

Evolutionary Algorithms for Hyperparameter Search in Machine Learning

Author:

Bai, Yu

Publication Date:

2021

DOI:

<https://doi.org/10.26190/unsworks/2002>

License:

<https://creativecommons.org/licenses/by/4.0/>

Link to license to see what you are allowed to do with this resource.

Downloaded from <http://hdl.handle.net/1959.4/100092> in <https://unsworks.unsw.edu.au> on 2024-03-29

Evolutionary Algorithms for Hyperparameter Search in Machine Learning

by

Yu Bai

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
THE UNIVERSITY OF
NEW SOUTH WALES



SYDNEY • AUSTRALIA

April 12, 2021

All rights reserved. This work may not be
reproduced in whole or in part, by photocopy
or other means, without the permission of the author.

© 2022 by Yu Bai

Thesis Abstract

Machine learning algorithms usually have a number of hyperparameters. The choice of values for these hyperparameters may have a significant impact on the performance of an algorithm. In practice, for most learning algorithms the hyperparameter values are determined empirically, typically by search. From the research that has been done in this area, approaches for automating the search of hyperparameters mainly fall into the following categories: manual search, greedy search, random search, Bayesian model-based optimization, and evolutionary algorithm-based search. However, all these approaches have drawbacks — for example, manual and random search methods are undirected, greedy search is very inefficient, Bayesian model-based optimization is complicated and performs poorly with large numbers of hyperparameters, and classic evolutionary algorithm-based search can be very slow and risks falling into local optima. In this thesis we introduce three improved evolutionary algorithms applied to search for high-performing hyperparameter values for different learning algorithms. The first, named EWLNB, combines Naive Bayes and lazy instance-weighted learning. The second, EMLNB, extends this approach to multiple label classification. Finally, we further develop similar methods in an algorithm, named SEODP, for optimizing hyperparameters of deep networks, and report its usefulness on a real-world application of machine learning for philanthropy.

EWLNB is a differential evolutionary algorithm which can automatically adapt to different datasets without human intervention by searching for the best hyperparameters for the models based on the characteristics of the datasets to which it is applied. To validate the EWLNB algorithm, we first use it to optimize two key parameters for a locally-weighted Naive Bayes model. Experimental evaluation of this approach on 56 of the benchmark UCI machine learning datasets demonstrate that EWLNB significantly outperforms Naive Bayes as well as several other improved versions of the Naive Bayes algorithms both in terms of classification accuracy and class probability estimation. We then extend the EWLNB approach in the form of the Evolutionary Multi-label Lazy Naive Bayes (EMLNB) algorithm to enable hyperparameter search for multi-label classification problems.

Lastly, we revise the above algorithms to propose a method, SEODP, for optimizing deep learning (DL) architecture and hyperparameters. SEODP uses a semi-evolutionary and semi-random approach to search for hyperparameter values, which is designed to evolve a solution automatically over different datasets. SEODP is much faster than other methods, and can adaptively determine different deep network architectures automatically. Experimental results show that compared with manual search, SEODP is much more effective, and compared with grid search, SEODP can achieve optimal performance using only approximately 2% of the running time of greedy search. We also use SEODP on a real-world social-behavioral dataset from a charity organization for a philanthropy application. This dataset contains comprehensive real-time attributes on potential indicators for candidates to be donors. The results show that SEODP is a promising approach for optimizing deep network (DN) architectures over different types of datasets, including a real-world dataset. In summary, the results in this thesis indicate that our methods address the main drawback of evolutionary algorithms, which

is the convergence time, and show experimentally that evolutionary-based algorithms can achieve good results in optimizing the hyperparameters for a range of different machine learning algorithms.

ORIGINALITY STATEMENT

☒ I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

COPYRIGHT STATEMENT

☒ I hereby grant the University of New South Wales or its agents a non-exclusive licence to archive and to make available (including to members of the public) my thesis or dissertation in whole or part in the University libraries in all forms of media, now or here after known. I acknowledge that I retain all intellectual property rights which subsist in my thesis or dissertation, such as copyright and patent rights, subject to applicable law. I also retain the right to use all or part of my thesis or dissertation in future works (such as articles or books).

For any substantial portions of copyright material used in this thesis, written permission for use has been obtained, or the copyright material is removed from the final public version of the thesis.

AUTHENTICITY STATEMENT

☒ I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis.

UNSW is supportive of candidates publishing their research results during their candidature as detailed in the UNSW Thesis Examination Procedure.

Publications can be used in the candidate's thesis in lieu of a Chapter provided:

- The candidate contributed **greater than 50%** of the content in the publication and are the "primary author", i.e. they were responsible primarily for the planning, execution and preparation of the work for publication.
- The candidate has obtained approval to include the publication in their thesis in lieu of a Chapter from their Supervisor and Postgraduate Coordinator.
- The publication is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in the thesis.

☒ The candidate has declared that **some of the work described in their thesis has been published and has been documented in the relevant Chapters with acknowledgement**

A short statement on where this work appears in the thesis and how this work is acknowledged within chapter/s:

Some of the work in the thesis has been published, all with acknowledged within the chapter/s:

Some of the contents in Chapter three has been published in Journal of Ambient Intelligence and Humanized Computing, 2021.

Some of the contents in Chapter four has been published in IJCAI-19.

Some of the contents in Chapter five has been published in SSCI 2020.

Candidate's Declaration



I declare that I have complied with the Thesis Examination Procedure.

Abstract

Machine learning algorithms usually have a number of hyperparameters. The choice of values for these hyperparameters may have a significant impact on the performance of an algorithm. In practice, for most learning algorithms the hyperparameter values are determined empirically, typically by search. From the research that has been done in this area, approaches for automating the search of hyperparameters mainly fall into the following categories: manual search, greedy search, random search, Bayesian model-based optimization, and evolutionary algorithm-based search. However, all these approaches have drawbacks — for example, manual and random search methods are undirected, greedy search is very inefficient, Bayesian model-based optimization is complicated and performs poorly with large numbers of hyperparameters, and classic evolutionary algorithm-based search can be very slow and risks falling into local optimal value.

In this thesis we introduce three improved evolutionary algorithms applied to search for high-performing hyperparameter values for different learning algorithms. The first, named EWLNB, combines Naive Bayes and lazy instance-weighted learning. The second, EMLNB, extends this approach to multiple label classification. Finally, we further develop similar algorithms, named SEODP, for optimizing hyperparameters of deep networks. The experiments show it is useful on a real-world application for donor detection.

EWLNB is a differential evolutionary algorithm which can automatically adapt to different datasets without human intervention by searching for the best hyperparameters for the models based on the characteristics of the datasets to which it is applied. To validate the EWLNB algorithm, we first use it to optimize two key parameters for a locally-weighted Naive Bayes model. Experimental evaluation of this approach on 56 of the benchmark UCI machine learning datasets demonstrate that EWLNB significantly outperforms Naive Bayes as well as several other improved versions of the Naive Bayes algorithms both in terms of classification accuracy and class probability estimation.

We then extend the EWLNB approach in the form of the Evolutionary Multi-label Lazy Naive Bayes (EMLNB) algorithm to enable hyperparameter search for multi-label classification problems.

Lastly, we revise the above algorithms to propose a method, SEODP, for optimizing deep learning (DL) architecture and hyperparameters. SEODP uses a semi-evolutionary and semi-random approach to search for hyperparameter values, which is designed to evolve a solution automatically over different datasets. SEODP is much faster than other methods, and can adaptively select the hyperparameters based on certain deep network architecture. Experimental results show that compared with manual search, SEODP is much more effective, and compared with grid search, SEODP can achieve optimal performance using only approximately 2% of the running time of greedy search. We also apply SEODP to a real-world social-behavioral dataset from a charity organization. This dataset contains comprehensive real-time attributes on potential indicators for candidates to be donors. The results show that SEODP is a promising approach for optimizing deep network (DN) architectures over different types of datasets, including a real-world dataset. In summary, the results in this thesis indicate that our methods address the main drawback of evolutionary algorithms, which is the convergence time,

and show experimentally that evolutionary-based algorithms can achieve good results in optimizing the hyperparameters for a range of different machine learning algorithms.

Acknowledgements

I would like to thank the following people, without whom I would not have been able to complete this research! My supervisor Mike Bain, whose insight and knowledge guide me through this research - both high level and detailed. My colleagues in UNSW where I was working full-time and and study part-time for over three years. And my biggest thanks to my family for all the support you have shown me through this research, the culmination of six years research and study. For my kids, sorry for being even grumpier than normal while I prepared the papers and wrote this thesis! And for my wife Yun, thanks for all your support - I am finally in the stage of submitting the thesis - although I started my PhD student earlier than you, you managed to graduate two years earlier than me.

List of Publications

As First Author:

- Optimizing weighted lazy learning and Naive Bayes classification using differential evolution algorithm, Journal of Ambient Intelligence and Humanized Computing, 2021.
- Using a Semi-Evolutionary Algorithm to Optimize Deep Network Hyper-Parameters with an Application to Donor Detection –SSCI 2020.
- Using Evolutionary Multi-label Classification to Predict Malfunctions in Virtual Storage Systems Monitoring, IJCAI-19 (ERA Rank A*).
- Evolutionary Lazy Learning for Naive Bayes Classification - International Joint Conference on Neural Networks – IJCNN 2016 (ERA Rank A).

As Co-Author:

- Residual-Duet Network with Tree Dependency Graph Representation for Question-answering Sentiment Analysis, SIGIR 2020(ERA Rank A*).
- Case Retrieval Based on Formal Concept Analysis, Journal of Computational and Theoretical Nanoscience, July 2016.

- Similarity model based on CBR and FCA, 2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD).

Contents

List of Publications	v
List of Figures	xiv
List of Tables	xv
1 Introduction	1
1.1 Evolutionary Weighted Lazy Naive Bayes	2
1.2 Multi-label Classification for VSS	4
1.3 Optimizing Hyperparameters for Deep Networks	7
1.4 Overview of Thesis Chapters	9
2 Related Work	10
2.1 Naive Bayes	10
2.2 Improved Naive Bayes	11
2.2.1 Structure extension	12
2.2.2 Attribute selection	12
2.2.3 Attribute weighting	12
2.2.4 Instance selection	13

2.2.5	Instance weighting	13
2.3	Lazy Learning	14
2.4	Differential Evolutionary Algorithms	15
2.5	Virtual Storage Monitoring System (VSMS)	16
2.5.1	Performance Monitoring	17
2.5.2	Disk Monitoring	18
2.6	Multi-label Classification	22
2.7	Deep Learning	22
2.8	Parameter Optimization	23
3	Evolutionary Weighted Lazy Naive Bayes (EWLNB)	27
3.1	Preliminaries	27
3.1.1	Notations and Definitions	27
3.1.2	Naive Bayes	28
3.2	Weighted Lazy Naive Bayes	29
3.3	EWLNB: Evolutionary Weighted Lazy Naive Bayes	33
3.3.1	Self-adaptive Parameter Optimization	33
3.3.2	EWLNB Framework	36
3.4	Experiments	39
3.4.1	Datasets and Parameters Setting	39
3.4.2	Baseline Methods	39
3.4.3	Evaluation Criteria	40
3.4.4	UCI benchmark learning tasks	42
3.4.5	Convergence and Learning Curves	45
3.5	Time Complexity	66

3.6	Effectiveness of EWLNB	67
4	VSMS and EMLNB	68
4.1	Virtual Storage Monitoring System	68
4.1.1	Main Technologies in VSMS	68
4.1.2	Data Collection	70
4.1.3	Data Visualization	73
4.1.4	User Interface	73
4.2	Evolutionary Multi-label Classification	81
4.2.1	Data Captured through VSMS	81
4.2.2	The Framework of MLNB	81
4.2.3	The Framework of EMLNB	82
5	Semi Evolutionary Algorithm to Optimize Deep Network Hyperparameter (SEODP)	85
5.1	Definitions	85
5.2	SEODP Algorithm	87
5.2.1	SEODP Architectures	87
5.2.2	The SEODP Framework	89
5.3	Experiments	92
5.3.1	Datasets	92
5.3.2	Experimental Platform	95
5.3.3	Convergence Learning Curves of SEODP	96
5.3.4	Analysis of Hyperparameters vs. Accuracy	97
5.3.5	SEODP vs. Greedy Search	112

6	Conclusions and Future Work	115
6.1	Contributions	116
6.1.1	EWLNB	116
6.1.2	EMLNB	116
6.1.3	SEODP	117
6.1.4	Summary	117
6.2	Limitations and Future Work	117
6.2.1	EWLNB	118
6.2.2	EWLNB	118
6.2.3	EWLNB	119
6.2.4	Summary	119

List of Figures

2.1	The Architecture of Ganglia. Ganglia is a hierarchical system, which can exhibit high complexity with scale.	19
2.2	An example of the interface for a SMART application.	20
2.3	A demonstration graph of system monitoring data created by MRTG. . .	21
3.1	EWLNB vs. competing algorithms: classification accuracy (ACC). . . .	50
3.2	EWLNB vs. competing algorithms: classification accuracy (ACC) — (continued).	51
3.3	EWLNB vs. competing algorithms: classification accuracy (ACC) — (continued).	52
3.4	EWLNB vs. competing algorithms: classification accuracy (ACC) — (continued).	53
3.5	EWLNB vs. competing algorithms: area under the ROC curve (AUC). .	54
3.6	EWLNB vs. competing algorithms: area under the ROC curve (AUC) — (continued).	55
3.7	EWLNB vs. competing algorithms: area under the ROC curve (AUC) — (continued).	56

3.8	EWLNB vs. competing algorithms: area under the ROC curve (AUC) — (continued).	57
3.9	Convergence learning curves of EWLNB for 4 datasets with large numbers of instances (a) artificial: 10218 instances.	58
3.10	Convergence learning curves of EWLNB for 4 datasets with large numbers of instances (b) letter: 4000 instances.	59
3.11	Convergence learning curves of EWLNB for 4 datasets with large numbers of instances (c) optdigits: 5620 instances.	60
3.12	Convergence learning curves of EWLNB for 4 datasets with large numbers of instances (d) page-blocks: 5473 instances.	61
3.13	Convergence learning curves of EWLNB for 4 datasets with large number of attributes (a) mfeat: 77 attributes.	62
3.14	Convergence learning curves of EWLNB for 4 datasets with large number of attributes (b) movement: 91 attributes.	63
3.15	Convergence learning curves of EWLNB for 4 datasets with large number of attributes (c) anneal.orig: 39 attributes.	64
3.16	Convergence learning curves of EWLNB for 4 datasets with large number of attributes (d) waveform: 41 attributes.	65
4.1	Conceptual overview of a Virtual Storage Monitoring System (VSMS).	69
4.2	The structure of SNMP.	71
4.3	The structure of MIB.	75
4.4	The extended structure of MIB.	76
4.5	Error Types of different priority.	77
4.6	The main user interface of VSMS.	78

4.7	Detailed information on a monitored item in VSMS.	79
4.8	The system architecture of VSMS.	80
5.1	Visualization of the Linear dataset.	93
5.2	Visualization of the Saturn dataset.	94
5.3	Convergence of SEODP on all datasets as learning curves.	96
5.4	“Linear” dataset hyperparameter optimization results with SEODP in 3D with $X = BatchSize$ and $Y = DenseLayerSize$ for $Z = Accuracy$	98
5.5	“Linear” dataset hyperparameter optimization results with SEODP in 2D for $X = BatchSize$ vs. $Y = DenseLayerSize$ with colour show- ing accuracy.	99
5.6	“Linear” dataset results in 2D for $Y = DenseLayerSize$ vs. $X =$ $BatchSize$ with colour showing accuracy.	100
5.7	“Saturn” dataset hyperparameter optimization results with SEODP in 3D with $X = BatchSize$ and $Y = DenseLayerSize$ for $Z = Accuracy$	102
5.8	“Saturn” dataset hyperparameter optimization results with SEODP in 2D for $X = BatchSize$ vs. $Y = DenseLayerSize$ with colour show- ing accuracy.	103
5.9	“Saturn” dataset results in 2D for $Y = DenseLayerSize$ vs. $X =$ $BatchSize$ with colour showing accuracy.	104
5.10	“Skin” dataset hyperparameter optimization results with SEODP in 3D with $X = BatchSize$ and $Y = DenseLayerSize$ for $Z = Accuracy$	106
5.11	“Skin” dataset hyperparameter optimization results with SEODP in 2D for $X = BatchSize$ vs. $Y = DenseLayerSize$ with colour showing accuracy.	107

5.12 “Skin” dataset results in 2D for $Y = DenseLayerSize$ vs. $X = BatchSize$ with colour showing accuracy.	108
5.13 “Donor” dataset hyperparameter optimization results with SEODP in 3D with $X = BatchSize$ and $Y = DenseLayerSize$ for $Z = Accuracy$	109
5.14 “Donor” dataset hyperparameter optimization results with SEODP in 2D for $X = BatchSize$ vs. $Y = DenseLayerSize$ with colour showing accuracy.	110
5.15 “Donor” dataset results in 2D for $Y = DenseLayerSize$ vs. $X = BatchSize$ with colour showing accuracy.	111

List of Tables

3.1	Detailed information of 56 UCI benchmark datasets.	41
3.2	The detailed experimental results: classification accuracy (ACC) %. . .	47
3.3	The detailed experimental results: area under the curve (AUC) %. . . .	48
3.4	Two-tailed t -test on classification accuracy (ACC).	49
3.5	Two-tailed t -test on area under the curve (AUC).	49
3.6	Effectiveness of EWLNB	49
5.1	CNN Architecture	88
5.2	CNN Parameters	88
5.3	MLN	89
5.4	MLN Parameters	89
5.5	Description of datasets.	92
5.6	Best hyperparameters for each dataset	112
5.7	Best vs. worst using greedy search	113
5.8	SEODP vs. Greedy Search	114

Chapter 1

Introduction

Current machine learning algorithms often rely for their success on setting values for hyperparameters — the parameters that the algorithm needs to be specified before training can proceed¹. At the moment finding a “good” set of values for these hyperparameters can only be done either from human knowledge or by some kind of search method. Although there many approaches to this, the problems with current solutions are that it is hard to get human knowledge in general, and existing search methods all have various issues, which we will explain in detail in Chapter 2 on related work. This thesis will address some of these issues, and to do so we will propose a number of methods based on the use of evolutionary algorithms for searching for the values of the hyperparameters.

In this chapter, we will briefly introduce how to use the evolutionary algorithms developed in this thesis — EWLNB, EMLNB and SEODP — to optimize hyperparameters for different models, and motivate the use of these methods by explaining their advantages.

¹These are, in general, completely separate from the parameters of the model that are learned by the algorithm from the training data.

1.1 Evolutionary Weighted Lazy Naive Bayes

Naive Bayes (NB), a classification model based on conditional probability, is the simplest form of Bayesian network classifier according to [25] and continues to be one of the most widely used machine learning algorithms. The problem of Bayesian networks is that they may not scale to large datasets from real-world applications due to the complexity of fitting models to data, given the structure of the model and the way parameters must be associated with that structure (essentially, estimating the full joint probability of the variables in the model).

To address this problem, NB relaxes the restriction of the dependency structures between attributes by simply assuming that attributes are conditionally independent. In this way, NB is able to handle a large number of classification problems and provide both high computational efficiency and good classification accuracy without examining relationships between attributes as shown in [83]. For the above reasons, NB has been found to be particularly suitable for learning tasks with high dimensional data, such as text classification by [15, 49] and fingerprint classification by [35].

By relaxing the restriction of the dependency, NB can be very efficient and accurate. However, in many real-world applications, the assumption of conditional independence in NB is often violated [73, 84]. Many approaches have been proposed to relax this assumption and while retaining the effectiveness of NB. The existing methods, such as [12, 8, 41, 39, 38, 77, 76, 78, 79, 81, 82, 84], can be roughly assigned to five different categories:

1. structure extension;
2. attribute selection;

3. attribute weighting;
4. instance selection;
5. instance weighting.

Structure extension methods extend the structure of NB in some way to represent the dependencies among attributes. Attribute selection (feature selection, e.g.) methods select a subset of the most relevant/discriminative features for model construction. Attribute weighting methods assign different weight values to attributes to improve the quality of the decision. Instance selection methods, such as the k-nearest neighbors or KNN, employ the principle of local learning to find a local training dataset and use it to build a classifier. Instance weighting methods assume that the contribution of each instance is independent but not the same, and assign different weight values to instances.

Among all these algorithms, [24] presented an algorithm LWNB which utilizes a weighted set of training instances to construct a new Naive Bayes model. However, after carefully examine the LWNB algorithm, we found that the performance of LWNB is influenced by parameter K and the calculation of the weighting is very complicated. To solve this problem, we propose a new algorithm model to weight the instances through expanding the neighbors according to their distance, then we apply EA to automatically learn two hyperparameters for the model.

To be specific, we introduce two important parameters in the model to determine the number of clones for an instance: *threshold* and *weight*. Both *threshold* and *weight* are real numbers, the *threshold* parameter acts as a threshold of the distance, only the neighbors within *threshold* will be cloned. This operates similarly to the K parameter of KNN algorithm - when K is changed, the result of the classification might change as well. The *weight* parameter actually acts as a zoom to scale the distance to calculate

the number of clones according to the characteristic - such as density of the dataset.

The advantages of EWLNB can be summarized as follows:

- EWLNB is a data-driven, self-adaptive method, which does not require explicit understanding of specification of datasets; and
- EWLNB is simple and can handle different learning tasks such as NB and KNN.

Experiments over different datasets in this thesis will demonstrate that the proposed EWLNB algorithm successfully finds the optimal parameters for the model.

1.2 Multi-label Classification for VSS

People have become used to uploading and downloading things from many different kinds of “Clouds” such as Dropbox, Amazon Web Service (AWS), and Google Drive. All these convenient and effective “Clouds” can be regarded as one of the important applications of Virtual Storage Systems (VSS) [72, 22, 7]. In brief, VSS is a type of promising technology for storage which is composed of numerous disks and connected by Internet. To be more specific, VSS is generally a huge array of disks which applies virtual technology to abstract over the details of physical storage media and thus make the storage system easy to use. Considering that VSS is such a complex system which involves so many distributed disks connected by internet, how to monitor the performance of such a system effectively is particularly important.

The monitoring of a Virtual Storage System basically includes two main parts. The first part is called performance monitoring, which includes the monitoring of CPU load, memory load, network flow, and so on. The other part is disk monitoring, which monitors the parameters and status of the disks in the VSS. There are basically two different

technologies that are widely used in performance monitoring - MRTG (Multi Router Traffic Grapher) [62, 61, 66] and Ganglia [58]. The main drawback of Ganglia is that it uses its own private protocol to collect data, which lacks universality. On the other hand, the main flaw of MRTG is that the monitoring data is stored in a type of log which consumes lots of resource. For disk monitoring, the most commonly used method is S.M.A.R.T (Self-Monitoring, Analysis and Reporting Technology, usually abbreviated as SMART) [59, 2], which can detect and report on various indicators of drive reliability. However, the main drawbacks of SMART can be grouped into two categories:

1. SMART can only list the real-time value of parameters of the disks which does not allow for any visualiation of the data; and
2. SMART is only designed to collect the data from the disks, and does not support any methods to analyze and predict the status of the disks.

To overcome the drawbacks of the situations described above, we propose a novel VSS monitoring system called Virtual Storage Monitoring System (namely VSMS). The proposed VSMS jointly integrates the Multi Router Traffic Grapher (MRTG), Self-Monitoring Analysis and Reporting Technology (SMART) and round-robin database tool (RRDtools) to construct a comprehensive monitoring system for both performance and disk monitoring. To be specific, VSMS can collect data from different devices using Simple Network Management Protocol (SNMP) and visualize and analyze the collected data by MRTG and RRDtools. Comparing with the existing VSS systems, the advantages of our VSMS can be summarized as follows:

1. VSMS uses RRDtools - a circular buffer based database, which makes VSMS more space efficient and thus run faster.

2. RRDtools is good at processing time series data, whereas SMART can only display real-time data.
3. VSMS can unify the data structures of different disk vendors through the use of a configuration file which can enable the processing and display of data in a standard way.
4. VSMS can visualize the collected data and provides an interface for users to monitor the devices in a convenient way.

After we construct the VSMS, we expect it to be very helpful if we can predict any malfunction of the system based on the monitoring data we collect. In our case, as there can be multiple reasons for a malfunction of the system, this actually leads to the definition of a multi-label classification problem. In machine learning, multi-label classification, and the strongly related problem of multi-output classification, are variants of the classification problem where multiple labels may be assigned to each instance.

More formally, multi-label classification is the problem of finding a model that maps an input vector x to a binary vector y (not necessarily of the same dimension — usually, but not always, x will be of lower dimensionality than y). In other words, the model assigns a value of 0 or 1 for each element (label) of y that is true for instance x .

In VSS, malfunction of any individual system component can be for different reasons — for example, due to CPU load, disk damage, and so on. These faults can happen solely or together. Therefore predicting the reasons for the malfunction of the system is a multi-label classification problem [55, 54, 53]. Similarly, we notice that there are also important hyperparameters that need to be optimized in the model. Inspired by the

good performance of EWLNB, we propose to apply and extend our previous methods to obtain a similar solution to solve the problem.

1.3 Optimizing Hyperparameters for Deep Networks

Deep learning (DL) is a sub-category of machine learning that focuses on learning the representation for the problem while simultaneously learning a solution to the problem. The DL extract features from the input data as it transformed through the multiple layers of of a deep artificial neural network. Although the features are typically learned by deep networks with little human domain knowledge, the use of DL has actually dramatically improved state of the art in many applications [4, 32, 34].

However, one of the drawbacks of DL is that the performance of DL algorithms can be sensitive to the architecture (i.e., the structure and configuration of the layers). Also, it is often hard to manually design the architecture due to the number of the hyperparameters required to specify. This become more difficult when the dataset is big. [5, 10, 36].

For the general multi-layer networks, these hyperparameters include the number of layers, the number of hidden nodes for each layer, and so on. For some specific DNs, such as Convolutional Neural Networks (CNNs), these hyperparameters also include the kernel size for a layer, the window size of maxpool layer, the stride of the convolution layer, and so on. Due to the fact that the number of hyperparameters that need to be tuned is large and the evaluation time for each combination of hyperparameters is often quite long, manually selecting a suitable network topology for a new dataset can be quite time-consuming and tedious. This drawback actually makes the optimization of DL particularly difficult and unstable, which hinders DL algorithms from applying to some real life scenarios [11, 63, 18, 19].

Studies of the performance of DN architectures on different datasets have shown some variations in results, where architectures that have good performance in some datasets may not have the same performance in other datasets [13]. In more detail, studies show that the results on one dataset may not transfer to another dataset with different characteristics, such as the prior probability distribution, number of features, number of training examples, and so on.

In most of the real applications, there is no prior knowledge to design the optimal architecture of DN, the selection often rely on subjective previous experience and trial-and-error. Due to the computationally expensive nature of DL algorithms, this can be quite inefficient and time consuming.

The reality is DL researchers have consensus on the DN architectures that work well for certain problems, however, when researchers want to scale the application of DL to more fields, how to find the optimal DN architectures in a specific domain, is still not clear.

To address the above problems, we revised and extended our previous algorithms to enable the application to optimize the hyperparameters for deep networks. We propose a new algorithm, named SEODP, to search for the a set of “good” DN hyperparameters which can automatically adapt to different datasets. In more detail, SEODP is a semi-evolutionary, semi-random algorithm to search the space of hyperparameters for a DN, which can automatically adapt to different datasets. The results of the the experiments in this thesis over different datasets show that SEODP can find the (approximately) optimal combination of hyperparameters in a much more efficient way.

1.4 Overview of Thesis Chapters

The remainder of the thesis is organized as follows. Chapter 2 reviews related work. Algorithm EWLNB is developed in Chapter 3. Algorithm EMLNB is developed in Chapter 4. Chapter 5 introduces the SEODP algorithm. The thesis concludes in Chapter 6 by summarising the results and discussing some directions for further research.

Chapter 2

Related Work

Our work in this thesis mainly comprises the three parts introduced in the previous chapter: Evolutionary Weighted Lazy Naive Bayes (EWLNB), Multi-label Classification for Virtual Storage Monitoring System (EMLNB), and SEODP to search and find the optimal DN hyperparameters. Our algorithms in each of these parts depends on several concepts and methods from the machine learning literature. In this chapter we review relevant related work to support each of these three parts which come later in the thesis.

2.1 Naive Bayes

A Bayesian network is a probabilistic graphical model that represents a set of variables and their conditional dependencies via a directed acyclic graph (DAG). Bayesian networks are ideal for taking an event that occurred and predicting the likelihood that any one of several possible known causes was the contributing factor. For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of

the presence of various diseases. However, the problem of Bayesian Networks are that they may not scale to the kind of large datasets that occur in real-world applications due to the complexity of modelling the complete joint distribution, which even when structured by the graph can have exponential complexity.

To address this problem, Naive Bayes (NB) was proposed. NB is the simplest form of Bayesian Network classifier according to [25] and continues to be one of the most widely used machine learning algorithms. NB relaxes the restriction of the dependency structures between attributes by simply assuming that attributes are conditionally independent, although this assumption is often incorrect in practice. In this way, NB is able to handle a large number of classification problems and provide both high computational efficiency and good classification accuracy without examining relationships between attributes as shown in [83]. For the above reasons, NB has been found to be particularly suitable for learning tasks with high dimensional data, such as text classification by [15, 49] and fingerprint classification by [35],

2.2 Improved Naive Bayes

Given the limitations of the NB conditionally independence assumption as we discussed, many approaches have been proposed to relax the assumption of independence and meanwhile retaining the effectiveness of NB. The existing methods can be roughly assigned to five different categories: (1) structure extension; (2) attribute selection; (3) attribute weighting; (4) instance selection; and (5) instance weighting.

2.2.1 Structure extension

Structure extension methods use directed arcs to represent the dependencies among attributes, which relaxes the assumption. For example, Tree Augmented Naive Bayes (TAN) extends the structure of NB into a tree-like structure, where an attribute can be connected by a directed arc [25]. Numerous TAN variants have been researched in the literature. Friedman [25] presented a TAN variant (simply CL-TAN) to learn tree-structured Bayesian networks based on the ChowLiu algorithm [16]. Keogh [48] proposed a Super-parent algorithm to learn TAN (simply SP-TAN), which is a greedy heuristic search algorithm with high time complexity.

2.2.2 Attribute selection

Attributes in high dimensional data are often correlated, redundant or noisy, which may result in over-fitting, low efficiency and poor performance in some learning tasks [14, 52]. Attribute selection (*i.e.*, feature selection) methods select a subset of the most relevant/discriminative attributes and eliminate those with little or no predictive information based on certain criteria. [51] proposed an algorithm called Selective Bayes Classifier (SBC). The forward greedy search mechanism used in SBC selects a subset of the most informative attributes from all of the attributes.

2.2.3 Attribute weighting

In most learning tasks, all attributes in a certain dataset are treated with the same weighting, while in reality, different attributes may make different contributions to the labels. To tackle this issue, attribute weighting methods assign different weight values to attributes to improve the quality of decisions [84]. Attribute weighting is quite similar to

feature selection, where weight values of informative attributes are set to 1, and those attributes with little or no predictive information are set to 0. To assign proper weight values for weighted Naive Bayes, lots of algorithms have been proposed to evaluate the relative importance of attributes, such as gain ratio ([85]), correlation-based algorithm ([27]), mutual information ([44]), Relief attribute ranking algorithm ([67]) and evolutionary algorithm [82, 83].

2.2.4 Instance selection

One practical problem in learning a Bayesian Network classifier is the high variance that results from the insufficiency of training data, especially when we want to build a *local* NB model over the original training dataset. To this end, instance selection methods (*i.e.*, local learning methods) employ the principle of local learning to select some instances to build the classifier [42]. One type of instance selection method is similar to data expansion, which adds the clones of instances to the original training set to promote the learning process. For example, [46] proposed a novel approach, namely Instance Cloning Local Naive Bayes (ICLNB), to deal with the problem of the insufficiency of training data. The experimental results in their studies achieved superior performance to other baselines in term of classification accuracy.

2.2.5 Instance weighting

Like attribute weighting and attribute selection, instance weighting is a generalized form that is different from instance selection but aims to give improved performance. Instance weighting methods attempt to assign different weight values to different instances to determine their worth to the labels. [24] presented an algorithm called Locally Weighted

Naive Bayes (LWNB), which uses a weighted set of training instances to construct a new naive Bayes model. In LWNB, K nearest neighbors of the test instance are first found and each of them is weighted in terms of its distance from the test instance. The performance of this method was verified on a large number of benchmark classification datasets from the UCI data repository.

2.3 Lazy Learning

Generally speaking, machine learning algorithms can be split into two categories: (1) eager learning, and (2) lazy learning. Most of the learning algorithms used in practice are eager learning methods, and NB is typical of this type. Eager learning methods try to build a generalized model by using the training instances, which then can be used to predict the label of test instances. The other type of learning methods are lazy learning methods. Lazy learning methods [88, 90], also called local learning methods, form part of instance selection methods. They generally work by selecting the k nearest instances of the test instance from the training dataset, and the selection is usually based on Euclidean distance, which is a commonly used distance metric. The most basic strategy is to use the k -nearest neighbor (KNN) method. In this case, the local training dataset simply consists of the k nearest selected instances. A variant of this method is commonly used, in which the local training dataset also consists of the k nearest selected instances, but these instances have different sizes of clones according to their weight (i.e., distance between the selected instance and the test instance). The greater the weight is (i.e., the smaller the distance), the more clones of the selected instance will be in the local training dataset.

Compared to an eager learning approach, a lazy learning method is typically more

local, enabling it to focus on the specific instance and minimize the effects of the noise and in some cases reduce the “bias” component of classification error [31]. Recently, lazy learning methods have been widely researched for various machine learning algorithms, including Naive Bayes [42], Support Vector Machine [21], and Artificial Neural Networks. There are several works on lazy Naive Bayes. For example, [40] presented a lazy learning algorithm, named Lazy Naive Bayes (LNB), to extend Naive Bayes for evaluating the performance of area under the curve on 36 UCI benchmark datasets. Also, [43] proposed a lazy learning-based Averaged One-Dependence Estimators (a variant for NB), called Lazy Averaged One-Dependence Estimators (LAODE) to validate its performance on a large number of benchmark datasets. Inspired by above lazy NB algorithms, we propose a novel weighted lazy learning method for Naive Bayes classification, namely WLNb. It is worth noting that two key parameters need to be tuned in WLNb. A detailed description of WLNb can be found in Section 3.2.

2.4 Differential Evolutionary Algorithms

The class of differential evolutionary (DE) algorithms [69] is one of the most promising in evolutionary algorithms (EAs). As a powerful and simple method for searching for optimized values for parameters, DE simulates the behaviors of biological evolution to accomplish the search for the best solution. Generally speaking, DE consists of three major components: (1) a mutation operator; (2) a crossover operator; and (3) a selection operator. In a learning problem, an assignment of values to the parameters to be optimized is simulated as an individual in the population. The mutation operation maintains the diversity of the population, the crossover operation passes outstanding genes to the next generation, and the selection operation selects good individuals with high fitness.

When the stopping condition is satisfied, the best individual (*i.e.*, assignment of values to parameters) can be achieved.

