F3: ***Feature Efficiency***

This meta-measurement counts the number of sample values that are separated by each feature and takes the maximum value across all the features. A sample is counted if it falls outside of the overlapping region between the two classes.

CL1: ***LD Classifier error rate***

This measurement is defined as the training error produced by the Linear Discriminant Classifier on the dataset that is being investigated and assessing its performance on the same dataset, therefore providing the, so called, training error rate. The classifier implementation is taken from the PRtools toolbox [20] and [74].

CQ1: ***QD classifier error rate***

In the case of this measurement the Quadratic Discriminant Classifier [20] and [74] is used but the methodology is the same as for the QD1 and CQ measurements.

CK1: ***KNN classifier error rate***

In the case of this measurement the K-Nearest Neighbour Classifier [20] and

[74] is used but the methodology is the same as for the CL1 and CQ1 measurements.

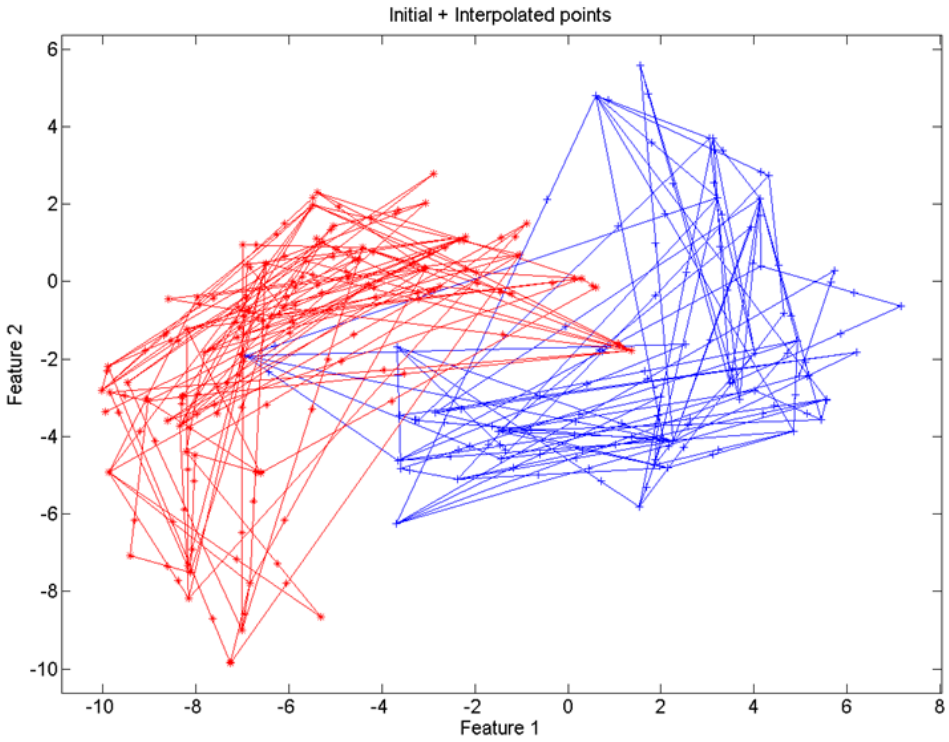


Figure 3.1: Addition of linearly interpolated data points to the banana dataset

LL1: ***LD classifier non-linearity***

The following three meta-measurements (LL1, LQ1 and LK1) make use of linearly interpolated points between pairs of points from the dataset that is being investigated.

Figure 3.1 shows the banana shaped dataset along with the interpolated points that were added along the segments that connect the pairs of points from the original dataset. The added points are labelled with the same label as the pair of points that were used to generate it.

The LL1 meta-measurement is calculated by training a Linear Discriminant Classifier on the original dataset and assessing its performance on the set of linearly interpolated points alone.

The LDC or Bayes-Normal-1 classifier employed here is described in [74] and the implementation is from the PRTools MATLAB Toolbox [20].

LQ1: ***QD classifier non-linearity***

This measurement uses the same methodology as in computing the LL1 and LK1 meta-measurements, with the only difference that the employed classifier is the Quadratic Discriminant Classifier (QDC) or also mentioned as Bayes-Normal-2 classifier, described in [74].

LK1: ***KNN classifier non-linearity***

This measurement uses the same methodology as in computing the LL1 and LQ1 meta-measurements, with the only difference that the employed classifier is the K-Nearest Neightbour classifier described in [74].

VC: ***Vector Correlation between features and labels***

The vector correlation measurement was inspired by the work of Hanson et. al [32] which has been adapted to work with dimensionality $d > 2$.

Given two sets of vectors: $\mathbf{z}_i = \begin{bmatrix} z_{i,1} & \dots & z_{i,j} \end{bmatrix}$ and $\mathbf{w}_i = \begin{bmatrix} w_{i,1} & \dots & w_{i,j} \end{bmatrix}$ with n vectors in each set, each vector having d components or having the dimensionality d.

$i = 1, \dots, n$ is the index of the vector in the set of n vectors, and $j = 1, \dots, d$ is the index of the j$^{\text{th}}$ component of the vector.

Individual feature statistical variation for each set of vectors has the following notation: $\sigma_{\mathbf{z},j}$ or $\sigma_{\mathbf{w},j}$ which represents the individual standard deviation of feature j of the first set of vectors $\mathbf{z}$ respectively the second set of vectors $\mathbf{w}$.

The vector correlation is defined as:

$$\text{VC} = sign(\xi)\sqrt{\frac{\sum_{k=1}^{d}\sum_{p=1}^{d}\left(\sigma_{\mathbf{z},k,\mathbf{w},p}\right)^2 + 2|\xi|}{\sigma_z^2\sigma_w^2}}$$

where, $\sigma_{\mathbf{z},k,\mathbf{w},p}$ is the covariance between the feature index k of vector set $\mathbf{z}$ and feature index p in vector set $\mathbf{w}$, both of which can take values between $1 \dots d$; and

$$\xi = det\left(\begin{bmatrix} \sigma_{\mathbf{z},k,\mathbf{w},p} & \dots & \sigma_{\mathbf{z},k,\mathbf{w},p} \\ \vdots & \ddots & \vdots \\ \sigma_{\mathbf{z},k,\mathbf{w},p} & \dots & \sigma_{\mathbf{z},k,\mathbf{w},p} \end{bmatrix}\right)$$

The two sets of vectors used to calculate the vector correlation are on one hand the raw feature values of dataset, and on the other hand, are the average class vectors corresponding for each vector in the raw feature set. Therefore, we are measuring how closely correlated are the feature values of the dataset to the class centres.

E1: ***Average Shannon entropy***

The Shannon entropy is calculated for each independent feature in the dataset, then it is averaged across all the features.

The Shannon information theoretic entropy of a set of values x_i with probabilities $p_i(x_i)$ is defined as [70]:

$$\mathrm{E1} = H(x) = -\sum_{i=1}^{n_x} p_i(x_i) \log(p_i(x_i))$$

We have used a non-parametric approach to estimate the probability distribution $p_i(x_i)$ without assuming any shape of the probability function. In this sense, the range $[\min_i(x), \max_i(x)]$ was divided into n_x number of bins having the width w which equals:

$$w = 2\ \mathrm{IQR}(x)\, n^{1/3}$$

where, $\mathrm{IQR}(x)$ is the inter-quartile range of the values in x. This heuristic of estimating the bin width is named the Freedman-Diaconis rule [26].

The values are counted into their corresponding bin and the resulting frequency count is then normalized by the total number of values found in x, therefore, $p_i(x_i)$ is a pseudo-probability estimation of a particular value x_i to be observed.

The convention was used that $0 \log(0) = 0$, since $\lim_{p \to 0} p \log(p) = 0$.

E2: ***Average MAXIMUM variance entropy***

The maximum variance entropy is calculated, according to the formula given below, for each independent feature in the dataset, then it is averaged across all the features.

$$\mathrm{E2} = V(x) = \frac{1}{2} \log(2\,\pi\,\boldsymbol{e}\ \mathrm{VAR}(x))$$

where, $\mathrm{VAR}(x)$ is the statistical variance of the variable x.

E3: ***Total Shannon entropy***

The total Shannon entropy is the summation instead of the average of the entropies (E1) corresponding to each feature column in the dataset.

E4: ***Total MAXIMUM variance entropy***

The total maximum variance entropy is the summation instead of the average of the entropies (E2) corresponding to each feature column in the dataset.

FE1: ***Feature evaluation criterion, Inter-Intra distance***

The following 6 meta-measurements (FE1 to FE6) are feature evaluation criterions obtained from the PRTools MATLAB toolbox [20] and described in [74].

This feature evaluation criterion was calculated using the following MATLAB statement: `FE1{i} = feateval(input_dataset, 'in-in');`

FE2: ***Sum of Mahalanobis distances***

This feature evaluation criterion was calculated using the following MATLAB statement: `FE2{i} = feateval(input_dataset, 'maha-s');`

FE3: ***Minimum Mahalanobis distances***

This feature evaluation criterion was calculated using the following MATLAB statement: `FE3{i} = feateval(input_dataset, 'maha-m');`

FE4: ***Sum of squared Euclidean distances***

This feature evaluation criterion was calculated using the following MATLAB statement: `FE4{i} = feateval(input_dataset, 'eucl-s');`

FE5: ***Minimum of squared Euclidean distances***

This feature evaluation criterion was calculated using the following MATLAB statement: `FE5{i} = feateval(input_dataset, 'eucl-m');`

FE6: ***1-Nearest Neighbour Leave-One-Out classification performance***

This feature evaluation criterion was calculated using the following MATLAB statement: `FE6{i} = feateval(input_dataset, 'NN');`

S1: ***Average Skewness***

Skewness is the ratio of the mean cubed deviation from the mean cube of the standard deviation [40] and [50]

$$\mathrm{S1} = \beta_1 = \left(\frac{\mathrm{E}\left[X - \mu_X\right]^3}{\sigma^3} \right)^2$$

The average is taken of the univariate skewness of each feature column of the dataset.

S2: ***Maximum Skewness***

The maximum skewness (S2) is calculated with the same formula as the average skewness (S1) but the maximum value across all the columns or features of the dataset is retained as opposed to the mean of S1.

S3: ***Minimum Skewness***

The minimum skewness (S2) is calculated with the same formula as the average skewness (S1) but the minimum value across all the columns or features of the dataset is retained as opposed to the mean of S1.

S4: ***Minimum absolute Skewness***

The minimum absolute skewness (S3) is calculated with the same formula as the average skewness (S1) but the minimum of the absolute value across all the columns or features of the dataset is retained as opposed to the mean of S1.

S5: ***Average Kurtosis***

The univariate kurtosis of a set of observations is defined as the ratio of the forth moment about the mean to the forth power of the standard deviation:

$$\text{S5} = \beta_2 = \left(\frac{E[X - \mu_X]^4}{\sigma^4} \right) - 3$$

The univariate kurtosis of each feature column of the dataset is calculated and the average value across all feature columns is stored.

S6: ***Maximum Kurtosis***

The maximum kurtosis is evaluated from the individual kurtosis values of the feature columns in the dataset calculated with the same formula as above.

S7: ***Average inter-features correlation***

In order to obtain the following meta-measurements (S7, S8 and S9), the sample Pearson correlation coefficients are calculated between pairs of feature columns of the dataset under investigation. The Pearson correlation coefficients are well established in the literature [22], [40], [62] and are defined as:

$$\text{S7} = \rho_{X,Y} = \frac{\sum_{i=1}^{n} \left(X_i - \bar{X}\right)\left(Y_i - \bar{Y}\right)}{\sqrt{\sum_{i=1}^{n} \left(X_i - \bar{X}\right)^2} \sqrt{\sum_{i=1}^{n} \left(Y_i - \bar{Y}\right)^2}}$$

The vectors X and Y are replaced, in turn, by the column feature values of the dataset. For the S7 meta-measurement we stored the average between all the pairs of correlation coefficients.

S8: ***Maximum inter-feature correlation***

The correlation coefficient is calculated as described in the meta-measurement S7 and for this meta-measurement the maximum value is stored.

S9: ***Minimum inter-feature correlation***

The correlation coefficient is calculated as described in the meta-measurement S7 and for this meta-measurement the minimum value is stored.

S10: ***Average feature to label correlation coefficient***

For the following three meta-measurements (S10 to S12), we calculated the correlation coefficients between each individual feature column and the class label vector of the dataset. The formula for calculating the correlation coefficients are the same as the one employed for the S7,S8 and S9 meta-measurements, but for the present meta-measurement (S10) we stored the mean value of all the coefficients calculated between the feature columns and the class labels.

S11: ***Minimum absolute feature to label correlation***

The correlation coefficient is calculated as described in the meta-measurement S10 and for this meta-measurement the minimum of the absolute value is stored.

S12: ***Maximum absolute feature to label correlation***

The correlation coefficient is calculated as described in the meta-measurement S10 and for this meta-measurement the maximum of the absolute value is stored.

S13: ***STATLOG*** γ

The following three data characterization measurements were inspired from the STATLOG project [50].

$$S13 = \gamma = 1 - \frac{2p^2 + 3p - 1}{6(p+1)(q-1)} \left(\sum_{i=1}^{q} \frac{1}{n_i - 1} - \frac{1}{n-q} \right)$$

where

p = the number of features or attributes of the dataset;

q = the number of classes in the dataset;

n = the total number of observations in the dataset;

n_i = the number of observations in the dataset that belong to class i,

$n = n_1 + n2 + \cdots + n_q$.

S14: ***STATLOG M***

This measurement is equal to Box's M test statistic, which is defined as:

$$S14 = M = \gamma \sum_{i=1}^{q} (n_i - 1) \log |S_i^{-1} S|$$

where
γ = S13 defined above;
S_i = the unbiased estimators of the covariance matrix of the samples belonging to the i-th class;
S = the unbiased estimator of the i-th sample covariance matrix;

S15: *STATLOG SD_ratio*

This measurement is the geometric mean ratio of standard deviations, expressed in the following form:

$$S15 = SD\_ratio = \exp\left(\frac{M}{p\sum_{i=1}^{q}(n_i - 1)}\right)$$

G1: *Boundary rotation angle*

The decision boundary between two classes in the dataset is obtained by training a K-Nearest Neighbour (K-NN) classifier using the whole given dataset.

Then, a probing dataset it created using a mesh of coordinate values in the range of [-1 , 1] for each dimension.

This probe dataset is then classified by the K-NN classifier in order to obtain a classification label. From this predicted label we can approximate where the decision boundary is lying by doing a linear interpolation between the coordinates of the probing mesh.

This decision boundary is then re-sampled with a constant number of sample points and by dividing the Euclidean length of the boundary into equal lengths. We used 18 number of sampling points empirically and because this will give us 16 angles between the consecutive segments.

This re-sampled decision boundary will be employed in the calculation of the following meta-measurements: G1, G2 and A1-A16.

The angle of the segment formed by the first point and the last point in the decision boundary made with the horizontal axis is measured and stored for this meta-measurement.

G2: *Number of intersections of the decision boundary with itself*

The re-sampled decision boundary is investigated by taking, in turn, all the segments and assessing whether they intersect any of the remaining line segments in the re-sampled decision boundary. If they do, then both of the segments are marked in a signalling matrix that they have intersected, in order not to

count twice for both the segments. Multiple intersections are also accounted for, by marking all pairs of segments that intersect as such in the signalling matrix. The signalling matrix has a number of lines and columns that equals the number of line segments.

The assessment whether two line segments intersect is done by solving the linear equations associated with the two line segments and then applying the boundary conditions.

In order to reduce the number of iterations, not all line segments are checked against all the other line segments, but instead, for a given line segment with index i, only the line segments with indices greater than i are checked, thus reducing the number of assessments.

G3: ***Length of the extracted decision boundary***

It has been suggested by previous research Macia et. al [45] and Prudencio et. al [63] that the length of the decision boundary is a good descriptor of dataset complexity. Macia et. al [45] estimate the length of the decision boundary by creating a Minimum Spanning Tree from the data points of a given dataset using the Euclidean distances as a dissimilarity metric, then they count the number of connecting points of opposite classes and divided by the total number of connections.

We used a direct approach in estimating the length of the decision boundary. Since, we already have a rough decision boundary obtained from using a KNN classifier, as described in the paragraphs pertaining the G1 meta-measurement, we just calculate the length of the decision boundary obtained from the KNN classifier.

A1-A16: ***Angle profile of the decision boundary***

We have chosen to give the name "angle profile" to the ordered set of angles between consecutive line segments in the decision boundary formed between the two classes in the dataset being investigated.

An example of angle profile is the following: 6.24°, -5.25°, -3.86°, -1.91°, -76.44°, -60.26°, -8.03°, -0.06°, 4.12°, 6.01°, 100.91°, 34.57°, 1.89°, 2.97°, 13.69°, 2.47°this corresponds to the re-sampled decision boundary in Figure 3.2.

In Figure 3.2 we show graphically the set of angles between all successive pairs of segments (depicted with thick black line) that were obtained by re-sampling the original estimate of the decision boundary (shown with the green line) between the two classes.

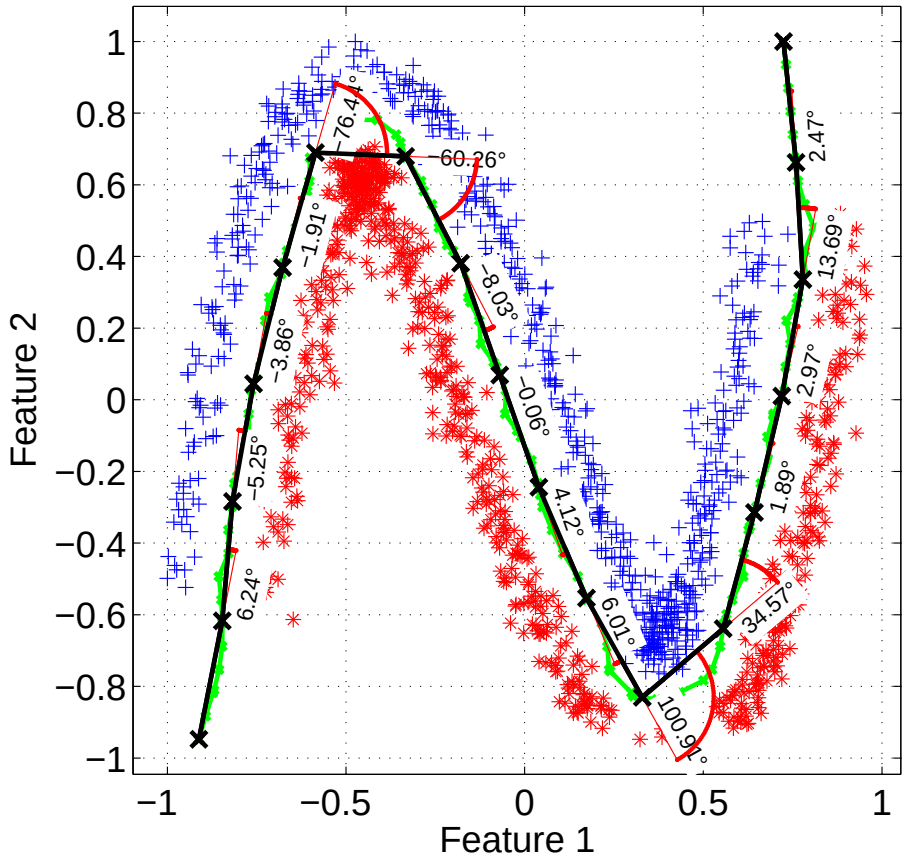


Figure 3.2: Example of an extracted angle profile of a decision boundary.

Table 3.1: List of Meta-Measurements

Nr.	Name	Description
1	F1(*)	Multi-dimensional Fisher's discriminant ratio
2	F2(*)	Volume of overlap
3	F3(*)	Feature efficiency
4	CL1(*)	LD classifier error rate
5	CQ1(*)	QD classifier error rate
6	LL1(*)	LD classifier non-linearity
7	LQ1(*)	QD classifier non-linearity
8	LK1(*)	KNN classifier non-linearity
9	VC	Vector Correlation between features and labels
10	E1	Average Shannon entropy
11	E2	Average MAXIMUM variance entropy
12	E3	Total Shannon entropy
13	E4	Total MAXIMUM variance entropy
14	FE1	Feature evaluation criterion, Inter-Intra distance
15	FE2	Sum of Mahalanobis distances
16	FE3	Minimum Mahalanobis distances
17	FE4	Sum of squared Euclidean distances
18	FE5	Minimum of squared Euclidean distances
19	FE6	1-Nearest Neighbour Leave-One-Out performance
20	S1	Average Skewness
21	S2	Maximum Skewness
22	S3	Minimum Skewness
23	S4	Minimum absolute Skewness
24	S5	Average Kurtosis
25	S6	Maximum Kurtosis
26	S7	Average inter-feature correlation
27	S8	Maximum Inter-feature correlation
28	S9	Minimum Inter-feature correlation
29	S10	Average feature to label correlation
30	S11	Minimum absolute feature to label correlation
31	S12	Maximum absolute feature to label correlation
32	S13	STATLOG γ
33	S14	STATLOG M
34	S15	STATLOG SD_ratio
35	G1	Boundary rotation angle
36	G2	Number crossings of the decision boundary with itself
37	G3	Length of the extracted decision boundary
38 - 53	A1 - A16	Decision boundary angle profile

3.3 Chapter Conclusions

In this chapter we have presented a brief introduction to dataset complexity evaluation and also described all of our proposed measurements that can be calculated from a training dataset that will be used in selecting the architecture of, single and latter modular, neural networks.

We have presented 53 measurements, that we named "Meta-Measurements". These meta-measurements are inspired from several fields, namely from: information theory entropy, pattern recognition feature evaluation, statistical data description and geometrical measurements of the decision boundary. These have been adapted and we have modified several these meta-measurements labelled F1, VC, E1-E4, to fit our purpose of data complexity evaluation.

Even though these individual features have been present in the literature in one form or another, they have not been used in this way to caracterize dataset complexity. This is why our approach is novel.

The prediction power of these meta-measurements will be investigated in the second part of Chapter 4 in Section 4.3 but also we are going to dedicate Chapter 5 to the investigation of the performance obtained by training several classifiers to recognize the order of polynomial that can be fitted onto a decision boundary, thus providing a complexity assessment of a given dataset. Furthermore, in Chapter 7 these meta-measurements will guide the architecture selection process in order to construct Modular Neural Networks for pattern classification.

Chapter 4

Data Description

In this chapter we will present the formalities of the data, describe the sources of realistic data, the pre-processing involved in importing the data into a common workable format and also the generation of artificial data, since both will be used in subsequent experiments.

The mathematical software package that was used throughout all the experiments described in this thesis is MathWorks MATLAB , Release R2012a together with the additional pattern recognition toolbox PRTools, Version 4.1.10 (released 25-June-2010), which can be downloaded from this location [20].

The data employed and generated in the present work is organized hierarchically in two layers. At the lower layer we have the "dataset" and at a higher layer we have the "database" which can contain several datasets.

A dataset, as we shall see later in this paragraph, contains amongst others, the raw sample feature values and their corresponding class labels. In general we can say, that one dataset contains the data associated with one pattern recognition problem.

The grouping of several datasets we shall denote as a "database", irrespective of any Relational DataBase Management Systems (e.g. MySQL, Oracle, IBM DB2 and many others). The word database shall be used in the rest of this document to refer to just the arrangement of datasets that have a common property. For example, the database of UCI Repository datasets.

A dataset will be denoted in this document by $\mathcal{D}_{type,index}$, where the type of dataset and the index identifying a particular dataset are used as subscripts. The designator types of datasets that we have examined are listed in Table 4.1. The subscript index will have a numeric value for the artificially generated datasets and a textual value for the datasets containing real life feature values.

The first two dataset designator types ($uniform$, $gaussian$) represent the group of synthetically generated datasets, while latter 2 designators (U, M) represent

Type designator	Description
$uniform$	Uniformly distributed random dataset with polynomial boundary
$gaussian$	Dataset of clouds of Gaussians with polynomial boundary
U	UCI repository datasets
M	MNIST machine learning benchmark dataset

Table 4.1: Type designator of the datasets

datasets collected from real life environments and made available for benchmarking machine learning algorithms, each of these types of datasets are going to be discussed in separate sections.

Any particular dataset indicated by $\mathcal{D}_{type,index}$it is in fact a conceptual grouping, of two major items:

- The matrix $\mathbf{X}$ of feature values corresponding to each sample item, with feature values belonging to the i^{th} sample are noted by $\mathbf{x}^{(i)} = [x_1^{(i)}, \ldots, x_n^{(i)}]^T$ and each individual feature value j of sample i being a real number $x_j^{(i)} \in \mathbb{R}$.
  $\mathbf{X}$ is an $(n \times m)$ matrix, with the samples arranged across the n rows, and the features of each sample arranged across the m columns.

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}^{(1)})^T \\ \vdots \\ (\mathbf{x}^{(i)})^T \\ \vdots \\ (\mathbf{x}^{(n)})^T \end{bmatrix}$$

  The individual components $\mathbf{x}^{(i)}$ of $\mathbf{X}$ are column vectors, hence they are transposed in order to be assigned to $\mathbf{X}$

- The vector of ground truth class labels ω, with the true labels of sample i are noted by $\omega^{(i)}$

$$\omega = \begin{bmatrix} \omega^{(1)} \\ \vdots \\ \omega^{(i)} \\ \vdots \\ \omega^{(n)} \end{bmatrix}$$

The conceptual analogy to $\mathcal{D}_{type,index}$was inspired from the programming paradigm that was employed. The data was stored in MATLAB PRTools' `dataset` objects have

the ability to store more information associated with a dataset object, for example it can store prior probabilities of the classes or other user information, which we will not describe here. Further information is available in the toolbox documentation [20] and associated book by [74]. However it is crucial to emphasize the implicit association in a dataset, between the sample features and their corresponding label for each sample.

During the course of our investigation we have used two main sources for our data, firstly the so called "*Real Life*" data and secondly, "*Synthetically Generated*" data.

Table 4.2: Summary of dataset categories.

Dataset category	Count	Details
Real life datasets		
– UCI ML Repository	10	Multi-class datasets (see table 4.3)
– MNIST benchmark	1	Multi-class dataset (see section **??**)
Synthetically generated datasets		
– Training data:		
– Uniformly distributed polynomial		2 class, polynomial boundary
– Normal separation	1.000	orders from 1 to 10, 100 each
– Increased separation	1.000	orders from 1 to 10, 100 each
– Decreased separation	1.000	orders from 1 to 10, 100 each
– Gaussian clouds		2 class, polynomial boundary
– Normal separation	1.000	orders from 1 to 10, 100 each
– Increased separation	1.000	orders from 1 to 10, 100 each
– Decreased separation	1.000	orders from 1 to 10, 100 each
– Testing data:		
– Uniformly distributed polynomial		2 class, polynomial boundary
– Normal separation	1.000	orders from 1 to 10, 100 each
– Increased separation	1.000	orders from 1 to 10, 100 each
– Decreased separation	1.000	orders from 1 to 10, 100 each
– Gaussian clouds		2 class, polynomial boundary
– Normal separation	1.000	orders from 1 to 10, 100 each
– Increased separation	1.000	orders from 1 to 10, 100 each
– Decreased separation	1.000	orders from 1 to 10, 100 each
Total number of datasets:	12.011	

The "Real life data", stands for the data that was obtained from the physical world by some sort of a measurement. These datasets contain samples belonging to more than two classes. This category hosts datasets from the UCI Repository of Machine Learning [25], as well as the MNIST dataset benchmark dataset of handwritten

digits.

The second category of data was generated deterministically by algorithms that will be described in section 4.2. Even though these datasets were generated deterministically they contain variability by employing pseudo-random number generators built-in the MATLAB software package.

As shown in Table 4.2, there are two databases of synthetically generated datasets having 6,000 datasets each as it will be described in the later section. The reason for their presence is determined by the need to reduce the bias of the training and testing algorithm which requires that the testing set is coming from the same model as the training set but has not been presented to the training algorithm. This behaviour is suggested by most references for best practices in assessing the pattern performance including L. I. Kuncheva [41] and the Proben1 benchmark problems & benchmarking rules by Prechelt [61].

In order to characterize a dataset's complexity we need the two types of data sources mentioned above. Obviously we need real-life data, since this is the type of data we would like our systems to operate upon. Beside the real-life data , we need to have some controlled examples of data, therefore we used the synthetically generated datasets. Since, these have a complexity that we can influence and vary during the generation process of such datasets.

4.1 Real Life Data

The datasets that fall within the "Real Life" category were obtained from one of the following sources:

- The UCI Machine Learning Reopository [25]
- The MNIST benchmark dataset for Neural Networks

The "Real life datasets" contain more than two distinct classes. When this is the case, then we must first decompose the c-class problem into a number, p of 2-class problems.

We need to have this decomposition in order to be able to calculate the complexity measurements we have discussed in the previous chapter, which, at the moment, can only operate with datasets that have only two classes.

The number of 2-class problems is given by the binomial coefficient:

$$p = \binom{c}{2} = \frac{c!}{2(c-2)!}$$

where,

p is the number of 2-class problems, and

c is the initial number of classes in the dataset.

The first sub-category is data retrieved from the UCI Machine Learning Repository [25] and is generally used as benchmarks for machine learning algorithms. This is a desirable characteristic of these datasets since the results produced on these datasets are comparable with other works in the same field.

4.1.1 UCI Machine Learning Repository

The details of the datasets that were employed from the UCI Machine Learning Repository [25] are listed in table 4.3. The table gathers the most essential details about the employed datasets, namely the name of the dataset, the number of features, the number of classes in the dataset, the average number of samples per class and the total number of samples per class.

Table 4.3: List of UCI datasets used.

Nr.	Dataset name	Number of features	Number of classes	Average samples per class	Total number of samples
1	abalone (*)	8	3	1,392.33	4,177
2	anneal (*)	31	5	179.6	898
3	iris	4	3	50	150
4	yeast (*)	8	10	148.4	1,484

Some of the values in the column relating to the average number of samples have an asterisk sign (*) corresponding to the dataset which has unequal number samples in each class. In this case only the "iris" dataset has equal number of samples per class (50 in this case), all the others have a varying number of samples per class, which can also be noticed by the fact that the average number of samples is not an integer, but a rational number, in some cases, not all.

Why were these datasets chosen?

These datasets were chosen for three main reasons:

- firstly, because many researchers used them in the past and there are comparative results on the exact same datasets;
- secondly, these datasets do not have missing values;
- and thirdly they have a fairly large number of samples within each dataset.

In the paragraphs to follow we are giving the details about the structure of each dataset from the UCI Repository along with references to the classification accuracies reported in the literature.

1. ***The Abalone Dataset***

   This classification task implies predicting the age of abalone sea snail from 8 physical measurements. Traditionally the age of abalone is determined by cutting the shell through the cone, staining it, and counting the number of rings through a microscope, which is time consuming.

   Number of features/attributes: 8

   Out these 8 features 7 are real valued and one has discrete values, having 3 possible categorical values.

   The summary of the attributes is given in Table 4.4.

   Predicted attribute: The number of rings of the abalone sea snail.

   Number of classes: 21

   Samples per class: [15, 57, 115, 259, 391, 568, 689, 634, 487, 267, 203, 126, 103, 67, 58, 42, 32, 26, 14, 6, 9]

   Total number of samples: 4168

Table 4.4: List of attributes of the Abalone dataset.

Name	Data type	Unit	Description
Sex	nominal	-	M, F, and I (infant) (predicted attribute)
Length	continuous	mm	longest shell measurement
Diameter	continuous	mm	perpendicular to length
Height	continuous	mm	with meat in shell
Whole weight	continuous	grams	whole abalone
Shucked weight	continuous	grams	weight of meat
Viscera weight	continuous	grams	gut weight (after bleeding)
Shell weight	continuous	grams	after being dried
Rings	integer		+1.5 gives the age in years

The UCI repository holds the dataset with 4177 samples, but during our experiments we had to remove 9 samples from the dataset because they were containing only 1 or 2 samples belonging to classes 1, 2, 24, 25, 26, 27 and 29. These samples clearly couldn't possibly convey enough information to be able to build a model for classification. However, the samples in this dataset are still highly overlapping which lead to the conversion of the labels from

values ranging between 1 – 29 into 3 age bands for the abalone snails and redistributing the aforementioned few samples into the 3 age bands. Grouping ring numbers 1 to 8 in the first age band, ring numbers 9 to 10 were assigned to band two, and finally ring numbers 11 and higher were assigned to the third age band.

From the original data the examples with missing values were removed (the majority having the predicted value missing), the UCI repository holds the data files that do not contain the samples with missing values, hence we considered that this dataset does not have samples with missing values. Also, the ranges of the continuous values have been scaled for use with an ANN (by dividing by 200).

Data comes from the original, non-machine-learning related, study of Warwick et. al. [53] from 1994.

2. ***The Annealing Dataset***

   This is a classification problem donated by David Sterling and Wray Buntine, which is related to the work published in 1988 [11].

   Number of features/attributes: 31

   Number of classes: 5

   Samples per class: [8, 99, 684, 67, 40]

   Total number of samples: 898

   Predicted Attribute: One of the categorical labels: "1", "2", "3", "5" and "U"

   The dataset stored in the UCI Repository had some shortcomings that were fixed in order to use this dataset. Firstly, there was a class described in the raw data file that didn't have any samples. This was class labeled "4" which has been removed since it was superfluous. The other issue was with constant feature values across all the samples of the dataset and didn't convey any information. These attributes were removed from the dataset, their index numbers in original UCI dataset were: [2, 19, 23, 26, 29 and 31]. Therefore, originally the dataset had 37 attributes, however in the present work we only used 31.

3. ***The Iris Flower Dataset***

   This is a very well known classification problem created by R.A. Fisher [23], Duda & Hart [19] and many others.

   Number of features/attributes: 4

   Number of classes: 3

Samples per class: [50, 50, 50]
Total number of samples: 150
Predicted Attribute: One of the categorical labels: "Iris Setosa", "Iris Versicolour" and "Iris Virginica"
The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other two, whereas the other two are not linearly separable from each other.

4. ***The Yeast Dataset***
Number of features/attributes: 8
Out these 8 features 7 are real valued and one is discrete feature.
Number of classes: 10
Samples per class: [463, 429, 244, 163, 51, 44, 35, 30, 20, 5]
Total number of samples: 1484
Predicted Attribute: Localization site of protein within a cell as a non-numeric or categorical attribute.

Classification accuracy of 55% was initially reported by [36] and more recently [57] reported a slightly lower best accuracy of 54% when using a mixture of 3 maximum entropy models. [3] reported an accuracy of 58.3% $\pm$0.6 using a modified boosting algorithm.

4.2 Synthetically Generated Data

In order to characterize the spatial distribution of the data-points in a pattern recognition dataset arriving to a pattern recognition system we have decided to use polynomials that can be superimposed to the decision boundary of the input data. Hence, in order to assess the performance of the systems we have developed, we needed some "ground truth" data that had polynomial decision boundaries of a known polynomial order. Therefore, we generated synthetically such datasets with the orders of the polynomials ranging from 1 to 10. These datasets will be discussed in the remainder of this section.

Before we dwell into the description of the algorithms used to generate the synthetic datasets, we have to stop to define what decision boundaries are.

A decision boundary is characteristic for a given dataset and a given classifier. Any classifier produces a decision boundary in the feature space of the given problem.

A decision boundary of a given pattern recognition problem is the locus of points in the feature space of the pattern recognition problem that has equal posteriori

probability produced by a given classifier. Therefore, if a new sample that is to be classified lies exactly on the decision boundary the classifier will not have enough information to assign it to one class or the other. It will assign a class label in a deterministic or random fashion, not based on any factual or learned information. However, this case is rather infrequent and does not present a major impediment in the pattern recognition world. The reason why we are concerned with decision boundaries is because any classifier produces a decision boundary.

We have generated datasets of two classes and two features (i.e. a 2 dimensional dataset) where the decision boundary formed between the two classes is a polynomial of a known order $n = [1, \ldots, 10]$.

The employed polynomials have the following general form:

$$P(x) = \sum_{i=0}^{n} a_i x^i = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0 x^0$$

where a_i are the coefficients of the polynomial and n is the highest power of x which appears in $P(x)$ or the order of the polynomial.

The data that we used in our experiments was artificially generated in order to have only two classes and a separating decision boundary between the two classes, a polynomial curve of a known order ranging from 1 to 10. Also, the data points that are generated have only 2 features (or a dimensionality of 2), therefore it can be represented in 2 dimensions and graphed easily.

The datasets that were generated can be grouped in two categories based on the distribution of the data points, as follows:

1. Uniformly distributed random feature values;

2. Normally distributed random feature values, that form Gaussian clouds around points lying on each side of the polynomial decision boundary.

Within each of these groups we have generated datasets with: positive separation (i.e.: more pronounced separation), negative separation (i.e.: less separation and more pronounced overlap in some cases) or neutral separation distance from the decision boundary itself.

We have generated 6,000 datasets as it was mentioned in table 4.2, which consists of datasets having polynomial orders from 1 to 10 and 100 datasets having the same order, in two categories of data point distribution (Uniform and Gaussian clouds) with 3 degrees of separation.

