Decision trees, random vector functional link neural network and their ensembles

Katuwal, Rakesh Kumar

2019

Katuwal, R. K. (2019). Decision trees, random vector functional link neural network and their ensembles. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/136779>

<https://doi.org/10.32657/10356/136779>

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License (CC BY-NC 4.0).

Downloaded on 26 Apr 2023 20:19:51 SGT

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

**DECISION TREES, RANDOM VECTOR FUNCTIONAL
LINK NEURAL NETWORK AND THEIR ENSEMBLES**

RAKESH KUMAR KATUWAL
SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING
2019

DECISION TREES, RANDOM VECTOR FUNCTIONAL LINK NEURAL NETWORK AND THEIR ENSEMBLES

RAKESH KUMAR KATUWAL

School of Electrical and Electronic Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirement for the degree of
Doctor of Philosophy

2019

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

09 Dec 2019

Date

RAKESH KUMAR
KATUWAL

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

09 Dec 2019

Date

A handwritten signature in blue ink, appearing to read 'P.N. Suganthan', written over a horizontal line.

P.N. SUGANTHAN

Authorship Attribution Statement

This thesis contains material from 4 paper(s) published in the following peer-reviewed journal(s)/from papers accepted at conferences in which I am listed as an author.

Chapter 3 is published as Rakesh Katuwal, P.N. Suganthan, and Le Zhang. An ensemble of decision trees with random vector functional link networks for multi-class classification. *Applied Soft Computing*, 70:1146-1153, 2018.

The contributions of the co-authors are as follows:

- Assoc. Prof. P.N. Suganthan provided the initial direction and edited the manuscript drafts.
- I wrote the codes for the experiments and analyzed the results. I also prepared the manuscript drafts.
- Le Zhang assisted in writing the codes and proofreading the manuscript.

Chapter 4 Section 4.3 is published as Rakesh Katuwal and P.N. Suganthan. Enhancing Multi-Class Classification of Random Forest using Random Vector Functional Neural Network and Oblique Decision Surfaces. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1-8, July 2018.

The contributions of the co-authors are as follows:

- Assoc. Prof. P.N. Suganthan provided the initial direction and proofread the manuscript.
- I prepared the codes and conducted all the experiments. I also wrote the drafts of the manuscript.

Chapter 4 Section 4.4 is published as Rakesh Katuwal, P.N. Suganthan, and Le Zhang. Heterogeneous oblique random forest. *Pattern Recognition*, 99, 107078.

The contributions of the co-authors are as follows:

- I conducted the experiments, analyzed results and wrote the manuscript drafts.
- Assoc. Prof. P.N. Suganthan provided insights and edited the manuscript.
- Le Zhang assisted in proofreading the manuscript.

Chapter 5 is published as Rakesh Katuwal and P.N. Suganthan. Stacked autoencoder based deep random vector functional link neural network for classification. *Applied Soft Computing*, 105854. doi:10.1016/j.asoc.2019.105854

The contributions of the co-authors are as follows:

- I wrote the codes for the experiments and analyzed the results. I also prepared the manuscript drafts.
- Assoc. Prof. P.N. Suganthan provided the initial direction and edited the manuscript drafts.

09 Dec 2019

Date

RAKESH KUMAR
KATUWAL

Acknowledgement

I am greatly indebted to a number of people without whom this thesis might not have been written.

I would like to dedicate this thesis to my mother and father. They have always been besides me through thick and thin. Research can be difficult at times, one may lose hope and lack focus. Also leaving home and country, barely visiting parents and siblings, and missing festive gatherings or important family events for several years are bound to take a toll. I would be lying if I said these things didn't get me during my PhD journey. I wouldn't have been able to survive this journey unscathed without my mother and father's support. My father once told me that life is then a misery, when despite doing everything you can, you are still poor and unable to pay for your food, but not when you are working hard towards fulfilling your ambitions in life. At times when I got PhD blues, recalling him saying that is what got me motivated and helped me focus towards my goals.

I offer my gratitude and appreciation to my supervisor, Assoc. Prof. P.N. Suganthan for his guidance, support and motivation. His constant mentoring, frequent research discussions helped me to keep pushing forward in making a contribution to the world. No words are enough to thank him for instilling research inquisitiveness into me.

I would like to express my gratitude to Dr. Le Zhang for his help and support especially during the early years of my PhD. My friends, Sumit, Ujjal, Dipu, Milan, Niroj, Shilpa and Pradeep, who have been my family in Singapore, deserve recognition in this thesis. This PhD journey would have been a tough ride without these people in my life. Many thanks to Mr. Yuen Sien Huan, and other staffs at Internet of Things Lab, for providing me work-space resources for my PhD research.

Lastly, my sincere thanks to SINGA scholarship program committee for providing me with financial assistance for my PhD and Nanyang Technological University staffs for their immense help during my study.

Abstract

Decision Tree (DT) and Neural Network (NN) are two popular machine learning algorithms. A decision tree consists of internal nodes and leaf nodes. Each internal node is associated with a test function while each leaf node represents a class label or contains probability distribution. A decision tree is effectively a sequence of IF-THEN rules which is intuitive to humans and simple to implement. Meanwhile, a neural network is configured to mimic the functionality of the animal brain using highly interconnected system of computational units called neurons. The neurons are basically realized in the form of non-linear activation functions which process numeric inputs and output. The neurons are interconnected by transmission channels called synapse which is represented by the weights of the network connection. The weights are typically learned using back-propagation (BP) algorithm. However, in randomization based neural networks such as Random Vector Functional Link (RVFL) neural network, some parts (parameters) of the network are randomly and appropriately generated and fixed while optimizing the rest using a closed form solution or an iterative procedure.

Ensemble of classifiers, also known as multiple classifier system, is a widely researched and frequently applied approach in machine learning. A multitude of studies corroborate that the combination of many classifiers leads to an improved performance compared to a single individual classifier. Decision trees and neural networks are typically used in an ensemble framework to obtain better performance.

Random Forest (RaF), an ensemble of decision trees, is extensively used to win several machine learning competitions and is considered one of the best algorithm. On the other hand, the popularity of Random Vector Functional Link (RVFL) neural network is also escalating because of its impressive performance in several diverse domains and its faster training time compared to conventional BP-trained NN. The first part of this thesis is based on random forest (including a hybrid ensemble classifier based on RaF and standard shallow RVFL) while the second part is based on multi-layer or deep random vector functional link neural network.

Decision trees in random forest usually employ binary splits at each node. The binary splits result in very deep trees. We utilize RVFL at the root node of the DTs to create multi-way splits. The proposed method provides a rich insight into the data by grouping the confusing or hard to classify samples and thus, provides an opportunity to employ fine-grained classification rule over the data. Also, by

using multi-way splits, we obtain shallow trees that agree with Ockham's razor principle of building smaller trees. Extensive experiments on several multi-class datasets show the efficacy of our method.

We also present our work in oblique decision trees. In oblique decision trees, an oblique (linear) hyperplane is employed instead of an axis-parallel (orthogonal) hyperplane at each non-leaf node to partition the data. Trees with such hyperplanes can better exploit the geometric structure in the data to increase the accuracy of the trees and reduce the depth. While the standard decision trees perform an exhaustive search for the best axis-parallel hyperplane, such an exhaustive search for the best oblique hyperplane is computationally expensive. Therefore, the present realizations of oblique decision trees do not evaluate many promising oblique splits to select the best. We present several oblique random forest variants that search for the best oblique hyperplane from a pool of homogeneous hyperplanes by employing one-vs-all approach. Similarly, we also propose a random forest of heterogeneous oblique decision trees that employ oblique splits generated via diverse linear classifiers at each non-leaf node on some top ranked partitions. In extensive experiments on 121 UCI machine learning datasets, 3 (out of 7) of our proposed oblique random forests take the top 3 ranks by outperforming other random forests and hundreds of other classifiers.

Deep learning, also known as representational learning, has sparked a surging interest in neural networks in recent years. Deep NN consists of several hidden layers stacked on top of each other wherein each hidden layer builds an internal representation of the data. In this thesis, we propose several deep random vector functional link neural networks variants. We also present an implicit ensemble of deep RVFL neural network which can be regarded as a marriage of ensemble and deep learning, that is simple and straight-forward to implement.

Table of Contents

Acknowledgement	i
Abstract	iii
List of Figures	xi
List of Tables	xiv
Abbreviations	xv
1 Introduction	1
1.1 Research Background	2
1.2 Motivations and Contributions	3
1.3 Overview of the Thesis	7
2 Literature Review	9
2.1 Ensemble Methods	9
2.1.1 Diversity	10
2.1.2 Bias-Variance-Covariance Analysis	11
2.2 Decision Trees and their ensemble	12
2.2.1 Random Forest	15
2.2.2 Oblique Random Forest	16
2.3 Artificial Neural Network	19
2.3.1 Single Layer Feedforward Neural Network	20
2.3.1.1 Random Vector Functional Link Network	21
2.3.1.2 Extreme Learning Machine	23
2.3.1.3 Kernel Extreme Learning Machine	24
2.3.1.4 Autoencoder	24
2.3.1.5 Denoising Autoencoder	25
2.3.1.6 Sparse pre-trained RVFL	26
2.3.2 Multi-layer (Deep) Neural Network	27
2.3.2.1 Randomization based Autoencoder	28
2.3.2.2 Multi-layer ELM	29
2.3.2.3 Hierarchical ELM	29
2.3.2.4 Multi-layer kernel ELM	29
2.3.2.5 Neural-response-based ELM (NR-ELM)	29

2.3.3	Neural Network Ensemble	29
2.3.3.1	Dropout	31
2.3.3.2	DropConnect	31
2.3.3.3	Stochastic Depth	32
2.3.3.4	Swapout	32
2.3.3.5	Snapshot Ensemble	32
3	An Ensemble of Decision Trees and RVFL	33
3.1	Introduction	33
3.2	Proposed Ensemble Method	35
3.3	Experimental setup	37
3.3.1	RVFL Configuration	37
3.3.2	Decision Tree Configuration	38
3.4	Results and Discussion	38
3.4.1	Comparison of RaF, obRaF, RaFL and obRaFL	41
3.4.2	On the effect of randomizing lambda and the number of hidden neurons	44
3.4.3	Computational Complexity	45
3.5	Conclusion	46
4	Oblique Random Forest	49
4.1	Introduction	49
4.2	ObRaF with linear and non-linear hyperplanes	52
4.3	ObRaF based on one-vs-all approach	53
4.4	Heterogeneous oblique random forest	55
4.4.1	Generating several partitions and their ranking	55
4.4.1.1	Ideal Gini score	57
4.4.1.2	Cluster separability	57
4.4.1.3	Employing Exhaustive search or Genetic Algorithm	59
4.4.2	Employing several linear classifiers at each node	62
4.5	Experiments	66
4.5.1	Machine Learning Datasets	66
4.5.2	Experimental Settings	66
4.5.2.1	Configuration of Heterogeneous Oblique Decision Tree	66
4.5.3	Performance Comparison	67
4.5.4	Computational Complexity Analysis	75
4.5.5	Overall Comparisons Among 190 Classifiers	75
4.5.5.1	Statistical Comparison	79
4.6	Conclusion	80
5	Stacked Autoencoder based Deep RVFL	83
5.1	Introduction	83
5.2	RVFL based deep neural network	85
5.2.1	Deep RVFL with direct links	85
5.2.2	Deep RVFL with dense direct links	87

5.2.3	Deep RVFL with denoising autoencoders	88
5.3	Experiment and Analysis	88
5.3.1	Performance Comparison	89
5.3.2	Complexity Analysis	91
5.3.3	Performance Comparison with state-of-the-art Deep Neural Networks	95
5.4	Conclusion	96
6	RVFL based Ensemble Deep Learning	99
6.1	Introduction	99
6.2	Deep RVFL for representational learning	100
6.2.1	Deep Random Vector Functional Link Network	100
6.2.2	Ensemble Deep Random Vector Functional Link Network . .	102
6.3	Experiments	103
6.3.1	Datasets	103
6.3.2	Compared Methods	104
6.3.3	Experimental Settings	105
6.3.4	Performance Comparison and Analysis	105
6.4	Conclusion	119
7	Summary	121
8	Conclusion and Future Work	125
8.1	Conclusion	125
8.2	Future Work	127
	Author's Publications	129
	Bibliography	131
	Appendix	143

List of Figures

2.1	Framework of an ensemble method. The three major steps in ensemble construction include: Data Generation, Base Prediction and Aggregated Prediction. The solid lines indicate the ensemble framework for Bagging type ensembles. The dashed line indicates the framework for Boosting type ensembles.	11
2.2	Classification boundaries generated by axis-parallel and oblique decision tree in a toy example for binary classification problem. Axis-parallel or staircase decision boundaries may fail to approximate the optimal decision boundary. However, oblique trees with linear hyperplanes can better approximate the complex decision boundary. Best viewed in color.	14
2.3	Architecture of SLFN.	21
2.4	Framework of RVFL (1994) and ELM (2004) networks. The structure of RVFL and ELM differ in the presence (absence) of direct links and bias term (not shown in the figure). The red lines represent the direct links (original features) from the input to the output layer. The weights for the blue lines are randomly generated from a certain range and kept fixed. Only the output weights (associated with red and black lines) need to be computed. Best viewed in color.	23
2.5	Architecture of H-ELM. It consists of feature encoding using multi-layer network built with randomization based AE and ELM classifier for final decision making.	30
3.1	A base classifier of our proposed ensemble method	37
3.2	Accuracy of RaFL and obRaFL with different values of C	45
3.3	Accuracy of RaFL and obRaFL with different number of hidden neurons	46
4.1	Ranking of different partitions based on their ideal gini values g_l and dunn index DI. LDA is used at each partition to generate a decision boundary. g_l refers to the ideal gini scores computed beforehand to rank the partitions while g refers to the actual gini scores after generating the decision boundary. The numbers within the curly braces represent the binary partitions. db means decision boundary. Best viewed in color.	61
4.2	Flowchart explaining GAOREXHSEARCH function of Algorithm 2.	62

4.3	For the GA based partitioning, we create the initial population by randomly assigning the classes inside the rectangular box (represents the 50% classes) mid-way between the two furthest classes ext1 and ext2 obtained using Bhattacharyya distance to either of the partitions.	62
4.4	Decision boundaries generated by different linear classifiers and their associated g . Best viewed in color.	63
4.5	Flowchart depicting our proposed method at each node	64
4.6	Statistical comparison of classifiers against each other based on Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.05$) are connected.	69
4.7	κ -Error diagrams of “optical” and “soybean” datasets with all forest based methods introduced in this chapter. x-axis = κ (diversity between two classifiers), y-axis = E_{ij} (average error of the pair of classifiers).	73
4.8	Centroids of the κ -Error clouds of the “optical” and “soybean” datasets.	74
4.9	Decision boundaries for the classification of the artificial spirals dataset	76
5.1	Architecture of sdRVFL(d). For the feature encoding part, randomization based AE is used followed by RVFL classifier.	86
5.2	Architecture of sdRVFL(dense). It is built hierarchically with ELM-AE based feature encoding followed by RVFL classifier in addition that each layer takes all preceding features as input.	87
5.3	Comparison of classifiers against each other with the Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.05$) are connected. The Nemenyi test is performed separately for each type of regularization based neural network.	92
6.1	Framework of a dRVFL network. It consists of several hidden layers stacked on top of each other whose parameters (weights and biases between the hidden layers) are randomly generated and kept fixed during the training. Only the output weights β_d need to be computed as in the shallow RVFL network.	102
6.2	Framework of ensemble deep RVFL network (edRVFL). It differs from dRVFL in that the computation of final output weight β_d is decomposed into several small β_{ed} . Specifically, each small β_{ed} is independently computed (treated as independent model) and the final output is obtained by using either majority voting or averaging of the models. Each higher level model is fed with original input data and the non-linearly transformed features from the preceding model. $\mathbf{O}_1, \dots, \mathbf{O}_L$ represents the output of each model.	104
6.3	Statistical comparison of classifiers against each other based on Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.05$) are connected.	108

6.4	Comparison of dRVFL and edRVFL in terms of accuracy (%) w.r.t different number of hidden nodes.	109
6.5	Training and testing times comparison of dRVFL and edRVFL w.r.t different number of hidden nodes. For the same number of hidden nodes, the training time of edRVFL is in average 3 times more than that of dRVFL.	110
6.6	Comparison of dRVFL and RVFL in terms of accuracy (%) with the same number of hidden nodes.	111
6.7	Comparison of edRVLF (implicit ensemble) and TedRVFL (true ensemble) in terms of accuracy (%).	112
6.8	Training and testing times comparison of edRVLF (implicit ensemble) and TedRVFL (true ensemble).	113
6.9	Performance variation of the proposed dRVFL (first row) and edRVFL(second row) methods in terms of accuracy (%) for fixed H. Different parameter combinations may result in different performance.	114
6.10	Performance variation of the proposed dRVFL (first row) and edRVFL(second row) methods in terms of accuracy (%) for fixed L. Different parameter combinations may result in different performance.	115
7.1	Statistical comparison of classifiers against each other based on Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.05$) are connected.	122

List of Tables

2.1	Random Forest	15
3.1	Proposed Hybrid Ensemble	36
3.2	Classification accuracy (%) of Different Methods	39
3.3	Average rank values based on classification accuracy of each method.	40
3.4	Average rank values of 4 methods in data sets with $N > 500$	42
3.5	Significance of the difference between each pairs of algorithms	43
3.6	Average testing accuracy (%) of different methods	44
3.7	Average number of nodes in a base classifier of each ensemble.	47
3.8	Average training time (in seconds) for each ensemble method	47
4.1	Accuracy (%) of Different Random Forest Based Methods	67
4.2	Average Friedman Rank Based on Classification Accuracy of Each Method on 121 UCI Datasets	69
4.3	Comparison in terms of % error (first 5 datasets) and % accu- racy(last 10 datasets)	74
4.4	Computational complexity of the linear classifiers	77
4.5	Training Time Comparison Relative to RaF	77
4.6	Average Tree Depth	77
4.7	Accuracy(%) of obRaF(H)-d in all 121 datasets	78
4.8	Overall comparison of 20 top ranked classifiers out of 190 methods in 121 datasets	79
4.9	Comparison of 20 Top Ranked Classifiers Out of 184 Methods	80
4.10	Statistical comparison of obRaF(H) and each of the other top 20 ranked classifiers	81
4.11	Wilcoxon signed-rank test based statistical comparison of the 20 top ranked classifiers out of 190 Methods	82
5.1	Accuracy (%) of deep neural networks with l_1 -regularization and ML-KELM	89
5.2	Accuracy (%) of deep neural networks with l_2 -regularization	90
5.3	Accuracy (%) of deep neural networks with elastic net regularization	90
5.4	Jointly calculated average Friedman rank based on classification accuracy of each method	93
5.5	Statistical Comparison of the Algorithms	94

5.6	Comparison of Hidden Nodes and Training Time of each algorithm in the bank dataset	94
5.7	Comparison of Hidden Nodes and Training Time of each algorithm in the glass dataset	95
5.8	Comparison of classification accuracy in the income prediction dataset.	96
5.9	Hyperparameters considered for the HTRU2 dataset [1].	96
5.10	Comparison of neural networks in the HTRU2 dataset in terms of AUC.	97
6.1	Overview of the datasets.	105
6.2	Accuracy (%) of the standard RVFL based methods on all the datasets	107
6.3	Comparison between SP-RVFL based methods in terms of accuracy (%)	116
6.4	Average Friedman rank based on classification accuracy of each method	117
6.5	Hyperparameters considered for deep RVFL networks	118
6.6	Hyperparameters considered for other FNNs [1]	118
6.7	Comparison of FNN methods	118
7.1	Overview of the datasets.	122
7.2	Comparison in terms of accuracy (%)	123
A1	Details of 121 UCI Datasets Used in this thesis	143

Abbreviations

AE	Autoencoder
ANN	Artificial Neural Network
CNNs	Convolutional Neural Networks
DAE	Denoising Autoencoder
DNNs	Deep Neural Networks
dRVFL	deep RVFL network
DT	Decision Tree
edRVFL	ensemble deep RVFL network
ELM	Extreme Learning Machine
GBTs	Gradient Boosted Trees
H-ELM	Hierarchical ELM
KELM	Kernel Extreme Learning Machine
KRR	Kernel Ridge Regression
LDA	Linear Discriminant Analysis
LR	Logistic Regression
ML-ELM	Multi-layer ELM
ML-KELM	Multi-layer kernel ELM
MPSVM	Multisurface Proximal Support Vector Machine
NN	Neural Network
NR-ELM	Neural-response-based ELM
obRaF	Oblique Random Forest
RaF	Random Forest
RNN	Recurrent Neural Networks
RR	Ridge Regression

RVFL	Random Vector Functional Link
sdRVF(dense)	Deep RVFL with dense direct links
sdRVFL(d)	Deep RVFL with direct links
sdRVFL-D(dense)	Deep RVFL with denoising autoencoders
SLFN	Single-hidden Layer Feedforward Neural Network
SP-RVFL	Sparse pre-trained RVFL
SVM	Support Vector Machine

Notation & Convention

Matrices	upper case boldface letters (e.g. $\mathbf{X}$)
Vectors	lower case boldface letters (e.g. $\mathbf{x}$)
Scalars	lower case letters (e.g. x)
Sets	calligraphic letters (e.g. $\mathcal{S}$)
Standard sets	upper case blackboard bold characters (e.g. $\mathbb{R}$)

Given a matrix $\mathbf{X}$ its

i^{th} row	$\mathbf{x}_{i:}$
i^{th} column	$\mathbf{x}_{:i}$
$(i, j)^{\text{th}}$ element	x_{ij}

Given a vector $\mathbf{x}$ its

i^{th} element	x_i
-------------------------	-------

Given a set $\mathcal{S}$ its

Cardinal number	$ \mathcal{S} $
-----------------	-----------------

Chapter 1

Introduction

In machine learning, a classification problem aims at identifying a discrete category of a new observation by learning on a training set of data containing observations with known categories. A classification problem can be mathematically formulated as:

$$y = f(\mathbf{x}, \theta) \in \mathcal{Y} \tag{1.1}$$

where $\mathbf{x}$ is a new observation, $f(\cdot)$ is the trained classification function or simply a ‘classifier’, θ is a parameter of the classifier, y is the category assigned by the classifier to $\mathbf{x}$, also known as prediction, and $\mathcal{Y}$ is the set of categories or class labels. Based on the cardinality of y , the classification problem is subdivided into two categories: binary and multi-class classification. In binary classification, only two classes are involved whereas in multi-class classification, there exists more than two classes. There are hundreds of classifiers [2] arising from different fields within computer science and mathematics that could be used to solve these classification problems.

In this thesis, we will discuss two popular classifiers: Decision Trees (DT) and Neural Network (NN). We will first give a brief overview of these classifiers in the following section, followed by the motivations and our major contributions. Finally, we will end this chapter with an overview of the contents of this thesis.

1.1 Research Background

“Many heads are better than one.” “Many are smarter than the Few.” An avalanche of such dicta, accentuating the power and wisdom of crowds or experts, are very popular in our society. Several studies based on sociological, psychological and other aspects of human life corroborate these statements [3, 4]. Such committee of experts are also extensively used in machine learning and is known as an ‘ensemble’. Understandably, an ensemble of classifiers, or multiple classifier systems employ several classifiers instead of a single classifier to improve the predictive performance. An ensemble is typically obtained by leveraging the ‘perturb and combine’ strategy. Specifically, an ensemble is obtained by perturbing the training set or injecting some randomness in each base (member) classifier and aggregating the outputs of these classifiers. Decision trees and neural networks are widely used as the base classifiers in the ensemble learning framework. Both decision trees and neural network are unstable classifiers whose performance greatly vary even when there is a small perturbation in training set or in construction [5]. Thus, an ensemble of decision trees or neural networks are much more robust and accurate than an individual tree or a neural network.

Decision trees are among the most widely used machine learning algorithm in practice. They enjoy several unique advantages. They are able to model nonlinear data well while being highly interpretable. The prediction made by the tree is a path that traverses from the root to a leaf consisting of a sequence of decisions, that can be understood as a sequence of IF-THEN rules, which is intuitive to humans. Thus, decision trees are widely popular in applications such as medical diagnosis or legal analysis where interpretability is important [6]. However, decision trees are not confined within such domains only. They are widely used to win Kaggle, or KDD Cup competitions [7]. It is a common practice to use an ensemble of trees such as Random Forest (RaF) [8], Gradient Boosted Trees (GBTs) [9] to boost the performance of trees at the expense of interpretability. In fact, recent studies that evaluated hundreds of classifiers posit Random Forest as the top ranked classifier [2, 10]. Depending on the type of question or rule at each non-leaf nodes, there are basically two types of decision trees: axis-aligned or orthogonal, and oblique or multivariate decision trees. In axis-parallel decision trees, the question is of the type “ $x_i < b_j$ ” (is feature i less than the threshold b_j) or “ $\mathbf{w}^T \mathbf{x} < b_j$ ” for oblique trees. In this thesis, we refer to an ensemble of axis-parallel decision trees as standard Random Forest (RaF), and an ensemble of oblique decision trees as

Oblique Random Forest (obRaF). Unless otherwise stated, Random Forest refers to both RaF and obRaF.

Neural network is another popular machine learning algorithm. A NN is configured to mimic the functionality of the animal brain using highly interconnected system of computational units called neurons. A typical neural network consists of an input layer, one or several hidden layers and a output layer. The predictive capability of a neural network derives from the hidden layer. The hidden layer consists of several hidden neurons or nodes that perform non-linear mappings between the input and the output. The hidden neurons are interconnected by transmission channels called synapse which is represented by the weights of the network connection. A typical NN is trained by minimizing a loss function. Specifically, the gradient of the loss function is computed with respect to all neural network parameters and the error is back-propagated to update the parameters. However, in randomization based neural networks such as Random Vector Functional Link (RVFL) [11] neural network, Extreme Learning Machine (ELM) [12] either closed form solutions or iterative procedures are used to train the network. Such networks are faster to train and have demonstrated good learning performance in several diverse domains.

Deep Learning, also known as representational learning, has sparked a surging interest in neural networks amongst the machine learning enthusiasts. The superiority of a deep learning model emanates from its potential ability to extract meaningful representations at different levels of the hierarchical model while disentangling a complex task into several simpler ones [13]. Deep neural networks typically consist of multiple hidden layers stacked together wherein each hidden layer builds an internal representation of the data [13, 14]. Deep Learning especially via Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNN) have revolutionized the vision and natural language processing domains.

1.2 Motivations and Contributions

Even though Deep Neural Networks (DNNs) have been extremely successful on various tasks such as images, speech, and text, there are many tasks in which DNNs under-perform compared to other models such as Random Forest [1, 7, 15]. Extensive evaluations on much richer datasets from the UCI Machine Learning Repository [16] with varying characteristics different from standard vision and

speech datasets also evince Random Forest as a top ranked classifier out of hundreds of classifiers [2, 10]. Thus, in the first part of the thesis, specifically in Chapter 3 and Chapter 4, we present techniques to further improve the performance of Random Forest.

Each non-leaf node of a decision tree partitions the input space into two, sending the training data into two child nodes. Generally, binary splits are popular with decision trees with very few researches on multi-way splits. Multi-way (Multi-branch) splits in decision trees have previously been studied in [17, 18]. However, such multi-way splits are cumbersome to determine and do not effectively improve the performance of decision trees. In Chapter 3, we employ a simple, fast RVFL neural network to perform multi-way partition at the root node of the decision tree. The RVFL partitions the training dataset into several subsets where confusing or difficult samples are grouped in the same subset. Such technique allows us to employ finer classification rules while focusing on confusing or difficult-to-classify samples. We demonstrate the effectiveness of this technique using several multi-class datasets.

A candidate hyperplane at each internal node of the decision tree is usually rated based on an impurity criterion. The impurity criterion gives higher rating to a hyperplane which results in higher purity of child nodes. However, impurity criterion commonly used in decision trees such as gini index, information gain and towing rule are not differentiable with respect to hyperplane parameters [19]. So, decision trees use some greedy search techniques for finding the best hyperplane at each node. The decision trees in RaF exhaustively search for a single feature or the best axis-parallel hyperplane. However, such axis-parallel decision boundaries may fail to exploit the geometric structure in the data. Thus, oblique splits are employed to better capture the geometric structure in the data. However, an exhaustive search for the best oblique hyperplane is computationally expensive [20]. Thus, linear hyperplanes are used in oblique decision trees without optimizing any criteria or the search for the oblique splits is generally based on heuristic approaches and are non-optimal. Typically, multiple classes are placed into two hyperclasses (except in case of binary classes) and a linear classifier is employed to generate an oblique split. Thus, the present realizations of oblique decision trees do not evaluate many promising oblique splits to select the best. In Chapter 4, we present a random forest of heterogeneous oblique decision trees that employ several linear classifiers at each non-leaf node on some top ranked partitions which are obtained via one-vs-all and two-hyperclasses based approaches and ranked

based on ideal Gini scores and cluster separability. The oblique hyperplane that optimizes the impurity criterion is then selected as the splitting hyperplane for that node. The approach is generic and any linear classifier can be used in our framework. We also show that our oblique random forests are more diverse and accurate than other random forest based methods.

Training DNNs using back-propagation (BP) technique is time-consuming, it may also fail to converge to a global minimum, thus, giving sub-optimal performance or lower generalization [21]. Also, such deep learning models require large amount of training data. While the usual image and speech datasets that are commonly used with deep learning models have abundant data, there are datasets from a wide variety of domains, such as agriculture, credit scoring, health outcomes, ecology and others, with very limited data size. The performance of the state-of-the-art deep learning models with standard feed-forward neural networks (FNNs) on such datasets are far from superior [15]. Apart from the conventional BP-trained neural networks, there is also a growing interest in the randomization based neural networks [11, 22–24] such as RVFL. In Chapter 5 and Chapter 6, we investigate the performance of deep learning models based on RVFL networks.