Similar to the Differential Evolution algorithm, other evolutionary algorithms (EAs), such as Genetic Algorithms (GA) [47], Particle Swarm Optimization (PSO) [71] and Artificial Immune Systems (AIS) [83], are all based on the idea of biological evolution to control and optimize the artificial system through a form of evolutionary “search”. Among all the EAs, the differential mutation operation is regarded as the simplest one since the operation has linear complexity. More detailed information on DE will be explained in a later part of the thesis using this algorithm. Besides DE, many effective mutation operators have been proposed in the literature, for example, by [87].

In recent years, DE has been widely investigated due to its simplicity and effectiveness, and its potential has been exploited in lots of applications, such as neural network training [6], feature selection [68], digital filter design [56], and extreme learning machine [89].

Inspired by these advantages of DE, we propose to use Differential Evolution algorithm to optimize two key parameters for WLNB. Our goal in this thesis is to propose an effective self-adaptive network parameter optimized method for WLNB. In general, our proposed EWLNB is a self-adaptive learning framework which can be scaled to different DE variants, as will be described in Chapter 3.

2.5 Virtual Storage Monitoring System (VSMS)

As we discussed in the previous chapter, VSS is a promising type of technology for storage which is composed of numerous disks and connected by Internet. Considering that VSS is such a complex system which involves so many distributed disks connected

by internet, how to monitor the performance of such a system effectively is particularly important.

There are many different types of technologies and architectures of VSMS which are currently used for the purpose of performance monitoring and disk monitoring. In this section, the commonly used technologies will be reviewed.

2.5.1 Performance Monitoring

Performance monitoring is the monitoring of all items which give some indication of the general performance of the system. If the performance monitoring of the system finds something unusual, it may imply that the system is not running in the best way — there may occur an overload or unbalanced load, just for example.

Generally speaking, performance monitoring includes the monitoring of factors like CPU load, memory load, network flow, and so on. The main technologies of performance monitoring are listed as follows:

1. Supermon & Clumon. The main drawbacks are:
 - (a) No exclusive database is included, so an external database must be used;
 - (b) Not robust when performing continuous data collection.
2. Parmon. The main drawbacks are:
 - (a) The Client program can be complicated;
 - (b) There is a higher possibility of breaking off when performing continuous data collection.
3. MRTG. The main drawbacks are:

- (a) Only two graphs are allowed in one page;
- (b) Monitoring data is stored in types of data structures in the log which consume lots of resources.

4. GANGLIA. The main drawbacks are:

- (a) Mainly used in facing large-scale clusters;
- (b) Uses its own private protocol to collect data, which lacks universality;
- (c) High complexity of the software, which is shown in Figure 2.1.

2.5.2 Disk Monitoring

The most commonly used method for disk monitoring is Self-Monitoring Analysis and Reporting Technology (SMART for short). SMART is a monitoring system which detects and reports on various parameters of drive reliability.

We can see one interface of SMART in Figure 2.2 which illustrates these points.

Although SMART is a powerful technology, SMART does, however, exhibit a number of drawbacks, as follows:

1. SMART can only display some real-time parameters of the disks and cannot visualize the data;
2. SMART is designed to collect the data from the disks, but cannot analyze and predict the data.

Due to the above disadvantages of the existing systems and technologies, in this thesis we will propose a novel Virtual Storage Monitoring System (VSMS) which jointly

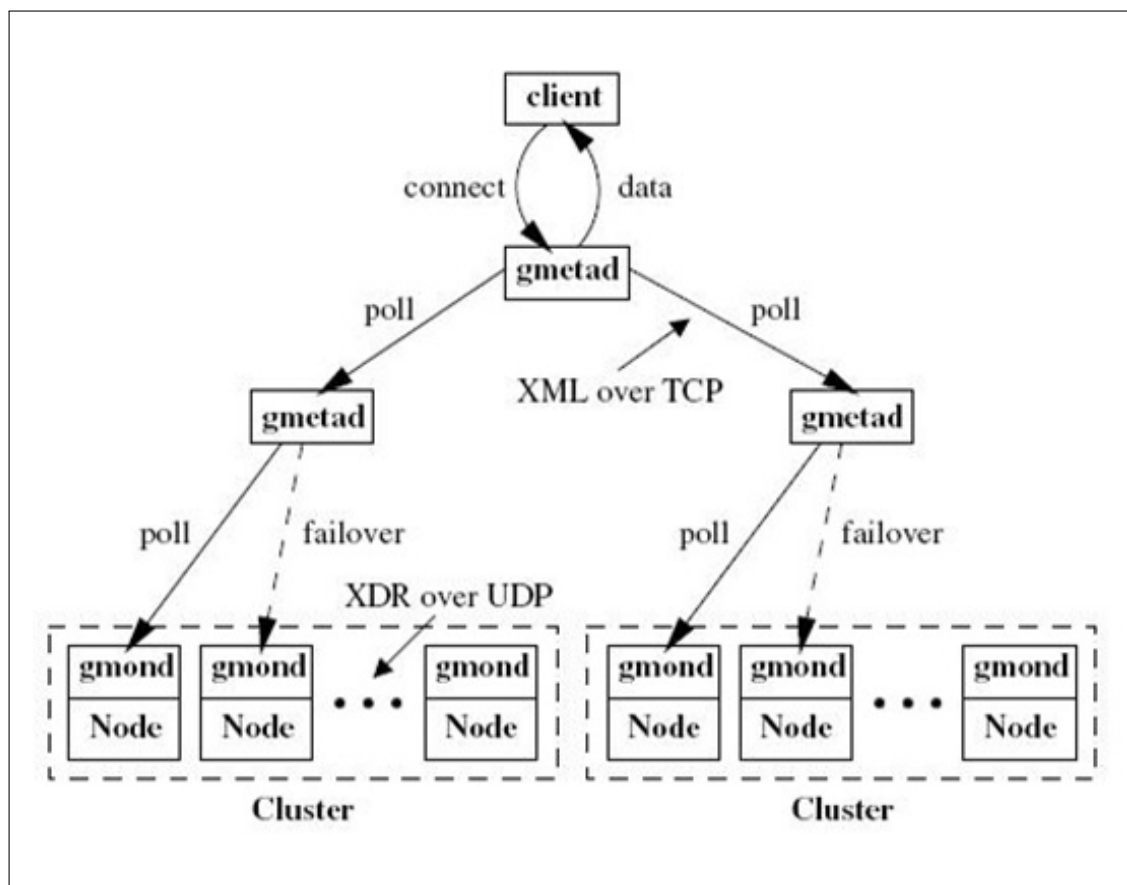
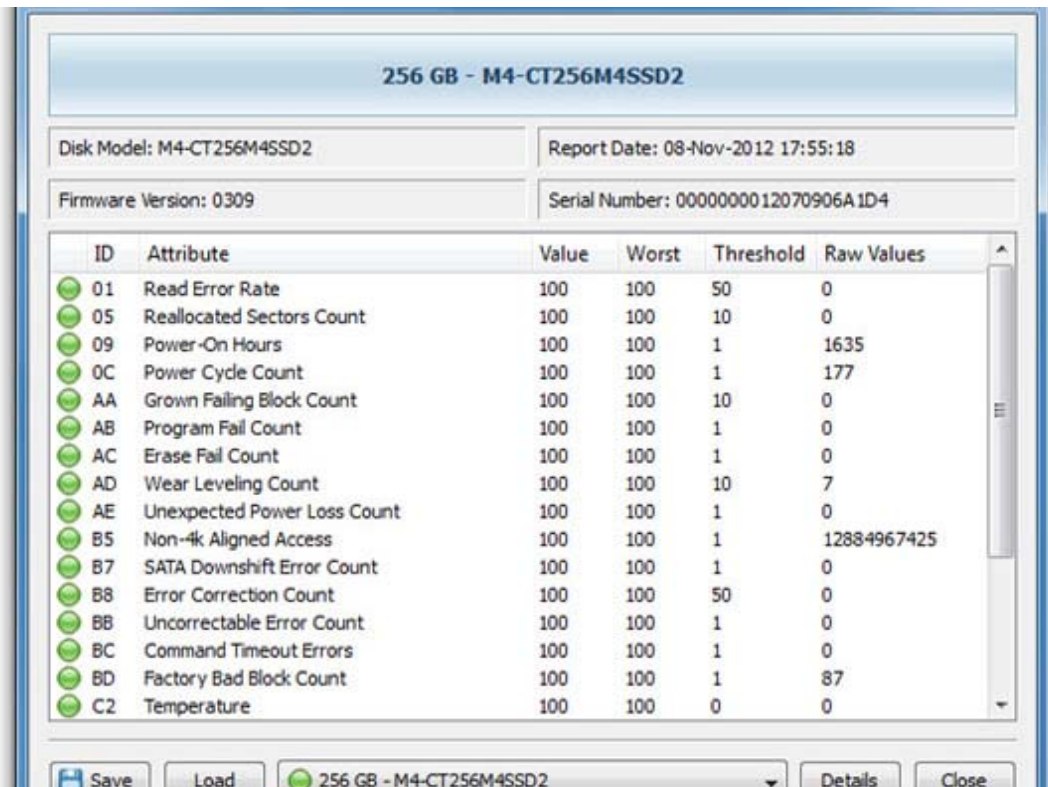


Figure 2.1: The Architecture of Ganglia. Ganglia is a hierarchical system, which can exhibit high complexity with scale.

incorporates MRTG (see Figure 2.3), SMART and RRDtools to construct a comprehensive monitoring system for both performance monitoring and disk monitoring.



256 GB - M4-CT256M4SSD2

Disk Model: M4-CT256M4SSD2 Report Date: 08-Nov-2012 17:55:18

Firmware Version: 0309 Serial Number: 0000000012070906A1D4

ID	Attribute	Value	Worst	Threshold	Raw Values
01	Read Error Rate	100	100	50	0
05	Reallocated Sectors Count	100	100	10	0
09	Power-On Hours	100	100	1	1635
0C	Power Cycle Count	100	100	1	177
AA	Grown Failing Block Count	100	100	10	0
AB	Program Fail Count	100	100	1	0
AC	Erase Fail Count	100	100	1	0
AD	Wear Leveling Count	100	100	10	7
AE	Unexpected Power Loss Count	100	100	1	0
B5	Non-4k Aligned Access	100	100	1	12884967425
B7	SATA Downshift Error Count	100	100	1	0
B8	Error Correction Count	100	100	50	0
BB	Uncorrectable Error Count	100	100	1	0
BC	Command Timeout Errors	100	100	1	0
BD	Factory Bad Block Count	100	100	1	87
C2	Temperature	100	100	0	0

Save Load 256 GB - M4-CT256M4SSD2 Details Close

Figure 2.2: An example of the interface for a SMART application.

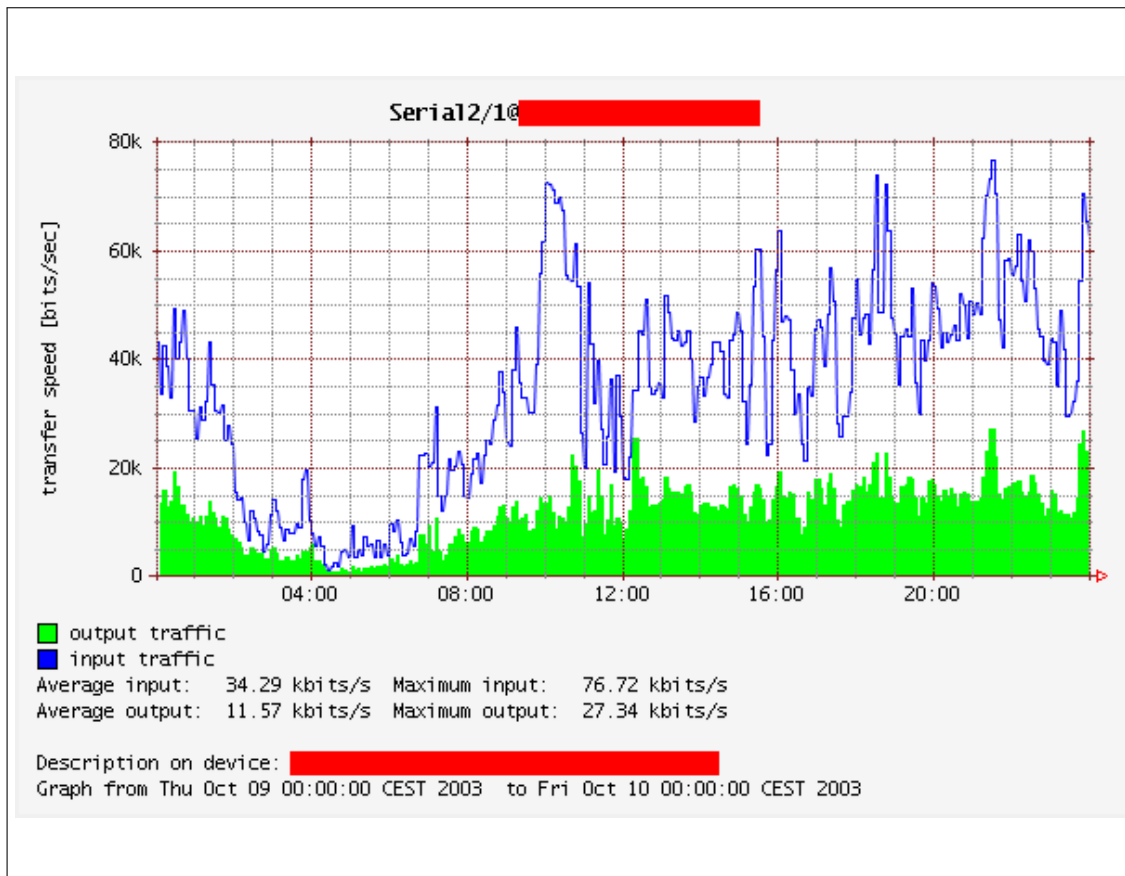


Figure 2.3: A demonstration graph of system monitoring data created by MRTG.

2.6 Multi-label Classification

In machine learning, multi-label classification, and the closely related problem of multi-output classification, are variants of the classification problem where multiple labels may be assigned to each instance.

More formally, multi-label classification is the problem of finding a model that maps an input vector x to a binary vector y . That is, the model assigns a value of 0 or 1 for each element (label) of y that is true for an instance x . In VSS, the malfunction of any individual system component can be for different reasons – for example, due to CPU load, disk damage, and so on.

These faults can happen individually or together. Therefore predicting the reasons for the malfunction of the system is a multi-label classification problem [55, 54, 53]. This motivates our development of an algorithm for the problem of finding hyperparameters for multi-label classification algorithms later in the thesis in Chapter 4.

2.7 Deep Learning

Deep learning (DL) has been making major advances in solving problems in artificial intelligence community over the last decade. It has turned out to be very good at discovering intricate structures in high-dimensional data and is therefore applicable to many domains of science, business and government. In addition to beating records in image recognition [50] and speech recognition [60], it has beaten other machine-learning techniques at predicting the activity of potential drug molecules, analysing particle accelerator data, reconstructing brain circuits [33], and predicting the effects of mutations in non-coding DNA on gene expression and disease. Perhaps more surprisingly, deep

learning has produced extremely promising results for various tasks in natural language understanding [20], particularly topic classification, sentiment analysis, question answering and language translation.

Studies have revealed that the architecture of DN is very important for the performance of the DL algorithms. Therefore, the question of how to choose suitable architectures for DN has been recognized as a very important topic and is attracting more and more attention recently.

2.8 Parameter Optimization

Research on the problem of hyperparameter selection in DL can be mainly put into following categories, following [9, 17]:

1. grid search;
2. manual search;
3. random search;
4. Bayesian model-based optimization (BMBP);
5. evolutionary algorithm (EA)-based search.

Here, manual search (MS) refers to the process of a human manually selecting hyperparameter sets to evaluate. This is straightforward and can make the deployment of the model very quick.

However, the drawbacks are quite obvious for at least, but not confined to, the following reasons:

- MS can be quite subjective, since different persons may have different intuitions or experience leading them to different design decisions for the architecture of the DN;
- MS can be quite time consuming, since even for a single dataset there will be a need to perform lots of different evaluations to find the optimal set of hyperparameters;
- MS is kind of blind, since there is no direction or instruction about how we can select the next combination of hyperparameters based on the result of previously assigned values for hyperparameters;
- Even when an architecture for a DN that works well for one dataset is found, when it is required to be applied to new datasets, people will still need to perform this manual search process again [37].

Grid search (GS) is the process of performing hyperparameter tuning in order to find the optimal values for a given model by exhaustively trying out all sets of hyperparameter values according to a pre-defined search space. To be specific, GS will build a model on each parameter combination possible in the search space. These parameter values can be seen as ordered points in the space defined by a number of dimensions, where each dimension corresponds to a single hyperparameter. Thus the size of the grid, and hence the search space, depends on the density of points (hyperparameter values) along each of these dimensions.

GS is widely used since it is easy and quick to implement, and allows the entire search space to be explored. However, it is important to note that grid-searching can be extremely computationally expensive, especially for a complex DN with lots of hyper-

parameters.

Moreover, for these complex DNs even though many of the hyperparameters may not be important for the performance of the DN, GS will still cover all the combinations of these hyperparameters which is actually a waste of time [64, 70].

On the other hand, random search (RS) may be a much more effective approach. RS falls into the family of methods that do not require complicated calculations and can therefore be very simple and quick to execute. Because of this simplicity, the application of RS is often quite easy and straightforward.

However, random search suffers from being non-adaptive (i.e., the hyperparameter settings that work well on one dataset may not work well on other datasets). What is more, in many cases, RS can be outperformed by a combination of manual and grid search [9, 57].

Although BMBP is a much more advanced method, it is also more complicated to apply and has been found to perform poorly with large numbers of hyperparameters.

Lastly, classic EA-based search can be very slow and has the risk of falling into local optima.

To address all these potential problems, we propose in Chapter 5 an alternative to the above approaches, based on combining some of the ideas in a new methodology. Our approach, called SEODP, uses a semi-evolutionary and semi-random approach to automatically determine the optimal, or approximately optimal, architecture of a DN for a given dataset.

In terms of performance, SEODP improves on the inefficiencies highlighted above of MS and GS, and improves on the accuracy problems of RS. It is also, we believe, not as complicated to apply as BMBP. What is more, SEODP can automatically adapt over different datasets to find the optimal DN architectures. Experiments show that SEODP

achieves good results on a number of different types of datasets, which includes a real-world dataset containing donation information from a charity organization.

Chapter 3

Evolutionary Weighted Lazy Naive Bayes (EWLNB)

In this chapter, we develop the details of the Evolutionary Weighted Lazy Naive Bayes (EWLNB) algorithm, introduced in the Chapter 1 of the thesis.

3.1 Preliminaries

3.1.1 Notations and Definitions

In this section, we introduce some of the notation and important definitions used in this chapter, and the remainder of the thesis, as follows.

- *Training dataset:* D^a represents the training dataset, $D^a = \{(\mathbf{x}_1^a, y_1^a), \dots, (\mathbf{x}_{N_a}^a, y_{N_a}^a)\}$, where $\mathbf{x}_i^a = \{x_{i,1}^a, \dots, x_{i,n}^a\}$ represents the i th instance with n attributes, y_i^a represents the class label of the i th instance, and N_a represents the size of the training dataset. The class label space $Y = \{c_1, \dots, c_m\}$

with m different value.

- *Test dataset:* \mathbf{D}^b represents the test dataset, $\mathbf{D}^b = \{(\mathbf{x}_1^b, y_1^b), \dots, (\mathbf{x}_{N_b}^b, y_{N_b}^b)\}$, where $\mathbf{x}_i^b = \{x_{i,1}^b, \dots, x_{i,n}^b\}$ represents the i th instance with n attributes, y_i^b represents the class label of the i th instance, and N_b represents the size of the test dataset. The class label space $Y = \{c_1, \dots, c_m\}$ with m different value.

3.1.2 Naive Bayes

According to the above definitions, after a transformation by application of Bayes' theorem, Naive Bayes predicts the test instance \mathbf{x}^b by Eq. (3.1):

$$c(\mathbf{x}^b) = \arg \max_{c_j \in \mathcal{C}} P(c_j) P(x_1^a, \dots, x_i^a, \dots, x_n^a | c_j) \quad (3.1)$$

Assuming that all attributes are conditionally independent, given the class, we can obtain the following:

$$P(x_1^a, \dots, x_i^a, \dots, x_n^a | c_j) = \prod_{i=1}^n P(x_i^a | c_j) \quad (3.2)$$

Combining Eqs. (3.1) and (3.2), Eq. (3.1) can be rewritten as follows, which is a typical representation of the Naive Bayes classifier:

$$c(\mathbf{x}^b) = \arg \max_{c_j \in \mathcal{C}} P(c_j) \prod_{i=1}^n P(x_i^a | c_j) \quad (3.3)$$

where $P(c_j)$ denotes the prior probability of class c_j in the training dataset and $P(x_i^a | c_j)$ represents the conditional probability attribute x_i^a conditioned by the given class c_j .

3.2 Weighted Lazy Naive Bayes

In this section, we present a novel algorithm named Weighted Lazy Naive Bayes (WLNb) to extend NB for classification. The most important feature of WLNb is that it finds the nearest instances of a test instance and adds copies of these neighbors (referred to as *clones*) to the training dataset to adjust the weighting of the training dataset. The commonly used distance metric for choosing nearest neighbors is Euclidean distance. The Euclidean distance between instance \mathbf{x}^a and instance \mathbf{x}^b can be given as follows:

$$d(\mathbf{x}^a, \mathbf{x}^b) = \sqrt{\sum_{i=1}^n (x_i^a - x_i^b)^2}; \quad (3.4)$$

where x_i^a and x_i^b are attributes in a training instance and test instance respectively.

In WLNb, the calculation of Eq. (3.4) is crucial for choosing the nearest neighbors of a test instance; however, it does not decide the number of clones of an instance. Here, we introduce two key parameters in WLNb to determine the number of clones of an instance: *threshold* and *weight*. Both *threshold* and *weight* are real numbers. To be more specific, the *threshold* parameter acts as a threshold on the distance, so that only the neighbors with distance less than *threshold* will be considered for cloning. This operates similarly to the K parameter of the KNN algorithm — when we vary K for KNN, the result of the classification might be changed. The *weight* parameter actually scales the distance to calculate the number of clones. Some of the main motivations regarding the choice of these two parameters to optimize are as follows.

- The distribution of different data sets might be quite different. In terms of distance, some data sets can be quite sparse, while others might have quite high density. so it is unrealistic to set a fixed threshold for all different datasets, the threshold needs to to adjust according to the distribution of different datasets. Same

for the *weight* parameter, datasets with different distribution (sparse or dense) should have different weighting to scale the distance and thus calculate the number of cloning. In general, for a sparse dataset, a relatively high weighing should be given, so more instances will be cloned to form the local training dataset. On the contrary, for a dense dataset, a low weighing should be given because there are already enough instances in the local training dataset.

- The distribution of a dataset can be different in different area. In terms of distance, dataset points can be sparse in one area but dense in other areas. To get a good performance of classification for the whole dataset, *threshold* and *weight* needs to overlook the whole dataset instead of only local neighbors, which needs to be optimized according to the distribution of the dataset.
- The distribution of the distance among neighbors can be skewed/even. For example, in binary classification, we find ten neighbors for a test instance, nine of the ten neighbors have value true for the target variable, only one neighbor have value false. However, the neighbor with value false is much more close to the test instance than the other neighbors with value true. How we should classify the test instance in this case? In our algorithm, the *weight* will act as an adapter to deal with these skewness.

Given a training dataset D^a and a test instance \mathbf{x}^b , we first use Eq. (3.4) to calculate the distance $d(\mathbf{x}^b, \mathbf{x}_i^a)$ between the test instance \mathbf{x}^b and each training instance \mathbf{x}_i^a in D^a . If the distance value $d(\mathbf{x}^b, \mathbf{x}_i^a)$ is less than the parameter *threshold*, we calculate the number of clones of this training instance by the following equation:

$$clonenb(\mathbf{x}_i^a) = [(\sqrt{numattri} - d(\mathbf{x}^b, \mathbf{x}_i^a)) * weight]. \quad (3.5)$$

Here, $numattri$ is the number of attributes of the dataset, $\lceil \cdot \rceil$ is the Gauss mark to get the integer of the real number. Since the dataset is discretized in the preprocessing stage, according to the distance function for category data in the WEKA library, the maximum distance between two instance in is $\sqrt{numattri}$ (which means none of the value for each attribute in the two instances is the same). So $\sqrt{numattri}$ is actually used as the upper range of the distance. As $d(\mathbf{x}^b, \mathbf{x}_i^a)$ increase, $[\sqrt{numattri} - d(\mathbf{x}^b, \mathbf{x}_i^a)]$ decrease, and $clonenb(\mathbf{x}_i^a)$ decrease. This actually means the closer a neighbor \mathbf{x}_i^a is to \mathbf{x}^b , the more copy this neighbor will be cloned, vice versa. After this, we add $clonenb(\mathbf{x}_i^a)$ clones of the training instance \mathbf{x}_i^a to dataset D_*^a . Here D_*^a actually acts as a new training dataset which is initialized by an empty set. After the clones of each instance within $threshold$ are added into D_*^a , a NB classifier is deployed on D_*^a to predict the class label $c(\mathbf{x}_i^b)$ of the test instance \mathbf{x}_i^b . The detailed learning procedure of WLNb is depicted in Algorithm 3.

Parameter vector $\mathbf{w} = \{w_1, w_2\}$ contains two key parameters of WLNb, in which w_1 presents parameter $threshold$ and w_2 presents parameter $weight$. The cloning of the instances based on the distance will actually change the prior probability of different levels for the target variable, which theoretically will make the later NB more accurate. In this paper, we aim to learn two optimal parameters for WLNb, which is based on a self-adaptive evolutionary process. Our method takes the objection function (the calculation of classification accuracy) into consideration, which improves the performance of classification accuracy according to the characteristic of a specific dataset.

Algorithm 1: WLNb: Weighted Lazy Naive Bayes

Input:

Training dataset D^a ; a test instance x^b ;
parameter vector $w = \{w_1, w_2\}$;

Output:

The classify result $c(x^b)$;

- 1: $D_*^a \leftarrow \emptyset$;
 - 2: **for all** each x_i^a in D^a **do**
 - 3: Compute $d(x^b, x_i^a)$ using the Euclidean distance function by Eq. (3.4);
 - 4: $threshold \leftarrow w_1, weight \leftarrow w_2$;
 - 5: **for all** the instances y_i with $d(x^b, x_i^a) < threshold$ **do**
 - 6: $clonenb \leftarrow [(\sqrt{numattr_i} - d(x^b, x_i^a)) * weight]$;
 - 7: Adding $clonenb$ instance x_i^a to D_*^a ;
 - 8: **end for**
 - 9: **end for**
 - 10: Build an Naive Bayes classifier on D_*^a and use this NB classifier to give the result of $c(x^b)$;
-

3.3 EWLNB: Evolutionary Weighted Lazy Naive Bayes

The proposed Evolutionary Weighted Lazy Naive Bayes (EWLNB) consists of two major steps to solve the key challenges described in the above sections. We first present a new Weighted Lazy Naive Bayes method (WLNb) to locally learn the class label of a test instance. Then, we use a Differential Evolution-based self-adaptive process (EWLNB) to auto-learn two key parameters in WLNb. We will introduce the learning framework of the proposed EWLNB in the following chapter.

3.3.1 Self-adaptive Parameter Optimization

In the proposed WLNb learning framework, there are two parameters to be optimized. We use a differential evolution-based self-adaptive process to auto-learn their optimal values, namely Evolutionary Weighted Lazy Naive Bayes (EWLNB). In our solution, two key parameters of EWLNB are simulated as an individual for the evolutionary process. Before introducing the details of the self-adaptive parameter optimization process, we define some notation, as follows:

- *Population*: $W = \{w_1, \dots, w_L\}$ represents the population (*i.e.*, the set of individuals), where L denotes the size of the population. $w_i = \{w_{i,1}, w_{i,2}\}$ represents a single individual in the population. $w_{i,1}, w_{i,2}$ indicates the first and second parameters in the first individual. The default value of L is 50.
- *Maximum Generations*: $MaxGen$ represents the maximum number of generations of the evolutionary process. After $MaxGen$ generations of evolution, the algorithm should be able to find the optimal combination of parameters. The default value of $MaxGen$ is 50.

- *Gene*: $w_{i,j}$ indicates the j th gene value of the i th individual.
- *Optimal Individual*: w_c represents the individual which has the best fitness (*i.e.*, best classification accuracy) on the test instances.
- *Mutation Rate*: F indicates the mutation rate in the mutation operation, which describes the probability of an individual being mutated. The default value of F is 0.5.
- *Crossover Rate*: CR indicates the crossover rate in the crossover operation. Crossover rate is the probability of an individual being crossover. The default value of CR is 0.5.

The following two hyperparameters are the ones that we are aiming to optimize using our algorithm

- *Threshold*: ***threshold*** indicates the threshold of the distance, only the neighbors with distance less than *threshold* will be considered for cloning. The range of *threshold* during initialization is $[0, \sqrt{numattr_i}]$ and this is the first parameter we want to optimize using EWLNB.
- *Weight for Cloning*: ***weight*** indicates the weight when an instance is cloned. The range of *weight* is $[1, 50]$ and this is the second parameter we want to optimize using EWLNB.

One point we need to mention is how to choose the best meta parameters of DE (such as $MaxGen$, L , F , CR) is a complicated and well discussed issue and we will not discuss the detail of this here but only use some ordinary value based on the experience. More detailed information can be found in Chapter 2.

To evaluate the performance of EWLNB, we use classification accuracy as the criterion. In other words, the calculation of classification accuracy is used as the fitness function [83, 89], which can be presented as:

$$ACC = \frac{1}{N^b} \sum_{i=1}^{N^b} \delta(c(\mathbf{x}_i^b), \mathbf{t}_i), \quad (3.6)$$

where N^b is the number of test instances, $c(\mathbf{x}_i^b)$ is the classification result of the i th instance in the test dataset and \mathbf{t}_i is the actual class value of the i th training instance. $\delta(c(\mathbf{x}_i^b), \mathbf{t}_i)$ is one if $c(\mathbf{x}_i^b) = \mathbf{t}_i$ and zero otherwise. In EWLNB, a good combination of two key parameters should correspond to high classification accuracy. Accordingly, we drive a self-adaptive updating process to obtain a good combination of two key parameters based on the highest classification accuracy as follows:

1) **Parameter Initialization:** During the evolutionary process, we randomly generate a set of individuals $W = \{\mathbf{w}_1, \dots, \mathbf{w}_L\}$ with the population size L , where $\mathbf{w}_i = \{w_{i,1}, w_{i,2}\}$ is the i th individual in the population. The population in t th generation can be represented as:

$$W^t = \begin{bmatrix} w_{1,1}^t & w_{1,2}^t \\ \vdots & \vdots \\ w_{L,1}^t & w_{L,2}^t \end{bmatrix}; \quad (3.7)$$

2) **Parameter Mutation:** To maintain the diversity of the individuals, mutation and crossover operations are applied. For any individual \mathbf{w}_i^t , a new variation individual \mathbf{v}_i^t in t th generation can be generated by mutation operation as follows:

$$\mathbf{v}_i^t = \mathbf{w}_c^t + F \cdot (\mathbf{w}_{r1}^t - \mathbf{w}_{r2}^t); \quad (3.8)$$

where $r1$ and $r2$ are two different integers uniformly chosen from the set

$\{1, 2, \dots, L\} \setminus \{i\}$. w_c^t is the optimal individual with best fitness value. F is the mutation rate.

2) **Parameter Crossover:** After the mutation operation, a binomial crossover operation is used to generate the final trial individual $u_i^t = [u_{i,1}^t, u_{i,2}^t]$, which can be formulated as follows:

$$u_{i,j}^t = \begin{cases} v_{i,j}^t, & \text{if } (rndreal(0,1) < CR \text{ or } j = j_{rand}) \\ w_{i,j}^t, & \text{otherwise} \end{cases}; \quad (3.9)$$

where j_{rand} is an integer randomly chosen in the range[1,2], and $rndreal(0,1)$ is a real number randomly generated from (0,1), CR is the crossover rate.

3) **Parameter Updating:** This process determines whether the trial individual u_i^t (generated from step 2) or the target individual w_i^t (generated from step 1) can survive to the next generation. In this process, a greedy search strategy is adopted. Only the variation individuals with higher ACC performance can replace the target individuals and survive to the next generation.

3.3.2 EWLNB Framework

Our proposed EWLNB method combines (1) WLNB construction (Section 3.2) and (2) adaptive parameter optimization (Section 3.3.1).

The construction of Weighted Lazy Naive Bayes (WLNB) is presented in Algorithm 3. Given a test instance x^b , WLNB first calculates the Euclidean distance between the test instance x^b and each instance x_i^a in the training dataset (line 3). The number of clones of the training instances is then determined (lines 4-8). Lastly, the NB classifier is trained by the extended training dataset and the underlying class label of the test instance is predicted (line 10).

Algorithm 4 describes the detailed process of self-adaptive parameter optimization for WLNB. During the initialization process, each individual w_i in population W will be initialized (line 1). During the *while* loop, the fitness of each individual w_i is calculated firstly by WLNB (line 4). The best individual w_c^t with highest fitness value $f[w_c^t]$ is then selected (line 5). Both mutation and crossover operations are used to maintain the diversity of the population, and variation individuals and trial individuals will be obtained respectively (lines 6-7). A greedy search strategy is applied to acquire the next population (line 8), and the evolutionary process is repeated until the stopping condition ($t \geq MaxGen$) is met. After the evolutionary process has been terminated, the labels of test instances in D^b are predicted by WLNB with the best parameter vector w_c (line 12).

Algorithm 2: EWLNB: Evolutionary Process for WLNB's parameter optimization

Input:

Evolution Population W ; The Size of Population L ;

Maximum Evolution Generation: $MaxGen$;

Training Dataset D^a , Validation Dataset D^b , Test Dataset D^c ;

Output:

Classification Results $Label(\mathbf{x}^c)$ of Test Instance $\mathbf{x}_t^c \in D^c$;

- 1: $W \leftarrow$ The initial $w_{i,j}$ value for each individual w_i is set to a real number;
 - 2: **while** $t \leq MaxGen$ **do**
 - 3: **for** $i=1$ to L **do**
 - 4: $f[w_i^t] \leftarrow$ Use D^a as the training dataset and D^b as the validation dataset to calculate the fitness of w_i^t in the population W^t ;
 - 5: $w_c^t \leftarrow$ Choose the best individual with highest fitness value;
 - 6: $v_i^t \leftarrow$ Apply mutation operation to obtain the variation individual by Eq. (3.8);
 - 7: $u_i^t \leftarrow$ Apply crossover operation to obtain the trial individual by Eq. (5.1);
 - 8: $W^{t+1} \leftarrow$ Apply a greedy search strategy to obtain the next generation. If $f[u_i^t] \geq f[w_i^t]$, individual u_i^t survive to the next generation, otherwise individual w_i^t can be retained.
 - 9: **end for**
 - 10: $t = t + 1$;
 - 11: **end while**
 - 12: $Label(\mathbf{x}^c) \leftarrow$ Apply w_c^t to MLNB to predict the labels of test instances in D^c .
 - 13: **return** The labels of test instances in D^c ;
-

3.4 Experiments

3.4.1 Datasets and Parameters Setting

We implemented the proposed method using the WEKA data mining tool [75]. To validate the performance of EWLNB, we use 56 benchmark datasets obtained from the UCI data repository [3]. Table 3.1 shows the characteristics of the 56 datasets. More details of the datasets can be found on the UCI website¹. In our experiments, we first replace all missing attribute values in our experiment using the unsupervised attribute filter “ReplaceMissingValues” in WEKA, which replaces all missing values of an attribute with the mean of the known values. We then apply the unsupervised filter “Discretize” in WEKA to discretize numeric attributes into nominal or categorical attributes, because the version of NB we use is designed for categorical attributes only.