The flowchart of the algorithm used to generate all the synthetic datasets can be seen in Figure 4.1. In this figure the blocks coloured in light blue represent

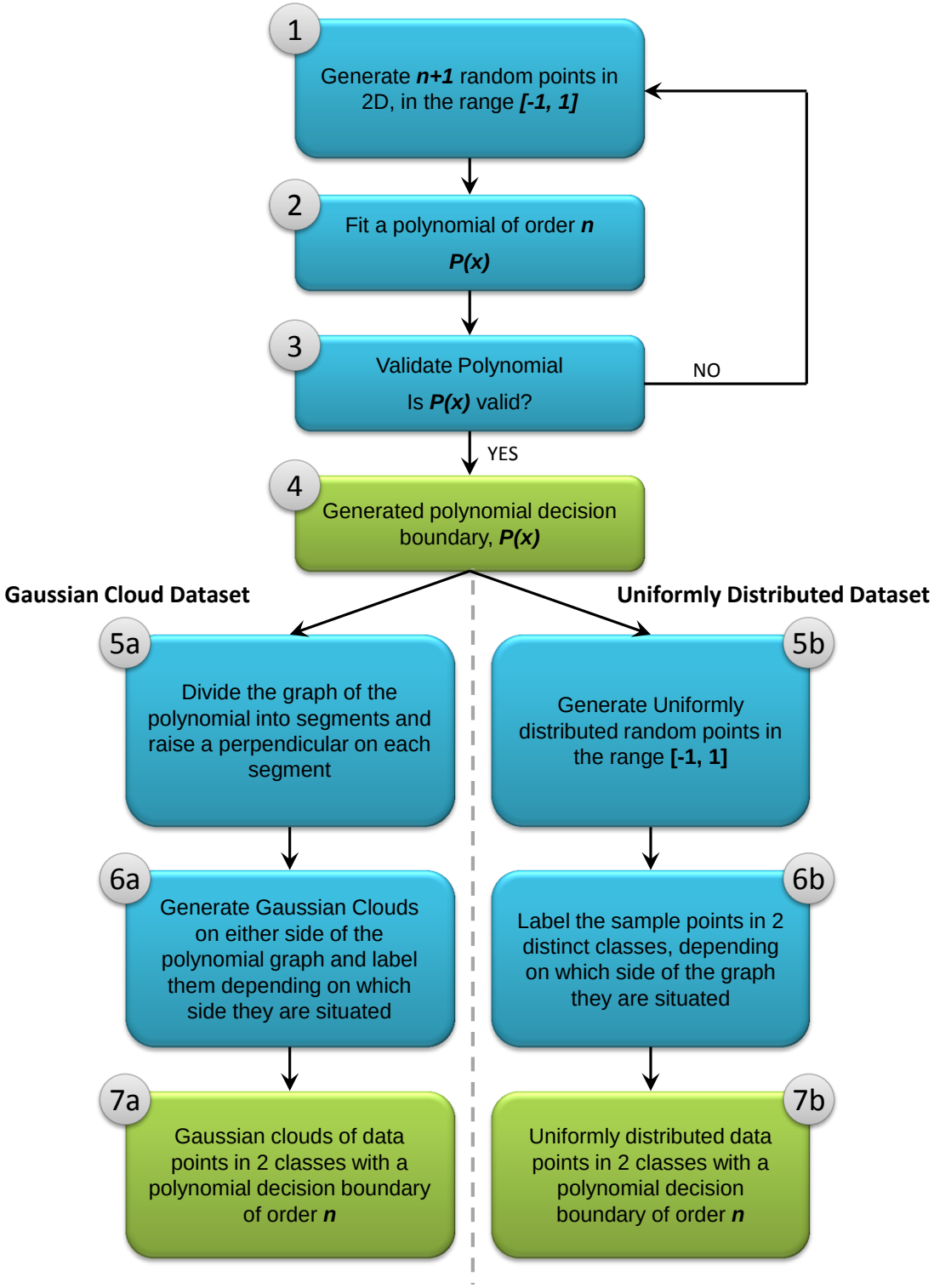


Figure 4.1: Flow chart of the artificial data generation algorithm.

algorithmic components, respectively the blocks coloured in green represent data outputs. From this figure we can see that the algorithm has 3 common algorithmic components that generate and validate the polynomial $P(x)$, these are numbered 1, 2 and 3, with the generated polynomial is numbered 4 in Figure 4.1. Whereas, for each category of datasets (i.e. Uniform and Gaussian Clouds) the final 2 algorithmic blocks are different. That is to say that in order to generate a dataset of the Gaussian Cloud type (labelled 7a) we need to follow steps 5a and 6a respectively. To generate a Uniformly distributed dataset (labelled 7b) we need to follow steps 5b and 6b.

In the following paragraphs we shall discuss the details of the common pathway of generating the synthetic datasets, while the details pertaining each dataset type will be discussed in sections 4.2.2 and 4.2.1.

The first step of the algorithm to generate the synthetic datasets depicted in Figure 4.1, is to obtain $n+1$ random pairs of coordinates in the 2-dimensional plane (x_k, y_k) where $x_k, y_k \in [-1, 1]$ and $k = 1, \ldots, n+1$.

These points are used in the second step which interpolates upon these $n+1$ points a unique polynomial $P(x)$ of order n.

The third step in the algorithm validates the polynomial by performing the following checks:

1. Sample points along the graph of the polynomial P(x), and make sure there are enough points that fall in the region bounded by -1 and 1 on either dimensions;

2. The graph of the polynomial $P(x)$ divides the region of the plane bounded by -1 and 1 on either dimensions into roughly the equal regions, without a major imbalance;

3. Verify that the number of tangent lines to graph of the polynomial is not less than n.

If all of these checks are passed, the polynomial $P(x)$ is deemed to be suitable for later use as a decision boundary for the synthetic dataset that is to be generated in both of the dataset categories mentioned above. When any of these checks fails, the polynomial will be abandoned, and a new one will be generated by jumping to the first step in the flowchart shown in Figure 4.1 and the process is repeated until a polynomial is found to pass all these checks. Rejecting a polynomial happens rather infrequently, but these checks are used to filter out malformed or trivial polynomials.

The tangent lines to the graph of the polynomial that are used in the validation checks mentioned above are exemplified in Figure 4.2. Since, there are infinite number of tangent lines to the graph of the polynomial, we have chosen to consider only

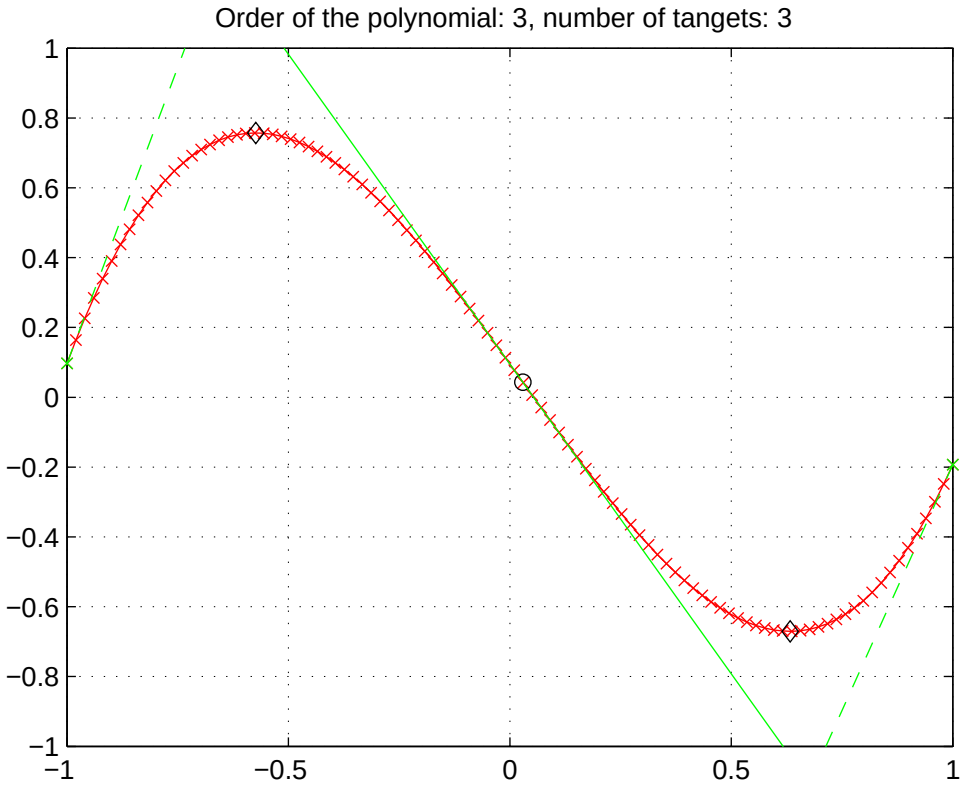


Figure 4.2: Plot of a third order polynomial, with its characteristic tangent lines.

a representative number of tangents to the graph, that is equal to n the order of the polynomial.

The considered tangent lines are the following:

- 2 tangent lines at the border of the graph, i.e. at the points $(-1, P(-1))$ and $(1, P(1))$. These are shown in Figure 4.2 as green dashed lines;
- Respectively, the tangent lines at the point of inflection of the graph of the polynomial. That is, at the location $(x_{inflection}, P(x_{inflection})$ where the second order derivative of $P(x)$ vanishes. This tangent line is shown in Figure 4.2 as the green full line.

The specifics generation of the datasets that have a polynomial boundary of a given order n is discussed next, in the following section.

4.2.1 Gaussian Cloud Datasets

We have generated Gaussian Cloud datasets with 3 types of separation between the two classes, as it was mentioned in Table 4.3, namely:

1. Neutral separation, see Figure 4.6;

2. Increased or positive separation;

3. Decreased or negative separation.

In order to describe the generation algorithm for the datasets of a Gaussian Cloud type, we have to refer to Figures 4.4 and 4.5.

From this figure we can observe how the datasets are generated.

Firstly, the graph of the polynomial $P(x)$ is sampled in the range [0, 1]. Any points that are outside the range [0, 1] are removed.

Then, line segments are considered by taking two consecutive points from the sampled graph of the polynomial. The endpoints of the line segments are depicted in Figures 4.4 and 4.5 by the blue diamond symbols.

For each of these line segments, perpendiculars are raised from the midpoint of each of the line segments on either side of the line segments. The length of the perpendiculars are given by the formula below and are show in the bar plot in Figure 4.3b), c) and d).

The endpoints of these perpendiculars form the centres of the cloud of points that will be generated. These endpoints are visible in Figures 4.4 and 4.5 as blue crosses with a circle around the cross.

These clouds of points will follow a Gaussian distribution:

$$X_{cloud,i} \sim \mathcal{N}(\mu_i, \Sigma_i)$$

where μ_i is the coordinate of the end of the perpendicular raised on the line segment obtained from sampling the graph of the polynomial and sigma is the circular covariance matrix:

$$\mathbf{\Sigma}_i = \begin{bmatrix} \sigma_i^2 & 0 \\ 0 & \sigma_i^2 \end{bmatrix}$$

The variance σ_i^2 is changing for each order $i = 1...10$ of the polynomial according to this formula:

$$\sigma_i^2 = 1/(200 \cdot i)$$

The perpendicular distance is varying with each order of the polynomial according to following formula:

$$PERPENDICULAR\_DISTANCE =$$

$$= \frac{MULTIPLICATIVE\_SEPARATION}{i} + SEPARATION$$

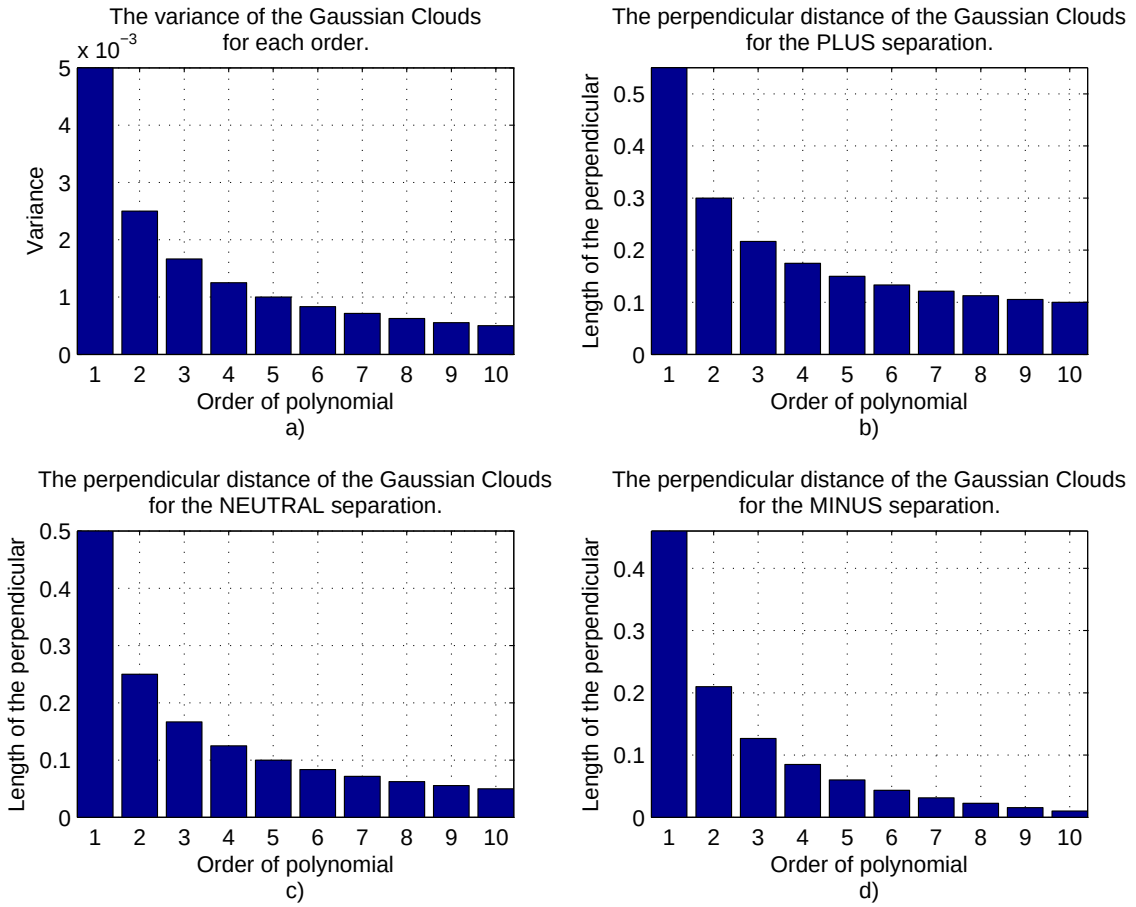


Figure 4.3: The parameters to generate the Gaussian Cloud datasets. a) the variance σ^2 for each order; The perpendicular distance for each order b) for the INCREASED separation; c) for the NEUTRAL separation; d) for the DECREASED separation.

Where the *SEPARATION* term is equal, in turn, to SEPARATION_PLUS, SEPARATION_NEUTRAL and SEPARATION_MINUS depending on what kind of dataset that is to be generated.

The values of the parameters discussed above are as follows:

Table 4.5: Values of the parameters to generate the Gaussian Cloud datasets

Name of the variable		*Value*
MULTIPLICATIVE_SEPARATION	=	0.50
SEPARATION_PLUS	=	0.05
SEPARATION_NEUTRAL	=	0.00
SEPARATION_MINUS	=	−0.04

The clouds of points that are generated on one side of the line segment will be assigned one class and the points generated on the other side of the line segment will be assigned another class, which is shown in Figures 4.4 and 4.5 as green and

red stars symbols.

This process is repeated for all the line segments the particular polynomial $P(x)$ has.

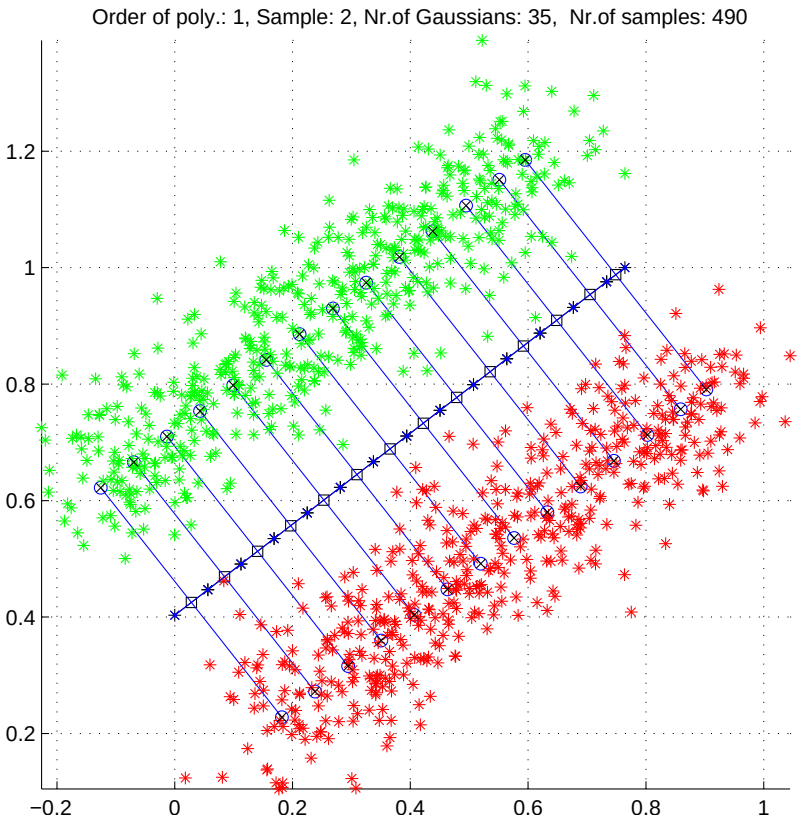


Figure 4.4: Detail showing how the Gaussian clouds are generated, for a polynomial of order 1.

Examples of Gaussian Cloud datasets are shown in Figure 4.6, where we can observe a sample for each polynomial order ranging from 1 to 10 of the decision boundary.

4.2.2 Uniformly Distributed Datasets

The generation algorithm for this dataset is outlined in Figure 4.1, and the steps labelled 1, 2, 3, 4, 5b and 6b. Steps 1 to 4 are already described in 4.2. The reminder of steps (5b and 6b) will be described in this section.

Firstly, the two dimensional plane region bounded by the range $[0, 1]$ is filled with points.

In the second step, the points generated in the above step are thresholded, that is, they are assigned one of two class labels depending whether they lie above or below the graph of the polynomial.

The Uniformly distributed datasets are generated with 3 types of separation:

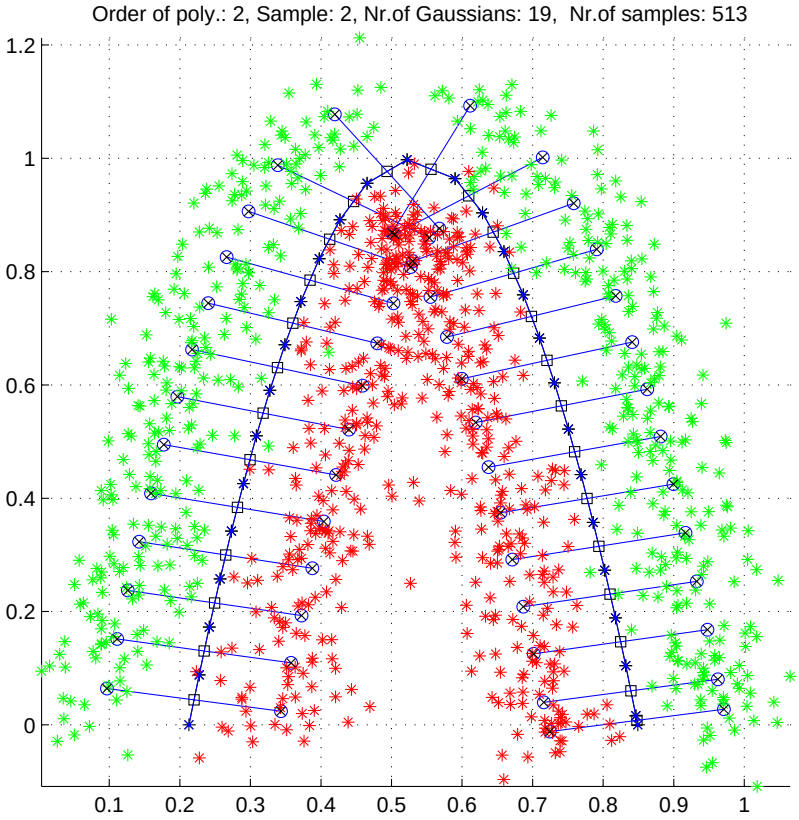


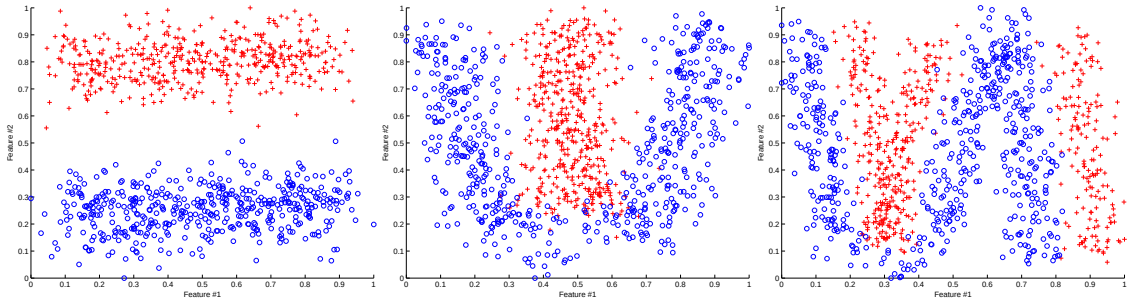
Figure 4.5: Detail showing how the Gaussian clouds are generated, for a polynomial of order 2.

1. Neutral separation, see Figure 4.7;

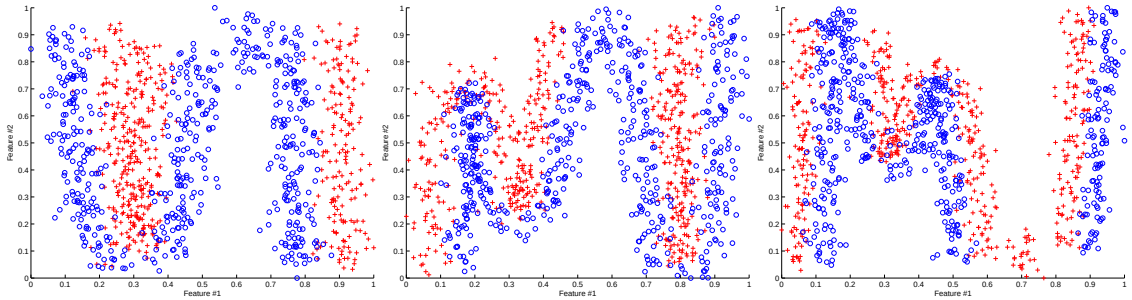
2. Increased or positive separation, see Figure 4.8;

3. Decreased or negative separation, see Figure 4.9.

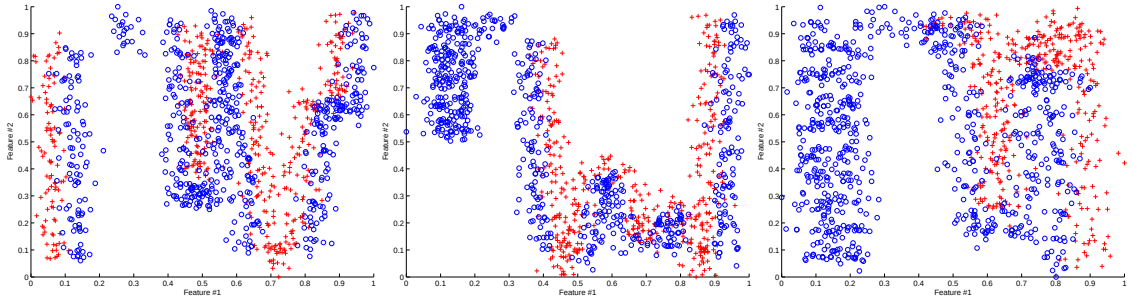
In order to produce the datasets with increased and decreased separation the data-points of one class are shifted along the vertical dimension with a constant.



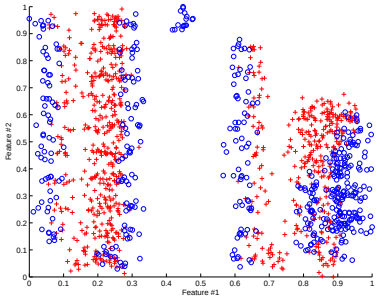
(a) Poly. order 1 (b) Poly. order 2 (c) Poly. order 3



(d) Poly. order 4 (e) Poly. order 5 (f) Poly. order 6

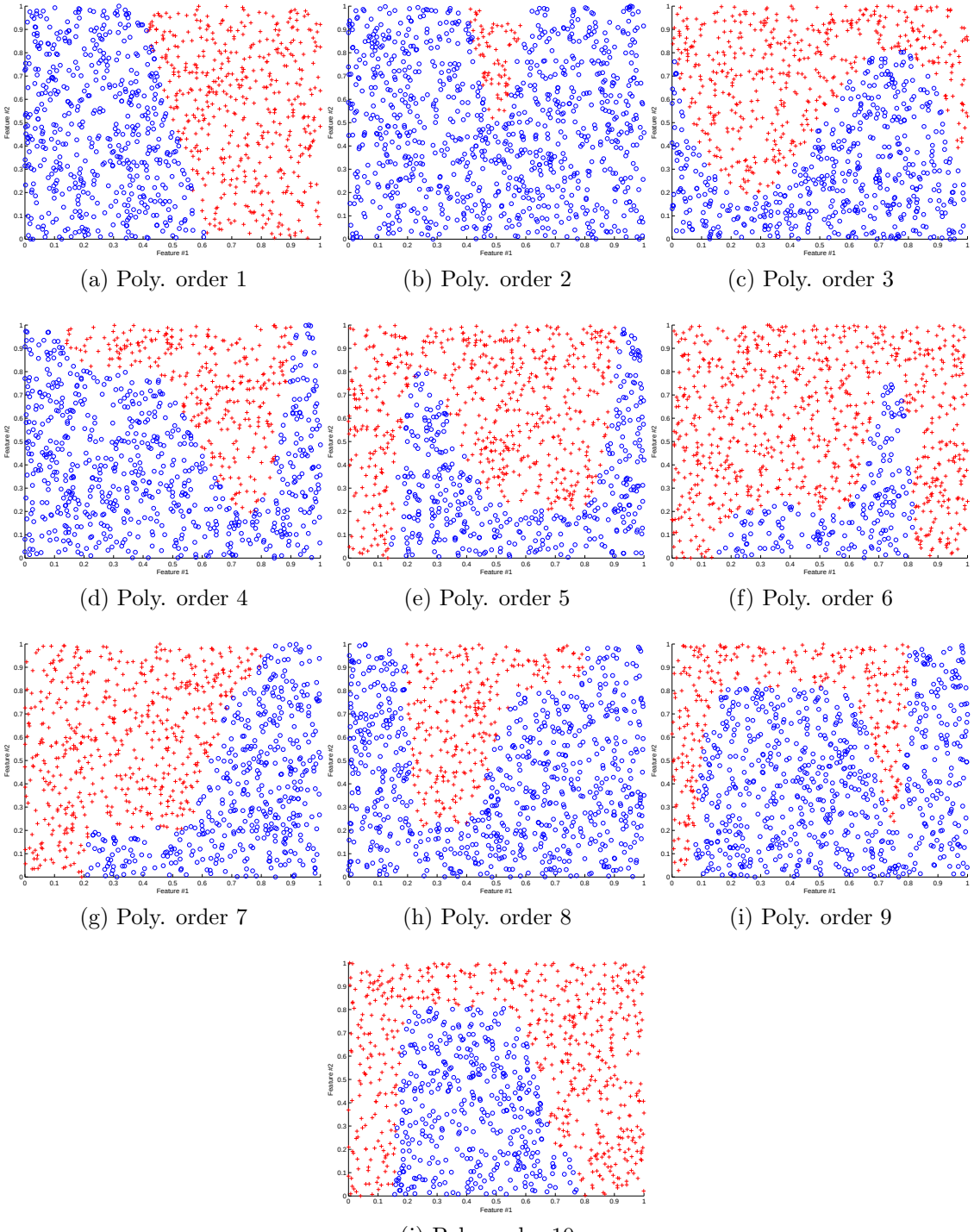


(g) Poly. order 7 (h) Poly. order 8 (i) Poly. order 9



(j) Poly. order 10

Figure 4.6: Scatter plots of Gaussian Clouds of points distributed along a polynomial boundary which has increasing order from 1 to 10, with neutral separation between the classes.



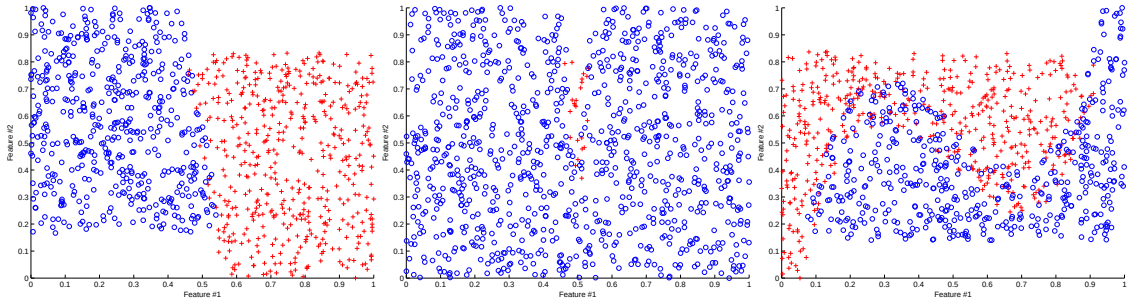
(a) Poly. order 1 (b) Poly. order 2 (c) Poly. order 3

(d) Poly. order 4 (e) Poly. order 5 (f) Poly. order 6

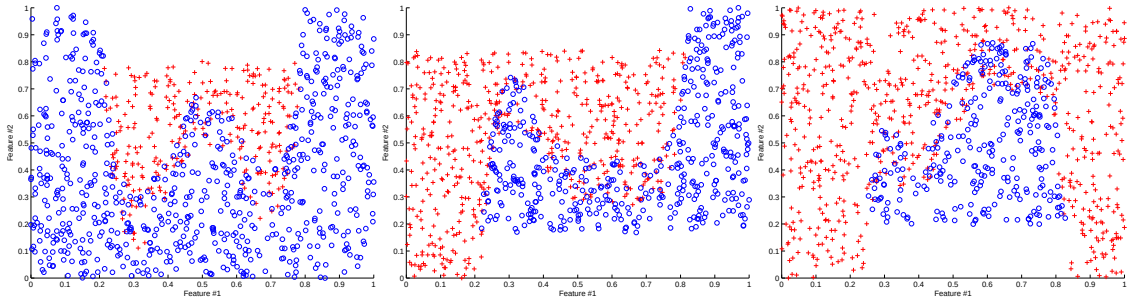
(g) Poly. order 7 (h) Poly. order 8 (i) Poly. order 9

(j) Poly. order 10

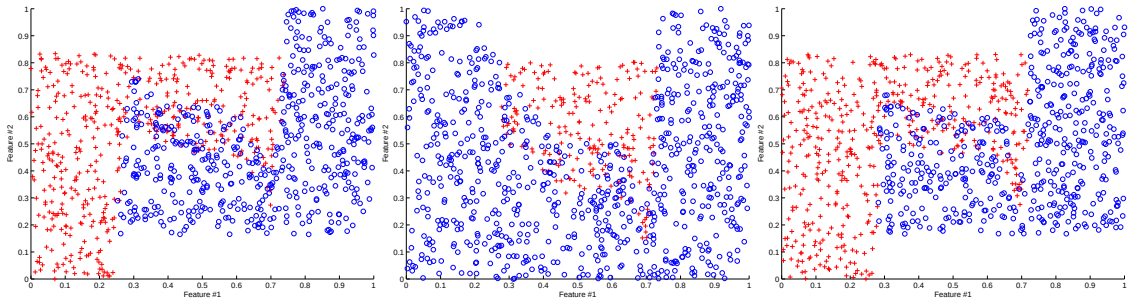
Figure 4.7: Scatter plots of uniformly distributed datasets with polynomial boundary of order 1 to 10, having NEUTRAL separation between the two classes.



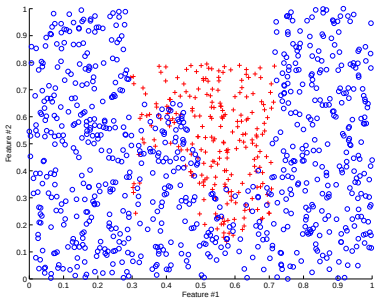
(a) Poly. order 1 (b) Poly. order 2 (c) Poly. order 3



(d) Poly. order 4 (e) Poly. order 5 (f) Poly. order 6

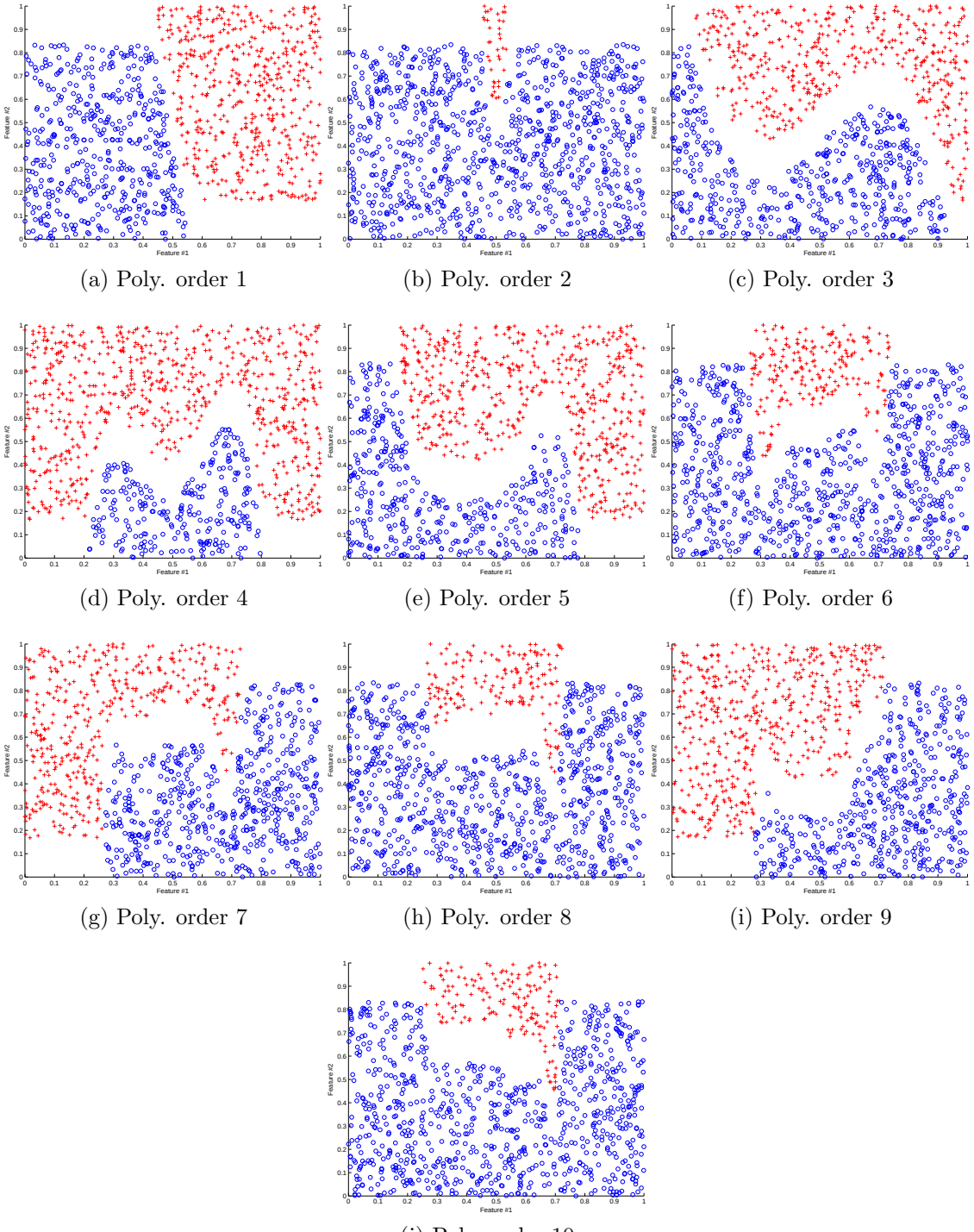


(g) Poly. order 7 (h) Poly. order 8 (i) Poly. order 9



(j) Poly. order 10

Figure 4.8: Scatter plots of uniformly distributed datasets with polynomial boundary of order 1 to 10, having an overlap between the two classes or DECREASED separation.