Ensembles of neural networks are much more robust and accurate than individual networks [25–28]. Because of the existence of several randomization operations in their training procedure, neural networks are regarded as unstable algorithms. It is therefore not surprising that two neural networks with identical architectures optimized with different initialization or slightly perturbed training data will converge to different solutions. This diversity can be exploited through ensembling, in which multiple neural networks are trained with slightly different training set or parameters and then combined with majority voting or averaging. Ensembling often leads to drastic reductions in error rates. However, this comes with an obvious trade off: computational cost. While ensembling shallow neural networks doesn't incur great computational cost, the same is not true for the ensembling of deep networks. In Chapter 6, we present an ensemble deep RVFL network which can be regarded as a marriage of ensemble and deep learning that is simple and straight-forward to implement.

To summarize, the main contributions of this thesis are as follows.

1. Both neural networks and decision trees are popular machine learning methods and are widely used to solve problems from diverse domains. These two classifiers are commonly used base classifiers in an ensemble framework. In

Chapter 3, we present an ensemble classifier based on the fusion of a fast RVFL neural network and decision trees. The proposed method provides a rich insight into the data by grouping together the confusing or hard to classify samples and thus, provides an opportunity to employ fine-grained classification rule over the data.

2. In Chapter 4, a random forest of heterogeneous oblique decision trees is proposed that employ several linear classifiers at each non-leaf node on some top ranked partitions which are obtained via one-vs-all and two-hyperclasses based approaches and ranked based on ideal Gini scores and cluster separability. We benchmark 190 classifiers on 121 UCI datasets. The results show that the oblique random forests proposed herein are the top 3 ranked classifiers.
3. In Chapter 5, several multi-layer RVFL neural network variants are presented by utilizing the framework of multi-layer ELM. Specifically, we introduce direct connections (feature reuse) from preceding layers to the fore layers of the network as in the original RVFL network. Such connections help to extract better higher level representations and also reduce the model complexity. Furthermore, we also introduce denoising criterion in the autoencoders to achieve better higher level representations than the ordinary autoencoders. Extensive experiments on several classification datasets show that our proposed multi-layer networks achieve overall better and faster generalization than the other relevant state-of-the-art multi-layer ELMs.
4. Inspired by the principles of Random Vector Functional Link (RVFL) network, in Chapter 6, we present a deep RVFL network (dRVFL) ¹ with stacked layers. The parameters of the hidden layers of the deep RVFL network are randomly generated within a suitable range and kept fixed while the output weights are computed using the closed form solution as in a standard RVFL network. We also propose an ensemble deep RVFL network (edRVFL) that can be regarded as a marriage of ensemble learning with deep learning. Unlike traditional ensembling approaches that require training several models independently from scratch, the ensemble deep RVFL network is obtained by training a single deep RVFL network once. Both dRVFL and edRVFL frameworks are generic and can be used with any RVFL variant. To illustrate this, we integrate the deep learning networks with a recently proposed

¹We use the terms ‘multi-layer’ and ‘deep’ interchangeably throughout this thesis.

Sparse pre-trained RVFL (SP-RVFL). Extensive experiments on benchmark datasets from diverse domains show the superior performance of our proposed deep RVFL networks.

1.3 Overview of the Thesis

This thesis consists of eight chapters organized as follows.

In **this Chapter**, we introduced the research background, the motivation behind our research in Random Forest and Random Vector Functional Link Neural Network, and our main contributions.

In **Chapter 2**, we will go through the basics of RaF and RVFL. We will discuss why ensemble methods such as RaF have better performance compared to a single decision tree. We will then introduce two popular ensembles of decision trees. We will also introduce the fundamentals of neural networks specifically randomization based neural networks such as RVFL.

In **Chapter 3**, we will introduce a hybrid classifier based on RaF and RVFL. Typically, a decision tree performs binary splits at each non-leaf node. Here, we utilize RVFL to create multi-way splits at the root node.

In **Chapter 4**, we will introduce another random forest classifier, heterogeneous random forest. The heterogeneous random forest differs from conventional random forests in that it employs diverse linear classifiers at each non-leaf node. We will discuss why the proposed heterogeneous idea is generic and how it results in accurate and diverse trees.

In **Chapter 5**, we will introduce several multi-layer RVFL networks. We will also discuss the importance of feature re-use in randomization based multi-layer networks such as RVFL.

In **Chapter 6**, we will introduce another deep RVFL neural network variant. We will also introduce an ensemble deep RVFL network.

In **Chapter 7**, we will present an overall comparison of the classifiers presented in this thesis on some real-world datasets.

Finally, in **Chapter 8**, we will summarize this thesis and recommend future research directions.

Chapter 2

Literature Review

In this chapter, we will present the literature of the classifiers and methodologies used in this thesis. Before delving into the classifiers, we will first discuss the ensemble construction process and some of the reasons behind its impressive performance. Next, we will discuss about two popular classifiers: decision trees and neural networks.

2.1 Ensemble Methods

An ensemble of classifier consists of several classifiers trained either independently or sequentially as shown in Fig. 2.1. Bagging [5] and Boosting [29] are the most popular ensemble techniques. The former consists of classifiers trained independently while in the latter, classifiers are trained sequentially. In Fig.2.1, $\mathbf{D}$ is the original training dataset. $\mathbf{D}^{(i)}$, $i \in \{1, \dots, L\}$ are the bootstrap versions of $\mathbf{D}$, $\mathbf{C}^{(i)}$ are the base predictors (base classifiers), and $\mathbf{C}_{en}$ is the combiner responsible for combining the predictions of the various classifiers [30]. The solid lines represent the framework for Bagging while the dashed lines portray Boosting.

Dietterich in [31] states that an ensemble of classifiers is more accurate than any of its individual members if the classifiers are diverse and accurate. Diversity can be either in the data or in the classifier itself. Data diversity is generally obtained by random redistribution of the training set as in Bagging or using weights to determine the distribution of the training set as in Boosting. Diversity can also be engraved in the classifier by injecting some randomness in each classifier of the ensemble. We will discuss more about diversity in Section 2.1.1. A classifier is

accurate if its performance is better than random guessing i.e. its error rate is lower than 0.5.

The success of ensemble methods can be attributed to three fundamental reasons: statistical, computational and representational [31]. According to the statistical viewpoint, a learning algorithm can be viewed as a search of the best hypothesis in a hypotheses space $\mathcal{H}$. In the presence of sufficient training data, it is feasible to obtain an optimal hypothesis for a given problem. However, in most cases, we only have limited training examples. In such cases, the learning algorithm can find different hypotheses in $\mathcal{H}$ that fit equally to the training data. This increases the risk of choosing a wrong hypothesis. However, by taking an average of these hypotheses we can avoid this risk.

Many learning algorithms such as neural networks or decision trees apply local optimization techniques that may stuck in local optima. For example, decision tree induction employ greedy local optimization at each node while neural networks apply gradient descent techniques to minimize an error function. However, the optimal training of both neural network and decision trees is NP-hard [32, 33]. This implies that, even though a learning algorithm can find the best hypothesis in principle, we actually may not be able to find it in practice. Thus, according to the computational viewpoint, ensemble methods that employ local searches from various starting points may achieve a better approximation of the unknown function than a single classifier.

The third reason is representational. It is possible that the best possible approximation of a function f may not be within $\mathcal{H}$. Thus, by combining the hypothesis of individual learning instances (base classifiers), an ensemble expands the space of the representable functions. In addition to these reasons, bias-variance-covariance decomposition [34], margin-theory [29] and strength-correlation [8] all corroborate that the ensemble methods improve the generalization of the classifier compared to a single classifier. We will present a brief overview of the bias-variance-covariance decomposition in Section 2.1.2 .

2.1.1 Diversity

Diversity is crucial in ensemble methods. If two classifiers make different errors on different data points, then these two classifiers are said to be diverse. Diversity within the base classifiers results in less correlation between the individual classi-

Figure 2.1: Framework of an ensemble method. The three major steps in ensemble construction include: Data Generation, Base Prediction and Aggregated Prediction. The solid lines indicate the ensemble framework for Bagging type ensembles. The dashed line indicates the framework for Boosting type ensembles.

fiers so that the examples that are misclassified are different from one individual classifier to another [31]. If the base classifiers are correlated, they are likely to misclassify the same examples. Then, an ensemble method is no better than its individual member. In general, diversity can be introduced in the ensemble methods via three ways: data diversity, parameter diversity and structural diversity [28].

1. Data Diversity: Different predictors are trained with slightly different datasets obtained from the original dataset.
2. Parameter Diversity: Different parameter sets are used to generate diverse base predictors. The outputs of the base predictors trained with same training dataset may vary with different parameter sets. Base predictors tuned with slightly different hyper-parameters results in diverse classifiers.
3. Structural Diversity: The structural diversity is obtained by including predictors with different architectures in an ensemble. An ensemble of such base predictors that vary in size, parameters and architecture is known as heterogeneous ensemble.

2.1.2 Bias-Variance-Covariance Analysis

The bias-variance-covariance insight was originally borrowed from the field of regression with squared loss as the loss function [35]. For classification problems, similar decomposition is inappropriate because of categorical class labels. Nevertheless, several ways to decompose error into bias, variance and covariance terms

in classification tasks have been proposed in [36–38]. In general, the error of an ensemble method can be decomposed as:

$$E[\bar{f} - t]^2 = \text{bias}^2 + \frac{1}{L} \text{var} + \left(1 - \frac{1}{L}\right) \text{covar}, \quad (2.1)$$

where

$$\begin{aligned} \text{bias} &= \frac{1}{L} \sum_i (E[f_i] - t), \\ \text{var} &= \frac{1}{L} \sum_i E[f_i - E[f_i]]^2, \\ \text{covar} &= \frac{1}{L(L-1)} \sum_i \sum_{j \neq i} E[f_i - E[f_i]](f_j - E[f_j]). \end{aligned} \quad (2.2)$$

where f_i is the output of each individual classifier, t is the target and L is the ensemble size. The error decomposition in Eq. (2.1) consists of three components: a bias term measuring the average difference between the prediction of the base classifier and the target function, a variance term measuring the average variability of the constituent base classifiers and a covariance term measuring the average pairwise difference between the base classifiers. This decomposition of the error shows that the performance of an ensemble method increases if the individual base classifiers are diverse and accurate. Other theoretical insights such as margin-theory and strength-correlation have also been shown to be equivalent to the bias-variance-covariance decomposition [28].

2.2 Decision Trees and their ensemble

Decision Tree (DT) is a popular learning algorithm. The popularity of decision tree stems from its simplicity and interpretability [39]. A decision tree is basically a set of questions organized in a hierarchical manner and represented graphically as a tree. A decision tree consists of nodes and edges. The nodes are either internal (split) or leaf (terminal). The internal nodes are split into two child nodes until a stopping criteria is met, after which they become a leaf. Each internal node is associated with a test function defined as:

$$f(\mathbf{x}; \Theta) = \begin{cases} 1 & \text{if } x_i < \Theta_2 \\ 0 & \text{otherwise.} \end{cases} \quad (2.3)$$

where $\mathbf{x} = (x_1, x_2, \dots, x_d) \in \mathbb{R}^d$ is a data point with x_i as some scalar feature response and $\Theta_2 \in \mathbb{R}$ is a threshold. The outcome determines the child node to which $\mathbf{x}$ is routed. For instance, 0 may represent left child node while 1 may represent right child node. A decision tree employs recursive partitioning of the training data into smaller subsets that further aid in classification by optimizing an impurity criterion such as information gain or gini impurity [39]. Each node chooses the best test function Θ^* from a pool of potential test functions by optimizing the impurity metric. The objective is to make the resulting child nodes as pure as possible, i.e containing training samples of a single class only. The gini impurity is defined as:

$$Gini = \frac{n_t^l}{n_t} \left[1 - \sum_i^K \left(\frac{n_{w_i}^l}{n_t^l} \right)^2 \right] + \frac{n_t^r}{n_t} \left[1 - \sum_i^K \left(\frac{n_{w_i}^r}{n_t^r} \right)^2 \right], \quad (2.4)$$

where n_t is the number of data samples in the parent node, n_t^l, n_t^r is the number of data samples that reach the left and right child nodes of the current parent node, and $n_{w_i}^r, n_{w_i}^l$ are the number of samples of class w_i in the child nodes. Each leaf node of the tree is associated with a class label. A sample thus, traverses through different internal nodes, dictated by the test function at each node, before reaching a leaf node. The class label of that sample is given by the class label of that particular leaf.

Based on the nature of test (split) functions, decision trees are categorized into two types: univariate (axis-parallel or orthogonal) and multivariate (oblique) [40]. In a univariate decision tree, the parameter Θ of the test function $f(\mathbf{x}; \Theta)$ is based on a single feature i.e. the node selects a single feature among the d features that best minimizes the impurity criterion. The result is an axis aligned hyperplane at each node as shown in Fig. 2.2. The final decision boundary produced by a tree is of staircase type. Such staircase boundary results in large number of internal nodes and may fail to properly approximate complex decision boundaries. Therefore, in multivariate decision trees, an oblique linear hyperplane is used at each node. Specifically, Θ depends on a linear combination of features as shown in Eq. (2.5). Since the decision boundary can orient in any direction to the axes, the trees with such hyperplanes is also known as oblique trees. The oblique decision boundary on a toy example is shown in Fig. 2.2.

$$f(\mathbf{x}; \Theta) = \begin{cases} 1 & \text{if } \sum_{i=1}^d w_i x_i < \Theta_2 \\ 0 & \text{otherwise} \end{cases} \quad (2.5)$$

Figure 2.2: Classification boundaries generated by axis-parallel and oblique decision tree in a toy example for binary classification problem. Axis-parallel or staircase decision boundaries may fail to approximate the optimal decision boundary. However, oblique trees with linear hyperplanes can better approximate the complex decision boundary. Best viewed in color.

where w_i is the weight coefficient for each feature in d .

A decision tree is generally regarded as an unstable classifier. The predictive capability of a decision tree greatly varies even when there is a small perturbation in the training set or in construction [41]. Large or fully-grown decision trees also overfit easily [42]. Thus, ensembles of decision trees are generally employed to achieve better generalization. An extensive comparison of hundreds of classifiers posit ensembles of decision trees as the best ranked classifiers [2, 10]. Next, we will discuss two popular ensemble of decision trees, Random Forest and Oblique Random Forest. Random Forest and Oblique Random Forest are employed in variety of tasks including image classification [43, 44], object recognition and detection [45, 46], action recognition and detection [47, 48], semantic segmentation [49, 50], visual tracking [51], facial expression recognition [52], regression problems [53], time-series forecasting [54] and so on.

2.2.1 Random Forest

A Random Forest (RaF) [8] is a committee of univariate decision trees that combines the concept of bagging and random subspace [55]. Based on bagging, each tree of the forest is grown independently (or in parallel) with bootstrapped samples from the training set whose distribution is similar to the original distribution of the training samples. Random subspace means Θ parameter of the test function at each node (cf. Eq. (2.3)) depends only a subset of features $q \subset d$. The rationale behind these two concepts is to propel stronger randomization and less correlation among the trees of the forest [8]. Each tree T_i in the ensemble casts a vote for $\mathbf{x}$. The final classification is based on the majority voting of all the trees T . The pseudo-code for Random Forest is presented in Table 2.1.

Table 2.1: Random Forest

Random Forest Algorithm:
<p>Given:</p> <p>$\mathbf{X}$ is the training dataset with dimension $N \times d$ where N is the number of training samples, d is the number of input features.</p> <p>$\mathbf{y}$ is the training labels with dimension $N \times 1$.</p> <p>L is the ensemble size or number of trees in the forest.</p> <p>$m_{try}(= q)$ is the number of features randomly selected to split at each non-leaf node.</p> <p>Training:</p> <ol style="list-style-type: none"> 1. Generate the training set for T_i ($i = 1, 2, \dots, L$) by drawing N bootstrapped samples from the training data. 2. Grow a tree T_i to the bootstrapped samples repeating the following steps at each node of the tree until it is fully grown. <ol style="list-style-type: none"> (a) Randomly select m_{try} features from the original feature size, and pick the best variable/split-point amongst them (cf. Eq. (2.4)). (b) Split the node into two nodes. <p>Testing:</p> <p>To classify a test data, it is pushed down in each tree of the forest. Each tree votes for a class. The predicted class for that data is the one with the most votes in the forest.</p>

Some of the appealing characteristics of Random Forest are:

1. inherent multi-class handling capability

2. less tunable parameters
3. robustness to label noise
4. suitable for parallel processing
5. enables variable (feature) selection
6. ability to handle categorical as well as numerical data

2.2.2 Oblique Random Forest

An Oblique Random Forest (obRaF) is an ensemble of multivariate or oblique trees [56]. Decision trees in standard RaF use local greedy search for finding the best hyperplane at each node. The decision trees exhaustively search for a single feature or the best axis-parallel hyperplane. However, such an exhaustive search for the best oblique hyperplane is computationally expensive [20]. Thus, linear hyperplanes are typically used in oblique decision trees without optimizing any criterion or the search for the best oblique split is generally based on heuristic approaches. Researches have proposed different methods to generate an oblique hyperplane. In Forest-LC [40], the search for the best oblique hyperplane is performed over a linear combination of two (or more) randomly selected features with random coefficient uniformly distributed over the interval $[-1,1]$. However, this method is highly heuristic and it is difficult to obtain an optimal hyperplane in finite search time. OC1 [20] uses a randomized search based on Forest-LC. However, it searches in one dimension at a time, which is computationally expensive in high-dimensional feature spaces. One alternative to optimize in all dimensions is by employing evolutionary approaches [57, 58]. Apart from these approaches, linear classifiers such as Support Vector Machine (SVM), Linear Discriminant Analysis (LDA), Logistic Regression (LR), Multisurface Proximal Support Vector Machine (MPSVM), Ridge Regression (RR) and so on are used to generate linear (oblique) hyperplanes in [19, 56, 59–61] respectively.

The tree induction process in obRaF is similar to that of RaF except the nature of the split functions (cf. Eqs. (2.3) and (2.5)). Below, we present a brief overview of some of the linear classifiers used in obRaF to generate linear splits.

Linear Discriminant Analysis : An LDA [62] is a popular technique for dimensionality reduction and classification. It reduces dimensionality while seeking to preserve as much of the class discriminatory information as possible. The op-

timal projection in LDA is obtained by minimizing the within-class distance and maximizing the between-class distance simultaneously, thus achieving maximum class discrimination. The objective function is:

$$\max \frac{w^T S_B w}{w^T S_w w}, \quad (2.6)$$

where $S_w = \sum_k \sum_{i \in C_k} (x_i - m_k)(x_i - m_k)^T$ is the within-class scatter matrix, m_k is the mean of class C_k , m is the global mean, $S_B = \sum_k n_k (m_k - m)(m_k - m)^T$ is the between-class scatter matrix. The solution for Eq. (2.6) is:

$$w = S_w^{-1} (m_1 - m_2). \quad (2.7)$$

With pooled covariance matrix and multi-variate normal distribution of the classes, the above solution becomes:

$$w = \Sigma^{-1} (\mu_2 - \mu_1), \quad (2.8)$$

where μ_1 and μ_2 are the mean parameters of class 1 and class 2 respectively. The threshold can be computed by setting the discriminant function of each class to be equal.

$$b = 0.5(\mu_2 + \mu_1)^T \Sigma^{-1} (\mu_2 - \mu_1) - \log \left(\frac{N_2}{N_1} \right), \quad (2.9)$$

where N_1 and N_2 are the number of samples of class 1 and 2 respectively and w and b are the weight coefficients and bias (parameters of the hyperplane).

Support Vector Machine: SVM [63] constructs an optimal separating hyperplane between two classes with the maximal margin. It aims to learn a discriminant function of the form:

$$g(x) = w^T \phi(\mathbf{x}) + b. \quad (2.10)$$

The above equation requires solving the following quadratic optimization problem:

$$\begin{aligned} & \text{minimize} \quad \frac{1}{2} w^T w + C \sum_{i=1}^N \xi_i \\ & \text{subject to} \quad y_i [w^T \phi(x_i) + b] \geq 1 - \xi_i, \\ & \quad \quad \quad \xi_i \geq 0 \end{aligned} \quad (2.11)$$

The Lagrangian function of the above problem is:

$$\text{maximize } \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j), \quad (2.12)$$

where $K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$ is the kernel function and α is the dual variable. The choice of kernel function (Gaussian, polynomial or linear) is data dependent. Solving Eq. (2.12) gives $w = \sum_{i=1}^N \alpha_i \phi(x_i)$, the hyperplane of SVM.

Least Squares Support Vector Machine: The least squares formulation of SVM, LSSVM [64], replaces the inequality constraints of SVM by equality constraints. Thus, the solution is obtained by solving a system of linear equations instead of a quadratic programming (QP) problem. The optimization problem is:

$$\begin{aligned} \text{minimize } & \frac{1}{2} w^T w + \gamma \frac{1}{2} \sum_{i=1}^N e_i^2 \\ \text{subject to } & y_i [w^T \phi(x_i) + b] = 1 - e_i \end{aligned} \quad (2.13)$$

Constructing the Lagrangian function results in a linear system instead of a Quadratic Programming (QP) problem.

$$\begin{bmatrix} 0 & 1_N^T \\ 1_N & \Omega + \gamma^{-1} I_N \end{bmatrix} \begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ Y \end{bmatrix} \quad (2.14)$$

with $Y = [y_1, \dots, y_N]^T$, $1_N = [1, \dots, 1]^T$ and $\alpha = [\alpha_1, \dots, \alpha_N]^T$. I_N is a $N \times N$ identity matrix and $\Omega_{ij} = \phi(x_i)^T \phi(x_j) = K(x_i, x_j)$.

Multisurface Proximal Support Vector Machine: The MPSVM [65] computes two clustering hyperplanes H_1 and H_2 for each of the two groups of data D_1 and D_2 , such that H_1 (or H_2) is closest to one class of data D_1 (or D_2) and as far as possible from the other class D_2 (or D_1). The clustering (proximal) planes are found from the eigen vectors corresponding to the smallest and largest eigen values of the following generalized eigenvalue problem:

$$Gz = \lambda H z, \quad z \neq 0 \quad (2.15)$$

where $z = [w \ b]^T$, $G = [A \ -e]^T [A \ -e]$, $H = [B \ -e]^T [B \ -e]$ and A, B are the matrices of class 1 and 2 respectively and e is a vector of ones.

Ridge Regression: A least squares minimizes the squared loss:

$$\underset{w}{\text{minimize}} \|Y - \mathbf{X}w\|^2, \quad (2.16)$$

where $\|\cdot\|$ denotes the L_2 norm. A ridge parameter λ is added to the above expression in RR [66] to control the trade-off between the bias and variance of the estimate that leads the following problem:

$$\underset{w}{\text{minimize}} \|Y - \mathbf{X}w\|^2 + \lambda\|w\|^2. \quad (2.17)$$

The closed form solution for the above problem is $w = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T Y$ where $\mathbf{I}$ is an identity matrix.

Logistic Regression: For a binary classification problem ($Y = \pm 1$), a LR [67] models the conditional probability as:

$$P(y = \pm 1|x) = \frac{1}{1 + e^{-yw^T x}}. \quad (2.18)$$

LR minimizes the following regularized negative log-likelihood:

$$C \sum_{i=1}^N \log(1 + e^{-y_i w^T x_i}) + \frac{1}{2} w^T w, \quad (2.19)$$

where $C > 0$ is a penalty parameter. The equivalent dual form of the above equation is:

$$\begin{aligned} &\underset{\alpha}{\text{minimize}} \quad \frac{1}{2} \alpha^T Q \alpha + \sum_{i=1}^N \alpha_i \log \alpha_i + (C - \alpha_i) \log(C - \alpha_i) \\ &\text{subject to} \quad 0 \leq \alpha_i \leq C, \quad i = 1, \dots, N \end{aligned} \quad (2.20)$$

2.3 Artificial Neural Network

Artificial Neural Network (ANN) is a learning model inspired by the human brain [68]. The human brain consists of colossal amount of computational units called neurons that are interconnected by information transmission channels called synapse. While a single neuron by itself cannot perform sophisticated intelligent tasks, the information exchange between a population of neurons via synapses in tandem culminates to tremendous capabilities of natural brain. The basic idea behind ANNs is to mimic the functionality of natural brain using highly intercon-

nected system of simple abstract model of neurons as computational units. The synapse is represented by the weights of the network connection which are updated or learnt during the training procedure. Typically, an ANN is trained using back-propagation (BP) technique so as to minimize the loss function (cross-entropy or mean square error or others) depending on the particular task. Specifically, a gradient of the loss function is computed with respect to the network parameters (weights) and the weights are iteratively updated in the direction of the negative gradient so as to minimize the loss function. The problems with BP based optimization method are slow training process, failure to converge to a single global minimum if there exists many local minima and sensitivity to learning rate setting [21]. Also, large amount of training data is required to optimize thousands of model parameters. Thus, such training procedure is too cumbersome and time consuming.

Apart from the conventional BP-trained neural networks, there has also been a growing interest in the class of randomization based neural networks [22–24]. Randomization based neural networks avoid the pitfalls of conventional BP-trained neural networks [21, 69, 70]. They remedy the aforementioned problems of BP-trained NNs by either randomly fixing the network configurations (such as the connections) or some network parameters while optimizing the rest by a closed form solution or an iterative procedure, or randomly corrupting the input data during the training procedure [71]. Such randomization based methods have demonstrated good learning performance in various domains [71–73]. Among the randomization based methods, Random Vector Functional Link (RVFL) [11] network has rapidly gained significant traction because of its superior performance in several diverse domains ranging from visual tracking [74], classification [10, 75], regression [76], to forecasting [77, 78].

We will introduce details of neural networks in the following sections.

2.3.1 Single Layer Feedforward Neural Network

A Single-hidden Layer Feedforward Neural Network (SLFN) consists of an input layer, a hidden layer and an output layer as shown in Fig. 2.3. Consider a training dataset $\mathbf{X}$ with N training samples, $\mathbf{X} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ where $\mathbf{x}_i \in \mathbb{R}^d$ is a d -dimensional input vector and $\mathbf{y}_i \in \mathbb{R}^c$ is a c -dimensional output vector. The

Figure 2.3: Architecture of SLFN.

output of a SLFN with N_h hidden nodes is:

$$y = g \left(\sum_{j=1}^{N_h} w_{jo} v_j + b_{ho} \right) \quad j = 1, 2, \dots, N_h, \quad (2.21)$$

$$v_j = f \left(\sum_{i=1}^d w_{ij} x_i + b_{ih} \right) \quad i = 1, 2, \dots, d, \quad (2.22)$$

where x_i is the input to the neuron i_i , $f(\cdot)$ and $g(\cdot)$ are the non-linear activation functions, v_j is the output of the hidden unit h_j , y is the output of the SLFN, w_{ij} is the weight of the connection between the input feature i_i and the hidden unit h_j , w_{jo} is the weight of the connection between the hidden unit h_j and the output, b_{ih} and b_{ho} are the biases.

To train a SLFN, the weights and biases are first randomly initialized and updated using back-propagation (BP).

2.3.1.1 Random Vector Functional Link Network

RVFL [11, 79] network is a SLFN in which the weights and biases of the hidden neurons are randomly and independently generated within a suitable range in

advance. A basic framework of the standard RVFL network is shown in Fig. 2.4(a). The inputs to the output layer in RVFL consist of both non-linearly transformed features $\mathbf{H}$ from the hidden layer and original input features $\mathbf{X}$. If d be the input data features and N_h be the number of hidden nodes, then there are total $d + N_h$ inputs for each output node. Since the hidden layer parameters are randomly generated and kept fixed during the training phase, only the output weights β_s need to be computed. We use β to represent the output weights of randomized neural networks hereafter. Thus, the resulting optimization problem can be mathematically represented as:

$$O_{RVFL} = \min_{\beta_s} \|\mathbf{D}\beta_s - \mathbf{Y}\|^2 + \lambda\|\beta_s\|^2, \quad (2.23)$$

where $\mathbf{D} = [\mathbf{H} \ \mathbf{X}]$ is the concatenation of hidden features and original features and λ is the regularization parameter.

Typically, Eq. (2.23) can be solved via a closed form solution using either ridge regression (i.e. $\lambda \neq 0$) or Moore-Penrose pseudo-inverse (i.e. $\lambda = 0$) [80]. Using Moore-Penrose pseudo-inverse, the solution is given by: $\beta_s = \mathbf{D}^+ \mathbf{Y}$ while using the regularized least squares (or ridge regression), the closed form solution is given by:

$$\text{Primal Space: } \beta_s = (\mathbf{D}^T \mathbf{D} + \lambda \mathbf{I})^{-1} \mathbf{D}^T \mathbf{Y}, \quad (2.24)$$

$$\text{Dual Space: } \beta_s = \mathbf{D}^T (\mathbf{D} \mathbf{D}^T + \lambda \mathbf{I})^{-1} \mathbf{Y}. \quad (2.25)$$

The training complexity in the RVFL network introduced by the matrix inversion operation can be circumvented by using either primal or dual solution depending on the sample size or total feature dimensions (i.e. input features plus total number of hidden neurons) [21].

RVFL, with hidden layer parameters sampled from uniform distribution within a proper range, is an efficient universal approximator [81]. In practice, the hidden layer parameters are sampled from the interval $[-S, +S]$ where S is a scale factor that is determined during the parameter tuning stage [82].

Randomization based neural networks greatly benefit from the presence of direct links from the input layer to the output layer as in RVFL network [25, 76, 78]. The original features are reused or propagated to the output layer via the direct links. The direct links act as a regularization for the randomization [82, 83]. It also helps to keep the model complexity low with the RVFL network being thinner

Figure 2.4: Framework of RVFL (1994) and ELM (2004) networks. The structure of RVFL and ELM differ in the presence (absence) of direct links and bias term (not shown in the figure). The red lines represent the direct links (original features) from the input to the output layer. The weights for the blue lines are randomly generated from a certain range and kept fixed. Only the output weights (associated with red and black lines) need to be computed. Best viewed in color.

and simpler compared to its other counterparts. With the Occam's Razor principle and PAC learning theory [84] advocating for simpler and less complex models, this makes the RVFL network attractive to use compared to other similar randomized neural networks.