The four parameters L , $MaxGen$, F and CR in our algorithm are set to 50, 50, 0.5 and 0.5, respectively. All reported results are obtained via 10 runs of 10-fold cross validation and all experiments are conducted on a computer with an Intel(R) Core(TM) 3.30 GHZ CPU and 8 GB RAM.

3.4.2 Baseline Methods

For comparison purposes, we compare EWLNB with several methods as baselines:

- *NB*: the standard Naive Bayes classifier with conditional attribute independence assumption [25];
- *LNB*: Lazy Naive Bayes classifier which calculates the distances of instance through attribute similarity [40];

¹<http://archive.ics.uci.edu/ml/datasets.html>

- *TreeAWNB*: Attribute weighted Naive Bayes with the weighting method according to the degree to which they depend on the values of other attributes [26];
- *MIAWNB*: Attribute weighted Naive Bayes using the mutual information weighted method [45];
- *IWNB*: Naive Bayes classifier with instances weighting [23];
- *KNN*: Instance-based lazy learning classifier [1];
- *TreeAWKNN*: Attribute weighted KNN with the weighting method according to the degree to which they depend on the values of other attributes [26];
- *MIAWKNN*: Attribute weighted KNN using mutual information weighted method [80].

3.4.3 Evaluation Criteria

In our experiments, the selected algorithms are evaluated on the criteria of classification accuracy (measured by ACC) and class ranking performance (measured by AUC). The ACC of each method is calculated by the percentage of correctly predicted samples in the test set, which can be achieved by Eq. (3.6).

In some machine learning applications, learning a classifier with accurate class ranking or class probability distributions is also desirable [86]. For example, in direct marketing, the limitation of the resources might only allow where promotion of the top x% customers during gradual roll-out, or different promotion strategies might be deployed for different customers according to the likelihood that they will buy certain products. To accomplish these learning tasks, ranking customers according to their likelihood of

Table 3.1: Detailed information of 56 UCI benchmark datasets.

Datasets	Instances	Attributes	Classes	Missing	Numeric
anneal	898	39	6	Y	Y
anneal.orig	898	39	6	Y	Y
artificial	10218	8	10	N	N
audiology	226	70	24	Y	N
autos	205	26	7	Y	Y
balance-scale	625	5	3	N	Y
breast-cancer	286	10	2	Y	N
breast-w	699	10	2	Y	N
car	1728	7	4	N	N
climate	540	21	2	N	N
colic	368	23	2	Y	Y
colic.orig	368	28	2	Y	Y
credit-a	690	16	2	Y	Y
cylinder	540	40	2	N	N
diabetes	768	9	2	N	Y
ecoli	336	8	8	N	N
Energy1	768	9	37	N	N
Energy2	768	9	38	N	N
glass	214	10	7	N	Y
hayes-roth	160	5	3	N	N
heart-c	303	14	5	Y	Y
heart-h	294	14	5	Y	Y
heart-statlog	270	14	2	N	Y
hepatitis	155	20	2	Y	Y
hypothyroid	3772	30	4	Y	Y
ionosphere	351	35	2	N	Y
iris	150	5	3	N	Y
kr-vs-kp	3196	37	2	N	N
labor	57	17	2	Y	Y
letter	20000	17	26	N	Y
lymph	148	19	4	N	Y
mfeat-f	2000	77	10	N	N
monks	556	7	2	N	N
movement	360	91	15	N	N
mushroom	8124	23	2	Y	N
newthyroid	215	6	3	N	N
optdigits	5620	65	10	N	N
page-blocks	5473	11	5	N	N
pendigits	10992	17	10	N	N
primary-tumor	339	18	21	Y	N
qar	1055	42	2	N	N
robot-24	5456	25	4	N	N
segment	2310	20	7	N	Y
sick	3772	30	2	Y	Y
sonar	208	61	2	N	Y
soybean	683	36	19	Y	N
spectrometer	531	102	48	N	N
splice	3190	62	3	N	N
steel	1941	34	2	N	N
texture	5500	41	11	N	N
vehicle	846	19	4	N	Y
vote	435	17	2	Y	N
vowel	990	14	11	N	Y
waveform	5000	41	3	N	Y
zoo	101	18	7	N	Y
credit-b	1000	21	2	N	Y

buying is more useful than simply classifying customers as buyers or non-buyers [83]. To evaluate the classifier performance in terms of class ranking and class probability distributions, we use AUC, which is calculated as follows:

$$E = \frac{P_0 - t_0(t_0 + 1)/2}{t_0 t_1} \quad (3.10)$$

where t_0 and t_1 are the number of negative and positive instances, respectively. $P_0 = \sum r_i$, with r_i denoting the rank of the i th negative instance in the ranked list. It is clear that AUC is essentially a measure of the quality of ranking. The above measure can only deal with the two-class problem. For multiple classes, [28] have proposed an extension to the 2-class AUC measure, calculated by:

$$E' = \frac{2}{g(g-1)} \sum_{i < j < L} E(c_i, c_j) \quad (3.11)$$

where g is the number of classes and $E(c_i, c_j)$ is the AUC of each pair of classes c_i and c_j .

3.4.4 UCI benchmark learning tasks

In this section, we first report the performance of standard NB and improved NB methods (including LNB, TreeAWN, MIAWN and IWN). We then present the performance of the standard KNN and improved KNN algorithms (including TreeAWKNN and MIAWKNN). Lastly, we compare the performance of the proposed EWLNB with other baselines.

Improved NB vs. standard NB

Tables 3.2 and 3.3 respectively show the detailed classification accuracy and area under the curve results of all compared methods. The respective standard deviation values are

also given in Tables 3.2 and 3.3. Tables 3.4 and 3.5 respectively illustrate the compared results of win/tie/loss record on classification accuracy and area under the curve. On each $w/t/l$ record, the algorithm in the corresponding row wins in w datasets, ties in t datasets, and loses in l datasets on the 56 UCI datasets, compared to the algorithm in the corresponding column. In general, the comparisons between the improved NB classifiers and the standard NB classifier can be summarized as follows:

1. LNB significantly outperforms NB on ACC (20 wins and 0 lose). In contrast, LNB shows its inferior performance on AUC (0 win and 40 loses).
2. TreeAWN shows comparative performance to NB in terms of ACC (5 wins and 2 loses), and shows great superiority to NB with respect to AUC (19 wins and 0 loses).
3. IWNB is slightly superior to NB with respect to ACC (8 wins and 0 loses). By contrast, IWNB is inferior to NB on AUC (1 win and 13 loses).
4. MIAWNB is an ineffective attribute weighting method for NB to improve classification accuracy performance. It shows inferior performance on ACC (5 wins and 9 loses). However, MIAWNB is slightly superior to NB on AUC (12 wins and 4 loses).

Improved KNN vs. Standard KNN

In this section, we report the comparative results between the standard KNN and the improved KNN methods. The detailed compared results with respect to ACC and AUC are shown in Tables 3.2 and 3.3. The two-tailed t -test results on ACC and AUC are reported in Tables 3.4 and 3.5 respectively. From these figures, we can see that not all

existing improved KNN methods can achieve good performance in different measures. Overall, the highlights can be presented as follows:

1. Compared to KNN, TreeAWKNN wins 4 datasets, ties 38 datasets and loses 4 datasets on ACC. In terms of AUC, TreeAWKNN slightly outperforms KNN (9 wins and 2 losses).
2. In terms of ACC, MIAWKNN shows comparative performance to KNN (10 wins and 10 losses). By contrast, MIAWKNN is inferior to KNN on AUC (3 wins and 8 losses).

Proposed EWLNB vs. Other Baselines

In Figures 3.1 to 3.4 and Figures 3.5 to 3.8, we comparatively report the performance of EWLNB with other baselines for all benchmark UCI datasets in terms of classification accuracy (ACC), and area under the ROC curve (AUC), respectively.

Note that the majority of data points fall below the diagonal line $y = x$ in these figures, illustrating the superior performance of the proposed EWLNB.

More illustrations of performance in terms of ACC and AUC are presented in Tables 3.2 and 3.3 (detailed results) and Tables 3.4 and 3.5 (two-tailed t -test results). In both Tables 3.2 and 3.3, the symbols • and ◦ represent statistically significant upgradation and degradation over the proposed EWLNB with a 95% confidence level. Our experimental results indicate that EWLNB shows significant gains compared to the other baseline methods on the above two evaluation criteria. In summary, our experimental results show:

1. EWLNB shows its superiority compared to NB. It significantly outperforms NB on both ACC and AUC (20 wins and zero losses).

2. Compared to the improved NB methods, EWLNB demonstrates the superior performance. EWLNB outperforms LNB (11 wins and 2 losses), TreeAWN (21 wins and 1 loss), MIAWN (14 wins and 8 losses) and IWN (14 wins and 8 losses). Similar superior performance can also be seen on AUC.
3. EWLNB shows good performance compared to KNN, TreeAWKNN and MIAWKNN. In terms of ACC, EWLNB outperforms KNN (19 wins and 3 losses), TreeAWKNN (14 wins and 8 losses) and MIAWKNN (15 wins and 4 losses). For AUC, EWLNB outperforms KNN (18 wins and 2 losses), TreeAWKNN (16 wins and 7 losses) and MIAWKNN (15 wins and 4 losses).

3.4.5 Convergence and Learning Curves

To investigate the convergence of the EWLNB, we report the relationship between the number of generations and the classification accuracy on 4 datasets with a large number of instances (these are denoted as “artificial”, “letter”, “optdigits” and “page-blocks”) and another 4 datasets with large number of attributes (these are denoted as “mfeat”, “movement”, “anneal.orig” and “wave”).

These convergence results are shown in Figures 3.9 to 3.12 for large numbers of instances, and 3.13 to 3.16 for large numbers of attributes. Each point on the curves of Figures 3.9 to 3.12 and Figures 3.13 to 3.16 corresponds to the mean classification accuracy from 10-fold cross validation under the underlying iteration with the current optimal individual, (*i.e.*, set of hyperparameter values).

The learning curves of EWLNB on eight datasets ascend rapidly in the preceding generations, and achieve higher classification accuracy in the latter generations, converging to a steady state.

To further investigate the convergence of EWLNB, we observed the “letter” dataset, which is a multiple classification dataset (26 classes) with 20000 instances and 17 attributes. Our result in Figure 3.10 shows that EWLNB achieves 84.30% classification accuracy, which is significantly higher than NB’s accuracy of 66.2% on the same dataset. The accuracy of the final convergence is better than LNB (76.00%), TreeAWN(66.3%), MIAWN(68%), IWN(67.1%), KNN(71.2%), TreeAWKNN(72.8%) and MIAWKNN(73.3%).

Similar improvements can also be observed from the remaining datasets. In some situations, our EWLNB obtains a better classification accuracy result than other methods compared in previous iterations, which illustrates the good convergence performance of EWLNB.

Table 3.2: The detailed experimental results: classification accuracy (ACC) %.

Datasets	EWLNB	LNB	TreeAWNB	MIAWNB	TreeAWKNN	MIAWKNN	KNN	NB	IWNB
anneal	97.44 ±1.83	97.44 ±1.58	94.65 ±2.72	87.73 ±5.50	97.55 ±1.47	97.10 ±1.68	95.88 ±1.97	94.32 ±2.38	97.33 ±1.59
anneal.orig	89.53 ±3.40	88.53 ±3.05	87.42 ±2.60	78.28 ±4.43	86.98 ±3.30	87.97 ±2.15	84.41 ±3.30	87.53 ±4.69	88.42 ±2.77
artificial	68.57 ±1.83	47.72 ±1.19	36.74 ±0.90	36.23 ±1.04	56.21 ±1.84	52.64 ±1.26	56.14 ±2.49	36.40 ±1.00	35.90 ±0.85
audiology	75.61 ±6.93	78.32 ±7.12	73.40 ±7.13	71.17 ±8.16	65.04 ±7.91	69.47 ±6.35	58.79 ±8.30	71.23 ±7.03	78.32 ±7.12
autos	72.98 ±13.25	74.57 ±10.59	66.29 ±12.42	65.79 ±10.33	68.31 ±7.56	65.86 ±8.17	62.52 ±8.03	64.83 ±11.18	66.69 ±13.87
balance-scale	90.24 ±1.93	91.04 ±1.55	90.72 ±1.67	90.40 ±1.50	79.67 ±5.45	78.24 ±2.56	83.84 ±4.71	91.36 ±1.38	90.40 ±1.52
breast-cancer	71.06 ±12.83	71.74 ±8.74	72.76 ±8.72	70.27 ±10.95	73.46 ±6.96	70.96 ±5.52	73.09 ±4.25	72.06 ±7.97	72.09 ±8.78
breast-w	97.42 ±1.76	97.42 ±1.89	97.42 ±1.76	97.28 ±1.84	94.28 ±3.93	95.42 ±3.49	93.99 ±3.42	97.28 ±1.84	97.42 ±1.89
car	89.93 ±3.83	86.92 ±1.73	84.37 ±2.33	81.37 ±1.97	90.97 ±2.04	85.82 ±2.84	93.52 ±1.32	85.53 ±2.49	85.19 ±1.73
climate	87.78 ±4.11	88.89 ±1.95	85.19 ±4.36	77.22 ±8.86	91.11 ±1.91	90.74 ±2.31	91.48 ±0.96	87.78 ±4.11	88.33 ±3.27
colic	79.35 ±4.07	80.99 ±5.80	80.98 ±5.07	83.67 ±7.86	84.76 ±5.07	83.68 ±5.90	83.13 ±6.29	78.81 ±5.05	80.17 ±5.52
colic.orig	76.88 ±5.40	75.81 ±5.19	74.99 ±4.62	73.91 ±3.94	79.31 ±6.06	76.37 ±5.10	69.82 ±3.40	75.26 ±5.26	74.99 ±5.43
credit-a	85.65 ±3.95	84.93 ±3.94	86.09 ±4.54	86.09 ±4.54	86.52 ±4.10	85.07 ±4.21	86.09 ±4.39	84.78 ±4.28	84.78 ±4.11
credit-g	76.60 ±4.45	76.80 ±4.18	74.70 ±4.22	71.50 ±4.43	74.20 ±2.62	74.70 ±4.81	71.90 ±3.28	76.30 ±4.76	76.50 ±4.81
cylinder	77.04 ±4.80	81.30 ±6.95	77.04 ±4.95	79.81 ±6.20	63.52 ±5.31	67.41 ±4.80	75.00 ±6.61	75.37 ±4.86	79.81 ±6.26
diabetes	75.92 ±5.89	75.78 ±5.94	75.66 ±6.58	76.57 ±5.20	73.44 ±4.12	73.44 ±3.83	69.02 ±2.19	75.40 ±5.85	74.62 ±5.85
ecoli	84.23 ±5.56	85.42 ±5.53	84.81 ±7.51	83.92 ±6.34	72.94 ±5.55	74.42 ±6.95	73.50 ±4.74	82.14 ±5.93	84.80 ±6.43
energy1	63.15 ±5.87	58.85 ±4.30	47.00 ±3.55	45.18 ±3.86	61.20 ±5.09	59.90 ±4.96	59.90 ±5.03	45.05 ±3.92	47.92 ±3.88
energy2	51.17 ±4.13	52.60 ±4.52	46.88 ±6.10	42.71 ±5.04	52.99 ±3.39	52.21 ±4.07	51.68 ±4.84	46.10 ±4.68	48.05 ±5.64
glass	58.44 ±8.06	59.85 ±7.39	58.90 ±10.43	58.05 ±11.11	61.21 ±8.84	58.01 ±12.39	57.06 ±8.03	60.32 ±9.69	59.83 ±11.97
hayes-roth	83.75 ±10.29	83.13 ±10.23	84.38 ±10.31	85.00 ±10.29	66.25 ±11.86	66.25 ±8.94	67.50 ±10.12	82.50 ±11.33	85.00 ±11.10
heart-c	83.13 ±5.84	81.48 ±6.21	83.14 ±4.65	83.13 ±5.17	81.13 ±9.25	81.77 ±8.29	81.09 ±9.77	84.14 ±4.16	81.83 ±6.57
heart-h	83.70 ±5.15	84.05 ±4.40	83.38 ±6.36	82.70 ±7.41	80.36 ±8.49	81.34 ±6.99	82.02 ±6.06	84.05 ±6.69	84.39 ±4.72
heart-statlog	82.59 ±4.95	82.22 ±5.47	83.70 ±4.68	83.70 ±4.68	80.00 ±5.00	81.11 ±7.50	82.22 ±7.37	83.70 ±5.00	84.07 ±3.92
hepatitis	86.33 ±7.95	85.67 ±7.57	84.46 ±8.72	84.50 ±10.51	81.83 ±6.23	80.63 ±3.21	84.50 ±6.22	83.79 ±8.79	85.71 ±8.09
hypothyroid	92.71 ±1.28	92.84 ±0.88	90.03 ±1.20	77.89 ±1.95	93.27 ±0.66	93.21 ±0.67	93.08 ±0.64	92.79 ±1.02	92.82 ±0.87
ionosphere	91.17 ±3.42	91.44 ±3.82	91.74 ±3.42	91.17 ±3.42	88.61 ±4.23	90.60 ±4.28	89.74 ±2.78	90.89 ±3.49	89.17 ±4.44
iris	94.00 ±8.58	96.67 ±4.71	96.00 ±4.66	96.67 ±4.71	94.00 ±5.84	92.67 ±6.63	93.33 ±6.29	94.67 ±8.20	94.67 ±8.20
kr-vs-kp	93.56 ±1.28	88.67 ±1.64	90.02 ±1.95	91.05 ±1.82	97.97 ±0.68	95.74 ±0.81	95.06 ±1.34	87.89 ±1.81	88.45 ±1.62
labor	93.33 ±11.65	90.00 ±14.05	88.00 ±11.46	90.00 ±14.05	86.00 ±16.39	87.67 ±16.93	85.67 ±14.49	93.33 ±11.65	91.67 ±14.16
letter	80.20 ±2.37	76.00 ±2.23	66.25 ±2.04	68.03 ±2.68	72.83 ±1.83	73.28 ±2.10	71.22 ±2.47	66.15 ±2.15	67.13 ±2.38
lymph	85.00 ±8.63	86.33 ±8.80	85.67 ±8.45	84.33 ±7.53	80.24 ±9.22	80.29 ±8.97	80.86 ±12.02	85.67 ±9.55	86.33 ±8.80
mfeat-f	78.55 ±2.19	79.15 ±2.91	78.10 ±2.11	78.45 ±1.79	76.75 ±0.82	76.00 ±1.96	72.00 ±2.71	77.15 ±1.96	77.30 ±2.37
monks	95.65 ±6.92	75.36 ±2.53	74.64 ±2.15	74.64 ±2.15	92.24 ±5.53	79.73 ±9.81	99.09 ±1.77	74.64 ±2.15	73.74 ±3.09
movement	72.22 ±8.18	80.83 ±5.92	64.44 ±10.04	64.17 ±9.48	50.00 ±9.07	48.89 ±10.73	58.89 ±10.04	63.89 ±10.48	73.61 ±7.55
mushroom	99.38 ±0.65	99.02 ±0.97	97.17 ±1.42	97.97 ±1.36	99.20 ±0.87	99.63 ±0.60	99.75 ±0.32	93.84 ±2.02	97.05 ±1.68
Newthyroid	94.83 ±5.21	92.99 ±5.08	93.44 ±4.60	95.71 ±5.70	84.18 ±6.73	83.29 ±8.00	80.52 ±5.90	92.08 ±4.46	92.99 ±5.08
optdigits	93.88 ±1.13	93.01 ±1.28	92.37 ±1.05	92.30 ±1.02	92.88 ±1.14	94.18 ±0.86	92.83 ±1.03	92.38 ±1.20	92.24 ±1.29
page-blocks	93.55 ±0.81	92.64 ±0.80	91.59 ±1.07	86.79 ±1.10	92.53 ±0.97	92.02 ±1.15	92.87 ±0.61	92.34 ±1.03	92.54 ±0.99
pendigits	94.93 ±0.98	92.86 ±0.71	87.00 ±1.06	86.61 ±0.95	95.89 ±0.51	93.40 ±0.83	95.56 ±0.51	87.04 ±1.23	88.21 ±1.06
primary-tumor	47.78 ±5.25	47.50 ±4.90	46.62 ±4.50	43.67 ±5.46	43.05 ±3.48	44.81 ±5.49	42.47 ±5.67	46.89 ±4.32	46.61 ±5.55
qar	80.29 ±4.42	81.05 ±3.37	79.63 ±5.15	78.40 ±4.97	84.26 ±4.00	80.38 ±3.71	83.41 ±4.28	79.81 ±4.43	80.95 ±3.66
robot-24	87.83 ±1.53	83.84 ±1.79	80.63 ±1.86	82.31 ±1.74	92.93 ±1.44	93.07 ±1.15	91.00 ±1.63	80.57 ±2.02	80.04 ±1.71
segment	93.90 ±1.56	91.26 ±1.67	89.57 ±1.90	87.62 ±2.08	90.52 ±1.31	89.26 ±2.16	89.65 ±1.84	88.92 ±1.95	89.91 ±2.03
sick	97.67 ±0.51	97.08 ±0.54	96.47 ±0.81	96.29 ±0.80	97.99 ±0.59	97.48 ±0.69	97.03 ±0.73	96.74 ±0.53	97.03 ±0.61
sonar	77.02 ±1.72	79.40 ±8.88	74.60 ±11.98	76.05 ±12.58	71.12 ±10.15	73.14 ±9.96	81.33 ±8.42	77.50 ±11.99	77.43 ±8.95
soybean	93.55 ±2.81	94.43 ±2.29	91.79 ±2.54	91.94 ±2.64	90.04 ±3.16	90.03 ±2.86	89.01 ±2.12	92.08 ±2.34	93.99 ±2.36
spectrometer	49.54 ±7.05	52.55 ±4.61	48.21 ±4.46	47.46 ±4.36	52.92 ±4.88	49.15 ±4.72	46.33 ±4.50	46.70 ±4.82	49.71 ±6.47
splice	95.33 ±1.43	95.86 ±0.87	95.74 ±1.11	94.80 ±1.36	89.34 ±1.93	91.07 ±2.18	83.26 ±2.42	95.36 ±1.00	95.55 ±0.94
steel	95.72 ±1.35	96.81 ±1.34	98.15 ±0.81	94.80 ±1.62	100.00 ±0.00	94.44 ±2.10	88.05 ±1.60	95.26 ±1.28	97.27 ±1.14
texture	92.05 ±1.02	90.31 ±0.96	79.60 ±1.46	79.42 ±1.29	95.96 ±0.50	95.51 ±0.87	95.36 ±0.67	79.51 ±1.33	80.15 ±1.24
vehicle	64.66 ±2.87	67.86 ±4.73	62.06 ±3.44	60.29 ±3.90	71.51 ±3.20	65.85 ±4.03	68.68 ±2.74	61.82 ±3.54	61.58 ±2.96
vote	91.74 ±4.06	90.82 ±3.72	91.74 ±3.91	91.97 ±3.61	96.33 ±3.25	95.64 ±3.94	92.90 ±3.61	90.14 ±4.17	90.60 ±3.78
vowel	90.71 ±3.89	87.68 ±2.22	67.98 ±5.06	66.97 ±5.55	73.54 ±5.36	62.73 ±5.25	67.68 ±4.04	67.07 ±4.21	66.26 ±5.72
waveform	83.30 ±4.37	83.50 ±4.72	79.00 ±4.35	78.70 ±3.68	76.70 ±5.83	77.50 ±4.60	74.60 ±4.62	79.70 ±4.00	80.50 ±4.03
zoo	95.18 ±6.65	96.18 ±6.54	95.09 ±5.18	95.00 ±7.07	89.18 ±8.57	83.27 ±11.28	89.18 ±9.78	94.18 ±6.60	96.18 ±6.54

•, ○: Statistically significant upgradation and degradation, respectively.

Table 3.3: The detailed experimental results: area under the curve (AUC) %.

Datasets	EWLNB	LNB	TreeAWNB	MIAWNB	TreeAWKNN	MIAWKNN	KNN	NB	IWNB
anneal	98.88 ± 1.68	97.51 ± 4.26	98.78 ± 1.77	98.83 ± 1.72	96.84 ± 6.96	98.35 ± 3.07	96.70 ± 7.80	98.76 ± 1.84	98.91 ± 1.69
anneal.orig	97.11 ± 5.24	92.06 ± 6.32	97.69 ± 3.19	97.51 ± 3.82	96.68 ± 5.82	97.69 ± 2.37	96.11 ± 6.07	96.79 ± 5.42	98.22 ± 2.07
artificial	95.98 ± 0.63	70.54 ± 0.89	81.04 ± 0.89	80.44 ± 0.98	93.78 ± 0.46	92.79 ± 0.52	93.56 ± 0.48	81.15 ± 0.94	80.93 ± 0.97
audiology	84.24 ± 1.64	82.54 ± 1.64	83.90 ± 1.43	84.21 ± 1.57	84.13 ± 1.57	83.84 ± 1.57	83.18 ± 1.48	83.85 ± 1.44	84.65 ± 1.71
autos	94.34 ± 4.00	89.03 ± 6.28	92.80 ± 3.60	92.44 ± 4.07	92.98 ± 3.20	92.12 ± 3.15	91.85 ± 3.47	91.96 ± 3.32	93.38 ± 3.69
balance-scale	83.43 ± 3.72	75.01 ± 3.30	86.73 ± 4.04	87.12 ± 4.18	63.46 ± 5.74	61.78 ± 3.16	65.61 ± 2.85	85.00 ± 4.03	84.76 ± 4.61
breast-cancer	70.43 ± 14.77	65.24 ± 11.29	71.15 ± 13.67	70.74 ± 13.23	61.43 ± 13.46	64.92 ± 14.36	66.46 ± 11.58	71.32 ± 13.81	70.49 ± 13.74
breast-w	99.26 ± 0.76	97.84 ± 1.44	99.26 ± 0.76	99.21 ± 0.82	98.03 ± 1.68	98.20 ± 2.49	98.86 ± 1.12	99.23 ± 0.83	99.26 ± 0.77
car	97.51 ± 2.35	73.88 ± 3.08	97.16 ± 1.09	97.28 ± 1.08	92.38 ± 3.22	93.63 ± 2.57	97.28 ± 2.06	92.72 ± 1.94	93.01 ± 2.00
climate	74.96 ± 10.04	50.64 ± 3.88	78.82 ± 10.17	82.38 ± 10.72	77.56 ± 10.43	81.06 ± 11.70	68.31 ± 12.64	76.98 ± 10.15	78.04 ± 9.21
colic	85.71 ± 4.50	79.57 ± 6.21	85.93 ± 5.50	87.59 ± 6.01	87.95 ± 5.93	88.01 ± 6.69	87.33 ± 5.42	84.42 ± 5.45	84.11 ± 5.74
colic.orig	83.89 ± 6.96	73.30 ± 6.74	83.13 ± 6.39	82.59 ± 5.43	83.57 ± 6.73	83.50 ± 5.98	77.46 ± 6.31	81.70 ± 7.23	84.09 ± 6.11
credit-a	92.20 ± 2.97	84.60 ± 4.01	92.18 ± 3.25	92.32 ± 3.22	91.35 ± 4.04	92.10 ± 3.76	91.88 ± 3.36	91.97 ± 3.14	91.92 ± 3.22
credit-g	79.07 ± 4.26	70.19 ± 4.50	79.79 ± 4.63	78.83 ± 5.25	76.98 ± 4.64	76.70 ± 6.11	75.99 ± 5.47	79.42 ± 4.52	79.24 ± 4.40
cylinder	86.63 ± 5.31	80.02 ± 7.39	85.57 ± 5.75	87.05 ± 6.17	69.07 ± 7.19	73.76 ± 6.94	83.13 ± 4.53	84.38 ± 5.42	87.65 ± 5.31
diabetes	82.90 ± 4.82	73.94 ± 7.18	83.22 ± 4.95	84.01 ± 4.83	80.86 ± 4.97	80.20 ± 4.62	77.93 ± 5.69	82.74 ± 4.94	82.58 ± 5.33
ecoli	94.84 ± 2.92	91.29 ± 1.98	94.57 ± 2.87	94.26 ± 2.93	90.79 ± 2.28	89.86 ± 2.24	92.57 ± 2.98	94.59 ± 2.67	95.07 ± 2.86
energy1	95.58 ± 0.97	83.19 ± 0.91	93.05 ± 1.42	93.07 ± 1.43	97.06 ± 0.69	96.57 ± 0.98	96.02 ± 0.82	92.96 ± 1.38	93.48 ± 1.53
energy2	93.41 ± 1.66	82.80 ± 1.97	92.66 ± 1.52	92.73 ± 1.55	95.55 ± 0.85	95.37 ± 0.62	94.42 ± 1.19	92.86 ± 1.50	92.91 ± 1.87
glass	86.84 ± 7.33	77.41 ± 4.17	83.25 ± 6.26	84.75 ± 6.08	88.37 ± 2.67	85.25 ± 6.88	85.95 ± 2.64	82.63 ± 6.07	89.00 ± 3.13
hayes-roth	95.00 ± 4.81	86.19 ± 8.33	96.08 ± 3.68	96.48 ± 3.56	84.81 ± 9.60	84.73 ± 8.53	85.74 ± 8.40	95.61 ± 4.28	96.48 ± 3.52
heart-c	84.17 ± 0.43	83.13 ± 0.66	84.17 ± 0.53	84.12 ± 0.54	83.82 ± 0.85	83.81 ± 0.79	83.96 ± 0.78	84.17 ± 0.50	84.13 ± 0.55
heart-h	83.84 ± 0.59	83.16 ± 0.54	83.86 ± 0.69	83.95 ± 0.68	83.65 ± 0.90	83.64 ± 0.76	83.78 ± 0.86	83.92 ± 0.63	83.86 ± 0.67
heart-statlog	91.72 ± 3.90	81.83 ± 5.84	91.72 ± 4.14	91.67 ± 4.55	88.83 ± 6.00	89.19 ± 5.13	90.72 ± 5.36	91.33 ± 5.15	91.11 ± 4.79
hepatitis	88.70 ± 9.41	79.26 ± 16.76	89.34 ± 7.65	89.34 ± 8.19	77.56 ± 10.01	82.89 ± 9.42	85.33 ± 12.56	89.90 ± 8.24	89.25 ± 8.75
hypothyroid	86.94 ± 11.13	78.36 ± 10.25	88.44 ± 9.50	88.77 ± 9.90	84.49 ± 10.96	83.28 ± 13.50	84.22 ± 11.02	87.53 ± 9.20	88.67 ± 9.37
ionosphere	94.84 ± 2.92	89.72 ± 4.75	94.46 ± 3.05	94.15 ± 2.93	92.50 ± 5.81	95.43 ± 3.00	94.29 ± 4.00	93.90 ± 3.21	93.89 ± 2.80
iris	98.93 ± 2.16	98.33 ± 2.36	99.20 ± 1.80	99.20 ± 1.80	98.93 ± 1.99	98.93 ± 1.76	98.47 ± 2.31	98.93 ± 2.16	98.80 ± 2.31
kr-vs-kp	98.40 ± 0.64	88.60 ± 1.63	96.88 ± 0.97	98.08 ± 0.76	99.80 ± 0.15	99.46 ± 0.18	99.05 ± 0.47	95.20 ± 1.28	95.62 ± 1.19
labor	100.00 ± 0.00	88.75 ± 17.13	98.75 ± 3.95	98.75 ± 3.95	92.50 ± 13.44	90.00 ± 24.15	97.50 ± 7.91	98.75 ± 3.95	100.00 ± 0.00
letter	98.57 ± 0.30	87.17 ± 1.26	95.69 ± 0.73	96.10 ± 0.64	96.62 ± 0.73	96.43 ± 0.62	96.88 ± 0.70	95.68 ± 0.73	96.22 ± 0.69
lymph	94.77 ± 4.69	93.19 ± 4.39	95.15 ± 4.88	94.81 ± 4.85	93.30 ± 5.55	93.38 ± 5.19	94.33 ± 5.33	95.01 ± 4.87	94.77 ± 4.52
mfeat-f	96.66 ± 0.67	90.55 ± 2.11	96.43 ± 0.73	96.79 ± 0.62	95.92 ± 0.87	96.20 ± 0.67	94.86 ± 0.96	96.24 ± 0.82	96.55 ± 0.69
monks	99.63 ± 1.13	75.35 ± 2.62	92.60 ± 6.63	70.06 ± 5.24	99.47 ± 1.47	83.74 ± 11.82	100.00 ± 0.00	72.71 ± 6.93	71.67 ± 6.10
movement	97.79 ± 1.08	88.13 ± 3.55	96.38 ± 1.50	96.24 ± 1.68	92.78 ± 2.89	93.33 ± 2.74	95.78 ± 1.38	96.26 ± 1.38	97.18 ± 1.20
mushroom	99.99 ± 0.03	98.96 ± 1.03	99.85 ± 0.11	99.87 ± 0.09	99.96 ± 0.11	99.95 ± 0.16	100.00 ± 0.00	99.59 ± 0.17	99.89 ± 0.09
Newthyroid	99.80 ± 0.48	87.78 ± 9.28	99.56 ± 1.00	99.48 ± 1.21	97.10 ± 3.29	95.50 ± 6.13	97.93 ± 2.32	99.56 ± 1.00	99.70 ± 0.62
optdigits	99.78 ± 0.09	96.44 ± 0.75	99.55 ± 0.12	99.55 ± 0.13	99.71 ± 0.12	99.75 ± 0.14	99.78 ± 0.12	99.54 ± 0.12	99.52 ± 0.15
page-blocks	92.67 ± 1.50	78.44 ± 5.63	89.86 ± 2.76	91.13 ± 2.09	88.21 ± 2.90	84.49 ± 2.60	88.94 ± 2.18	88.49 ± 3.19	90.37 ± 2.28
pendigits	99.77 ± 0.09	95.50 ± 0.44	98.73 ± 0.19	98.67 ± 0.19	99.74 ± 0.14	99.45 ± 0.17	99.83 ± 0.08	98.70 ± 0.20	98.95 ± 0.17
primary-tumor	85.13 ± 2.79	79.56 ± 1.39	85.38 ± 2.86	85.34 ± 2.53	82.97 ± 3.06	83.27 ± 2.18	83.00 ± 2.71	85.05 ± 2.96	85.14 ± 3.24
qar	88.85 ± 4.20	81.09 ± 2.89	88.13 ± 5.08	86.58 ± 5.42	89.72 ± 3.02	86.43 ± 3.75	89.93 ± 3.55	88.06 ± 5.14	88.71 ± 4.29
robot-24	97.96 ± 0.37	89.93 ± 1.26	95.03 ± 0.83	95.35 ± 0.81	98.97 ± 0.44	98.77 ± 0.50	98.75 ± 0.38	94.80 ± 0.85	94.89 ± 0.84
segment	99.47 ± 0.27	94.89 ± 1.15	98.45 ± 0.49	98.16 ± 0.60	98.99 ± 0.33	98.84 ± 0.37	98.96 ± 0.29	98.37 ± 0.52	98.51 ± 0.51
sick	98.11 ± 0.75	89.74 ± 2.33	96.07 ± 2.54	94.94 ± 3.41	99.03 ± 0.59	97.09 ± 2.38	98.04 ± 1.22	95.92 ± 2.48	95.83 ± 2.38
sonar	85.81 ± 9.84	78.80 ± 8.53	86.40 ± 9.50	86.22 ± 10.10	79.58 ± 10.02	82.53 ± 8.12	87.92 ± 7.95	86.79 ± 9.83	85.00 ± 8.30
soybean	99.94 ± 0.03	99.05 ± 0.47	99.91 ± 0.05	99.91 ± 0.07	99.84 ± 0.17	99.73 ± 0.31	99.79 ± 0.13	99.90 ± 0.07	99.94 ± 0.04
spectrometer	87.38 ± 1.03	81.90 ± 0.84	87.52 ± 0.87	87.54 ± 0.86	87.11 ± 1.03	86.75 ± 1.17	86.46 ± 1.36	87.45 ± 0.86	87.71 ± 0.78
splice	99.50 ± 0.25	96.31 ± 1.17	99.51 ± 0.21	99.46 ± 0.24	98.70 ± 0.49	98.35 ± 0.76	97.91 ± 0.72	99.41 ± 0.22	99.44 ± 0.22
steel	98.85 ± 0.70	95.43 ± 1.96	99.41 ± 0.57	98.76 ± 0.81	100.00 ± 0.00	98.58 ± 0.78	95.11 ± 1.10	98.47 ± 0.98	99.28 ± 0.62
texture	99.29 ± 0.20	94.76 ± 0.67	97.02 ± 0.37	97.02 ± 0.38	99.78 ± 0.08	99.71 ± 0.13	99.81 ± 0.11	96.94 ± 0.38	97.12 ± 0.37
vehicle	85.45 ± 3.39	72.22 ± 3.58	80.99 ± 3.69	79.89 ± 3.87	89.34 ± 1.24	84.75 ± 2.78	88.65 ± 2.73	80.85 ± 3.73	81.29 ± 3.03
vote	97.84 ± 1.43	91.20 ± 3.61	97.58 ± 1.59	97.56 ± 1.73	97.53 ± 2.90	98.06 ± 2.95	98.10 ± 1.04	96.79 ± 1.95	96.87 ± 1.82
vowel	99.43 ± 0.67	93.60 ± 1.34	96.13 ± 0.70	96.23 ± 0.84	97.34 ± 0.98	94.45 ± 1.99	96.36 ± 0.74	96.19 ± 0.72	96.10 ± 0.85
waveform	96.56 ± 0.98	85.56 ± 4.22	95.61 ± 1.35	95.26 ± 1.39	91.22 ± 2.66	91.87 ± 2.01	89.64 ± 3.12	95.41 ± 1.36	95.34 ± 1.37
zoo	98.57 ± 1.66	96.43 ± 5.86	98.57 ± 1.66	99.05 ± 1.23	96.43 ± 3.22	95.83 ± 4.10	98.57 ± 1.66	98.57 ± 1.66	98.57 ± 1.66

•, ◦: Statistically significant upgradation and degradation, respectively.