(a) Poly. order 1 (b) Poly. order 2 (c) Poly. order 3

(d) Poly. order 4 (e) Poly. order 5 (f) Poly. order 6

(g) Poly. order 7 (h) Poly. order 8 (i) Poly. order 9

(j) Poly. order 10

Figure 4.9: Scatter plots of uniformly distributed datasets with polynomial boundary of order 1 to 10, having INCREASED separation between the two classes.

4.3 Meta-Measurement Validation

In order to determine the degree of suitability of our proposed Meta-Measurements, we investigated the training error of 5 classifiers and also the unsupervised clustering of the Meta-Measurements values obtained from the synthetically generated database of datasets, for which we have the ground truth labelling of the polynomial order of the decision boundary within each individual dataset.

The validation of meta-measurements was conducted on the synthetically generated training database of datasets, described in Section 4.2 on page 47 and summarized in Table 4.2 on page 42.

For this purpose we divided the training database of 6,000 datasets in 8 groups, in the following manner:

1. Gaussian Cloud polynomial decision boundary (which have been abbreviated as: GC-Poly-2D);
2. Gaussian Cloud with increased separation (GC-Poly-Plus-2D);
3. Gaussian Cloud with decreased separation (GC-Poly-Minus-2D);
4. Uniform polynomial boundary (U-Poly-2D);
5. Uniform with increased separation (U-Poly-Plus-2D);
6. Uniform with decreased separation (U-Poly-Minus-2D);
7. all the above groups combined into one(ALL-Poly-2D);
8. and finally all the above groups combined into one, but without applying the feature reduction step (ALL-Poly).

On each individual group of datasets we have calculated all the Meta-Measurements as described in previous section (Section 3.2) and appended the polynomial order of each originating dataset as a label. This also forms a dataset which lends itself to be analysed, we shall call this dataset the Meta-Measurement Dataset.

Because of the large number of features in the Meta-Measurement Dataset (i.e. 51), we employed Fisher mapping, also known as Linear Discriminant Analysis (LDA), to reduce the number of features to 2. Fisher mapping or LDA maximizes the inter-class scatter and minimizes the within class scatter, which is tied to R.A.Fisher's name [22] and described in many pattern recognition texts such as [28], [19], [77] just to name a few.

The Fisher or LDA mapping was used to reduce the 51 features of the Meta-Measurement Dataset to just 2 spatial features with all the benefits that the Fisher

mapping has to offer, but mainly in the interest of visual inspection of the Meta-Measurements.

This reduced 2D Meta-Measurement Dataset and also the whole Meta-Measurement Dataset with all the features are then used in following four types of experiments:

1. train and test the performance of 5 classifiers, for which the obtained decision regions are plotted;

2. to perform unsupervised clustering then compare this labels produced by clustering with the ground truth labels;

3. evaluate the contribution of each feature, by ranking the feature's usefulness according to a calculated criterion;

4. calculate the error introduced in the synthetic data generation process. We are measuring the Mean Squared Error (MSE) between the graphs of the original generating polynomial and the estimated boundary produced by the generation of 2-class datasets with a decision boundary given by the original polynomial (as described in 4.2) then approximating this decision boundary with a KNN classifier.

In the following four sections we are going to describe the experiments mentioned above.

4.3.1 Classification Training Error and Decision Regions

The classification errors of the LDC, QDC, KNN, 1NN and 3NN classifiers obtained for each of the Meta-Measurement groups of data can be seen in Table 4.6. Also, in Table 4.6 we can find the references to the figures where the decision regions produced by each classifier can be found.

The classifiers have been chosen for their simplicity of operation which can reveal the possible underlying structure of the data.

The Linear Discriminant Classifier (LDC) and Quadratic Discriminant Classifier (QDC) were chosen because they assume normally distributed data. The K-Nearest Neighbour classifier (KNN) and its variations using only 1 nearest neighbour(1NN), respectively 3 nearest neighbours(3NN), do not assume any particular distribution of the data and this is why they are so successful, however these require to store all the training samples for their operation. A description of their internal workings can be found in [20].

The MATLAB code that was used to produce the training error estimates and the plots of the decision regions is shown in the box below.

Table 4.6: Training errors of 5 classifiers on the 8 groups of Meta-Measurements

	Classifier error rates					2D-Decision
Database name	LDC	QDC	KNN	1NN	3NN	Regions
GC-Poly-2D	29.40%	29.26%	27.34%	0.00%	20.05%	See Fig. 4.10
GC-Poly-Plus-2D	40.66%	42.58%	37.91%	0.00%	27.06%	See Fig. 4.12
GC-Poly-Minus-2D	3.02%	3.02%	3.02%	0.00%	2.06%	See Fig. 4.14
U-Poly-2D	48.21%	47.66%	44.64%	0.00%	31.18%	See Fig. 4.16
U-Poly-Plus-2D	46.84%	45.88%	45.60%	0.00%	30.63%	See Fig. 4.18
U-Poly-Minus-2D	49.86%	48.35%	40.66%	0.00%	33.10%	See Fig. 4.20
ALL-Poly-2D	50.41%	49.59%	47.34%	0.00%	33.20%	See Fig. 4.22
ALL-Poly	44.21%	22.92%	0.00%	0.00%	31.85%	N/A

```
REDUCED_NUMBER_OF_DIMMENSIONS = 2;

transformed_fisher_metames_set = fisher_transform( ...
    meta_measurements, ...
    [], ...
    [], ...
    REDUCED_NUMBER_OF_DIMMENSIONS);

% Classify the FISHER features
w1 = ldc(transformed_fisher_metames_set);
e1 = testc(transformed_fisher_metames_set, w1);

figure();
scatterd(transformed_fisher_metames_set, ...
         REDUCED_NUMBER_OF_DIMMENSIONS);
plotc(w_to_plot, 'col');
legend({'Class 1','Class 2','Class 3','Class 4','Class 5', ...
        'Class 6','Class 7','Class 8','Class 9','Class 10'}, ...
        'Location', 'NorthEastOutside');
```

By examining the error rates of the 5 classifiers in Table 4.6 we observe that there are three behaviour patterns to be seen in the values shown in this table.

1. Firstly, we see that for 7 out of 8 groups of Meta-Measurements and 4 out of

5 classifiers, the classification error is spread between over 31% to just under 64% (we omit the the 4^{th} column of values corresponding to 1NN and the 3^{rd} row of values corresponding to the GC-Poly-Minus-2D dataset). A distinct decrease in the error rate is to be seen by the nearest neighbourhood classifiers as opposed to the Bayesian classifiers of first and second orders.

It should be noted that the KNN classifier is optimized to minimize the "Leave-One-Out" Error and it takes a variable number of neighbours opposed to 1 or 3 number of neighbours used by the 1NN and 3NN classifiers, that are fixed by design.

Also, it is important to take into account that the classification problem posed by us here implies a decision between 10 classes, the orders ranging from 1 to 10 of the polynomial that can be fitted onto the decision boundaries of the synthetically generated datasets. In these circumstances, the chance of guessing the correct order by randomly guessing with a uniform probability distribution is 1 in 10 or 10%. Which actually would equate to a 90% error rate, since $error\_rate = 100\% - accuracy\_rate$.

Therefore, the error rate achieved by these classifiers on the mentioned groups of datasets is better than random guessing by a factor of 1.5 to 3 times reduced error rate.

2. Secondly, the 1NN classifier consistently achieves a perfect classification represented by the 0.00% error rate, which is an indication that it is actually over-trained and it will produce a poor generalization performance.

   As a side note, when we do an experiment and train a 1NN classifier on a 2-dimensional dataset that is composed of sample points of the two classes are drawn from the same uniform distribution with values for both classes in the same range $[a, b]$, that is, the two classes are completely confused, we find that the training error of this classifier is exactly 0.0%. In other words, the 1NN classifier is notoriously over-fitting the training data. Which makes sense, because if we think about how the 1NN classifier operates, it stores all the training samples, then when it is tested on the same training data, all the distances to the closest learned samples are zero. Therefore, the training error for this classifier is in most cases meaningless, it does not convey any information about what the classifier has learnt because it actually has stored all the training data perfectly. Nonetheless, if the 1NN classifier is tested on data that has not been used in the training of the classifier, this testing error is an estimation of what the classifier has learnt.

3. Thirdly, the error rate on the GC-Poly-Minus-2D group of Meta-Measurements is in-between the two behaviours described above. It is not a perfect classification and it is nowhere close to the theoretical maximum achieved by randomly guessing. This suggests that the classifiers trained on the GC-Poly-Minus-2D group of Meta-Measurements had extracted the meaningful patterns existing in this dataset and it is very likely that the generalization performance on this dataset and these given classifiers will be good.

The Table 4.6 also lists the figure numbers of the decision regions produced by the LDC classifier on each of the training groups of datasets in 2-dimensions. From these figures we can see the actual distributions of the Meta-Measurement Dataset values transformed by a Fisher Mapping into 2-dimensions, obtained for each known order of the polynomial decision boundary present in the synthetically generated datasets.

It has to be said that the Fisher Mapping is not orthogonal and the Fisher Mapping was calculated for each of the Meta-Measurement groups individually. Therefore, the scatter-plots in the figures mentioned in 4.6 are not directly comparable.

Nonetheless, the scatter-plots are extremely useful in observing how the Meta-Measurements separate the individual polynomial orders of the decision boundary. Also, the scatter-plots are useful to assess how the classifier (in this case the Linear Discriminant Classifier), has learned and produced its decision regions.

There is a "Not Available" (or "N/A") entry in Table 4.6 for the "ALL-Poly" group, that corresponds to the Meta-Measurements calculated on all of the synthetic datasets, in this instance we have not applied Fisher Mapping and therefore this is not a 2-dimensional dataset, for this dataset we cannot produce the scatter plot of the decision regions.

From all of the observations made above we can draw the conclusion that the best classifier to use in order to classify the order of the polynomials to be fitted onto the decision boundary of an new incoming dataset would be the one trained on the GC-Poly-Minus Meta-Measurement dataset.

4.3.2 Clustering Validation

The second set of validation checks that have been conducted are the unsupervised clustering validations. For this purpose we have used the same Meta-Measurement datasets as described in the previous section, but we have applied two types of unsupervised clustering algorithms:

1. Hierarchical Clustering [37];

2. K-Means Clustering.

Both unsupervised clustering algorithms were given the correct number of clusters they are suppose to find, then the algorithms had been run, and the labelling that was produced by each of algorithms is checked against the ground truth labellings of each Meta-Measurement datasets. The results of this experiment can be seen in Table 4.7 along with the reference to the figure that shows the dendrogram produced by the hierarchical clustering for each of Meta-Measurement datasets.

Table 4.7: Hierarchical Clustering and K-Means Clustering error rates on the 8 groups of Meta-Measurement datasets

	Clustering method		
Database name	H-Clust	K-Means	Dendrogram
GC-Poly-2D	47.80%	33.65%	See Fig. 4.11
GC-Poly-Plus-2D	78.30%	44.64%	See Fig. 4.13
GC-Poly-Minus-2D	22.53%	15.38%	See Fig. 4.15
U-Poly-2D	64.15%	57.14%	See Fig. 4.17
U-Poly-Plus-2D	67.03%	56.18%	See Fig. 4.19
U-Poly-Minus-2D	66.62%	53.85%	See Fig. 4.21
ALL-Poly-2D	86.36%	57.44%	See Fig. 4.23
ALL-Poly	81.68%	80.06%	N/A

From Table 4.7, we can observe that the Hierarchical Clustering algorithm is consistently outperformed by the K-Means algorithm in the error rate of identifying the correct clustering labels, even for the same Meta-Measurement Dataset. The Hierarchical Clustering algorithm produces error rates between about 86% to 48% (in the best case), whereas the K-Means clustering algorithm is producing error rates between 80% and 51% in 6 out 7 scenarios but a very low error rate of just 15.38% for the GC-Poly-Minus-2D dataset.

While even Stephen C. Johnson, the proposer of the Hierarchical Clustering algorithm [37], has noted when he was invited to comment in Thomson Reuter's Current Content - This Week's Citation Classic [38] about his article from 1967, that had more than 770 citations, he wrote: "it is very easy to get a computer's blessing without confronting the data's deficiencies". For this reason we have plotted the decision surfaces produced by the LDC classifier and also the dendrograms that stand at the basis of the Hierarchical Clustering algorithm.

However, if we examine the layout of the polynomial orders produced by the transformed meta-measurements in Figures 4.10, 4.12, 4.16, 4.18, 4.20 and 4.22, we can see that for most of the experiments that points with different labels are confused

and overlapping, with the exception of the "GC-Poly-Minus-2D" group of datasets, where the different labels are visibly separated as it can be seen in Figure 4.14.

The dendrograms shown in Figures 4.11, 4.13, 4.15, 4.17, 4.19, 4.21 and 4.23 represent on the horizontal axis the number of samples in each cluster, while the vertical axis represents the Euclidean distance between the clusters that were found.

A dendrogram should be interpreted as a tree-like graph that shows the distances where two clusters are merged into one. There are several inverted "U"-shaped lines in a typical dendrogram, which might have uneven legs. The length of each leg represents the distance to the nearest cluster.

Therefore, we would like to maximize the distance between clusters for a good classification.

A full dendrogram shows all the clusters that can be formed from a given dataset, up-to the cluster containing only one sample, which is impractical when there are many samples in a dataset. For this reason, we select a number of clusters that are to be shown in the dendrogram.