2.3.1.2 Extreme Learning Machine

Extreme Learning Machine (ELM) [12], developed in 2004, can be viewed as a variant of RVFL without direct links and bias term (see Figure 2.4(b)). Thus, Eq. (2.23) becomes

$$O_{ELM} = \min_{\beta_E} \|\mathbf{H}\beta_E - \mathbf{Y}\|^2 + \lambda \|\beta_E\|^2. \quad (2.26)$$

Its solution is:

$$\text{Primal Space: } \beta_E = (\mathbf{H}^T \mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{H}^T \mathbf{Y}, \quad (2.27)$$

$$\text{Dual Space: } \beta_E = \mathbf{H}^T (\mathbf{H} \mathbf{H}^T + \lambda \mathbf{I})^{-1} \mathbf{Y}. \quad (2.28)$$

2.3.1.3 Kernel Extreme Learning Machine

Kernel Extreme Learning Machine (KELM) [85] is obtained by applying kernel trick to Eq. (2.27). It has the same final solution formulation as that of Kernel Ridge Regression (KRR) [86]. Based on the Representer Theorem [87], β can be expressed as a linear combination of the samples in the feature space $\phi(x)$ as $\beta = \sum_i \alpha_i \phi(x_i)$. The learning objective thus, becomes:

$$O_{KRR} = \min_{\alpha} \|Y - \mathbf{K}\alpha\|^2 + \lambda \alpha^T \mathbf{K}\alpha, \quad (2.29)$$

Its closed form solution is:

$$\alpha = (\mathbf{K} + \lambda \mathbf{I})^{-1} \mathbf{Y}, \quad (2.30)$$

where $\mathbf{K}$ is a kernel matrix and $\mathbf{K}_{ij} = k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$. Commonly used kernel functions are Gaussian kernel, linear kernel, and polynomial kernel. Using the kernel trick, it is possible to compute the inner product without having to explicitly compute the coordinates of the samples in feature space through $\phi(x)$.

2.3.1.4 Autoencoder

An Autoencoder (AE) [88] consists of two parts, an encoder and a decoder. The AE differs from the SLFNs described in the preceding sections in that the output of the encoder transforms the original representation into some rich, meaningful representation such that the decoder is able to reconstruct the original representation from the encoder's representation. Based on the number of hidden neurons, the encoder either compresses or extracts higher-dimensional representation from the original input representation [88].

Encoder The encoder f maps an input $\mathbf{x} \in \mathbb{R}^d$ to hidden representation $\mathbf{h}(\mathbf{x}) \in \mathbb{R}^{d'}$. It consists of an affine mapping followed by a non-linearity:

$$\mathbf{h} = f(\mathbf{x}) = a_f(\mathbf{W}\mathbf{x} + \mathbf{b}_h), \quad (2.31)$$

where a_f is a non-linear activation function, $\mathbf{W}$ is a $d' \times d$ weight matrix and $\mathbf{b}_h$ is a bias vector of dimension d' . The parameter set is represented by $\Theta = \{\mathbf{W}, \mathbf{b}_h\}$.

Decoder The decoder g maps the hidden representation $\mathbf{h}$ back to a reconstruc-

tion $\mathbf{y}$:

$$\mathbf{y} = g(\mathbf{h}) = a_g(\mathbf{W}'\mathbf{h} + \mathbf{b}_g), \quad (2.32)$$

where a_g is the decoder's activation function, typically a linear activation function. The decoder's parameters represented by Θ' are a bias vector $\mathbf{b}_g$ and weight matrix $\mathbf{W}'$.

The parameters Θ and Θ' of an autoencoder are obtained by minimizing the reconstruction error on a training set of examples $\mathbf{X}$ by solving the following optimization problem:

$$\arg \min_{\Theta, \Theta'} \mathbb{E}(L(\mathbf{X}, g(f(\mathbf{X})))), \quad (2.33)$$

where L is the reconstruction error. For linear reconstruction, the squared error $L(x, y) = \|x - y\|^2$ is used. An autoencoder can be trained either via BP or using closed form solution or an iterative procedure. We will discuss an AE's training via closed form solution and iterative procedure in the later sections.

2.3.1.5 Denoising Autoencoder

One amongst the several variants of AE is Denoising Autoencoder (DAE) [89]. The DAE is expected to learn a stable and robust higher level representation without simply copying the input representation or learning an identity mapping. This is achieved by corrupting the input $\mathbf{X}$ before sending it through the autoencoder, which is then trained to reconstruct the clean version (i.e. to denoise). This lead to the following optimization problem:

$$\arg \min_{\Theta, \Theta'} \mathbb{E}(L(\mathbf{X}, g(f(\tilde{\mathbf{X}})))) , \quad (2.34)$$

where $\tilde{\mathbf{X}}$ is the corrupted version of $\mathbf{X}$ obtained from a corruption process $q(\tilde{\mathbf{X}}|\mathbf{X})$. Typically, the input is corrupted by either adding Gaussian noise $\tilde{\mathbf{X}} = \mathbf{X} + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2 I)$ or by using a binary masking noise, where a fraction ν of randomly chosen input components have their value set to 0. The degree of regularization depends on the degree of the corruption (σ or ν).

2.3.1.6 Sparse pre-trained RVFL

In a standard RVFL network, the hidden layer parameters ($\mathbf{w}$ and $\mathbf{b}$) are randomly generated and kept fixed thereafter. Even though RVFL has demonstrated its efficacy in various domains, its performance is often challenged by randomly assigned hidden layer parameters. To alleviate this issue, the authors in [90] proposed an unsupervised parameter learning based RVFL known as sparse pre-trained RVFL (SP-RVFL). In an SP-RVFL network, an autoencoder with l_1 regularization is employed to learn the hidden layer parameters. Specifically, the optimization problem for the autoencoder is given by:

$$O_{\varpi} = \min_{\varpi} \|\tilde{\mathbf{H}}\varpi - \mathbf{X}\|^2 + \|\varpi\|_1, \quad (2.35)$$

where $\mathbf{X}$ is the input, $\tilde{\mathbf{H}}$ is the hidden layer matrix obtained via random mapping and ϖ is the output weight matrix of the autoencoder. For simplicity, the optimization problem of Eq. (2.35) is rewritten as:

$$O_{\varpi} = p(\varpi) + q(\varpi), \quad (2.36)$$

where $p(\varpi) = \|\tilde{\mathbf{H}}\varpi - \mathbf{X}\|^2$ is the reconstruction loss function and $q(\varpi) = \|\varpi\|_1$ is the l_1 norm regularization. The above optimization problem, Eq. (2.36) is solved using a fast iterative shrinkage-thresholding algorithm (FISTA) [91]. The implementation details of the FISTA algorithm are as follows:

1. Compute the Lipschitz constant γ of the gradient of the loss function ∇p .
2. Initialize $\mathbf{z}_1 = \varpi_0$, $t_1 = 1$. Then, for iteration $j(j \geq 1)$ the following holds:

- (a) The weight matrix is updated as $\varpi_j = s_{\gamma}(\mathbf{z}_j)$ where s_{γ} is given by

$$s_{\gamma} = \min_{\varpi_j} \left\{ \frac{\gamma}{2} \|\varpi_j - (\varpi_{j-1} - \frac{1}{\gamma} \nabla p(\varpi_{j-1}))\|^2 + q(\varpi_j) \right\}. \quad (2.37)$$

- (b) The coefficient t_{j+1} can be calculated as

$$t_{j+1} = \frac{1 + \sqrt{1 + 4t_j^2}}{2}. \quad (2.38)$$

(c) For $\mathbf{z}_{j+1}$, we have

$$\mathbf{z}_{j+1} = \boldsymbol{\varpi}_j + \left(\frac{t_j - 1}{t_{j+1}} (\boldsymbol{\varpi}_j - \boldsymbol{\varpi}_{j-1}) \right). \quad (2.39)$$

The $\boldsymbol{\varpi}$ pre-trained by sparse-autoencoder is then used as the weights of the hidden layer of a standard RVFL. The hidden biases are then computed as:

$$\hat{b}_i = \frac{\sum_{j=1}^d \varpi_{ij}}{d}, \quad i = 1, 2, \dots, N. \quad (2.40)$$

With the pre-trained hidden layer parameters, the output of the hidden layer $\mathbf{H}$ of RVFL is computed as:

$$\mathbf{H} = g(\mathbf{X}\boldsymbol{\varpi} + \hat{\mathbf{b}}), \quad (2.41)$$

where $g(\cdot)$ is a non-linear activation function. Only the Eq. (2.41) in SP-RVFL differs from a standard RVFL network with $\mathbf{H}$ of a standard RVFL given by $\mathbf{H} = g(\mathbf{X}\mathbf{w} + \mathbf{b})$ where $\mathbf{w}$ and $\mathbf{b}$ are randomly generated. The optimization problem of SP-RVFL then becomes similar to the optimization problem given in Eq. (2.23). Finally, Eq. (2.24) and Eq. (2.25) are used to compute the output weights β_s as in a standard RVFL network.

2.3.2 Multi-layer (Deep) Neural Network

Randomization based neural networks such as RVFL and ELM are extensively used for different machine learning tasks owing to their simplicity and good performance. With the growing surge of representational (or deep) learning, ELM has also been used as a building block for multi-layer or deep neural networks. Such randomized neural networks are a preferable choice because of their faster training time, lower computational requirements which are some of the crucial properties for time or memory sensitive machine learning tasks. A randomization based AE was first introduced in [92]. Using such randomization based AEs, several variants of multi-layer ELM have been proposed in the literature. In this section, we briefly review randomization based autoencoder with some of the notable multi-layer ELM networks in case of supervised learning especially for classification tasks.

2.3.2.1 Randomization based Autoencoder

In a randomization based AE [92], the input $\mathbf{X}$ is first randomly mapped to a latent representation $\mathbf{h}$ by the encoder. Thus, the encoder parameter set Θ is randomly generated and kept fixed. Depending on the type of regularization imposed in learning the weight matrix $\mathbf{W}'$ of the decoder, three variants of randomization based AE have been proposed so far as discussed below.

Randomization based AE with l_2 regularization. To obtain the decoder parameter set Θ' for the reconstruction of the input, either Eq. (2.27) or Eq. (2.28) is utilized depending on the complexity. Specifically, in [92], the weight matrix of the decoder $\mathbf{W}'$ is obtained using the following equations:

$$\text{Primal Space: } \mathbf{W}' = (\mathbf{H}^T \mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{H}^T \mathbf{X}, \quad (2.42)$$

$$\text{Dual Space: } \mathbf{W}' = \mathbf{H}^T (\mathbf{H} \mathbf{H}^T + \lambda \mathbf{I})^{-1} \mathbf{X}. \quad (2.43)$$

Randomization based AE with l_1 regularization. The l_1 regularized AE has the optimization problem as given in Eq. (2.35). The resulting optimization problem is then solved using FISTA technique as described in Section 2.3.1.6.

Randomization based AE with elastic-net regularization. To further increase the robustness of the AE, the authors in [93], employ elastic net regularization. The optimization problem then is given by:

$$\min_{\beta} \|\mathbf{H}\beta - \mathbf{Y}\|^2 + \lambda \left(\alpha \|\beta\|_1 + \frac{(1 - \alpha)}{2} \|\beta\|_2^2 \right), \quad (2.44)$$

where α controls the proportion of l_1 and l_2 norm and λ is the regularization parameter for elastic-net penalty. Alternating Direction Method of Multipliers (ADMM) [94] is used to solve the above optimization problem.

Kernel-Randomization based AE. Similar to the AEs as discussed above, KELM-AE [95] learns the data transformation from the hidden layer to the output layer except that the input matrix is mapped into a kernel matrix as discussed in Section 2.3.1.3.

Using these randomization based AEs as building blocks, following variants of multi-layer ELM exist.

2.3.2.2 Multi-layer ELM

The Multi-layer ELM (ML-ELM) uses randomization based AE with l_2 regularization [92]. In ML-ELM, several hidden layers are stacked on top of each other in hierarchical manner where the final layer is used for decision making (classification). In such structure, each hidden layer builds an internal representation from the input data with the output of the previous layer used as the inputs of the current one.

2.3.2.3 Hierarchical ELM

The Hierarchical ELM (H-ELM) [96] consists of two components: feature encoding using ELM and an ELM based classifier as shown in Fig. 2.5. For feature extraction, it uses sparse AE (ELM regularized with l_1 norm). The extracted features are then used by ELM classifier (ELM regularized with l_2 norm) for final decision making. In terms of architecture, the difference between ML-ELM and H-ELM is the random feature mapping layer in HELM before classification.

2.3.2.4 Multi-layer kernel ELM

The architecture of Multi-layer kernel ELM (ML-KELM) [95] is similar to H-ELM except that the hidden layers of randomization based AE and ELM classifier are replaced by a kernel matrix $\mathbf{K}$ as discussed in Section 2.3.1.3.

2.3.2.5 Neural-response-based ELM (NR-ELM)

The Neural-response-based ELM (NR-ELM) [93] also follows the architecture of H-ELM with multi-layer ELM based feature mapping with ELM classifier. It employs ELM with elastic-net regularization to learn more compact and meaningful output weights. The NR-ELM is especially used for image classification where the SIFT descriptors [97] are used to extract local features before feeding into the NR-ELM.

2.3.3 Neural Network Ensemble

Like decision trees, neural network is another popular base classifier of an ensemble method. While popular ensemble techniques such as bagging and boosting

Figure 2.5: Architecture of H-ELM. It consists of feature encoding using multi-layer network built with randomization based AE and ELM classifier for final decision making.

can be conveniently used to create an ensemble of shallow neural networks, it is prohibitively expensive for large (deep) neural networks. In large neural networks, there exists thousands of parameters that need to be learned or adjusted during the training procedure. Such networks require specialized hardware such as GPU and days or weeks of training. Thus, training an ensemble of deep neural networks is highly expensive [98]. To grasp the advantage of both ensembles and deep learning, instead of training several individual models independently ('explicit ensembles'), there exists several methods that mimic the conventional ensemble frameworks during the training operation ('implicit ensembles'). More specifically, if the individual models in a neural network ensemble do not share weights as in conventional bagging and boosting based methods, then such an ensemble is known as explicit ensemble. In an implicit ensemble, the individual models share weights [27]. Below, we briefly introduce some of the popular ensemble techniques for large (deep) neural networks.

2.3.3.1 Dropout

Dropout [98] is a form of regularization for fully connected neural network layers. Each hidden neuron of a layer is kept with retention probability p , otherwise being set to 0 with probability $(1 - p)$.

Dropout generally applied to the outputs of a fully connected layer can be written as:

$$\mathbf{H} = m * a(\mathbf{X}\mathbf{w}) \quad (2.45)$$

where $\mathbf{H}$ denotes the output of hidden layer, $*$ denotes element wise product and m is a binary mask vector of size d with each element, j , drawn independently from $m_j \sim \text{Bernoulli}(p)$. Here, $a(\cdot)$ is the activation function.

A single neural network with N hidden units can be seen as a collection of 2^N possible thinned neural networks [98]. During each training iteration, a new thinned network is sampled and trained thus, training a neural network is equivalent to training a collection of 2^N thinned networks with extensive weight sharing. At test time, no nodes are dropped but the weights of the network are scaled-down. That means, if a node is retained with probability p during the training, the outgoing weights of that node are multiplied by p at the test time. This is equivalent to combining 2^N networks with shared weights into a single neural network. So, Dropout can be viewed as an implicit ensemble technique. Dropout reduces overfitting by preventing the co-adaptation of nodes. That means it prevents the network units from collaborating to memorize the training examples. During each training iteration, such technique discourages the network from relying on single features, but rather consider the output of committees of features. Due to the regularizing effect, it improves the generalization performance by reducing the complexity and thus the variance of neural network.

2.3.3.2 DropConnect

DropConnect [99] is the generalization of Dropout in which each connection, rather than each hidden unit, can be dropped with probability $(1 - p)$. Dropout and DropConnect both introduce sparsity in the model while only differing on the fact whether some connections or neurons are randomly dropped. DropConnect also closely resembles random subspace [55] where instead of all the available features

only some randomly selected features are used for model generation. Thus, with DropConnect the fully connected layer becomes a sparsely connected layer. For a DropConnect layer, the output is given as:

$$\mathbf{H} = a((M * \mathbf{w})\mathbf{X}) \quad (2.46)$$

where M is a binary mask that indicates whether a connection is retained or dropped and $M_{ij} \sim \text{Bernoulli}(p)$. Like Dropout, DropConnect is also an implicit ensemble technique.

2.3.3.3 Stochastic Depth

In deep networks trained with stochastic depth [100], a subset of layers is randomly dropped during each training iteration to create an implicit ensemble of networks with varying depth at test time. Like Dropout and DropConnect, stochastic depth also results in implicit ensembles.

2.3.3.4 Swapout

Swapout [101] is another implicit ensembling technique that generalizes Dropout and Stochastic Depth. A neural network trained with Swapout averages a set of architectures covered by Dropout and Stochastic Depth.

2.3.3.5 Snapshot Ensemble

In case of snapshot ensembling [27], a neural network is trained using a cyclic learning rate schedule (Cyclic Cosine Annealing). At the end of each training cycle, the model converges to different local minima. The model weights are saved (also known as ‘snapshot’) each time the model converges, resulting into L model snapshots after training L cycles. This is equivalent to L different models but with the same training cost of a single model trained with a standard schedule. Unlike ensemble techniques discussed above, this is an explicit ensemble technique. But snapshot ensemble is similar to the above methods in that it requires training a single model only.

Chapter 3

An Ensemble of Decision Trees and RVFL

Both decision trees and neural networks are widely used in an ensemble framework. In this chapter, we will present a hybrid ensemble classifier based on decision trees and random vector functional link neural network [102]. The RVFL neural network effectively partitions the training data into several subsets which would be handled by the subsequent expert classifier. This approach is generic and any classifier can be used [103, 104]. We will then present the performance comparison of the hybrid ensemble with the relevant classifiers in several multi-class datasets.

3.1 Introduction

There exists copious researches on decision tree and neural network ensembles. In this chapter, we focus on the hybrid ensemble incorporating both decision tree and neural network. In [105], the authors use decision trees to empirically determine the number of neurons needed in three layers of neural network. [106] extends this idea by mapping stacked RaF to Convolutional Neural Networks (CNN) for semantic segmentation. Similarly, [107] uses decision trees to identify most important variables from breast cancer data set and use those variables as input for their neural network architecture. In a nutshell, [105, 107] focus on inducing neural networks via decision trees and are non-ensemble methods. On the other hand, [49] uses randomized Multi Layer Perceptrons (rMLP) as split functions in each node of the trees. With the surging popularity of representational

learning, there is a recent trend of incorporating representation learning with decision trees [108].

Each non-leaf node of a decision tree partitions the input space into two, sending the training data into two child nodes. Generally, binary splits are popular with decision trees with very few researches on multi-way splits. Multi-way (Multi-branch) splits in decision trees have previously been studied in [17, 18]. In [109], correlation is used to find the best single feature and thresholds to split the training data into multiple branches are computed by SVM. However, such multi-way splits are cumbersome to determine and do not effectively improve the performance of decision trees. In [110], a deep neural network is used to perform a hierarchical partition of the data as in decision trees while creating the clusters of confusing classes. The classes are clustered by employing spectral co-clustering algorithm over confusion matrix computed over the validation dataset. However, it is computationally expensive and requires large dataset.

In this chapter, we investigate the performance of an ensemble of classifiers built with decision trees and neural networks. Since the proposed ensemble is a hybrid ensemble of decision trees and RVFL, we name the new classifiers as RaFL and obRaFL for the ensemble of RVFL with axis-parallel trees and oblique trees respectively. In RaFL and obRaFL, a decision tree is grown for each subset based on the samples partitioned by RVFL. Each subset consists of training samples with majority of the samples from one class and the rest from other classes. Unlike decision trees that partition the training data into two subsets at each non-leaf node, the RVFL in our proposed method partitions the original data into K distinct subsets at the root node. For datasets with two classes, the RVFL in our proposed method partitions the training samples such that the two classes will have the same training set. This results in two identical decision trees, grown with the same training set, in each base classifier. Thus, to avoid duplicating trees (training set) and to obtain disparate trees grown with different training sets in each base classifier, we only consider multi-class classification problems in our work. Our method inherits the intrinsic nature of RaF, that means, it can also be implemented in parallel using multi-cores or parallel architectures using CPU or GPU [111–113].

3.2 Proposed Ensemble Method

First we describe the construction of the hybrid ensemble of classifiers, then we discuss the motivation behind the proposed method.

In the training phase, each training sample $\mathbf{x}_i$ is passed to RVFL. The output of RVFL is a score for each class in that particular dataset. For example, the output of RVFL for $\mathbf{x}_i$ as $[0.4, 0.5, 0.05, 0.05]$ represents the probability of $\mathbf{x}_i$ belonging to class 1,2,3 and 4 respectively. The class with the highest score or probability is the predicted class by RVFL for that particular sample. In the example, class 2 is the predicted class by RVFL for $\mathbf{x}_i$. However, we can see that, there is only a small difference between the probabilities of class 1 and class 2. In such cases, $\mathbf{x}_i$ can be easily misclassified as belonging to class 2, when its actual label can be class 1. Since RVFL can misclassify the training data, two classes with the highest and second-highest scores are selected as the potential classes of each training sample. This means, each training sample is placed in two clusters or subsets based on the prediction scores given by RVFL. The subsets, equal to the number of classes in each dataset, contain samples from different classes that are difficult to classify for RVFL. Basically, each subset consists of training samples with majority of the samples from one class and the rest from other classes. We then use an expert classifier, in our case decision trees, to effectively classify the samples in those subsets. This technique is generic and any classifier can be used after the data partitioning by RVFL. In cases, when the prediction of RVFL is worse, meaning the true class of x_i is neither the highest nor the second-highest class, the training sample is still placed in its true class. The algorithm of our proposed method is presented in Table 3.1 and its visual representation in Fig. 3.1.

In this chapter, unlike the standard decision trees, we apply an efficient neural network (RVFL) to firstly divide the data into ‘ K ’ partitions (K : the number of classes in a dataset) where we grow a decision tree thereafter to improve the accuracy. In each subset/data partition, the class distribution of samples is unique i.e. majority of the samples are from one class and the rest from other classes. The samples from the other classes are those that are “hard” to classify by RVFL. Such partitioning is possible by utilizing the output scores given by RVFL. Decision trees are used thereafter because:

- They naturally handle multiple classes. No need to convert them to a series of binary classification problems (like one-vs-one or one-vs-all) as in

Table 3.1: Proposed Hybrid Ensemble

Algorithm:
<p>Given:</p> <p>$\mathbf{X}$ is the training dataset with dimension $N \times d$ where N is the number of training samples, d is the number of input features.</p> <p>$\mathbf{y}$ is the training labels with dimension $N \times 1$.</p> <p>L is the ensemble size or number of trees in the forest.</p> <p>Training:</p> <ol style="list-style-type: none"> 1. Generate the training set for T_j ($j = 1, 2, \dots, L$) by drawing N bootstrapped samples from the training data. 2. Construct a RVFL network using T_j. 3. For $i = 1, \dots, N$ <ol style="list-style-type: none"> (a) For each data $\mathbf{x}_i$, select the classes with the highest and the second highest scores and store $\mathbf{x}_i$ as the training data for the two decision trees belonging to those two classes. <p>Testing:</p> <p>In the testing phase, a test data is first passed to RVFL. Similar to the training phase, two classes with the highest and the second highest scores are selected based on the output of RVFL. The test data is pushed down in the decision trees of those classes. The final prediction is based on the majority votes of the decision trees.</p>

other popular classifiers such as SVM. Since the samples from other classes are sparse, converting them to such binary problems might deteriorate the performance of the other classifiers however, DT/RaF are robust against imbalance problem if such exists [114–116].

- Instead of growing very large trees as done in RaF, the size of the trees in our proposed ensemble method is relatively small due to the K -partitions performed by RVFL at the root node. Thus, the tree construction process can be distributed among several workstations or cores without exhausting the resources. This property makes our method ideal for parallelization.
- Normally, binary splits are popular with decision trees. The binary splits result in very deep trees. However, by using multi-way splits as proposed here, we can obtain shallow trees that agrees with Ockham's razor principle of building smaller trees. [17, 18] present decision trees with multi-way splits, but their technique of splits generation is very complex (based on clustering

Figure 3.1: A base classifier of our proposed ensemble method

algorithms, significance tests) and the accuracy of such trees is also generally lower than decision trees with binary splits. But, we show that implementing multi-way splits at the root node can further improve the performance of decision trees.

3.3 Experimental setup

The proposed method is evaluated in 65 multi-class UCI datasets [16]. The number of classes in these datasets range from 3 (minimum) to 100 (maximum). We follow the procedures as in [2]. A 4-fold cross validation is developed using the whole data. For those data sets, where the training-testing partition is already available (such as annealing and audiology-std), the classifier is trained on the predefined training set and evaluated on the test set. The ensemble size L is fixed at 500 [2]. Table A1 in the Appendix gives an overview of the data sets used here.

3.3.1 RVFL Configuration

To introduce diversity in the base classifiers, we employ bagging and randomize RVFL in each base classifier. We use the same parameter setting as in [2, 82] where each RVFL randomly picks the activation function and network parameters from the parameter settings listed below:

1. Number of hidden neurons, $\mathcal{N} = 3:203$
2. $\lambda (= (1/2)^C)$ in ridge regression, $C = -5:14$
3. Activation Functions: *radbas*, *sine* and *tribas*
4. Range of the randomization for weights $[-S, +S]$ and bias $[0, S]$, where $S = 2^t$ with $t = -1.5:0.5:1.5$

The RVFL has direct links from input layer to output layer with bias term in the output neuron.

3.3.2 Decision Tree Configuration

We evaluate the method using both axis-parallel and oblique decision trees. The oblique decision tree is based on [19]. It employs MPSVM to generate a linear split at each node (cf. Chapter 2). For the decision trees, we set the m_{try} parameter (number of features selected at each node during the growing phase of the tree) equal to $\sqrt{d}$ where, d is the feature dimension. Similarly, the *minleaf* parameter, which specifies the minimum number of observations per tree leaf, is set to 1 as default [19].

3.4 Results and Discussion

To validate the effectiveness of RaFL and obRaFL, we compare the classification accuracy obtained by our method with RaF, obRaF and RVFL. The average accuracy over the datasets by each method is shown in Table 3.2. For fair comparison, RaF and obRaF is also built with 500 trees as done in [2]. Our results slightly differ from [2] for RaF and [19] for obRaF as we use the same configuration for all classifiers i.e the classifiers are trained with same 500 bags and they all use $\sqrt{d}$ features at each non-leaf node. The RVFL results are tuned for *radbas* activation function and extracted from [82]. The statistical test is based on Friedman test as suggested in [117], and we use the rank of each classifier to reflect its performance as in [2]. In this approach, the classifiers are ranked for each dataset separately, with the best performing algorithm getting the rank of 1, the second best rank 2..., average ranks are assigned in case of ties. Based on the results of each algorithm as presented in Table 3.2, we summarize the overall ranking of each method in Table 3.3.

Table 3.2: Classification accuracy (%) of Different Methods

Datasets	RaF	RaFL	obRaF	obRaFL	RVFL
abalone	64.99	64.75	65.37	65.16	65.73
annealing	56	76.25	76	76	24
arrhythmia	73.45	70.58	65.04	65.71	68.14
audiology-std	84	69	68	60	64
balance-scale	88.3	87.5	90.22	90.87	90.87
breast-tissue	75	73.08	72.12	75.96	64.42
car	97.16	97.45	95.95	97.16	93
contrac	54.28	52.51	51.22	50.68	54.08
ctg-10classes	86.72	87.15	82.77	84.23	79.85
ctg-3classes	94.59	94.96	92.7	92.8	90.77
chess-krvk	70.48	75.6	68.19	70.25	37.07
conn-bench-vowel	98.48	99.46	99.78	100	98.27
dermatology	97.8	97.8	96.98	96.7	97.53
ecoli	88.69	88.99	87.5	86.9	87.8
energy-y1	95.18	95.05	92.32	91.93	93.62
energy-y2	89.71	89.45	89.19	89.58	90.36
flags	66.15	64.06	56.25	55.21	52.6
glass	74.53	73.58	73.11	71.23	68.4
hayes-roth	85.71	89.29	85.71	89.29	85.71
heart-cleveland	56.91	57.57	59.54	58.88	62.5
heart-va	38	35.5	36	37.5	38
image-segmentation	14.29	29.54	14.9	29.32	26.95
iris	95.95	95.95	97.3	97.3	98.65
led-display	73.7	72.3	72.8	71.6	75
lenses	87.5	87.5	87.5	83.33	83.33
letter	96	96.36	96.53	96.28	79.6
libras	78.06	83.89	86.11	84.44	82.22
low-res-spect	92.11	91.92	91.54	91.73	91.35
lung cancer	53.13	50	43.75	46.88	62.5
lymphography	84.46	86.49	87.84	86.49	84.46
molec-biol-splice	95.39	95.01	89.74	90.37	81.68
nursery	99.34	99.43	98.81	98.96	92.85
oocMer12F	91.67	92.16	91.96	92.35	85.69
oocTris5B	91.78	92	93.42	93.75	95.06
optical	97.16	96.7	96.72	97.09	95.6
page-blocks	97.22	97.33	97.04	97.08	95.56
pendigits	95.31	95.8	96.94	97.13	95.63
pb-MATERIAL	91.35	90.38	93.27	92.31	92.31
pb-REL-L	74.04	73.08	79.81	76.92	77.88
pb-SPAN	65.22	64.13	66.3	66.3	71.73
pb-TYPE	70.19	69.23	69.23	71.15	68.27
plant-margin	85.5	83.56	82.82	82.06	77.19
plant-shape	64.06	67.38	70.25	70.19	58.38
plant-texture	82.19	82.5	82.06	81.75	80.75
post-operative	72.73	70.45	70.45	69.32	72.73
primary-tumor	56.1	54.57	54.27	53.35	56.1
seeds	94.23	94.23	93.75	94.23	95.67
semeion	93.97	93.03	92.78	92.84	89
soybean	38.56	39.96	63.83	70.48	48.4
st-image	97.66	97.79	97.53	97.79	94.19
st-landstat	90.55	90.66	90.5	91.1	85.65
st-shuttle	99.97	99.97	99.88	99.89	97.26
st-vehicle	74.17	77.37	77.25	77.61	82.94
steel plates	77.73	79.02	77.37	78.2	75
synthetic-control	98.17	98.67	99.33	99.5	97.17
teaching	62.5	60.53	62.5	61.84	55.26
thyroid	98.95	98.95	95.6	95.71	93.93
vc-3classes	85.06	84.09	85.39	86.04	83.77
waveform	84.7	84.76	86.2	86.06	86.8
waveform-noise	86.56	86.44	86.24	86.82	86.24

(continued on next page)

Datasets	RaF	RaFL	obRaF	obRaFL	RVFL
wine	98.86	98.86	99.43	99.43	100
w-quar-red	68.31	69	68.06	68.63	59.63
w-quar-white	68.38	68.44	69.24	69.77	55.25
yeast	62.67	63.48	63.14	62.94	60.18
zoo	99	98	99	98	97

We follow the naming convention of [2].