Table 3.4: Two-tailed t -test on classification accuracy (ACC).

	LNB	TreeAWNB	MIAWNB	TreeAWKNN	MIAWKNN	KNN	NB	IWNB
TreeAWNB	2/35/19							
MIAWNB	1/28/27	2/44/10						
TreeAWKNN	11/31/14	19/28/9	22/26/8					
MIAWKNN	5/37/14	15/32/9	19/30/7	3/43/10				
KNN	9/29/18	16/29/11	19/27/10	4/38/14	10/34/12			
NB	0/36/20	2/49/5	9/42/5	9/28/19	7/35/14	13/29/14		
IWNB	0/40/16	5/49/2	15/38/3	10/30/16	9/37/10	14/30/12	8/48/0	
EWLNB	11/43/2	21/34/1	23/33/0	14/34/8	15/37/4	19/34/3	20/36/0	14/34/8

Table 3.5: Two-tailed t -test on area under the curve (AUC).

	LNB	TreeAWNB	MIAWNB	TreeAWKNN	MIAWKNN	KNN	NB	IWNB
TreeAWNB	43/13/0							
MIAWNB	43/12/1	6/44/6						
TreeAWKNN	38/16/2	13/32/11	13/32/11					
MIAWKNN	34/21/1	9/36/11	10/36/10	0/47/9				
KNN	36/19/1	12/35/9	12/33/11	2/45/9	8/45/3			
NB	40/16/0	0/37/19	4/40/12	10/32/14	7/38/11	9/33/14		
IWNB	43/13/0	5/44/7	7/41/8	10/34/12	8/39/9	12/31/13	13/42/1	
EWLNB	46/10/0	16/39/1	15/40/1	16/33/7	15/37/4	18/36/2	20/36/0	17/38/1

Table 3.6: Effectiveness of EWLNB

DATASET	Number of Class	ACC(EWLNB)	ACC(NB)	Improvement Rate	IR
vowel	11	90.71	67.07	35%	1
letter	26	80.2	66.15	21%	0.78
artificial	10	68.57	36.4	88%	0.424
autos	6	72.98	64.83	13%	0.048
energy1	37	63.15	45.05	40%	0.014
energy2	38	51.17	46.1	11%	0.013

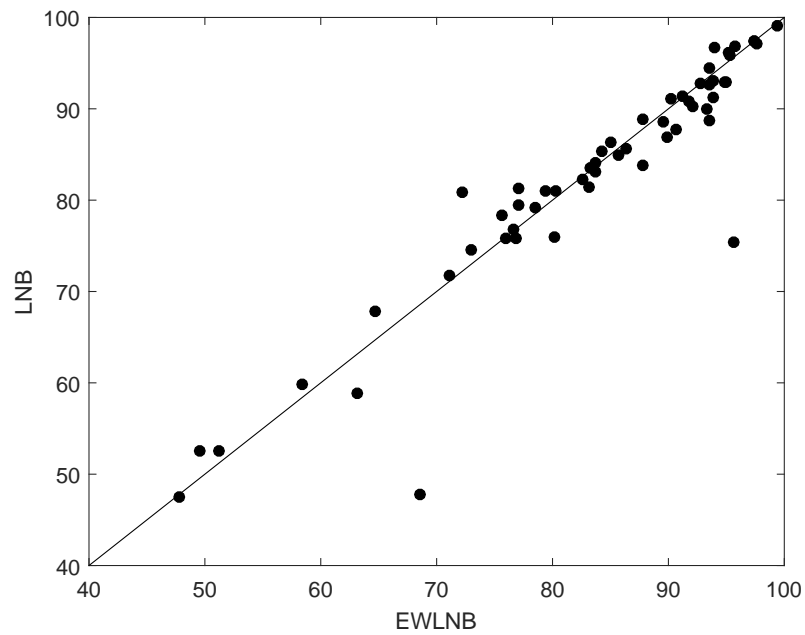
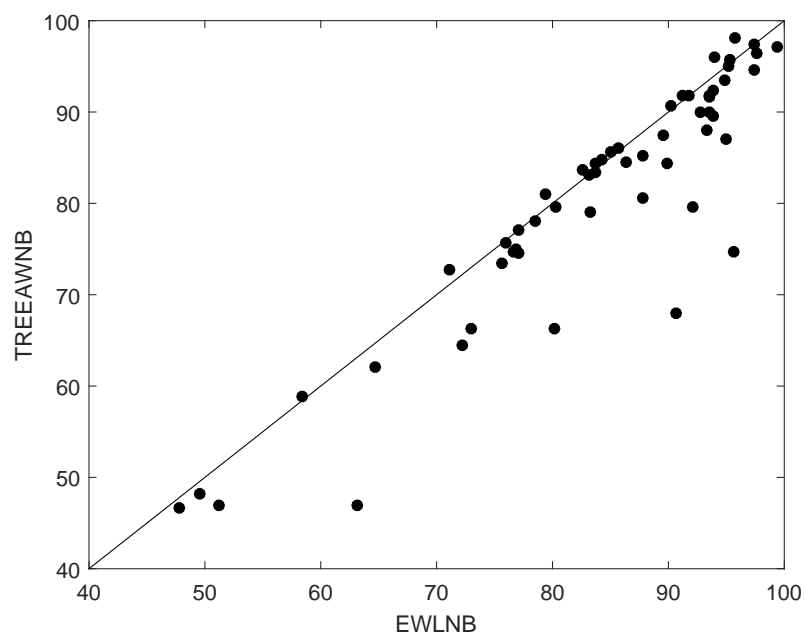
(a) *EWLNB vs. LNB*(b) *EWLNB vs. TREEAWNB*

Figure 3.1: EWLNB vs. competing algorithms: classification accuracy (ACC).

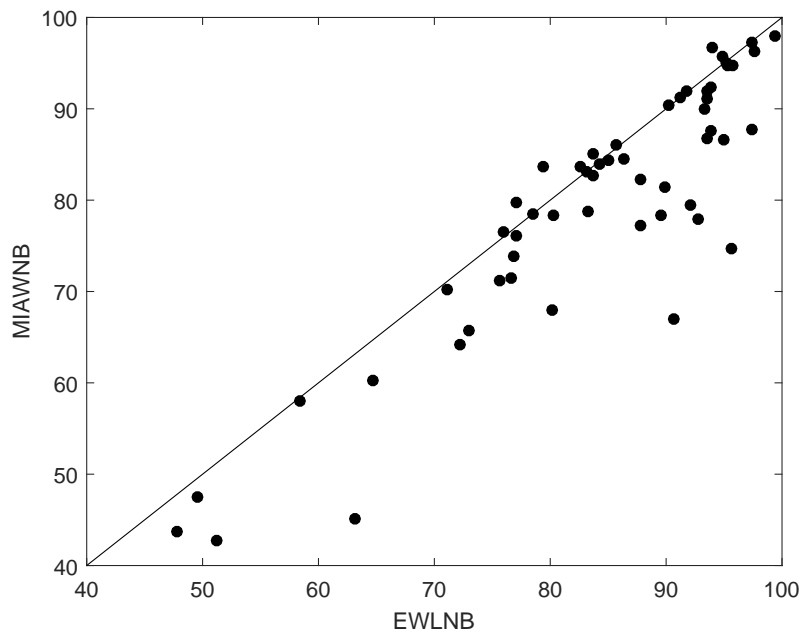
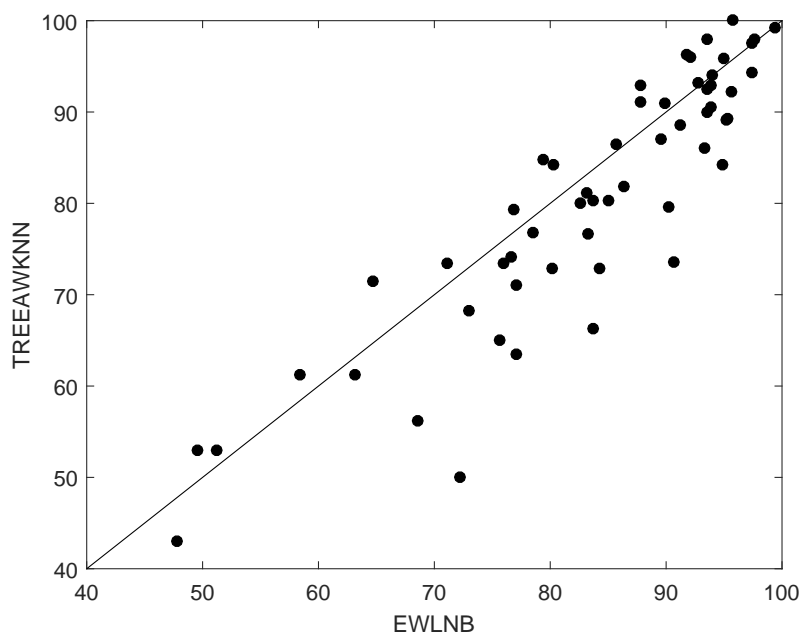
(a) *EWLNB vs. MIAWNB*(b) *EWLNB vs. TREEAWKNN*

Figure 3.2: EWLNB vs. competing algorithms: classification accuracy (ACC) — (continued).

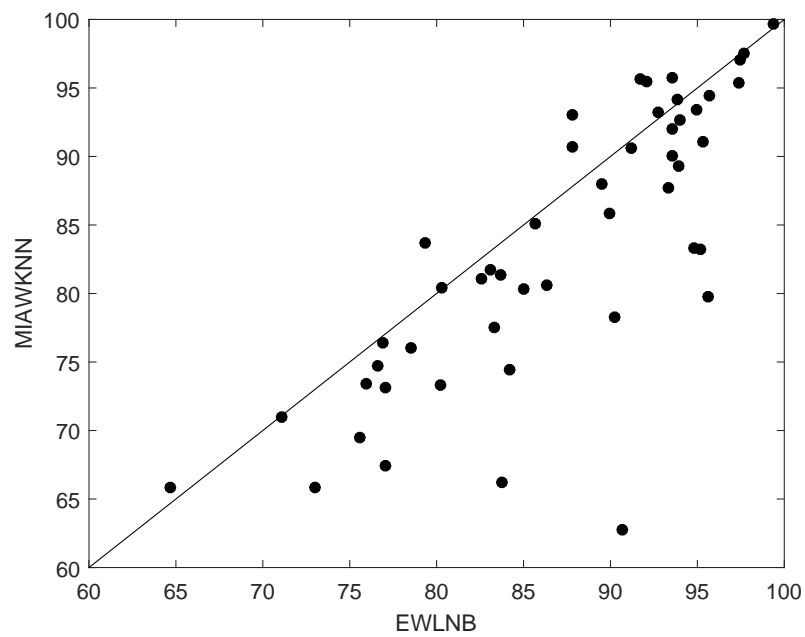
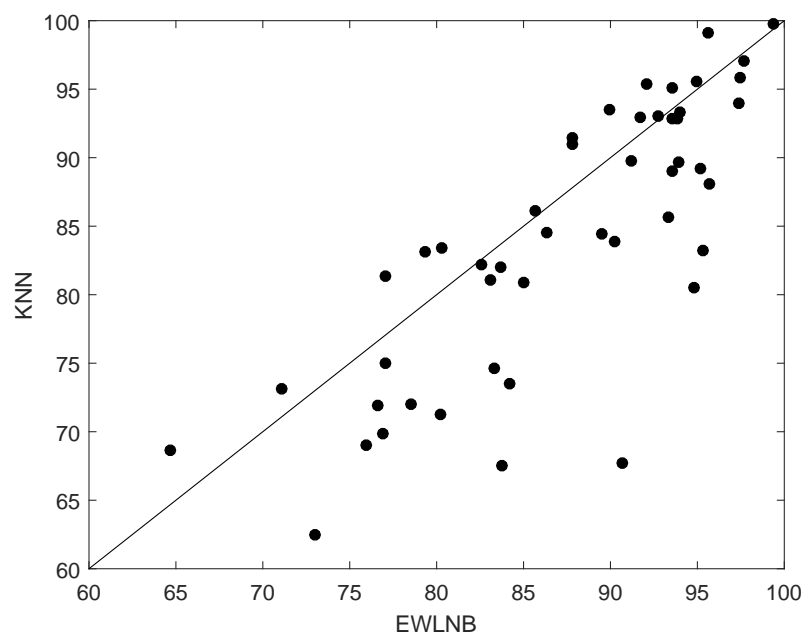
(a) *EWLNB vs. MIAWKNN*(b) *EWLNB vs. KNN*

Figure 3.3: EWLNB vs. competing algorithms: classification accuracy (ACC) — (continued).

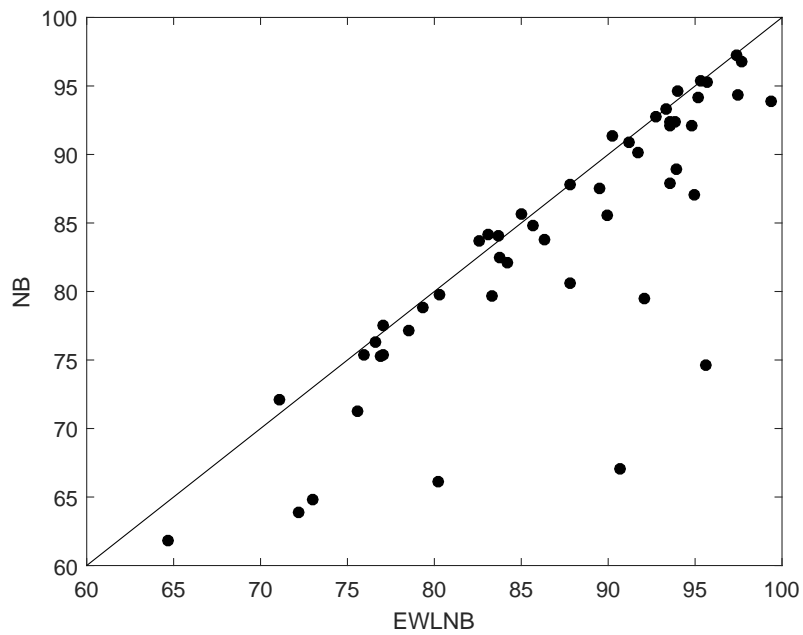
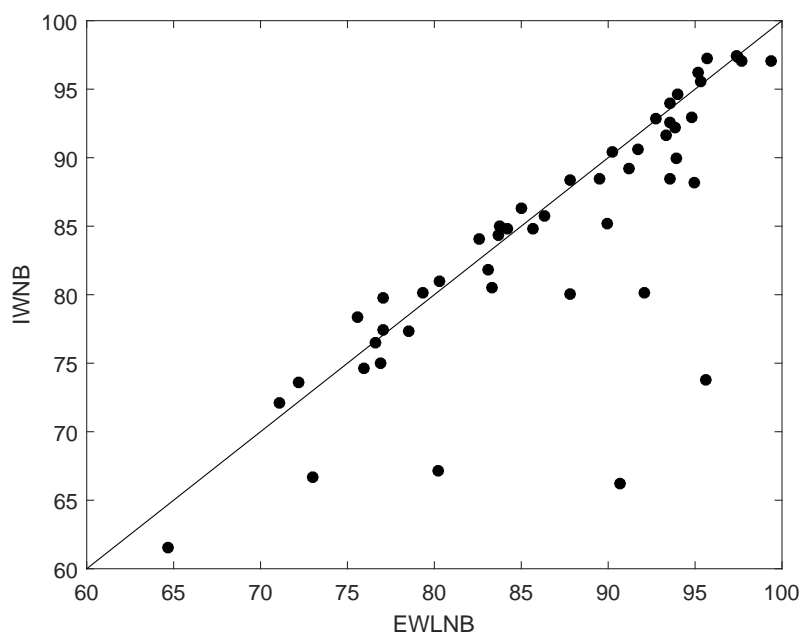
(a) *EWLNB vs. NB*(b) *EWLNB vs. IWNB*

Figure 3.4: EWLNB vs. competing algorithms: classification accuracy (ACC) — (continued).

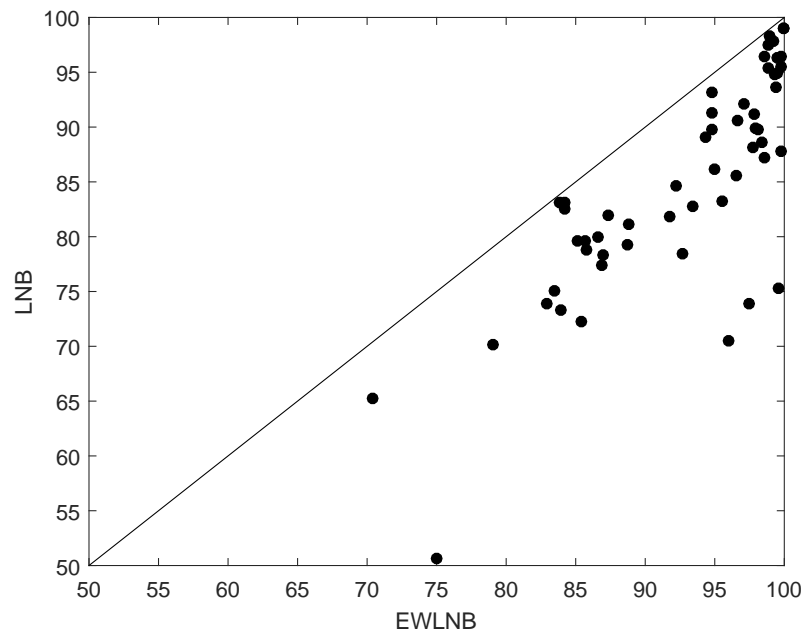
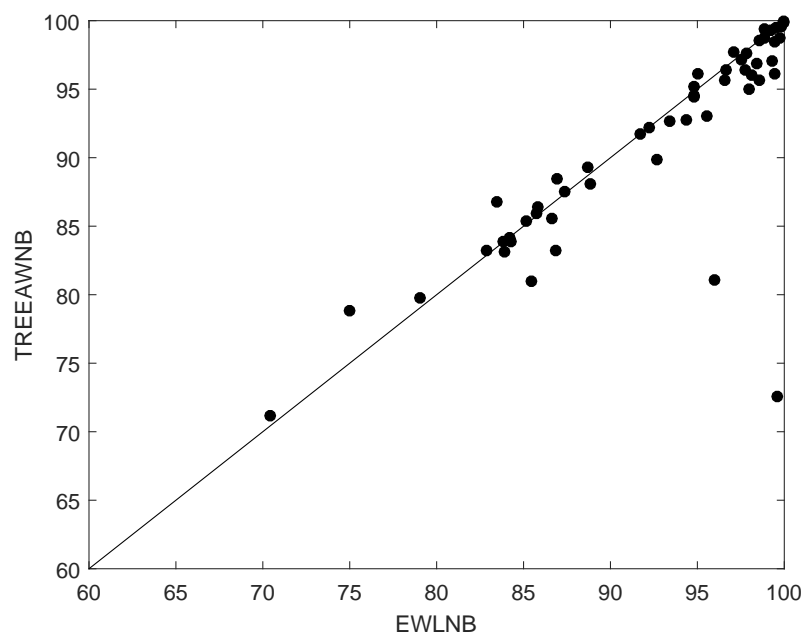
(a) *EWLNB vs. LNB*(b) *EWLNB vs. TREEAWNB*

Figure 3.5: EWLNB vs. competing algorithms: area under the ROC curve (AUC).

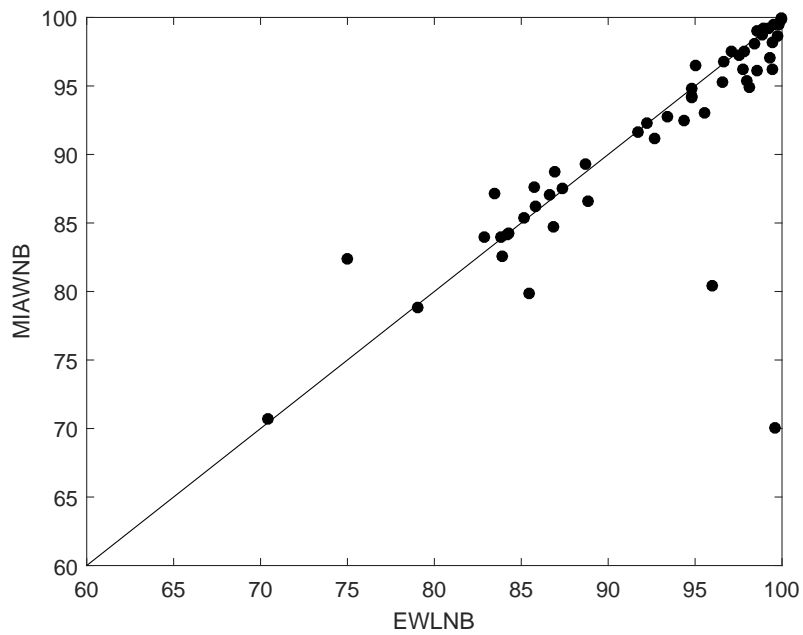
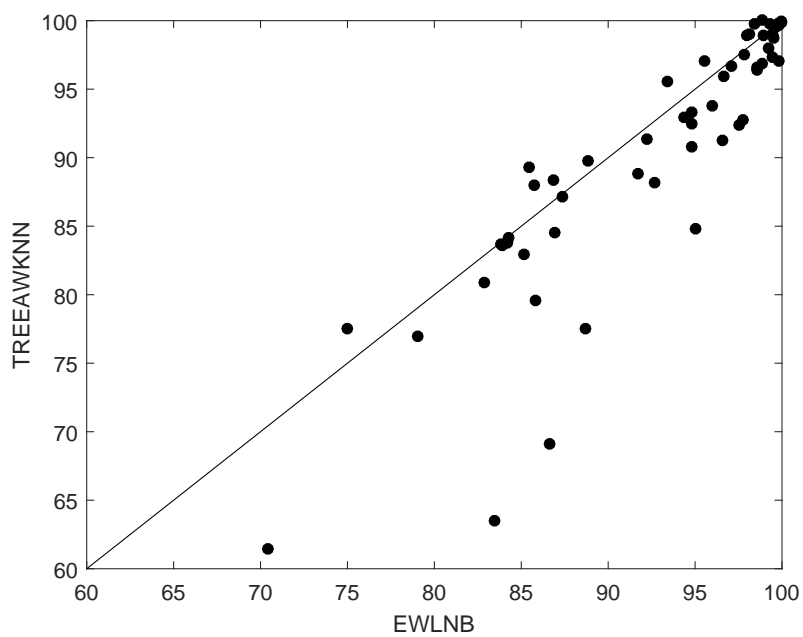
(a) *EWLNB* vs. *MIAWNB*(b) *EWLNB* vs. *TREEAWKNN*

Figure 3.6: EWLNB vs. competing algorithms: area under the ROC curve (AUC) — (continued).

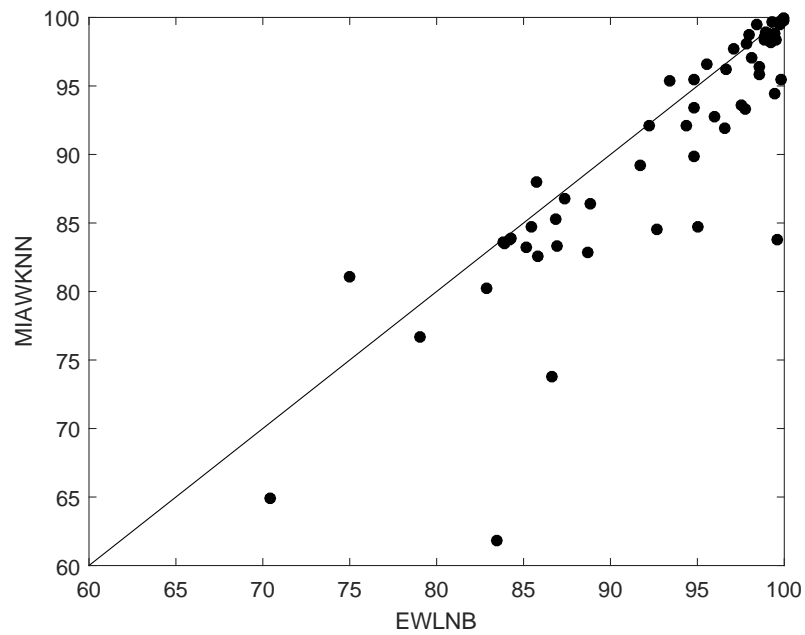
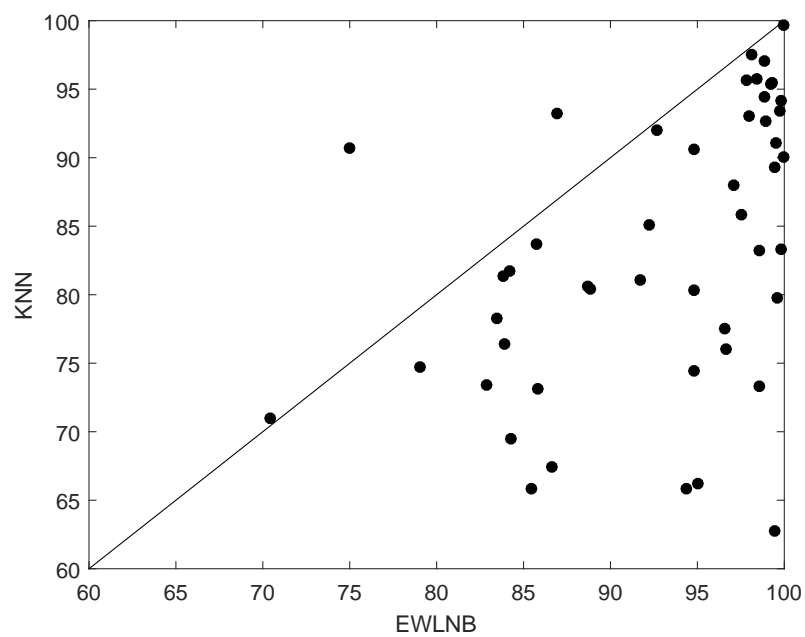
(a) *EWLNB* vs. *MIAWKNN*(b) *EWLNB* vs. *KNN*

Figure 3.7: EWLNB vs. competing algorithms: area under the ROC curve (AUC) — (continued).

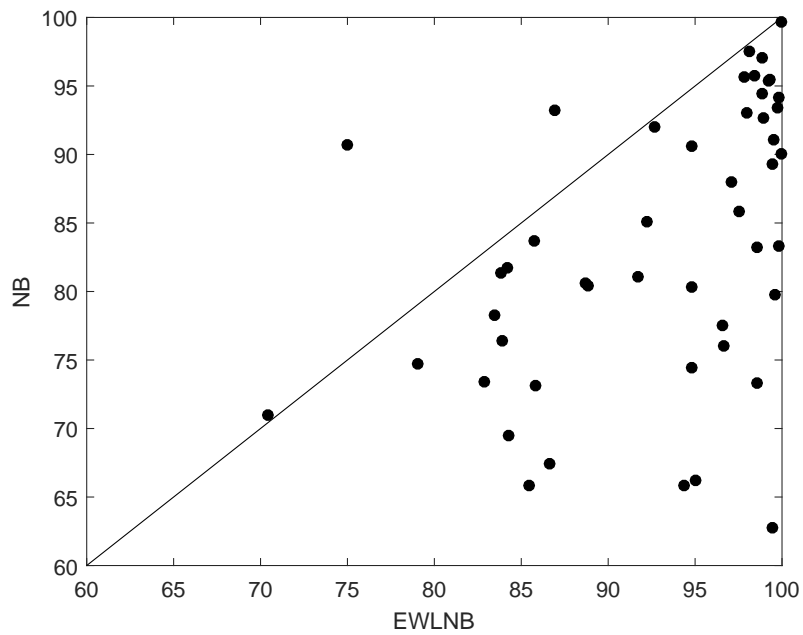
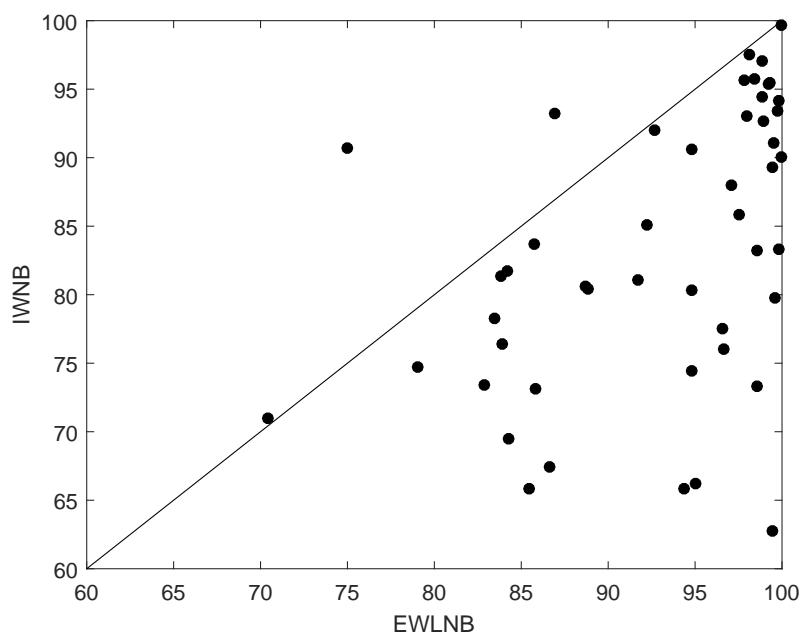
(a) *EWLNB* vs. *NB*(b) *EWLNB* vs. *IWNB*

Figure 3.8: EWLNB vs. competing algorithms: area under the ROC curve (AUC) — (continued).

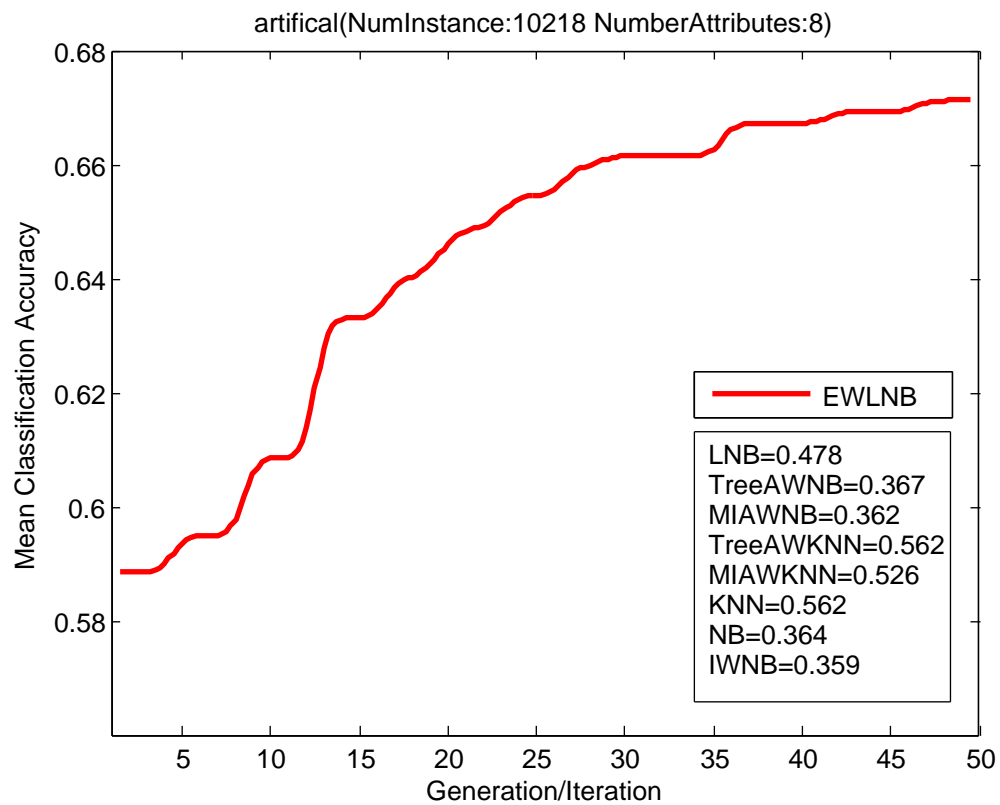


Figure 3.9: Convergence learning curves of EWLNB for 4 datasets with large numbers of instances (a) artificial: 10218 instances.