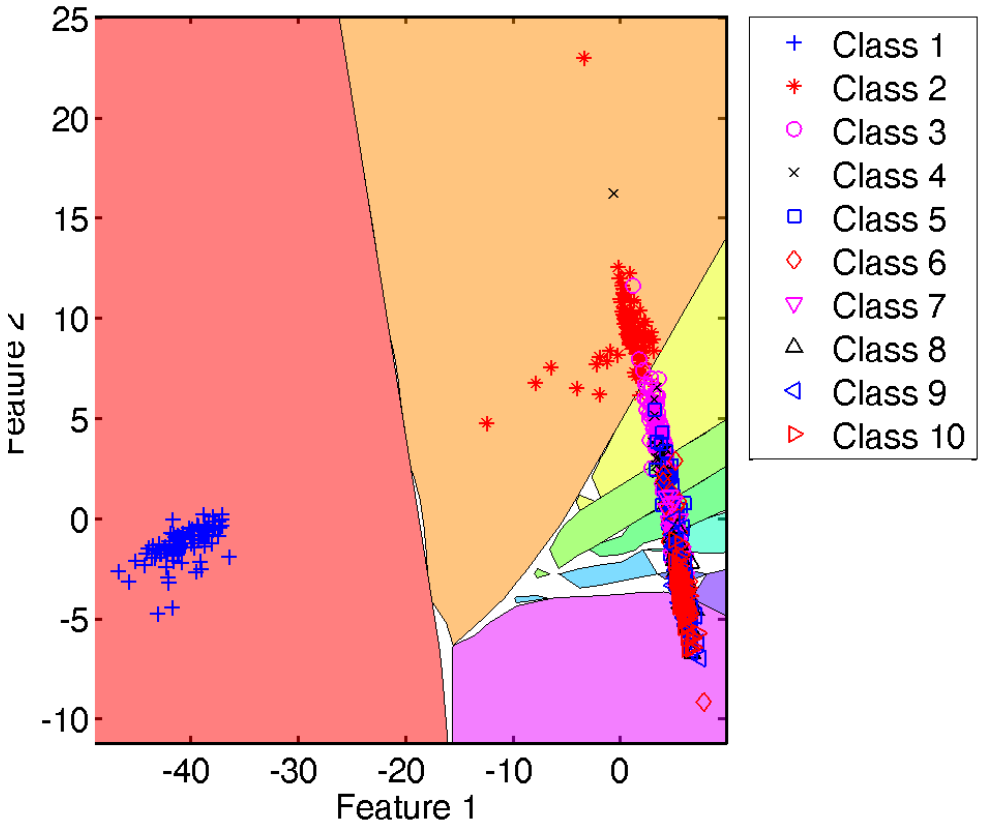


Figure 4.10: LDC decision regions, on the GC-Poly-2D database

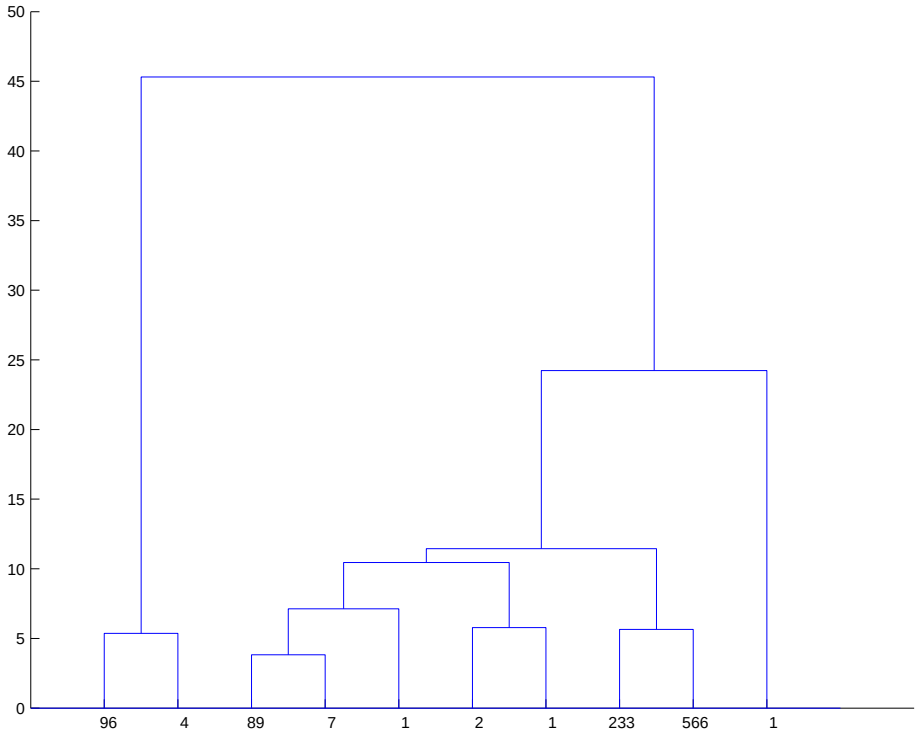


Figure 4.11: Hierarchical clustering dendrogram of the GC-Poly-2D database

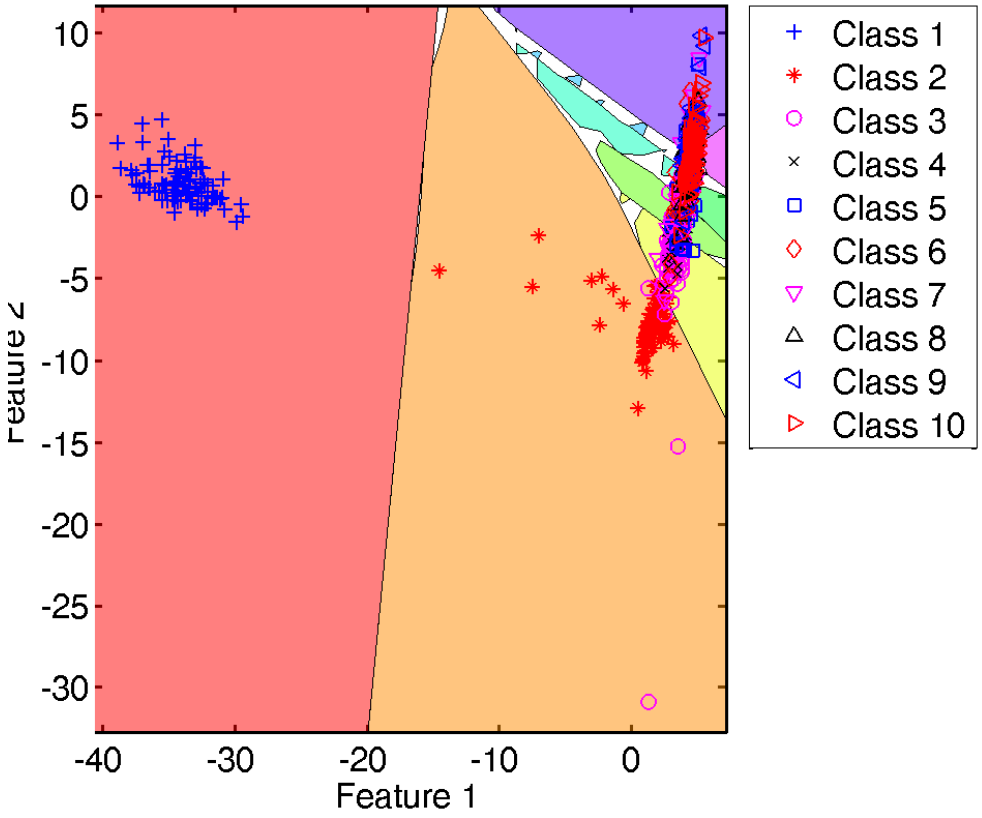


Figure 4.12: LDC decision regions, on the GC-Poly-Plus-2D database

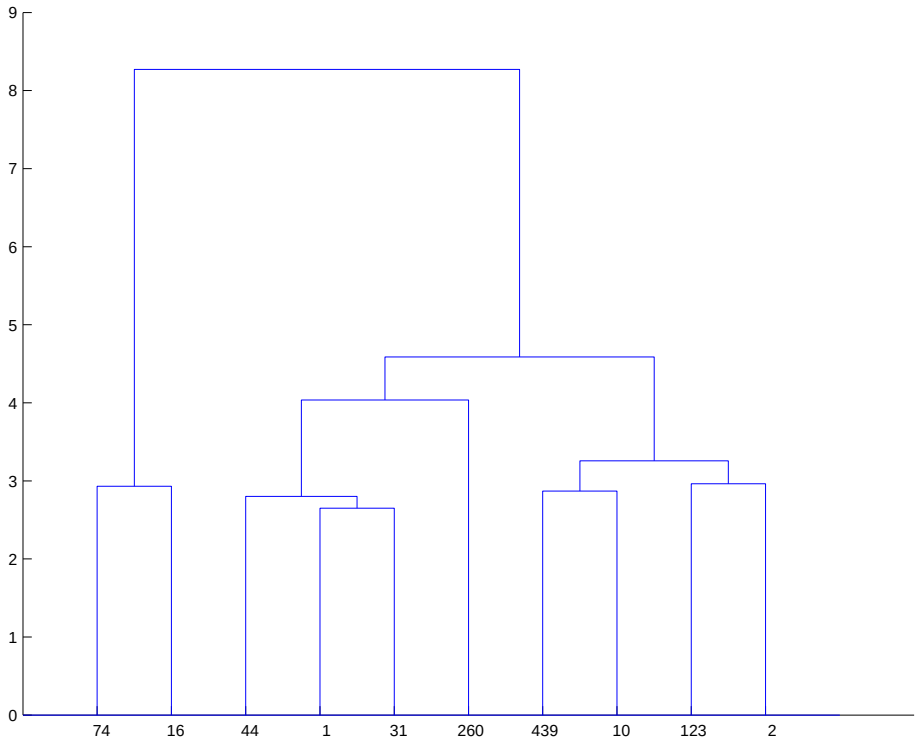


Figure 4.13: Hierarchical clustering dendrogram of the GC-Poly-Plus-2D database

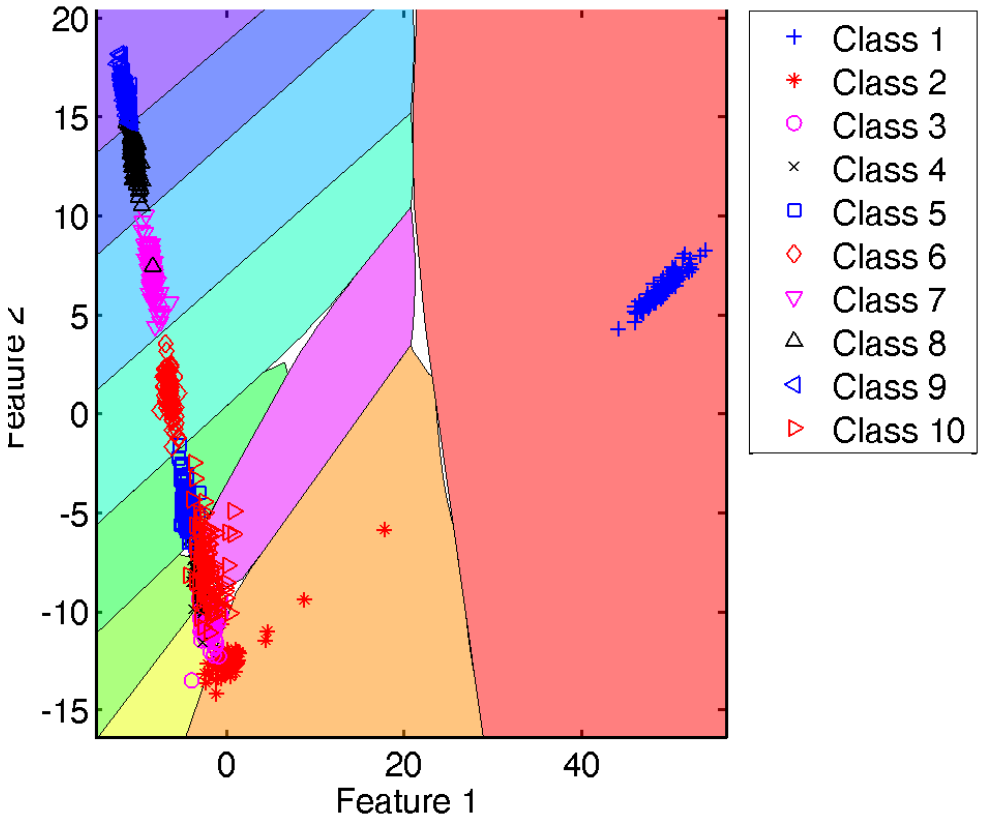


Figure 4.14: LDC decision regions, on the GC-Poly-Minus-2D database

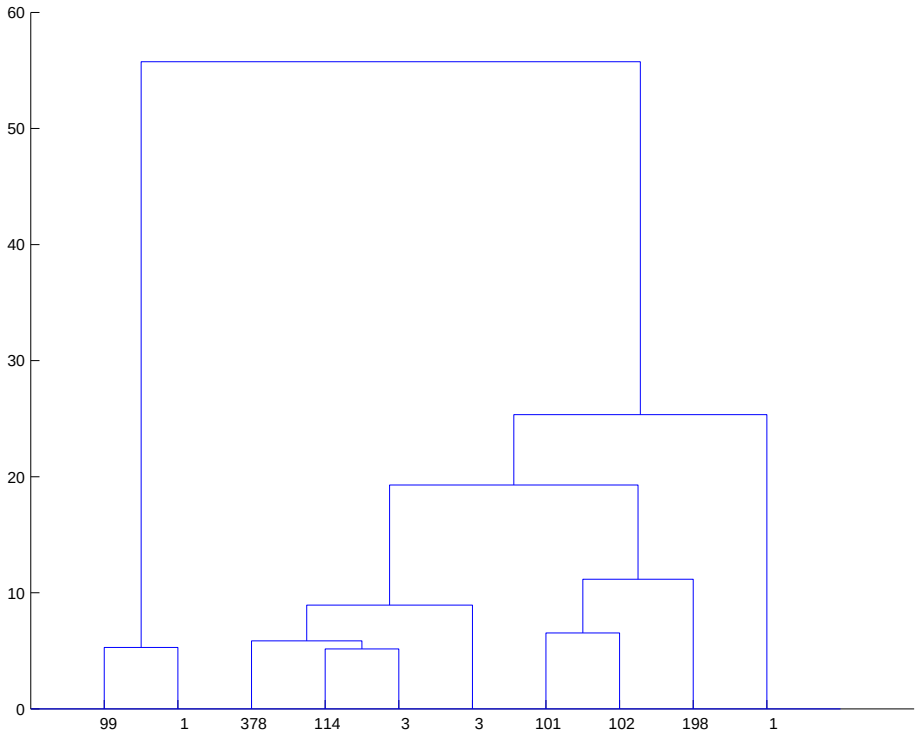


Figure 4.15: Hierarchical clustering dendrogram of the GC-Poly-Minus-2D database

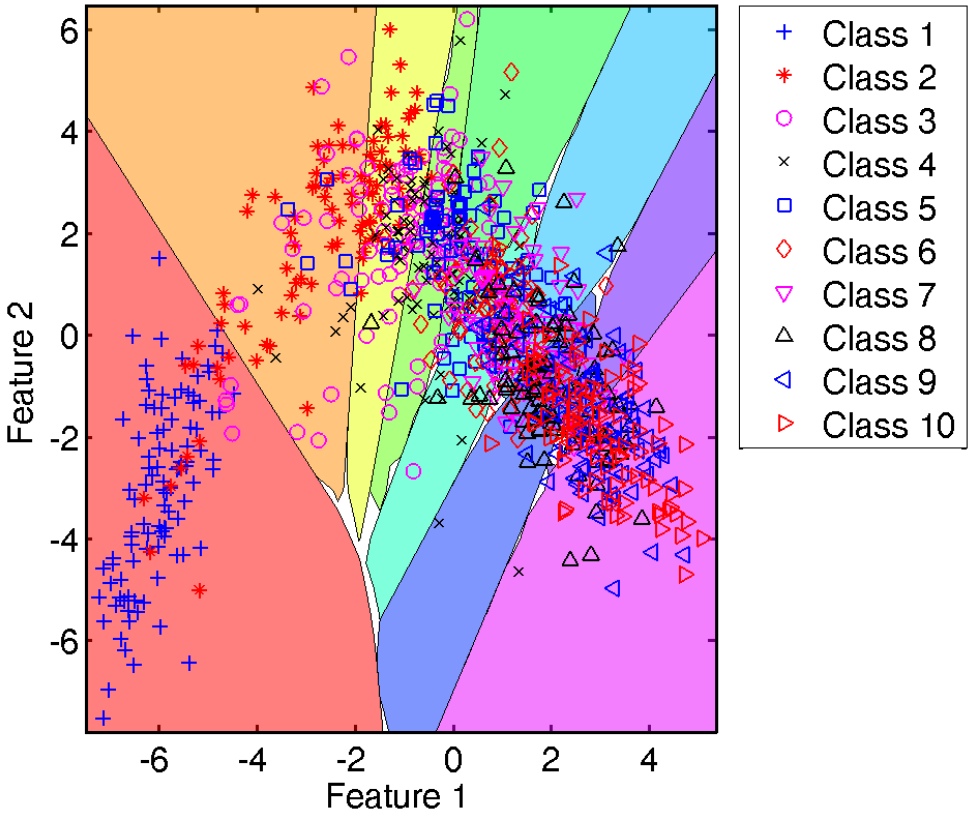


Figure 4.16: LDC decision regions, on the U-Poly-2D database

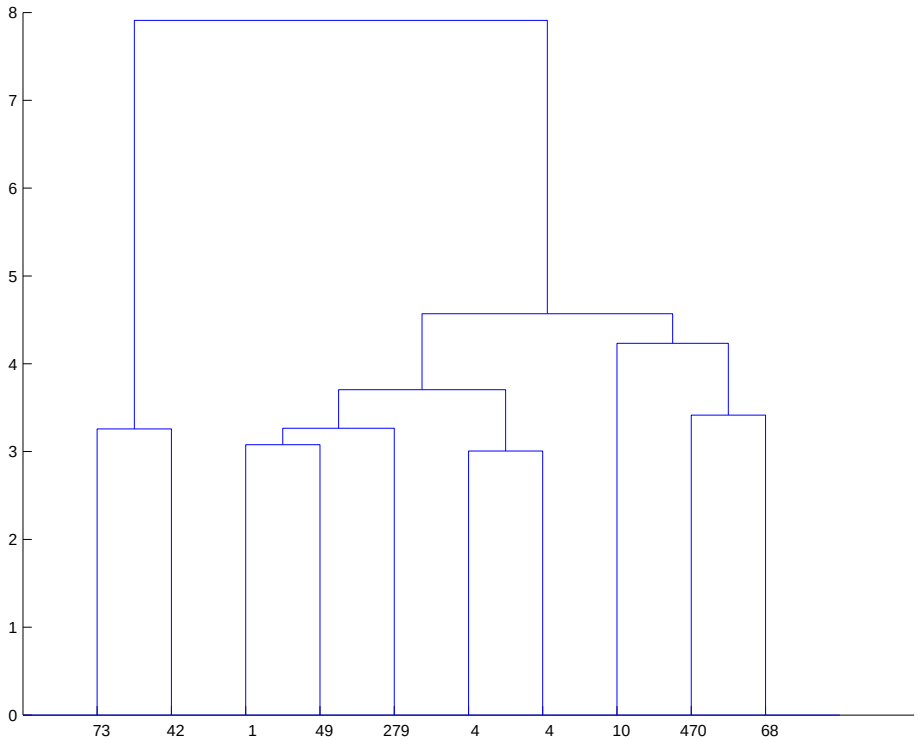


Figure 4.17: Hierarchical clustering dendrogram of the U-Poly-2D database

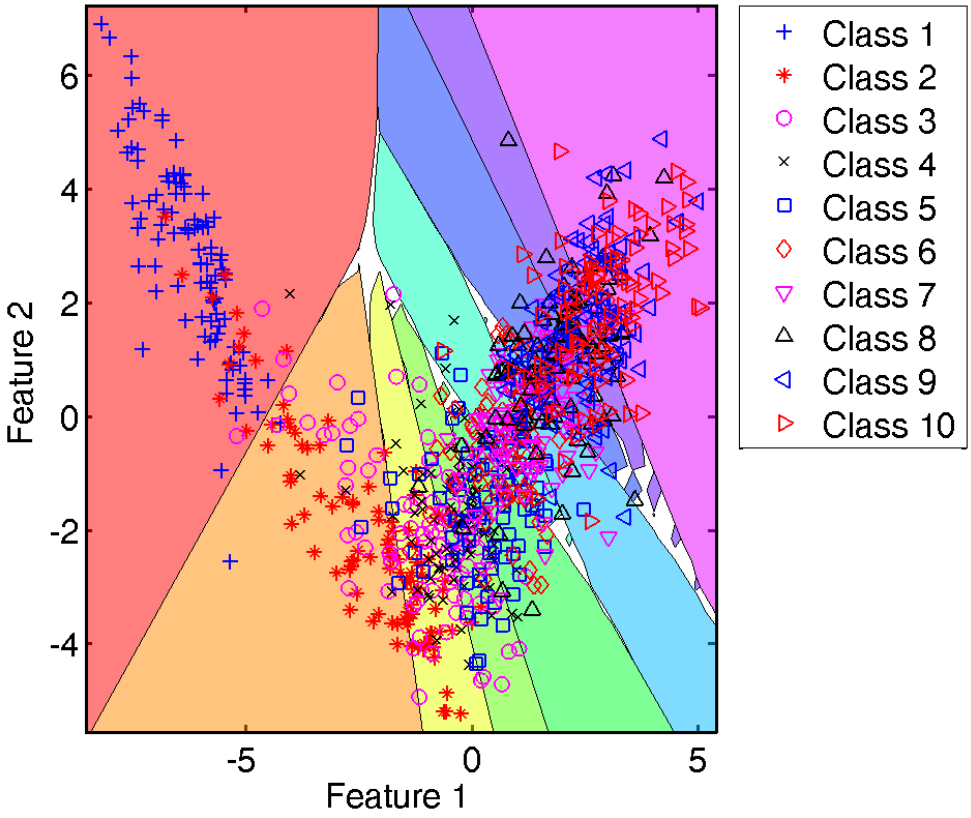


Figure 4.18: LDC decision regions, on the U-Poly-Plus-2D database

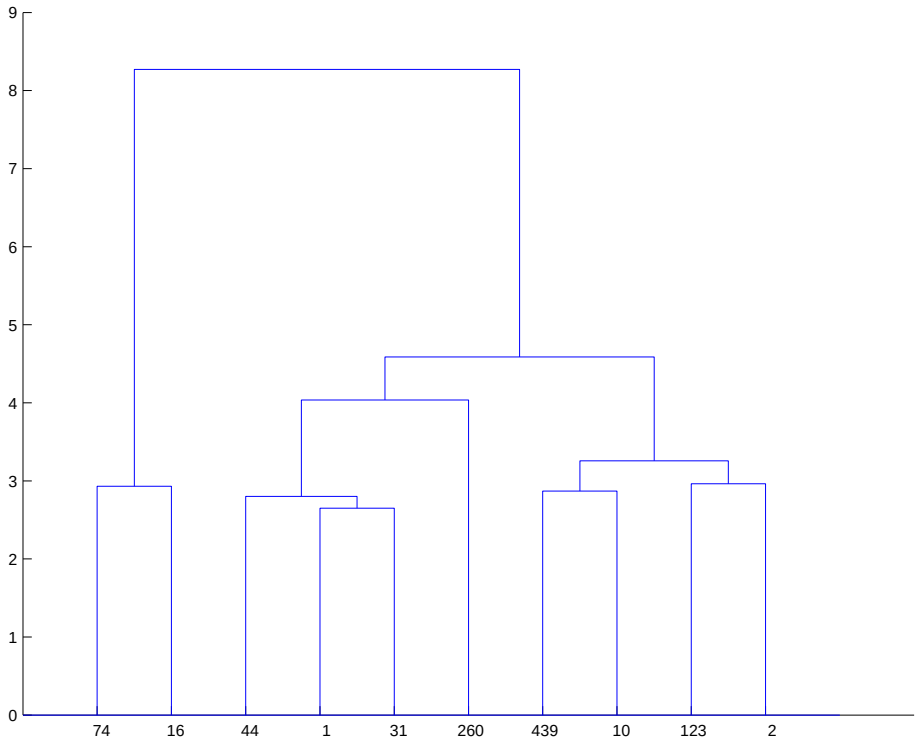


Figure 4.19: Hierarchical clustering dendrogram of the U-Poly-Plus-2D database

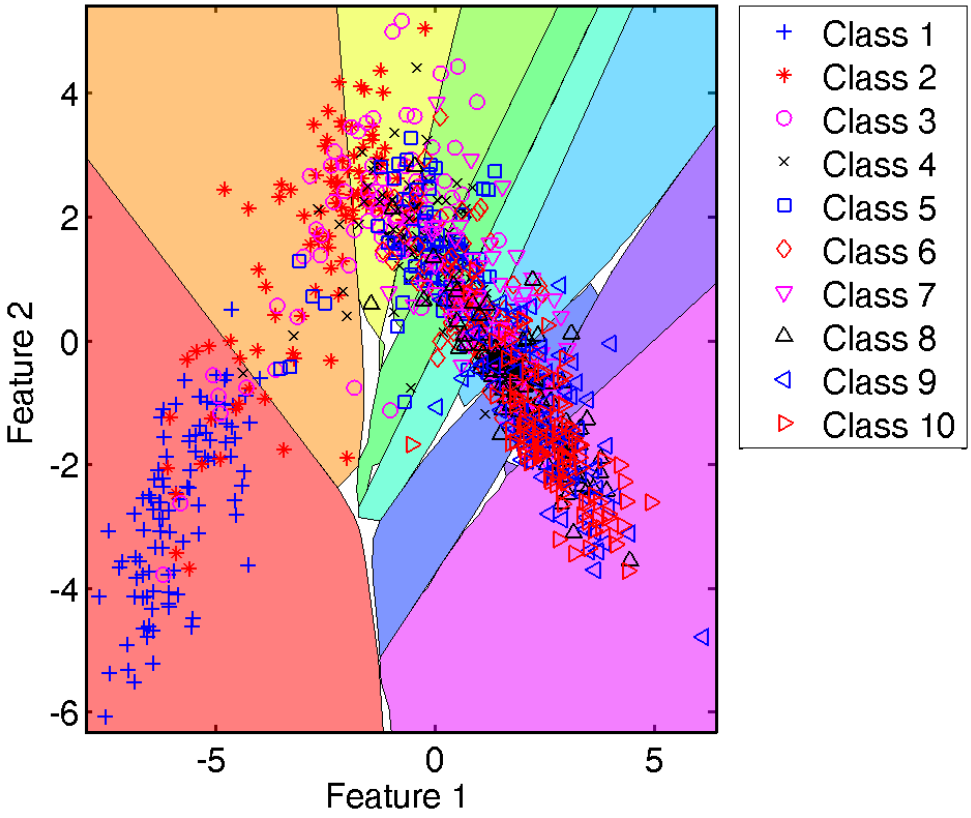


Figure 4.20: LDC decision regions, on the U-Poly-Minus-2D database

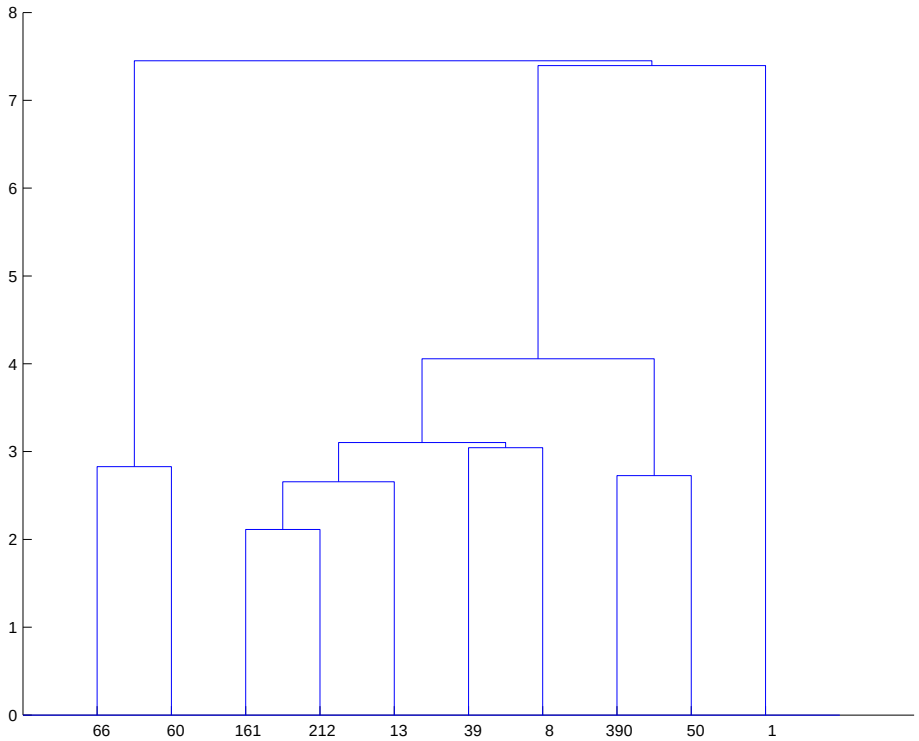


Figure 4.21: Hierarchical clustering dendrogram of the U-Poly-Minus-2D database

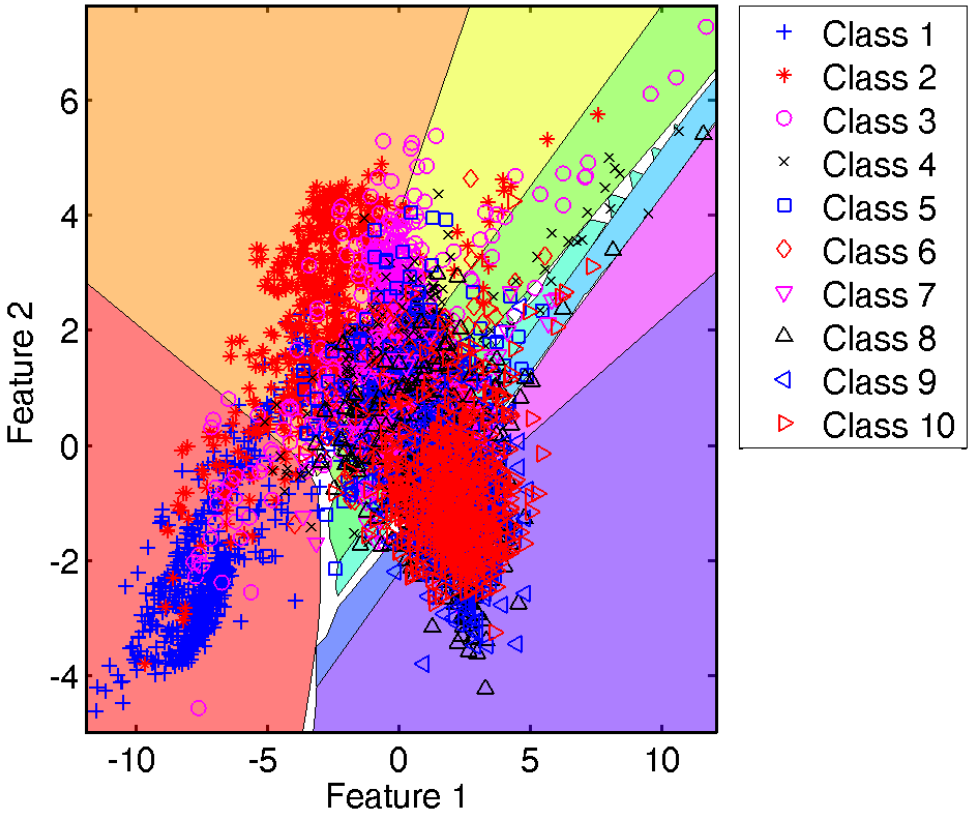


Figure 4.22: LDC decision regions, on the ALL-Poly-2D database

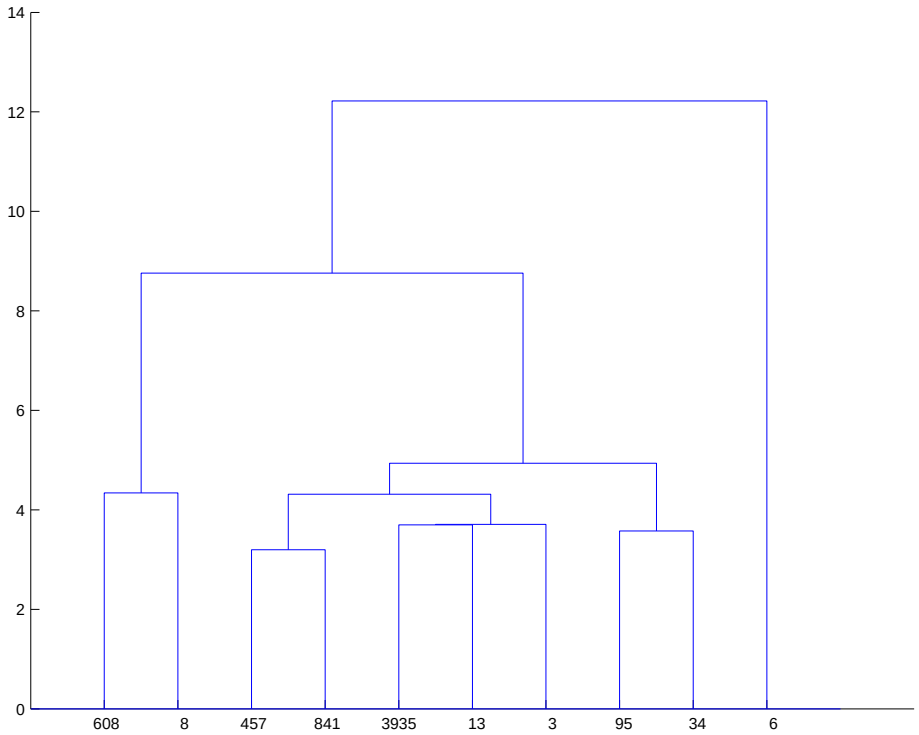


Figure 4.23: Hierarchical clustering dendrogram of the ALL-Poly-2D database

4.3.3 Meta-Measurement Feature Ranking

The meta-measurement values were computed on the synthetically generated datasets described in Chapter 4.2, then we have ranked each of the meta-measurements to see how much discrimination power it possesses. We have done this by assessing the ranking using two criterions:

1. Inter-Intra distance criterion;

2. 1-Nearest Neighbour (1NN) leave-one out classification accuracy.

Each criterion is applied in turn to each individual feature, then the values of the criterions is sorted in descending order to yield the ranking of the features in decreasing importance to the selected criterion.

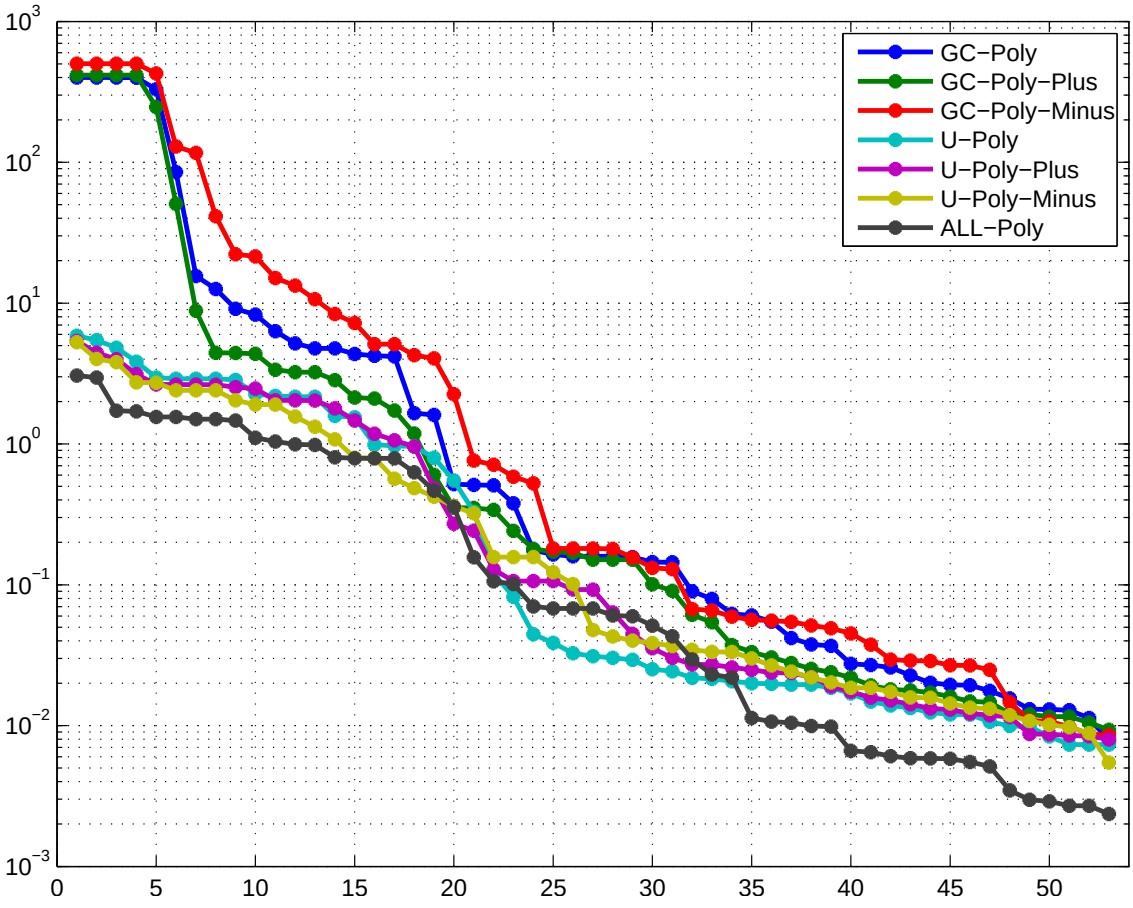


Figure 4.24: Feature efficiency of Meta-Measurement datasets given by INTER-INTRA criterion.

The first criterion we used to rank each individual meta-measurement feature is the Inter-Intra distance which is defined as the ratio between the scatter between the class means and the scatter within classes. A large value of this criterion indicates

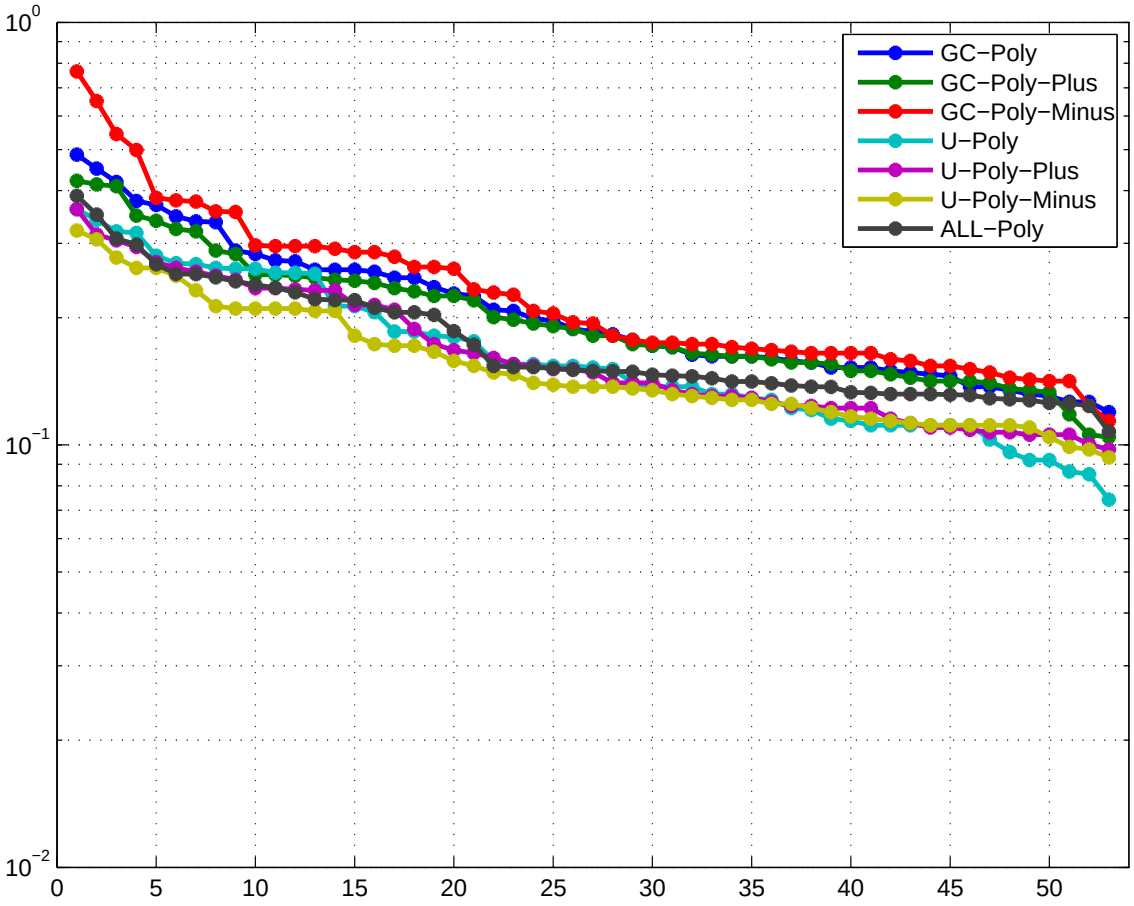


Figure 4.25: Feature efficiency of Meta-Measurement datasets given by 1NN criterion.

a better separation, the values of this criterion are plotted in Figure 4.24, where on the horizontal axis we have the feature number and on the vertical logarithmic axis we have the Inter-Intra criterion value.

The second criterion we used is the leave-one-out classification accuracy of the first Nearest Neighbour, the values obtained in this manner are plotted in Figure 4.25, where on the horizontal axis we have the feature number and on the vertical logarithmic axis we have the leave-one-out 1NN classification accuracy criterion value.

The actual ranking of the features is presented in an innovative way for each criterion in Figure 4.26 and Figure 4.27. We have chosen to depict graphically the way the ranking order of the features changes for 7 datasets.

Using the two sets of figures described above we can identify what feature produced a particular value of the criterion.

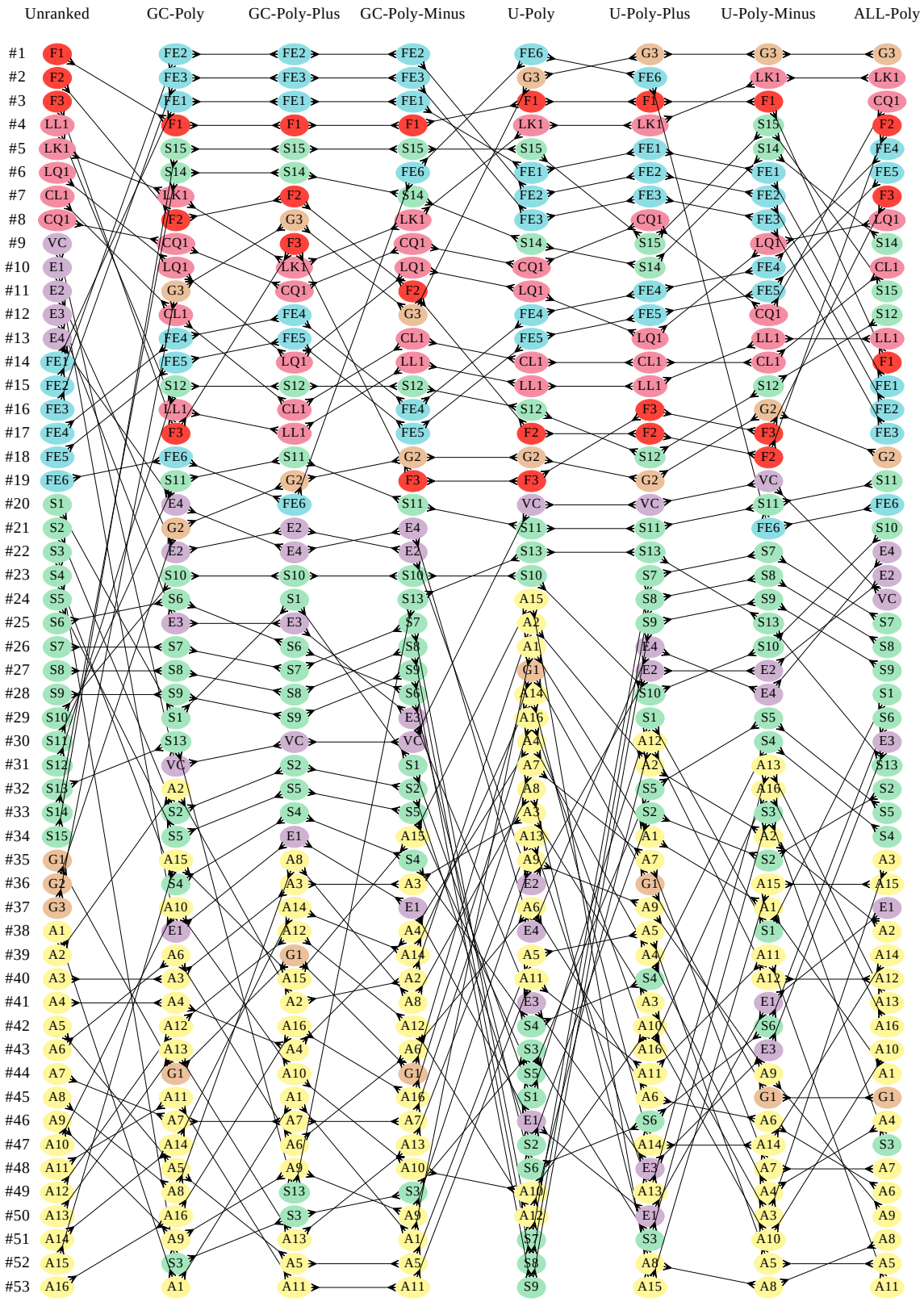


Figure 4.26: Feature ranking of Meta-Measurement datasets according to INTER-INTRA criterion.

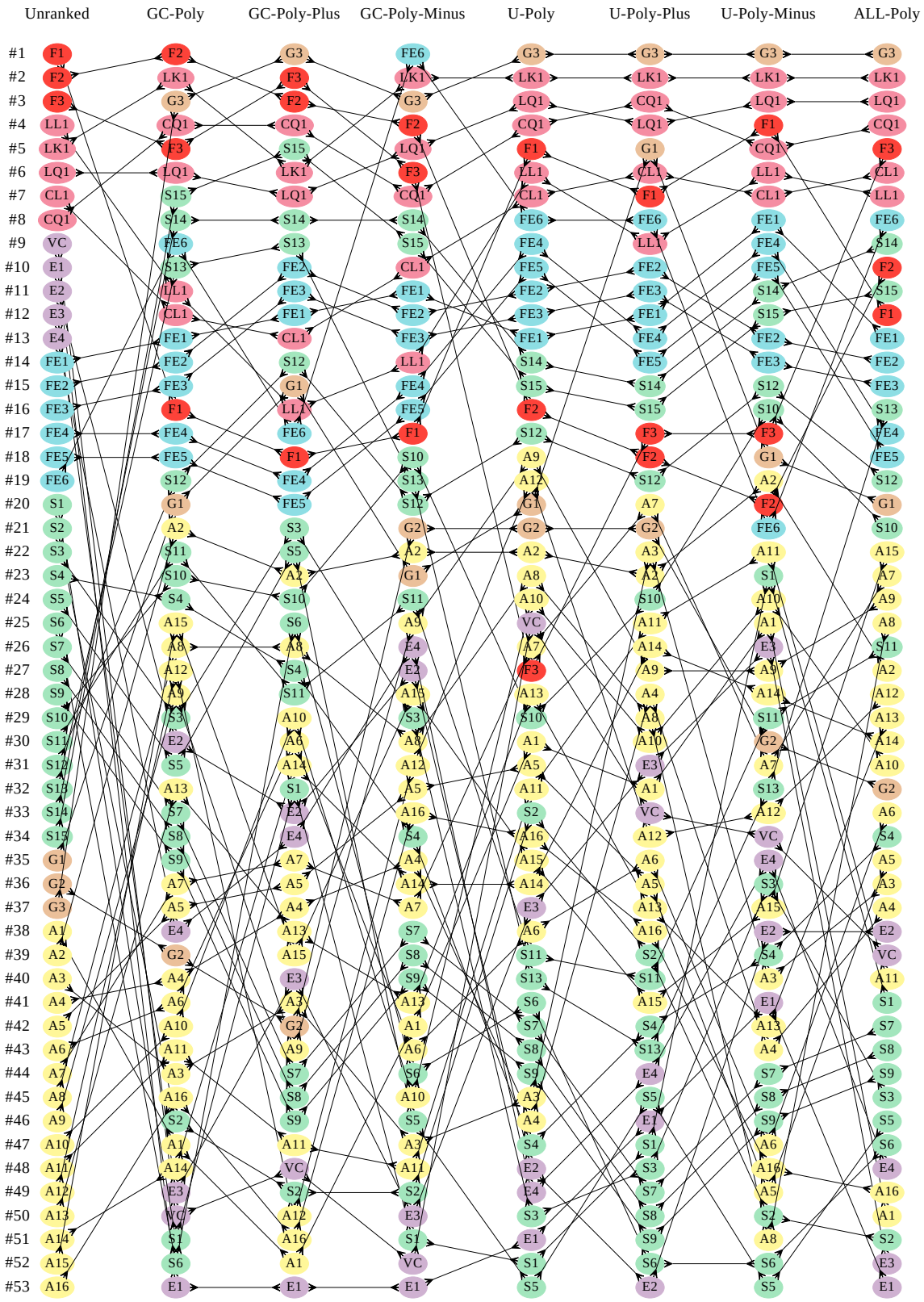


Figure 4.27: Feature ranking of Meta-Measurement datasets according to 1NN criterion.

4.3.4 Evaluation of Error Introduced During the Creation of Datasets

In this sub-section we are going to evaluate the discrepancy that was potentially produced when the synthetic datasets were created. We wanted to validate the datasets that we have generated. We needed to have 2-class datasets that have a polynomial boundary of a particular order, but how far away were the decision boundaries that can be estimated from the data points alone from the original generating polynomial? To answer this question we have measured the Mean Squared Error between the original generating polynomial and the decision boundary estimated by a KNN classifier, the results are aggregated per individual orders and shown in a boxplot format in Figure 4.28.

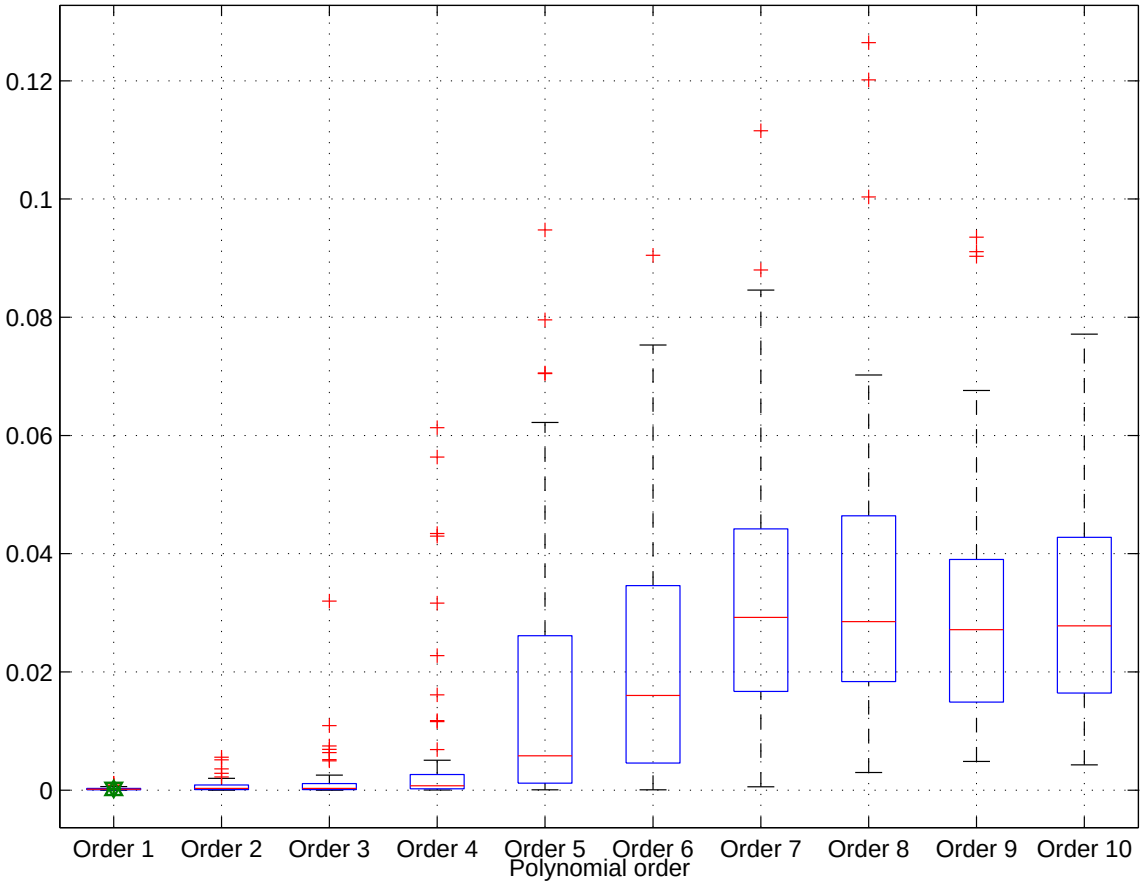


Figure 4.28: MSE between the original polynomial and the extracted decision boundary in the datasets

From this figure we can observe that the error introduced is below 0.01 for the first 4 polynomial orders that we generated and then gradually increases for higher orders.

4.4 Chapter Conclusions

In this chapter we have presented the realistic datasets and synthetic datasets that we will be using in the experimental sections of this thesis.

For the synthetic datasets we have shown two methods of generating datasets, namely the Gaussian Cloud method and the Uniformly Distributed method.

The presented data generation named Gaussian Clouds is novel method and is designed to alleviate the problems that occur when the separation is increased between the two classes. It does this by creating a number of clouds of Gaussian distribution along the perpendicular to the graph that is being used as a generating curve.

This chapter contains samples of datasets relating to each category of datasets that we have generated.

Chapter 5

Decision Boundary Approximation Using Polynomials

It is well known that, according to Stone-Weierstrass approximation theorem [72] and [76], polynomials can approximate as closely as desired any continuous function defined on an interval.

Decision boundaries in pattern recognition tasks can be viewed as discrete points sampled from a continuous function defined on an interval; hence, fitting a polynomial is a sensible choice to characterize the complexity of such a decision boundary.

Decision boundaries represent a are very important characteristic of every pattern classifier system. In fact, every classifier's main role is to estimate the boundary present between the two (or more) classes within a pattern recognition problem, to the lowest possible error. Therefore, examining the decision boundary formation process in a pattern classifier is of an upmost importance. In our work we have focused on the formation of the decision boundary within Feed-Forward Backpropagation NNs. Our interest in decision had beared its fruits in this current chapter and chapter 6

Within this chapter we are investigating the shape of the decision boundary formed between $c = 2$ classes of 2-dimensional feature values in a generic pattern recognition system, as described by Duda & Hart [19], in order to obtain a complexity measure of the pattern recognition problem at hand that will guide the automatic model selection process used in a later phase of automated classifier selection described in Chapter 7.

The proposed method of estimating the order of the polynomial using Meta-Measurements has surpassed the prediction accuracy of statistical methods that assess the goodness of fit as shown further in this chapter and in published conference proceedings [30].

5.1 Motivation

The motivation for our research arose from the desire to select a suitable classifier for the given classification problem or to choose a neural network architecture that improves the classification performance because it tries to detect the shape of the data points of the pattern recognition problem instead of relying on heuristics or rules of thumb based on the number of samples.

Our experimental investigation assesses the ability of several statistical methods suggested in the past surveys [80],[75], and our proposed machine learning approach to predict the order of the polynomial that can approximate the decision boundary formed between two classes of feature values. In order to do this the statistical methods are obtaining the decision boundary between the two classes from a K-Nearest-Neighbour classifier and then trying to estimate the optimal polynomial order to fit to the decision boundary to optimize the Least Squares Error of fitting. On the other hand, our proposed method calculates 53 meta-measurements from the feature values themselves as described in Chapter 3, then we tested several classifiers that were trained offline on these meta-measurements for predicting the order of the polynomial that can be fitted onto the decision boundary based on information learned from these measurements.

5.2 Polynomial Order Predicting Methods

We have investigated two major categories of order prediction methods:

1. Statistical prediction methods, where we have evaluated 13 prediction criteria with their associated algorithms for obtaining the prediction, presented in Sub-Section 5.2.1.

2. Machine Learning method having Meta-Measurements as features, in this category of order prediction methods we have investigated the performance of 6 classifiers on the Meta-Measurements presented in Sub-Section: 5.2.2.

In the following sub-sections we will describe these two categories in more detail.

5.2.1 Statistical Prediction Methods

The input to both methods are the feature values of sample points belonging to one dataset and the output is the prediction of the polynomial order that can estimate the decision boundary between the two classes of patterns in the dataset.

The statistical methods need to obtain a rough estimate of the decision boundary separating the two classes. This was achieved by employing a K-Nearest Neighbour classifier which is non-parametric and does not assume any distribution underlying in the data. The process of selecting the polynomial order to approximate the decision boundary is done on this rough estimate of the decision boundary.

The considered polynomial models to choose from have the following generic form:

$$y = \beta_0 + \beta_1 x + \cdots + \beta_p x^p + \epsilon$$

where $p = 1, , k$ with $k = 10$ and ϵ is the error term.

The boundary is obtained using a sequential sampling of the KNN mapping using the COUNTOURC MatLab function.

We have implemented the following criterions for predicting the order of the polynomial that can be fitted onto the decision boundary:

1. Kolmogorov-Smirnov pairwise test on the residuals
2. "Biggest Jump" algorithm on SSE
3. "Biggest Jump" algorithm on the Standard Deviation
4. Predicted polynomial order from the POLYDEG.M function created by Damien Garcia, [29]
5. Minimum AIC
6. Runs Test, Maximum P-Value
7. Chi2 H Value, Algorithm chooses first occurrence of the value 1 (i.e. where the null hypothesis can be rejected)
8. Chi2 Minimum P-Value
9. K-S H Value, (Same algorithm as 7)
10. K-S Minimum P-Value
11. Hardcoded threshold on the Standard Deviation, set at 0.10
12. T-Test on the SSE, that tries to determine the outlier starting from the highest fitted polynomial order towards the lowest polynomial order
13. T-Test on the Standard Deviation (same algorithm as 12)

13 methods used to predict the order of the polynomial. Not all of them are independent of one another.

The statistical model selection criterions that we implemented are the following: Mean Squared Error (STAT MSE)

$$MSE(p) = (SSE(p))/(n - (p + 1))$$

Mallows Cp (STAT MCp) [46]

$$MCp = (SSE(p))/(MSE(p)) - (n - (p + 1))$$

Akaikes Information Criterion (STAT AIC) [10]

$$AIC(p) = ln(SSE(p)) + (p + 1)(2/n)$$

Bayesian Information Criterion (STAT BIC) [69]

$$BIC(p) = ln(SSE(p)) + (p + 1)(ln(n)/n)$$

For all the statistical methods above, $p =$ is the order of the polynomial that will be tested, which is a positive integer in the range p=1, , 10; $k =$ is the largest order of the polynomial to be fitted; $n =$ represents the number of samples;

SSE(p) = residual sum of squares for the polynomial model of order p.

The algorithm for deciding upon what order to predict was the same for all 4 implemented methods, namely choose the order of the polynomial that minimizes the 4 corresponding criterion values.