Table 3.3: Average rank values based on classification accuracy of each method.

	RaF	RaFL	obRaF	obRaFL	RVFL
Rank	2.79	2.66	3.12	2.79	3.62

Lower rank reflects better performance.

We also perform a statistical comparison of the algorithms using the Friedman test [117]. The Friedman test compares the average ranks of the classifiers, $R_j = \sum_i r_i^j$ where, r_i^j is the rank of the j -th of the m classifier on the i -th of M data sets. The null hypothesis is that the performance of all the classifiers are similar with their ranks R_j being equal.

Let M and m denote the number of data sets and classifiers respectively. When M and m are large enough, the Friedman statistic

$$\chi_F^2 = \frac{12M}{m(m+1)} \left[\sum_j R_j^2 - \frac{m(m+1)^2}{4} \right], \quad (3.1)$$

is distributed according to χ_F^2 with $m-1$ degrees of freedom under the null hypothesis. However, in this case, χ_F^2 is undesirably conservative. A better statistic is given by

$$F_F = \frac{(M-1)\chi_F^2}{M(m-1) - \chi_F^2}, \quad (3.2)$$

which is distributed according to F -distribution with $(m-1)$ and $(m-1)(M-1)$ degrees of freedom. If the null-hypothesis is rejected, the Nemenyi post-hoc test [118] can be used to check whether the performance of two among m classifiers is significantly different. The performance of two classifiers is significantly different if the corresponding average ranks of the classifiers differ by at least the critical

difference (CD),

$$CD = q_\alpha \sqrt{\frac{m(m+1)}{6M}}, \quad (3.3)$$

where critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$. α is the significance level and is equal to 0.05. By simple calculations we obtain, $\chi_F^2 = 12.54$ and $F_F = 3.24$.

With 5 algorithms and 65 data sets, F_F is distributed according to the F -distribution with $5 - 1 = 4$ and $(5 - 1)(65 - 1) = 256$ degrees of freedom. The critical value for $F_{(4,256)}$ for $\alpha = 0.05$ is 2.4, so we reject the null-hypothesis. Based on the Nemenyi test, the critical difference here is $CD = q_\alpha \sqrt{(k(k+1))/(6N)} = 2.728 * \sqrt{5 * 6 / (6 * 65)} \simeq 0.75$. Based on the average rank values of each method and CD obtained from Nemenyi test, we can conclude that RaF, RaFL, and obRaFL are significantly better than RVFL. Even though obRaF is better than RVFL, the statistical test fails to find any significant difference between them.

3.4.1 Comparison of RaF, obRaF, RaFL and obRaFL

Based on the results so far, although it is difficult to find any significant difference, RaFL works slightly better than the other methods. RaFL is better than RaF in 30 of 65 datasets, worse than RaF in 28 datasets, and equal in 7 sets. When calculating for each dataset the difference between the accuracy of RaFL and RaF, the sum of positive differences (RaFL is better) is 68.73, while the negative ones (RaF is better) sum 48.82. Similarly, obRaFL is better than obRaF in 36 datasets, worse than obRaF in 25 datasets, and equal in 4 sets. Calculating the difference between the accuracy of oRaFL and obRaF, the sum of positive differences (obRaFL is better) is 49.3, while the negative ones (obRaF is better) sum 31.27.

Since RaF performs slightly better than obRaF in multi-class datasets, similar pattern is observed in RaFL and obRaFL. The average accuracy of RaFL and obRaFL over all the data sets are 80.63% and 80.62% respectively. RaFL has the highest average accuracy among all the 4 methods followed by obRaFL.

From the results obtained in Table 3.2, it is worth noticing that RaFL and obRaFL generally under-perform in datasets with sample size $N < 500$. Out of 65 data sets, there are 28 such datasets. RaF performs better than RaFL in 16 datasets

out of 28, worse in 7 data sets, and equal in 5. Similarly, obRaF is better than obRaFL in 15 datasets, worse in 10, and equal in 3. 57.14% (16 out of 28) of RaF's overall wins over RaFL emanate from such datasets while for obRaF, it is 60%. On further investigating the small size datasets, we notice that RaFL is better than or comparable to RaF only in balanced (number of samples from all the classes are equal) or slightly unbalanced (disparity between the number of samples from different classes) datasets. On highly unbalanced datasets like arrhythmia (Minority class % = 0.44), glass (Minority class % = 4.2) etc., RaF generally outperforms RaFL. However, on some unbalanced data sets like ecoli (Minority class % = 0.59) and lymphography (Minority class % = 1), RaFL still wins over RaF since there are very few samples from the minority classes in the test data. This analysis is congruent with the performance of obRaF and obRaFL in those data sets. For unbalanced data sets with $N > 500$, we observe no such effect. Considering only those 37 datasets with $N > 500$ (both balanced and unbalanced), we present the overall ranking of each method in Table 3.4.

Table 3.4: Average rank values of 4 methods in data sets with $N > 500$.

	RaF	RaFL	obRaF	obRaFL
Rank	2.55	2.09	3.01	2.33

Lower rank reflects better performance.

On repeating the statistical test for new rank values, we obtain $\chi_F^2 = 7.98$ and $F_F = 2.79$. With 4 algorithms and 37 data sets, F_F is distributed according to the F -distribution with $4 - 1 = 3$ and $(4 - 1)(37 - 1) = 108$ degrees of freedom. The critical value for $F_{(3,108)}$ for $\alpha = 0.05$ is 2.68, so we reject the null-hypothesis. Based on the Nemenyi test, the critical difference here is $CD = q_\alpha \sqrt{(k(k+1))/(6N)} = 2.569 * \sqrt{4 * 5 / (6 * 37)} \simeq 0.77$. Since 2.09 and 3.01 are the only pair of average rank whose difference is larger than 0.77, we can conclude that the RaFL is significantly better than obRaF for data sets with $N > 500$. Although the difference is not significant, obRaFL also works slightly better than RaF. The average accuracy of RaF, obRaF, RaFL and obRaFL over these data sets are 85.38%, 85.6% , 86.22%, and 85.83% respectively.

Furthermore, we use sign test to compare each pair of algorithms as in [10]. Out of N datasets, if the number of wins is at least $N/2 + 1.96\sqrt{N}/2$, the algorithm is significantly better with $p < 0.05$. The tied matches are split evenly between the two classifiers; if there is an odd number of them, we ignore one. The results

are summarized in Table 3.5.

Table 3.5: Significance of the difference between each pairs of algorithms

Method	Significance
(RaF, RaFL)(13,24)	Yes
(RaF, obRaF)(23,14)	No
(RaF, obRaFL)(17,19)	No
(RaFL, obRaF)(25,12)	Yes
(RaFL, obRaFL)(21,15)	No
(obRaF, obRaFL)(10,26)	Yes

The numbers in the bracket represent the number of times each method wins.

From Table 3.5, we can observe that, under the sign test, RaFL is significantly better than RaF and obRaF. obRaFL is also statistically better than obRaF. However, it is important to note that these observations are based on the data sets with $N > 500$.

In a nutshell, our proposed method (RaFL and obRaFL) perform better than RaF and obRaF for those datasets (both balanced and unbalanced) with $N > 500$. RaFL and obRaFL also works better than or comparable to RaF and obRaF with small ($N < 500$) and balanced or slightly unbalanced datasets. The performance of our proposed method is poor when the datasets are small and highly unbalanced. A reason for this is, for small and highly unbalanced datasets, the accuracy of RVFL is generally low. RVFL in our proposed method determines the corresponding decision trees through which a test data needs to be pushed down. A fairly accurate RVFL predicts the first and the second classes with more confidence. That means, the partitions obtained via RVFL approximate the clusters of classes (samples) according to their difficulty in classifying. Hence, the data is likely to be correctly classified by the two decision trees associated with those two subsets. For $N < 500$ and highly unbalanced datasets, RVFL usually gives mediocre performance and the test data is pushed down in the decision trees trained with different data rather than the ones trained with similar data as the test data. Hence, such test data is likely to be misclassified. To corroborate this statement, we present the testing accuracy of RVFL in 6 highly unbalanced datasets (3 data sets with $N > 500$ and the other 3 with $N < 500$) in Table 3.6. The accuracy here, is the average accuracy of an ensemble of RVFL (500 RVFLs) calculated based on the prediction made by RVFL of a sample's true class be-

longing to either the highest or the second highest class. That means, a correct decision is made by RVFL if the test sample's predicted class (either the one with the highest or second-highest scores) matches its true class. This is represented by RVFL (Top 2) in the table. Similarly, we also present the accuracy of RVFL (here, the test sample's predicted class is the one with the highest score) and RaF in these data sets for comparison purposes.

Table 3.6: Average testing accuracy (%) of different methods

	Datasets	RVFL	RVFL (Top 2)	RaF
$(N < 500)$	arrhythmia	68.14	68.73	73.45
	flags	52.6	52.8	66.15
	glass	68.4	77.2	74.53
$(N > 500)$	car	93	97.17	97.16
	ctg-10classes	79.85	86.74	86.72
	nursery	92.85	97.57	99.34

From Table 3.6, we can see that the performance of RaF is better than RVFL and comparable to or also better than RVFL (Top 2) in almost all the datasets. Thus, by using DT after RVFL we are trying to improve the accuracy further. However, RVFL gives mediocre performance for small ($N < 500$) and unbalanced datasets even when selecting Top 2 classes. Because of the RVFL's inferior performance in such small and highly unbalanced datasets, the overall accuracy of the proposed ensemble method in such datasets is also poor. However, for larger datasets, the testing accuracy of RVFL is fair and hence, the performance of the ensemble method is also significantly better than RaF and obRaF.

3.4.2 On the effect of randomizing lambda and the number of hidden neurons

λ and the number of hidden neurons ($\mathcal{N}$) are two important parameters of RVFL. Generally, these parameters are tuned for each dataset. However, in our method, we choose the values of λ (or C) and $\mathcal{N}$ randomly from suitable ranges. Although this may result in non-optimal RVFL models, it may offer some advantages in generating diverse base classifiers fast. This may lead to a better performance by reducing variance of the overall system. The objective of our proposed method is to obtain diverse RVFL models in each base classifier which in turn results

Figure 3.2: Accuracy of RaFL and obRaFL with different values of C

in diverse data partitions hence, diverse decision trees, and thus varying these parameters in each base classifier fits our objective.

To demonstrate the effect of λ (or C) and $\mathcal{N}$ in the accuracy of the ensemble method, we plot a graph of accuracy vs C and accuracy vs $\mathcal{N}$ in two different data sets as shown in Fig. 3.2 and Fig. 3.3 respectively. From the plots, we can see that a careful selection of these parameters is required to obtain reasonable performance or performance close to the ones obtained when randomizing them for different RVFL models. However, as stated earlier, ensemble methods require diversity and such diversity can easily be obtained by randomizing the values of λ and $\mathcal{N}$ which results in diverse RVFL models. On the other hand, tuning RVFL for best values would increase the computational time and complexity without any significant improvement in the performance.

3.4.3 Computational Complexity

The objective here is to investigate the complexity introduced by the hybridization instead of comparing the computational complexities associated with RaF and obRaF. In Table 3.7, we summarize the average number of nodes in a decision tree in each case. First 14 datasets (see Table 3.2) are selected for this purpose. However, similar results pertain to other datasets too. Intuitively, we can state that the decision trees in RaFL and obRaFL are relatively smaller compared to those in RaF and obRaF because the trees in RaFL and obRaFL are grown with

Figure 3.3: Accuracy of RaFL and obRaFL with different number of hidden neurons

reduced dataset. One may argue that the smaller decision trees are unstable and hence, may lead to increased bias. However, the nature of construction of trees (multiple branches at the root node) in our proposed method is able to alleviate this problem. In Table 3.8, we report the training time for each ensemble. However, the code for the proposed ensemble method is not optimized for speed. From the table, we can infer that our proposed method gives an improved result compared to their standard counterparts in expense of additional computational time. The trees here are grown sequentially, however with the use of multi-core and parallel/distributed architectures in CPU and GPU, the training time would decrease significantly.

3.5 Conclusion

In this chapter, we presented a new ensemble of classifiers based on decision trees and random vector functional link networks. Since our method evolves from RaF and RVFL, we named our proposed method as RaFL (RVFL with axis-parallel trees) and obRaFL (RVFL with oblique trees). In each base classifier of RaFL and obRaFL, decision trees were induced based on the samples partitioned by RVFL. We evaluated the performance of our proposed methods on 65 multi-class data sets from UCI repository. RaFL and obRaFL performed better than other methods (RaF, obRaF, RVFL) for datasets with relatively high number of samples

Table 3.7: Average number of nodes in a base classifier of each ensemble.

Datasets	RaF	RaFL	obRaF	obRaFL
Abalone	533.01	373.13	713.8	500.38
Annealing	44.54	20.818	78.51	35.28
Arrhythmia	52.94	7.8	92.33	12.56
Audiology-std	28.52	2.96	40.52	3.96
Balance-scale	60.02	44.85	60.21	44.79
Breast-tissue	19.18	4.89	19.18	6.32
Car	81.8	47.68	84.13	50.51
Contrac	229.35	166.44	148.55	120.57
Ctg-10classes	190.28	41.46	329.42	68.14
Ctg-3classes	148.56	61.69	148.56	102.95
Chess-krvk	1942.68	346.95	2801.54	435.04
Conn-bench-vowel	78.72	16.21	114.63	22.39
Dermatology	21.49	9.23	27.62	11.52
Ecoli	29.67	8.16	42.33	10.93

Table 3.8: Average training time (in seconds) for each ensemble method

Datasets	RaF	RaFL	obRaF	obRaFL
Abalone	1472.49	2370	382.51	820.45
Annealing	114.52	263.02	62.69	101.23
Arrhythmia	433.11	530.02	96.48	263.51
Audiology-std	32.33	41.2	45.71	51.34
Balance-scale	37.2	75.59	43.4	105.37
Breast-tissue	12.78	34.4	15.06	34.11
Car	68.03	120.52	59.09	140.05
Contrac	212.73	351.94	86.47	319.46
Ctg-10classes	630.36	927.31	221.63	816.66
Ctg-3classes	95.91	640.76	86.47	346.21
Chess-krvk	2290.29	5559.42	2936.18	5940
Conn-bench-vowel	273.09	310.68	86.92	314.06
Dermatology	35.91	64.61	19.15	60.11
Ecoli	52.57	71.89	27.71	50.52

($N > 500$). Overall, RaFL is better than the rest of the methods followed by obRaFL. RaFL and obRaFL also achieved higher average accuracy than RaF, obRaF and RVFL.

Chapter 4

Oblique Random Forest

In this chapter, we will first present one-vs-all based oblique random forests. In such oblique random forests, the training data at each node is partitioned using one-vs-all technique and a linear classifier is employed at each partition to generate an oblique hyperplane. The linear (oblique) split that best minimizes the impurity criterion is then selected as the splitting hyperplane. Oblique random forests with several linear classifiers are constructed. Next, we will present a heterogeneous oblique random forest that employs diverse linear classifiers at each node on some candidate partitions generated using one-vs-all and hyperclasses based technique. The latter oblique random forest expands the search space for the best oblique split while using multiple linear classifiers at some top ranked partitions. A preliminary version of this idea was implemented in [25] and extended in [119].

4.1 Introduction

A candidate hyperplane at each internal node of the decision tree is usually rated based on an impurity criterion. If the class distribution at a node is nearly uniform, then the node is highly impure and if a class has predominance over others, then the impurity is low. Given a set of training samples and a hyperplane at a node, it is straightforward to identify the set of training samples that fall in the left and the right child nodes of this parent node [120]. The impurity criterion gives higher rating to a hyperplane which results in higher purity of child nodes. However, impurity criterion commonly used in decision trees such as gini index, information gain and towing rule are not differentiable with respect to hyperplane

parameters [19]. So, decision trees use some greedy search techniques for finding the best hyperplane at each node. The decision trees in random forest exhaustively search for a single feature or the best axis-parallel hyperplane. However, such exhaustive search for the best oblique hyperplane is computationally expensive [20]. Thus, linear hyperplanes are used in oblique decision trees without optimizing any criteria or the search for the oblique splits are generally based on heuristic approaches and are non-optimal. To circumvent such issue, we present oblique decision trees that search for an optimal linear hyperplane at each node from a finite pool of competitive hyperplanes by optimizing the Gini-impurity criterion similar to the trees in RaF.

In standard (axis-parallel) decision trees, the optimal split at each node is chosen from the $n \cdot q$ possible splits, where n is the unique feature values in $\mathbf{x}$ and q is typically set to the square root of feature dimension d . That means, the exhaustive search for the ‘optimal’ orthogonal split grows linearly in the number of training samples at each node and q . However, there are at most $2^q \cdot \binom{n}{q}$ distinct oblique splits at each node and the exhaustive search of the ‘optimal’ oblique split is NP-hard [20].

From extensive applications of random forests in many diverse domains, it is evident that the recursive partitioning of the training data while optimizing some impurity criterion aid random forest to generalize better and by the virtue of subspace and ensemble method, RaF is able to achieve state-of-the-art performances. As reported in [19], employing linear classifiers in each internal node can lead to both “accurate” and “diverse” decision trees which is of vital importance for the success of random forests [8, 121]. This is further verified by the large-scale benchmarking with state-of-the-art classifiers in [2, 10]. However, employing a single type of linear classifier may not always be optimal due to the No-Free-Lunch theorem [122]. Such a limitation can be curbed by employing a simple strategy: selecting one best linear classifier from a pool of diverse linear classifiers. Depending on the dataset or subset of training samples at each node, one type of hyperplane may be best suited for partitioning the samples at that particular node. That means, such hyperplane can best optimize the impurity criterion compared to the others. Hence, selecting the one from a pool using an impurity criterion endows the random forest to generalize better with more “accurate” and “diversified” tree structure. In this chapter, we employ six linear classifiers namely Support Vector Machines (SVM), Multisurface Proximal SVM (MPSVM), Linear Discriminant Analysis (LDA), Least Squares SVM (LSSVM), Ridge Regression (RR) and

Logistic Regression (LR) due to their good performance in several domains [2]. Extension of the proposed method to other linear classifiers is straightforward. The key contributions of this chapter are summarized as follows:

- First, we propose one-vs-all based oblique random forest using six linear classifiers.
- The one-vs-all based formulation requires evaluating K decision boundaries generated by a linear classifier where K is the number of classes at each node. An oblique random forest employing n linear classifiers requires nK evaluations at each node. Such procedure is computationally expensive when the number of classes is very large. Also, such one-vs-all based partitioning results in unbalanced trees while limiting the searches to only K evaluations. Thus, we expand the search space of the oblique splits by considering both one-vs-all and two-hyperclasses based partitioning. Intuitively, this increases the computational cost. To avoid evaluating a large number of partitions, we use either an optimization algorithm or an exhaustive search depending on the number of classes at each node in conjunction with ideal gini score and cluster separability to selectively filter the partitions and employ linear classifiers only on the top ranked partitions.
- We then, propose a heterogeneous oblique random forest that searches for the best splitting hyperplane at each node by optimizing the impurity criterion as in standard random forest. Unlike conventional oblique random forest that employs only one type of linear classifier for a splitting hyperplane at every node, the heterogeneous oblique random forest employs diverse linear classifiers throughout the nodes of the trees that can better exploit the geometric structure in the data. The proposed approach is generic and any linear classifier can be used in our framework.
- The trees in the proposed oblique random forests proposed are generally shallow and require less training time compared to standard random forests with the exception of the heterogeneous oblique random forest which requires more training time.
- In extensive experiments on 121 UCI machine learning datasets, we find that 3 (out of 7) of our proposed oblique random forest variants take the top 3 ranks by outperforming standard (axis-parallel) random forest and oblique random forest variants and hundreds of other classifiers [2, 10].

4.2 ObRaF with linear and non-linear hyperplanes

As the search for the optimal oblique hyperplane is cumbersome, many heuristic search methods based ObRaF such as Forest-LC [40], OC1 [20] have been proposed in the literature. Apart from these heuristic approaches, linear classifiers such as SVM, MPSVM, LDA, LR have also been used to generate oblique hyperplanes. In [56, 60], LDA is used to generate linear splits. However, they consider only binary classification problems. In [59, 123], SVM is used to generate a linear split at each node. They either employ kernel clustering algorithm [59] or some distance metrics such as Euclidean/Mahalanobis [123] to break down multi-class classification problems into binary classification problems. [19] employs MPSVM to generate linear split at each node. It uses Bhattacharyya distance to group multiple classes into two hyperclasses. Among the two clustering hyperplanes generated at each node, the one that best optimizes the impurity criterion is selected. However, the hyperplane selection is restricted between the two hyperplanes. In this chapter, we increase the search space for the best linear split and also employ diverse linear classifiers. [61] employs logistic regression in several partitions ($2^{K-1} - 1$ partitions where K is the number of classes at each node) and selects the one that best optimizes the impurity criterion. However, such procedure is computationally expensive for large number of classes. All of these approaches either do not optimize impurity criterion or do not perform any search for the optimal oblique hyperplanes or are computationally expensive. The computational complexity associated with the linear classifiers impede the search or optimization process. In [124, 125], the authors propose to compute the weight coefficients of the oblique hyperplane at each node by continuously optimizing the log loss function of each node by means of gradient descent. However, several hyper-parameters need to be tuned at each node which greatly increases the computational complexity of the trees (complexity multiplies when growing the forest) without any significant gain in the performance. Similarly, in [49], a randomized multi-layer perceptron is used as a splitting function at each node of the tree. However, selecting the neural network's model complexity is an issue here since, using a much more complex model might result in overfitting. Thus, the authors propose to use a heuristic approach to select the neural network hyper-parameters that depend on the number of training samples reaching a particular node. This method is also computationally expensive as it has to train a neural network at each node using back-propagation and such method may not result in optimal network parameters at the lower nodes where there are only few samples. In [44], the decision nodes

do not perform hard routing as in standard decision trees but instead perform soft (stochastic) routing and the node (linear split) parameters are obtained by gradient descent based approach by minimizing the global loss function. Our proposed approach integrates an impurity optimization technique for optimal oblique hyperplane search in ObRaF as in standard RaF without incurring great computational costs by evaluating numerous partitions before actual hyperplane generation.

4.3 ObRaF based on one-vs-all approach

Many popular binary classifiers such as SVM use “one-vs-all” approach to break-down a multi-class classification problem into several binary classification problems. A caveat is that it may not always be the best method to deal with multi-class problems. Such methods can however, be integrated intact at the internal nodes of the decision trees. A linear classifier at each node doesn’t need to be a perfect classifier but simply aid in further classification or partitioning. Thus, at each node of the decision tree, a linear classifier can be employed to separate one class from all other classes. That means, the best linear hyperplane at each node can be obtained by searching amongst the ‘ K ’ hyperplanes obtained by transforming a multi-class classification problem into ‘ K ’ binary classification problems¹. Thus, a linear decision boundary can be selected at each node that best optimizes the impurity criterion. In the best scenario, a linear classifier may result in all samples of a class in one child node while the rest of the training samples in the other child node which is exactly the objective of decision trees i.e. to make child nodes purer than the parent nodes. Thus, employing linear hyperplanes with impurity optimization technique can help to better exploit the geometric structure of the data than axis-parallel hyperplanes.

At each node of the tree, K oblique hyperplanes generated using a linear classifier are compared and the one that best optimizes Eq. (4.1) is selected as the splitting hyperplane. The optimization problem is given as:

$$\max(Gini_p - Gini_c), \quad (4.1)$$

¹The total number of classes in a dataset is denoted by C and the number of classes at each node by K . $C \geq K$ where $C = K$ at the top root node.

Algorithm 1 Oblique Decision Tree (obDT) Training [25]

```

1: Require: Labeled training data  $(X, Y)$ 
2: Require: Maximum tree depth  $\mathcal{D}$ 
3: Output: Linear decision boundaries ( $w$ 's and  $b$ 's)
4: for  $d_T$  from 1 to  $\mathcal{D}$  do
5:   Check stopping criteria at each node at depth  $d_T$ .
6:   Create  $K$  data partitions based on one-vs-all approach.
7:   Train a linear classifier on each partition.
8:   Compute Eqs. (4.2) and (4.3).
9:   return  $w$  and  $b$  of the hyperplane that best optimizes Eq. (4.1) at each
       node at depth  $d_T$ .
10: end for

```

where

$$Gini_p = 1 - \sum_{i=1}^K \left(\frac{n_{w_i}}{n_t} \right)^2, \quad (4.2)$$

$$Gini_c = \frac{n_t^l}{n_t} \left[1 - \sum_{i=1}^K \left(\frac{n_{w_i}^l}{n_t^l} \right)^2 \right] + \frac{n_t^r}{n_t} \left[1 - \sum_{i=1}^K \left(\frac{n_{w_i}^r}{n_t^r} \right)^2 \right] \quad (4.3)$$

where $Gini_p$ and $Gini_c$ are the values of Gini impurity at the parent node and child nodes respectively, n_t is the number of data samples in the parent node, n_t^l, n_t^r is the number of data samples that reach the left and right child nodes of the current parent node, and $n_{w_i}^l, n_{w_i}^r$ are the number of samples of class w_i in the left and right child nodes respectively.

The construction algorithm of the one-vs-all based oblique decision tree is presented in Algorithm 1 [25].

One of the issues with the “one-vs-all” technique is that it is computationally expensive when the number of classes is very large. Thus, generating K hyperplanes at each node can be computationally expensive in such cases. Since the linear hyperplane at each node is selected from a pool of hyperplanes based on the impurity criteria (more specifically Eq. (4.1)), it results in pure child nodes faster than previous exhaustive approaches (in case of RaF) and obRaF without impurity optimization approaches. Thus, the trees in such ‘one-vs-all’ oblique random forest variants are generally shallow than the standard trees [25]. This may negate the complexity associated with generating many hyperplanes at each node and it would also improve the classification testing time.

4.4 Heterogeneous oblique random forest

Different linear classifiers employ different techniques to generate a linear decision boundary (maximum margin hyperplane in case of SVM, clustering hyperplane in case of MPSVM, etc.). One family of linear classifier may not always create the best partition or exploit the geometric structure of the data as the decision boundary depends on the training samples falling at each node. By using different partitions at each node, we can search for the best linear decision boundary for that node among several linear decision boundaries and such boundaries may be obtained from different families of linear classifiers. Thus, we employ several linear classifiers viz., SVM, MPSVM, LDA, LSSVM, RR and LR with decision trees to construct several variants of one-vs-all based oblique decision trees at first.

Based on our experimental results, we observe that, even though some of the linear classifier based oblique random forests consistently perform better (have high ranks compared to others), they are not always the best oblique random forest variants for every dataset. This is congruent with the famous No-Free-Lunch theorem [122] which states that no single classifier is the best for every dataset. Thus, even though some of the linear classifier based variants have lower ranks compared to others, we can still integrate them to form a heterogeneous linear classifiers based oblique random forest [119]. Based on our original formulation as described in 4.3, this would require evaluating n linear classifiers in K binary partitions thus, requiring nK evaluations at each node. Also, in theory, it is highly desirable to obtain highly balanced trees but in practice, it is extremely difficult to guarantee balanced trees [126]. An innate issue with one-vs-all based decision trees is that it leads to heavily unbalanced trees. Another issue with such one-vs-all based partitioning method is that it limits the search space of the optimal oblique splits to K choices. Even though the exhaustive search of the optimal oblique split is NP-hard, it is beneficial to increase the search space of the oblique splits to obtain the best split. Thus, to address or solve the above issues, we employ a hyperclasses based partitioning along with one-vs-all partitioning while using several linear classifiers at each node as described in the subsequent sections.

4.4.1 Generating several partitions and their ranking

In standard decision trees, each axis-parallel split is rated based on an impurity criterion. Specifically, all the splits at each non-leaf nodes are associated with

an impurity measure (we denote this by $g(i) = Gini_p - Gini_c$ using Eq. (4.2) and Eq. (4.3)) and the one with the maximum value (cf. Eq. (4.1)) is the selected split ('best' split) at that particular node. However, an exhaustive search for the optimal oblique split is NP-hard. To extenuate an exhaustive search, we restricted the search space to limited oblique splits in one-vs-all based oblique random forests. We employed a linear classifier to generate K hyperplanes via K one-vs-all partitions and selected the one based on Eq. (4.1). In other words, instead of looking for the optimal oblique split in the whole search space, we utilized the recursive partitioning property of decision trees (separating samples of one class from all other classes) to generate few oblique splits and compared their $g(i)$'s for the selection of the best oblique split. We observed that in case of one-vs-all partitions, partitions with higher positive training samples (large number of training samples for the one class) basically had higher $g(i)$ values compared to other partitions². This is intuitive since separating large number of samples of one class produces purer child nodes (cf. to Eq. (4.2) and Eq. (4.3)). In a nutshell, we can employ the information available at each node to obtain the best split.

Generally, in case of oblique decision trees, a multi-class classification problem is treated as binary classification problem at each node. Thus, the purity of child nodes depends on the partitioning of the classes and the quality of the split. For the latter, we employ several linear classifiers as discussed in the subsequent section. For the former, we combine the one-vs-all partitioning idea of [25] and hyperclasses based partitioning approach of [19, 61]. In hyperclasses based partitioning, several classes are grouped in two hyperclasses (say c_1 and c_2) based on a criterion. For example, with 5 classes $\{1, 2, 3, 4, 5\}$ at a node, they can be grouped into two hyperclasses $c_1 = \{1, 2, 5\}$ and $c_2 = \{3, 4\}$ based on some criteria. [19] uses the Bhattacharyya distance between the classes to group them into two classes while [61] simply uses all possible choices of hyperclasses.