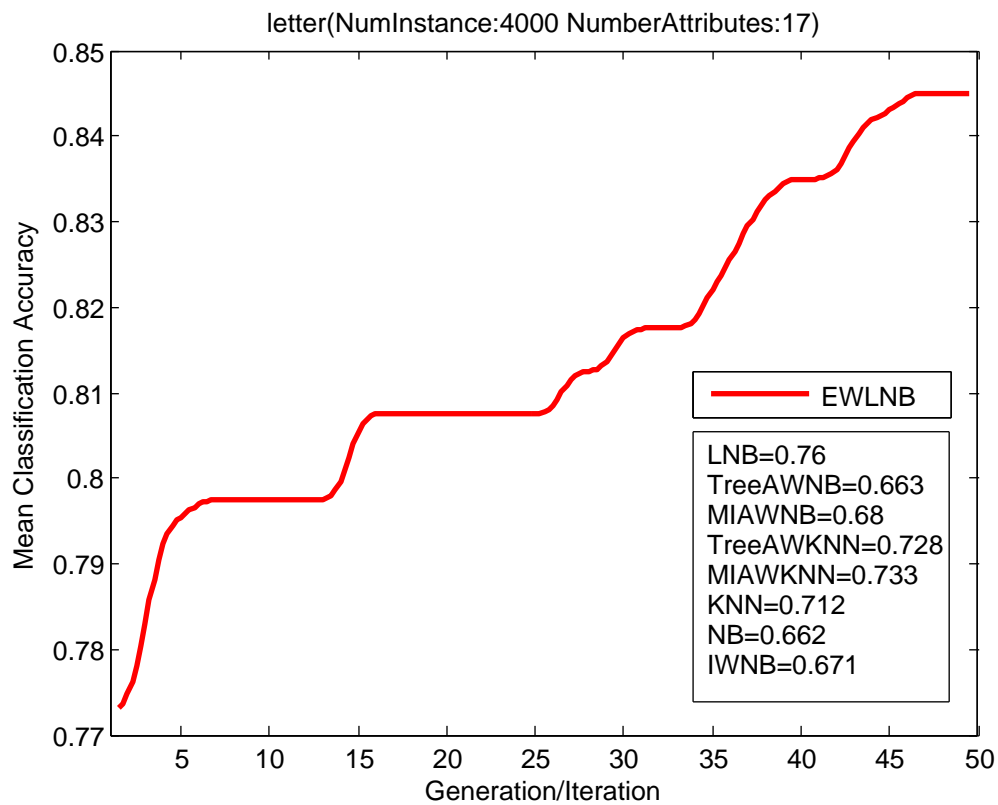


Figure 3.10: Convergence learning curves of EWLNB for 4 datasets with large numbers of instances (b) letter: 4000 instances.

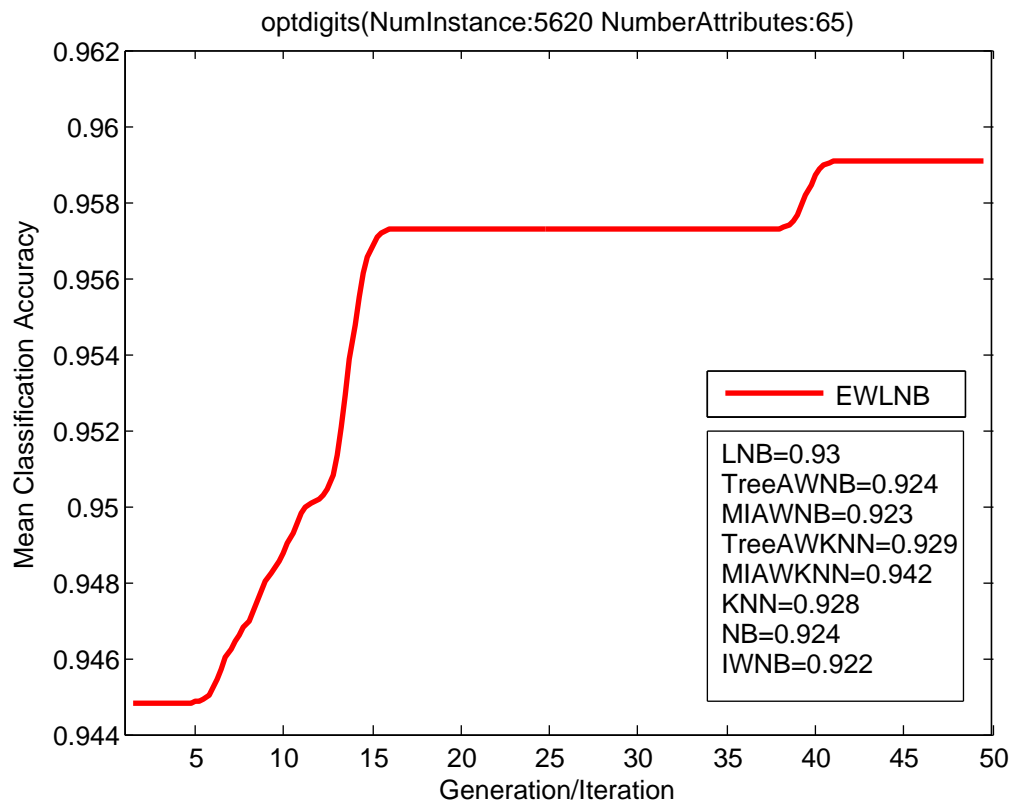


Figure 3.11: Convergence learning curves of EWLNB for 4 datasets with large numbers of instances (c) optdigits: 5620 instances.

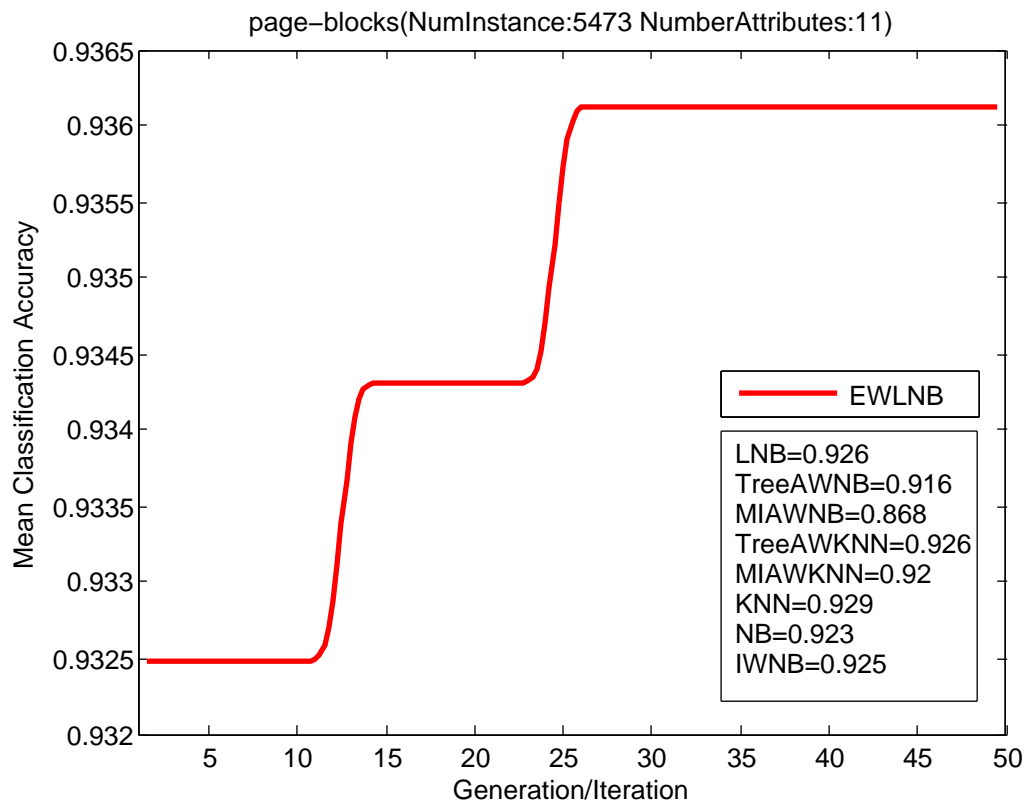


Figure 3.12: Convergence learning curves of EWLNB for 4 datasets with large numbers of instances (d) page-blocks: 5473 instances.

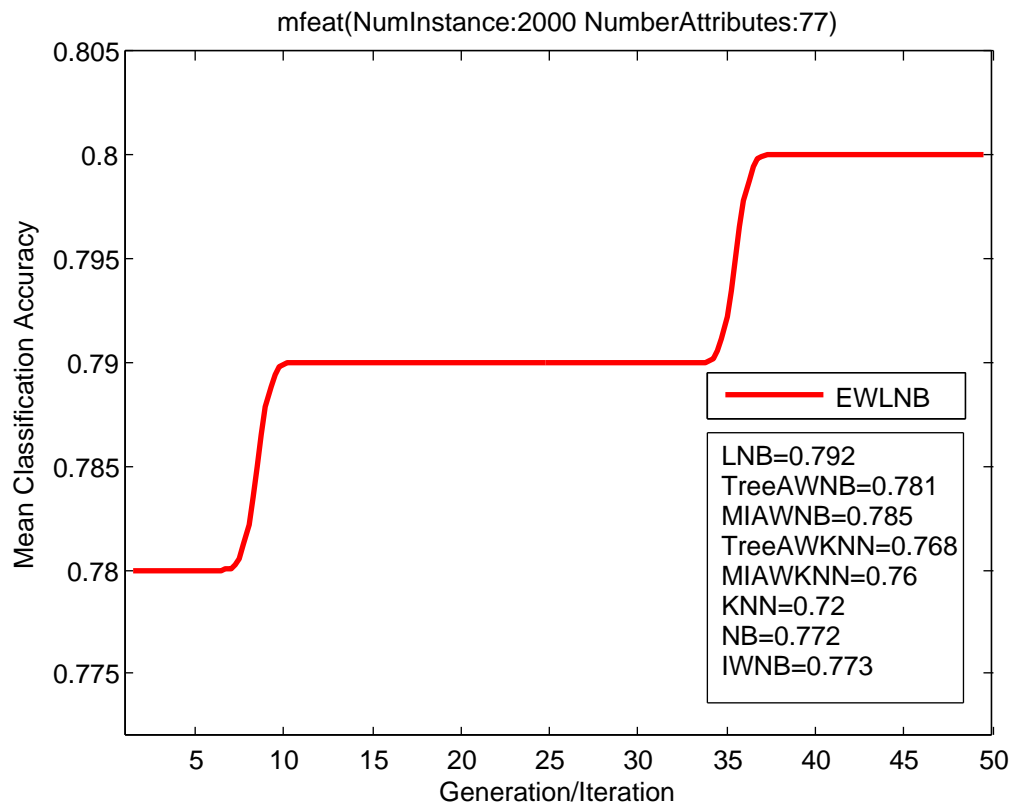


Figure 3.13: Convergence learning curves of EWLNB for 4 datasets with large number of attributes (a) mfeat: 77 attributes.

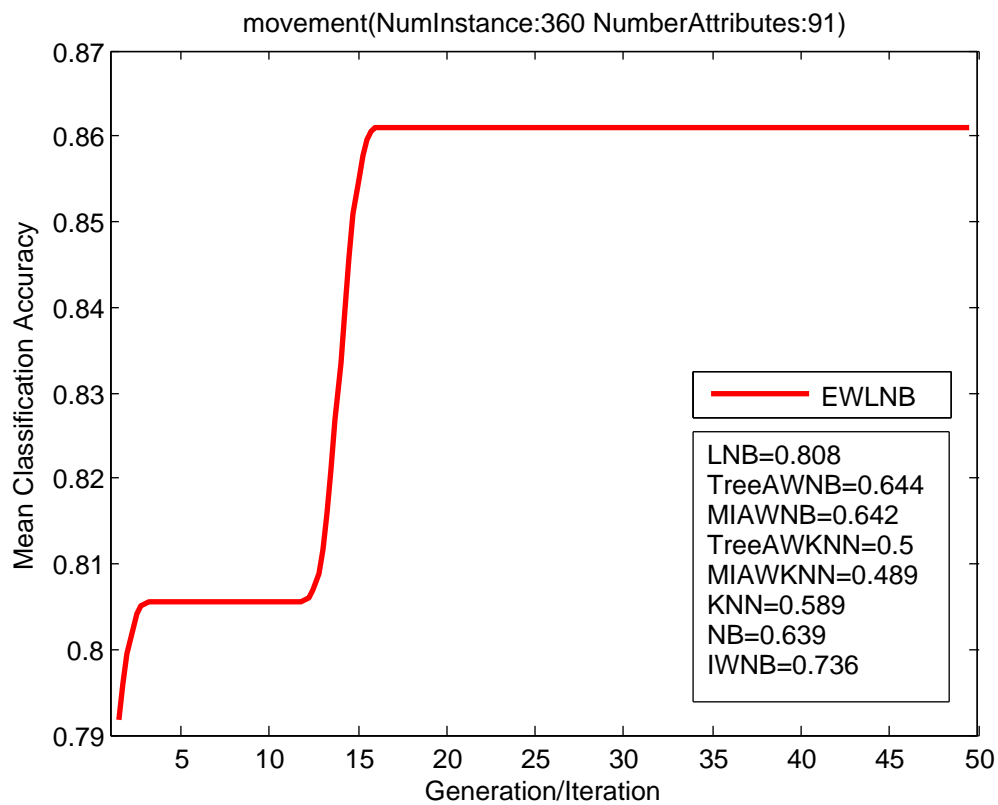


Figure 3.14: Convergence learning curves of EWLNB for 4 datasets with large number of attributes (b) movement: **91** attributes.

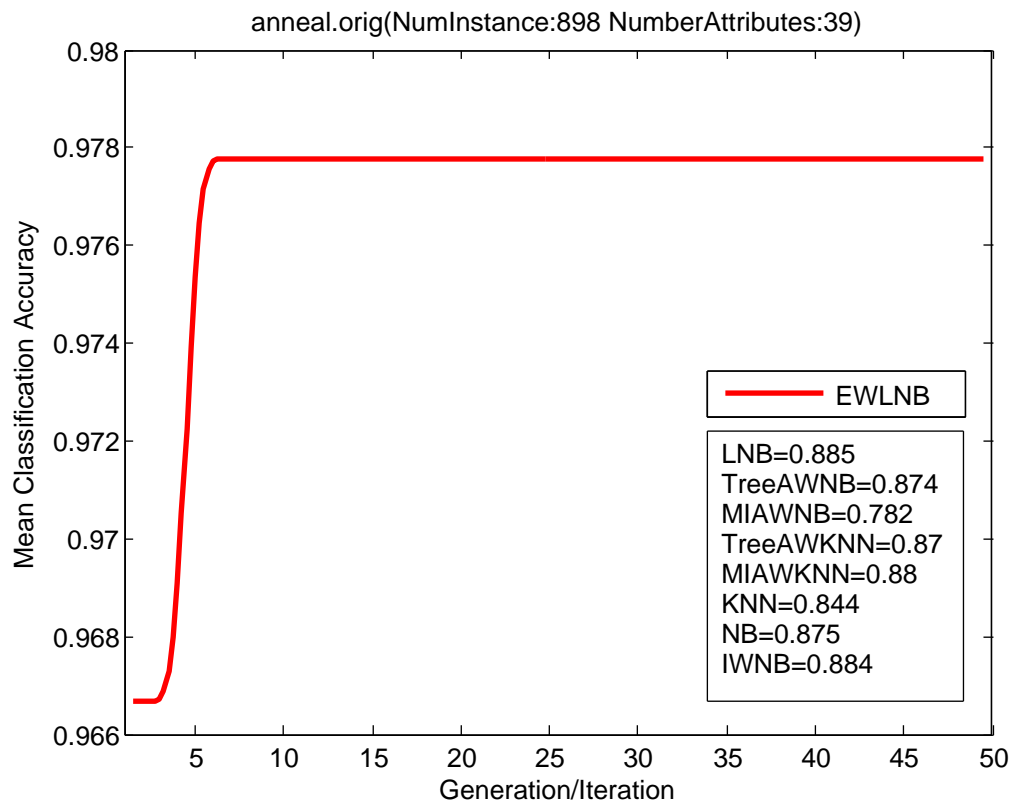


Figure 3.15: Convergence learning curves of EWLNB for 4 datasets with large number of attributes (c) anneal.orig: 39 attributes.

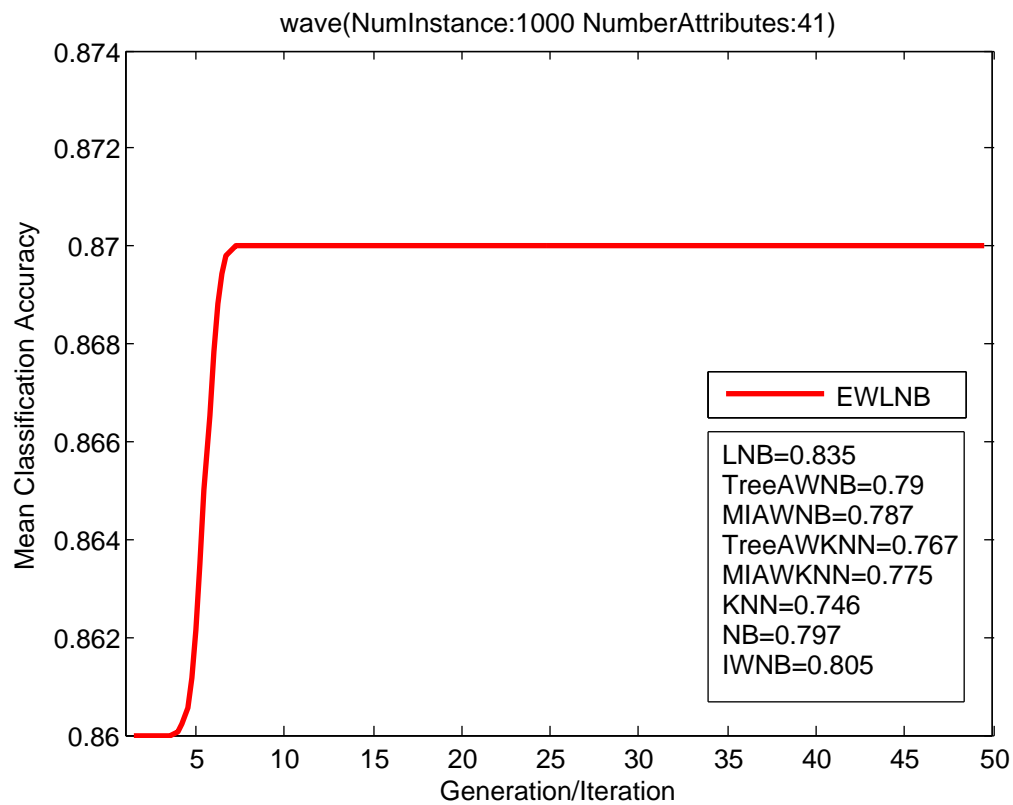


Figure 3.16: Convergence learning curves of EWLNB for 4 datasets with large number of attributes (d) waveform: 41 attributes.

3.5 Time Complexity

The time complexity of EWLNB can be mainly attributed to the following two processes: (1) WLNB, and (2) updating of the population.

Prior to the evaluation of the WLNB model, a KNN classifier needs to be trained from D^a with N_a instances, which will take $\mathcal{O}(N_a \cdot numattri)$, where $numattri$ is the number of attributes (this is because KNN needs to scan the whole training set, and all attributes, to find the K nearest neighbors).

To validate the WLNB classifier on a validation dataset D^b with N_b instances, the time complexity will be $\mathcal{O}(N_a \cdot numattri \cdot N_b)$.

The subsequent evolution of the population (e.g., the selection, clone, mutation, and update steps) are all based on the size of the population L , with the corresponding overall time complexity of $\mathcal{O}(L)$.

So, given the $MaxGen$, the total time complexity U is shown by Eq. (3.12).

$$U = \mathcal{O}(MaxGen \cdot L \cdot N_a \cdot numattri \cdot N_b) \quad (3.12)$$

Because $N_a < N$, and $N_b < N$, where N is the total number of dataset, Eq. (3.12) can be rewritten as

$$\begin{aligned} U &= \mathcal{O}(MaxGen \cdot L \cdot N_a \cdot numattri \cdot N_b) \\ &\leq \mathcal{O}(MaxGen \cdot L \cdot N \cdot numattri \cdot N_b) \\ &\leq \mathcal{O}(MaxGen \cdot L \cdot N \cdot numattri \cdot N) \\ &\leq \mathcal{O}(MaxGen \cdot L \cdot N^2 \cdot numattri) \end{aligned} \quad (3.13)$$

Eq. (3.13) shows that the total time complexity of EWLNB is bounded by four important factors: (1) the total number of instances N ; (2) the number of attributes

$numattr_i$; (3) the size of the population L ; and (4) the Maximum Evolution Generation $MaxGen$.

3.6 Effectiveness of EWLNB

When we examine the learning curves of EWLNB, we find that the improvements of the ACC through EWLNB are different for different datasets. To further investigate the effectiveness of EWLNB, we analyze the influence of the imbalance degree of a dataset, which is used to measure the degree of class imbalance for a dataset. Following Zong's work [91], we define imbalance ratio (IR) as the following equation:

$$IR = \frac{Min\#(t_i)}{Max\#(t_i)}, i = 1, \dots, m; \quad (3.14)$$

where m is the number of classes of the dataset, and $\#(t_i)$ indicates the number of instances in each class. From Eq. (3.14), we can easily see that the maximal IR value is 1, which indicates that all class labels are of the same size (*i.e.*, have the same distribution). Generally speaking, if the IR value is closer to 1, the data distribution is more balanced. Otherwise, the data distribution is imbalanced.

To validate the effectiveness of EWLNB on datasets with different class distributions, we analyze six datasets with different IR values: “vowel”, “letter”, “artificial”, “autos”, “energy1”, and “energy2”. From Table 3.6, we can see that the proposed EWLNB performs well on different datasets with different IR values, mainly because it uses a self-adaptive evolutionary process for parameter optimization to ensure its algorithmic performance.

Chapter 4

VSMS and EMLNB

In this chapter, we will build on the the work of the previous chapter and introduce the Virtual Storage Monitoring System (VSMS) problem and our proposed Evolutionary Multi-label Lazy Naive Bayes Classification (EMLNB) algorithm to address it.

4.1 Virtual Storage Monitoring System

Figure 4.1 shows an overview of our Monitoring System. The system first collects data from the disk array and the standard MIB (Management Information Base), then it splits the data into two groups. For one group, the data is used for visualization. For the other one, the data is used for prediction and alteration. VSMS is a centralized system, it uses SNMP to collect the data from different devices in the virtual storage network.

4.1.1 Main Technologies in VSMS

In this section, we begin by introducing three main technologies in our proposed VSMS, including Round-Robin Database Tool (RRDTools), Simple Network Management Pro-

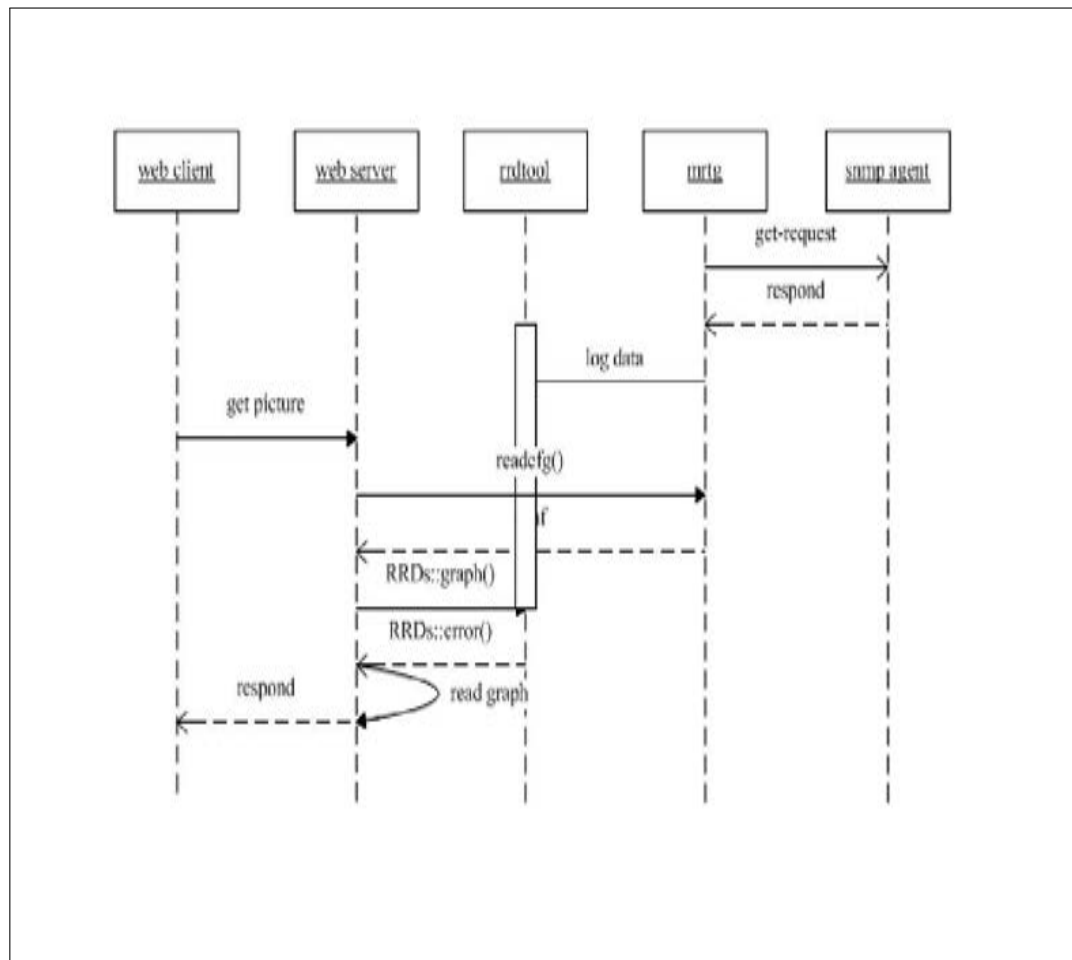


Figure 4.1: Conceptual overview of a Virtual Storage Monitoring System (VSMS).

tool (SNMP) and Multi Router Traffic Grapher (MRTG).

RRDTools

RRDtools (short for Round-Robin Database Tool) is designed to handle time series data stored from a system log, such as CPU load, temperatures, network bandwidth, etc. The data are stored in a circular buffer based database, thus the system storage capacity remains constant over time.

SNMP

VSMS uses Simple Network Management Protocol (SNMP for short) [29, 30] to collect data from different devices. SNMP is an Internet-standard protocol for collecting and organizing information of devices on networks. Devices that generally support SNMP include switches, routers, servers and so on. Because of its effectiveness and simplicity, SNMP is widely used in network management systems to monitor network-attached devices.

MRTG

MRTG uses SNMP to read the parameters of different devices, logs the data and creates graphs based on the collected data. The graphs are embedded into webpages which can be viewed from browser. Figure 4.6 shows a demo graph created by MRTG.

4.1.2 Data Collection

VSMS uses SNMP to collect data from different devices. SNMP has a typical manager/agent architecture, which is as shown in Figure 4.2. From Figure 4.2 we can see that SNMP contains three main parts: (1) SNMP manager; (2) SNMP agent; and (3) network. The SNMP manager centralizes and manages all the data collected from different devices. The SNMP agents run on the different devices to collect the data from the specific devices. The network can be regarded as a “bridge” that transmits the data from different SNMP agents to the SNMP manager. In VSMS, the SNMP manager service is running on the server where VSMS is located, while the SNMP agents service are running on different disk arrays or network devices. All these agents send the collected data to the manager by SNMP.

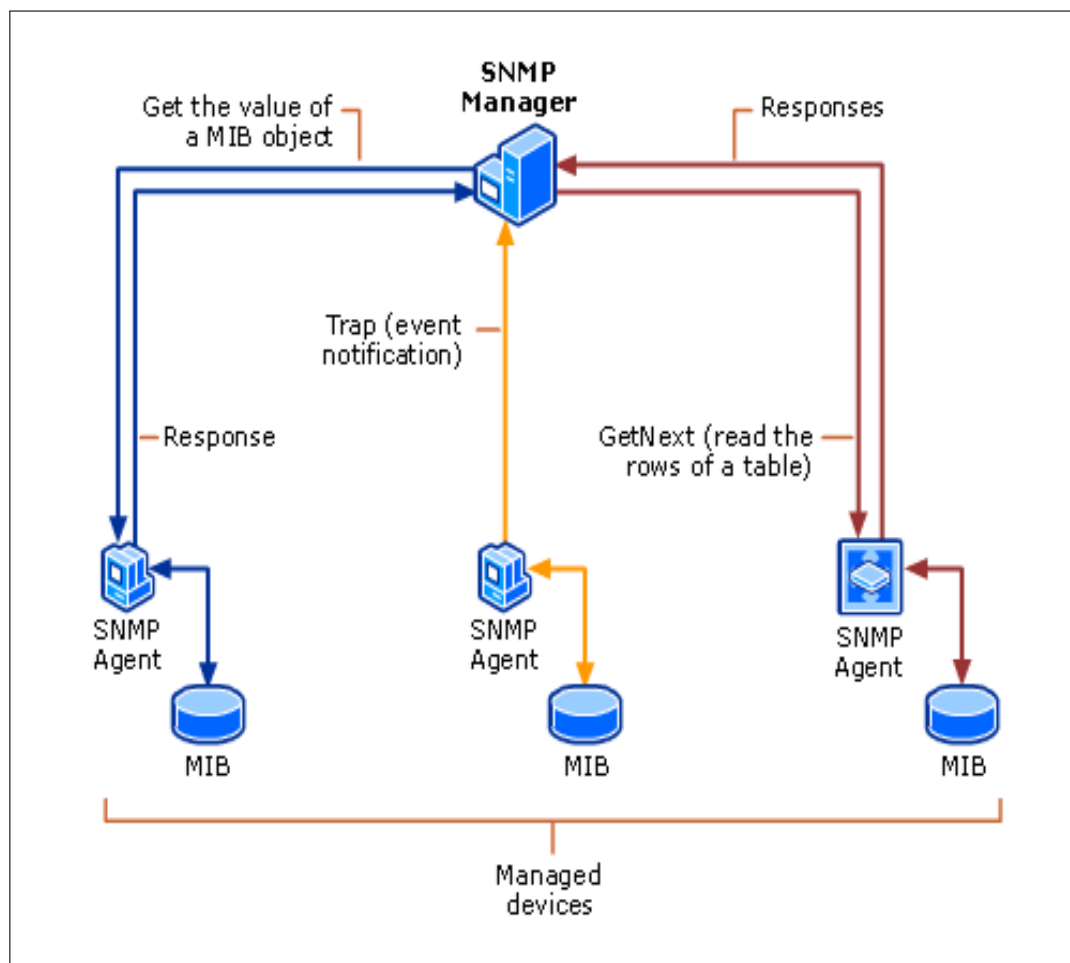


Figure 4.2: The structure of SNMP.

The data usually comes from two places: one is the disk array, the other one is the MIB, which is a standard library that contains the data regarding the net flow, cpu usage, memory usage, and so on. The basic structure of MIB can be found in Figure 4.3. MIB uses object identifiers (OID) to locate the device and get the specific data of the device. OID is based on a hierarchical name structure named “OID tree”. The OID tree uses a sequence of names, of which the first name identifies a top-level “node” in the tree, and the following ones provide further identification of edges leading to the sub-nodes

beneath the top-level, and so on, to any depth.

Each leaf of the OID tree represents a specific and unique object which can be referred to later for further use. One obvious advantage of OID is that we can extend the OID tree and add our own objects (generally under the node 19040, or any node which is called private node).

In VSMS, the temperature of a disk is not a standard object of MIB, so we need to extend the MIB and add the object of disk temperature for monitoring. Figure 4.4 shows the extended MIB after we add the temperature of disk (disktemp) and flight height of the magnetic head (fly-height). We can see that the extension is performed by way of 19040.diskmib.diskobj.disktemp and 19040.diskmib.diskobj.flight-height accordingly in Figure 4.4.

The collected data is split into two groups. For data that indicates the general performance of the system, such as net-flow, memory usage, and disk temperature, VSMS uses MRTG+RRDtools to visualize the data. This helps to monitor the system in a more clear and straightforward way. For data that indicates some fatal failure of the disk array, such as Raw_Read_Error_Rate and Seek_Error_Rate, the system will raise an alert immediately.

Moreover, for the parameters that do raise alerts, we can give different priorities to the alerts, according to the severity of the problem. This means that we can perform different strategies to handle the problems with different priorities. In our work, two different types of alerts are defined. One is called Pre-Fail, with a higher severity. One is called Old-Age, with a lower severity. This is shown in Figure 4.5.

4.1.3 Data Visualization

VSMS uses SNMP+MRTG+RRDtools to visualize the collected data. The drawback of MRTG is that it can only draw two pictures in one page, and the monitoring data is stored in a type of log data format which consumes lots of space resource. To solve this problem, we combine MRTG with RRDtools, which can perform the task of data visualization in a better way. The data of RRDtools refreshes every five minutes, which means the graphs refresh every five minutes. This time interval can be set through a configuration file and thus can be adjusted according to different requirements.

4.1.4 User Interface

Main Interface

Figure 4.6 displays the main user interface of our VSMS system. The left panel of the user interface displays which devices are monitored at the moment. There are two drop-down menus on the top left of the interface. The first one is used for choosing which monitoring item to display (net flow, disk temperature, cpu usage, or disk usage, for example). The second one is used for specifying the time interval. From these interfaces, VSMS can display different visualization patterns of the data by hour, by day, by week, and so on.

Detailed Interface

The VSS can display the detailed information of a monitoring item through a click on the figures on the main interface. Figure 4.7 shows this kind of detailed information, in

this case for network-flow by day.

The VSS Monitoring System mainly consists of two different modules — the monitoring module and the multi-label classification module. The monitoring module was explained in the above section. In the following section we will introduce the multi-label classification algorithm.