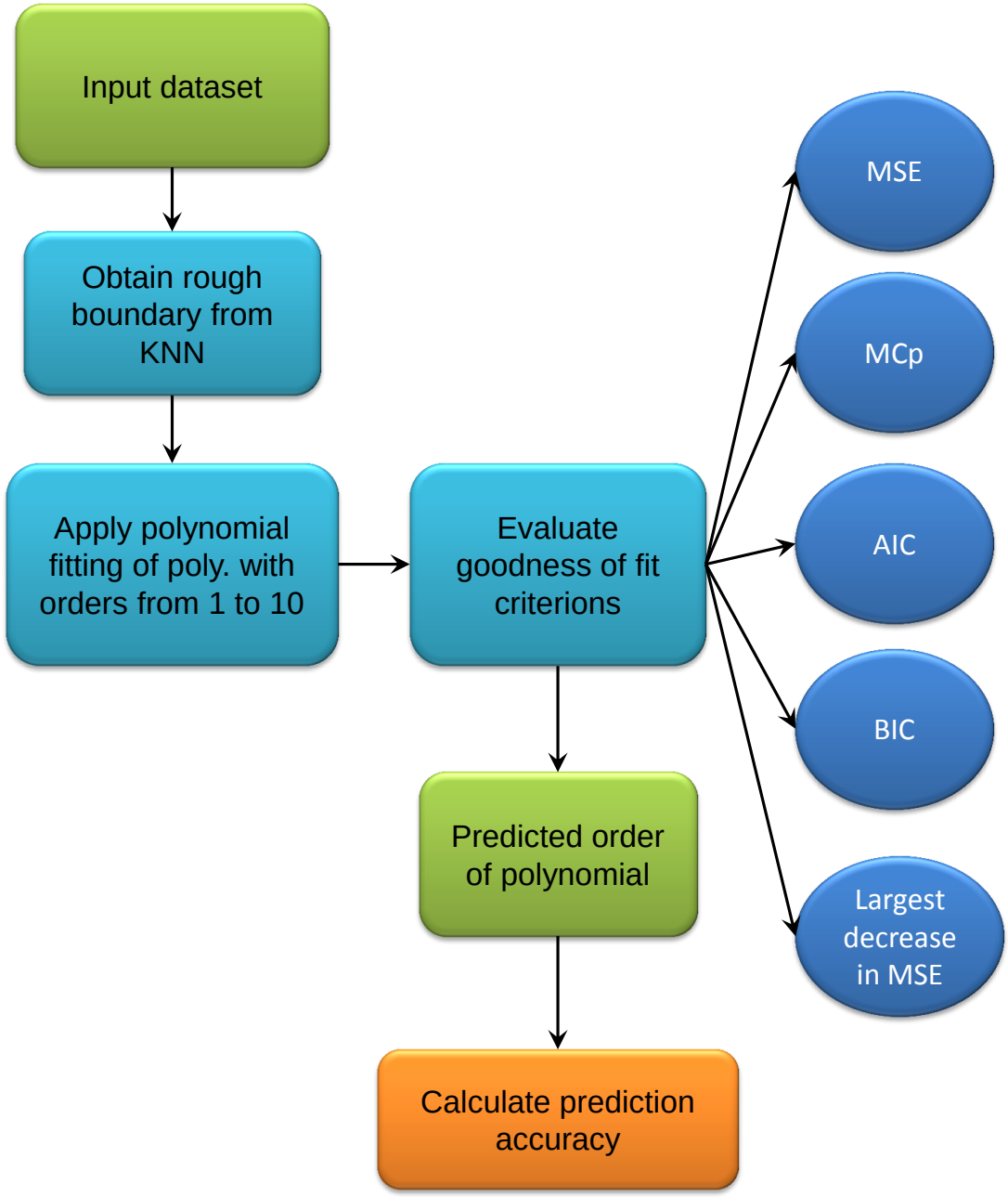


Figure 5.1: Overview of the system to test the prediction accuracy of the statistical methods of predicting the polynomial order.

5.2.2 Meta-Measurements Method

Prediction method using a classifier trained on the computed Meta-Measurements

This method takes a different approach to predicting the order of the polynomial to be approximated to the decision boundary of the two classes within a dataset then the statistical methods. It calculates all the 34 meta-measurements described in Section 3 on the datasets generated artificially as shown in Section 4.2. These computed meta-measurements form the training set of a classifier. The trained classifier is then used on another database (called the testing set) of another 6,000 datasets that are generated by the same function as the first, only this time we used a different initial seed of the random number generator, hence the two databases are distinct but are generated by the same principles, and the latter will contain data that has not been presented to classifier in its training stage. In our investigation we have used 6 different classifiers, namely:

1. Linear classifier based on normal densities (LDC);
2. Quadratic classifier based on normal densities (QDC);
3. K-nearest neighbour classifier (KNNC), where the parameter K is optimized to minimize the leave one out error;
4. 1-nearest neighbour (1NNC);
5. 3-nearest neighbour (3NNC);
6. Fishers minimum least square linear classifier (FISHERC).

Empirically the quadratic classifier (QDC), proved to be the best suited classifier for the problem at hand.

We have evaluated the performance of the 4 statistical methods of predicting the order of the polynomial and our proposed method using meta-measurements tested on two different scenarios, each of which has different interpretations of correct classification counts.

In the first scenario the accuracy is calculated by counting only the exact matches between the predictions generated by each method against the true order of the polynomial.

Whereas, in the second scenario, the accuracy is calculated by counting the exact matches between the predictions and the true order, but also allows for order over-estimation by including as a correct prediction where the order of the polynomial is greater than the true order.

All the experiments were carried out by writing functions and scripts for the MATLAB 2012 programming environment and we have also used portions from the academic version of PRTools [20].

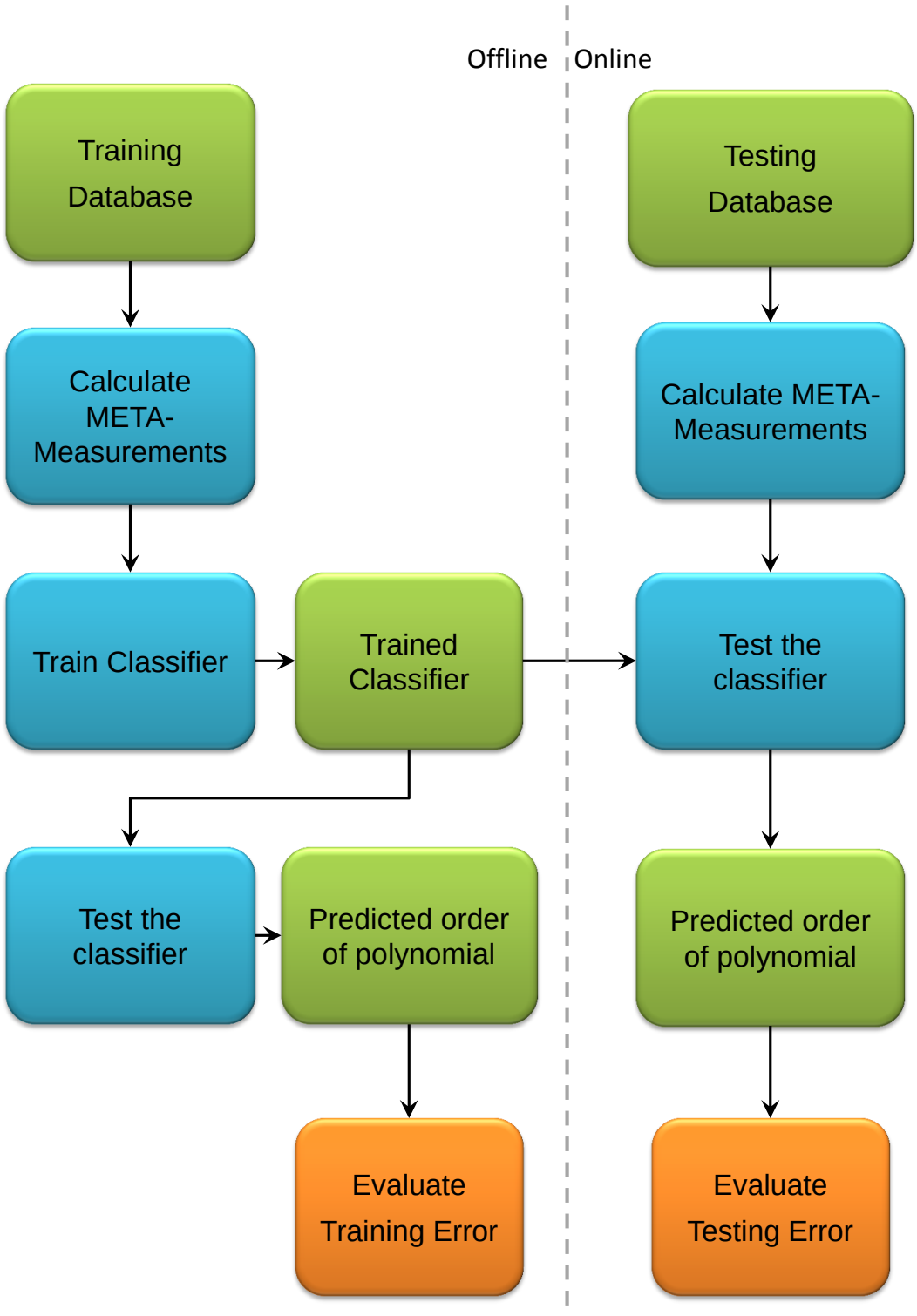


Figure 5.2: Overview of the system to test the prediction accuracy of the Meta-measurement method of predicting the polynomial order.

5.3 Comparative Results

The following pages compare the experimental results of predicting the order of the polynomial to be best fitted on the decision boundary that is present in our synthetically generated dataset.

Table 5.1 presents the prediction accuracy of statistical methods of estimating the goodness of fit using 5 criterions across two evaluation scenarios (detailed in previous section).

For scenario one, the best prediction accuracy is achieved by the Largest Decrease in Mean Standard Error (L.I. MSE) with 17.85% and a standard deviation of 0.41% with the other 4 selection criterion methods all having an accuracy close or below 10%. Given that all these experiments needed to predict the correct from a range of 1 to 10, having a accuracy of 10% is as good as random guessing the order. Having an accuracy lower than 10% means that the given selection criterions are worse at predicting the correct order than random guessing it. The low accuracy of exact estimating exactly the goodness of fit was an unexpected find of our research.

For scenario two, the prediction accuracy increases dramatically. The best selection criterion is now the Minimum Mean Standard Error (MIN MSE) criterion with 69.99% across the 10 cross-validation runs, which represents a more than seven fold increase in accuracy as compared to scenario one. This scenario is more lenient by accepting an over-estimation of the predicted order, which is fine in most cases but it leads to selecting a too complicated model for the given dataset.

The results summarized in table 5.1 are expanded in Figure 5.3 to show a graphical box-plot of the obtained accuracies and in Table 5.3 that shows the non-parametric evaluation of the distribution of accuracy values resulting from the 10-fold cross-validation for scenario one.

The results summarized in table 5.1 are expanded in Figure 5.4 to show a graphical box-plot of the obtained accuracies and in Table 5.4 that shows the non-parametric evaluation of the distribution of accuracy values resulting from the 10-fold cross-validation for scenario two.

From the comparison of the results obtained by the statistical methods of predicting the order of the polynomial using we can confirm that these methods cannot exactly predict the correct order of the polynomial at most they can predict with an accuracy slightly greater than random guessing. Furthermore, when assessed on predicting the correct order or over-estimating they achieve moderate results (e.g. 69.75%).

Table 5.2 shows the results of the polynomial order prediction method using meta-measurements and 5 trained classifiers over the same two scenarios (the exact

prediction scenario and the exact prediction with overestimation scenario) but using 3 sets of meta-measurements:

- MM Group 1 - The complexity Meta-Measurements (MM) with indices between 1 and 34;
- MM Group 2 - The graphical angle profile Meta-Measurements (MM) with indices between 38 and 53;
- MM Group 3 - All the Meta-Measurements (MM) with indices between 1 and 53;

These 3 groups of Meta-Measurements were selected in order to find which group is better suited to do the prediction and whether the angle profile Meta Measurements are increasing the prediction accuracy when used together with the complexity measurements.

From table 5.2 we observe that the complexity Meta-Measurements in group 1, are achieving the best result on their own and the addition of the Meta-Measurements in Group 2 lowers the prediction accuracy to the levels of the prediction accuracy of Group 2 alone. Therefore, we shouldn't use all Meta-Measurements, Group 1 is achieving 93% in scenario 1 of exact classification and 96.62% in scenario 2, where overestimating is allowed.

The difference between the best statistical accuracy 17.85% in scenario 1 and 69.75% in scenario 2, is astonishing when compared to the best results achieved by the Complexity Meta-Measurements which achieve 93% and 96.63% in their respective scenarios.

The results summarized in table 5.2 are expanded in Figure 5.4 to show a graphical box-plot of the obtained accuracies and in Table 5.4 that shows the non-parametric evaluation of the distribution of accuracy values resulting from the 10-fold cross-validation for scenario two.

The results summarized in table 5.2 are expanded in Figure 5.4 to show a graphical box-plot of the obtained accuracies and in Table 5.4 that shows the non-parametric evaluation of the distribution of accuracy values resulting from the 10-fold cross-validation for scenario two.

The results summarized in table 5.2 are expanded in Figure 5.4 to show a graphical box-plot of the obtained accuracies and in Table 5.4 that shows the non-parametric evaluation of the distribution of accuracy values resulting from the 10-fold cross-validation for scenario two.

Table 5.1: Summary of polynomial order prediction accuracies using Statistical methods evaluated in two classification scenarios.

	MIN MSE	MIN MCp	AIC	MIN BIC	L.D. MSE
Statistical methods					
Scenario #1 Average:	9.77%	9.79%	8.74%	7.76%	17.85%
Standard deviation:	±0.38	±0.35	±0.38	±0.34	±0.41
Scenario #2 Average:	69.99%	69.75%	56.85%	40.14%	30.28%
Standard deviation:	±0.47	±0.48	±0.71	±0.52	±0.34

Table 5.2: Summary of polynomial order prediction accuracies using Meta-Measurement methods evaluated in two classification scenarios.

	Bayes Normal 1	Bayes Normal 2	1-NN Classifier	3-NN Classifier	Fisher Classifier
Complexity Meta Measurements - with indices from 1 to 34					
Scenario #1 Average	40.06%	37.06%	93.04%	77.67%	38.91%
Standard Deviation	(±0.51)	(±0.40)	(±0.32)	(±0.33)	(±0.59)
Scenario #2 Average	73.78%	87.17%	96.62%	87.84%	68.81%
Standard Deviation	(±0.70)	(±0.70)	(±0.27)	(±0.44)	(±0.81)
Graphical Meta Measurements - with indices from 38 to 53					
Scenario #1 Average	23.94%	40.69%	82.29%	56.70%	24.88%
Standard Deviation	(±0.42)	(±0.72)	(±0.47)	(±0.37)	(±0.44)
Scenario #2 Average	48.23%	59.91%	89.11%	68.91%	46.97%
Standard Deviation	(±0.62)	(±3.22)	(±0.44)	(±0.59)	(±0.59)
All Meta Measurements - with indices from 1 to 53					
Scenario #1 Average	41.62%	46.45%	82.44%	56.88%	40.56%
Standard Deviation	(±0.45)	(±2.81)	(±0.53)	(±0.43)	(±0.56)
Scenario #2 Average	72.02%	66.98%	88.67%	68.02%	65.70%
Standard Deviation	(±0.74)	(±5.05)	(±0.34)	(±0.40)	(±0.65)

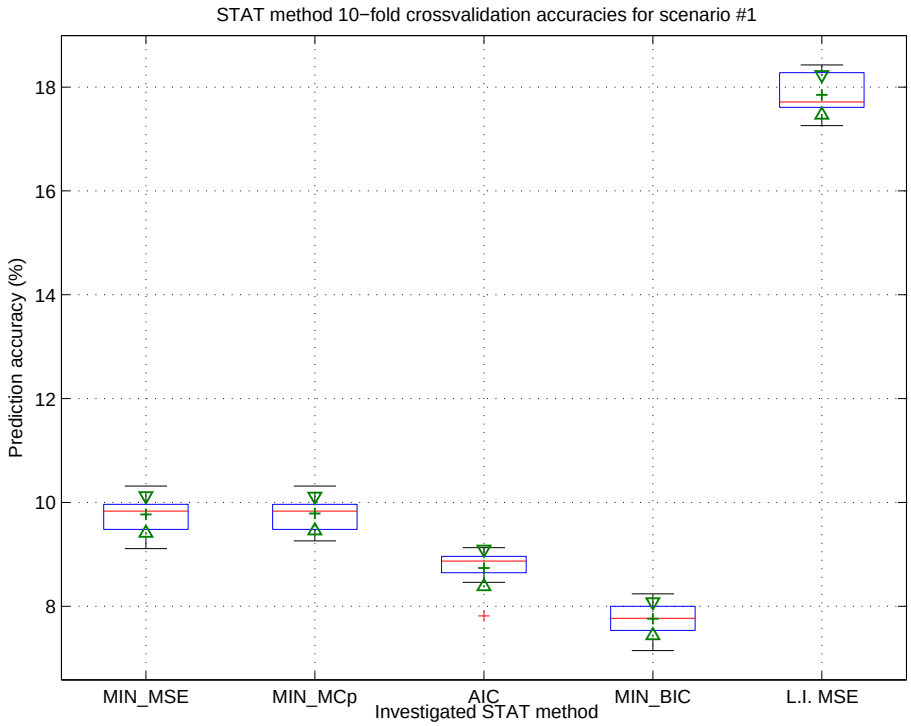


Figure 5.3: Polynomial order prediction accuracy boxplot of the Statistical methods, evaluated on scenario #1.

Table 5.3: Non-parametric evaluation of polynomial order prediction accuracy using Statistical methods, evaluated on scenario #1.

	MIN MSE	MIN MCp	AIC	MIN BIC	L.D. MSE
Average value	9.77	9.79	8.74	7.76	17.85
Standard deviation	±0.38	±0.35	±0.38	±0.34	±0.41
Minimum	9.11	9.26	7.81	7.15	17.26
Lower whisker	9.11	9.26	8.46	7.15	17.26
Lower 25% percentile	9.48	9.48	8.65	7.54	17.61
Median	9.83	9.83	8.87	7.77	17.71
Upper 75% percentile	9.96	9.96	8.96	8.00	18.28
Upper whisker	10.31	10.31	9.13	8.24	18.43
Maximum	10.31	10.31	9.13	8.24	18.43
Percentage in Q1	20.0%	20.0%	10.0%	20.0%	20.0%
Percentage in Q2	50.0%	50.0%	50.0%	50.0%	50.0%
Percentage in Q3	30.0%	30.0%	30.0%	30.0%	30.0%
Percentage of outliers	0.0%	0.0%	10.0%	0.0%	0.0%
Number of outliers	0	0	1	0	0
Number of samples	10	10	10	10	10

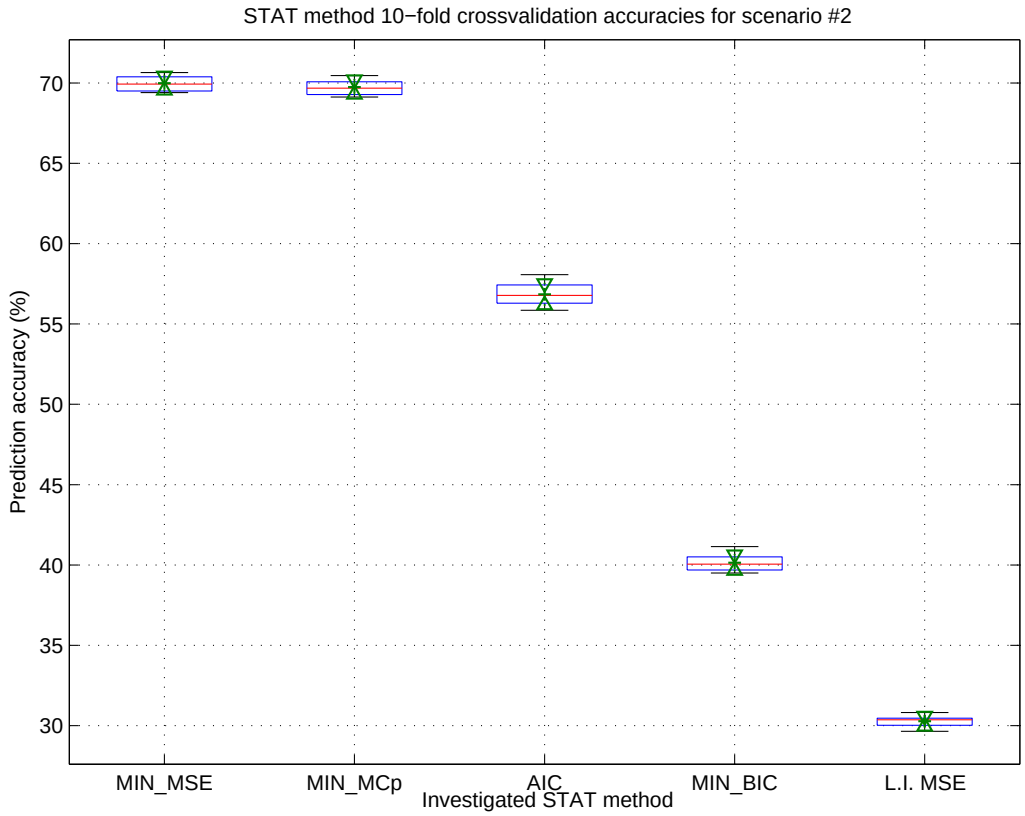


Figure 5.4: Polynomial order prediction accuracy boxplot of the Statistical methods, evaluated on scenario #2.

Table 5.4: Non-parametric evaluation of polynomial order prediction accuracy using Statistical methods, evaluated on scenario #2.

	MIN MSE	MIN MCp	AIC	MIN BIC	L.D. MSE
Average value	69.99	69.75	56.85	40.14	30.28
Standard deviation	±0.47	±0.48	±0.71	±0.52	±0.34
Minimum	69.41	69.13	55.85	39.50	29.65
Lower whisker	69.41	69.13	55.85	39.50	29.65
Lower 25% percentile	69.50	69.28	56.30	39.69	30.02
Median	69.94	69.69	56.78	40.05	30.36
Upper 75% percentile	70.39	70.07	57.43	40.50	30.46
Upper whisker	70.65	70.46	58.07	41.15	30.81
Maximum	70.65	70.46	58.07	41.15	30.81
Percentage in Q1	20.0%	20.0%	20.0%	10.0%	20.0%
Percentage in Q2	50.0%	50.0%	50.0%	60.0%	50.0%
Percentage in Q3	30.0%	30.0%	30.0%	30.0%	30.0%
Percentage of outliers	0.0%	0.0%	0.0%	0.0%	0.0%
Number of outliers	0	0	0	0	0
Number of samples	10	10	10	10	10

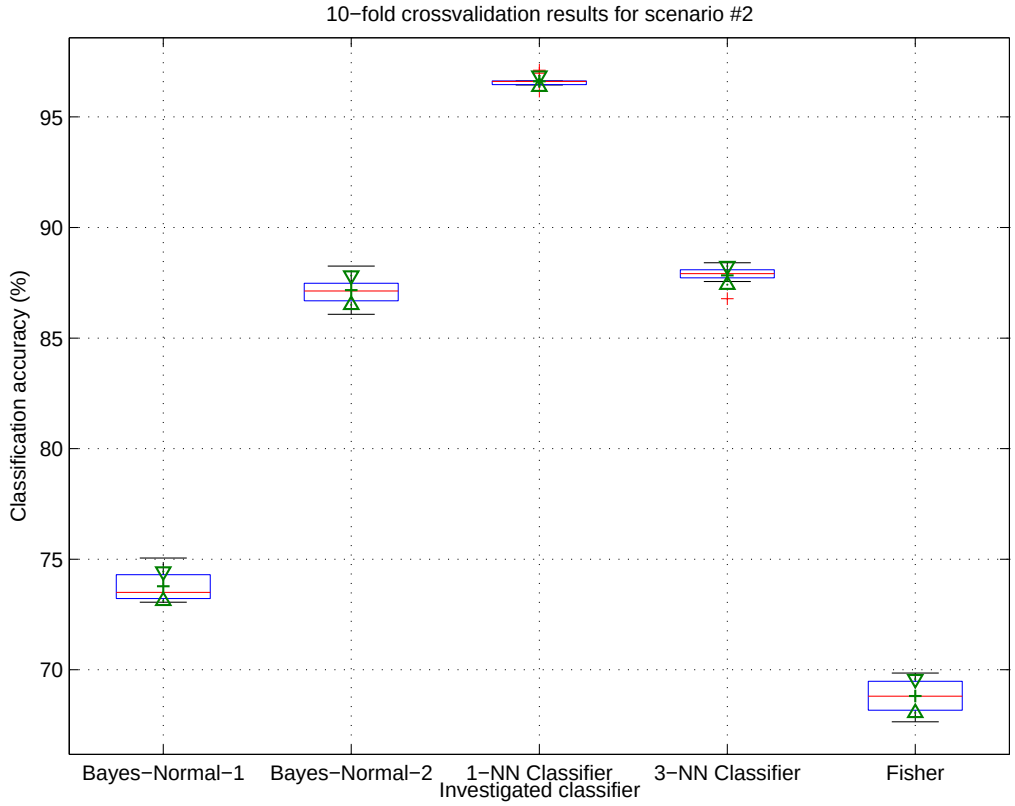


Figure 5.5: Polynomial order prediction accuracy boxplot of the Meta-Measurements Group 1, evaluated on scenario 2.

Table 5.5: Non-parametric evaluation of polynomial order prediction accuracy using Meta-Measurements Group 1, evaluated on scenario #2.

	Bayes Normal 1	Bayes Normal 2	1-NN Classifier	3-NN Classifier	Fisher Classifier
Average value	73.78	87.17	96.62	87.84	68.81
Standard deviation	±0.70	±0.70	±0.27	±0.44	±0.81
Minimum	73.06	86.07	96.17	86.78	67.65
Lower whisker	73.06	86.07	96.44	87.56	67.65
Lower 25% percentile	73.22	86.69	96.46	87.72	68.17
Median	73.50	87.13	96.60	87.92	68.81
Upper 75% percentile	74.30	87.48	96.63	88.09	69.48
Upper whisker	75.06	88.26	96.63	88.41	69.85
Maximum	75.06	88.26	97.11	88.41	69.85
Percentage in Q1	20.0%	20.0%	10.0%	10.0%	20.0%
Percentage in Q2	50.0%	50.0%	50.0%	50.0%	50.0%
Percentage in Q3	30.0%	30.0%	10.0%	30.0%	30.0%
Percentage of outliers	0.0%	0.0%	30.0%	10.0%	0.0%
Number of outliers	0	0	3	1	0
Number of samples	10	10	10	10	10

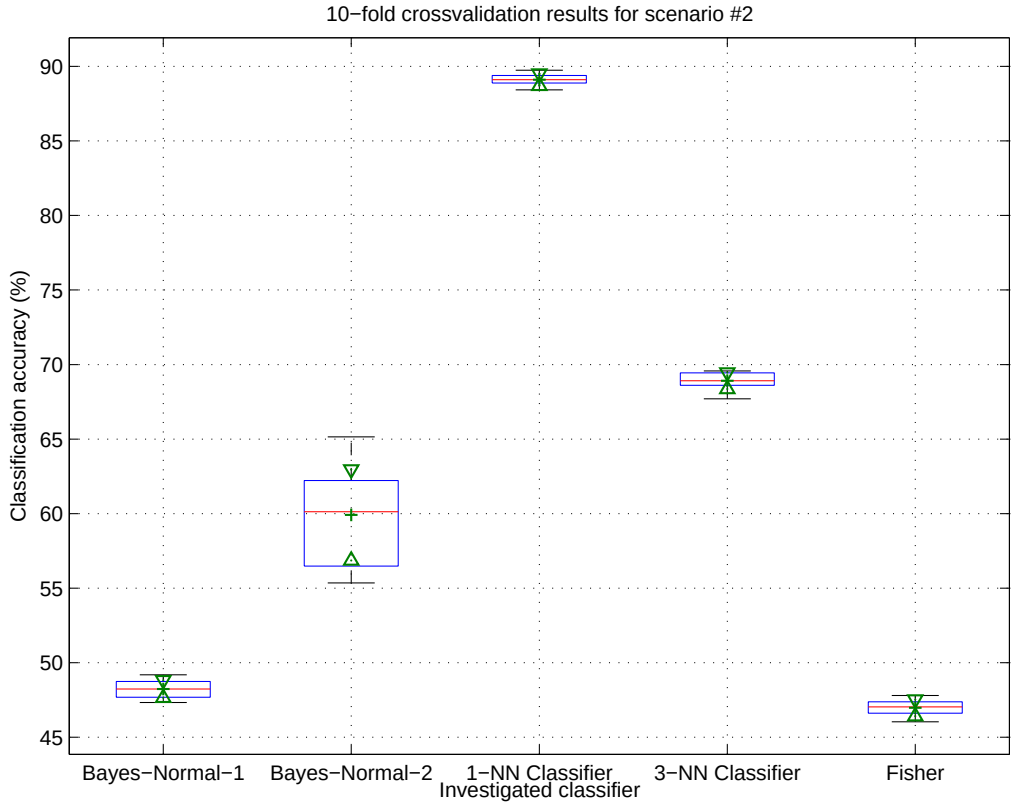


Figure 5.6: Polynomial order prediction accuracy boxplot of the Meta-Measurements Group 2, evaluated on scenario 2.

Table 5.6: Non-parametric evaluation of polynomial order prediction accuracy using Meta-Measurements Group 2, evaluated on scenario #2.

	Bayes Normal 1	Bayes Normal 2	1-NN Classifier	3-NN Classifier	Fisher Classifier
Average value	48.23	59.91	89.11	68.91	46.97
Standard deviation	±0.62	±3.22	±0.44	±0.59	±0.59
Minimum	47.33	55.35	88.43	67.70	46.04
Lower whisker	47.33	55.35	88.43	67.70	46.04
Lower 25% percentile	47.69	56.48	88.89	68.61	46.61
Median	48.23	60.12	89.11	68.92	47.03
Upper 75% percentile	48.74	62.22	89.39	69.44	47.37
Upper whisker	49.19	65.15	89.74	69.57	47.80
Maximum	49.19	65.15	89.74	69.57	47.80
Percentage in Q1	20.0%	20.0%	20.0%	20.0%	20.0%
Percentage in Q2	50.0%	50.0%	50.0%	50.0%	50.0%
Percentage in Q3	30.0%	30.0%	30.0%	30.0%	30.0%
Percentage of outliers	0.0%	0.0%	0.0%	0.0%	0.0%
Number of outliers	0	0	0	0	0
Number of samples	10	10	10	10	10

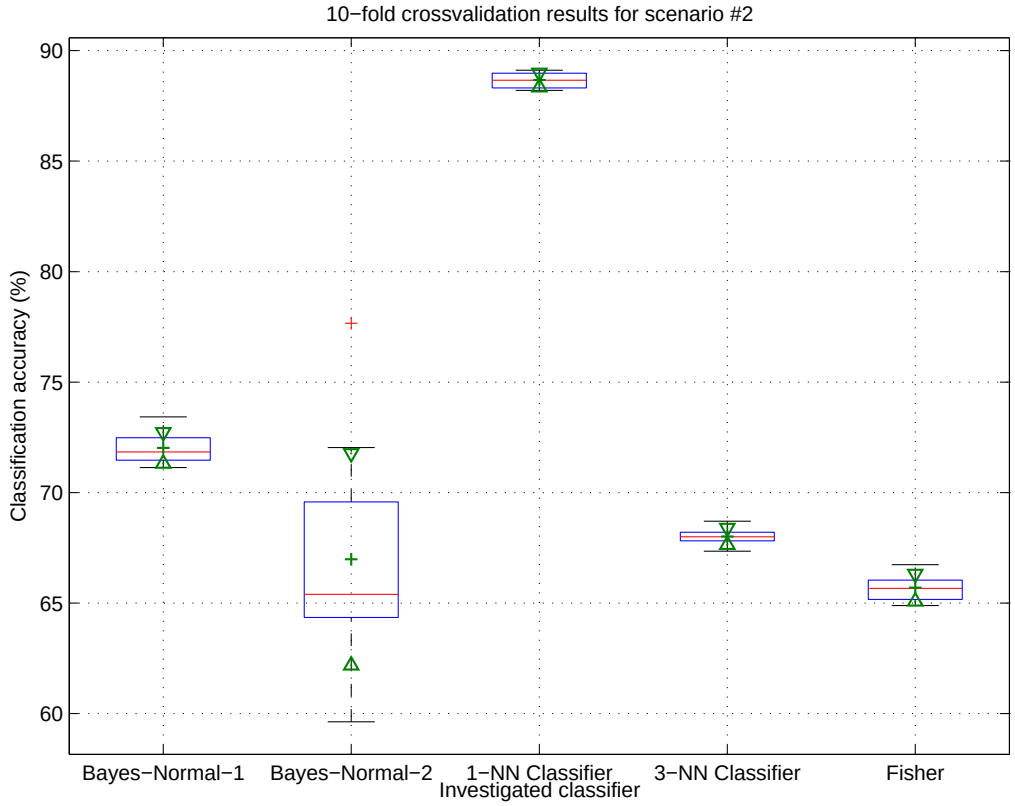


Figure 5.7: Polynomial order prediction accuracy boxplot of the Meta-Measurements Group 3 (ALL), evaluated on scenario 2.

Table 5.7: Non-parametric evaluation of polynomial order prediction accuracy using Meta-Measurements Group 3, evaluated on scenario #2.

	Bayes Normal 1	Bayes Normal 2	1-NN Classifier	3-NN Classifier	Fisher Classifier
Average value	72.02	66.98	88.67	68.02	65.70
Standard deviation	±0.74	±5.05	±0.34	±0.40	±0.65
Minimum	71.13	59.63	88.20	67.35	64.89
Lower whisker	71.13	59.63	88.20	67.35	64.89
Lower 25% percentile	71.46	64.35	88.31	67.81	65.17
Median	71.84	65.40	88.66	68.00	65.67
Upper 75% percentile	72.48	69.57	88.98	68.20	66.04
Upper whisker	73.43	72.04	89.11	68.70	66.74
Maximum	73.43	77.67	89.11	68.70	66.74
Percentage in Q1	20.0%	20.0%	20.0%	20.0%	20.0%
Percentage in Q2	50.0%	50.0%	50.0%	50.0%	50.0%
Percentage in Q3	30.0%	20.0%	30.0%	30.0%	30.0%
Percentage of outliers	0.0%	10.0%	0.0%	0.0%	0.0%
Number of outliers	0	1	0	0	0
Number of samples	10	10	10	10	10

5.4 Chapter Conclusions

This chapter presented the evaluation of statistical methods for estimating the order of the polynomial that can approximate efficiently the decision boundary of two dimensional, two class pattern recognition problems and compared these with our proposed method of accomplishing the same task using a machine learning approach and the proposed meta-measurements.

The advantages and disadvantages of the best statistical method and our proposed method were highlighted with experimental results.

Our proposed method achieved an accuracy of 96.62% as opposed to 69.99% when accepting that over-estimating the order of the polynomial that is to be predict is acceptable. Which, for most of the cases it is, but it does lead to more complicated models, that can over-fit or have convergence problems. Therefore, it is desirable to aim for exact prediction of the polynomial order. In this scenario our proposed achieved 93% accuracy as opposed to just under 18%, which is not substantially more than random guessing.

In the light of these results, we believe that our proposed method has a large achievement in advancing the field.

The presented approach can be applied to pattern recognition problems with $C > 2$ number of classes by using task decomposition. However, further research needs to address feature spaces with higher number of dimensions.

The work presented in this chapter will be used throught the rest of the work, namely in Chapter 6, where we investingate further the decision boundary of Neural Networks and propose a novel method of adapting the initial connection weights of the network to improve classification error.

Furthermore, the Meta-Measurement method proposed in this chapter will be used in Chapter 7, where we construct Modular Neural Networks for which the number of hidden neurons will be predicted with this method.

Chapter 6

Neural Network Weight Adaptation

In this chapter we shall talk about a technique developed by the authors, to improve the classification accuracy and reduce the number of training epochs of the feed-forward back-propagation neural networks when trained on data that has a decision boundary that is polynomial or it can be approximated to a polynomial.

The decision boundary that a back-propagation feed forward neural network can produce is first investigated analytically and validated graphically. The decision boundary is dependent on the number of neurons which are present in the hidden layer of the network. From this investigation we have arrived to the idea tune the initial starting weights of the network and found that this would have several desirable outcomes compared to the control scenario when the initial weights are initialized with random values. This conclusion has been reached after comparing the proposed weight adaptation scheme to a control experiment.

We found that the tuning of the initial values of the connection weight of the neural network according to our proposed method has the follows benefits:

- The classification error rate is reduced;
- The number of training epochs needed before the training algorithm meets the stopping criteria is decreased.

This has been achieved by tuning the initial values of the weights of the neurons so that they reproduce the location of tangent lines to the polynomial graph at the inflection points of the graph where the second order derivative becomes zero and also the tangents to the graphs of the polynomial at the boundaries of the considered

graphing ranges (e.g. [0, 1]). These tangent lines will be referred to later in this chapter as *"characteristic tangent lines"*.

6.1 Decision Boundaries of Neural Networks

In order to investigate the decision boundary produced by a NN, we first have to take account of the factors that can influence the shape of the decision boundary, these are generically named the architecture of the NN.