At each node of a decision tree, there are $2^{K-1} - 1$ possible ways of partitioning K classes [61]. For example, if there are 10 classes at a node, these classes can be arranged in $2^{10-1} - 1 = 511$ partitions. For large number of classes, the number of partitions is rampant. Thus, in this thesis, we selectively filter the partitions and use only the top ranked partitions to train the linear classifiers. Specifically, we use ideal gini score and clustering validity index to rank the partitions. Rank-

²We call the 'one' class as positive class and the 'all' class as the negative class in case of one-vs-all partitioning.

ing the partitions enable us to use few significant partitions while avoiding the insignificant partitions (those likely to have small $g(i)$). Before delving into the partitioning approach and ranking, we first introduce the two criteria used to rank the partitions.

4.4.1.1 Ideal Gini score

The ideal gini score refers to the gini value obtained in the ideal case where an oblique split perfectly separates all the samples of one class from the other class. Note that, in this chapter, we breakdown a multi-class problem into binary class problem recursively. Thus, in one-vs-all partitioning the ideal case is equivalent to separating all samples of the positive class from the rest of the classes while in hyperclasses based partitioning, it is equivalent to separating all the samples of one hyperclass from the other. Computing ideal gini score does not require generating a hyperplane but simply computing $g(i)$ assuming the scenario when all the samples of one class (or hyperclass) are sent to the right node and all others to the left node ³. A higher gini score means purer child nodes. If a split with high ideal gini score (g_I) is able to replicate its g_I , it is likely to have higher actual gini score (g). Computing g_I beforehand, without actually training a linear classifier gives us an indication which partition the oblique split should focus on. By training the linear classifiers on the partitions with higher g_I which are likely to give higher g , we can ignore the partitions with lower g_I which are likely to give lower g .

4.4.1.2 Cluster separability

g_I is a measure of the purity of the child nodes in an ideal setting with the assumption that the partitions are well-separated and a linear decision boundary can exactly classify or separate them. However, in the real settings, this may not always be true. Thus, we also need to consider the geometric structure of the data at each node when ranking the partitions. A partition may have high g_I but low g if the partitions are difficult or hard to separate. Thus, we use a cluster validity index to evaluate the separability of each partition or cluster.

³We represent the ideal gini score $g_I(i)$ as g_I and actual gini score $g(i)$ as g respectively and interchangeably throughout the chapter hereafter. We also use the terms partitions and hyperclasses interchangeably throughout the chapter.

There are several cluster validity indices in the literature. The cluster validity indices aim to identify the sets of clusters that are compact and well-separated. Since we are interested in binary decision trees or binary partitioning, it is highly desirable that the two partitions are well separated. Dunn index (DI) [127] is one of the popular cluster validity index. The Dunn index for k clusters is defined by:

$$DI_k = \min_{i=1,\dots,k} \left\{ \min_{j=1+i,\dots,k} \left(\frac{diss(c_i, c_j)}{\max_{m=1,\dots,k} diam(c_m)} \right) \right\}, \quad (4.4)$$

where $diss(c_i, c_j) = \min_{x \in c_i, y \in c_j} \|x - y\|$ is the dissimilarity between the clusters c_i and c_j (also known as inter-cluster distance) and $diam(C) = \max_{x,y \in C} \|x - y\|$ is the intra-cluster function (diameter) of the cluster. A higher DI indicates better clustering. We use Bhattacharyya distance as the distance metric as it is considered a good measure for class separability between two classes [19]. The Bhattacharyya distance between two classes c_1 and c_2 with means and co-variances (μ_1, Σ_1) and (μ_2, Σ_2) respectively, is defined as:

$$B(c_1, c_2) = \frac{1}{8}(\mu_2 - \mu_1)^T \left(\frac{\Sigma_1 + \Sigma_2}{2} \right)^{-1} (\mu_2 - \mu_1) + \frac{1}{2} \ln \frac{|(\Sigma_1 + \Sigma_2)/2|}{\sqrt{|\Sigma_1| |\Sigma_2|}} \quad (4.5)$$

Since we only consider binary partitions and their separability, the two partitions need to be well separated but not necessarily compact, thus, we can ignore the denominator part of DI i.e. the diameter of the cluster. The inter-cluster distance is the distance between the two closest classes, one in each partition. Because of the recursive partitioning in decision trees, it is highly likely to have only a few samples of some classes at a node. Typically, in a one-vs-all based partitioning, the separability of the partitions is affected when the class (with only a few samples) belonging to the negative class is very close to the positive class. This plummets the separability of the partitions even when the partitions are well-separated barring the few samples. In such a scenario, the DI associated with such partition is low even when g_1 or g can be high. To alleviate such issue, we take the weighted mean of the distances of the closest and the second closest classes of the negative class with the positive class while computing the inter-cluster distance.

The DI criterion complements the g_1 criteria. That means, well separated partitions with higher g_1 are likely to have higher g or purer child nodes compared to other partitions. We demonstrate this using a toy example as shown in Figure 4.1. The toy example consists of 4 classes $\{1,2,3,4\}$ at a node. The samples of

each class are normally distributed with different means for each class. With 4 classes, there are 7 ($2^{4-1} - 1$) ways of partitioning them as shown by the seven plots. For each partition, we compute the g_I and DI and sort them based on their final rank (obtained via averaging the two ranks). We then train an LDA to find a linear decision boundary to separate the partitions and compute g afterwards. Based on Figure 4.1, we can see that the top ranked partitions (with higher g_I and DI) actually have higher g . On close observations, we can also see that even though the partition $\{1,3\}\{2,4\}$ has higher g_I than the partition $\{4\}\{1,2,3\}$ (cf. Figure 4.1(b) and Figure 4.1(c)), the partition $\{4\}\{1,2,3\}$ is well-separated than the partition $\{1,3\}\{2,4\}$. On averaging the two ranks obtained via g_I and DI, the partition $\{4\}\{1,2,3\}$ is ranked higher than the partition $\{1,3\}\{2,4\}$. On computing g after obtaining the decision boundary, we can see that the partition $\{4\}\{1,2,3\}$ has higher g than partition $\{1,3\}\{2,4\}$. Similar explanation pertains to Figure 4.1(e) and Figure 4.1(f). This shows that we can sort the partitions by averaging the ranks obtained via g_I and DI criteria and is useful to filter out the insignificant partitions as discussed in the next section.

4.4.1.3 Employing Exhaustive search or Genetic Algorithm

Depending on the number of classes at each node, the partitions are obtained through either Genetic Algorithm (GA) or exhaustive search. Genetic Algorithm (GA) is a computational method used to solve optimization and search problems by mimicking biological evolution via bio-inspired operators such as mutation, crossover and selection [128]. In GA, a population of candidate solutions to a problem is evolved towards better solutions at each iteration. The population in each iteration is called a generation. Each individual solution is normally composed of strings of ones and zeros that can be mutated and altered. The evolution usually starts from a population of randomly generated individuals. In each generation, the fitness (objective function of the optimization problem) of every individual solution in the population is evaluated. The more fit individuals (also known as parents) are selected from the current population and are used to produce the children via crossover and mutation for the next generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. The algorithm terminates when either a maximum number of generations have been produced, or a satisfactory fitness level has been reached for the population. For more details on GA, readers can refer to [128].

(a) Rank 1, $DI = 18.99$, $g_I = 0.344$, $g = 0.344$ (b) Rank 2, $DI = 27.95$, $g_I = 0.291$, $g = 0.291$ (c) Rank 3, $DI = 14.51$, $g_I = 0.308$, $g = 0.272$ (d) Rank 4, $DI = 9.3$, $g_I = 0.281$, $g = 0.256$ (e) Rank 5, $DI = 11.49$, $g_I = 0.114$, $g = 0.11$ (f) Rank 6, $DI = 6.01$, $g_I = 0.215$, $g = 0.106$

(g) Rank 7, $DI = 8.39$, $g_I = 0.213$, $g = 0.067$

Figure 4.1: Ranking of different partitions based on their ideal gini values g_I and dunn index DI . LDA is used at each partition to generate a decision boundary. g_I refers to the ideal gini scores computed beforehand to rank the partitions while g refers to the actual gini scores after generating the decision boundary. The numbers within the curly braces represent the binary partitions. db means decision boundary. Best viewed in color.

We evaluate an equal number of one-vs-all partitions and hyperclasses based partition at each node. For example, if there are K one-vs-all partitions, we obtain another top K hyperclasses based partition from GA or exhaustive search and re-rank these $2K$ partitions. We do this because for K classes, there are $2^{K-1} - 1 - K$ possible hyperclasses based partitions. For large number of classes, the number of hyperclasses based partition drastically increases. For 11 classes, there are 1012 hyperclasses based binary partitions and for 100 classes, there are 6.33×10^{29} possible partitions! To avoid evaluating a colossal number of partitions, we employ the idea of [19]. In [19], several classes at each node are grouped into two hyperclasses based on the pairwise Bhattacharyya distance between the classes. Such partition may have a higher DI but not necessarily higher g_I . Thus, we introduce a slight variation in this approach. Specifically, keeping the two furthest classes obtained from the pairwise Bhattacharyya distance approach fixed, we randomly assign 50% (value used in our experiment) of the classes mid-way between the two furthest classes to either of the partitions. We rank all these partitions based on g_I and DI and select the top K hyperclasses based partitions. When $K \geq 8$, we employ GA to select these partitions as shown in Algorithm 2 and Figure 4.2.

Figure 4.2: Flowchart explaining GAOREXHSEARCH function of Algorithm 2.

Figure 4.3: For the GA based partitioning, we create the initial population by randomly assigning the classes inside the rectangular box (represents the 50% classes) mid-way between the two furthest classes ext1 and ext2 obtained using Bhattacharyya distance to either of the partitions.

4.4.2 Employing several linear classifiers at each node

Employing a single type of linear classifier at a node may not yield the best partition. Such a limitation can however, be remedied by applying several diverse linear classifiers. Depending on the training samples at each node, one type of hyperplane (from many) may be best suited for partitioning the samples at that particular node. We employ several diverse linear classifiers at each node resulting in heterogeneous oblique decision tree. To elucidate the motivation behind using several linear classifiers, we use one of the partitions of the toy example described above. Specifically, we use the third ranked partition $\{1,3\}\{2,4\}$ and employ three different linear classifiers LDA, MPSVM and LR to generate three decision boundaries. As can be seen in Figure 4.4, all three classifiers generate different decision boundaries to separate the partition with the decision boundary generated by LR having the highest g . Since the selection of the split is based on the one

Figure 4.4: Decision boundaries generated by different linear classifiers and their associated g . Best viewed in color.

with the highest g , the decision boundary generated by LR is selected as the best oblique split for that particular node.

As discussed in previous sections, even for 10 classes at a node, the total number of possible partitions is 511. Using all 6 classifiers (in our case) would mean generating 3066 different decision boundaries at that particular node and comparing them. This count increases with the increase in the number of classes, tree depth and the number of linear classifiers used. To eschew this astronomical computational burden, we only employ linear classifiers in some selective partitions based on their ranking. Even though it is compelling to use all linear classifiers only at the top ranked partition to reduce the computational requirements, it is intuitive to select some few top ranked partitions to make the algorithm robust.

The overall algorithm and flowchart for constructing a heterogeneous oblique decision tree is presented in Algorithm 2 and Figure 4.5 respectively.

Since decision trees employ recursive partitioning, the number of training samples reaching the nodes closer to the leaf nodes are very few. Because of very few samples, the co-variance matrix Σ (cf. Eq. (2.8)), or the matrices G and H (cf. Eq. (2.15)) may not always be positive definite. This is commonly known as the small sample size (SSS) problem. In case of decision trees, to deal with this issue, two regularization approaches can be used: Tikhonov regularization and axis-parallel split regularization [19]. We employ the axis-parallel split regularization in our oblique random forest variants as it requires no additional parameter to

Figure 4.5: Flowchart depicting our proposed method at each node

Algorithm 2 Heterogeneous Oblique Decision Tree Training

```

1: Require: Labelled training data  $(X, Y)$ 
2: Require: Maximum tree depth  $\mathcal{D}$ 
3: for  $d_T$  from 1 to  $D_{max}$  do
4:   Check the stopping criteria for all nodes at depth  $d_T$ .
5:   if  $K \geq 5$  at a node then
6:     Use GAOREXHSEARCH function to get the  $K$  hyperclasses based
7:     partitions and also create  $K$  one-vs-all partitions.
8:     Sort the  $2K$  partitions obtained in line 6 using their average ranks
9:     obtained via both  $g_I$  and DI.
10:    Repeatedly divide the sorted partitions into halves and train the
11:    linear classifiers subsequently on the upper half to get several linear
12:    decision boundaries and compute  $g$ . (refer to Section 4.5.2.1)
13:    If the number of partitions in the upper half  $\leq 5$ , train remaining
14:    linear classifiers on all remaining partitions.
15:  else
16:    Train all linear classifiers on all possible partitions and compute  $g$ .
17:  end if
18:  Select the decision boundary with the highest  $g$  as the splitting hyper-
19:  plane at depth  $d_T$ .
20: end for

```

```

1: function GAOREXHSEARCH ▷ refer to Sections 4.4.1.3 and 4.5.2.1
2:   if  $K < 8$  then ▷ Employ Exhaustive Search
3:     Generate all possible hyperclasses based partitions.
4:     Sort all the partitions obtained in line 3 based on their average
5:     ranks obtained via  $g_I$  and DI.
6:   else ▷ Employ Genetic Algorithm
7:     Compute the pairwise Bhattacharyya distance between the classes.
8:     Find the two classes with largest distance and label them ext1 and
9:     ext2 respectively.
10:    Assign every remaining classes to ext1 and ext2 depending on their
11:    proximity to these classes.
12:    Create initial population by randomly assigning 50% of the classes
13:    mid-way between ext1 and ext2 to either one of the them.
14:    for  $i$  from 1 to  $maxGeneration$  do
15:      Employ GA and sort all the partitions at each generation using
16:      their average ranks.
17:    end for
18:  end if
19:  return the top-ranked  $K$  partitions
20: end function

```

tune and also gives good generalization [10]. Thus, the oblique decision boundary reverts to axis-parallel split when no optimal linear hyperplane can be generated.

4.5 Experiments

4.5.1 Machine Learning Datasets

We compare the classifiers on 121 datasets from the UCI repository as done in [2, 10]. Details of the datasets are presented in Table A1 in the Appendix.

4.5.2 Experimental Settings

For random forest based methods, the number of trees is set to 500. Only the number of features randomly selected at each node, m_{try} , is tuned as in [2, 10]. For linear classifiers such as SVM, RR and LSSVM there is an additional regularization term whose value is randomly selected from the ranges specified in [2]. Specifically, the parameter C of SVM is chosen from the interval $2^{-2}, \dots, 2^7$, λ of RR from $10^{-5}, \dots, 10^5$ and γ of LSSVM from $2^{-5}, \dots, 2^5$. Based on our experiments, tuning the regularization term at each node is computationally expensive as there exist hundreds and thousands of nodes in a single decision tree and there is no significant performance gain. Because decision trees perform recursive partitioning and the trees are fully grown until pure child nodes are obtained, random selection approach is suitable for such methods [10, 129]. We also observed that this approach fosters diversity within the trees (due to hyper-parameter randomization) which is an important criterion for the construction of ensemble methods.

4.5.2.1 Configuration of Heterogeneous Oblique Decision Tree

For the heterogeneous oblique decision tree, we employ all linear classifiers used in this chapter in the order of LDA, MPSVM, LSSVM, LR, SVM and RR. These classifiers are ordered based on their performance. These ordered classifiers are subsequently used (beginning with LDA and followed by the rest) in the top ranked half obtained after repeated division of the sorted partitions. For the GA based partitioning, the population size is set to $2 \times K$ while maximum generation is set to 5. We empirically found that setting these values leads to satisfactory results.

4.5.3 Performance Comparison

Table 4.1 reports the classification accuracy of each random forest based classifier on each dataset. From the table, we can see that the oblique random forests with linear decision boundaries generalize better than the random forest with axis-parallel boundaries. Also, the number of wins is distributed across various oblique random forest variants which motivated us to construct the heterogeneous oblique random forest. We follow the procedure of [2, 10], and use the rank of each classifier to depict its performance. In this approach, each classifier is ranked based on its performance, that means, the highest performing classifier is ranked 1, the second highest rank 2,..., average ranks are assigned in case of ties.

In Table 4.2, we present the mean ranks of the classifiers over all 121 UCI datasets. For binary datasets, ObRaF and obRaF(M) are the same. Thus, when computing the mean rank we use the same accuracy values for these methods and both are assigned average ties. As can be seen, the heterogeneous oblique random forest variant, obRaF(H) obtains the best rank followed by obRaF(L) and obRaF(M). We also perform a statistical comparison of the algorithms using the Friedman test (cf. Chapter 3).

Based on simple calculations we obtain, $F_F = 11.23$. With 9 classifiers and 121 datasets, F_F is distributed according to the F -distribution with $9 - 1 = 8$ and $(9 - 1)(121 - 1) = 960$ degrees of freedom. The critical value for $F_{(8,960)}$ for $\alpha = 0.05$ is 1.94, so we reject the null-hypothesis. Based on the Nemenyi test, the critical difference is $CD = q_\alpha \sqrt{(n(n + 1))/(6N)} = 3.102 * \sqrt{9 * 10/(6 * 121)} \simeq 1.09$. From Fig. 4.6, we can see that obRaF(H) is statistically better than standard random forest and oblique random forest.

Table 4.1: Accuracy (%) of Different Random Forest Based Methods

Datasets	RaF	obRaF	obRaF(L)	obRaF(RR)	obRaF(S)	obRaF(LS)	obRaF(LR)	obRaF(M)	obRaF(H)
abalone	65.61	65.49	65.83	65.45	65.35	65.23	65.42	65.95	65.95
ac-inflam	100	100	100	100	100	100	100	100	100
acute-nephritis	100	100	100	100	100	100	100	100	100
adult	86.11	85.96	85.97	85.52	85.58	85.92	85.9	85.96	86.1
annealing	76.38	76.38	76.38	95.98	96.86	76.38	76.38	76.38	96.23
arrhythmia	77.21	76.99	78.1	70.35	72.12	66.15	72.79	77.01	78.68
audiology-std	81.98	82.56	81.4	84.88	84.3	84.88	85.47	81.4	87.21
balance-scale	90.06	90.87	95.19	94.07	93.75	94.71	92.15	94.07	95.99
balloons	87.5	81.25	87.5	75	81.25	93.75	87.5	81.25	93.75
bank	90.22	90.13	89.8	89.54	90.04	90.2	90	90.13	89.87
blood	75.67	77.41	76.07	76.74	75.94	76.47	76.47	77.41	77.41
breast-cancer	75	74.65	74.65	73.59	72.89	76.06	73.94	74.65	74.3
bc-wisc	97.57	97.57	97.43	97.57	97.43	97.57	97.71	97.57	97.86
bc-wisc-diag	96.13	97.36	97.36	95.07	97.18	95.95	97.89	97.36	97.36
bc-wisc-prog	79.08	80.1	80.1	78.57	81.12	79.59	80.61	80.1	80.1
breast-tissue	71.15	72.12	72.12	72.12	75	69.23	69.23	70.19	73.08
car	98.09	98.32	98.15	98.38	98.44	98.5	97.92	98.44	98.78
ctg-10classes	87.29	94.96	88.18	84.7	85.22	86.25	86.06	87.34	85.73
ctg-3classes	95.1	86.91	95.43	92.8	94.54	95.06	94.16	95.15	93.41
chess-krvk	84.93	83.83	84.13	79.02	81.19	85.13	79.28	84.47	88.1
chess-krvkp	99.47	99.53	99.53	99.12	99.12	98.72	99.34	99.53	99.31
congress-voting	61.47	61.47	61.24	61.47	61.7	61.47	61.24	61.47	61.7

(continued on next page)

Datasets	RaF	obRaF	obRaF(L)	obRaF(RR)	obRaF(S)	obRaF(LS)	obRaF(LR)	obRaF(M)	obRaF(H)
conn-bench-sonar	82.21	83.17	84.13	78.85	83.17	82.69	82.21	83.17	81.25
conn-bench-vowel	98.81	99.03	100	99.57	99.78	99.89	99.84	100	100
connect-4	86.87	87.4	87	83.43	82.1	86.8	86.28	87.4	87.4
contrac	53.67	53.4	51.29	51.49	51.29	52.72	52.31	52.79	51.83
credit-approval	88.37	87.65	88.08	86.77	86.34	88.37	86.77	87.65	86.63
cylinder-bands	80.86	80.86	81.25	82.62	78.91	80.27	80.86	80.86	80.08
dermatology	97.53	98.08	98.08	97.53	97.53	97.25	97.53	97.25	97.8
echocardiogram	84.85	86.36	84.85	84.85	84.85	83.33	84.85	86.36	84.09
ecoli	87.8	87.8	87.5	86.1	87.2	87.2	88.39	87.5	87.8
energy-y1	95.44	96.09	96.09	95.96	95.18	96.22	95.44	96.48	96.22
energy-y2	89.97	89.32	90.1	90.1	89.19	90.1	88.8	89.32	90.1
fertility	89	90	89	88	89	89	89	90	88
flags	69.27	68.23	65.1	63.02	55.21	61.98	60.42	66.15	63.02
glass	77.83	74.06	75.47	73.11	73.58	70.75	70.28	75.47	76.95
haberman-survival	74.01	73.03	73.68	69.41	73.68	73.36	73.36	73.03	73.03
hayes-roth	87.5	87.5	89.29	88.39	87.5	86.61	86.61	88.39	88.39
heart-cleveland	57.57	58.55	57.57	57.57	57.24	58.88	59.54	58.22	57.89
heart-hungarian	84.93	83.56	85.27	84.59	83.9	84.59	85.62	83.56	85.49
heart-switzerland	41.13	46.77	50	50	44.35	51.61	45.16	46.77	46.77
heart-va	37	37.5	38	35	34.5	38.5	36	36.5	38.55
hepatitis	82.69	82.69	82.69	85.9	84.62	83.97	82.69	82.69	84.62
hill-valley	49.13	49.42	70.63	48.72	49.09	49.9	49.17	49.42	69.14
horse-colic	62.87	60.29	60.29	60.29	60.29	61.76	60.29	60.29	60.29
ilpd-indian-liver	70.55	73.12	72.77	72.77	72.09	72.09	73.29	73.12	72.77
image-segmentation	14.29	14.52	14.29	28.37	14.31	14.29	15.58	14.29	17.29
ionosphere	92.33	94.02	93.75	94.03	92.9	92.33	93.75	94.02	94.89
iris	95.95	96.62	95.95	95.95	95.95	95.95	95.95	95.95	97.97
led-display	69.4	72.8	69.7	71.5	71.4	71.2	71.6	70.6	71.1
lenses	83.3	83.3	83.33	79.17	79.17	87.5	79.17	83.33	83.33
letter	96.35	96.78	96.97	96.12	96.61	96.69	94.44	97.2	97.9
libras	78.89	82.78	86.94	81.39	82.78	83.89	83.89	84.17	87.22
low-res-spect	91.54	91.92	91.17	90.98	91.12	91.73	91.92	92.11	92.11
lung cancer	56.25	53.13	50	50	50	46.88	50	56.25	50
lymphography	85.81	84.46	87.16	86.49	87.84	86.49	87.84	86.49	87.84
magic	87.22	87.27	87.59	87.09	87.35	87.21	87.71	87.27	88.1
mammographic	82.71	81.04	81.04	81.15	80.73	82.19	80.42	81.04	83.13
miniboone	93.2	93.38	93.4	95.5	93.7	92.8	91.11	93.38	94.2
molec-biol-promoter	92.31	84.62	89.42	84.62	87.5	91.35	88.46	84.62	89.42
molec-biol-splice	95.29	94.84	93.1	92.97	87.58	93.29	92.13	94.29	86.9
monks-1	61.11	61	60.94	60.19	67.36	60.19	60.53	61	57.7
monks-2	66.2	66.44	66.67	66.78	66.2	65.97	67.19	66.44	66.84
monks-3	52.78	52.78	52.78	53.01	53.36	52.78	54.86	52.78	55.84
mushroom	100	100	100	100	100	100	100	100	100
musk-1	86.76	88.45	86.97	84.66	88.66	87.61	88.45	88.45	88.45
musk-2	97.67	96.92	97.41	96.97	96.97	97.74	97.88	96.92	97.84
nursery	99.64	99.59	99.71	99.37	99.39	99.67	99.65	99.66	99.61
oocMer12F	92.45	92.55	92.35	92.35	92.35	92.65	91.86	92.35	92.75
oocMer14D	78.82	82.25	85.78	84.8	81.57	79.71	82.06	82.25	84.02
oocTris2F	81.03	83.77	84.98	80.15	82.79	81.14	83.77	83.77	84.76
oocTris5B	91.78	92.54	95.18	93.86	93.64	94.63	93.42	93.97	94.96
optical	97.01	96.93	97.2	97.02	97.15	96.88	96.99	97.11	97.19
ozone	97.16	97.16	97.04	97.16	97.16	97.16	97.12	97.16	96.88
page-blocks	97.33	97.37	97.24	97.06	97.3	97.39	97.19	97.24	97.24
parkinsons	89.8	89.29	92.86	91.33	89.9	90.31	89.8	89.29	93.88
pendigits	95.28	96.12	96.95	97.01	97.16	97.01	96.34	96.93	97.14
pima	75.65	75.52	75.13	74.87	74.61	74.61	75.65	75.52	76.36
pb-MATERIAL	90.38	92.31	93.27	93.27	92.31	92.31	93.27	92.31	93.27
pb-REL-L	71.15	77.88	72.12	78.85	76.92	76.92	74.04	75.96	76.92
pb-SPAN	65.7	63.04	66.3	69.57	71.74	67.39	66.3	66.3	68.48
pb-T-OR-D	89	89	88	88	89	89	88	88	88
pb-TYPE	63.46	68.27	68.27	72.12	68.27	70.19	72.12	70.19	69.23
planning	72.22	71.11	71.67	71.67	72.22	72.22	72.22	71.11	72.67
plant-margin	86.19	85.38	86.94	85.13	84.56	83.94	83.31	85.19	87.44
plant-shape	64	66.38	74	71.94	68.31	69.69	62.3	69.06	75.12
plant-texture	82.69	82.5	83.56	81.79	82.44	83.37	83.1	82.44	84.81
post-operative	72.73	72.73	72.73	72.73	72.73	72.73	72.73	72.73	72.73
primary-tumor	56.1	55.18	56.1	55.79	54.27	54.57	55.49	55.79	57.18
ringnorm	96.95	92.43	97.78	97.42	96.55	96.93	97.74	92.43	98.92
seeds	93.75	94.23	95.67	92.31	92.31	92.31	92.79	94.23	96.19
semeion	93.91	93.59	93.91	93.34	93.84	91.77	93.66	94.22	96.3
soybean	39.1	36.7	45.15	14.1	79.12	38.43	67.62	38.56	72.34
spambase	94.52	94.87	94.76	94.61	94.96	94.74	95.04	94.87	95.5
spect	68.95	66.26	62.63	54.84	61.83	65.59	54.84	66.26	65.05
spectf	91.98	91.98	91.98	91.98	91.98	91.98	91.98	91.98	91.98
st-aus-credit	68.02	68.02	67.88	68.17	68.02	68.02	67.88	68.02	68.22
st-german-credit	78.6	78.9	78.9	75.8	78.2	79	78.3	78.9	77.8
st-heart	87.31	86.57	86.19	83.58	88.06	86.94	85.07	86.57	86.94
st-image	97.79	97.8	98.18	97.79	97.66	98.35	98.09	98.18	98.4
st-landstat	90.89	95.09	90.07	89.54	91.06	90.66	89.62	90.63	91.96
st-shuttle	99.99	99.97	99.95	99.95	99.97	99.97	99.98	99.99	100
st-vehicle	75.59	78.32	83.89	82.46	76.66	82.94	79.15	82.82	84.9
steel plates	78.71	77.94	78.61	77.42	77.94	77.84	78.25	79.12	78.63
synthetic-control	98.67	99	99.5	98.67	99.5	99.5	99.17	99.5	99.17
teaching	62.1	62.5	63.16	62.5	62.5	63.16	62.5	62.5	63.82
thyroid	98.95	98.95	98.98	98.53	98.69	98.92	98.75	98.95	98.5
tic-tac-toe	98.95	97.8	98.54	97.59	98.01	98.64	97.7	97.8	97.91
titanic	78.64	78.68	78.95	78.36	78.45	78.32	78.95	78.68	78.95
trains	100	87.5	87.5	75	87.5	100	87.5	87.5	87.5
twonorm	97.58	97.85	97.88	97.51	97.88	97.45	97.85	97.85	97.73
vc-2classes	83.44	84.09	83.77	83.12	84.42	83.77	84.42	84.09	85.39
vc-3classes	84.42	85.06	85.39	81.82	84.42	84.74	84.74	86.04	85.71
wall-following	99.73	99.7	96.79	97.56	95.77	98.13	98.59	97.84	96.92
waveform	84.84	86.24	86.24	86.08	86.38	85.88	86.04	86.58	86.28
waveform-noise	86.02	86.48	87.22	85.66	86.46	84.1	87.12	87.12	87.06
wine	98.86	98.96	100	100	99.43	99.43	98.86	99.43	98.3
w-qua-red	68.56	68.19	70.31	69.75	68.87	69.75	69.44	69.94	70.38

(continued on next page)

Datasets	RaF	obRaF	obRaF(L)	obRaF(RR)	obRaF(S)	obRaF(LS)	obRaF(LR)	obRaF(M)	obRaF(H)
w-qu-a-white	68.77	68.97	69.28	68.83	69.46	69.14	69.3	69.83	69.71
yeast	62.33	62.26	63.34	62.8	62.87	62.47	61.73	63.27	63.81
zoo	99	99	99	99	99	99	99	99	99
Mean Acc.	81.99	82.04	82.64	81.51	82.18	82.19	81.95	82.3	83.36

We follow the naming convention of [2].