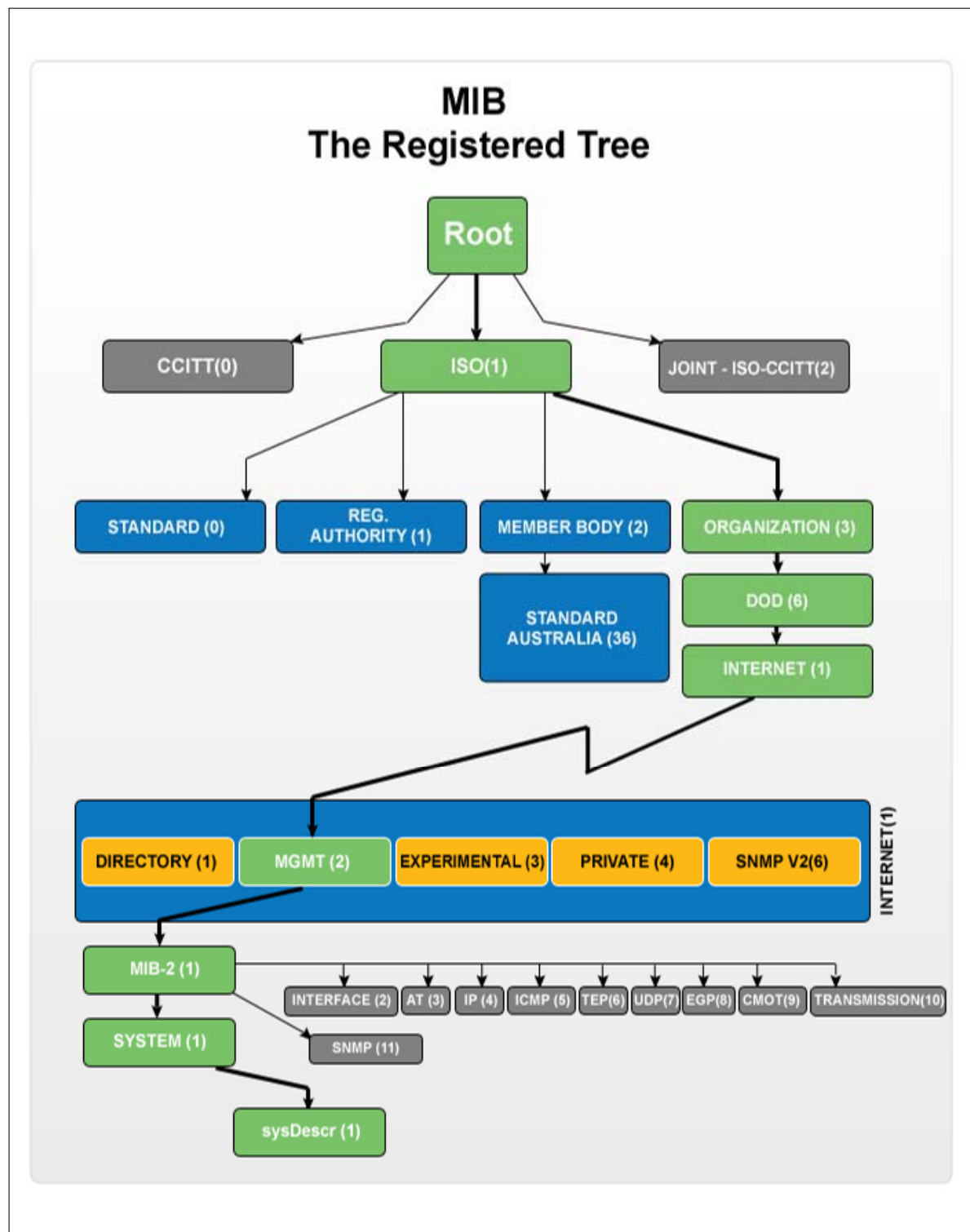


Figure 4.3: The structure of MIB.

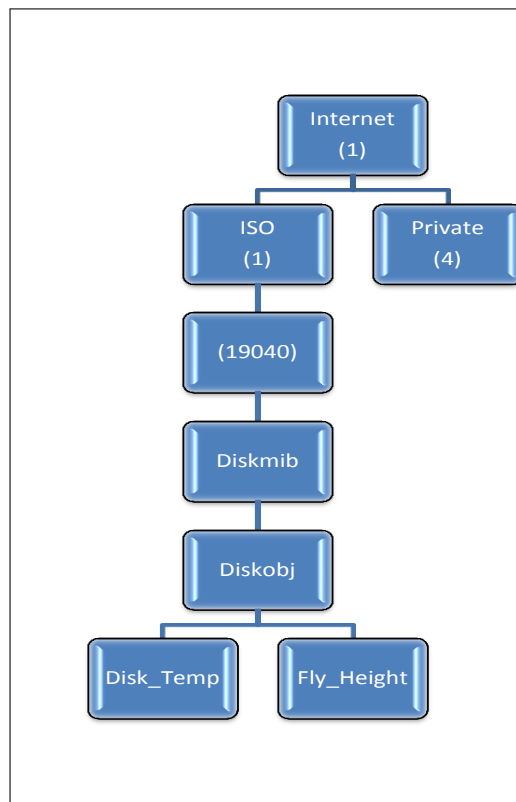


Figure 4.4: The extended structure of MIB.

The first Error Type: Pre-Fail, which means there may exist fatal errors in the disk.

```
30001 => {  
    Name => "RawReadErrorRateChanged",  
    Description => "S.M.A.R.T information RawReadErrorRate has changed",  
    Cause => "Read data error",  
    Action => "Backup, Change Disk immediately"  
}
```

The Second Error Type: Old-Age, which means there may exist potential problem in the disk.

```
30002 => {  
    Name => "PoweronHours",  
    Description => "S.M.A.R.T information indicate the overall Power on Hours of the disk",  
    Cause => "Old Age",  
    Action => "Backup, Change Disk if possible"  
}
```

Figure 4.5: Error Types of different priority.

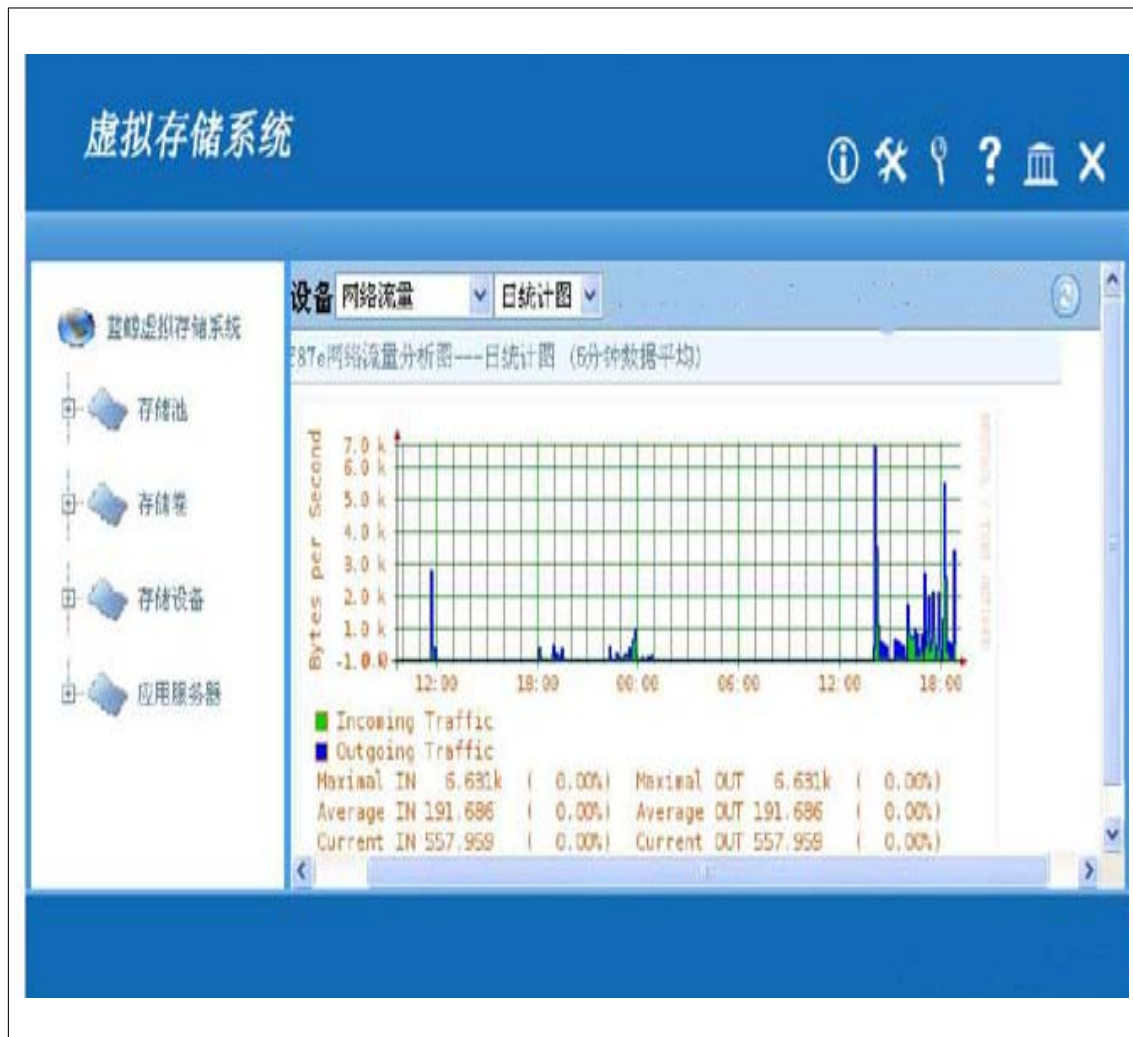


Figure 4.6: The main user interface of VSMS.

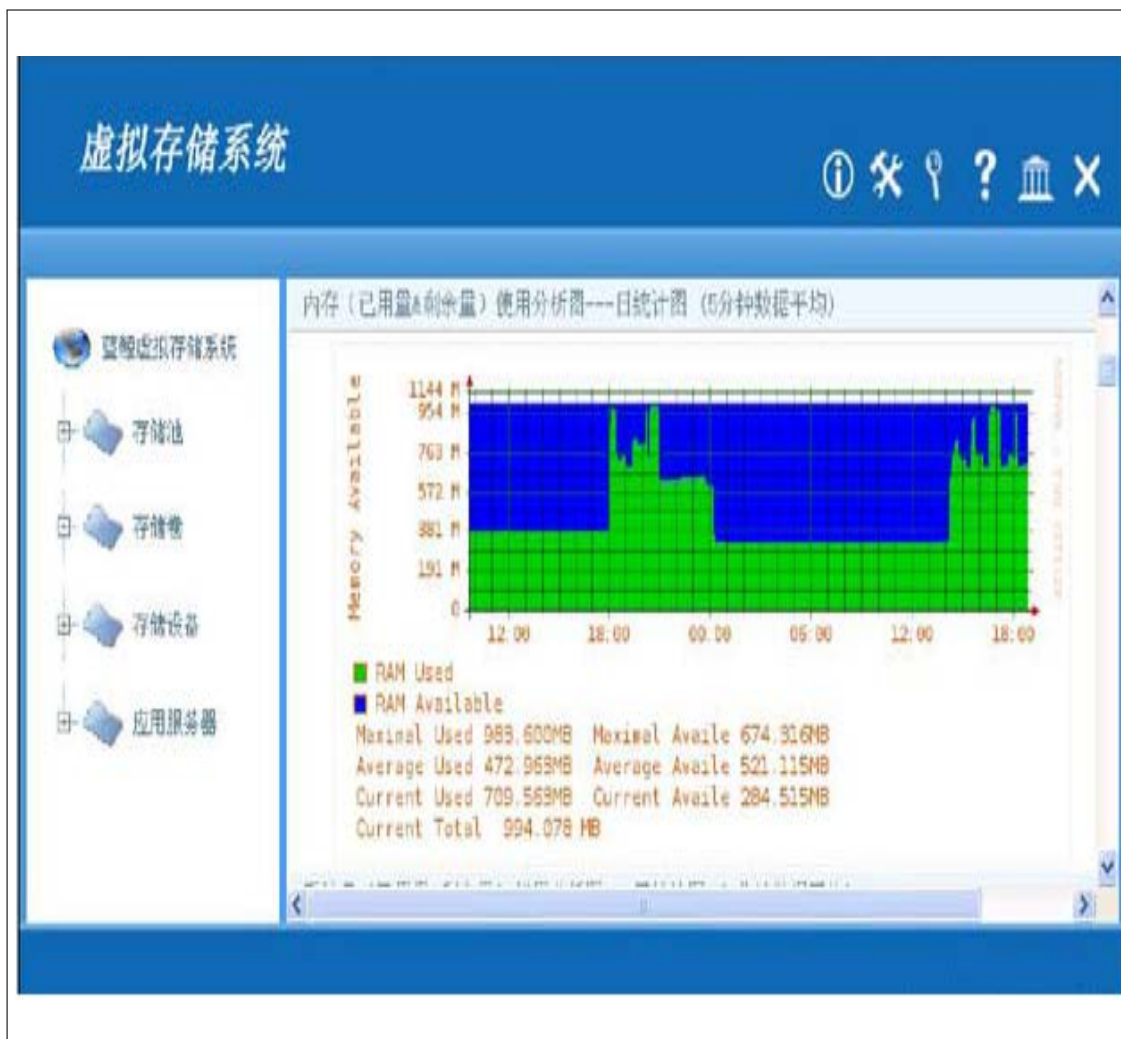


Figure 4.7: Detailed information on a monitored item in VSMS.

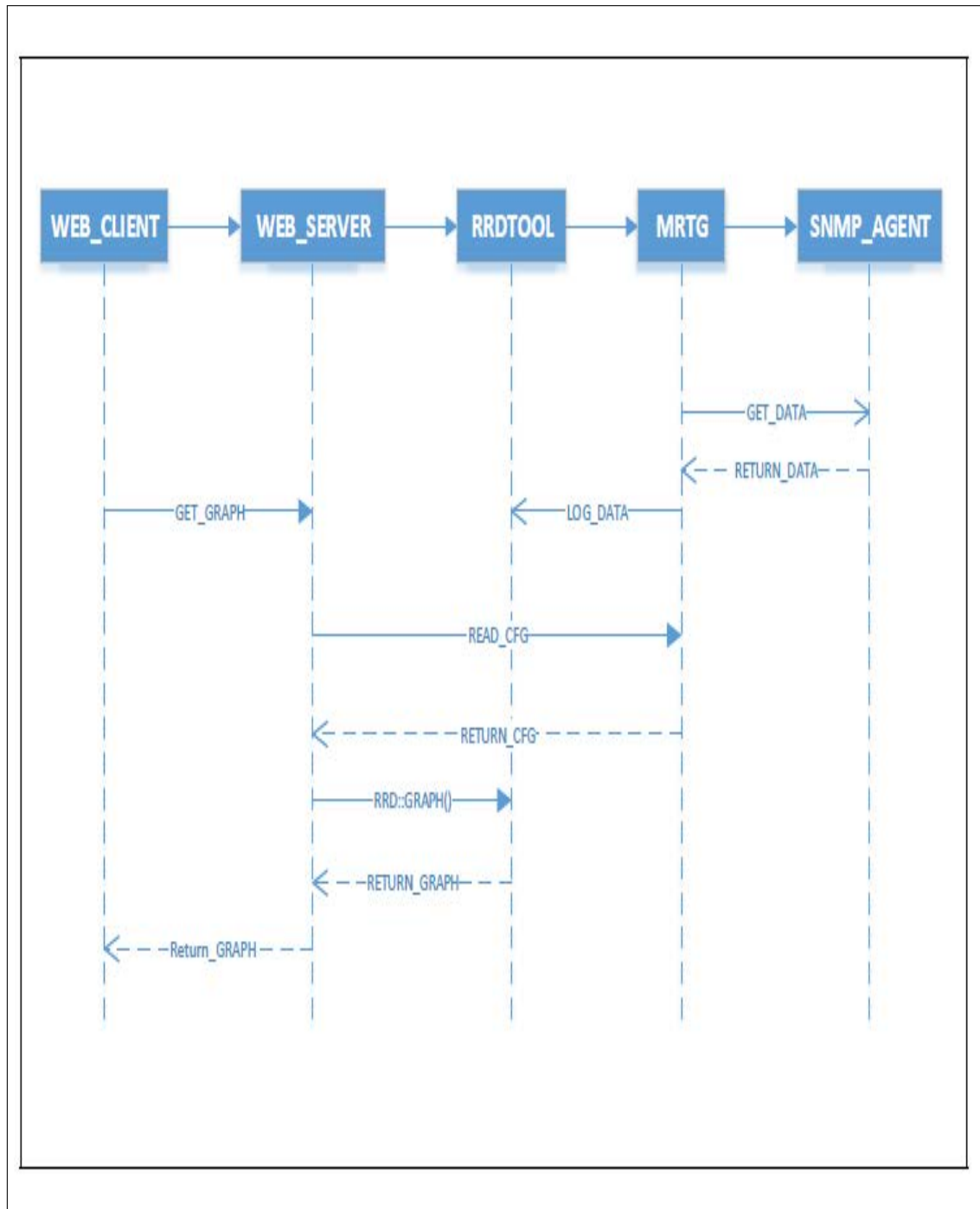


Figure 4.8: The system architecture of VSMS.

4.2 Evolutionary Multi-label Classification

Since a multi-label classification task also has many parameters that need optimization, evolutionary algorithms can be used to address this problem. The proposed Evolutionary Multi-label Lazy Naive Bayes (EMLNB) classifier algorithm has two main steps, as follows. Firstly, a Multi-label Lazy Naive Bayes (MLNB) classification algorithm locally learns the class labels of a test instance. Secondly, a Differential Evolution-based self-adaptive process is used to find two key parameters in MLNB.

4.2.1 Data Captured through VSMS

A VSMS will be expected to capture data on any features of components relevant to system failure, such as disk Read Error Rate, Power-On Hours, CPUs, GPUs, Network Devices, and so on. The classes of failure for each system component may include errors such as Pre-fail and Old-Age. Typically there is an severity order over failure classes, indicating the priority of actions to be taken to correct the failure.

4.2.2 The Framework of MLNB

The main difference between EWLNB and EMLNB is that EMLNB extends EWLNB to deal with the multi-label classification task. Instead of just predicting a single value to label a test instance in EWLNB, we require EMLNB to predict multiple values to label a test instance.

This introduces another important class of hyperparameters for EMLNB, namely the threshold used as the cutoff for the probabilities of the predicted values.

In MLNB, there are many parameters that are very important. We choose two of them: the number of nearest neighbors K of KNN ; and the threshold of the probability

used by the Multi-label classification. The commonly used distance metric for choosing nearest neighbors is Euclidean distance. MLNB is shown in Algorithm 3:

Algorithm 3: MLNB: Multi-label Lazy Naive Bayes Classification

Input:

Training dataset D^a ; a testing instance x^b ;

parameter vector $w = \{w_1, w_2\}$;

Output:

Classification results (Multi-label) $SetsofLabels(x^b)$;

- 1: $D_*^a \leftarrow D^a$ to prevent operation on the original dataset;
 - 2: $K \leftarrow w_1, threshold \leftarrow w_2$;
 - 3: **for all** each x_i^a in D^a **do**
 - 4: Compute $d(x^b, x_i^a)$ using Euclidean distance;
 - 5: **end for**
 - 6: Find the K Nearest Neighbors of x_i^a as KN ;
 - 7: Build an Naive Bayes classifier over the KN ;
 - 8: Use this NB classifier to give the probability estimation of all the labels of x^b ;
 - 9: Use all the labels with probability estimation $\geq threshold$ as the predicted labels of x^b ;
-

4.2.3 The Framework of EMLNB

To optimize the hyperparameters for MLNB, we propose the use of evolutionary search of the space, leading to the Evolutionary MLNB. The framework of EMLNB is shown in Algorithm 4: We use Hamming Loss, i.e., the fraction of incorrectly predicted labels to the total number of labels, as the fitness function $f[]$ for Algorithm 4.

Algorithm 4: EMLNB: Evolutionary Multi-label Lazy Naive Bayes Classification

Input:

Evolution Population W , Size of Population L , Evolution Generation $MaxGen$;
Training Dataset D^a , Validation Dataset D^b , Testing Dataset D^c ;

Output:

Classification Results $SetsofLabels(x^c)$ of Testing Instance $x_t^c \in D^c$;

- 1: $W \leftarrow$ The initial $w_{i,j}$ value for each individual w_i is set to a real number;
 - 2: **while** $t \leq MaxGen$ **do**
 - 3: **for** $i=1$ to L **do**
 - 4: $f[w_i^t] \leftarrow$ Use D^a as the training dataset and D^b as the validation dataset to calculate the fitness of w_i^t in the population W^t ;
 - 5: $w_c^t \leftarrow$ Choose the best individual with highest fitness value;
 - 6: $v_i^t \leftarrow$ Apply mutation operation to obtain the variation individual;
 - 7: $u_i^t \leftarrow$ Apply crossover operation to obtain the trial individual ;
 - 8: $W^{t+1} \leftarrow$ Apply a greedy search strategy to obtain the next generation. If $f[u_i^t] \geq f[w_i^t]$, individual u_i^t survive to the next generation, otherwise individual w_i^t can be retained.
 - 9: **end for**
 - 10: $t = t + 1$;
 - 11: **end while**
 - 12: $SetsofLabels(x^c) \leftarrow$ Apply w_c^t to MLNB to predict the labels of testing instances in D^c .
 - 13: **return** The labels of testing instances in D^c ;
-

The development and definition of Algorithm 4 is similar to the way in which we developed the EWLNB algorithm in Chapter 3.

Due to the inability to access relevant data captured from a VSMS, we have not been able to perform any experiments with a real-world monitoring dataset using our EMLNB approach. This will be one part of our future work.

Chapter 5

Semi Evolutionary Algorithm to Optimize Deep Network Hyperparameter (SEODP)

In this chapter, we will introduce details of the SEODP algorithm.

As we have mentioned above, studies of DN architectures over different datasets have shown complicated results. Architectures that show good performance over some datasets may not have good performance over other datasets with different characteristics. To address this problem, we propose the use of the SEODP algorithm to automate hyperparameter search for deep networks.

5.1 Definitions

The main motivation of SEODP is that normal Differential Evolution (DE) can converge very slowly, and might fall into local optima. However, it can be a useful algorithm for

optimization problems. To address this issue, we propose to add a random selection process as part of the DE process.

In the proposed SEODP learning framework, there are certain number of parameters to be optimized. We use a semi-evolution process to auto-learn their optimal values. In our solution, each combination of hyperparameters is simulated as an individual in the population for the evolutionary process to work on. Before introducing the details of the self-adaptive parameter optimization process, we define a number of notational elements, as follows:

- *SizeOfParameters*: *SizeOfParameters* indicates the number of parameters to be optimized.
- *NumberOfClassLabels*: *NumberOfClassLabels* indicates the number of labels that the class variable has.
- *Population*: $W = \{w_1, \dots, w_L\}$ represents the population (*i.e.*, the set of individuals), where L denotes the size of the population. $w_i = \{w_{i,1}, w_{i,2}\}$ represents a single individual in the population. Each individual represents a unique combination of hyper parameters. The default value of L is 50.
- *Maximum Evolution Generation*: *MaxGen* represents the maximum number of generations of the evolutionary process. This represents an assumption that after a certain number of generations of the evolutionary process, the algorithm should be able to find the optimal combination of parameters, or a good approximation to it. The default value of *MaxGen* is 10.
- *Gene*: $w_{i,j}$ indicates the j th gene value of the i th individual.

- *Mutation Rate*: F indicates the mutation rate for the mutation operation, which defines the proportion of individuals that are mutated in each generation. The default value of F is 0.5.
- *Crossover Rate*: CR indicates the crossover rate for the crossover operation. Crossover rate is the probability of an individual being part of a crossover. The default value of CR is 0.5.
- *Maximum Crossover Population*: $MaxCP$ represents the maximum number of individuals in the Crossover Population. The default value of $MaxGen$ is 50.
- *Best HyperParameter*: $BestHP$ represents the top $BestHP$ combination of hyperparameters we choose for the specific model.

5.2 SEODP Algorithm

The general idea is that SEODP is a semi-evolutionary algorithm. The evolutionary method guarantees that individuals with the best fitness are selected, and the random method guarantees that the mutation operation will be performed, and SEODP will not fall into a local optimal value.

5.2.1 SEODP Architectures

We choose different types of DN architectures according to different types of data. For image datasets we choose a CNN model, whereas for non-image datasets we choose a compact Multi-Layer Network (MLN) model, due to its simplicity and effectiveness.

To be specific, a CNN is used for the classification of the Animal Image dataset, and MLNs are used for the other datasets.

Table 5.1: CNN Architecture

CNN Architecture
Convolutional Layer
MaxPool
Convolutional Layer
MaxPool
DenseLayer
OutputLayer

Table 5.2: CNN Parameters

Parameters	Value Range
<i>BatchSize</i>	[1,100]
<i>KernelSize</i>	[1,5]
<i>MaxpoolSize</i>	[1,5]
<i>DenseLayerSize</i>	[1,50]

Table 5.3: MLN

MLN Architecture
Dense Layer
Output Layer

Table 5.4: MLN Parameters

Parameters	Value Range
<i>BatchSize</i>	[1,100]
<i>DenseLayerSize</i>	[1,100]

The overall accuracy of the classification is used as the fitness function. The details of the DN architectures and hyperparameters are shown in Tables 5.1 to 5.4.

Table 5.1 shows the architecture of the CNN, and Table 5.2 shows the hyperparameters and the value ranges for the CNN model.

Table 5.3 shows the architecture of the MLN, and Table 5.4 shows the hyperparameters and the value ranges for the MLN model.

5.2.2 The SEODP Framework

The evolutionary process of SEODP basically has two stages. The first stage performs the evolutionary and random search operations. The operation of these processes in the proposed framework for SEODP is shown in Algorithm 5.

For every generation, SEODP selects the top $L * F$ individuals with the best fitness, and then randomly generates the other $L * (1 - F)$ individuals to replace the $L * (1 - F)$ individuals having lower fitness. This approach guarantees that the best individuals will be selected for each generation (due to the evolutionary process), and that the algorithm

does not fall into locally optimal values (due to the random mutation). After the loop of *MaxGen* generations, the population should contain the individuals with optimal fitness.

The second stage performs the crossover operation, which is a genetic operator used to combine the genetic information of two parents to generate new offspring. In the SEODP algorithm, the crossover operation is implemented by mixing the genes of two parents with a certain probability (the crossover rate). For example, the parameter for *KernelSize* from individual i and the parameter for *MaxpoolSize* from individual j are selected to generate the new individual $w_{i,j}$.

Equation 5.1 shows more more detailed information on the crossover operation:

$$u_k = \begin{cases} w_i^k, & \text{if } (rndreal(0, 1) < CR) \\ w_j^k, & \text{otherwise} \end{cases} ; \quad (5.1)$$

where u_k is the value of the k th gene for the individual u , w_i^k is the value of the k th gene for individual w_i , w_j^k is the value of the k th gene for individual w_j , $rndreal(0, 1)$ is a real number randomly generated from (0,1), and CR is the crossover rate.

Algorithm 5: SEODP

Input:

Evolution Population W ; The Size of Population L ;

Maximum Evolution Generation: $MaxGen$;

Training Dataset D^a , Testing Dataset D^b ;

Output:

Sets of individuals (Combination of HyperParameters) that have best fitness over testing dataset D^b ;

- 1: $W \leftarrow$ The initial $w_{i,j}$ value for each individual w_i is set to a whole number with the value range;
 - $t \leftarrow 1$;
 - 2: **while** $t \leq MaxGen$ **do**
 - 3: **for** $i=1$ to L **do**
 - 4: $f[w_i^t] \leftarrow$ Use w_i^t as combination of hyperparameters to build the DN model over D^a , use D^b to calculate the fitness of w_i^t in the population W^t ;
 - 5: **end for**
 - 6: $W_c \leftarrow$ Choose the best $L * F$ individuals with highest fitness value;
 - 7: $W_r \leftarrow$ Random generate $L * (1 - F)$ individuals as mutation operation;
 - 8: $W^{t+1} \leftarrow$ Combine the W_c and W_r to form the new generation of population W^{t+1} ;
 - 9: $t = t + 1$;
 - 10: **end while**
 - 11: $i \leftarrow 1$;
 - 12: **while** $i \leq MaxCP$ **do**
 - 13: $w_i, w_j \leftarrow$ Randomly select two different individuals in W^{MaxGen} ;
 - 14: $u_{i,j} \leftarrow$ Use 5.1 to perform crossover operation to generation new individuals $u_{i,j}$, add $u_{i,j}$ to the Crossover population U ;
 - 15: $i = i + 1$;
 - 16: **end while**
 - 17: $U_c \leftarrow$ Choose the best number of $BestHP$ individuals with highest fitness value;
 - 18: **return** U_c ;
-

5.3 Experiments

The SEODP algorithm was implemented and evaluated on a number of standard datasets, as well as on a novel real-world dataset.

5.3.1 Datasets

A number of different types of datasets were used to validate the effectiveness of SEODP. Detailed information of the datasets are shown in Table 5.5.

The Linear and Saturn datasets are popular for binary classification. Visualizations of these two datasets are shown in Figures 5.1 and 5.2. The Skin dataset is from the UCI data repository [3]. The Animal Image dataset is provided as part of the Java deep learning platform we used for the implementation — Deeplearning4j. This dataset contains images of animals falling into four categories: bear, deer, duck, and turtle.

The Donor dataset is a a novel real-world dataset from a charity organization that holds appeal and contact history information of potential philanthropic donors. Modelling can make use of features of the dataset such as the number of phone calls, number of events attended, and geographical information. In the experiments a sample of 5000

Table 5.5: Description of datasets.

Datasets	Instances	Attributes	Classes	Characteristic
Linear	1200	2	2	Linearly separable
Saturn	600	2	2	Simple and not linearly separable
Skin	245057	3	2	Big dataset, large number of instances
Animal Image	83	256*256	4	Image dataset, large number of attributes
Donors	10000	6	2	Novel Real-world Dataset

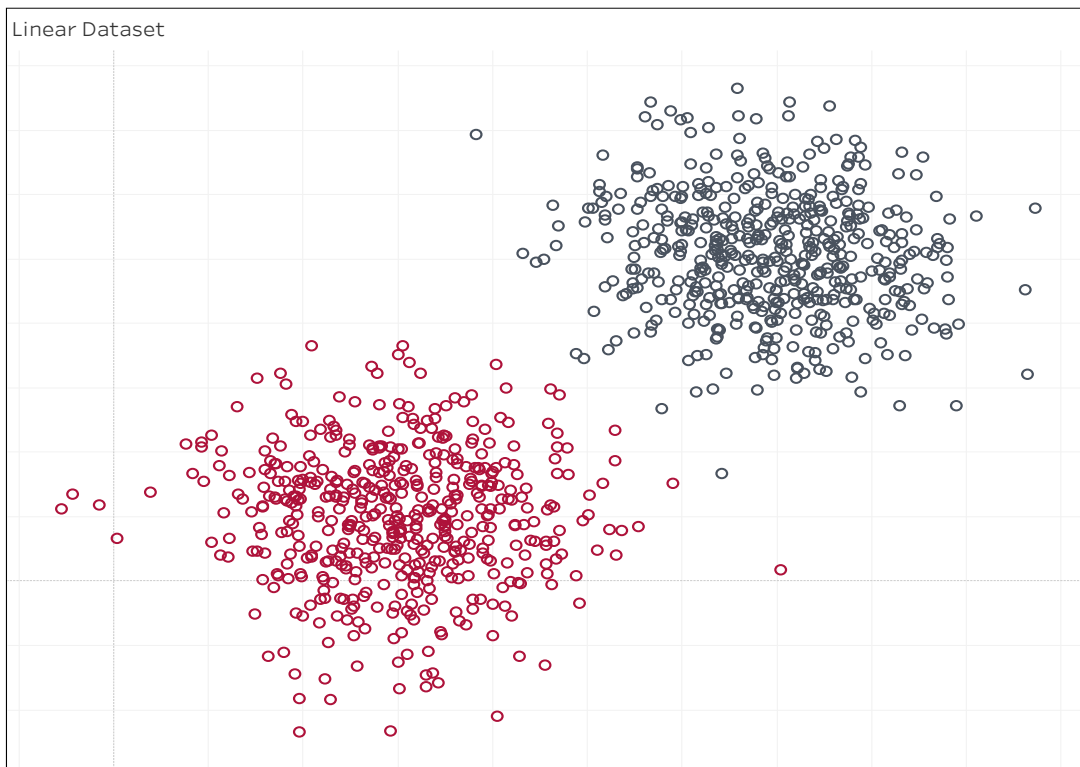


Figure 5.1: Visualization of the Linear dataset.

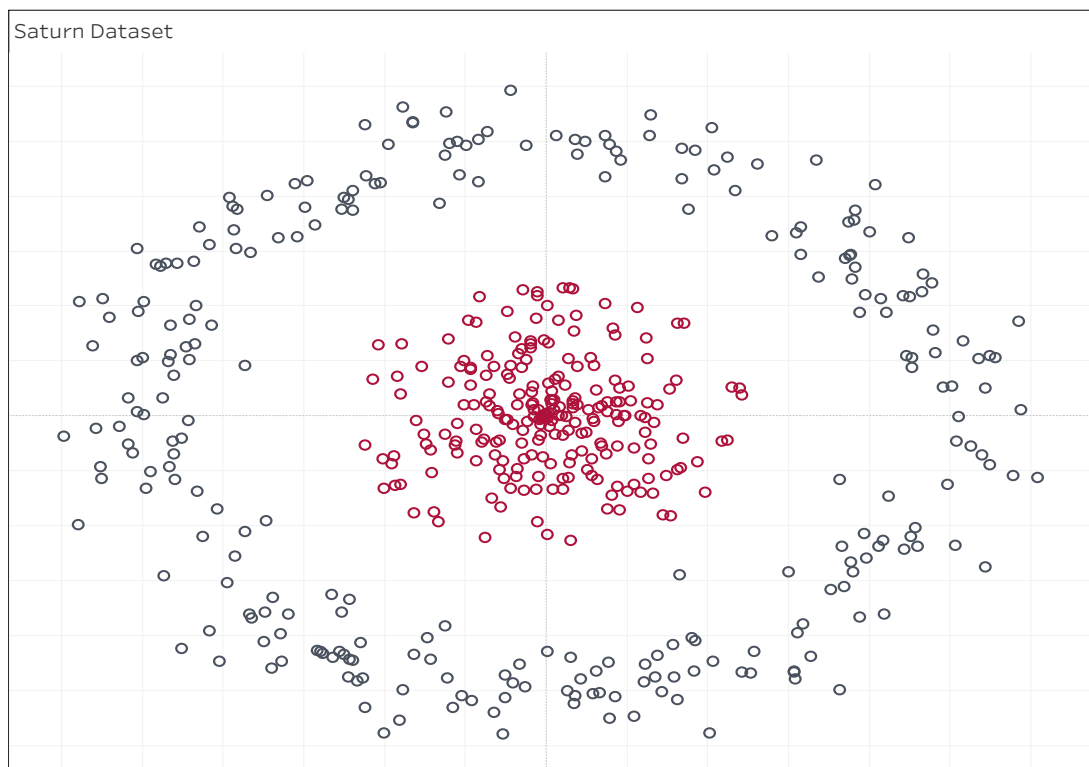


Figure 5.2: Visualization of the Saturn dataset.

donor and 5000 non-donor records are used for the classification.