The generic architecture of a neural network is defined by:

1. The number of input nodes
2. The number of layers
3. The number of hidden nodes
4. The number of output nodes
5. The presence/absence of feedback loops
6. The "connectedness" of the network (fully or partially connected)
7. The topological/hierarchical organisation
8. The type of activation function of the nodes

We are concerned with just the number of hidden neurons in the hidden, middle, layer of the neural network. While we assume constant values to all the other parameters of the network architecture.

The neural networks we for which we are investigating the decision boundaries have the following characteristics:

1. Two input neurons nodes, since the classification problems we are trying to solve have two dimmensions;
2. Have an input, output and a single hidden layer;
3. A variable number of hidden nodes, ranging from 1 to 10;
4. Two output nodes, since the data we are trying to classify has two classes;
5. The networks we investigate do not have feedback loops (they are feed-forward networks);

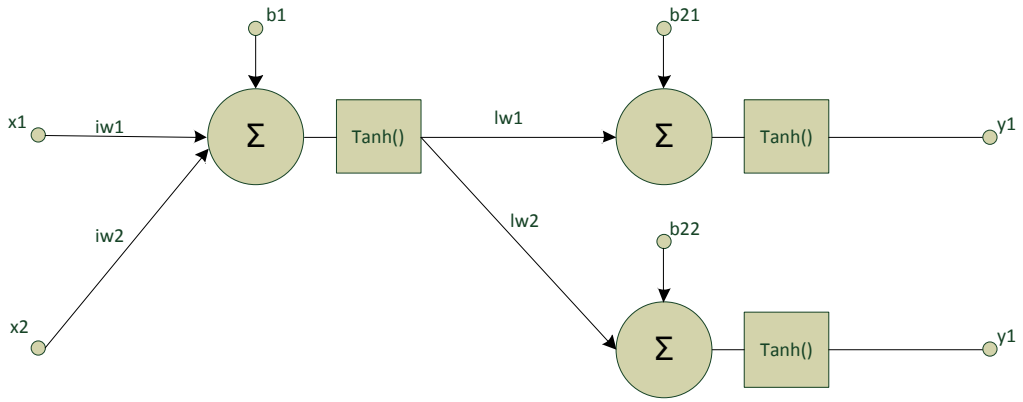


Figure 6.1: Schematic of a 2-1-2 neural network.

6. Are fully connected, missing connections can be training by setting the connection weigths to zero;

7. Their topological layout is equivalent to a directed graph;

8. The activation function is the hyperbolic tangent.

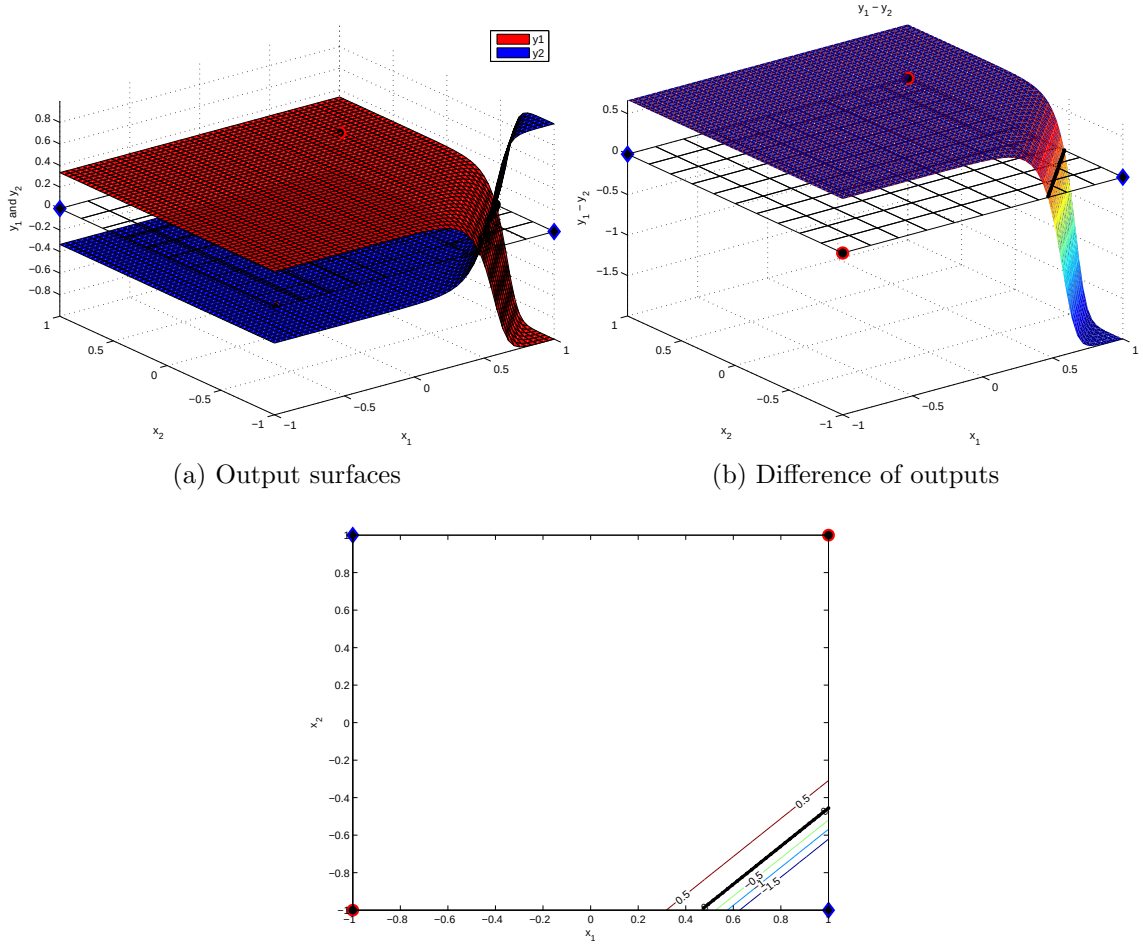
Having considered what architecture of NN we are taking into account during investigation we have set out to analytically derive the formulation of the decision boundary produced by the NN for a given number of hidden neurons in the hidden layer.

We have started to investigate the simplest NN with just one hidden node/neuron in the hidden layer, the schematic of such a network can be seen in Figure 6.1. The first step undertaken in the analytical derivation of the decision boundary was to express the outputs of the NN given the inputs, this is given by Equation 6.1. Which is easy to do since the data flows through the NN just in one direction since the NN is feed-forward.

$$\begin{cases} y_1 = \tanh\left(b_{21} + lw_1 \tanh\left(b_1 + iw_1 x_1 + iw_2 x_2\right)\right) \\ y_2 = \tanh\left(b_{22} + lw_2 \tanh\left(b_1 + iw_1 x_1 + iw_2 x_2\right)\right) \end{cases} \tag{6.1}$$

Since the decision boundary is defined to be the locus of points of equal probability, we make the outputs equal $y_1 = y_2$ and then solve for x_2 with respect to x_1, this gives us Equation 6.2.

$$x_2(x_1) = \frac{\frac{1}{2}\log\left(-\frac{b_{21}-b_{22}+lw_1-lw_2}{b_{21}-b_{22}-lw_1+lw_2}\right) - b_1}{iw_2} - \frac{iw_1}{iw_2}x_1 \tag{6.2}$$



(a) Output surfaces

(b) Difference of outputs

(c) Contour

Figure 6.2: Output produced by a 2-1-2 NN trained on the XOR problem. a) surfaces produced by each individual output neuron; b) decision surface obtained by $y_1 - y_2$; c) contour plot of the output surface $y_1 - y_2$.

In order to validate the obtained decision boundary expression we have plotted the individual outputs y_1, y_2 as the blue and red surface plots. Superimposed on this graph we have plotted the calculated decision boundary (obtained as described above) with a thick black line. There are also 4 data points represented on this graph by black circles with blue and red thick outlines, which represent the datapoints corresponding to the XOR problem. The surface plots are calculated by sampling the x_1 and x_2 values in the $[-1, +1]$ range. For the simple NN architecture with one node in the hidden layer and some arbitrary values for the connection weights we obtained the following plots, shown in Figure 6.2. This figure has three sub-figures, sub-figure a) shows the individual outputs y_1, y_2; sub-figure b) shows the combined output $y_1 - y_2$ which is the decision boundary produced by the network; sub-figure c) shows the combined output $y_1 - y_2$ as a contour. The examination of these figures

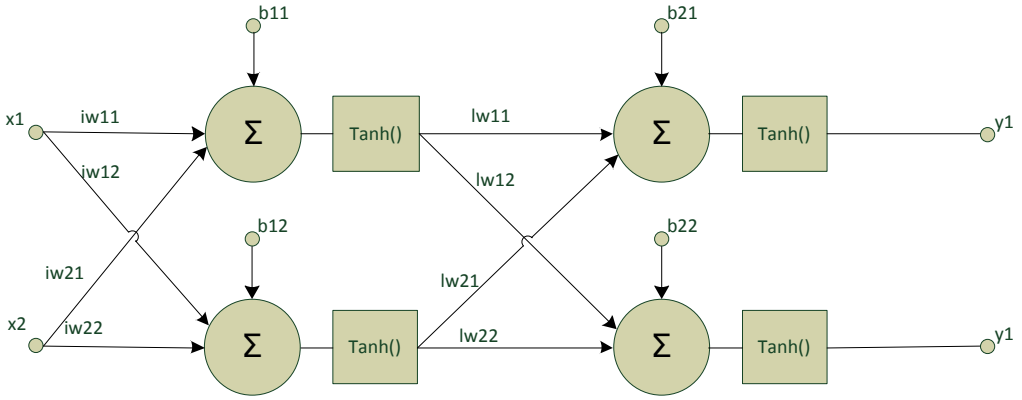


Figure 6.3: Schematic of a 2-2-2 neural network.

and many other examples of NNs we have concluded that our model of the decision boundary is perfectly aligned with the actual decision boundary that is formed by the examined NN.

Next we tried to apply the same derivation process to the NN with two hidden nodes, pictured schematically in Figure 6.3 and wrote the Equation 6.3 that expresses the outputs in terms of the inputs.

$$\begin{cases} y_1 = \tanh\left(b_{21} + lw_{11}\tanh\left(b_{11} + iw_{11}x_1 + iw_{21}x_2\right) + \right. \\ \qquad \left. + lw_{21}\tanh\left(b_{12} + iw_{12}x_1 + iw_{22}x_2\right)\right) \\ y_2 = \tanh\left(b_{22} + lw_{12}\tanh\left(b_{11} + iw_{11}x_1 + iw_{21}x_2\right) + \right. \\ \qquad \left. + lw_{22}\tanh\left(b_{12} + iw_{12}x_1 + iw_{22}x_2\right)\right) \end{cases} \tag{6.3}$$

Trying to solve the Equation 6.3 for x_2 with respect to x_1 when $y_1 = y_2$ is impossible to express using algebraic operators and Taylor approximations or other functional expansions are not feasible.

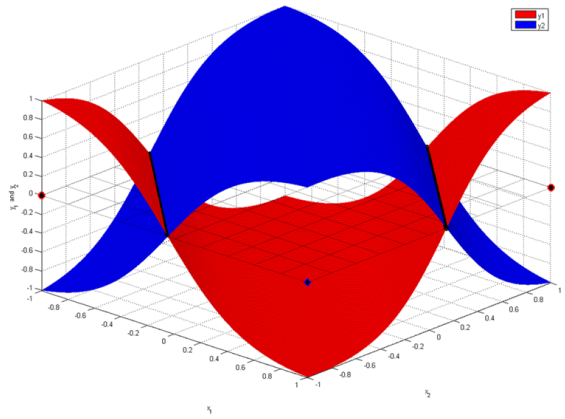
However, if we replace the non-linear, hyperbolic tangent, function with a linear function then we obtain a good approximation to the following system of equations:

We looked at the linear expression of the inputs to the hidden nodes:

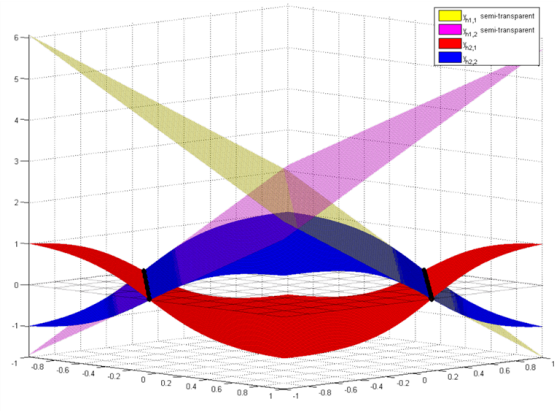
$$\begin{cases} y_1 = (b_{11} + iw_{11}x_1 + iw_{21}x_2) \\ y_2 = (b_{12} + iw_{12}x_1 + iw_{22}x_2) \end{cases} \tag{6.4}$$

For which we obtain the following solutions:

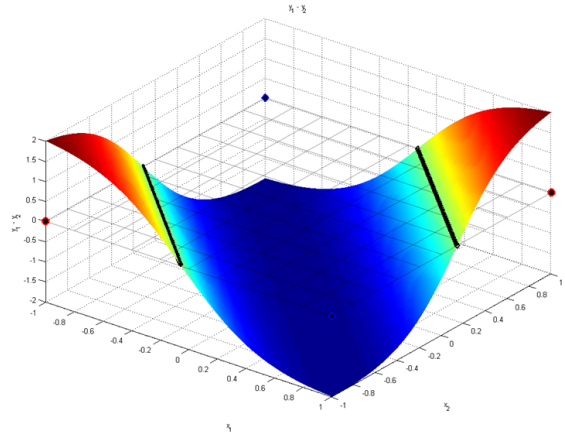
$$\begin{cases} x_2(x_1) = -\dfrac{b_{11}}{iw_{21}} - \dfrac{iw_{11}}{iw_{21}}x_1 \\ x_2(x_1) = -\dfrac{b_{12}}{iw_{22}} - \dfrac{iw_{12}}{iw_{22}}x_1 \end{cases} \tag{6.5}$$



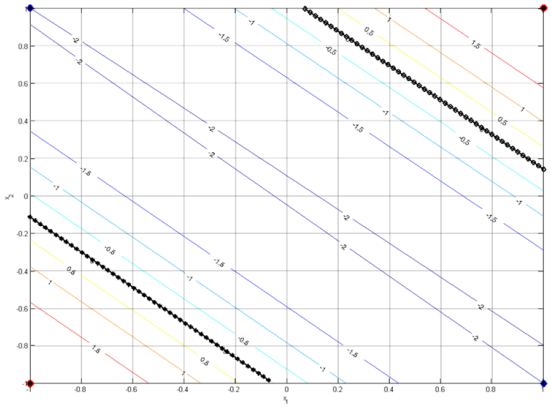
(a) Output Surfaces



(b) Output Decision Planes



(c) Difference of outputs



(d) Output Contour

Figure 6.4: Output produced by a 2-2-2 NN trained on the XOR problem. a) surfaces produced by each individual output neuron; b) individual surfaces with decision planes produced by each neuron; c) decision surface obtained by $y_1 - y_2$; d) contour plot of the output surface $y_1 - y_2$.

These linear equations 6.5 define planes in the feature space x_1,x_2 and the intersection of these planes with the "zero plane" is approximating the decision boundary that the 2-2-2 NN will produce. For validating our findings we have trained a 2-2-2 NN on the XOR binary classification problem and plotted same three types of subfigures a), c) and d), as shown for the examination of the 2-1-2 NN, but for the NN with two nodes in the hidden layer we have added an extra Sub-Figure, subfigure b), to show the planes formed by the system of equations 6.5. The Sub-Figure 6.4a) shows the calculated output surfaces y_1,y_2 against the NN inputs x_1,x_2 along with the thick black lines that were obtained analytically above; b) show the same as a) but with the decision planes superimposed; c) this sub-figure shows the final output $y_1 - y_2$; d) depiction of the contour-plots formed by the calculated output of the NN $y_1 - y_2$ and the analytically obtained decision boundary shown with thick black

lines.

6.2 Weight Adaptation Methodology

We have developed the following methodology to adapt the initial values of the connection weights of the NN that is required to perform the classification on a given dataset. We are going to trial our methodology on Uniformly Distributed synthetic data that is generated using the same principles as described in section 4.2, having polynomial order ranging from 1 to 10, with a known polynomial expression. The neural networks are then adapted to have their decision boundaries aligned closely to the characteristic tangent lines of the generating polynomial. Then the dataset is split into 90% training set and 10% testing set out of the 1,000 data points. One thousand datasets are generated for each polynomial order giving a total of 10,000 datasets and 10,000 adapted and later trained neural networks which will be assessed on their classification performance. Thus we obtain 10,000 classification error rates for each NN on one separate dataset. The same exact training and testing datasets will also be used to train the control experiment NN where the connection weights are not adapted. The results of the comparison between our proposed method of adapting connection weights and the control experiment are shown in the following sections.

6.3 Total Variance Distance

Before we dive into the presentation of our experimental results, we are going to make a slight detour, and present some aspects of our methodology for comparison. For this reason we are going to describe the use of the statistical total variance distance between two probability distributions.

Let us consider that we have two Gaussian distributions of the error values of our two sets of experiments, we shall name these e_1 and e_2. For clarity we will also assign some arbitrary values to σ and μ.

$$e_1 \sim \mathcal{N}(\mu_1, {\sigma_1}^2) \quad \text{where} \begin{cases} \mu_1 = 0 & \text{and} \\ \sigma_1 = 4 \end{cases} \tag{6.6}$$

$$e_2 \sim \mathcal{N}(\mu_2, {\sigma_2}^2) \quad \text{where} \begin{cases} \mu_2 = 5 & \text{and} \\ \sigma_2 = 1 \end{cases} \tag{6.7}$$

A graphical representation of the two Gaussian distributions is shown in Figure

6.5. This figure will be used to describe how the Total Variance Distance was calculated.

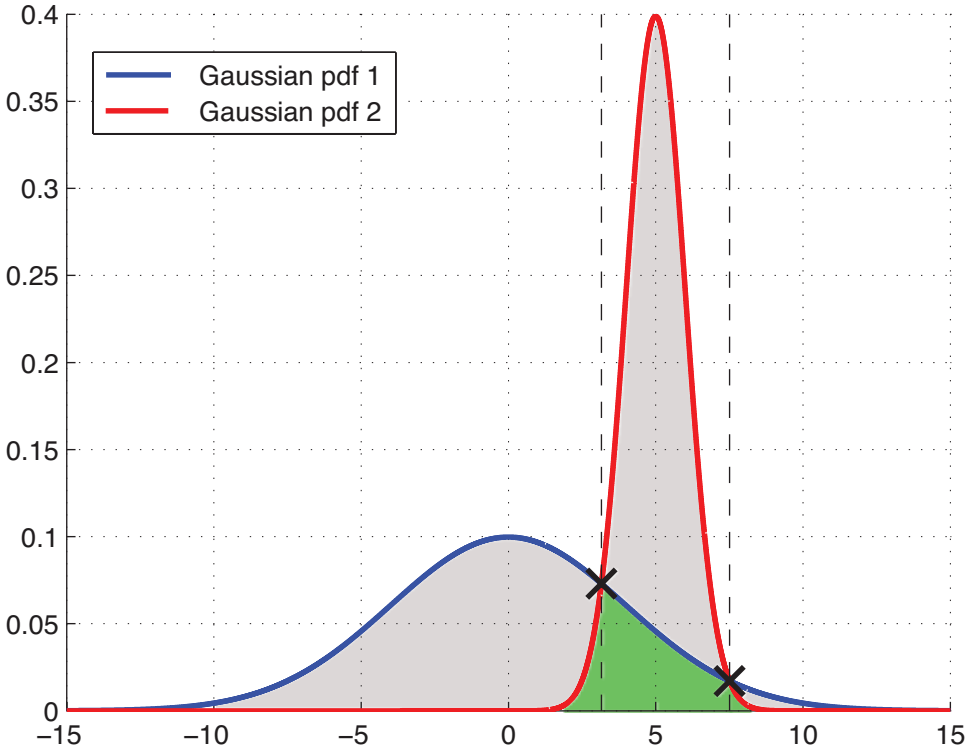


Figure 6.5: Graphical representation of the Total Variance Distance between two Gaussian distributions.

Two Gaussian probability distributions of are compared using the Total Variance Distance adapted from Levin, Peres and Wilmer [42]. This distance metric was chosen because it conveys the dissimilarity information between the two distributions in one single real number that belongs to the semi-closed interval $[0, 1)$, where a distance of 0 corresponds to two distributions which have exactly the same means and standard deviations, while the distance value will tend to reach the value 1 asymptotically, for the ever dissimilar Gaussian distributions, but will not reach the value 1, it will get infinitesimally close to 1. The two distributions however skewed they may be, they still have an intersection point and a very small (sometimes negligible) common overlap area.

The Total Variance Distance (or TV distance) effectively measures the common area under the two probability distribution functions.

The Total Variance distance on discrete events $\|\cdots\|_{TV\,discrete}$ is defined as:

$$\|e_1 - e_2\|_{TV\,discrete} = \frac{1}{2}\sum_{x\in\Omega}|e_1(x) - e_2(x)| \tag{6.8}$$

However, we are not concerned with the measurement of the distance between discrete probability distributions, we would like to have the same measurement of distance for continuous probability distributions. Therefore, we modify the above definition by summing the absolute values between the integrals of the probability distributions on the intervals where the probability distributions have a constant relationship to one another. That is to say, that we split the interval $(-\infty, \infty)$ where we do the integration, to intervals where one particular probability distribution function is always larger than the other. The splits are therefore given by the points of intersection between the two probability distribution functions.

In order to write down the analytical form of the total variance distance for the continuous case of the probability density function we need to define the set of points of intersection between the two functions, which we will later see that it will consist of mostly two points, that we shall refer to as x_1 and x_2. Also, we will assume that $x_1 \leq x_2$.

The two intersection points x_1 and x_2 are represented on Figure 6.5 by the large crosses and the values are represented by the vertical dashed lines.

Now we can write down the analytical form the total variance distance for continuous probability distributions $\|\cdots\|_{TV}$, as:

$$\begin{aligned}\|e_1 - e_2\|_{TV} = \frac{1}{2}\Bigg(&\left|\int_{-\infty}^{x_1} pdf_1(x)\,\mathrm{d}x - \int_{-\infty}^{x_1} pdf_2(x)\,\mathrm{d}x\right| + \\ &+ \left|\int_{x_1}^{x_2} pdf_1(x)\,\mathrm{d}x - \int_{x_1}^{x_2} pdf_2(x)\,\mathrm{d}x\right| + \\ &+ \left|\int_{x_2}^{+\infty} pdf_1(x)\,\mathrm{d}x - \int_{x_2}^{+\infty} pdf_2(x)\,\mathrm{d}x\right|\Bigg)\end{aligned} \tag{6.9}$$

In order to obtain the values for x_1 and x_2, we equalize the two Gaussian probability distributions. The Gaussian probability distribution function (pdf) has the following analytic form, for each of the two distribution separately:

$$pdf_1(x) = \frac{1}{\sigma_1\sqrt{2\pi}}\boldsymbol{e}^{-\frac{(x-\mu_1)^2}{2{\sigma_1}^2}} \tag{6.10}$$

and

$$pdf_2(x) = \frac{1}{\sigma_2\sqrt{2\pi}}\boldsymbol{e}^{-\frac{(x-\mu_2)^2}{2{\sigma_2}^2}} \tag{6.11}$$

By equalling the equations of the two Gaussian distributions (6.10) and (6.11) we obtain (6.13) and solving for x we obtain the coordinates of the intersection points. Also, we observe that, generally, the two graphs have at most two intersection points since the equation is essentially a quadratic equation in x (after we apply the logarithm with base $\boldsymbol{e}$ to both sides of the equation), unless of course, the two distributions have the same σ and μ in which case the two distributions will have an infinite number of common points. However this case is trivial and will be dealt with separately.

$$pdf_1(x) = pdf_2(x) \tag{6.12}$$

$$\frac{1}{\sigma_1\sqrt{2\pi}}\boldsymbol{e}^{-\frac{(x-\mu_1)^2}{2{\sigma_1}^2}} = \frac{1}{\sigma_2\sqrt{2\pi}}\boldsymbol{e}^{-\frac{(x-\mu_2)^2}{2{\sigma_2}^2}} \tag{6.13}$$

We proceed to solve for x by taking the logarithm of both sides of (6.13):

$$\ln\left(\frac{1}{\sigma_1\sqrt{2\pi}}\cdot\boldsymbol{e}^{-\frac{(x-\mu_1)^2}{2{\sigma_1}^2}}\right) = \ln\left(\frac{1}{\sigma_2\sqrt{2\pi}}\cdot\boldsymbol{e}^{-\frac{(x-\mu_2)^2}{2{\sigma_2}^2}}\right) \tag{6.14}$$

and rearranging:

$$\ln\left(\frac{1}{\sigma_1\sqrt{2\pi}}\right) - \frac{(x-\mu_1)^2}{2{\sigma_1}^2} = \ln\left(\frac{1}{\sigma_2\sqrt{2\pi}}\right) - \frac{(x-\mu_2)^2}{2{\sigma_2}^2} \tag{6.15}$$

$$\frac{(x-\mu_2)^2}{2{\sigma_2}^2} - \frac{(x-\mu_1)^2}{2{\sigma_1}^2} = \ln\left(\frac{\sigma_1}{\sigma_2}\right) \tag{6.16}$$

$$\frac{{\sigma_1}^2(x-\mu_2)^2 - {\sigma_2}^2(x-\mu_1)^2}{2{\sigma_1}^2{\sigma_2}^2} = \ln\left(\frac{\sigma_1}{\sigma_2}\right) \tag{6.17}$$

$${\sigma_1}^2(x-\mu_2)^2 - {\sigma_2}^2(x-\mu_1)^2 = 2{\sigma_1}^2{\sigma_2}^2\cdot\ln\left(\frac{\sigma_1}{\sigma_2}\right) \tag{6.18}$$

finally to obtain this quadratic form in x:

$$({\sigma_1}^2 - {\sigma_2}^2)\cdot x^2 + 2(\mu_1{\sigma_2}^2 - \mu_2{\sigma_1}^2)\cdot x +$$

$$+ {\mu_2}^2{\sigma_1}^2 - {\mu_1}^2{\sigma_2}^2 - 2{\sigma_1}^2{\sigma_2}^2 \cdot \ln\left(\frac{\sigma_1}{\sigma_2}\right) = 0 \tag{6.19}$$

By using term identification we can discover the coefficients of the quadratic equation. We will use the following notation for the coefficients of x:

$$\begin{cases} \alpha &= {\sigma_1}^2 - {\sigma_2}^2 \\ \beta &= 2(\mu_1{\sigma_2}^2 - \mu_2{\sigma_1}^2) \\ \gamma &= {\mu_2}^2{\sigma_1}^2 - {\mu_1}^2{\sigma_2}^2 - 2{\sigma_1}^2{\sigma_2}^2 \cdot \ln\left(\frac{\sigma_1}{\sigma_2}\right) \end{cases} \tag{6.20}$$

Hence, we obtain a standard quadratic form for our initial equation (6.13):

$$\alpha x^2 + \beta x + \gamma = 0 \tag{6.21}$$

Therefore the solutions of the quadratic equation, given the notation we employed, is the following conditional set with four branches:

$$x \in \begin{cases} \left\{-\frac{\left(\beta - \sqrt{\beta^2 - 4\,\alpha\,\gamma}\right)}{2\,\alpha}, -\frac{\left(\beta + \sqrt{\beta^2 - 4\,\alpha\,\gamma}\right)}{2\,\alpha}\right\} & \text{if} \quad \alpha \neq 0 \\ \left\{-\frac{\gamma}{\beta}\right\} & \text{if} \quad \alpha = 0 \wedge \beta \neq 0 \\ \mathbb{C} & \text{if} \quad \alpha = 0 \wedge \beta = 0 \wedge \gamma = 0 \\ \emptyset & \text{if} \quad \alpha = 0 \wedge \beta = 0 \wedge \gamma \neq 0 \end{cases} \tag{6.22}$$

The first branch in (6.22) is satisfied when $|\sigma_1| = |\sigma_2|$, since the standard deviation is always positive we can drop the absolute value bars and the condition to have the first branch as: $\sigma_1 = \sigma_2$.

The second branch of (6.22) is satisfied when the standard deviations are the same $\sigma_1 = \sigma_2$ and $\beta \neq 0$ (which implies that the means are different), in this case the equation will have a double solution that will be equal i.e. $x_1 = x_2 = \frac{\mu_1 - \mu_2}{2}$.

The third branch of the same equation (6.22), represents the case when the two distributions have exactly the same means and standard deviations.

Finally, the last branch of equation (6.22), is never possible, because that will imply that if $\beta = 0 \implies \mu_1 = \mu_2$ and $\gamma \neq 0 \implies {\mu_1}^2 \neq {\mu_2}^2$, which is impossible. Therefore, the original equation (6.13) and its quadratic equivalent equation (6.21) will always have two distinct, two equal or an infinite number of solutions.

If we substitute the values of the Gaussian distributions shown in Figure 6.5 into (6.22) we obtain the intersection points of the two Gaussian distributions to have

the horizontal coordinates $x_1 = 3.1573$ and $x_2 = 7.5094$ respectively.

The methodology of comparing two Gaussian distribution using the Total Variance Distance for continuous probability distributions is found to be useful and will be employed in the next section, when the Gaussian distributions of two performance indicators are use to compare the efficiency of adapting the initial weights of NNs.

6.4 Experimental Results Assessment

We have conducted two sets of experiments to determine the usefulness of our proposed method: one experiment is the control experiment where weight adaptation is not used, the other experiment is using the weight adaptation method that we have just discussed. Both sets of experiments consist of creating 10,000 datasets of polynomial order, where each dataset is classified by a feed-forward neural network. The classification error of each individual neural network is retained then the mean and standard deviation is computed. The mean and standard deviation is then compared with the mean and standard deviation of a "control" experiment in which the weight adaptation algorithm is disabled, therefore establishing a baseline for comparison.

For both the control the experiment and our proposed weight adaptation experiment we assess the following groups of performance assessment measures:

- parametric (Gaussian) and non-parametric (percentile) distribution estimation of the classification error rate;
- the total variance distance between the estimated probability distribution of the classification error;
- the number of epochs needed to reach the best validation error of the neural network.

Using 10,000 datasets, a neural network is trained and tested for each dataset, based on the methodology described in Section 6.2 for each of the two experiment sets. Figure 6.6, subfigure a), shows the histogram of the percentage error rate produced by the two experiments. On the horizontal axis we show the percentage error as the bin centre. While on the vertical axis the percentage of NNs that have achieved a classification error that falls in the respective histogram bin. From this sub-figure, we can observe that more than 5% of the NNs produce a classification error between 0% and 4% in the experiment that uses our proposed weight initialization.

The second Sub-Figure 6.6 b), shows the Gaussian probability distribution of the NNs to achieve a classification error rate for the two experiments. Performing a statistical significance test we conclude that the probability distributions estimated for the control experiment and the experiment which uses our proposed weight adaptation methodology shows that at the significance level of 5% the means of the two sample sets do not come from the same population.

To examine further the classification error, we looked at non-parametric distribution estimators in the form of percentage quartiles. For this reason we have included the boxplot in Figure 6.7 and a summary in Table 6.1. From the tabular form of the non-parametric distribution estimation, we observe a slight difference of 0.6% in the overall average value of the classification error across all of the 10,000 datasets and a halving of the standard deviation of the classification error, which suggests a narrower grouping of the classification error rates around the mean. Another insight which was gained from the non-parametric analysis was the reduction of the outliers by 5% or 528 samples, produced by our proposed weight adaptation as compared to the control experiment.

The evaluations which were discussed in the paragraphs above, evaluate the classification error as a whole on the 10,000 datasets that have polynomial decision boundaries ranging from 1 to 10. Therefore, we also investigated in Figure 6.8 the classification error grouped by the polynomial order of the decision boundary within the datasets.

From Figure 6.8 we can see that the mean classification error for the control experiment (shown as the dark blue bars) is higher for the first polynomial orders of 1 to 8 than the corresponding classification error of the weight adapted networks (shown as light blue bars). For order 9 and 10, the control experiments achieve marginally smaller error rate than the weight adapted NNs, but these datasets have very complicated decision boundaries. However, the weight adapted networks outperform their regular/control counterparts by performing much better on lower order polynomial decision boundaries, especially for the first order polynomial boundary. Also, the standard deviation of weight adapted networks is much smaller, meaning they are more consistent, this is especially true for lower order polynomial boundaries in the datasets employed.

The next analysis group we have performed was to employ the newly adapted Total Variance Distance measure we have discussed in Section 6.3 on the estimated Gaussian distributions of the classification error rates produced by the control and our weight adapted neural networks. The results of this analysus is shown in Figure 6.9, from which we can observe that Total Variance Distance shows a more than 0.9 value of dissimilarity between the two estimated error distributions in case of the

first order polynomial decision boundaries and a decreasing dissimilarity as the order of the polynomial is increasing. This presents consistent and concise formulation of the findings presented so far.

The final group of performance assessment measures we investigated the number of training epochs required for the NNs to reach the best performing epoch (on the validation set). The histograms of the best epochs and Gaussian distribution estimates of the two sets of experiments are shown in Figures 6.10 and the non-parametric evaluation results are shown in Figure 6.11 and Table 6.2.

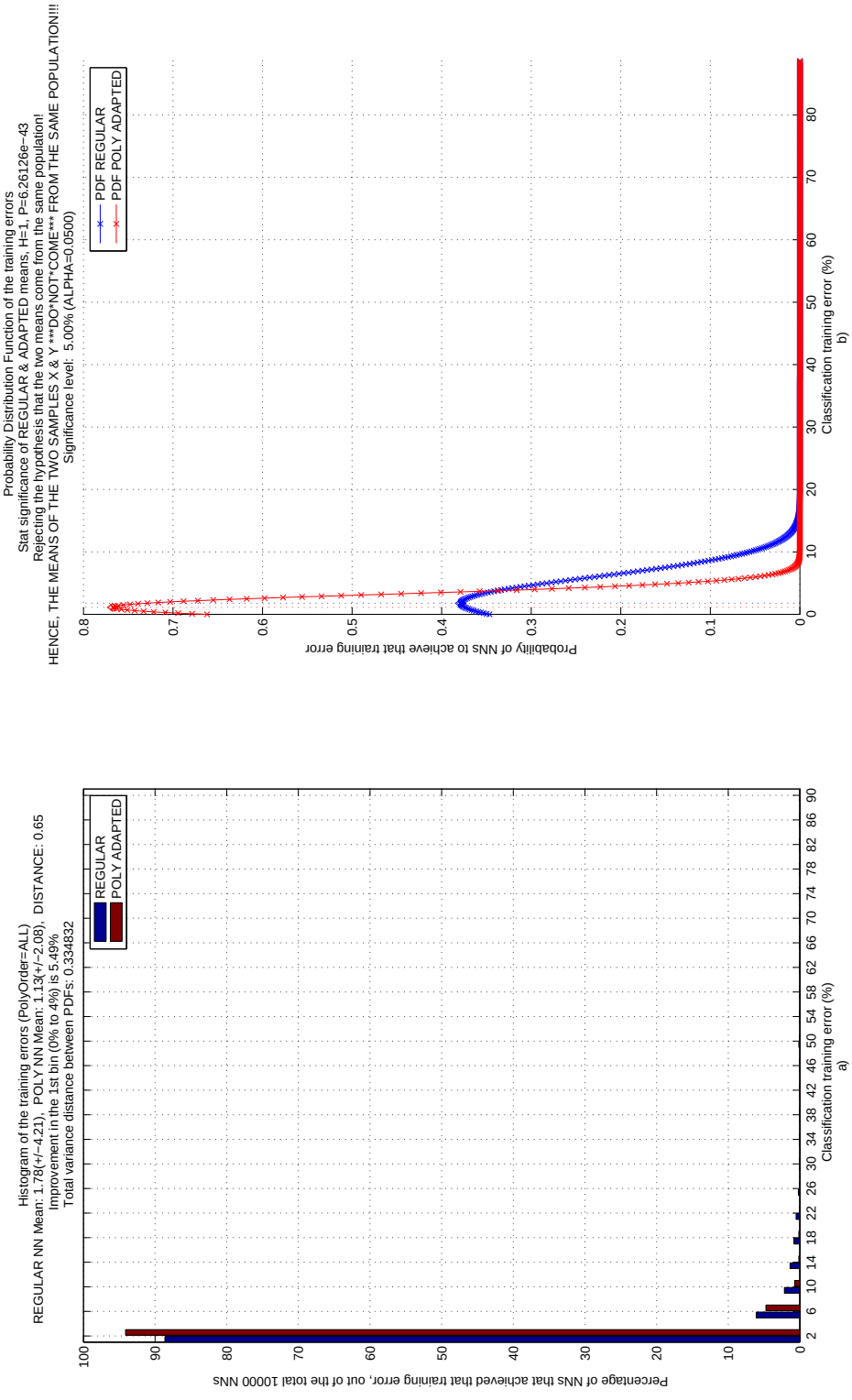


Figure 6.6: Histogram of the classification errors produced by training 10,000 NNs a) comparison of the error rates produced without weight adaptation(blue) and weight adapted NNs (red); b) Fitted probability distributions.