Table 4.2: Average Friedman Rank Based on Classification Accuracy of Each Method on 121 UCI Datasets

	RaF	obRaF [10]	obRaF(L)	obRaF(RR)	obRaF(S)	obRaF(LS)	obRaF(LR)	obRaF(M)	obRaF(H)
Rank	5.4	5.07	4.28	6.13	5.65	5.14	5.46	4.43	3.44

L, RR, S, LS, LR, M and H in the brackets along with obRaF refer to our proposed LDA, Ridge Regression, SVM, LSSVM, LR, MPSVM and heterogeneous oblique random forests respectively. Lower rank reflects better performance.

Figure 4.6: Statistical comparison of classifiers against each other based on Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.05$) are connected.

Diversity-Error Diagrams

We visualize the random forest based methods in terms of “diversity” between the individual decision trees and their “accuracy” (or error). A visualization technique for classification ensembles is suggested in [130]. This approach was used in [131, 132] to discuss the performance of various random forest methods. In this visualization technique, pair-wise diversity is measured for two classifiers’ outputs. A κ coefficient represents the pair-wise diversity. It evaluates the level of agreement between two classifier outputs while correcting for chance [131]. For C class labels, κ is defined on $C \times C$ coincidence matrix $\mathcal{M}$ of the two classifiers. The entry m_{ij} in $\mathcal{M}$ is the proportion of the testing data, which one classifier labels as k th class while the other labels as the j th class. The agreement between the two

classifiers is given by the κ coefficient as follows:

$$\kappa = \frac{p_r(a) - p_r(e)}{1 - p_r(e)}, \quad (4.6)$$

where $p_r(a)$ is the observed agreement between the classifiers, and $p_r(e)$ is the hypothetical probability of agreement by chance. The observed agreement between the classifiers is given by:

$$p_r(a) = \sum_i m_{ii}. \quad (4.7)$$

and the hypothetical probability of agreement by chance by:

$$p_r(e) = \sum_k \left[\left(\sum_i m_{ki} \right) \left(\sum_j m_{jk} \right) \right]. \quad (4.8)$$

If two classifiers produce the same class labels for every data samples, it means the classifiers are in complete agreement for which $\kappa = 1$. If the classifiers are independent, their agreement will be the same as agreement by chance thus, $\kappa = 0$. Independence is not always the desired scenario in classification ensembles [131]. For an ensemble of L classifiers, $L(L - 1)/2$ pair-wise diversities or kappa coefficients are measured. The kappa-error diagram plots the κ coefficients for the two classifiers on the x-axis with the averaged individual classifier errors $E_{ij} = (E_i + E_j)/2$ on the y-axis. As small values of κ indicate better diversity or low correlation and small values of E_{ij} indicate better accuracy, the most desirable pairs of classifiers lie in the bottom left corner.

Fig. 4.7 shows the diversity-error diagrams for two datasets. All random forest methods consist of 500 decision trees, therefore, there are 124750 dots in each plot. We choose these two datasets because they have a predefined training and testing partitions. The ensembles were trained on the training sets and the κ -error diagrams were plotted based on the performance in testing sets. The scales in the diagrams are kept same for each dataset (in some cases, the axes are adjusted for better viewing) but may vary from one dataset to another. Fig. 4.8 plots the centroids of the cloud of kappa-error plots in the same plot for each dataset.

As shown in the plots, different random forest based methods introduced in this chapter possess varying characteristics. In optical dataset, obRaF(LS) is the most diverse classifier while obRaF(LR) is the most accurate classifier. However, en-

semble classifiers such as obRaF(L) and obRaF(H), with a fine blend of accuracy and diversity between the individual classifiers have the best overall performance in this dataset. Similarly, in the soybean dataset, obRaF(RR) is the most diverse classifier while obRaF(S) is the most accurate classifier. This dataset is challenging and has large fluctuations between the performance of the classifiers. In this dataset, all the ensemble classifiers have κ very close to zero which might not be the best scenario for this dataset. obRaF(S), with the most accurate individual classifiers and κ slightly far away from zero compared to other classifiers, has the best overall performance in this dataset. In both datasets, the obRaF(H) has a good blend of accuracy and diversity. Thus, it consistently performs better in almost all the datasets.

Comparison with stand-alone differential decision tree (sNDF)

Here, we compare the performance of heterogeneous oblique random forest (obRaF(H)) and differential decision trees (sNDF) [44] in several datasets. The comparison in terms of accuracy is reported in Table 4.3. The top 5 datasets in Table 4.3 are also used in [44] and for those datasets, we report the results obtained from 10 repetitions of the experiment. To perform an extensive comparison, we include 10 additional datasets used above with pre-defined training and testing partitions. For the 10 datasets, we implement sNDF using the source code ⁴ available online and report the test accuracy. From Table 4.3, we can see that our proposed obRaF(H) can be considered as another competitive baseline for the practitioners.

Decision Boundaries

To visualize the decision boundaries of Random Forest and Oblique Random Forest variants, we generate a three class artificial spirals dataset as shown in Fig. 4.9(a). An axis-parallel decision tree recursively partitions the input space as shown in Fig. 4.9(b). Thus, it approximates the spiral decision boundary using several staircase-like decision hyperplanes. However, as shown in Fig. 4.9(c), a smoother decision boundary can be obtained by employing a number of decision trees. The rest of the decision boundaries are of oblique random forests and their variants introduced in this thesis. Fig.4.9(k) shows the decision boundary of a single heterogeneous decision tree. The decision boundary generated is much smoother compared to that generated by a single axis-parallel decision tree. As can be seen from the figures, the proposed oblique random forests generate much

⁴<https://github.com/jingxil/Neural-Decision-Forests>

(a) 'optical' dataset

(b) ‘soybean’ dataset

Figure 4.7: κ -Error diagrams of “optical” and “soybean” datasets with all forest based methods introduced in this chapter. x-axis = κ (diversity between two classifiers), y-axis = E_{ij} (average error of the pair of classifiers).

Figure 4.8: Centroids of the κ -Error clouds of the “optical” and “soybean” datasets.

Table 4.3: Comparison in terms of % error (first 5 datasets) and % accuracy (last 10 datasets)

Datasets	sNDF [44]	obRaF(H)
G50c	17.4±1.52	11.44±1.35
Letter	2.92±0.17	3.5±0.13
USPS	5.01±0.24	5.1±0.2
MNIST	2.8±0.12	2.68±0.17
Char74k	16.04±0.2	16.0±0.25
annealing	69	96.23
audiology-std	72	87.21
hill-valley	52.64	69.14
image-segmentation	31.19	17.29
monks-3	69.67	55.84
optical	96.99	97.19
pendigits	96.97	97.14
soybean	81.12	72.34
st-landsat	89.35	91.96
thyroid	98.19	98.5

sNDF is the shallow neural decision forests of [44]. For the first 5 datasets, the test error is averaged for 10 different repetitions as in [44].

smoother decision surfaces which better represent the data than the axis-parallel equivalent.

4.5.4 Computational Complexity Analysis

Here, we analyze the computational limitations of the oblique decision trees proposed in this chapter. As the computational time required to grow an oblique tree is highly variable, we instead focus on the complexity of partitioning a single node. Consider a training set $\mathbf{X}$ which is a $N \times d$ matrix with N samples and d features. As each node is associated with different sizes of training data, for the sake of simplicity, here we consider the complexity at the root node. Let K denote the number of classes at the root node. The number of splits that need to be evaluated in the one-vs-all based oblique decision trees is upper bounded by K . The complexity of the linear classifiers used in this chapter is presented in Table 4.4. Regardless of the computational time for calculating the Gini scores, the complexities associated with each type of oblique decision trees at the root node are $\mathcal{O}(Nd^2 \cdot K)$ for obRaF(L), obRaF(RR), obRaF(LS), obRaF(LR), $\mathcal{O}(N^2d \cdot K)$ for obRaF(S), and $\mathcal{O}(d^3 \cdot K)$ for obRaF(M) respectively. As the heterogeneous decision tree employs all the linear classifiers several times, the overall computational complexity for obRaF(H) is the aggregate of all the complexities associated with each linear classifier. In Table 4.5 and Table 4.6, we present the training time and average tree depth of each method in some selected datasets respectively. Our proposed oblique random forest methods generally generate shallow decision trees and require less training time.

4.5.5 Overall Comparisons Among 190 Classifiers

In this section, we compare the performance of the oblique random forests introduced in this chapter with the 183 classifiers on 121 UCI datasets as in [2, 10]. Thus, including seven oblique random forest variants proposed in this thesis, there are altogether 190 classifiers. However, as discussed in [135], there are some flaws in the experimental procedures in [2]. In 102 of the 121 benchmarked datasets without predefined training and testing sets, half of the test examples are used as a validation set for hyper-parameter tuning. This error has likely inflated the performance of some of the benchmarked classifiers with tunable hyperparameters. To address this issue, we use obRaF(H) in an out-of-the-box fashion, without

Figure 4.9: Decision boundaries for the classification of the artificial spirals dataset

Table 4.4: Computational complexity of the linear classifiers

Algorithm	Complexity
LDA	$\mathcal{O}(Nd^2)$ [133]
RR	$\mathcal{O}(Nd^2)$
SVM	$\mathcal{O}(N^2d)$ [133]
LS-SVM	$\mathcal{O}(Nd^2)$ [134]
LR	$\mathcal{O}(Nd^2)$ [133]
MPSVM	$\mathcal{O}(d^3)$ [65]

The complexity of an iterative algorithm such as LR is analyzed for one iteration, and so their actual running time may be slower.

Table 4.5: Training Time Comparison Relative to RaF

Datasets	RaF	obRaF	obRaF(L)	obRaF(RR)	obRaF(S)	obRaF(LS)	obRaF(LR)	obRaF(M)	obRaF(H)
pendigits	1	0.88	0.63	1.29	1.14	1.2	1.1	0.83	2.7
plant-shape	1	0.9	0.88	1.1	0.77	1.3	2.01	0.75	2.3
st-image	1	0.73	0.34	2.1	1.09	1.2	0.48	0.38	1.8

doing any parameter tuning. Specifically, we use default parameters, 500 trees and set $m_{try} = \sqrt{\#features}$ for each dataset and report its performance in all 121 datasets in Table 4.7. We name this obRaF(H) as obRaF(H)-d. As shown in Table 4.7, even without parameter tuning, obRaF(H)-d is still the top-ranked classifier. Next, we present the overall 20 best performing classifiers in Table 4.8 and Table 4.9. From the tables, we can see that obRaF(H) is the top-ranked algorithm in each case. Although the No-Free-Lunch theorem says that no single classifier is always the best, it is intuitive to implement the most robust classifier that generalizes better in most datasets. Notably, the top three ranked methods are oblique random forest variants introduced in this thesis.

Table 4.6: Average Tree Depth

Datasets	RaF	obRaF	obRaF(L)	obRaF(RR)	obRaF(S)	obRaF(LS)	obRaF(LR)	obRaF(M)	obRaF(H)
pendigits	288	282	260	268	393	475	331	267	162
plant-shape	271	327	267	260	330	350	371	302	237
st-image	101	75	87	77	147	83	73	127	57

Table 4.7: Accuracy(%) of obRaF(H)-d in all 121 datasets

Datasets	Accuracy(%)	Datasets	Accuracy(%)
abalone	65.59	molec-biol-splice	92.22
ac-inflam	100	monks-1	57.18
acute-nephritis	100	monks-2	66.67
adult	85.97	monks-3	54.17
annealing	96.48	mushroom	100
arrhythmia	71.24	musk-1	88.24
audiology-std	86.05	musk-2	96.73
balance-scale	95.03	nursery	99.59
balloons	93.75	oocMerl2F	92.55
bank	90.13	oocMerl4D	84.22
blood	76.87	oocTris2F	83.33
breast-cancer	74.3	oocTris5B	93.31
bc-wisc	97.57	optical	97.44
bc-wisc-diag	97.18	ozone	97.08
bc-wisc-prog	82.65	page-blocks	97.26
breast-tissue	74.04	parkinsons	90.82
car	98.21	pendigits	96.86
ctg-10classes	85.88	pima	75.78
ctg-3classes	93.03	pb-MATERIAL	93.27
chess-krvk	85.1	pb-REL-L	74.04
chess-krvkv	98.81	pb-SPAN	69.57
congress-voting	61.47	pb-T-OR-D	89
conn-bench-sonar	84.62	pb-TYPE	72.12
conn-bench-vowel	100	planning	71.11
connect-4	86.7	plant-margin	87.44
contrac	51.49	plant-shape	75.12
credit-approval	86.63	plant-texture	84.81
cylinder-bands	80.47	post-operative	72.73
dermatology	97.8	primary-tumor	60.98
echocardiogram	84.09	ringnorm	97.61
ecoli	87.5	seeds	93.75
energy-y1	95.31	semeion	93.84
energy-y2	89.45	soybean	74.47
fertility	88	spambase	95.37
flags	60.94	spect	62.9
glass	72.17	spectf	91.98
haberman-survival	69.74	st-aus-credit	65.12
hayes-roth	89.29	st-german-credit	77.1
heart-cleveland	57.89	st-heart	87.69
heart-hungarian	86.64	st-image	98.05
heart-switzerland	45.97	st-landstat	90.6
heart-va	38	st-shuttle	100
hepatitis	83.97	st-vehicle	79.03
hill-valley	58.25	steel plates	77.99
horse-colic	60.29	synthetic-control	99.17
ilpd-indian-liver	73.29	teaching	63.16
image-segmentation	15.05	thyroid	98.6
ionosphere	94.32	tic-tac-toe	99.06
iris	96.62	titanic	78.95
led-display	74.8	trains	87.5
lenses	79.17	twonorm	97.85
letter	96.78	vc-2classes	85.06
libras	83.88	vc-3classes	84.42
low-res-spect	92.11	wall-following	99.7
lung cancer	46.88	waveform	86.36
lymphography	88.51	waveform-noise	87.12
magic	87.43	wine	98.86
mammographic	81.35	w-qua-red	69.44
miniboone	93.4	w-qua-white	69.24
molec-biol-promoter	88.46	yeast	62.7
		zoo	99
		Mean Acc.	82.83

Table 4.8: Overall comparison of 20 top ranked classifiers out of 190 methods in 121 datasets

Method	Rank	Friedman Rank	Mean Accuracy
obRaF(H)*	1	29.22	83.39
obRaF(L)*	2	34.28	82.64
obRaF(M)*	3	35.28	82.3
MPRaF	4	36.73	82.05
CoKRR-max	5	37.84	81.79
MPRRoF	6	38.14	82.35
parRaF_t	7	38.6	81.96
RaF_t	8	38.73	82.3
obRaF(LS)*	9	39.37	82.19
obRaF(S)*	10	39.92	82.19
obRaF(LR)*	11	40.33	81.95
svm_C	12	42.29	81.77
svmPoly_t	13	44.57	80.31
KRR	14	45.55	81.52
rForest_R	15	45.65	81.94
svmRadialCost_t	16	46.32	80.54
obRaF(RR)*	17	46.83	81.51
svmRadial_t	18	48.98	80.17
RVFL ensemble	19	49.27	80.94
C5.0_t	20	49.38	80.56

“*” indicates the methods introduced in this work. We follow the naming conventions of [2, 10]. This comparison is based on the performance of the classifiers stated in [2, 10].

4.5.5.1 Statistical Comparison

We use a Wilcoxon signed-rank test to determine the significance among each pair of the top 20 ranked classifiers as suggested in [136]. As discussed in [135], some of the benchmarked datasets fail to run for some classifiers. In such cases, missing values were imputed by setting accuracy for benchmarks with errors to 82% in [2]. However, this biases the results and [135] performs the statistical comparison by removing the benchmarked datasets that fail to run for any top 20 ranked classifier. We follow similar procedure as in [135] and report the comparison results in Tables 4.10 and 4.11. One can see from the tables that obRaF(H) is statistically significantly better than all other top 19 ranked classifiers. Thus, obRaF(H) could be considered as another competitive baseline for the practitioners.

Table 4.9: Comparison of 20 Top Ranked Classifiers Out of 184 Methods

Method	Rank	Friedman Rank	Mean Accuracy
obRaF(H)-u	1	33.39	82.83
MPRaF	2	33.69	82.05
CoKRR-max	3	34.63	81.79
MPRRoF	4	35.07	82.35
parRaF_t	5	35.5	81.96
RaF_t	6	35.73	82.3
svm_C	7	39.31	81.77
svmPoly_t	8	40.91	80.31
KRR	9	42.17	81.52
rForest_R	10	42.27	81.94
svmRadialCost_t	11	42.96	80.54
svmRadial_t	12	45.34	80.17
RVFL ensemble	13	45.45	80.94
C5.0_t	14	45.82	80.56
avNNet_t	15	47.18	79.37
nnet_t	16	48.65	79.52
BG_LibSVM_w	17	50.04	80.77
pcaNNet_t	18	50.29	78.65
mlp_t	19	50.5	80.25
RotationForest_t	20	50.91	80.61

obRaF(H)-d is the untuned (with default settings) obRaF(H). Here, we exclude other oblique random forest variants proposed in this chapter. We compare obRaF(H)-d with the classifiers in [2, 10].

4.6 Conclusion

In this chapter, we first presented several variants of oblique random forest employing linear classifiers at the internal nodes of the tree. The oblique splits were generated based on one-vs-all partitioning. We also presented a heterogeneous variant employing diverse linear classifiers trained on some selective top ranked partitions. The partitions were created using both one-vs-all and two-hyperclasses based approaches and ranked using the ideal gini scores and cluster separability. We showed that these two criteria can be used to rank the partitions. The proposed oblique random forests are more accurate and diverse than other random forest methods. This work also expands the comparison of classifiers as done in [2, 10] on 121 UCI datasets.

Table 4.10: Statistical comparison of obRaF(H) and each of the other top 20 ranked classifiers

Method	Rank	Wilcoxon signed-rank p, excluding errors
obRaF(L)*	2	2e-04
obRaF(M)*	3	4e-04
MPRaF	4	4e-05
CoKRR-max	5	2e-03
MPRRoF	6	1e-04
parRaF_t	7	4e-04
RaF_t	8	2e-04
obRaF(LS)*	9	1e-06
obRaF(S)*	10	1e-08
obRaF(LR)*	11	1e-08
svm_C	12	8e-04
svmPoly_t	13	3e-05
KRR	14	1e-03
rForest_R	15	4e-06
svmRadialCost_t	16	6e-04
obRaF(RR)*	17	1e-09
svmRadial_t	18	4e-07
RVFL ensemble	19	3e-07
C5.0_t	20	3e-07

The statistical comparison is done using a Wilcoxon signed-rank test. The test is performed by removing the datasets which failed to run for any one of the top 20 ranked classifiers instead of incorrectly imputing them [135].

Chapter 5

Stacked Autoencoder based Deep RVFL

In this chapter, we present several deep RVFL variants [137] by utilizing the framework of multi-layer ELM. Specifically, we introduce direct connections (feature reuse) from preceding layers to the fore layers of the network as in the original RVFL network. Furthermore, we also introduce denoising criterion to extract robust higher level representations.

5.1 Introduction

Deep neural networks typically consist of multiple hidden layers stacked together. Each hidden layer builds an internal representation of the data with the hidden layers closer to the input layer learning simple features such as edges and layers above them learning sophisticated (complex) features [13, 14]. With such stacked layers, deep learning models typically have thousands of model parameters that need to be optimized during the training phase. These networks are typically trained using back-propagation (BP) technique so as to minimize the loss function (cross-entropy or mean square error or others depending on the particular task). In addition to be time-consuming, such models may fail to converge to a global minimum, thus, giving sub-optimal performance or lower generalization [21]. Also, such deep learning models require large amount of training data. While the usual image and speech datasets that are commonly used with deep learning models have abundant data, there are datasets from a wide variety of domains, such as

agriculture, credit scoring, health outcomes, ecology and others, with very limited data size. The performance of the state-of-the-art deep learning models on such datasets are far from superior [7, 15].

Among the randomization based methods [22–24], RVFL network has rapidly gained significant traction because of its superior performance in several diverse domains. Randomization based neural networks greatly benefit from the presence of direct links from the input layer to the output layer as in RVFL network [25, 76, 78]. The original features are reused or propagated to the output layer via the direct links. The direct links act as a regularization for the randomization [82, 83]. It also helps to keep the model complexity low with the RVFL network being thinner and simpler compared to its other counterparts. With the Occam's Razor principle and PAC learning theory [84] advocating for simpler and less complex models, this makes the RVFL network attractive to use compared to other similar randomized neural networks.

Extreme learning machine (ELM), a variant of RVFL, has been widely used for building randomization based multi-layer or deep models [92, 93, 95, 96, 138]. To create deep models, multiple randomization based AEs are stacked together as in conventional deep learning models such as stacked autoencoders (SAEs) [139], deep belief network (DBN) [140]. Among several variants of ELM based multi-layer neural networks, a popular framework is Hierarchical ELM (H-ELM) [96] that consists of two components: unsupervised multi-layer feature encoding and supervised feature classification. For the feature extraction part, several randomization based autoencoders trained independently are stacked together while the original ELM is employed for feature classification part. In this chapter, we extend the H-ELM framework by using direct links as in the original RVFL network while maintaining its advantages of lower complexity and training efficiency. The key contributions of this chapter are summarized as follows:

- We extend the H-ELM framework [96] by using direct connections from the preceding layers to the fore layers of the network. Based on this idea, we propose several variants of multi-layer networks known as multi-layer RVFL networks. The direct connections enable feature reuse which enables the neural network to generalize better. This work is similar to the works in [141, 142] but there are several key differences. First, the deep neural networks proposed in this chapter do not require back-propagation based learning. Second, unlike in [141, 142], the deep neural networks proposed in this chapter are created by stacking several randomization based AEs

followed by a RVFL classifier where each randomization based AE is an independent feature extractor.

- To extract better higher level representation by the unsupervised feature extraction part, we introduce a denoising criterion in the autoencoders. We also propose a deep RVFL network with both direct connections and denoising criterion.
- With extensive experiments on 20 classification datasets, we show that our proposed methods generalize better and faster compared to other relevant state-of-the-art deep neural networks.

5.2 RVFL based deep neural network

In this section, we propose three deep RVFL variants. Since the deep learning models are built hierarchically with high level feature extraction followed by final classification layer, all three variants utilize H-ELM architecture with unsupervised feature extraction using randomization based AE followed by RVFL classifier for decision making. Our proposed frameworks differ from multi-layer ELMs introduced in Chapter 2 (cf. Section 2.3.2.2) in such that we introduce feature re-use at different parts of the multi-layer networks via direct links from the preceding layers and also replace the ELM classifier by RVFL.

5.2.1 Deep RVFL with direct links

In the Deep RVFL with direct links (sdRVFL(d)) framework (Fig. 5.1), before unsupervised feature extraction, the input raw data is randomly mapped to a feature space to exploit hidden information in the original input space. Then, a L -layer unsupervised learning is performed to obtain complex high-level features. The output of each hidden layer can be written as:

$$\mathbf{H}_i = a(\mathbf{H}_{i-1} \cdot \boldsymbol{\beta}), \text{ where } i \in [1, L] \quad (5.1)$$

where $\mathbf{H}_i$ is the output of the i -th layer, $\mathbf{H}_{i-1}$ is the output of the $(i-1)$ th layer, $a(\cdot)$ denotes the activation function of the hidden layers, and $\boldsymbol{\beta}$ represents the output weights. Each hidden layer of sdRVFL(d) is an independent feature extractor. As the layers increase, the resulting feature becomes more compact or complex.

Figure 5.1: Architecture of sdRVFL(d). For the feature encoding part, randomization based AE is used followed by RVFL classifier.

After unsupervised hierarchical training, the resultant outputs of the L -th layer, i.e., $\mathbf{H}_L$, are viewed as the high-level complex features extracted from the input data. Unlike H-ELM, in case of sdRVFL(d), the input features to the RVFL classifier is a concatenation of the original raw features $\mathbf{X}$ and the penultimate hidden layer features $\mathbf{H}_L$. If $\mathbf{X}^c$ is the input to the RVFL classifier, it is defined as:

$$\mathbf{X}^c = [\mathbf{H}_L, \mathbf{X}] \quad (5.2)$$

Convolutional RVFL(CRVFL) [74] employs similar technique with direct links from the input to the final decision making layer. However, sdRVFL(d) differentiates itself from convolutional RVFL (CRVFL) in such that the sdRVFL(d) is built hierarchically where RVFL classifier follows unsupervised feature extraction. That means, unlike in CRVFL, the concatenated features in sdRVFL(d) are randomly perturbed before passing to the RVFL classifier to improve the generalization performance [71, 96].

Figure 5.2: Architecture of sdRVFL(dense). It is built hierarchically with ELM-AE based feature encoding followed by RVFL classifier in addition that each layer takes all preceding features as input.

5.2.2 Deep RVFL with dense direct links

In the Deep RVFL with dense direct links (sdRVFL(dense)) framework (Fig. 5.2), the direct connections are introduced from a preceding layer to all subsequent layers. Thus, the input to the subsequent layer is the features from all preceding layers. Because of its dense connectivity, we refer to this sdRVFL architecture as sdRVFL(dense). If $\mathbf{X}^{(L)}$ is the input to the L -th layer, it is represented as:

$$\mathbf{X}^{(L)} = [\mathbf{X}, \mathbf{H}_1, \dots, \mathbf{H}_{L-1}] \quad (5.3)$$

where $[\mathbf{X}, \mathbf{H}_1, \dots, \mathbf{H}_{L-1}]$ refers to the concatenation of the features produced in layers $0, \dots, L-1$. As discussed above, each randomization based AE acts as an independent feature extractor. By reusing the features from the lower complexity levels, the aim is to extract more meaningful and compact features.

5.2.3 Deep RVFL with denoising autoencoders

In case of non-linear mapping such as neural networks, training with noise has different effect to training with l_1 or l_2 weight decay and results in qualitatively different outcome (features) than training with a weight decay regularization [89]. Thus, to extract better higher level representations, the inputs to the randomized based AE are corrupted by adding Gaussian noise as discussed in Section 2.3.1.5. Otherwise, the architecture of Deep RVFL with denoising autoencoders (sdRVFL-D(dense)) is identical to that of sdRVFL(dense).

5.3 Experiment and Analysis

In this section, we compare the performance of our proposed methods with relevant multi-layer (deep) ELMs. Multi-layer ELMs, H-ELM and ML-KELM have been shown to outperform other multi-layer neural networks such as SAE, SDA, Deep Boltzman Machines (DBM), multilayer perceptron (MLP-BP) and ML-ELM in [95, 96]. Even though our proposed methods can be integrated with any multi-layer ELM frameworks, in this chapter, we use H-ELM as our base framework as it follows the common architecture of deep learning models with separate feature extraction followed by a classification part and also it has good generalization capability [96]. To show the effectiveness and efficiency of our proposed methods, we conduct extensive experiments and analysis. The analysis is done over two parts: 1) individually: comparison of the multi-layer algorithms with the same weight regularization 2) jointly: overall comparison with the relevant state-of-the-art multi-layer ELMs.

The algorithms are compared over 20 UCI datasets whose details are presented in Table A1 in the Appendix. For a fair comparison, we use the publicly available data partitions (training, validation and testing partitions) of [2]. Following the practice in [95, 96], the number of hidden layers L is set to 3 for all experiments. The number of hidden nodes for an AE is tuned over 10:200 with a step-size of 10 while the hidden nodes for ELM/RVFL classifier is tuned over 100:2000 with a step-size of 100. Meanwhile, the regularization parameter λ in each layer is set to be $(1/C)$ where C is set as 10^x , $\{x = -7, -5, \dots, 7\}$. For ML-KELM, RBF kernel is used as in [95] and the kernel parameter σ in each layer is set as 10^x , $\{x = -7, -5, \dots, 7\}$. For deep RVFL using denoising AE, the noise variance is tuned over $\{0.05, 0.1, 0.15, 0.3, 0.5, 0.75\}$. The H-ELM and ML-KELM algorithms are

Table 5.1: Accuracy (%) of deep neural networks with l_1 -regularization and ML-KELM

ML-KELM [95]	Dataset	H-ELM(l_1) [96]	sdRVFL(d- l_1)	sdRVFL(dense- l_1)	sdRVFL-D(dense- l_1)
63.1	abalone	63.77	64.6	66.48	66.28
64.82	arrhythmia	72.12	72.2	72.79	72.9
89.03	bank	89.2	89.68	89.82	89.45
94.62	chess-krvk	99	99.25	99.2	99.19
60.78	congress-voting	61.24	61.47	61.93	61.93
52.38	contrac	54.08	54.82	55.43	55.53
89.64	ctg-3classes	90.68	91.57	92.37	92.33
75.71	ctg-10classes	82.39	82.82	82.2	82.91
54.72	glass	68.87	70.28	71.7	71.28
93.1	letter	93.15	95.34	95.45	95.56
81.27	molec-biol-splice	82.4	82.59	82.97	83.19
42.59	monks-3	78.7	83.33	83.8	83.41
95.85	musk-2	98.32	98.68	99.3	99.32
90.98	oocTris2F	92.06	92.94	93.33	93.92
86.7	st-image	95.28	96.1	96.62	96.79
92.59	spambase	92.67	93.41	93.61	93.96
86.16	wall-following	89.46	89.24	90.91	90.91
86.7	waveform	86.16	86.72	86.36	86.58
85.72	waveform-noise	86.08	86.28	86.86	86.48
58.15	w-qu-white	55.49	55.6	58.78	59.19
77.23	Mean Acc.	81.55	82.34	82.99	83.05
-	Avg. Friedman Rank	3.9	2.75	1.8	1.55

Lower rank reflects better performance.

implemented using the source codes ¹ ² available online.