All datasets are preprocessed to replace missing values, and then normalized.

5.3.2 Experimental Platform

We used the Java language and the Java deep learning platform — Deeplearning4j — to implement the algorithm of SEODP. The relevant information concerning the computer resources we used for the experiments are as follows:

- *System Model*: HP Z240 Tower Workstation.
- *System Type*: x64-based PC.
- *Processor*: Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz, 3401 Mhz, 4 Core(s), 8 Logical Processor(s).
- *OS Name*: Microsoft Windows 10 Enterprise.
- *Total Physical Memory*: 15.9 GB.
- *Nd4jBackend*: CpuBackend.
- *Number of threads used for BLAS*: 4.
- *Memory Allocated*: 3.5 GB.

ND4J is a scientific computing and linear algebra library, written in the programming language Java, operating on the Java virtual machine (JVM). ND4J supports two types of backends, CPU mode and GPU mode. CPU mode is used for these experiments.

Basic Linear Algebra Subprograms (BLAS) is a standard software library containing a set of low-level routines for performing common linear algebra operations, such as vector addition, scalar multiplication and dot products.

The results of applying the above processes for the five different datasets show clearly in the next section that SEODP is able to optimize the selection of hyperparameters and fitness in a very time-effective way.

5.3.3 Convergence Learning Curves of SEODP

The performance of SEODP in Figure 5.3 shows that the algorithm is quite effective in searching for the optimal hyperparameters for different types of datasets.

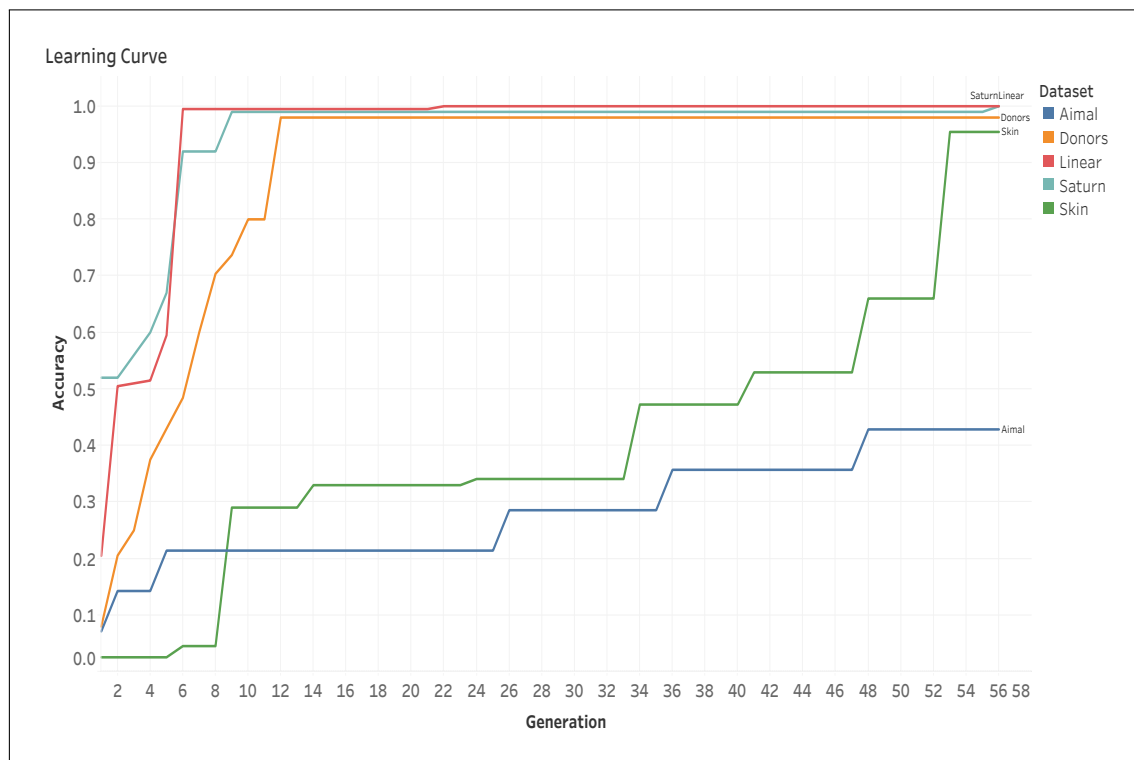


Figure 5.3: Convergence of SEODP on all datasets as learning curves.

Over all the datasets, accuracy increases as the number of generations increases.

For some of the datasets, SEODP converges at a very early stage. For example, SEODP achieves 100% accuracy on dataset Linear, and 99% accuracy on dataset Saturn, at the 6th and 9th iteration, respectively, which is a fairly early convergence.

One result that needs to be emphasized is SEODP actually achieves 98% accuracy in our real Donor dataset at the 12th iteration, which is a very good result.

5.3.4 Analysis of Hyperparameters vs. Accuracy

Since we only choose two parameters for SEODP to optimize for MLNs in these experiments — *BatchSize* and *DenseLayerSize* — we use a Java library called jzy3d to generate 3D visualizations to show the impact of hyperparameters on accuracy of the trained models. This allows a more detailed analysis of the effect of the choice of different values for hyperparameters on the accuracy, i.e., what the SEODP algorithm is trying to optimize.

For all the following 3D figures, the X axis represents the first hyperparameter, which is the *BatchSize* of the DN, the Y axis represents the second hyperparameter which is the *DenseLayerSize* of the DN, and Z represents the accuracy of the DN model using X, Y as hyperparameter values. The other figures for each dataset show the effect of tuning in hyperparameter space by plotting X against Y , and *vice versa*, with the colour gradient from the 3D plots indicating the accuracy in each case.

“Linear” dataset

Figures 5.4 to 5.6 show the visualizations for the “Linear” dataset, which indicate that, in general:

- The accuracy decreases as the *BatchSize* increases. When a relatively small

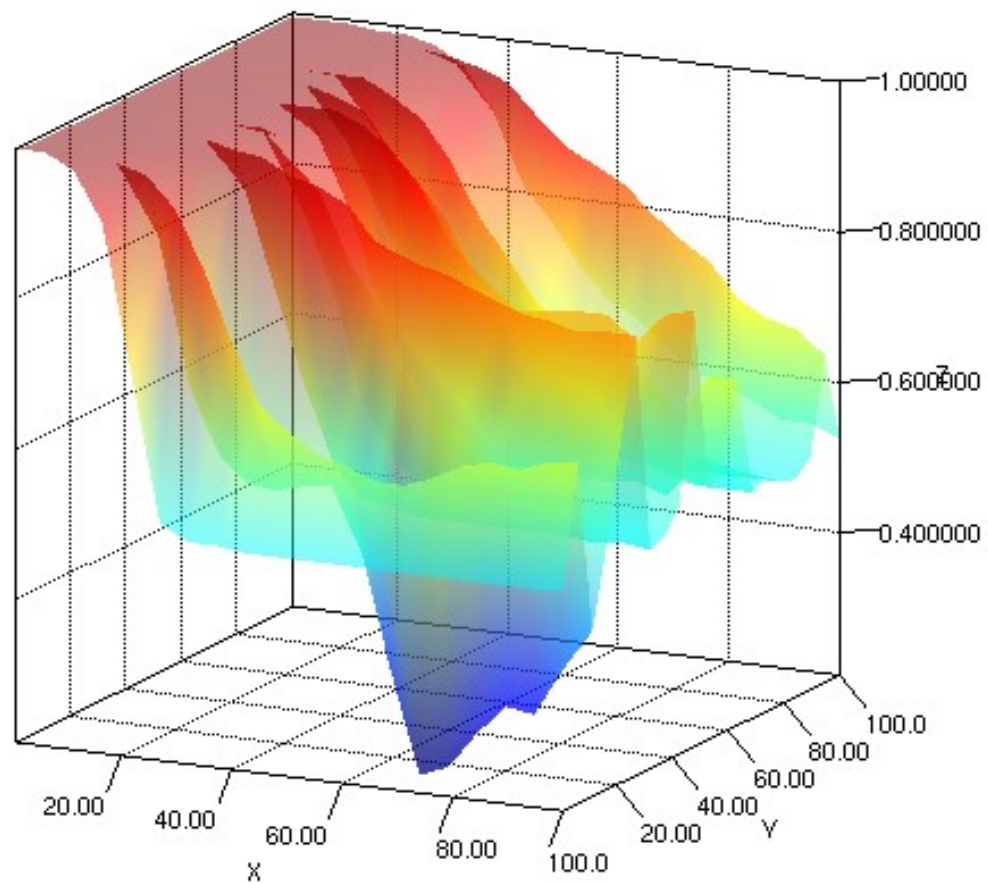


Figure 5.4: “Linear” dataset hyperparameter optimization results with SEODP in 3D with $X = BatchSize$ and $Y = DenseLayerSize$ for $Z = Accuracy$.

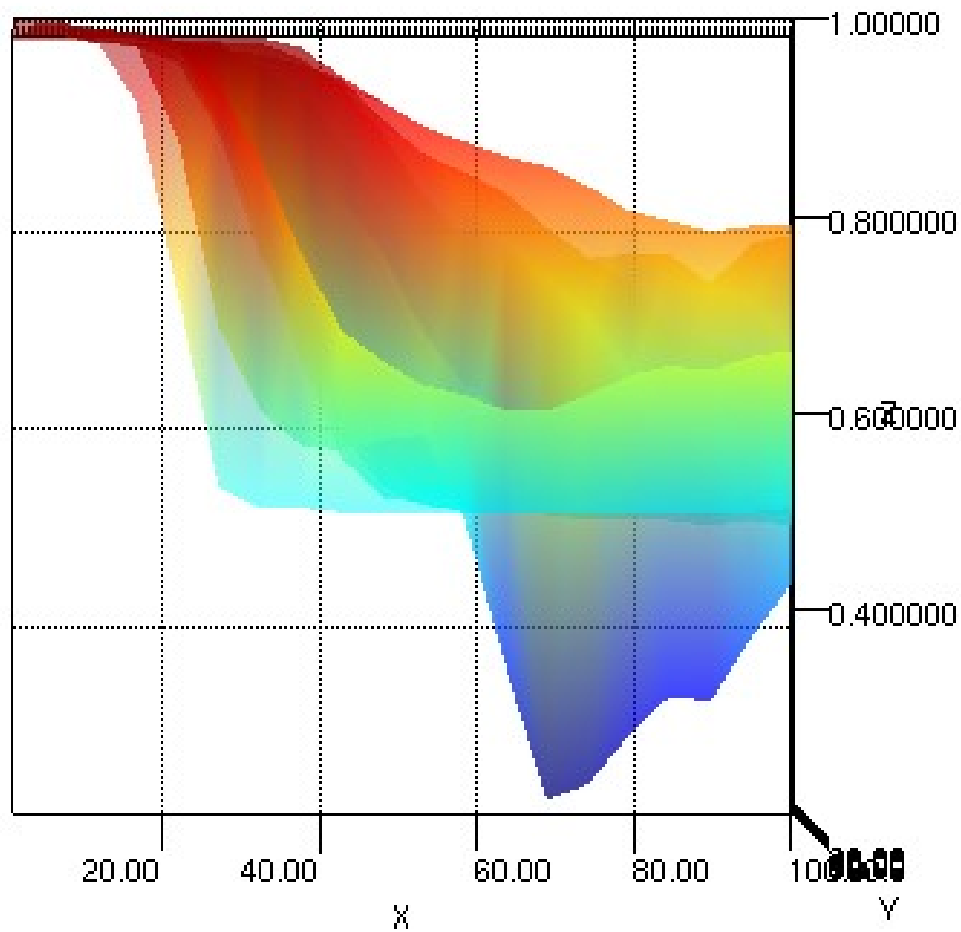


Figure 5.5: “Linear” dataset hyperparameter optimization results with SEODP in 2D for $X = BatchSize$ vs. $Y = DenseLayerSize$ with colour showing accuracy.

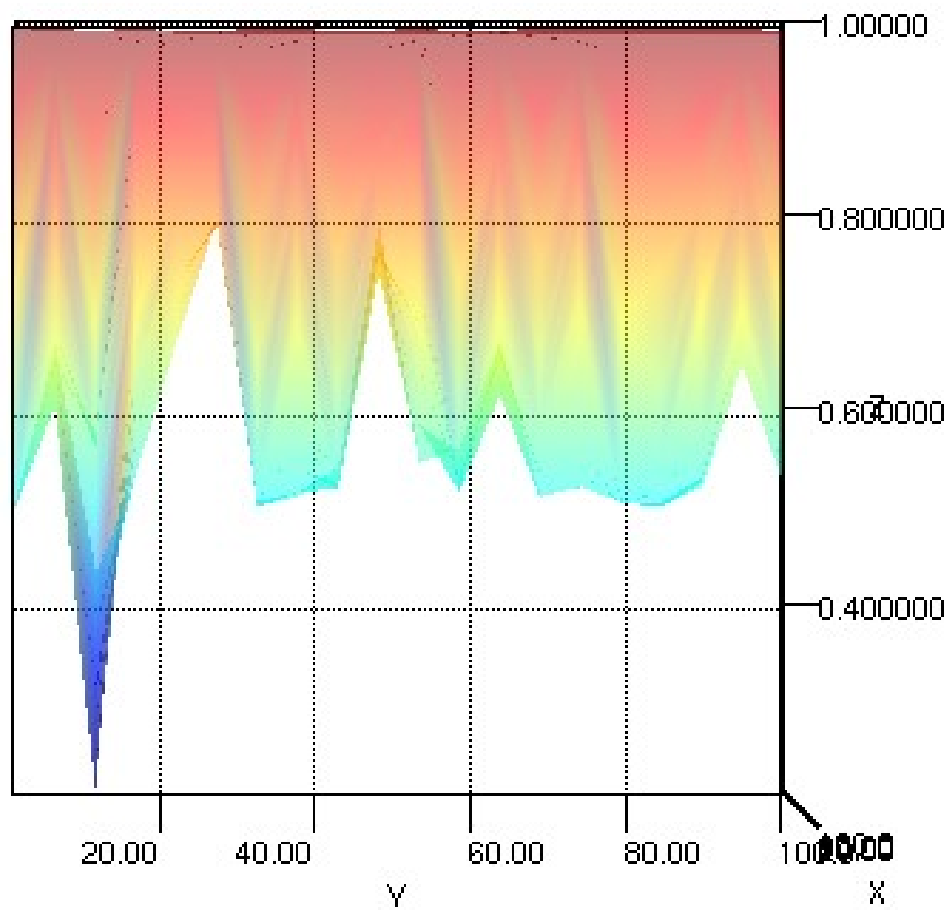


Figure 5.6: “Linear” dataset results in 2D for $Y = DenseLayerSize$ vs. $X = BatchSize$ with colour showing accuracy.

BatchSize (less than 10, for example) is chosen, the accuracy is around 100% regardless of the value for the *DenseLayerSize*. When a relatively large *BatchSize* (greater than 60, for example) is chosen, the accuracy varies from around 5% to 85%, due to the different choices of *DenseLayerSize*.

- There is no obvious trend concerning how accuracy is influenced by the *DenseLayerSize*. For most values, the accuracy varies from 50% to 100%. However, there are some exceptions: for values between 30 and 50, the accuracy varies from around 80% to 100%, which indicates a good choice for *DenseLayerSize*. On the contrary, with value 10, the accuracy varies from around 1% to 100%, which is very unstable, and thus indicates a bad choice for *DenseLayerSize*.

“Saturn” dataset

Figures 5.7 to 5.9 show the visualizations for the “Saturn” dataset, which indicates that in general:

- The accuracy decreases as the *BatchSize* increases — a similar pattern to the Linear dataset.
- The MLN performs better when the *DenseLayerSize* is greater than 20, similar to the Linear dataset. In this case, the accuracy varies from 52% to 100%. When the value is less than 20, the accuracy varies from around 33% to 98%, which is below average.

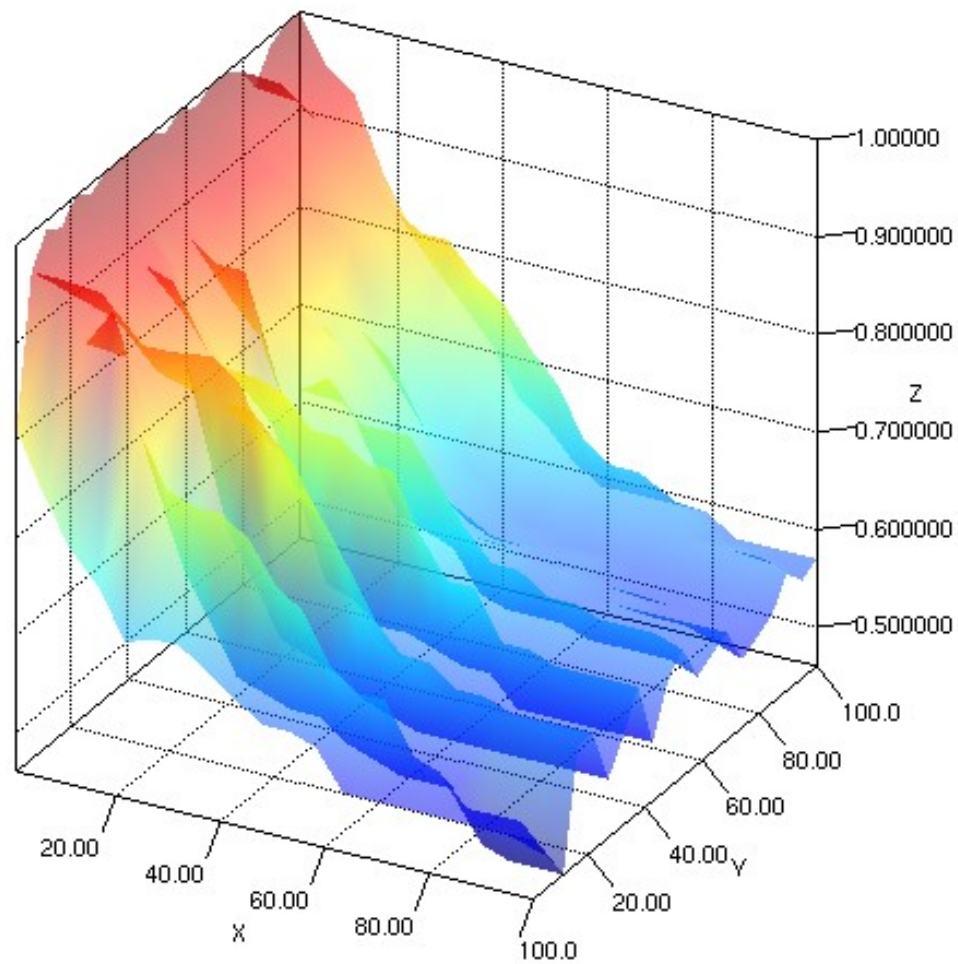


Figure 5.7: “Saturn” dataset hyperparameter optimization results with SEODP in 3D with $X = BatchSize$ and $Y = DenseLayerSize$ for $Z = Accuracy$.

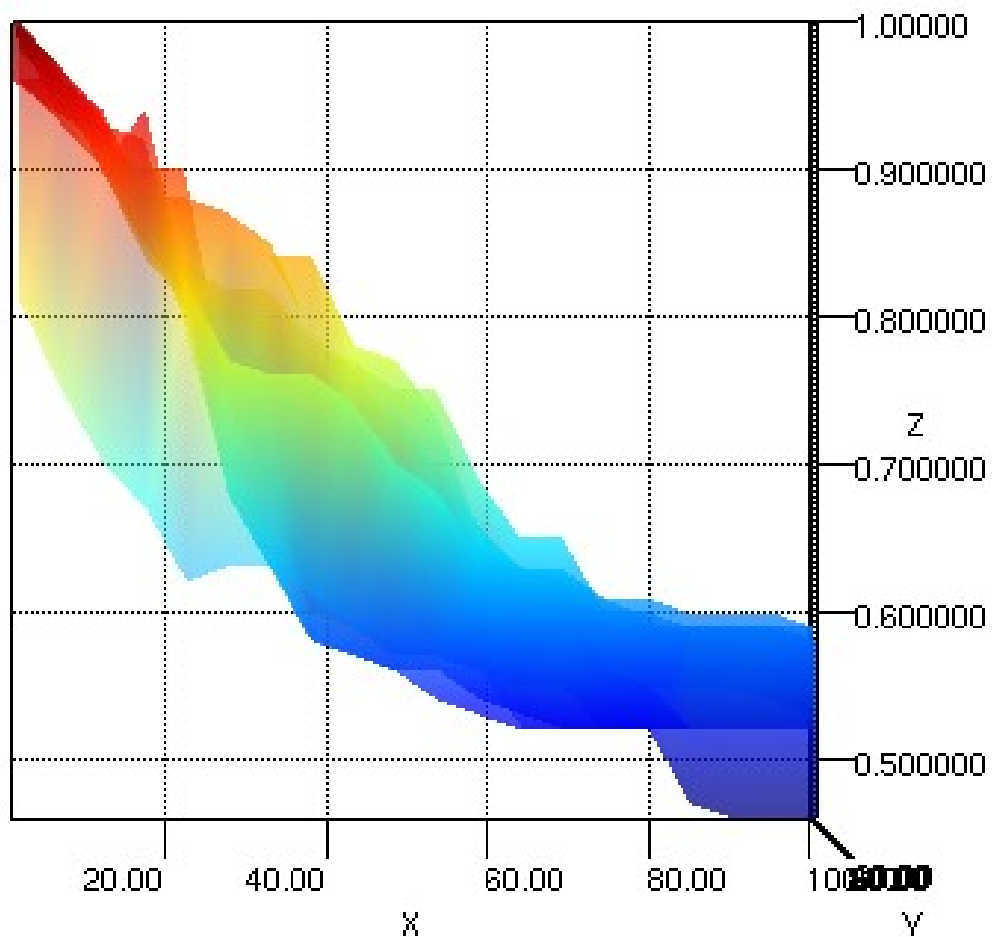


Figure 5.8: “Saturn” dataset hyperparameter optimization results with SEODP in 2D for $X = \text{BatchSize}$ vs. $Y = \text{DenseLayerSize}$ with colour showing accuracy.

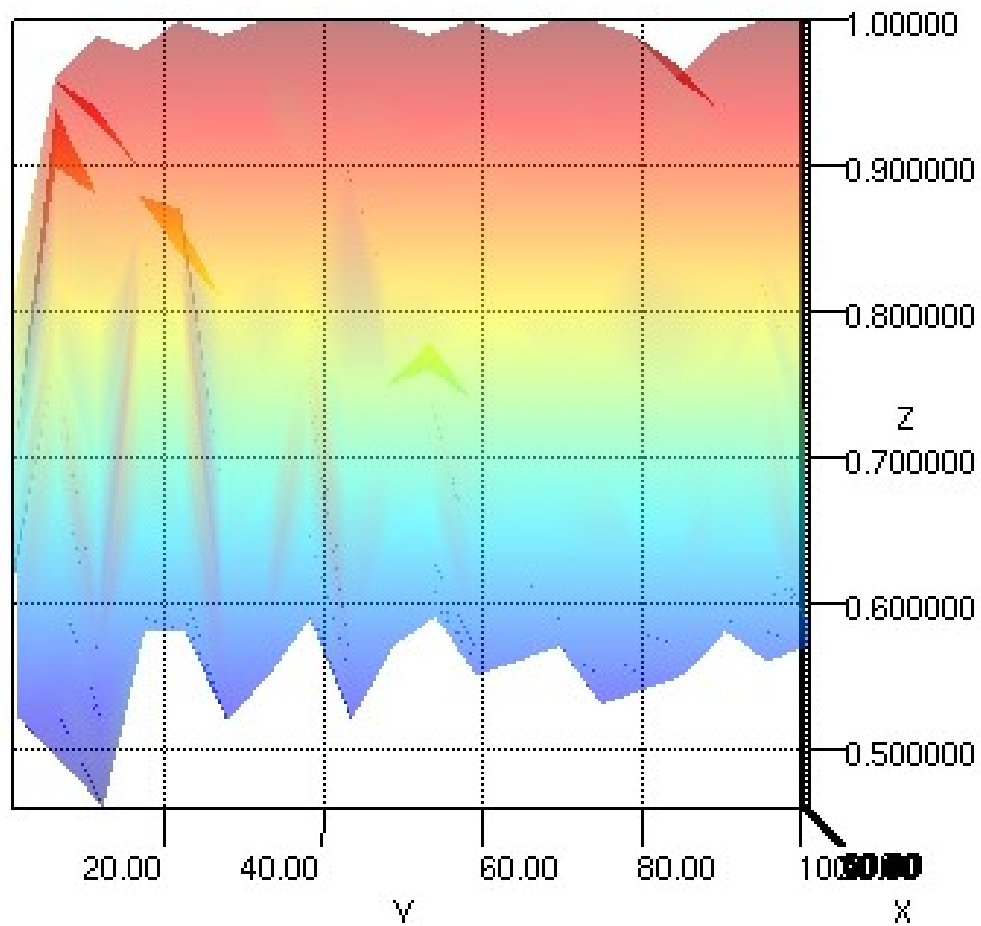


Figure 5.9: “Saturn” dataset results in 2D for $Y = DenseLayerSize$ vs. $X = BatchSize$ with colour showing accuracy.

“Skin” dataset

Figures 5.10 to 5.12 show the visualizations for the “Skin” dataset, which indicate that in general:

- There is no obvious trend between accuracy and *BatchSize*, which is different from what is observed with the previous datasets. The range in accuracy over hyperparameter space is wide — from 0 to 100%.
- The MLN performs much better when *DenseLayerSize* is around 50. In this case, accuracy varies from 40% to 100%. For *DenseLayerSize* with values other than 50, the accuracy can vary from around 0 to 100%, which is very unstable.

“Donor” dataset

Figures 5.13 to 5.15 show the visualizations for the “Donor” dataset, which indicate that:

- The MLN performs better when *BatchSize* is greater than 10. In this case, the accuracy varies between 13% and 99.3%. When the value of *DenseLayerSize* is less than 10, the accuracy varies from around 0 to 99.3%, depending on *DenseLayerSize*.
- The MLN performs best when *DenseLayerSize* is around 20, 45, 55 and 70, and it performs worst when *DenseLayerSize* is around 5, 35, 50, 60, and 95.

Best hyperparameters obtained for each dataset

Table 5.6 shows details of the best hyperparameters obtained using the SEODP algorithm for each dataset.

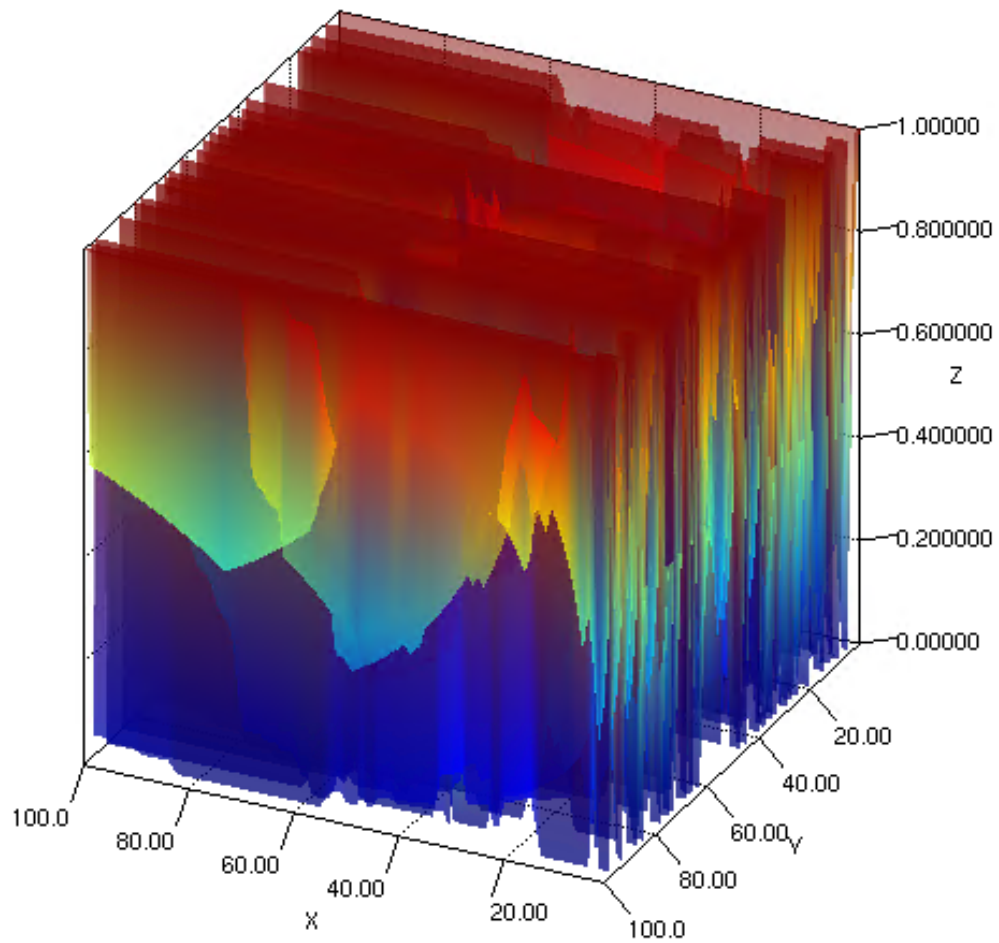


Figure 5.10: “Skin” dataset hyperparameter optimization results with SEODP in 3D with $X = \text{BatchSize}$ and $Y = \text{DenseLayerSize}$ for $Z = \text{Accuracy}$.

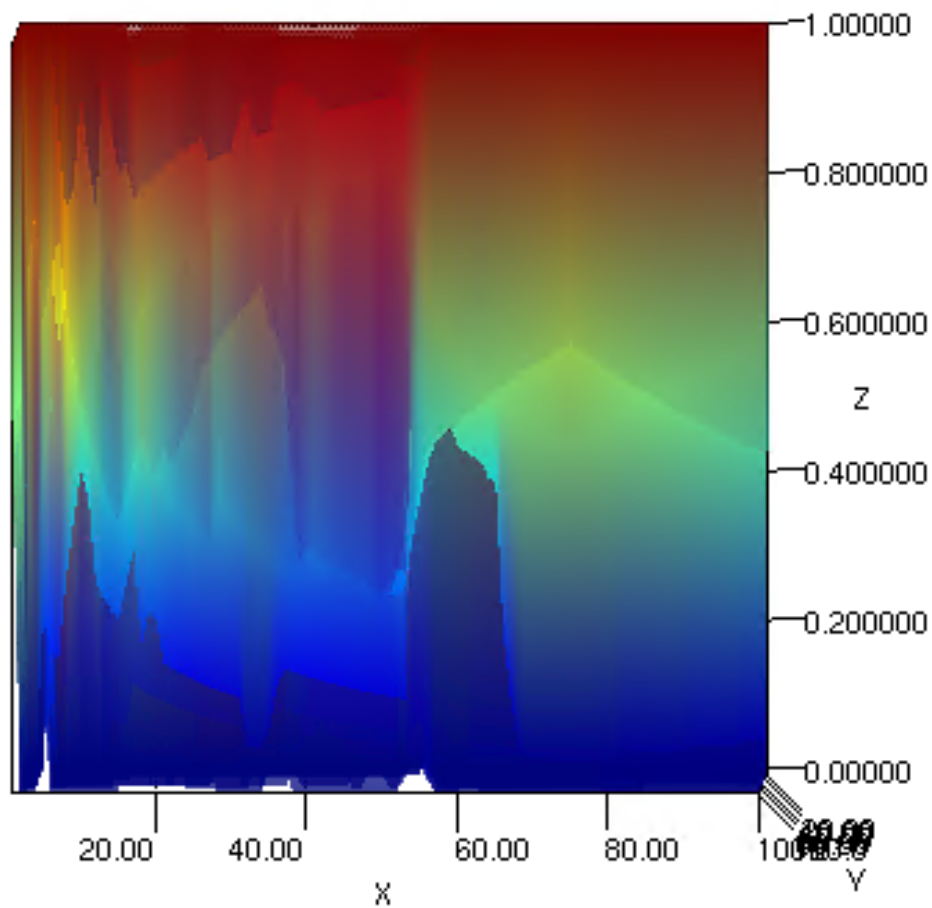


Figure 5.11: “Skin” dataset hyperparameter optimization results with SEODP in 2D for $X = BatchSize$ vs. $Y = DenseLayerSize$ with colour showing accuracy.

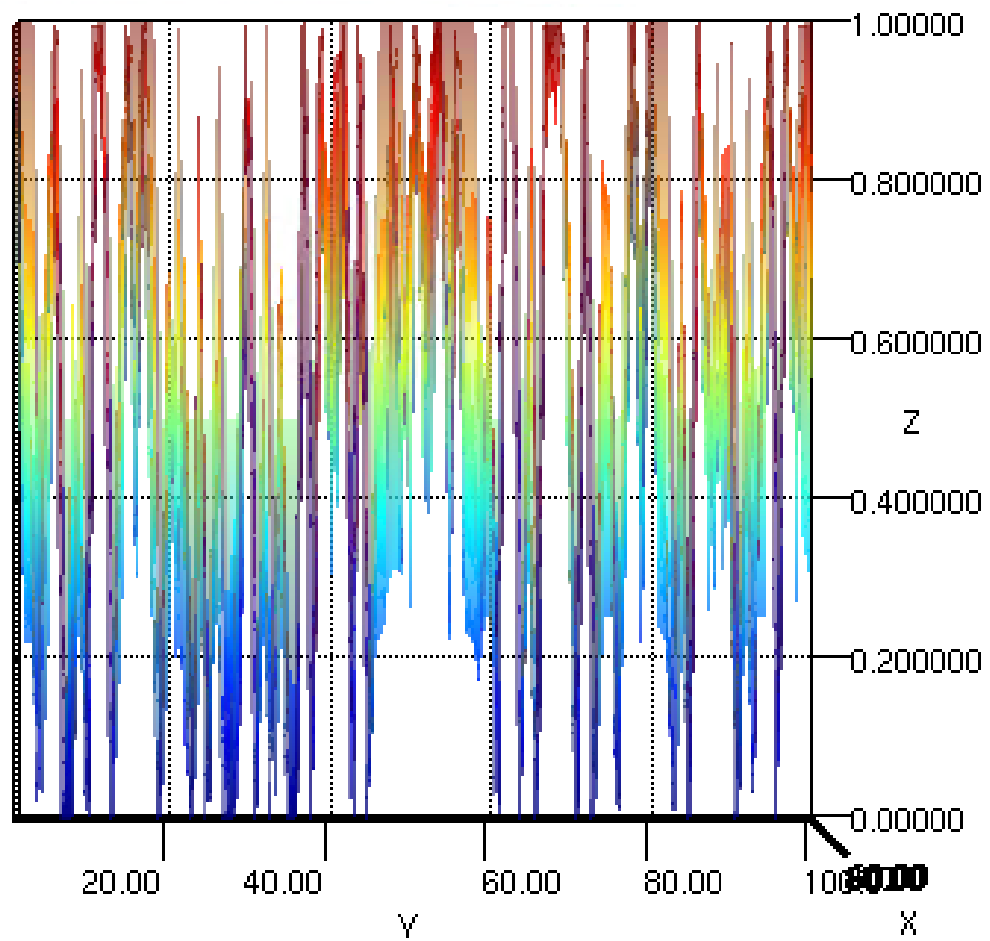


Figure 5.12: “Skin” dataset results in 2D for $Y = DenseLayerSize$ vs. $X = BatchSize$ with colour showing accuracy.