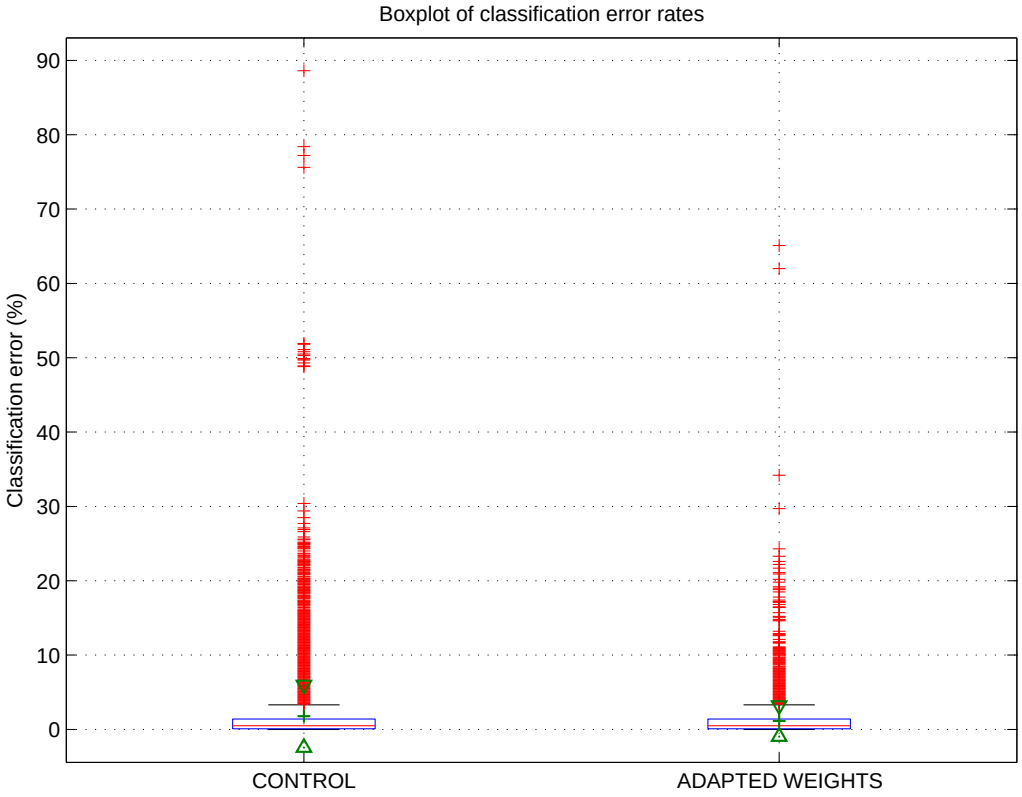


Figure 6.7: Boxplot of overall classification errors of 10,000 trained NNs.

Table 6.1: Statistics of the error distributions.

	CONTROL	ADAPTED WEIGHTS
Average value	1.7775	1.1301
Standard deviation	4.2129	2.0796
Minimum	0	0
Lower whisker	0	0
Lower 25% percentile	0.1	0.1
Median	0.5	0.5
Upper 75% percentile	1.4	1.4
Upper whisker	3.3	3.3
Maximum	88.6	65.1
Percentage in Q1	17.12%	16.95%
Percentage in Q2	57.12%	57.56%
Percentage in Q3	12.35%	17.46%
Percentage of outliers	13.41%	8.03%
Number of outliers	1,294	766
Total number of samples	10,000	10,000

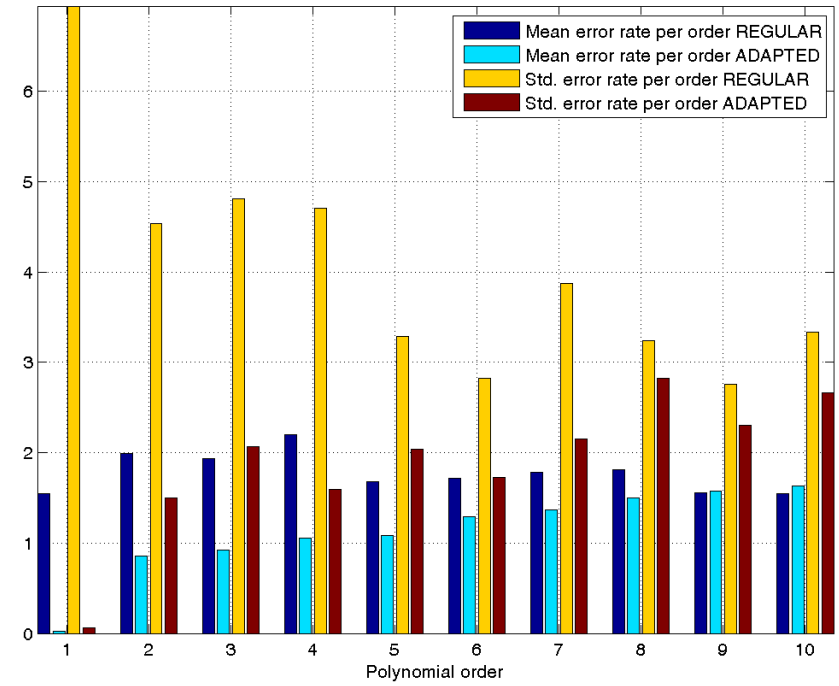


Figure 6.8: Comparison of the average error rate and its standard deviation for the regular and weight adapted experiments.

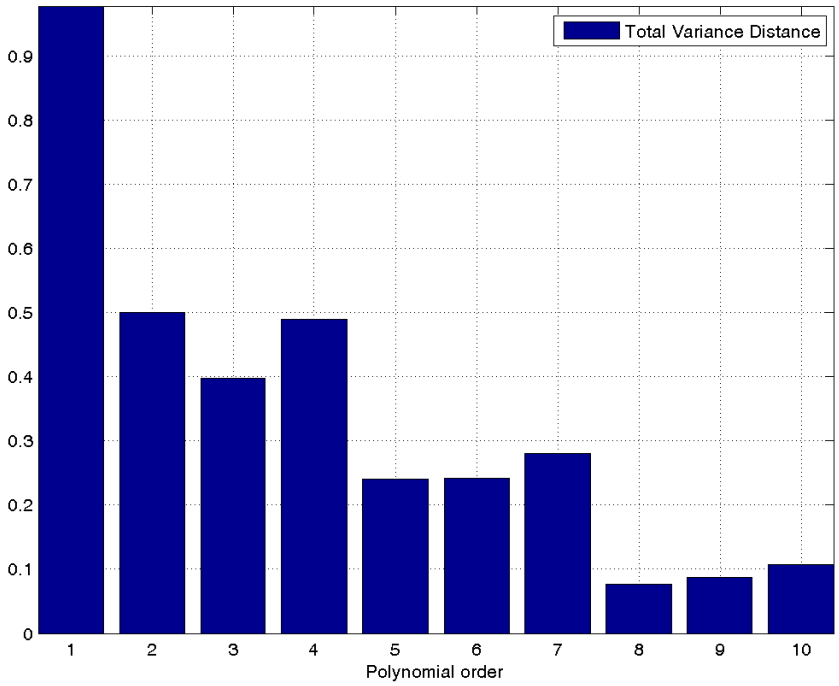


Figure 6.9: Total Variance Distance between the probability distributions of the error rates produced with and without weight adaptation.

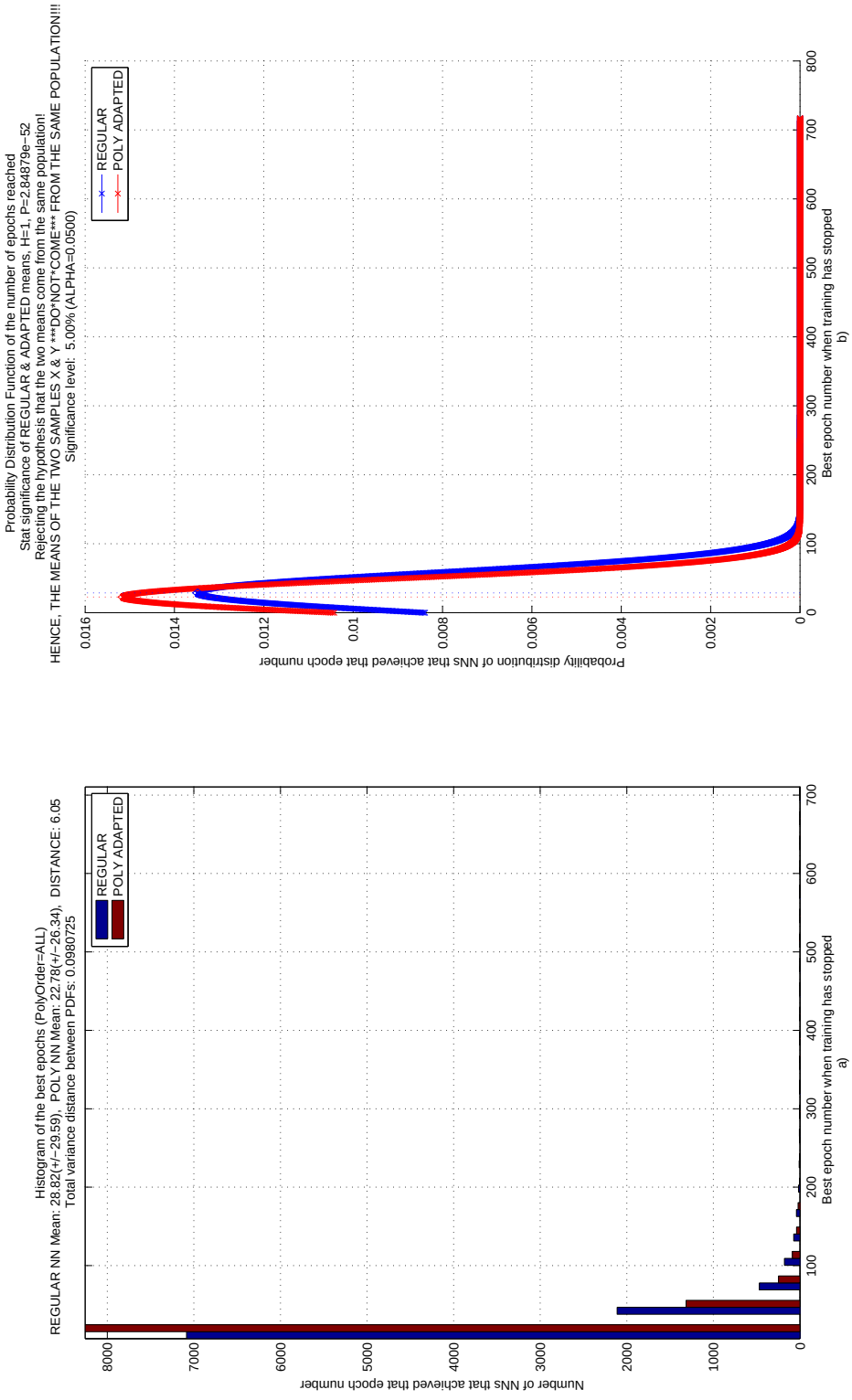


Figure 6.10: Histogram of the best training epoch achieved by training 10,000 NNs a) comparison of the best epoch produced without weight adaptation(blue) and weight adapted NNs (red); b) Fitted probability distributions.

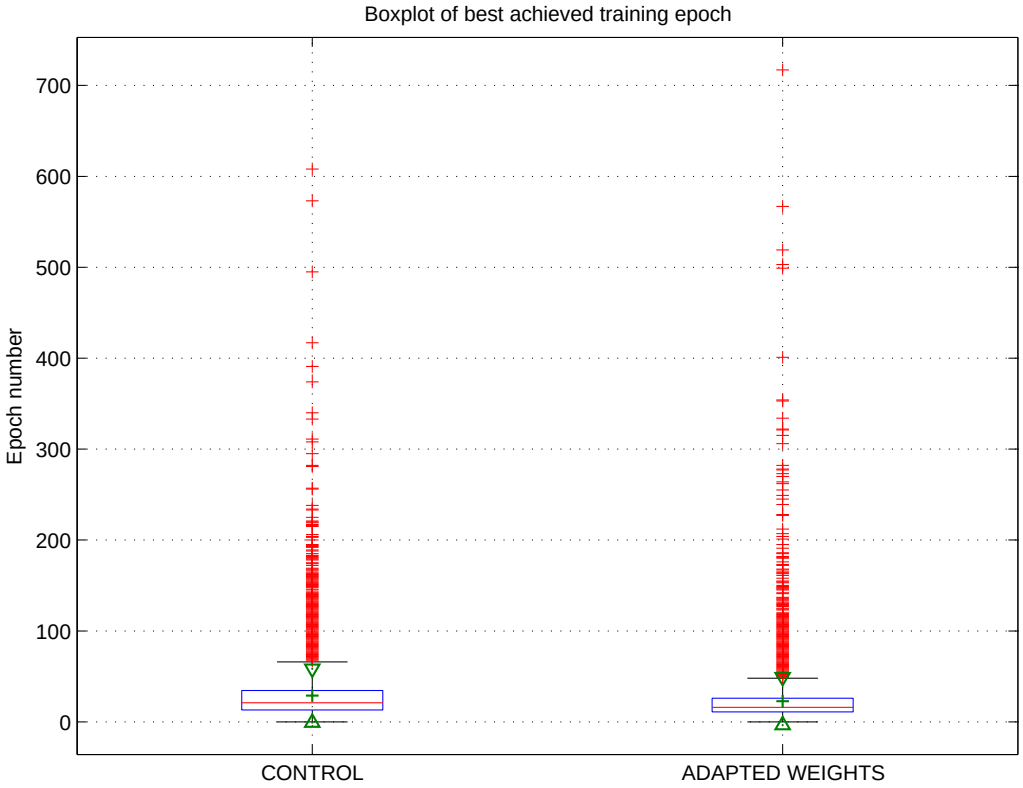


Figure 6.11: Boxplot of best training epoch of 10,000 trained NNs.

Table 6.2: Statistics of the best epoch distributions.

	CONTROL	ADAPTED WEIGHTS
Average value	28.8212	22.776
Standard deviation	29.5947	26.3375
Minimum	0	0
Lower whisker	0	0
Lower 25% percentile	13	11
Median	21	16
Upper 75% percentile	34.5	26
Upper whisker	66	48
Maximum	608	717
Percentage in Q1	22.10%	21.77%
Percentage in Q2	52.90%	52.81%
Percentage in Q3	17.75%	17.89%
Percentage of outliers	7.25%	7.53%
Number of outliers	710	727
Total number of samples	10,000	10,000

6.5 Chapter Conclusions

In this chapter we have described the problems encountered when we tried to determine the analytical expression of the decision boundary formed by a feed-forward neural network. We have formulated two conjectures:

Conjecture one:
The geometrical complexity of the decision boundary, a Neural Network can produce, is determined by its architecture.

Conjecture two:
The problem of finding the analytical expression of the decision boundary of a Neural Network is analogous to finding the roots of a polynomial of order 5 or higher analytically. Which is impossible by using algebraic operators, proven by the Abel-Ruffini theorem (Paolo Ruffini 1799 and Niels Henrik Abel in papers published in 1824 and 1826).

We tried two approximation methods for finding the analytic expression of the decision boundary:

1. 2nd order polynomial approximation of $\boldsymbol{e}^{\theta+tx_2}$ We could have tried higher polynomial orders 4, 6, 8 ... but solving such high order polynomials is cumbersome and impossible using algebraic expressions.

2. 3rd order polynomial and McLaurin series approximation of $\tanh(x)$. We could have tried orders 5, 7, 9, but we faced the same problems.

Finally, we obtained good results of approximating the decision boundary by replacing the non-linear transfer function of the neuron with a linear function and investigating the contribution of individual neurons.

By using these findings we have implemented and tested a direct method to adjust the initial weights of a neural network to approximate the decision boundary found in the training data. This method shows an improvement in the number of epochs needed to learn the pattern recognition problem and reduced the probability of the neural network being trapped in local minimum of the error surface.

Chapter 7

Modular Neural Network Construction

The challenge of solving pattern recognition problems has led us to the construction of a modular neural network system, as described by Mitzias and Mertzios [52], Lu and Ito [44] , but further enhanced by our proposed Meta-Measurement architecture selection methodology detailed in Chapter 5 This architecture selection methodology has proven to be better at estimating the polynomial order complexity than statistical methods as shown in Chapter 5 and in our published work [31].

The current chapter will describe the following items in detail:

1. The creation of a modular neural network system to be used for multi-class pattern recognition;
2. The designed system's performance evaluation on the synthetic datasets;
3. And finally, the designed system's performance evaluation on realistic datasets and comparison with other baseline classifiers.

The main benefit of our proposed method is that it overcomes over-fitting by consistently selecting a number of hidden nodes for the architecture of the neural network modules that is closer to the ideal number, that was used to generate synthetic datasets with polynomial decision boundaries for our experiments. The Meta-Measurement method also proved itself to be superior to statistical methods of assessing the goodness of fit.

7.1 Experimental Setup

The polynomial order of the decision boundary present in a 2-class dataset is predicted using the Meta-Measurement method and the best Statistical Method presented in previous chapters. The order of the polynomial will be used as the number of hidden nodes in the architecture selection process for creating a Modular Neural Network (MNN). This MNN is then used to classify a dataset and the performance of such a MNN architecture will be assessed.

The outline of the steps involved in creating the Modular Neural Network (MNN) for pattern recognition are shown in Figures 7.1 and 7.2. The system diagram had to be split on two separate pages for readability.

In order to create and test a Modular Neural Network we have split the data available for the given pattern recognition problem into 3 sets, a training set, a validation set and a testing set. The training set is used to update the parameters of the neural network module. The validation set is used to determine at what epoch to stop updating the parameters of the neural network module. Finally, the testing set, is used exclusively as ground truth provider to calculate the accuracy of the prediction given by the MNN.

The steps involved in creating and testing the MNN are as follows:

1. Partition the initial data available into three sets: training, validation and testing sets;
2. Compute the eigen-values/vectors from the training set alone;
3. Apply PCA to reduce the dimensionality of the dataset to 2 (this step is used only for the statistical methods to work, and is applied uniformly to all other methods);
4. Normalize the scale of the resulting transformed datasets based on parameters estimated from the training set alone;
5. Split the training and validation sets using a 1-versus-1 scheme into p sub-datasets;
6. Create all the modules of the MNN by applying these steps for each of p sub-datasets;
   (a) Obtain the meta-measurements or statistical polynomial order predictions for each sub-dataset;
   (b) Create and train each a neural network module. Stop the training when Mean Squared Error calculated on the validation set is increasing.

After completing this step training of the MNN has completed. Now the MNN is ready to be used in testing mode or in running mode. The testing procedure requires the following steps:

1. Apply the testing set, that has not been split into multiple sub-datasets, but is in its raw form, to all of the modules in the MNN;

2. Fuse the decisions from all the modules to form the output decision. This forms the class prediction that the MNN makes given the testing set.

3. Compare the predictions made by the MNN to the labellings of the testing set. The result of this comparison is used as the classification error rate that will be reported.

It is very important to note that the training and validation sets are assumed to be known at the time of designing the MNN. But, the testing set is assumed to be unknown at the time of designing the system. Therefore only the training and validation sets are split into p 2-class sub-datasets.

We have taken extreme care in designing this system to reduce the biases introduced by improper handling of the data available for estimating the parameters of the classifier system (training and validation sets) and not to be contaminated with the testing set.

Positive bias can be introduced in the performance evaluation of a system if we use all of the available data to obtain the eigen-vectors during a PCA feature reduction step. Also, a positive bias will be present in the evaluation if we use the whole dataset to obtain the normalization parameters (mean vectors and scaling factor vectors). In both cases, only the portion of the dataset that is reserved for training is used to obtain the eigen-vectors and normalization parameters.

We have clearly separated the steps taken to mitigate this positive bias as it can be seen in Figure 7.1. Another important detail that has to be mentioned is that the "Sample Mean" calculated during the PCA step is not the same as the "Sample Mean" that is used to normalize the data after the PCA step.

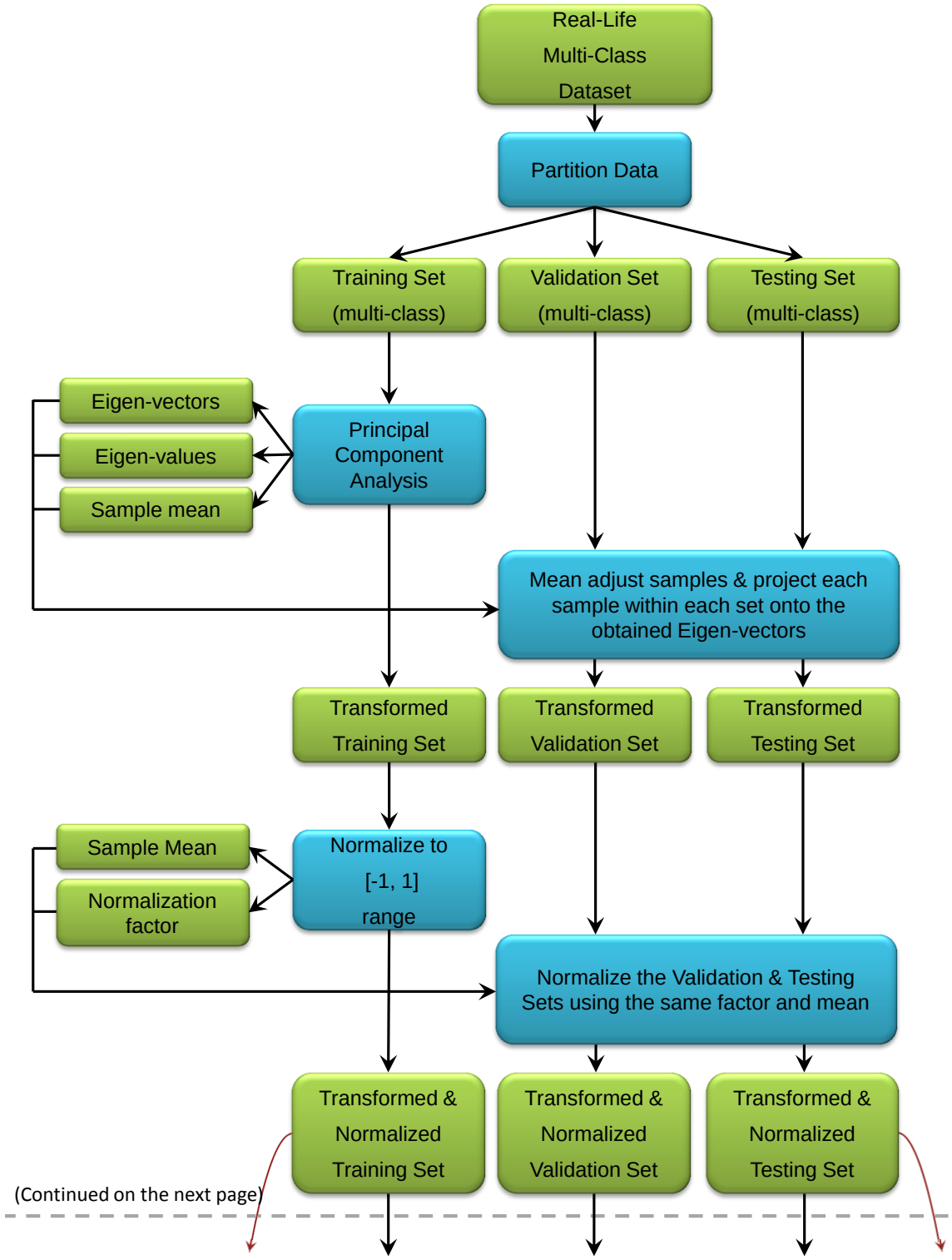


Figure 7.1: Overview of the steps involved in assessing the classification performance of the MNN created using a number of hidden nodes suggested by the Meta-Measurement method (Continued on next page)

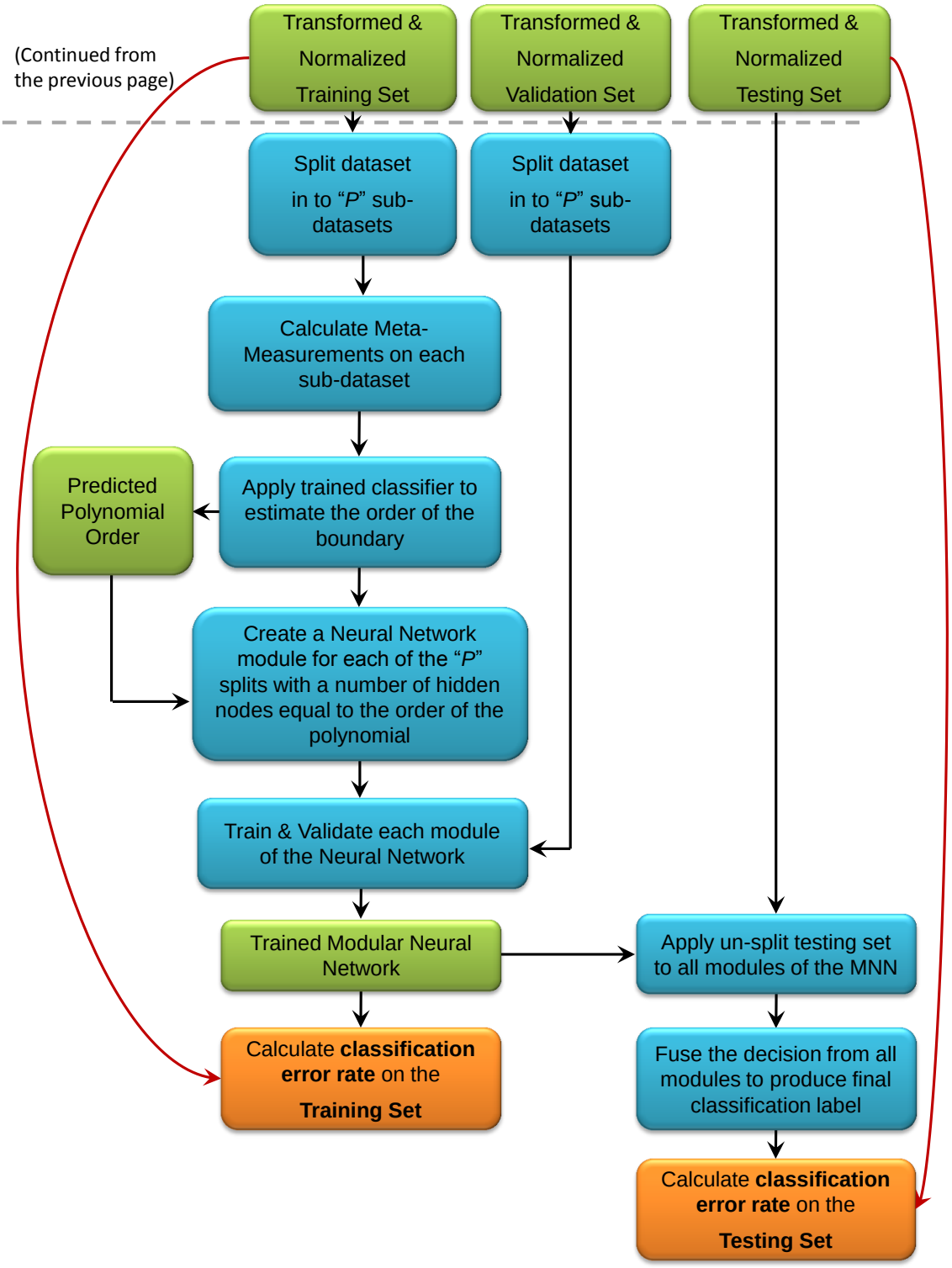


Figure 7.2: Overview of the steps involved in assessing the classification performance of the MNN created using a number of hidden nodes suggested by the Meta-Measurement method (Continuation from the previous page)

7.2 Results on Synthetic Data

We have assessed the performance of 1,000 MNNs, created using the procedure described in the previous section, on 1,000 synthetically generated 2-class datasets that are described in Section 4.2, with a decision boundary of known order between 1 to 10.

The number of hidden nodes corresponding to each NN module is suggested by one of the 5 prediction methods:

1. Our own novel method using so called Meta-Measurements (denoted as META in subsequent figures);

2. Statistical methods that assess the fit of a polynomial of order 1 to 10 to the decision boundary (denoted as STAT in subsequent figures);

3. An Oracle that guesses the order (between 1 to 10) of the polynomial that can be fitted to the decision boundary by randomly drawing samples from a uniformly distributed probability distribution (denoted as RAND in subsequent figures);

4. An Oracle that always guesses the true order of the polynomial decison boundary since it is known from the data generation step (denoted as TRUE in subsequent figures);

5. An Oracle that always returns the maximum possible order, which in our case is 10 (denoted as ALL10 in subsequent figures).

The last three architecture prediction methods are included as a simple comparison to analyse the validity of our proposal.

We have analysed the performance of the MNN systems from 5 points of view.

- Firstly, we summarize the total number of parameters that were suggested by each of the 5 prediction methods in Figure 7.3 which shows a bar chart with the total count of all the connection weights in all the generated NN modules.

  The bar graph shows for each bar the actual total count of connection weights and the relative percentage variation considering the number of connections suggested by the TRUE oracle as the base of comparison, which has a 100% percentage in its corresponding bar.

  From this figure we can observe that the RAND, META and TRUE methods of suggesting the architecture of modules, produce roughly the same number of connection weights, about 23,000 and relative percentage close to 100%.

The other two prediction methods (STAT and ALL10) are overestimating the number of parameters. Obviously, the ALL10 method overestimates grossly the number of parameters by always using the most complicated NN module regardless of the complexity of the dataset. This method also provides the upper limit to the number of parameters.

- The second type of analysis we have done is to create, for each architecture suggestion method, a histograms showing the number of NNs achieving one of the following performance evaluation measures:
  - classification performance, see Figure 7.4; The horizontal axis shows the bin centres of the histogram while vertical axis shows the number of NN modules to fall in each bin. The bins have following centre location: While the bin edges are the following:
  - best achieved training epoch, see Figure 7.5; The horizontal axis shows the bin centres of the histogram while vertical axis shows the number of NN modules to fall in each bin. The bins have following centre location: While the bin edges are the following:
  - best achieved training time, see Figure 7.6. The horizontal axis shows the bin centres of the histogram while vertical axis shows the number of NN modules to fall in each bin. The bins have following centre location:
    [0.36s, 1.04s, 1.73s, 2.41s, 3.09s, 3.78s, 4.46s, 5.14s, 5.83s, 6.51s]
    The bin width is 0.68 seconds, while the bin edges are the following:

Table 7.1: Histogram bin locations for the best epochs.

Bin number	Lower bin location	Upper bin location
1	0.02s	0.70s
2	0.70s	1.38s
3	1.38s	2.06s
4	2.06s	2.74s
5	2.74s	3.42s
6	3.42s	4.10s
7	4.10s	4.78s
8	4.78s	5.46s
9	5.46s	6.14s
10	6.14s	6.82s

- The final analysis is showing the average gradient between all 1,000 NN module.

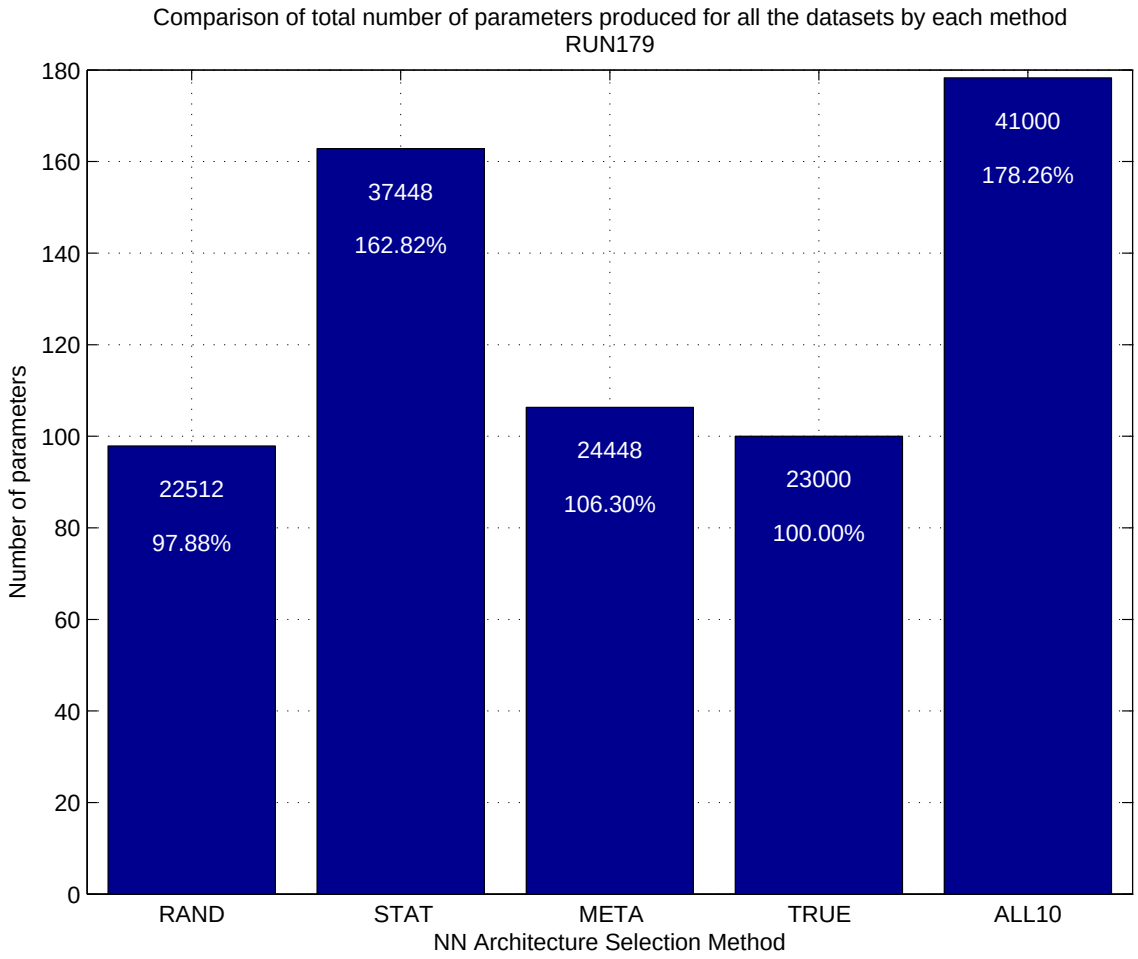


Figure 7.3: Total number of parameters for all the 1000 NNs trained and assessed with each architecture prediction method

All the networks in the 5 architecture prediction methods seem to be learning the problems at the same rate as it can be seen from Figure 7.7.

The bars for all of the 5 architecture suggestion methods are shown grouped on the same figure, one figure for each performance evaluation measure per figure.

From Figure 7.4 we can observe a common trend for all of the 5 architecture suggestion methods, they all seem to perform in the same fashion regardless of the chosen architecture.

The same is true for the best epoch reached, as it can be seen from Figure 7.5.

Only slight variations are to be observed among the 5 architecture prediction methods.

However, when we examine the training time required for the individual NN modules we observe that the ALL10 method produces a slightly larger number of NN that need a time between 2.74seconds & 3.42seconds to train.