5.3.1 Performance Comparison

As mentioned above, the algorithms are compared over two parts. In the first part, we compare the deep networks using the same weight regularization separately. In Tables 5.1, 5.2 and 5.3, we present the classification accuracy of each method in each dataset. As can be seen from these tables, sdRVFL-D(dense) algorithms have the highest average accuracy for each type of regularization followed by sdRVFL(dense) and sdRVFL(d). The base frameworks, H-ELM algorithms have the lowest accuracy in each case. However, comparing the classifiers using average accuracy is susceptible to outliers and may atone for a classifier's poor performance in one dataset with an excellent performance on the other. Thus, we follow the procedure of [2, 10], and use the Friedman rank of each classifier to assess its performance. From the same tables, we can see that sdRVFL-D(dense) is the top ranked algorithm in each case followed by sdRVFL(dense) and sdRVFL(d). This shows that our proposed deep neural networks with direct and dense connections consistently perform better than other algorithms.

¹http://www.ntu.edu.sg/home/egbhuang/elm_codes.html,

²<https://www.fst.umac.mo/en/staff/fstcmv.html#software>

Table 5.2: Accuracy (%) of deep neural networks with l_2 -regularization

Dataset	H-ELM(l_2)	sdRVFL(d- l_2)	sdRVFL(dense- l_2)	sdRVFL-D(dense- l_2)
abalone	63.79	65.71	66.38	66.64
arrhythmia	70.35	71.02	72.45	72.9
bank	89.62	89.23	89.8	89.87
chess-krvk	99.28	99.03	98.97	99.12
congress-voting	61.24	61.93	61.7	61.93
contrac	52.85	53.74	54.08	54.35
ctg-3classes	91.2	91.43	92.14	92.23
ctg-10classes	80.6	81.69	82.58	83.1
glass	68.87	69.34	69.81	69.4
letter	93.62	94.91	95.46	95.31
molec-biol-splice	75.94	83.03	83.56	83.47
monks-3	48.84	81.1	81.48	81.32
musk-2	97.67	98.48	98.83	98.76
oocTris2F	91.27	92.45	93.53	93.16
st-image	92.89	96.58	96.58	96.75
spambase	92.04	93.59	93.78	93.89
wall-following	88.84	89.61	90.38	90.89
waveform	86.16	86.36	86.66	86.66
waveform-noise	85.9	86.12	86.52	86.5
w-qua-white	55.49	55.6	59.27	59.93
Mean Acc.	79.32	82.04	82.69	82.8
Avg. Friedman Rank	3.8	2.95	1.8	1.45

H-ELM(l_2) is the hierarchically built ML-ELM [92] with separate feature extraction and classification.

Table 5.3: Accuracy (%) of deep neural networks with elastic net regularization

Dataset	H-ELM(elas)	sdRVFL(d-elas)	sdRVFL(dense-elas)	sdRVFL-D(dense-elas)
abalone	63.63	63.65	65.54	65.68
arrhythmia	67.92	71.46	72.57	72.46
bank	89.29	89.54	89.38	89.49
chess-krvk	99.06	99.06	99.1	99.12
congress-voting	61.7	61.77	61.93	61.93
contrac	55.57	54.82	54.96	54.19
ctg-3classes	91.24	91.96	91.96	92.42
ctg-10classes	82.06	82.06	82.5	82.86
glass	67.45	67.45	68.4	68.4
letter	93.59	95	95.36	95.31
molec-biol-splice	82.21	83.53	82.94	82.56
monks-3	81.26	81.94	83.33	81.71
musk-2	99	99.1	99.8	99.77
oocTris2F	91.51	91.57	93.53	93.92
st-image	95.71	96.53	96.62	96.62
spambase	92.52	93.48	93.33	93.48
wall-following	90.14	90.8	90.82	90.85
waveform	86.28	86.5	86.48	86.5
waveform-noise	85.06	86.24	85.98	86.48
w-qua-white	55.7	56.49	59.48	59.17
Mean Acc.	81.54	82.15	82.7	82.74
Avg. Friedman Rank	3.77	2.65	1.95	1.62

H-ELM(elas) is the generalization of NR-ELM [93] for non-image datasets without SIFT descriptors and maximum pooling operation.

We also perform a statistical comparison of the algorithms for each regularization type using the Friedman test as done in previous chapters.

Based on simple calculations we obtain, $\chi_F^2(l_1) = 40.98$, $\chi_F^2(l_2) = 41.82$, $\chi_F^2(\text{elastic}) = 31.92$ and $F_F(l_1) = 40.93$, $F_F(l_2) = 43.7$ and $F_F(\text{elastic}) = 21.59$. With 4 classifiers for each regularization type and 20 data sets, F_F is distributed according to the F -distribution with $4 - 1 = 3$ and $(4 - 1)(20 - 1) = 57$ degrees of freedom. The critical value for $F_{(4,56)}$ for $\alpha = 0.05$ is 2.76, so we reject the null-hypothesis. Based on the Nemenyi test, the critical difference is $CD = q_\alpha \sqrt{(m(m + 1))/(6M)} = 2.569 * \sqrt{4 * 5/(6 * 20)} \simeq 1.04$. From Figure 5.3, we can see that our proposed methods are statistically significantly better than the base frameworks or H-ELM multi-layer networks.

To determine the best network, we also perform an overall comparison of the networks presented in this chapter. First, we compute the average Friedman rank of all the algorithms as shown in Table 5.4. From the table, we can see that our proposed methods are the top-ranked algorithms. We also perform a statistical comparison of the algorithms to test the significance between each pair of algorithms. Based on simple calculations we obtain, $\chi_F^2 = 138.48$ and $F_F = 25.92$. With 13 classifiers and 20 data sets, F_F is distributed according to the F -distribution with $13 - 1 = 12$ and $(13 - 1)(20 - 1) = 228$ degrees of freedom. The critical value for $F_{(12,228)}$ for $\alpha = 0.05$ is 1.79, so we reject the null-hypothesis. Based on the Nemenyi test, the critical difference is $CD = q_\alpha \sqrt{(m(m + 1))/(6M)} = 3.313 * \sqrt{13 * 14/(6 * 20)} \simeq 4.08$. The outcome of the statistical test is reported in Table 5.5. The empty entry indicates that no statistically significant difference is observed. + means the method in the corresponding row is statistically better than the method in the corresponding column, Similarly, - means the method in the corresponding row is statistically worse than the method in the corresponding column.

5.3.2 Complexity Analysis

In this section, we compare the multi-layer networks in terms of model complexity (number of hidden nodes) and training time. The training time of H-ELM based networks depends on the number of hidden nodes, J while that of ML-KELM depends on the size of the training data, T . Since the autoencoder part of the network requires less number of hidden neurons, the training time of H-ELM based

Figure 5.3: Comparison of classifiers against each other with the Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.05$) are connected. The Nemenyi test is performed separately for each type of regularization based neural network.

Table 5.4: Jointly calculated average Friedman rank based on classification accuracy of each method

Algorithm	Rank
ML-KELM [95]	11.95
H-ELM(l_2) [92]	10.60
H-ELM(l_1) [96]	10.32
H-ELM(elas) [93]	9.70
sdRVFL(d- l_2) [†]	8.35
sdRVFL(d-elas) [†]	7.37
sdRVFL(d- l_1) [†]	6.52
sdRVFL(dense-elas) [†]	5.72
sdRVFL(dense- l_2) [†]	5.12
sdRVFL-D(dense-elas) [†]	5.00
sdRVFL-D(dense- l_2) [†]	3.77
sdRVFL(dense- l_1) [†]	3.57
sdRVFL-D(dense- l_1) [†]	2.97

[†] are the methods introduced in this chapter. Lower rank reflects better performance.

networks is heavily influenced by the number of hidden nodes in the classifier part of the network (ELM/RVFL with l_2 regularization) which requires matrix inversion of size $T \times T$ or $J \times J$ whichever is smaller. In a standard implementation, the matrix inversion of a matrix of size $(T \times T)$ requires $\mathcal{O}(T^3)$ time and $\mathcal{O}(T^2)$ memory [143]. For RVFL classifiers, direct links should also be considered when computing $J \times J$. ML-KELM, on the other hand, requires the matrix inversion of the kernel matrix of size $T \times T$. For the complexity analysis, we select the largest and the smallest datasets used in this chapter, bank and glass datasets respectively. We report the number of hidden nodes in each layer and training time of each algorithm in these two datasets in Table 5.6 and 5.7 respectively. From these tables, we can see that our proposed methods have substantially lower model complexity (less number of hidden nodes) and require less training time than H-ELM networks and ML-KELM. Because of the direct connections (feature reuse), the RVFL classifier in our proposed methods requires less number of hidden neurons for decision making. Also, the autoencoders in our proposed algorithms are able to extract better higher level representations compared to other algorithms which is evident by the accuracy of the classifier that uses it as input.

Table 5.5: Statistical Comparison of the Algorithms

	sdRVFL-D(dense- l_1)	sdRVFL(dense- l_1)	sdRVFL-D(dense- l_2)	sdRVFL-D(dense-elas)	sdRVFL(dense- l_2)	sdRVFL(dense-elas)	sdRVFL(d- l_1)	sdRVFL(d-elas)	sdRVFL(d- l_2)	H-ELM(elas)	H-ELM(l_1)	H-ELM(l_2)	ML-KELM
sdRVFL-D(dense- l_1)								+	+	+	+	+	+
sdRVFL(dense- l_1)									+	+	+	+	+
sdRVFL-D(dense- l_2)									+	+	+	+	+
sdRVFL-D(dense-elas)										+	+	+	+
sdRVFL(dense- l_2)										+	+	+	+
sdRVFL(dense-elas)											+	+	+
sdRVFL(d- l_1)												+	+
sdRVFL(d-elas)	-												+
sdRVFL(d- l_2)	-	-	-										
H-ELM(elas) [93]	-	-	-	-	-								
H-ELM(l_1) [96]	-	-	-	-	-	-							
H-ELM(l_2) [92]	-	-	-	-	-	-	-						
ML-KELM [95]	-	-	-	-	-	-	-	-					

+ means the method in the corresponding row is statistically better than the method in the corresponding column. Similarly, - indicates that the method in the corresponding row is statistically worse than the method in the corresponding column.

Table 5.6: Comparison of Hidden Nodes and Training Time of each algorithm in the **bank** dataset

Algorithm	Hidden Nodes	Total Hidden Nodes	Training Time (ms)
H-ELM(l_1) [96]	N1=N2=170; N3=1800	2140	608.8
sdRVFL(d- l_1)	N1=N2=160; N3=900	1220	397.8
sdRVFL(dense- l_1)	N1=N2=110; N3=400	620	148.1
sdRVFL-D(dense- l_1)	N1=N2=140; N3=600	880	250.9
H-ELM(l_2) [92]	N1=N2=110; N3=1700	1920	612.6
sdRVFL(d- l_2)	N1=N2=80; N3=400	560	109.3
sdRVFL(dense- l_2)	N1=N2=90; N3=300	480	112.6
sdRVFL-D(dense-l_2)	N1=N2=60; N3=300	420	100.8
H-ELM(elas) [93]	N1=N2=40; N3=700	780	157.9
sdRVFL(d-elas)	N1=N2=10; N3=700	720	139.4
sdRVFL(dense-elas)	N1=N2=100; N3=100	300	104.6
sdRVFL-D(dense-elas)	N1=N2=190; N3=200	580	178.2
ML-KELM [95]	-	-	3470

The best performing algorithm for this dataset is in bold.

Table 5.7: Comparison of Hidden Nodes and Training Time of each algorithm in the **glass** dataset

Algorithm	Hidden Nodes	Total Hidden Nodes	Training Time (ms)
H-ELM(l_1) [96]	N1=N2=10; N3=1800	1820	14.7
sdRVFL(d- l_1)	N1=N2=80; N3=500	660	10.5
sdRVFL(dense-l_1)	N1=N2=30; N3=400	460	9.4
sdRVFL-D(dense- l_1)	N1=N2=50; N3=400	500	9.7
H-ELM(l_2) [92]	N1=N2=130; N3=700	960	13.7
sdRVFL(d- l_2)	N1=N2=90; N3=600	780	10.9
sdRVFL(dense- l_2)	N1=N2=40; N3=100	180	6.8
sdRVFL-D(dense- l_2)	N1=N2=50; N3=100	200	7.8
H-ELM(elas) [93]	N1=N2=150; N3=1000	1300	12.8
sdRVFL(d-elas)	N1=N2=140; N3=200	480	9.4
sdRVFL(dense-elas)	N1=N2=30; N3=100	160	4.05
sdRVFL-D(dense-elas)	N1=N2=40; N3=100	180	4.12
ML-KELM [95]	-	-	11.1

5.3.3 Performance Comparison with state-of-the-art Deep Neural Networks

In this section, we evaluate the performance of our proposed methods on two datasets described below and compare with other state-of-the-art neural networks.

Income Prediction: The income prediction dataset [144] consists of a person’s social background such as race, sex, work-class etc. and the task is to predict whether this person makes over 50K a year. The dataset consists of 32,561 training samples and 16,281 testing samples. We evaluate the performance of our proposed methods using the network structure as described in [108]. Specifically, neural networks with two hidden layers trained using the target propagation $NN^{TargetProp}$ [145] and standard back-propagation $NN^{BackProp}$ respectively are implemented in [108]. We use the same number of hidden layers and compare the performance of neural network based methods³ in Table 5.8. For other hyperparameters, we follow the experimental settings described in Section 5.3. From the table, we can see that our methods are either comparable or outperform the neural networks trained with both target propagation and standard back-propagation.

Prediction of pulsars in the HTRU2 dataset: The High Time Resolution Universe Survey (HTRU2) dataset [146] consists of 1,639 real pulsars and 16,259 spurious signals. The task here, is to identify pulsars in radio signals. For the comparison, we use the experimental settings of [1]. The number of hidden units and layers are selected from the values presented in Table 5.9. For other hyperparameters, we follow the experimental settings described in Section 5.3. We compare

³Multi-Layered Gradient Boosting Trees (mGBDT) has the best performance of 87.42% in this dataset. For the sake of comparison, we consider only neural network based methods.

Table 5.8: Comparison of classification accuracy in the income prediction dataset.

Method	Accuracy(%)
$NN^{TargetProp}$	84.91
$NN^{BackProp}$	85.34
sdRVFL(dense- l_1) [†]	85.38
sdRVFL(dense- l_2) [†]	85.33
sdRVFL(dense-elas) [†]	85.34
sdRVFL-D(dense- l_1) [†]	85.42
sdRVFL-D(dense- l_2) [†]	85.36
sdRVFL-D(dense-elas) [†]	85.41

[†] are the methods introduced in this paper. The results for $NN^{TargetProp}$ and $NN^{BackProp}$ are copied from [108]. For a two-hidden layer neural network, sdRVFL(d) and sdRVFL(dense) have the same network structure. Thus, we report the performance for the latter i.e. sdRVFL(dense).

Table 5.9: Hyperparameters considered for the HTRU2 dataset [1].

Hyperparameter	Considered Values
Number of hidden units	{256,512,1024}
Number of hidden layers	{2,4,8,16}

our methods with seven other deep neural networks (top-7 of [1]) in terms of AUC in Table 5.10. From the table, we can see that our methods are either comparable or outperform the other state-of-the-art feed-forward neural networks (FNNs).

5.4 Conclusion

In this chapter, we proposed three deep RVFL variants using H-ELM framework. Specifically, inspired by the better performance of RVFL over ELM, we extended the H-ELM framework by incorporating direct connections from different parts of the network as in the original RVFL. The deep networks are thus, labelled as sdRVFL. The RVFL classifier at the rear end of the network utilizes diverse features from different feature extraction layers for the final decision making. To extract better higher level representations, we also introduced the denoising concept with such networks. Thus, our proposed methods achieve better generalization compared to the other relevant state-of-the-art multi-layer ELMs such as H-ELM

Table 5.10: Comparison of neural networks in the HTRU2 dataset in terms of AUC.

Method	AUC
SNN	0.9803±0.010
MSRAinit	0.9791±0.010
WeightNorm	0.9786±0.010
Highway	0.9766±0.009
LayerNorm	0.9762±0.011
BatchNorm	0.9760±0.013
ResNet	0.9753±0.010
sdRVFL(d- l_1) [†]	0.9805±0.001
sdRVFL(d- l_2) [†]	0.9766±0.009
sdRVFL(d-elas) [†]	0.9800±0.001
sdRVFL(dense- l_1) [†]	0.9809 ± 0.008
sdRVFL(dense- l_2) [†]	0.9768±0.009
sdRVFL(dense-elas) [†]	0.9809±0.001
sdRVFL-D(dense- l_1) [†]	0.9810±0.001
sdRVFL-D(dense- l_2) [†]	0.9788±0.011
sdRVFL-D(dense-elas) [†]	0.9811±0.00

[†] are the methods introduced in this paper. The results for other seven deep neural networks are copied from [1].

and ML-KELM. Furthermore, our proposed methods have lower model complexity and can be trained faster.

Chapter 6

RVFL based Ensemble Deep Learning

In the previous chapter, we presented several deep RVFL networks using randomization based autoencoders. In this chapter, we will present another deep RVFL variant that is a derivative of the shallow RVFL network. Specifically, the deep RVFL network is built by stacking several hidden layers whose parameters are randomly generated and kept fixed as in shallow RVFL network. We will also present an implicit ensemble of such deep RVFL network [147].

6.1 Introduction

Training neural networks with standard back-propagation technique entails longer training times. Thus, ensembles of neural networks can quickly become uneconomical as the training cost increases linearly. RVFL neural network, on the other hand, can be trained very quickly within seconds or minutes compared to hours or days required by BP-trained neural networks [27, 83, 96]. In this chapter, we investigate the performance of deep learning and ensemble deep learning models based on RVFL networks. Thus, we present deep neural networks based on RVFL while maintaining its advantages of lower complexity, training efficiency and good generalization. We also present an ensemble of such deep networks without incurring any significant training costs. Specifically, we propose an ensemble deep RVFL network which can be regarded as a marriage of ensemble and deep learning that is simple and straight-forward to implement. The key contributions of this chapter are summarized as follows:

- We propose a deep RVFL network (dRVFL), an extension of RVFL for representational learning. The dRVFL network consists of several hidden layers stacked on top of each other. The parameters of the hidden layers are randomly generated and kept fixed while only the output weights need to be computed. Thus, the deep RVFL network emanates from the standard RVFL

network.

- We also propose an implicit ensembling approach called ensemble deep RVFL framework (edRVFL), a marriage of ensembling learning with deep learning. Instead of training L neural networks independently from scratch as in traditional ensembling method, we only train a single deep RVFL network. The ensemble consists of L models equivalent to the number of hidden layers in the single deep RVFL network. The ensemble is trained in such a way that the higher models (equivalent to higher layers in deep RVFL network) utilize both the original features (from direct links as in standard RVFL network) and non-linearly transformed features from the preceding layers. Thus, the framework is consistent with the tenets of both ensemble learning and deep learning at the same time. The training cost of edRVFL is slightly higher than that of a single dRVFL network while it is significantly lower than that of traditional ensembles.
- The deep learning models (dRVFL and edRVFL) presented in this chapter are generic and are applicable with any RVFL variant. We create deep learning models using both standard RVFL and recently proposed sparse pre-trained RVFL (SP-RVFL) [90].
- With extensive experiments on several real-world classification datasets, we show that our proposed deep RVFL models (dRVFL and edRVFL) have superior performance compared to other relevant neural networks.

6.2 Deep RVFL for representational learning

In this section, we introduce our proposed deep learning frameworks based on RVFL. We first describe the deep RVFL network in Section 6.2.1. We then elucidate our proposed ensemble deep RVFL network in Section 6.2.2.

6.2.1 Deep Random Vector Functional Link Network

The Deep Random Vector Functional Link (dRVFL) network is an extension of the shallow RVFL network in the context of representation learning or deep learning. The dRVFL network is typically characterized by a stacked hierarchy of hidden layers as shown in Fig. 6.1. The input to each layer in the stack is the output of the preceding layer wherein each layer builds an internal representation of the input data. Although the stacked hierarchical organization of hidden layers in dRVFL network allows a general flexibility in the size (both in width and depth) of the network, for the sake of simplicity here we consider a stack of L hidden layers each of which contains the same number of hidden nodes N .

For the ease of notation, we omit the bias term in the formulas. The output of

the first hidden layer is then defined as follows:

$$\mathbf{H}^{(1)} = g(\mathbf{X}\mathbf{W}^{(1)}), \quad (6.1)$$

while for every layer > 1 it is defined as:

$$\mathbf{H}^{(L)} = g(\mathbf{H}^{(L-1)}\mathbf{W}^{(L)}), \quad (6.2)$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times N}$ and $\mathbf{W}^{(L)} \in \mathbb{R}^{N \times N}$ are the weight matrices between the input-first hidden layer and inter hidden layers respectively. These parameters (weights and biases) of the hidden neurons are randomly generated within a suitable range and kept fixed during the training. $g(\cdot)$ is the non-linear activation function. The input to the output layer is then defined as:

$$\mathbf{D} = [\mathbf{H}^{(1)} \ \mathbf{H}^{(2)} \ \dots \ \mathbf{H}^{(L-1)} \ \mathbf{H}^{(L)} \ \mathbf{X}]. \quad (6.3)$$

This design structure is very similar to the standard shallow RVFL network wherein the input to the output layer consists of non-linear features from the stacked hidden layers along with the original features. The output of the dRVFL network is then defined as follows:

$$Y = \mathbf{D}\boldsymbol{\beta}_d. \quad (6.4)$$

The output weight $\boldsymbol{\beta}_d \in \mathbb{R}^{(NL+d) \times K}$ (K : the number of classes) is then solved using Eq. (2.24) and Eq. (2.25).

From Eq. (6.3) and Eq. (6.4), one can see that in dRVFL there exists a linear combination between the features and the output layer weight matrix $\boldsymbol{\beta}_d$ i.e. a weighted sum of the features from the hidden layers including the input layer. In the training stage, this directly enables the model to weigh differently the contribution of each type of features originating from different layers.

It is also worth mentioning that our proposed dRVFL network differentiates itself from the deep learning architecture proposed in [148] in threefold: 1) dRVFL is inspired by a shallow RVFL network with the output layer consisting of both non-linearly transformed features and original features via direct links. In contrast, the deep learning architecture proposed in [148] does not consider the original features during the output weight computation, 2) we investigate the performance of the dRVFL network in classification problems while [148] explores time-series problems, and 3) the deep learning framework, dRVFL is generic and can be used with any RVFL variant.

Figure 6.1: Framework of a dRVFL network. It consists of several hidden layers stacked on top of each other whose parameters (weights and biases between the hidden layers) are randomly generated and kept fixed during the training. Only the output weights β_d need to be computed as in the shallow RVFL network.

6.2.2 Ensemble Deep Random Vector Functional Link Network

The framework of the ensemble deep RVFL network (edRVFL) is shown in Fig. 6.2. It serves three purposes: 1) instead of using only the higher level representations (features extracted from the final hidden layer) of the data as in a conventional deep learning model [96] for classification, it employs rich intermediate features also for final decision making. 2) the ensemble is obtained by training a single dRVFL network once with a training cost slightly higher than that of a single dRVFL but cheaper than training several independent models of dRVFL. 3) like dRVFL, the edRVFL framework is generic and any RVFL variant can be used with it.

The computation of the output weight β_d in the dRVFL network described in Section 6.2.1 requires matrix inversion of size either $(T \times T)$ or $((NL + d) \times (NL + d))$, whichever is small (cf. to primal and dual solutions, Eq. (2.24) and Eq. (2.25)) where T is the training data size, L is the number of hidden layers, N is the number of hidden nodes at each layer and d is the dimension of the data. For simplicity, we omit the bias terms. In a standard implementation, the matrix inversion of a matrix of size $(T \times T)$ requires $\mathcal{O}(T^3)$ time and $\mathcal{O}(T^2)$ memory [143]. In case of dRVFL, this is equivalent to either $\mathcal{O}(T^3)$ or $\mathcal{O}((NL + d)^3)$ time and either $\mathcal{O}(T^2)$ or $\mathcal{O}((NL + d)^2)$ memory. Such scaling is prohibitive when all N , L , T and d are large. Inversion of a large matrix can also result in out-of-memory failures thus, requiring powerful and high performance hardware [143]. Depending on the

dataset, all these parameters can be actually large. For example, a dataset used in this chapter has 7000 training samples with 5000 features. A dRVFL network with 10 hidden layers and 100 hidden nodes in each layer, requires matrix inversion of size (6000×6000) using primal solution (using dual solution would require matrix inversion of size (7000×7000)). Thus, we decompose the computation of the final output weight β_d of dRVFL into several small β_{ed} in edRVFL. Specifically, each small β_{ed} is independently computed (treated as independent model) and the final output is obtained by using either majority voting or averaging of the models. Each small β_{ed} requires the matrix inversion of size either $(T \times T)$ or $((N + d) \times (N + d))$. Without direct links, it would require an inversion of size either $(T \times T)$ or $(N \times N)$. However, as discussed in the preceding sections, direct links are essential parts of randomized neural networks [25, 76, 78, 82, 83]. The significance of such direct links is also discussed later in Section 6.3.4.

The input to each hidden layer is the non-linearly transformed features from the preceding layer as in dRVFL along with the original input features (direct links) as in standard RVFL. The direct links act as a regularization for the randomization. The output of the first hidden layer is then defined as follows:

$$\mathbf{H}^{(1)} = g(\mathbf{X}\mathbf{W}^{(1)}), \quad (6.5)$$

while for every layer > 1 it is defined as:

$$\mathbf{H}^{(L)} = g([\mathbf{H}^{(L-1)}\mathbf{X}]\mathbf{W}^{(L)}). \quad (6.6)$$

The output weights β_{ed} are then solved independently using Eq. (2.24) and Eq. (2.25).

The mechanism of obtaining several models while training only a single model as in ensemble deep RVFL network (edRVFL) is related to the implicit ensembling techniques described in Chapter 2. Like other implicit ensembles, edRVFL can even be ensembled further if enough resources are available during training.

6.3 Experiments

6.3.1 Datasets

The experiments are performed on 13 publicly available real-world classification datasets from various domains used in [90] which include two biomedical datasets (Carcinom and Lung), two human face image datasets (ORL and Yale), four hand-written digit datasets (Binary Alphabet(BA), Gisette, a portion of MNIST and USPS), two object recognition datasets (COIL20, COIL100) and three text datasets (BASEHOCK, RCV1 and TDT2). Table 6.1 gives an overview of the 13 real-world application datasets.

Figure 6.2: Framework of ensemble deep RVFL network (edRVFL). It differs from dRVFL in that the computation of final output weight β_d is decomposed into several small β_{ed} . Specifically, each small β_{ed} is independently computed (treated as independent model) and the final output is obtained by using either majority voting or averaging of the models. Each higher level model is fed with original input data and the non-linearly transformed features from the preceding model. $O_1, \dots, O_L$ represents the output of each model.

6.3.2 Compared Methods

To verify the effectiveness of our proposed deep learning frameworks, we perform comparisons against relevant algorithms (shallow RVFL networks, randomization based multi-layer neural networks and ensembles of RVFL). The compared methods are enumerated as follows:

1. ELM: extreme learning machine [12]; shallow RVFL without direct links and bias.
2. RVFL: standard shallow RVFL network [11].
3. SP-RVFL: sparse pre-trained RVFL [90]; state-of-the-art RVFL network.
4. HELM: hierarchical ELM [96], a multi-layer network based on ELM; has superior performance compared to other relevant deep learning methods such as Stacked Auto-Encoders (SAE) [139], Stacked Denoising Auto-Encoders (SDA) [89], Deep Belief Networks (DBN) [149], Deep Boltzmann Machines (DBM) [150].
5. dRVFL: deep RVFL proposed in this chapter.
6. dRVFL(-O): dRVFL without direct links but with bias.

Table 6.1: Overview of the datasets.

Domain	Dataset	#Patterns	#Features	#Class
Biology	Carcinom	174	9182	11
	Lung	203	3312	5
Face	ORL	400	1024	40
	Yale	165	1024	15
Handwritten Digits	BA	1404	320	36
	Gisette	7000	5000	2
	MNIST	4000	748	10
	USPS	1000	256	10
Object	COIL20	1440	1024	20
	COIL100	7200	1024	100
Text	BASEHOCK	1993	1000	2
	RCV1	9625	1000	4
	TDT2	9394	1000	30

For datasets preprocessing and further details, please refer to [90].

7. edRVFL: ensemble deep RVFL proposed in this chapter.
8. edRVFL(-O): edRVFL without direct links but with bias.
9. dSP-RVFL: SP-RVFL based dRVFL
10. edSP-RVFL: SP-RVFL based edRVFL

6.3.3 Experimental Settings

To compare the different algorithms, we follow the experimental settings of [90]. The number of hidden neurons N is set to 100 [90]. For deep RVFL based methods, the same number of hidden neurons is used at each layer with the maximum number of hidden layers L set to 10 for each dataset. Meanwhile, the regularization parameter λ in each layer is set as $(1/C)$ where C is tuned over the range $2^x, \{x = -6, -4, -2, \dots, 12\}$. The experimental results reported are obtained by averaging results from 10-fold cross-validation.

6.3.4 Performance Comparison and Analysis

In this section, we compare our proposed deep RVFL based frameworks against pertinent methods. Specifically, we first compare standard RVFL based methods followed by SP-RVFL based methods respectively.

Comparison between standard RVFL based methods

The classification accuracy of each algorithm in each dataset is presented in Table 6.2. From the table, one can see that the ensemble deep RVFL (edRVFL) proposed in this chapter has the best accuracy in 12 out of 13 datasets. The edRVFL has comparable performance to dRVFL in 3 datasets while it outperforms dRVFL in all other datasets. We follow the procedure of [2] and use the Friedman rank of each classifier to assess its performance. From the same table, one can see that edRVFL is the top ranked algorithm followed by dRVFL.