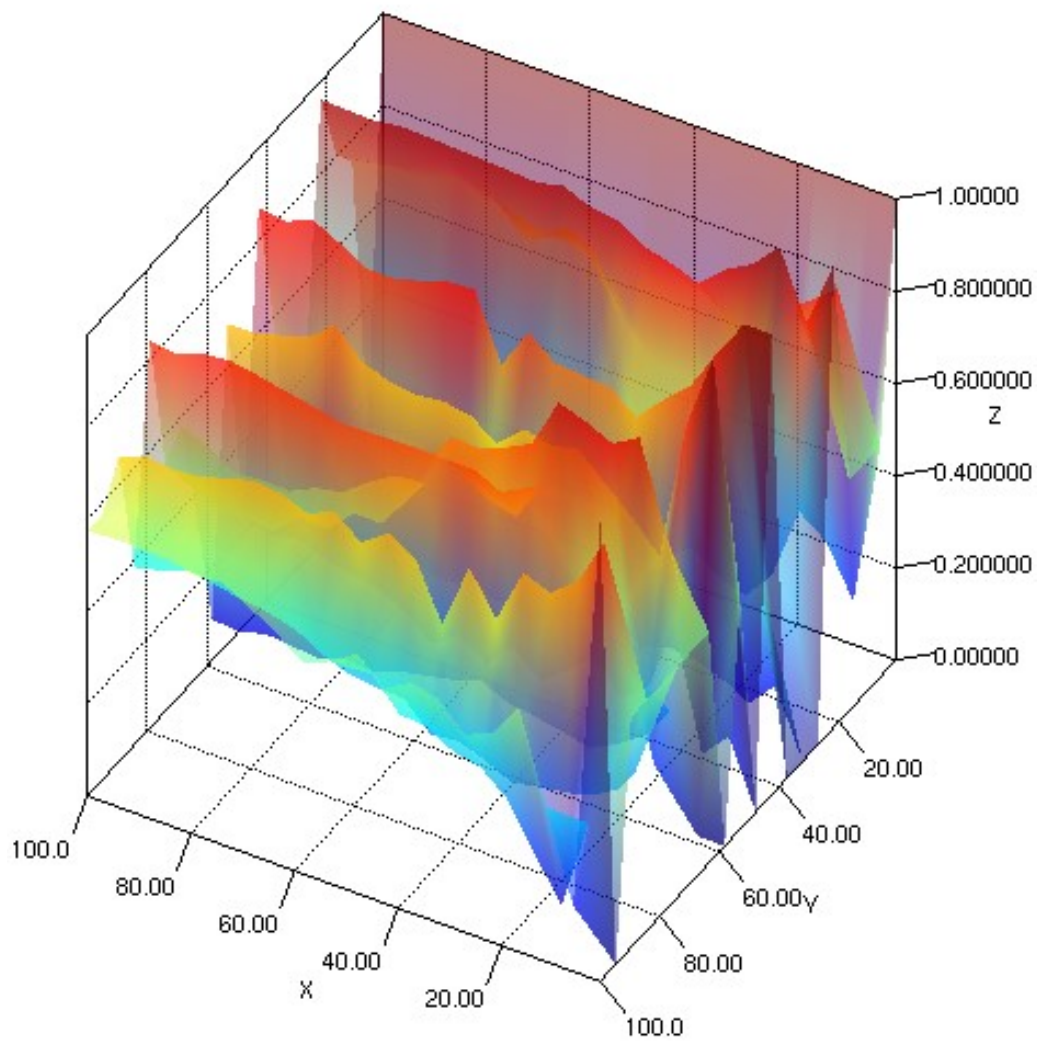


Figure 5.13: “Donor” dataset hyperparameter optimization results with SEODP in 3D with $X = \text{BatchSize}$ and $Y = \text{DenseLayerSize}$ for $Z = \text{Accuracy}$.

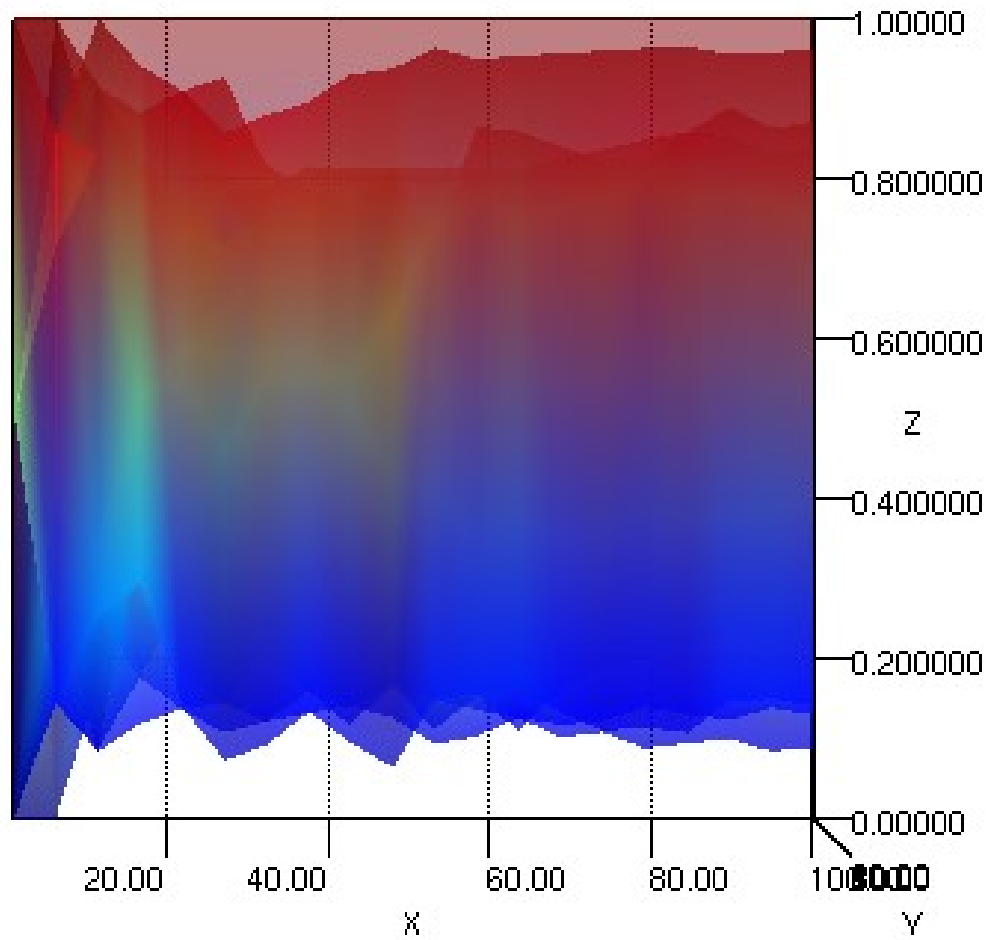


Figure 5.14: “Donor” dataset hyperparameter optimization results with SEODP in 2D for $X = \text{BatchSize}$ vs. $Y = \text{DenseLayerSize}$ with colour showing accuracy.

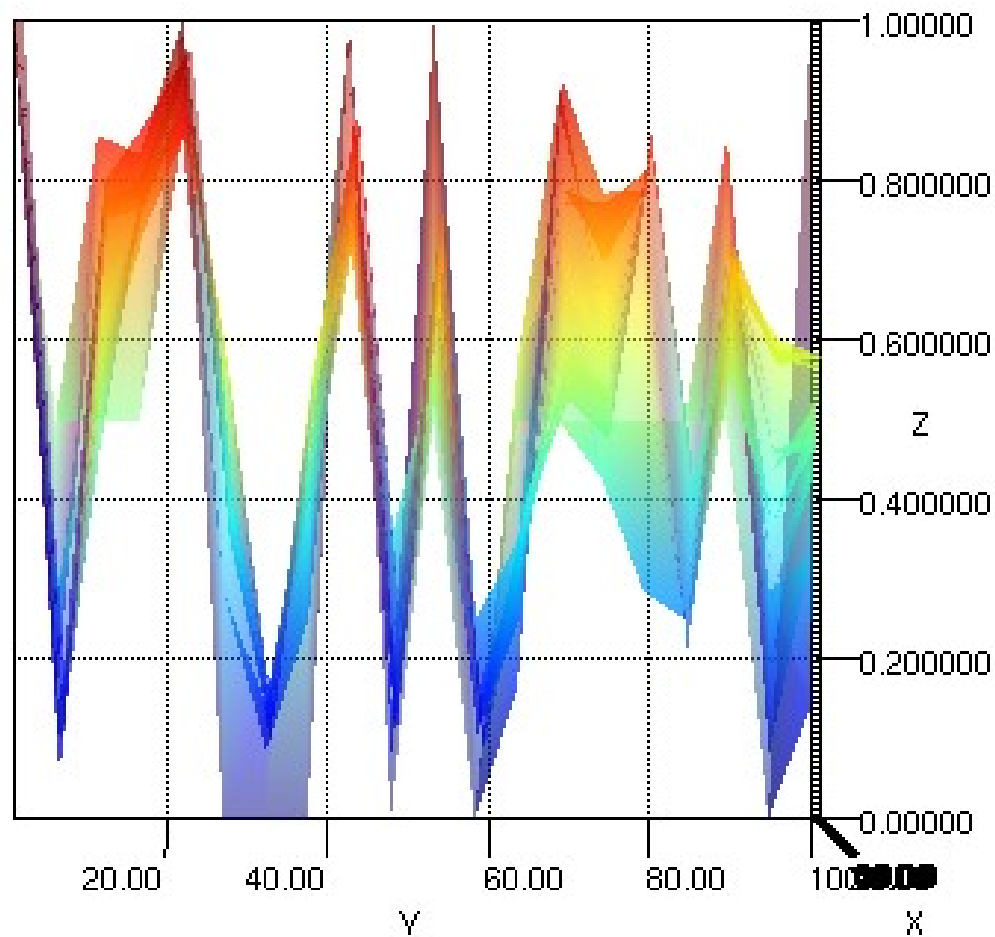


Figure 5.15: “Donor” dataset results in 2D for $Y = DenseLayerSize$ vs. $X = BatchSize$ with colour showing accuracy.

Table 5.6: Best hyperparameters for each dataset

Linear		Saturn		Skin		Donor		Animal	
Parameters	Accuracy	Parameters	Accuracy	Parameters	Accuracy	Parameters	Accuracy	Parameters	Accuracy
[1, 2]	100.00%	[1, 17]	100.00%	[68, 38]	100.00%	[3, 84]	98.00%	[63, 5, 3, 6]	42.86%
[1, 3]	100.00%	[1, 25]	100.00%	[88, 13]	100.00%	[2, 78]	98.00%	[53, 2, 1, 32]	42.86%
[1, 5]	100.00%	[1, 29]	100.00%	[91, 13]	100.00%	[3, 55]	98.00%	[38, 2, 3, 26]	42.86%
[1, 8]	100.00%	[1, 34]	100.00%	[83, 70]	100.00%	[2, 71]	98.00%	[4, 1, 1, 41]	42.86%
[2, 2]	100.00%	[1, 39]	100.00%	[91, 100]	100.00%	[45, 71]	91.11%	[15, 1, 2, 19]	42.86%
[2, 3]	100.00%	[1, 41]	100.00%	[56, 2]	100.00%	[89, 84]	91.01%	[20, 1, 5, 46]	42.86%
[2, 4]	100.00%	[1, 44]	100.00%	[88, 13]	100.00%	[89, 71]	89.89%	[15, 1, 2, 19]	42.86%
[2, 5]	100.00%	[1, 54]	100.00%	[83, 70]	100.00%	[72, 36]	88.89%	[20, 1, 5, 46]	42.86%
[2, 6]	100.00%	[1, 58]	100.00%	[12, 2]	100.00%	[9, 54]	88.89%	[63, 5, 3, 6]	42.86%
[2, 8]	100.00%	[1, 60]	100.00%	[91, 70]	100.00%	[72, 36]	88.89%	[20, 1, 5, 46]	42.86%
[2, 9]	100.00%	[1, 65]	100.00%	[68, 38]	100.00%	[9, 54]	88.89%	[34, 2, 4, 47]	35.71%
[2, 10]	100.00%	[1, 71]	100.00%	[88, 13]	100.00%	[89, 36]	88.76%	[66, 1, 5, 15]	35.71%
[2, 11]	100.00%	[1, 76]	100.00%	[68, 38]	100.00%	[44, 71]	88.64%	[34, 2, 3, 47]	35.71%

From the table, we can see that, in general, for datasets “Linear” and “Saturn”, the accuracy decreases as *BatchSize* increases, and the accuracy is relatively high when *DenseLayerSize* is greater than 20. However, for datasets “Skin” and “Donor”, the pattern is not clear, which indicates we might need to change the basic structure of the DNs, such as adding more layers or using different types of DNs — CNNs, Recurrent Neural Networks (RNNs), or some other architectures.

5.3.5 SEODP vs. Greedy Search

To validate the effectiveness of SEODP, we ran a greedy search of the hyperparameter space for the MLN architecture over the above four non-image datasets. Note that for the “Animal” image dataset, because a CNN model is used to perform the classification, it is quite difficult to perform a greedy search over the parameter space. The reason is that for the CNN model there are dependencies between the selection of individual

Table 5.7: Best vs. worst using greedy search

Datasets	Accuracy		Parameter	
	Best	Worst	Best	Worst
Linear	100.00%	1.00%	[11, 83]	[100, 19]
Saturn	100.00%	24.00%	[3, 74]	[71, 6]
Skin	100.00%	0.00%	[68, 38]	[68, 46]
Donor	99.30%	0.00%	[3,23]	[30,51]

hyperparameters.

For example, one restriction is $0 < KH \leq inHeight + 2 * padH$. Here, KH is the kernel height, $inHeight$ is the input height for this layer, and $padH$ is the padding height. We will not go into detail here concerning the formula, but the point is that the selection of one hyperparameter, $MaxpoolSize$, for example, is dependent on the selection of another hyperparameter, such as $KernelSize$. Such dependencies among the set of hyperparameters makes a greedy search for the CNN architecture very hard.

For the MLN model, since the model has two hyperparameters, $BatchSize$ and $DenseLayerSize$, both with a value range of $[1, 100]$, the overall parameter space is $100 * 100$ when incrementing in steps of 1, which means we need to train and test 10,000 models with different combinations of hyperparameters.

Table 5.7 shows the highest and lowest accuracies found, and one of the corresponding hyperparameters used to achieve each accuracy. The results show that the choice of values for hyperparameters are very important for the performance of the model, as the range of accuracy is quite wide — for the Donor dataset, the range of accuracies actually varies from 0 to 99.3%.

Table 5.8 shows a comparison between SEODP and greedy search. The table shows

Table 5.8: SEODP vs. Greedy Search

Datasets	Accuracy		Runtime	
	SEODP	Grid Search	SEODP	Grid Search
Linear	100%	100%	40s	2106s
Saturn	100%	100%	22s	1139s
Skin	100%	100%	397s	18847s
Donors	98%	99.3%	105s	5504s

that for all the five datasets, SEODP achieves almost the same level of accuracy, but only uses less than 2% of the running time of greedy search. Furthermore, for the “Linear”, “Saturn” and “Skin” datasets, SEODP actually achieves the same accuracy as greedy search, which is 100%. This shows clearly that deep learning with the use of SEODP to search for hyperparameters is a very efficient, as well as effective, method.

Additioanlly, for the “Animal” image dataset, the demo model in the Deeplearning4j library scores an accuracy of 14.29%, while using SEODP to tune the hyperparameters secures an accuracy of 42.86%, which is much better than the benchmark.

In this work, we only chose two hyperparameters to optimize, since our purpose is to show how the framework works. It is very straightforward to apply the same framework to optimize more hyperparameters — as long as there are no constraints between the candidate hyperparameters. For larger numbers of hyperparameter in the optimization task, the run-time and accuracy of SEODP will be different. The actual effect observed will depend on what hyperparameters will be chosen to be optimized, and how those specific hyperparameter will affect the deep networks.

Chapter 6

Conclusions and Future Work

This thesis started from the observation that machine learning algorithms usually have a number of hyperparameters, and that the choice of values for these hyperparameters may have a significant impact on the performance of these algorithms. There is currently no standard solution to the general problem of searching for values for hyperparameters that reliably produce the best performance (or a close approximation to it) for a range of machine learning algorithms. Furthermore, as machine learning models and algorithms increase in complexity in order to achieve higher performance (such as predictive accuracy) the complexity of hyperparameter spaces to be searched will increase also.

However, since this space is not, in general, structured in a way that typical optimization or search algorithms can be applied, often machine learning practitioners instead tend to apply either brute-force algorithms or *ad hoc* manual hyperparameter tuning.

An alternative approach to approximate the search for an optimal set of values for hyperparameters is the use of evolutionary algorithms. These are suitable for the hyperparameter space search problem since they do not require *a priori* assumptions about the structure of the search space. In this thesis, we introduced three improved evolutionary-

based algorithms to solve different problems in the area of hyperparameter search for machine learning algorithms. These were implemented for different types of machine learning algorithms, and for different types of hyperparameters. Evaluation of our methods showed that they lead to improved performance on a number of metrics.

6.1 Contributions

6.1.1 EWLNB

Firstly, we proposed a unique Weighted Lazy Naive Bayes algorithm (WLNB) to learn the class labels of the test instances, using a lazy learning method. To improve the performance of WLNB, a self-adaptive evolutionary process was used to optimize two key parameters in WLNB, which enhanced the algorithmic performance according to the NB objective function. Experiments and comparisons on 56 UCI benchmark datasets demonstrated that our proposed method (*i.e.*, EWLNB) achieved superior performance to other baselines in terms of classification accuracy, and the area under the curve. The studies of convergence learning curves, and the imbalance ratio of datasets, also demonstrated the effectiveness of EWLNB.

6.1.2 EMLNB

Next, we investigated current monitoring methods for Virtual Storage Systems (VSS) and analyzed their existing shortcomings. Motivated to address the existing shortcomings, we presented a new Virtual Storage Monitoring System (VSMS) with the aim of improving the performance of monitoring systems. Our VSMS jointly integrated the methods of MRTG, SMART and RRDtools to construct a comprehensive

monitoring and analysis system for both performance and disk monitoring. In order to analyze the reasons for malfunctions VSS, we proposed a Multi-label Lazy Naive Bayes (MLNB) model for VSMS, and then used a similar evolutionary procedure to our first method to optimize the hyper-parameters for MLNB, thus forming the multi-class method EMLNB.

6.1.3 SEODP

Lastly, as an extension to and improvement of the above algorithms we proposed a new method, SEODP, for optimizing the architecture and hyperparameters of Deep Neural Networks by using a combined semi-evolution and semi-random approach. The experimental results, including on a real-world social behavioral dataset from a charity organization for the task of predicting candidate donors, showed that SEODP is a promising approach.

6.1.4 Summary

Moreover, in general, all the experimental results indicate that evolutionary-based algorithms can achieve very good results in optimizing the parameters of different machine learning algorithms.

6.2 Limitations and Future Work

No thesis is perfect, and we need to make some statement regarding the limitations of the work, but this can be a guide to follow-up research.

6.2.1 EWLNB

The main limitation of EWLNB is that it is still not clear which hyperparameters are important for the models and should be chosen to optimize. Some other limitations include: how to choose the initialized population so that the evolutionary algorithms does not fall into local optima; how to speed up the evolutionary process to approach the global optimal value; also, how to adjust the learning rate to ensure it is not too large to miss the global optimum. Since differential evolution (DE) is a fairly simple, although powerful, evolutionary algorithm, it may be possible for future work to investigate useful extensions or variations of DE specifically oriented to parameter search for machine learning. Further analysis of DE for this kind of search over different datasets, including synthetic datasets as part of controlled experiments, should be carried out to better understand the advantages and disadvantages of DE for such tasks.

6.2.2 EMLNB

The main limitation of the EMLNB is that in this thesis we did not have access to the necessary data to perform the experiments. Also, EMLNB is designed to work offline now, more work need to be done to integrate EMLNB with VSMS to perform online monitoring and prediction in the future.

More specifically, there needs to be further work on how to design and implement the evolutionary search process in an online setting. For example, some work on online optimization could be used as a guide on how to implement this.

6.2.3 EWLNB

For SEODP, the limitations are mainly that the analysis of the hyperparameters for CNN in our experiments was not sufficient. In addition, more detailed research on how the different characteristics of datasets affect the optimal hyperparameters for a model is required. More generally, for deep networks the complexity of the architectures means that SEODP should be tested on tasks of finding values for many more hyperparameters simultaneously. This will let us investigate how well the method can scale to large numbers of hyperparameters, and whether the good level of performance can be maintained as the size of the hyperparameter space increases.

6.2.4 Summary

From all of the above work, it is clear that hyperparameters play a vital role for machine learning models, and how to optimize the hyperparameters for machine learning models is an important topic. There are many methods to perform the optimization for hyperparameters, including manual search, random search, greedy search and evolutionary search. Among all these methods, in this thesis we have shown that evolutionary search has its own advantages, if used suitably. In particular, we have shown that it can find a set of high-performing hyperparameters using less time compared to other methods.

However, the application of evolutionary search can be quite complicated, and can depend on the choice among many possible design options in setting up the evolutionary algorithm. How to address the above problems in order to use evolutionary algorithms in a more effective way will also form part of our future work.

Bibliography

- [1] D. W. Aha, D. Kibler, and M. K. Albert. Instance-based learning algorithms. *Machine learning*, 6(1):37–66, 1991.
- [2] A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, L. N. Bairavasundaram, T. E. Denehy, F. I. Popovici, V. Prabhakaran, and M. Sivathanu. Semantically-smart disk systems: past, present, and future. *ACM SIGMETRICS Performance Evaluation Review*, 33(4):29–35, 2006.
- [3] A. Asuncion and D. Newman. UCI Machine Learning Repository, 2007.
- [4] P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 2014.
- [5] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag. Collaborative hyperparameter tuning. In *International Conference on Machine Learning*, pages 199–207, 2013.
- [6] Y. Bazi, N. Alajlan, F. Melgani, H. Alhichri, S. Malek, and R. R. Yager. Differential evolution extreme learning machine for the classification of hyperspectral images. *IEEE Geoscience and Remote Sensing Letters*, 11(6):1066–1070, 2014.
- [7] L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*, 5(2):78–101, 1966.

- [8] D. Berend and A. Kontorovich. A finite sample analysis of the Naive Bayes classifier. *Journal of Machine Learning Research*, 16:1519–1545, 2015.
- [9] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.
- [10] J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 141–148. PMLR, 2013.
- [11] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyperparameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.
- [12] P. Bermejo, J. A. Gamez, and J. M. Puerta. Speeding up incremental wrapper feature subset selection with Naive Bayes classifier. *Knowledge-Based Systems*, 55:140 – 147, 2014.
- [13] T. M. Breuel. The effects of hyperparameters on SGD training of neural networks. *arXiv preprint arXiv:1508.02788*, 2015.
- [14] D. Cai, C. Zhang, and X. He. Unsupervised feature selection for Multi-Cluster data. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 333–342, 2010.
- [15] J. Chen, H. Huang, S. Tian, and Y. Qu. Feature selection for text classification with Naive Bayes. *Expert Systems with Applications*, 36(3):5432–5435, 2009.

- [16] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462 – 467, 1968.
- [17] M. Claesen and B. De Moor. Hyperparameter search in machine learning. *arXiv preprint arXiv:1502.02127*, 2015.
- [18] A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 215–223, 2011.
- [19] A. Coates and A. Y. Ng. The importance of encoding versus training with sparse coding and vector quantization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 921–928, 2011.
- [20] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.
- [21] E. Comak and A. Arslan. A support vector machine using the lazy learning approach for multi-class classification. *Journal of Medical Engineering and Technology*, 30(2):73–7, 2006.
- [22] S. R. Dunham. Virtual storage and block level direct access of secondary storage for recovery of backup data, July 31 2001. US Patent 6,269,431.
- [23] E. Frank, M. Hall, and B. Pfahringer. Locally weighted Naive Bayes. In *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 249–256, 2002.

- [24] E. Frank, M. Hall, and B. Pfahringer. Locally weighted Naive Bayes. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 249–256, 2012.
- [25] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997.
- [26] M. Hall. A decision tree-based attribute weighting filter for Naive Bayes. *Knowledge-Based Systems*, 20(2):120–126, 2007.
- [27] M. A. Hall. Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*, pages 359–366, 2000.
- [28] D. Hand and R. Till. A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. *Machine Learning*, 45:171–186, 2001.
- [29] C. Hare. Simple Network Management Protocol (SNMP), 2011.
- [30] D. Harrington, R. Presuhn, and B. Wijnen. RFC3411: An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks, 2002.
- [31] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2nd edition, 2009.
- [32] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

- [33] M. Helmstaedter, K. L. Briggman, S. C. Turaga, V. Jain, H. S. Seung, and W. Denk. Connectomic reconstruction of the inner plexiform layer in the mouse retina. *Nature*, 500(7461):168–174, 2013.
- [34] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29, 2012.
- [35] J.-H. Hong, J.-K. Min, U.-K. Cho, and S.-B. Cho. Fingerprint classification using one-vs-all support vector machines dynamically ordered with Naive Bayes classifiers. *Pattern Recognition*, 41(2):662 – 671, 2008.
- [36] H. Hoos and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *International Conference on Machine Learning*, pages 754–762, 2014.
- [37] F. Hutter, J. Lücke, and L. Schmidt-Thieme. Beyond manual tuning of hyperparameters. *KI-Künstliche Intelligenz*, 29(4):329–337, 2015.
- [38] L. Jiang, Z. Cai, D. Wang, and H. Zhang. Improving Tree-augmented Naive Bayes for class probability estimation. *Knowledge-Based Systems*, 26:239 – 245, 2012.
- [39] L. Jiang, Z. Cai, H. Zhang, and D. Wang. Not so greedy: Randomly Selected Naive Bayes. *Expert Systems with Applications*, 39(12):11022–11028, 2012.
- [40] L. Jiang and Y. Guo. Learning lazy Naive Bayesian classifiers for ranking. In *Proceedings of the 24th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 412–416, 2005.

- [41] L. Jiang, D. Wang, Z. Cai, and X. Yan. Survey of improving Naive Bayes for classification. In *Proceedings of the International Conference on Advanced Data Mining and Applications (ADMA)*, pages 134–145, 2007.
- [42] L. Jiang and H. Zhang. Learning instance greedily cloning Naive Bayes for ranking. In *Proceedings of Fifth IEEE International Conference on Data Mining (ICDM)*, pages 202–209, 2005.
- [43] L. Jiang and H. Zhang. Lazy averaged one-dependence estimators. In *Proceedings of the 19th Conference of the Canadian Society for Computational Studies of Intelligence, Canadian AI (CAI)*, pages 515–525, 2006.
- [44] L. Jiang, H. Zhang, and Z. Cai. A Novel Bayes Model: Hidden Naive Bayes. *Knowledge and Data Engineering IEEE Transactions on*, 21(10):1361–1371, 2009.
- [45] L. Jiang, H. Zhang, Z. Cai, and D. Wang. Weighted average of one-dependence estimators. *Journal of Experimental and Theoretical Artificial Intelligence*, 24(2):219–230, 2012.
- [46] L. Jiang, H. Zhang, and J. Su. Instance Cloning Local Naive Bayes. *Lecture Notes in Computer Science*, 3501:280–291, 2005.
- [47] D. Kenneth. Genetic algorithms: a 30 year perspective. *Perspectives on Adaptation in Natural and Artificial Systems*, 11:11–32, 2005.
- [48] E. Keogh and M. Pazzani. Learning augmented Bayesian classifiers: A comparison of distribution-based and classification-based approaches. In *Proceedings*

- of the International Workshop on Artificial Intelligence and Statistics (AISTATS)*, pages 225–230, 1999.
- [49] H. Kim and S.-S. Chen. Associative Naive Bayes classifier: Automated linking of gene ontology to medline documents. *Pattern Recognition*, 42(9):1777 – 1785, 2009.
- [50] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.
- [51] P. Langley and S. Sage. Induction of selective Bayesian classifiers. *Uncertainty Proceedings*, pages 399–406, 2013.
- [52] Z. Li, Y. Yang, J. Liu, X. Zhou, and H. Lu. Unsupervised feature selection using nonnegative spectral analysis. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1026–1032, 2012.
- [53] W. Liu and I. W. Tsang. Making decision trees feasible in ultra-high feature and label dimensions. *The Journal of Machine Learning Research*, 18(1):2814–2849, 2017.
- [54] W. Liu, I. W. Tsang, and K.-R. Müller. An easy-to-hard learning paradigm for multiple classes and multiple labels. *The Journal of Machine Learning Research*, 18(1):3300–3337, 2017.
- [55] W. Liu, D. Xu, I. W. Tsang, and W. Zhang. Metric learning for multi-output tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(2):408–422, 2019.

- [56] B. Luitel and G. K. Venayagamoorthy. Differential evolution particle swarm optimization for digital filter design. *Evolutionary Computation*, pages 3954–3961, 2008.
- [57] H. Mania, A. Guy, and B. Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.
- [58] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.
- [59] G. Memik, M. T. Kandemir, and A. Choudhary. Design and evaluation of a smart disk cluster for DSS commercial workloads. *Journal of Parallel and Distributed Computing*, 61(11):1633–1664, 2001.
- [60] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černocký. Strategies for training large scale neural network language models. In *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*, pages 196–201. IEEE, 2011.
- [61] T. Oetiker. Monitoring your IT gear: the MRTG story. *IT Professional*, 3(6):44–48, 2001.
- [62] T. Oetiker and D. Rand. Mrtg: The multi router traffic grapher. In *LISA*, volume 98, pages 141–148, 1998.
- [63] N. Pinto, D. Doukhan, J. J. DiCarlo, and D. D. Cox. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS Computational Biology*, 5(11):e1000579, 2009.

- [64] F. J. Pontes, G. Amorim, P. P. Balestrassi, A. Paiva, and J. R. Ferreira. Design of experiments and focused grid search for neural network parameter optimization. *Neurocomputing*, 186:22–34, 2016.
- [65] C. Ratanamahatana and D. Gunopulos. Scaling up the Naive Bayesian Classifier: Using Decision Trees for Feature Selection. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pages 399–406, 2002.
- [66] J. Riudavets, K. F. Navarro, E. Lawrence, R. Steele, and M. Messina. Multi router traffic grapher (MRTG) for body area network (BAN) surveillance. *WSEAS Transactions on Computers*, 3(6):1856–1862, 2004.
- [67] M. Robnik-Šikonja and I. Kononenko. Theoretical and Empirical Analysis of ReliefF and RReliefF. *Machine Learning*, 53(1-2):23–69, 2003.
- [68] U. K. Sikdar, A. Ekbal, S. Saha, O. Uryupina, and M. Poesio. Differential evolution-based feature selection technique for anaphora resolution. *Soft Computing*, 19(8):2149–2161, 2014.
- [69] R. Storn and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.
- [70] I. Syarif, A. Prugel-Bennett, and G. Wills. SVM parameter optimization using grid search and genetic algorithm to improve classification performance. *Telkomnika*, 14(4):1502, 2016.

- [71] F. Van den Bergh and A. P. Engelbrecht. A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):225–239, 2004.
- [72] R. van der Linden, D. Halls, S. Waterhouse, and P. Benoit. Systems and methods for establishing a cloud bridge between virtual storage resources, 2013. US Patent 8,578,076.
- [73] G. I. Webb, J. R. Boughton, F. Zheng, K. M. Ting, and H. Salem. Learning by extrapolation from marginal to full-multivariate probability distributions: decreasingly Naive Bayesian classification. *Machine Learning*, 86(2):233–272, 2012.
- [74] I. G. Webb, R. J. Boughton, and Z. Wang. Not So Naive Bayes: Aggregating One-Dependence Estimators. *Machine Learning*, 58(1):5–24, 2005.
- [75] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.
- [76] T.-T. Wong. A hybrid discretization method for Naive Bayesian classifiers. *Pattern Recognition*, 45(6):2321 – 2325, 2012.
- [77] T.-T. Wong and L.-H. Chang. Individual attribute prior setting methods for Naive Bayesian classifiers. *Pattern Recognition*, 44(5):1041 – 1047, 2011.
- [78] J. Wu and Z. Cai. A Naive Bayes probability estimation model based on self-adaptive differential evolution. *Journal of Intelligent Information Systems*, 42(3):671–694, 2014.
- [79] J. Wu, Z. Cai, S. Pan, X. Zhu, and C. Zhang. Attribute weighting: How and when does it work for Bayesian Network Classification. In *Proceedings of the*

- International Joint Conference on Neural Networks (IJCNN)*, pages 4076–4083, 2014.
- [80] J. Wu, Z.-h. Cai, and S. Ao. Hybrid Dynamic K-nearest-neighbour and Distance and Attribute Weighted Method for Classification. *International Journal Computer Applications in Technology*, 43(4):378–384, 2012.
- [81] J. Wu, S. Pan, Z. Cai, X. Zhu, and C. Zhang. Dual instance and attribute weighting for Naive Bayes classification. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 1675–1679, 2014.
- [82] J. Wu, S. Pan, X. Zhu, Z. Cai, P. Zhang, and C. Zhang. Self-adaptive attribute weighting for Naive Bayes classification. *Expert Systems with Applications*, 42(3):1487–1502, 2015.
- [83] J. Wu, S. Pan, X. Zhu, P. Zhang, and C. Zhang. Sode: Self-adaptive one-dependence estimators for classification. *Pattern Recognition*, 51:358–377, 2016.
- [84] N. A. Zaidi, J. Cerquides, M. J. Carman, and G. I. Webb. Alleviating Naive Bayes attribute independence assumption by attribute weighting. *Journal of Machine Learning Research*, 14(1):1947–1988, 2013.
- [85] H. Zhang and S. Sheng. Learning weighted Naive Bayes with accurate ranking. In *Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM)*, pages 567–570, 2004.
- [86] H. Zhang and J. Su. Naive Bayesian Classifiers for Ranking. In *Proceedings of the 15th European Conference on Machine Learning (ECML)*, pages 501–512, 2004.

- [87] J. Zhang and A. C. Sanderson. JADE: Adaptive Differential Evolution With Optional External Archive. *IEEE Transactions on Evolutionary Computation*, 13(5):945–958, 2009.
- [88] M. L. Zhang and Z. H. Zhou. ML-KNN : A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048, 2007.
- [89] Y. Zhang, J. Wu, Z. Cai, P. Zhang, and L. Chen. Memetic Extreme Learning Machine. *Pattern Recognition*, 58:135–148, 2016.
- [90] Z. Zheng and G. I. Webb. Lazy Learning of Bayesian Rules. *Machine Learning*, 41(1):53–84, 2010.
- [91] W. Zong, G.-B. Huang, and Y. Chen. Weighted extreme learning machine for imbalance learning. *Neurocomputing*, 101:229–242, 2013.