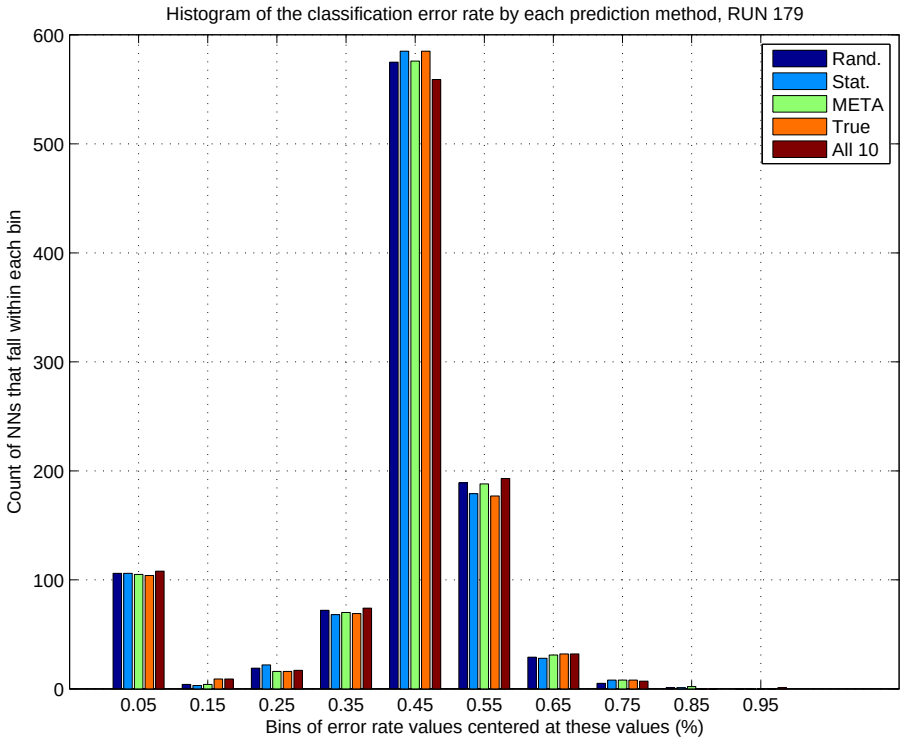


Figure 7.4: Histogram of the number of NNs achieving a particular error rate divided into 10 bins

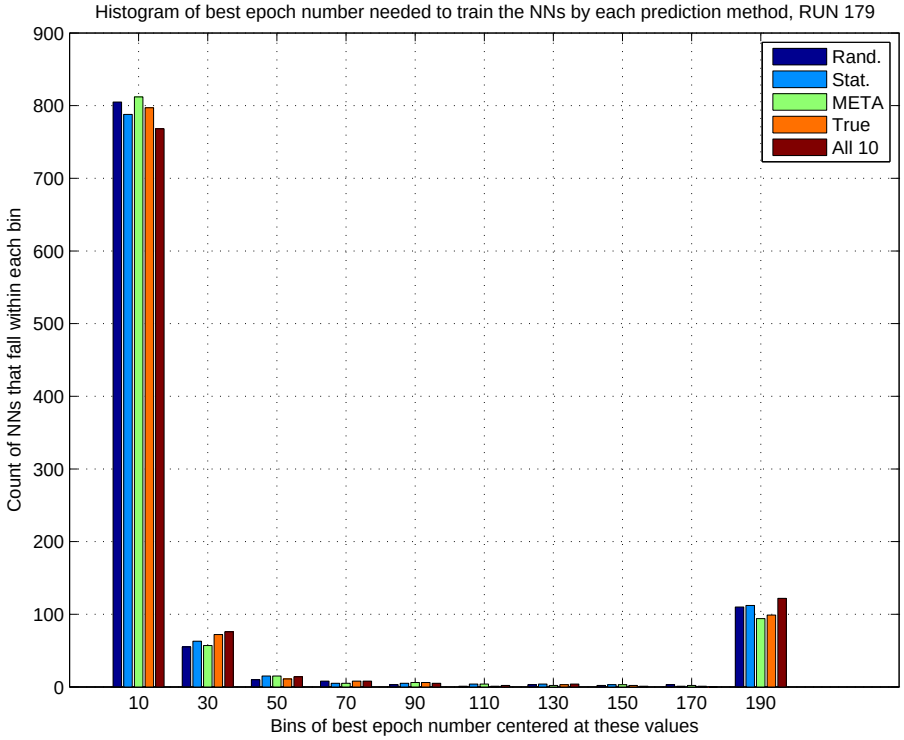


Figure 7.5: Histogram of the number of NNs that have obtained the best training epoch divided into 10 bins

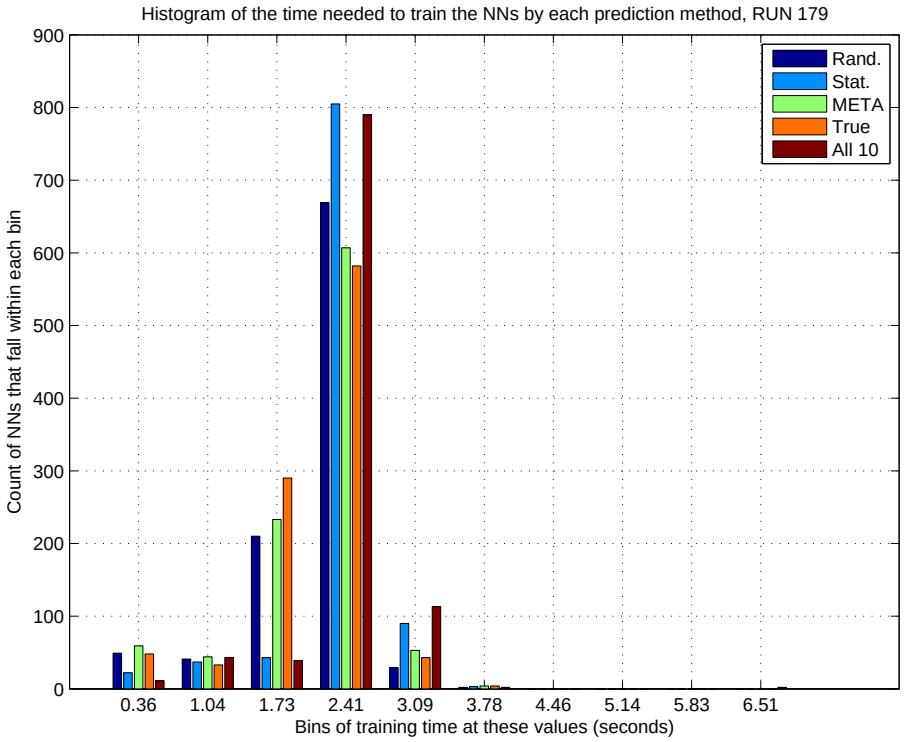


Figure 7.6: Histogram of the number of NNs that have obtained the training time divided into 10 bins

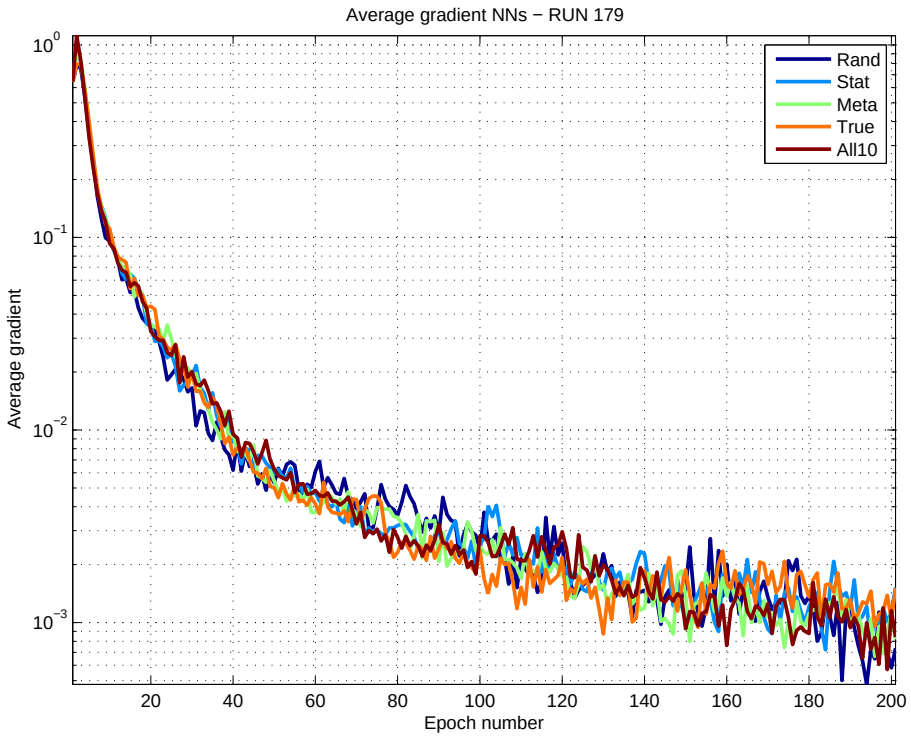


Figure 7.7: Neural Network gradient averaged for all of the 1000 trained networks within each of the 5 architecture selection methods

7.3 Results on Realistic Data

This section contains the experimental results obtained from the training and and testing of various Modular Neural Network (MNN) realizations using different module types compared to some baseline monolithic classifiers that were trained and tested on the entire dataset without building modular configuration.

The experiments were conducted on two datasets, namely the "Iris" flower dataset and "Yeast" datasets described in Section 4.1.1.

The architecture of each module of the MNN is suggested by one of the following methods:

1. Our own novel method using Meta-Measurements to determine the number of hidden nodes for each module of the network (denoted as MNN-META);
2. A variation of the above mentioned Meta Measurement method that tries 9 rotations of the input data and chooses the module with the lowest training error (denoted as MNN-META-ROT);
3. An oracle based method that suggests the number of hidden nodes of each module by randomly drawing a sample from a uniform distribution (denoted as MNN-RAND);
4. An oracle based MNN construction method that always suggests to use 10 hidden nodes for each module of the MNN (denoted as MNN-ALL10);
5. An SVM with Gaussian Kernels [14] of order predicted by the Meta-Measurement method (denoted as MSVM-META);
6. An SVM with Gaussian Kernels [14] of order set to be always 10 (denoted as MSVM-ALL10);
7. Ensemble of classifiers (denoted as ENS);
8. K-Nearest Neighbour Classifier with the parameter determined by the "leave-one-out" method (denoted as KNNC);
9. Nearest Neighbour Classifier (denoted as 1NNC);
10. Nearest Neighbour Classifier considering 3 neighbours(denoted as 3NNC);
11. Linear Discriminant Classifier (denoted as LDC);
12. Quadratic Discriminant Classifier (denoted as QDC);

13. Fisher Discriminant Classifier (denoted as FISHERC);

14. Single/Monolithic Support Vector Machine classifier trained & tested on the entire input dataset(denoted as SVM-GK);

15. A large NN is used with 50 nodes in the hidden layer for each module (denoted as BIG-NN).

The results of testing all these MNNs and baseline classifiers on the Iris flower dataset is summarized in Figure 7.8, where the 10 fold cross-validation classification error rates are are shown as bar chart, one bar for each classifier that was tested on the UCI iris flower dataset, described in Section 4.1.1. From this figure we can observe that the lowest classification error of 2.7% is achieved by the LDC classifier, the large monolithic neural network (denoted as BIG-NN in Figure 7.8) has produced an average error rate of 4%. Whereas, the modular neural networks that had the architecture suggested by our Meta-Measurement method produced an error rate of 11.3% (MNN-META) and 4.7% (MNN-META-ROT). This has the following consequences:

- The very large NN denoted by BIG-NN in Figure 7.8 is definitely over-sized for the problem at hand, by having 50 nodes in the hidden layer. This is because the modular NN created using our Meta-Measurement architecture suggesting method (MNN-META-ROT) has achieved an error rate which is very close to the big monolithic NN. This was done using 3 modules, each module consisting of a NN with less then 10 neurons in the hidden layer;

- Our proposed method effectively reduced over-fitting of neural network by choosing less than 10 neurons for each module;

- Even the modular neural network (MNN-RAND) performed quite well, achieving a 5.3% error rate, but it has to be taken into account that this method suggested the number of hidden nodes in the hidden layer by choosing randomly numbers between 1 and 10. Meaning, that it was choosing from a limited pool of possibilities. If it were to choose from numbers between 1 and 100, then it would have averagely chosen around 50 nodes, which would result in over-sized networks. Our method, also bounded by the same number of choices, consistently chose less then 10 nodes for the hidden layer, which suggests that even if it would had more possibilities to choose from it would still chose a low number of nodes for the architecture of the hidden layer.

The classification error rates of testing the MNNs and baseline classifiers on the Yeast dataset can be seen in Figure 7.9.

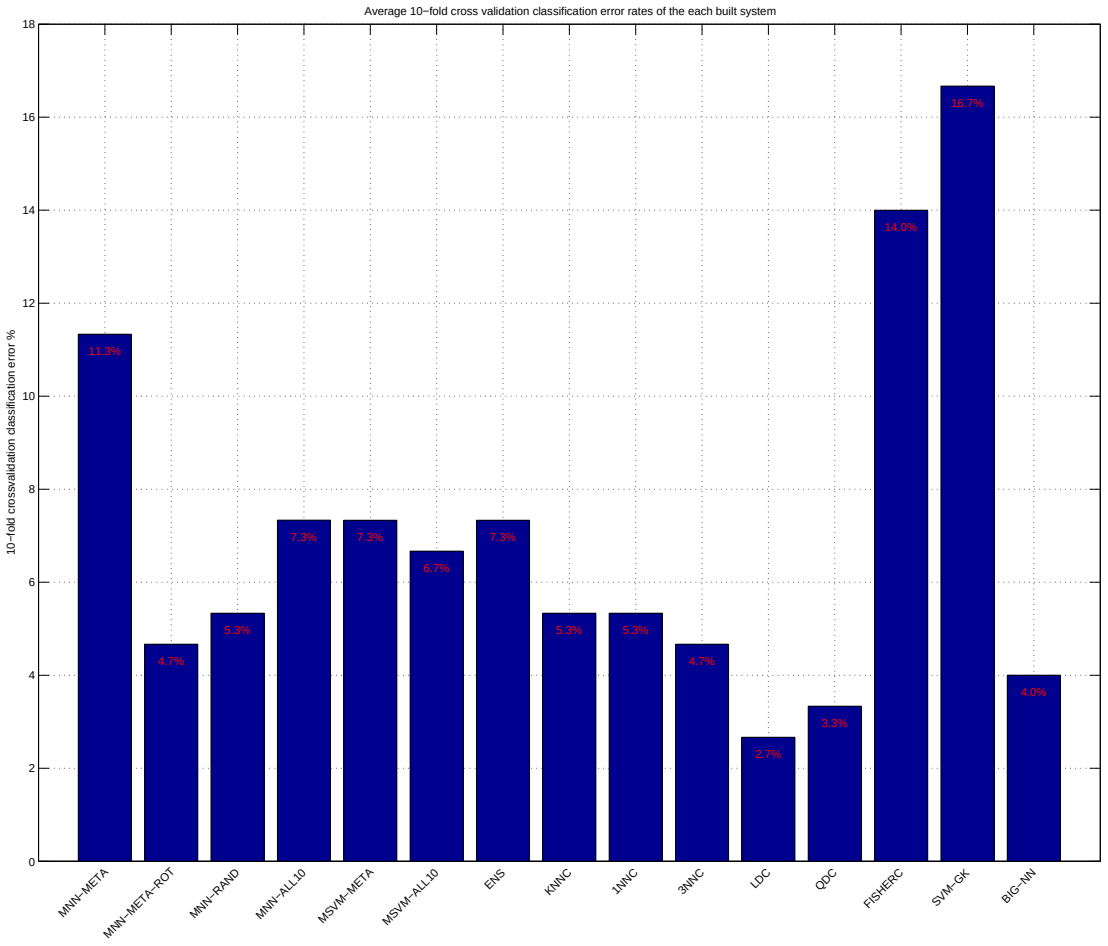


Figure 7.8: 10-fold crossvalidation classification error rates of the MNNs built with different methods of estimating the number of hidden nodes for each module. Used dataset: UCI Flower Iris

The Yeast dataset is a more difficult pattern recognition problem than the Iris flower dataset, this is suggested by the previously achieved best classification error rate of 41.7%, which is due to a larger overlap of the classes in the feature space. The Yeast pattern recognition problem is also more difficult because it has 10 classes to discriminate, so the theoretical uniform distribution of random guessing would yield a 90% error rate of guessing the correct class without any prior information.

The experimental results show the following error rates, in increasing order: 40.4% (LDC), 41.4% (SVM-GK), 42.7% (KNNC). The large network (BIG-NN) achieved a 49% error rate while our Meta-Measurement methods produced the following classification error rates: 66.6% (MNN-META) and 54.5% (MNN-META-ROT), which have not surpassed the baseline classifiers, but this has to do with

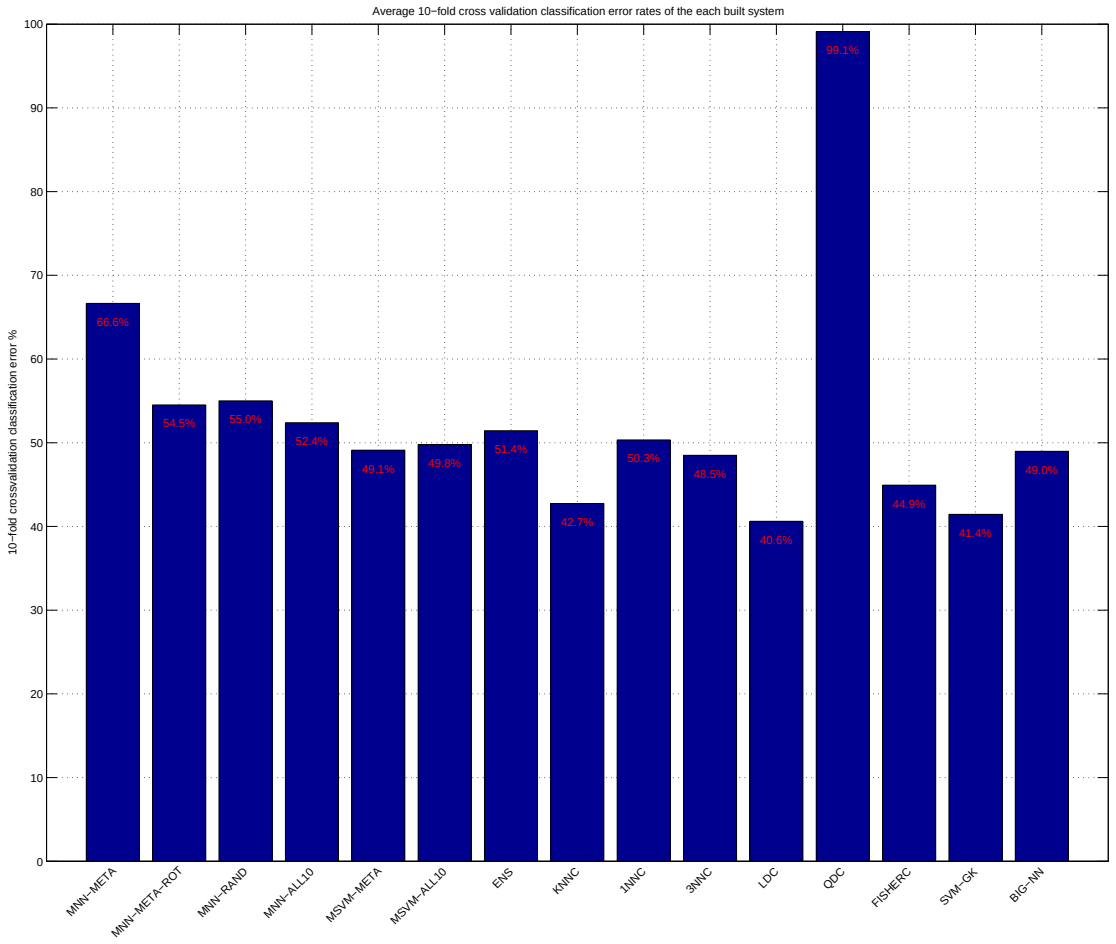


Figure 7.9: 10-fold crossvalidation classification error rates of the MNNs built with different methods of estimating the number of hidden nodes for each module. Used dataset: UCI Yeast

the choice of maximum number of hidden nodes to be ten. Therefore, our proposed method were limited in what they could learn and hence produced a higher classification error.

From this experiment we can conclude the following:

- We can see that even the large NN with 50 nodes, could only achieve an error rate of 49%, which is almost 9% higher than the lowest achieved by the LDC classifier. If we would have extended the highest number of hidden nodes to be more than 50 then we could have possibly achieved comparable results with the other baseline classifiers;
- Implementing dataset rotation within the MNN-META-ROT experiment has proved beneficial;

- The Meta Measurement method helped improve the classification accuracy of the modular SVM trial, where we can see an improvement in the classification error rate of (MSVM-META) 0.7% as opposed to the (MSVM-ALL10).

The exaggerated 99.1% error rate produced by the Quadratic Discriminant Classifier is due to a systematic error of the underlying classifier, which was out of the scope of our work to investigate.

All the presented classification error rates have been obtained by running the same experiment in a 10-fold cross-validation setup.

7.4 Chapter Conclusions

This chapter presented the experimental results from assessing the performance of Modular Neural Networks on synthetic an real life pattern recognition problems using the architecture selection method described in Section 5.

Our methods of architecture selection show promising results for solving the problem of over-fitting of neural networks which can be applied to pattern recognition problems having any number of classes by splitting the original problem in sub-problems that are solved by modules consisting of back-propagation neural networks.

As it was shown in Sub-Section 7.2, the selected number of hidden nodes of the NNs were very close to the true number required by the synthetically generated data and preserved the same classification performance and training time as compared to the other control experiments that were conducted.

The META method selected overall 6.3% more hidden nodes than the truly required number which was considered as the baseline for comparison, while the statistical method recommended 62.8% more hidden nodes in the architecture of the NNs, thus greatly over-fitting the problems at hand.

On real-life pattern recognition problems our proposed methods have shown comparable results while not surpassing the baseline classifiers have successfully overcome over-fitting. On the UCI Iris flower dataset the MNN-META-ROT method achieved 4.7% classification error rate, while only surpassed by LDC (2.7%), QDC (3.3%) and the vastly larger BIG-NN with 50 hidden nodes that achieved 4.0% error rate in a 10-fold cross-validation scenario.

On the UCI Yeast dataset the MNN-META-ROT method achieved 54.5% error rate in discriminating the 10 classes, compared to the 41.7% reported in the literature, which was surpassed by the LDC classifier with 40.6% error rate, but it has done this while being vastly under-sized to solve this problem at hand.

Chapter 8

Conclusions

This final chapter contains the summary of work presented in this thesis, highlights the contributions of this thesis and provides a provides further directions worth of investigation that have arisen from our research but could not be pursued because of time limitations.

8.1 Summary of Work

Our contributions can be summarized as follows, based on the aims and objectives set forth in Section 1.2:

1. We have successfully developed algorithms to generate data synthetically with a polynomial of a given order as the decision boundary between the two classes. See Chapter 4, Section 4.2.

2. We have identified 53 meta-measurements, to characterize the complexity of datasets in Section 3.2.

3. A machine learning method, called 'Meta-Measurement' method, was developed to estimate the polynomial order of the decision boundary between the data points of a two class pattern recognition problem. Very good success rate was achieved in the testing of this method compared with statistical methods of evaluating the goodness of fit. See Section 4.3 and Chapter 5.

4. We have proposed a method of improving the learning rate of neural networks by adapting the initial weights of the network to fit the polynomial functions that can be superimposed on the decision boundary found in the training set. See Chapter 6 for details.

5. Modular Neural Networks were constructed, whereby each module has the architecture suggested by the "Meta-Measurement" method. Our method was compared to Statistical methods of goodness of fit and also compared with 3 oracle based methods. The oracles were random guessing, stating the true order or always suggesting the maximum order anytime they were questioned.

   The classification performance of the MNNs built with suggestions made by these 5 methods is presented in Section 7.2 of Chapter 7.

6. The MNNs trained on realistic datasets were explored in Section 7.3.

8.2 Contributions

The main contributions to knowledge that our research has provided are as follows:

1. We have found a novel method of estimating the goodness of fit for polynomial approximation that vastly surpasses the statistical methods that were tested;

2. Using this selection method we have developed a method for suggesting the number of hidden nodes in the architecture of a multi-layered back-propagation neural network, having a single hidden layer of nodes in the hidden layer. By this method we have reduced the chances of the NN to overfit the data given;

3. We have investigated and found a great correlation between the decision boundary in a 2-class pattern recognition problem, a polynomial that can be superimposed on this decision boundary and the number of hidden nodes in a NN that can solve that pattern recognition problem;

4. We have developed a method to fine tune the connection weights of the NN to approximate the decision boundary to be found in the training data.

8.3 Future Directions of Research

Further researched can be channelled in the following directions:

1. Investigating the decision boundaries graphical in higher dimensional spaces, higher than 2;

2. The Meta-Learning, architecture selection and classifier selection are fields that can benefit from great improvement. We have limited our scope to associating

polynomial functions to decision boundaries. But other functions lend themselves to be investigated. For example the choices available for model selection do not need to be restricted to polynomial functions with orders between 1 to 10, for example Bezier curves and trigonometric functions can be added to the choices a meta-learning algorithm has;

3. Investigating polynomial orders higher than 10 or at least provide the possibility to the architecture selection algorithm to choose from polynomial orders, for example 50 or 100;

4. Investigate the architecture suggestion for more than 1 hidden layer, which has the potential to vastly improve the network capacity of learning non-linear problems.

References

[1] Igor Aleksander. *Advanced Neural Computers, chapter: The Logic of Neural Cognition*. International Symposium on Neural Networks for Sensory And Motor Systems, 1990.

[2] Aristotle. *Physics / Translated with commentaries and glossary by Hippocrates G. Apostle.* Bloomington : Indiana University Press, 1969.

[3] Vassilis Athitsos and Stan Sclaroff. Boosting nearest neighbor classifiers for multiclass recognition. *Proceedings of IEEE Workshop on Learning in Computer Vision and Pattern Recognition*, page pp. 45, 2005.

[4] Gasser Auda and Mohamed Kamel. Modular neural networks: A survey. *International Journal of Neural Systems*, Volume: 9, Number: 2:129–151, 1999.

[5] Farooq Azam. *Biologically Inspired Modular Neural Networks*. PhD thesis, Blacksburg, Virginia, May 2000.

[6] Mitra Basu and Tin Kam Ho. *Data Complexity in Pattern Recognition*. Springer (Advanced Information and Knowledge Processing), 2006.

[7] J. C. Bezdek. *What is Computational Intelligence?* Computational Itelligence Imitating Life. IEEE Press, New York, 1994.

[8] Christopher M. Bishop. *Neural networks for pattern recognition*. Oxford: Clarendon Press, 1995.

[9] P. A. Blamire. The influence of relative sample size in training artificial neural networks. *International Journal of Remote Sensing*, 17 (1):223–230, 1996.

[10] Hamparsum Bozdogan. Model selection and akaikes information criterion (aic): The general theory and its analytical extensions. *Psychometrika*, Volume 52, Issue 3:pp. 345–370, 1987.

[11] W.L Buntine and D. Stirling. Interactive induction. *Artificial Intelligence Applications, 1988., Proceedings of the Fourth Conference on*, pages pp. 320–326, 1988.

[12] José-Ramón Cano. Analysis of data complexity measures for classification. *International Journal of Expert Systems with Applications*, Vol. 40:pp. 4820–4831, 2013.

[13] George D. C. Cavalcanti, Tsang Ing Ren, and Breno A. Vale. Data complexity measures and nearest neighbor classifiers: A practical analysis for meta-learning. *IEEE 24th International Conference on Tools with Artificial Intelligence*, Vol. 1:pp. 1065 –1069, 2012.

[14] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A Library for Support Vector Machine, March 2013.

[15] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, Volume: 10 Issue: 7:pp. 1895–1923, 1998.

[16] Christopher H. Donahue and Hyojung Seo. Attaching valueas to actions: Action and outcome encoding in the primate caudate nucleus. *The Journal of Neuroscience*, Volume: 28, Number: 18:pp. 4579–4580, 2008.

[17] Y. Dote and S. J. Ovaska. Industrial applications of soft computing: A review. *Proceedings of IEEE*, 89(9):1243–1265, 2001.

[18] Ke-Lin Du and M. N. S. Swamy. *Neural networks and statistical learning.* Springer London, 2014.

[19] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* Wiley Interscience, 2nd edition, 2001.

[20] Robert P.W. Duin and et. al. PRTools: The Matlab Toolbox for Pattern Recognition. Published electronically: `http://www.prtools.org/`, Last retrieved: 2012.

[21] A. P. Engelbrecht. *Computational Intelligence: An Introduction.* John Wiley & Sons, Ltd., 2nd edition, 2007.

[22] R. A. Fisher. *Statistical Methods for Research Workers, 13th ed.* Hafner, 1958.

[23] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Annual Eugenics also in "Contributions to Mathematical Statistics" (John Wiley, NY, 1950)*, Volume 7, Part II:179–188, 1936.

[24] G. M. Foody. Using prior knowledge in artificial neural network classification with a minimal training set. *International Journal of Remote Sensing*, 16 (2):301–312, 1995.

[25] A. Frank and A. Asuncion. UCI Machine Learning Repository. Published electronically: `http://archive.ics.uci.edu/ml`, Last retrieved: 2013.

[26] David Freedman and Persi Diaconis. On the histogram as a density estimator: l_2 theory. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, Vol. 57, Issue 4:pp 453–476, 1981.

[27] Hsin-Chia Fu, Yen-Po Lee, and et al. Divide-and-conquer learning and modular perceptron networks. *IEEE Transactions on Neural Networks*, Volume 12, Number: 2, 2001.

[28] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition, Second Edition*. Academic Press, 1990.

[29] Damien Garcia. AIC polynomial degree estimation function for MATLAB . Published electronically: `http://www.biomecardio.com/matlab/polydeg.html`, Last retrieved: 2010.

[30] Bogdan G. Gherman and Konstantinos Sirlantzis. Data complexity assessment for constructing modular artificial neural networks. In *School of Engineering and Digital Arts Conference*, 20 January 2012.

[31] Bogdan G. Gherman and Konstantinos Sirlantzis. Polynomial order prediction using a classifier trained on meta-measurements. *4th International Conference on Emerging Security Technologies*, pages pp. 117–120, 2013.

[32] Brian Hanson, Katherine Klink, Kenji Matsuura, Scott M. Robeson, and Cort J. Willmott. Vector correlation: Review, exposition, and geographic application. *Annals of the Association of American Geographers*, Vol. 82, No. 1:pp.103–116, 1992.

[33] Simon S. Haykin. *Neural networks and learning machines*. Harlow; London: Pearson Education, 3rd international edition, 2009.

[34] Donald Hebb. *The Organization of Behavior*. John Wiley, New York, 1949.

[35] Tin Kam Ho and Mitra Basu. Complexity measures of supervised classification problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume: 24, Issue: 3:pp. 289 – 300, 2002.

[36] Paul Horton and Kenta Nakai. A probablistic classification system for predicting the cellular localization sites of proteins. *Intelligent Systems in Molecular Biology, St. Louis, USA*, pages pp. 109–115, 1996.

[37] Stephen C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32:241–254, 1967.

[38] Stephen C. Johnson. This week's citation classic. *ISI Current Contents*, Number 25, 24 June 1985.

[39] T. Kavzoglu. Increasing the accuracy of neural network classification using refined training data. *Environmental Modelling & Software*, Volume: 24: Issue: 7:850–858, 2009.

[40] M.G. Kendall, A. Stuart, and J.K. Ord. *The advanced Theory of Statistics, Vol 3, Design and Analysis and Time Series, Chapter 44*. Griffin, London, Forth Edition, 1983.

[41] Ludmila I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. A Wiley-Interscience publication, 2004.

[42] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov Chains and Mixing Times*. American Mathematical Society, 2009.

[43] Ming Li and Paul Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer -Verlag, 2nd edition, 1997.

[44] Bao-Liang Lu and Masami Ito. Task decomposition and module combination based on class relations: A modular neural network for pattern classification. *IEEE Transactions on Neural Networks*, Volume: 10, Number: 5, 1999.

[45] Núria Macià, Ester Bernadó-Mansilla, and Albert Orriols-Puig. Preliminary approach on synthetic data sets generation based on class separability measure. *19th International Conference on Pattern Recognition, ICPR 2008*, pages pp. 1–4, 2008.

[46] C. Mallows. Some comments on cp. *Technometrics*, Vol. 15, No. 4:pp. 661–675, 1973.

[47] R. Marks. Intelligence: Computational versus artificial. *IEEE Transactions on Neural Networks*, 4(5):737–739, 1993.

[48] Warren S. McCulloch and Walter Pitts. *A logical calculus of the ideas imminent in nervous activity.* Bulletin of Mathematical Biophysics. V. 5, 1943.

[49] Kishan Mehrotra, Chilukuri K. Mohan, and Sanjay Ranka. *Elements of Artificial Neural Networks, 2nd Printing.* MIT Press, 2000.

[50] D. Michie, D.J. Spiegelhalter, and C.C. Taylor. *Machine Learning, Neural and Statistical Classification.* New York; London : Ellis Horwood, 1994.

[51] Tom M. Mitchell. *Machine Learning.* McGraw Hill, 1997.

[52] Dimitris A. Mitzias and Basil G. Mertzios. A neural multiclassifier system for object recognition in robotic vision applications. *Science Direct, Measurement*, 36:315–330, 2004.

[53] Warwick J. Nash, Tracy L. Sellers, Simon R. Talbot, Andrew J. Cawthorn, and Wes B. Ford. The population biology of abalone (haliotis species) in tasmania. i. blacklip abalone (h. rubra) from the north coast and islands of bass strait. *Sea Fisheries Division, Technical Report No. 48*, 1994.

[54] N. J. Nilsson. *Learning Machines: Foundations of Trainable Pattern Classifying Systems.* New York; McGraw Hill, 1965.

[55] N. J. Nilsson. *The quest for artificial intelligence: a history of ideas and achievements.* Cambridge University Press, 2010.

[56] M. Pal and P. M. Mather. An assessment of effectiveness of decision tree methods for land cover classification. *Remote Sensing of Environment*, 86 (4):554–565, 2003.

[57] Dmitry Pavlov, Alexandrin Popescul, David M. Pennock, and Lyle H. Ungar. Mixtures of conditional maximum entropy models. *Proceedings of International Conference on Machine Learning*, pages pp. 584–591, 2003.

[58] Plato. *Republic : 1-2.368c4 / Plato ; with an introduction and commentary by Chris Emlyn Jones.* Warminster : Aris & Phillips, 2007. Ancient Greek and English text on facing pages.

[59] T. Pogio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.

[60] Panayiota Poirazi, Costas Neocleous, and et al. Classification capacity of a modular neural network implementing neurally inspired architecture and training rules. *IEEE Transactions on Neural Networks*, Volume: 15, Number: 13, 2004.

[61] Lutz Prechelt. Proben1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules. Technical Report 21/94, Fakultät für Informatik, Universität Karlsruhe, September 30, 1994.

[62] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing, 2nd ed.* Cambridge University Press, 1992.

[63] Ricardo B. C. Prudencio, Carlos Soares, and Teresa B. Ludemir. Uncertainty sampling methods for selecting datasets in active meta-learning. *The 2011 International Joint Conference on Neural Networks (IJCNN)*, pages pp. 1082–1089, 2011.

[64] Thomson Reuters. ISI Web Of Knowledge. Published electronically: `http://www.webofknowledge.com/`, Last retrieved: February 2014. Query String: "Topic=(neural networks) Refined by: [excluding] General Categories=(ARTS & HUMANITIES) Timespan=All Years".

[65] Eric Ronco and Peter Gawthrop. Modular neural networks: a state of the art. Technical report: Csc-95026 (may 12, 1995), Centre for System and Control, University of Glasgow UK, 1995.

[66] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, American Psychological Association*, Volume 65, Number 6:pp. 386–408, 1958.

[67] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation.* in Parallel Distributed Processing: Explorations in the Microstructures of Cognition, 1986.

[68] Wolfram Schultz, Peter Dayan, and P. Read Montague. A neural substrate of prediction and reward. *Science*, Vol. 275, No: 5306:pp. 1593–1599, 1997.

[69] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, Volume 6, Number 2:pp. 461–464, 1978.

[70] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, Vol. 27:pp. 379–423, 623–656, 1948.

[71] J. M. Sotoca, R. A. Mollineda, and J. S. Sánchez. A meta-learning framework for pattern classification by means of data complexity measures. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial*, Vol. 29:pp. 31–38, 2006.

[72] M. H. Stone. The generalized weierstrass approximation theorem. *Mathematics Magazine*, Volume 21, Number 4 (Mar. - Apr., 1948):pp. 167–184, 1948.

[73] A. N. Tikhonov. On solving incorrectly posed problems and method of regularization. *Doklady Akademii Nauk USSR*, 151:501–504, 1963.

[74] F. van der Heijden, R. P. W. Duin, D. de Ridder, and D. M. J. Tax. *Classification, Parameter Estimation and State Estimation, An Engineering Approach using Matlab*. John Wiley & Sons, Ltd., 2004.

[75] Aki Vehtari and Janne Ojanen. A survey of bayesian predictive methods for model assessment, selection and comparison. *Statistics Surveys*, Volume 6:pp.142–228, 2012.

[76] W.R. Wade. *An Introduction to Analysis, 2nd Ed.* Upper Saddle River, NJ: Prentice-Hal, 2000.

[77] Andrew R. Webb. *Statistical Pattern Recognition*. John Wiley & Sons, Ltd., 2002.

[78] B. Widrow and M. E. Hoff. Adaptive switching circuits. *Proceedings IRE WESCON Conference,New York*, pages pp. 96–104, 1960.

[79] B. Widrow and M. A. Lehr. 30 years of adaptive neural networks: Perceprons, MADALINE and backpropagation. *Proceedings of IEEE*, Volume 78, Number 9:pp. 1415–1442, 1990.

[80] Mingyang Xu and Michael Golay. Survey of model selection and model combination. *Social Science Research Network (SSRN)*, (January 1, 2008):Available at SSRN: `http://ssrn.com/abstract=1742033` or `http://dx.doi.org/10.2139/ssrn.1742033`, 2008.

[81] Yong Xu and Guangming Lu. Analysis on fisher discriminant criterion and linear separability of feature space. *International Conference on Computational Intelligence and Security*, Vol. 2:pp. 1671–1676, 2006.