We also perform a statistical comparison of the algorithms using the Friedman test. Based on simple calculations we obtain, $\chi_F^2 = 68.11$ and $F_F = 82.64$. With 7 classifiers and 13 datasets, F_F is distributed according to the F -distribution with $7 - 1 = 6$ and $(7 - 1)(13 - 1) = 72$ degrees of freedom. The critical value for $F_{(6,72)}$ for $\alpha = 0.05$ is 2.22, so we reject the null-hypothesis. Based on the Nemenyi test, the critical difference is $CD = q_\alpha \sqrt{(m(m + 1))/(6M)} = 2.948 * \sqrt{7 * 8/(6 * 13)} \simeq 2.49$. From Fig. 6.3, we can see that edRVFL is statistically better than ELM, RVFL, HELM and dRVFL(-O) while dRVFL is statistically better than ELM, RVFL, and dRVFL(-O). The difference of ranks between the randomized multi-layer networks, HELM and dRVFL is 2.35 (0.14 less than CD). The dRVFL has superior performance (around 5.4% times more accurate) in almost all the datasets compared to HELM except BA dataset. Some of the biggest improvements of dRVFL over HELM in average are in Carcinom (8.01%), ORL (7.75%), Yale (15.66%), COIL100 (14.7%), RCV1 (5.06%) and TDT2 (10.91%) datasets. This indicates the superior generalization ability (representational capability) of dRVFL over HELM.

In addition, we select 5 datasets from each domain (Lung from biology, ORL from face, USPS from digits, COIL20 from object and RCV1 from text), and report the experimental results for different number of hidden nodes in dRVFL and edRVFL in Fig. 6.4. The number of hidden nodes in each dataset is varied from 10 to 100 with a step-size of 10. One can see from Fig. 6.4, setting $N = 100$ is appropriate for both dRVFL and edRVFL. Increasing the number of hidden nodes generally increases the performance of the network until some point after which it becomes stable. Similarly, we also compare the training and testing times of dRVFL and edRVFL in these 5 datasets in Fig. 6.5. As can be seen from the figure, the training times of both dRVFL and edRVFL increase with the increase in the number of hidden nodes while there is only a slight increase in the testing time for both cases.

Comparison between RVFL and dRVFL

From Table 6.2, it can be observed that the deep learning framework proposed in this chapter, dRVFL, is more accurate than its baseline (shallow) RVFL network by approx. 5.74%. Some of the biggest improvements are in Yale (9.67%), BA (7.91%), Gisette (6.09%), COIL20 (6.24%), BASEHOCK (9.85%), RCV1 (6.18%) and TDT2 (14.59%) datasets. We also compare dRVFL and RVFL with the same number of hidden nodes in each dataset in Fig. 6.6. The dRVFL network is on average 3.61% more accurate than shallow RVFL network with the same number of hidden nodes. This accentuates the benefits of representational learning in

Table 6.2: Accuracy (%) of the standard RVFL based methods on all the datasets

Dataset	ELM [90]	RVFL [90]	HELM [96]	dRVFL [†]	dRVFL(-O) [†]	edRVFL(-O) [†]	edRVFL [†]
Carcinom	62.94±12.46	97.05±3.42	90.85±7.13	98.86±2.41	80.46±10.94	97.12±4.01	98.86±2.41
Lung	88.5±9.44	95.5±4.38	95.57±5.45	97.05±4.19	92.52±8.91	96.57±4.03	97.05±4.19
ORL	69±6.69	94.5±2.43	91.25±3.58	99±1.75	89±3.57	99±1.75	99±1.75
Yale	59.38±12.5	77.5±11.51	71.51±5.78	87.17±7.83	70.85±12.83	86.03±7.96	88.58±6.92
BA	52.21±5.21	57.42±4.27	67.09±3.31	65.33±4.11	55.13±2.17	59.97±3.48	66.11±3.34
Gisette	83.04±2.11	92.07±1.17	95.17±0.93	98.16±0.48	83.39±2.1	98.17±0.55	98.21±0.55
MNIST	79.13±1.74	88.11±1.15	87.6±1.35	88.12±1.32	84.07±1.71	86.52±2.06	92.8±1.92
USPS	91.1±1.6	91.9±2.57	91.35±2.2	93.3±2.18	91.45±3	92.3±1.93	94.1±2.07
COIL20	92.78±1.43	93.41±2.18	98.54±0.51	99.65±0.49	98.54±0.69	98.82±0.93	99.86±0.29
COIL100	64.07±2.09	85.16±1.4	75.28±1.07	90.06±0.91	81.65±0.72	87.51±0.77	90.5±0.82
BASEHOCK	73.37±3.63	88.19±3.18	96.39±1.22	98.04±1.07	83.59±2.54	97.34±1.11	98.04±0.76
RCV1	79.51±1.74	87.57±0.73	88.69±1.25	93.75±0.71	79.52±2.18	92.34±0.76	93.86±0.69
TDT2	61.32±2.21	81.92±0.83	85.6±0.68	96.51±0.6	80.31±1.28	94.73±0.43	96.51±0.53
Mean Acc.	73.56±4.83	86.95±3.01	87.29±2.65	92.69±2.15	82.34±4.09	91.26±2.29	93.35±2.01
Avg. Friedman Rank	7	4.54	4.35	2	5.73	3.08	1.3

The results for ELM and RVFL are directly copied from [90]. [†] are the methods introduced in this chapter. The best results for each dataset is given in bold. Lower rank reflects better performance.

Figure 6.3: Statistical comparison of classifiers against each other based on Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.05$) are connected.

case of multi-layer (deep) networks wherein each hidden layer extracts meaningful feature representation from its input.

Comparison of edRVFL with True Ensembles

Here, we compare the implicit ensembles of dRVFL (edRVFL) with its true ensemble (TedRVFL) in terms of performance and training complexities. The true ensemble method, TedRVFL averages dRVFL methods trained independently as in [10]. The training and testing accuracy of edRVFL and TedRVFL is presented in Fig. 6.7. For edRVFL, the number of models corresponds to the number of hidden layers L while for TedRVFL, this corresponds to an ensemble of L dRVFL models with L hidden layers. As can be observed from the figure, the training accuracy of both edRVFL and TedRVFL increase with the increase in the number of models. However, this isn't the case for the test accuracy thus, requiring the best parameter search (in this case L). However, from Fig. 6.7, one can see that with proper selection of L , the edRVFL can achieve either comparable or even better test accuracy compared to TedRVFL. Similarly, we also compare the training and testing times of edRVFL and TedRVFL in 5 datasets in Fig. 6.8. As can be seen from the figure, the training times of both edRVFL and TedRVFL increase with the increase in the number of models while there is only a slight increase in the testing times for both cases. As the number of models increases, the training time of TedRVFL increases sharply while that for edRVFL only increases slightly. In a nutshell, edRVFL has comparable or better performance than TedRVFL while requiring significantly less training time.

Effect of direct links

The significance of direct links in case of randomized shallow neural networks has been extensively articulated in the literature. Here, we investigate the effect of direct links in the case of randomization based deep neural networks, specifically in the dRVFL and edRVFL methods introduced in this chapter. The dRVFL(-O) and edRVFL(-O) are the deep learning methods equivalent to dRVFL and edRVFL respectively without direct links. The experimental results of dRVFL(-O) and edRVFL(-O) on the real-world classification datasets are presented in Table 6.2. The dRVFL(-O) differs from HELM [96] in that HELM uses only the

Figure 6.4: Comparison of dRVFL and edRVFL in terms of accuracy (%) w.r.t different number of hidden nodes.

Figure 6.5: Training and testing times comparison of dRVFL and edRVFL w.r.t different number of hidden nodes. For the same number of hidden nodes, the training time of edRVFL is in average 3 times more than that of dRVFL.

Figure 6.6: Comparison of dRVFL and RVFL in terms of accuracy (%) with the same number of hidden nodes.

last layer features extracted from the feature extractor part for final classification while dRVFL uses all the hidden layer features. From the table, we can see that the difference in accuracy between dRVFL and dRVFL(-O) is approx. 10.35% while that between edRVFL and edRVFL(-O) is approx. 2.09%. In dRVFL (refer to Fig. 6.1), in addition to the randomly generated hidden layer features, the input to the output nodes contains original features via the direct links. This enables the network to weigh higher the discriminative features while ignoring the redundant or less-important features. As in the case of randomized shallow neural networks, the direct links act as regularization for the randomization. Without direct links, the inputs to the output of dRVFL is simply the hidden layer features (generated randomly) resulting in a sub-optimal performance. To avoid such issue (sub-optimal performance) in case of edRVFL, the input to each hidden layer is a concatenation of original features via direct links and the hidden layer features from the preceding layer (except in case of the first hidden layer).

Parameter Sensitivity Analysis

In dRVFL and edRVFL, the number of hidden layers L , the number of hidden nodes N and the regularization parameter C need to be properly selected for each dataset. To further analyze the dRVFL and edRVFL methods, we conduct a sensitivity study for the parameters L , H and C in case of two datasets COIL20 and USPS. The parameters H and C are common parameters for both shallow and deep RVFL based methods while only parameter L is relevant for deep learning methods. We show the experimental results in two different settings: varying L while keeping H fixed and varying H while keeping L fixed. Specifically, we employ a grid search strategy to vary these parameters. The parameter L is varied from 2:10 with a step-size of 1, the parameter H is varied from 10:100 with a step-size of 10. Similarly, the C parameter is varied within 2^x , $\{x = -6, -4, -2, 0, 2, 4, 6, 8, 10, 12\}$. As can be seen from Figs. 6.9 and 6.10, different combination of the parameters results in different performance. Therefore, it is necessary to determine the suitable values of the parameters L , H and C for each dataset.

Figure 6.7: Comparison of edRVFL (implicit ensemble) and TedRVFL (true ensemble) in terms of accuracy (%).

Figure 6.8: Training and testing times comparison of edRVFL (implicit ensemble) and TedRVFL (true ensemble).

Figure 6.9: Performance variation of the proposed dRVFL (first row) and edRVFL(second row) methods in terms of accuracy (%) for fixed H. Different parameter combinations may result in different performance.

Figure 6.10: Performance variation of the proposed dRVFL (first row) and edRVFL(second row) methods in terms of accuracy (%) for fixed L. Different parameter combinations may result in different performance.

Comparison between SP-RVFL based methods

The deep learning frameworks dRVFL and edRVFL proposed in Sections 6.2.1 and 6.2.2 respectively are generic and any RVFL network can be used as a base model with both dRVFL and edRVFL. Here, we use SP-RVFL [90], a recently proposed state-of-the-art RVFL network to create dRVFL and edRVFL networks. Specifically, we term the deep architecture using SP-RVFL as deep sparse pre-trained RVFL (dSP-RVFL) and its ensemble as edSP-RVFL. The SP-RVFL is described in Section 2.3.1.6. We run the experiments on all the datasets presented in Table 6.1 and report the experimental results of dSP-RVFL and edSP-RVFL in Table 6.3. From the table, one can see that our deep learning frameworks dSP-RVFL and edSP-RVFL have better performance compared to SP-RVFL. Also, edSP-RVFL has either same or better performance than dSP-RVFL.

Table 6.3: Comparison between SP-RVFL based methods in terms of accuracy (%)

Dataset	SP-RVFL [90]	dSP-RVFL	edSP-RVFL
Carcinom	97.64±3.04	98.27±2.79	98.27±2.79
Lung	96±4.38	97.05±4.19	97.05±4.19
ORL	97.25±2.3	99.5±1.58	99.5±1.58
Yale	86.87±9.88	87.17±7.38	87.17±7.38
BA	65.57±5.66	68.52±2.65	73.08±2.67
Gisette	98.13±1.45	98.21±0.41	98.21±0.41
MNIST	91.26±0.82	93.43±1.4	95.02±1.5
USPS	92.9±1.97	93.5±1.68	95±1.6
COIL20	98.75±1.39	98.96±0.67	99.72±0.36
COIL100	89.76±1.24	93.38±1.26	94.26±0.86
BASEHOCK	91.4±2.77	97.79±0.98	97.79±0.98
RCV1	93.38±0.61	93.9±0.61	94.6±0.5
TDT2	93.29±0.86	96.17±0.63	96.17±0.63
Mean Acc.	91.7±2.58	93.52±2.01	94.29±1.95
Avg. Friedman Rank	3	1.73	1.26

The results for SP-RVFL are directly copied from [90] for all the datasets except for the RCV1 dataset where we were unable to replicate the reported result (94.84±0.64) within the given level of variability. Thus, SP-RVFL's performance in RCV1 is based on our implementation.

We also perform a statistical comparison of the algorithms using the Friedman test as in Section 6.3.4. Based on simple calculations we obtain, $\chi_F^2 = 20.54$ and $F_F = 71.23$. With 3 classifiers and 13 datasets, F_F is distributed according to the F -distribution with $3 - 1 = 2$ and $(3 - 1)(13 - 1) = 24$ degrees of freedom. The critical value for $F_{(2,24)}$ for $\alpha = 0.05$ is 3.4, so we reject the null-hypothesis. Based on the Nemenyi test, the critical difference is $CD = q_\alpha \sqrt{(m(m+1))/(6M)} = 2.344 * \sqrt{3 * 4/(6 * 13)} \simeq 0.92$. From Table 6.3, one can see that both dSP-RVFL

Table 6.4: Average Friedman rank based on classification accuracy of each method

Algorithm	Ranking
ELM	10
dRVFL(-O) [†]	8.73
RVFL	7.54
HELM	7.19
edRVFL(-O) [†]	5.61
SP-RVFL	5.46
dRVFL [†]	3.46
dSP-RVFL [†]	2.65
edRVFL [†]	2.3
edSP-RVFL [†]	2.03

[†] are the methods introduced in this chapter. Lower rank reflects better performance.

and edSP-RVFL are statistically significantly better than their baseline method, SP-RVFL.

Overall Comparison

We present an overall comparison of the algorithms using the Friedman rank in Table 6.4. From the table, one can see that the deep learning frameworks introduced in this chapter have the best ranks compared to other algorithms. Specifically, the ensemble deep learning frameworks (edSP-RVFL and edRVFL) obtain the top ranks followed by the single deep learning frameworks (dSP-RVFL and dRVFL).

Performance Comparison with other state-of-the-art Deep Neural Networks

Here, we compare the top two ranked algorithms, edRVFL and edSP-RVFL against other standard feed-forward neural networks (FNNs). The standard FNNs are the methods employed in [1] which are listed below.

1. SNN: Self-normalizing neural network
2. MSRAinit: FNNs without normalization and with relu activation function and Microsoft weight initialization
3. WeightNorm: FNNs with weight normalization
4. Highway: Highway networks
5. LayerNorm: FNNs with layer normalization

Table 6.5: Hyperparameters considered for deep RVFL networks

Hyperparameter	Considered Values
Number of hidden units	{512,1024,2048,4096}
Maximum number of hidden layers	32
layer form	rectangular

Table 6.6: Hyperparameters considered for other FNNs [1]

Hyperparameter	Considered Values
Number of hidden units	{256,512,1024}
Number of hidden layers	{2,3,4,8,16,32}
learning rate	{0.01,0.1,1}
dropout rate	{0,0.05,0.5}
layer form	{rectangular,conic}

6. BatchNorm: FNNs with batch normalization

7. ResNet: Residual networks adapted to FNNs using residual blocks with 2 or 3 layers

In Tables 6.5 and 6.6, we show the hyperparameters considered for deep RVFL and other FNNs respectively. “Rectangular” layers have constant number of hidden units in each layer, while “conic” layers start with the given number of hidden units in the first layer and then decrease the number of hidden units to the size of the output layer according to the geometric progression. The best hyperparameters are obtained using the validation set. For the comparison, we use 10 UCI datasets with varying sample sizes where N refers to the number of training data points and M refers to the number of input features. The performance of each method in each dataset is shown in Table 6.7. The results for the FNN methods are taken from [1]. From Table 6.7, we can see that the deep RVFL neural networks are competitive against other deep neural networks in tabular datasets.

Table 6.7: Comparison of FNN methods

Dataset	N	M	SNN	MS	HW	ResNet	BN	WN	LN	edRVFL [†]	edSP-RVFL [†]
adult	48842	15	84.76	84.87	84.53	84.84	84.99	84.53	85.17	84.98	85.45
connect-4	67557	43	88.07	88.31	85.99	87.16	87.29	88.33	88.56	86	87.07
hill-valley	1212	101	52.48	51.16	50	53.96	50.5	49.34	50.5	58.42	60.23
image-seg	2310	19	91.14	90.9	90.24	89.19	84.81	89.38	88.38	91.9	92.24
letter	20000	17	97.26	97.12	89.84	97.62	97.96	95.8	97.42	97.53	97.6
magic	19020	11	86.92	86.29	86.73	87.23	87.13	86.9	86.2	86.62	86.54
miniboone	130064	51	93.07	92.5	92.7	92.54	92.62	92.72	93.13	92.67	92.8
optical	5620	63	97.11	96.66	96.44	96.27	97.16	96.38	97.55	98.11	97.89
pendigits	10992	17	97.06	97.14	96.71	97.08	97.34	96.2	96.57	97.43	97.48
st-landsat	6435	37	91	90.75	91.1	90.55	90.4	89.25	90.4	90.75	91.6
Avg. Rank	-	-	4.3	5.55	6.55	5.3	4.8	6.75	5.1	3.95	2.7

[†] refers to the ensemble deep RVFL networks developed in this thesis. The results for other FNNs are taken from [1]. Lower rank refers better performance.

6.4 Conclusion

In this chapter, we first presented a deep learning model (dRVFL) based on random vector functional link network. As in a standard RVFL network, the parameters of the hidden layers were randomly generated and kept fixed with the output weights computed analytically using a closed form solution. The dRVFL network while extracting rich feature representations through several hidden layers also acts as a weighting network thereby, providing a weight to features from all the hidden layers including the original features obtained via direct links. We then proposed an ensemble dRVFL, edRVFL, which combines ensemble learning with deep learning. Instead of training several models independently as in traditional ensembles, edRVFL can be obtained by training a deep network only once. The training cost of edRVFL is slightly greater than that of a single dRVFL network but significantly lower than that of the traditional ensembles. Both dRVFL and edRVFL are generic and any RVFL variant can be used with them. To demonstrate this generic nature, we developed sparse-pretrained RVFL (SP-RVFL) based deep RVFL networks (dSP-RVFL and edSP-RVFL). The SP-RVFL uses a sparse-autoencoder to learn the hidden layer parameters of RVFL as opposed to randomly generating them as in standard RVFL. Extensive experiments on 23 classification datasets showed that our deep learning RVFL networks achieve better generalization compared to pertinent randomized neural networks.

Chapter 7

Summary

The first part of this thesis is based on heterogeneous ensemble classifiers. In Chapter 3, we presented a heterogeneous ensemble classifier based on the fusion of a fast neural network, RVFL and decision trees. In Chapter 4, we proposed a random forest of heterogeneous oblique decision trees that employs several linear classifiers at each non-leaf node. The second part of this thesis is based on deep randomized neural networks. In Chapter 5, we developed several deep RVFL variants constructed via randomization based autoencoders. Similarly, in Chapter 6, we presented another variant of deep RVFL that mirrors standard RVFL network and an ensemble deep RVFL that fuses ensemble learning with deep learning. The experimental results on several datasets demonstrated the effectiveness of the proposed methods.

In this chapter, we conduct an overall comparison among the top ranked classifiers presented in the preceding chapters. Specifically, the methods include hybrid ensemble classifiers (RaFL and obRaFL) from Chapter 3, top three ranked oblique random variants (obRaF(M), obRaF(L) and obRaF(H)) from Chapter 4, top three ranked deep RVFL variants (MRVFL-D(dense- l_2), MRVFL(dense- l_1) and MRVFL-D(dense- l_1)) from Chapter 5 and top three ranked classifiers (edRVFL, dSP-RVFL and edSP-RVFL) from Chapter 6 respectively. To conduct a fair experimental comparison, we evaluate all the above learning models in the same 13 publicly available real-world classification datasets used in Chapter 6. To help the readers recall the details of the datasets, we again present an overview of the datasets in Table 7.1.

For the random forest based methods, we set the ensemble size to 500 and $m_{try} = \sqrt{\#features}$. Because these datasets have large feature dimensions, we do not tune the m_{try} parameter. For the deep RVFL based methods, we use the same parameter settings of Chapter 6. The performance of each learning algorithm in each dataset is shown in Table 7.2. MRVFL(1), MRVFL(2) and MRVFL(3) in Table 7.2 represent MRVFL(dense- l_1), MRVFL-D(dense- l_1) and MRVFL-D(dense- l_2) respectively. Similar to the preceding chapters, the Friedman test and Nemenyi post-hoc test are applied to rank the algorithms and test the significance between

Table 7.1: Overview of the datasets.

Domain	Dataset	#Patterns	#Features	#Class
Biology	Carcinom	174	9182	11
	Lung	203	3312	5
Face	ORL	400	1024	40
	Yale	165	1024	15
Handwritten Digits	BA	1404	320	36
	Gisette	7000	5000	2
	MNIST	4000	748	10
	USPS	1000	256	10
Object	COIL20	1440	1024	20
	COIL100	7200	1024	100
Text	BASEHOCK	1993	1000	2
	RCV1	9625	1000	4
	TDT2	9394	1000	30

Figure 7.1: Statistical comparison of classifiers against each other based on Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.05$) are connected.

the classifiers. Likewise, we obtain, $\chi_F^2 = 5047$ and $F_F = 8.477$. With 11 classifiers and 11 datasets (2 datasets with NA are removed for fair comparison as in Chapter 4), F_F is distributed according to the F -distribution with $11 - 1 = 10$ and $(11 - 1)(11 - 1) = 100$ degrees of freedom. The critical value for $F_{(10,100)}$ for $\alpha = 0.05$ is 4.56, so we reject the null-hypothesis. Based on the Nemenyi test as shown in Fig. 7.1, the critical difference is $CD = q_\alpha \sqrt{(m(m+1))/(6M)} = 3.219 * \sqrt{11 * 12 / (6 * 11)} \simeq 4.56$.

From the results obtained, following conclusions can be made.

1. Among the random forest based methods, the oblique random forests proposed in Chapter 4 perform better. Specifically, obRaF(H) has the best performance followed by obRaF(L) and obRaF(M).

Table 7.2: Comparison in terms of accuracy (%)

Dataset	RaFL	obRaFL	obRaFL	obRaFL(L)	obRaFL(M)	obRaFL(H)	MIRVFL(1)	MIRVFL(2)	MIRVFL(3)	edRVFL	DSP-RVFL	edDSP-RVFL
Carcinom	89.61±10.53	89.61±10.53	90.23±8.58	87.94±9.93	91.93±7.68	98.27±1.56	98.27±1.56	98.27±1.56	98.27±1.56	98.86±2.41	98.27±2.79	98.27±2.79
Lung	90.14±6.11	90.14±6.11	93.12±5.28	92.6±4.75	96.05±5.15	97.05±4.19	97.05±4.19	97.05±4.19	97.05±4.19	97.05±4.19	97.05±4.19	97.05±4.19
ORL	98.25±2.65	98.25±2.65	96.7±3.13	96±2.93	98.5±2.11	99.25±1.65	99.25±1.69	99.25±1.69	99.5±1.58	99±1.75	99.5±1.58	99.5±1.58
Yale	82.39±6.7	82.39±6.7	79.3±8.58	78.71±6.86	82.98±6.3	86.62±7.43	87.21±6.67	86.58±6.92	88.58±6.92	88.58±6.92	87.17±7.38	87.17±7.38
BA	64.6±3.61	64.39±3.92	71.58±3.12	72.51±2.87	72.07±3.18	72.79±2.67	73.5±3.9	75.07±2.68	66.11±3.41	68.52±2.65	73.08±2.67	73.08±2.67
Gisette	NA	NA	97.1±0.34	97.23±0.35	97.2±0.6	98.19±0.52	98.24±0.41	98.4±0.45	98.21±0.55	98.21±0.55	98.21±0.41	98.21±0.41
MNIST	83.57±2.66	83.57±2.66	83.57±2.66	83.57±2.66	83.57±2.66	93.53±1.11	93.8±1.32	94.95±1.28	92.8±1.92	93.43±1.4	95.02±1.5	95.02±1.5
USPS	86.8±5.01	86.8±5.01	86.8±5.01	86.8±5.01	86.8±5.01	94.7±1.96	95.05±1.55	95.8±2.1	94.1±2.07	93.5±1.68	95±1.6	95±1.6
COIL20	98.68±0.83	98.26±0.94	99.93±0.22	99.93±0.22	99.93±0.22	99.93±0.22	99.93±0.22	100	99.86±0.29	98.96±0.67	99.72±0.36	99.72±0.36
COIL100	95.38±0.44	94.92±0.56	98.32±0.36	99.13±0.34	99.67±0.3	96.88±0.39	97±0.32	97.39±0.35	90.5±0.82	93.38±1.26	94.26±0.86	94.26±0.86
BASEHOCK	NA	NA	98.29±1.04	96.94±1.56	94.08±1.37	97.79±1.06	97.99±0.58	98.04±1.07	98.04±0.76	97.79±0.98	97.79±0.8	97.79±0.8
RCV1	92.4±0.68	91.95±0.8	93.73±0.64	93.72±0.79	93.5±0.76	93.43±0.52	93.46±0.58	93.49±0.57	93.86±0.69	93.9±0.61	94.6±0.5	94.6±0.5
TD2T	88.31±0.64	83.92±0.99	94.01±0.66	94.98±0.61	92.97±0.97	96.25±0.66	96.27±0.56	96.54±0.63	96.51±0.53	96.17±0.63	96.17±0.63	96.17±0.63
Mean Acc. ¹	88.19±3.62	87.65±3.71	90.97±3.04	90.77±2.99	91.48±2.79	94.2±1.84	94.38±1.81	94.69±1.79	93.35±2.01	93.52±2.01	94.29±1.95	94.29±1.95
Mean Acc. ²	88.19±3.62	87.65±3.71	89.76±3.47	89.63±3.36	90.72±3.12	93.52±2.03	93.7±2.05	94.05±1.98	92.47±2.26	92.71±2.25	93.62±2.18	93.62±2.18
Avg. Rank	9.27	9.72	7.27	7.55	6.55	4.73	3.73	2.95	4.94	5.32	3.95	3.95

RaFL and obRaFL are applicable only on multi-class datasets. Therefore for binary datasets, results are not available as indicated by NA. MIRVFL(1), MIRVFL(2) and MIRVFL(3) represent MIRVFL(dense-1), MIRVFL-D(dense-1) and MIRVFL-D(dense-2) respectively. This is done to fit the table within the dimensions of the page. Mean Acc.¹ is the average accuracy of each method based on the datasets on which it runs successfully. Mean Acc.² is the average accuracy based only on the datasets where all methods run successfully.

2. The deep RVFL based methods consistently perform better in all the datasets except two datasets where oblique random forests have the best performance. This demonstrates the attractiveness and potential of deep learning models in real-world classification datasets.
3. The top two ranked algorithms (MRVFL-D(dense- l_2) and MRVFL-D(dense- l_1)) are based on denoising autoencoders. This indicates that the denoising criterion propels the algorithm to learn stable and robust feature representations thus, improving the generalization of the algorithm.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis, we studied two popular machine learning algorithms: decision trees and neural network. Both decision trees and neural networks are extensively used in various applications. When traversing from the root to the leaf nodes, a decision tree can be perceived as a sequence of IF-THEN rules which is intuitive to humans. Meanwhile, a neural network consists of highly interconnected system of neurons designed to mimic the animal brain. Typically, ensembles of decision trees such as random forest are employed in practice to obtain better performance. Based on the nature of the split functions, there exists two types of decision trees: axis-parallel and oblique. We explored both types of decision trees and their ensembles. In the context of neural networks, we studied randomization based neural networks such as RVFL. In RVFL, the hidden layer parameters are randomly generated and kept fixed, while the output weights are learned during the training procedure. We discussed both shallow and deep RVFL neural networks. Below we summarize the main results of this thesis.

1. In Chapter 3, we developed a hybrid ensemble classifier based on the fusion of RVFL neural network and decision trees. The RVFL network was used to partition the training data into several subsets which would be handled by the subsequent decision trees. This is equivalent to creating multi-way splits at the root node of the decision trees. Such partitioning provides a rich insight into the data by grouping together the confusing or hard to classify samples and thus, provides an opportunity to employ fine-grained classification rule over the data. We developed ensembles using both axis-parallel and oblique decision trees which were termed as RaFL (RVFL with axis-parallel trees) and obRaFL (RVFL with oblique trees) respectively. Extensive experiments on several multi-classes datasets showed that such hybrid classifiers perform better than individual ensembles of trees at the expense of additional computational cost.

2. In Chapter 4, we presented several one-vs-all based oblique random forests. In such oblique random forests, the training data at each node is partitioned using one-vs-all technique and a linear classifier is employed at each partition to generate an oblique hyperplane. The linear (oblique) split that best minimized the impurity criterion was then selected as the splitting hyperplane. The one-vs-all based formulation requires evaluating K decision boundaries generated by a linear classifier where K is the number of classes at each node. An oblique random forest employing n linear classifiers thus, requires nK evaluations at each node. Such procedure is computationally expensive when the number of classes is very large. Also, such one-vs-all based partitioning results in unbalanced trees while limiting the searches to only K evaluations. Thus, to expand the search space of the oblique splits, we considered both one-vs-all and two-hyperclasses based partitioning. However, this increases the computational cost. To avoid evaluating a large number of partitions, we either employed an optimization algorithm or an exhaustive search depending on the number of classes at each node in conjunction with ideal gini score and cluster separability to selectively filter the partitions. We showed that these techniques could be used to rank the partitions. Then, we developed a random forest of heterogeneous oblique decision trees that employed several linear classifiers at each non-leaf node. The proposed oblique random forests were shown to be more accurate and diverse than other random forest methods using diversity-error diagrams. We benchmarked 190 classifiers on 121 UCI datasets. The results showed that the oblique random forests proposed herein are the top 3 ranked classifiers.
3. In Chapter 5, we developed several deep RVFL neural network variants by utilizing the framework of multi-layer ELM. Specifically, we introduced direct connections (feature reuse) from preceding layers to the fore layers of the network as in the original RVFL network. We also introduced denoising criterion in the autoencoders to achieve stable and robust higher level representations than the ordinary autoencoders. The deep RVFL networks were labelled as sdRVFL. Extensive experiments on several classification datasets showed that sdRVFLs achieve overall better and faster generalization than the other relevant state-of-the-art multi-layer ELMs. We also showed that sdRVFLs built with denoising autoencoders generalize better than those built with ordinary autoencoders.
4. In Chapter 6, we presented another variant of deep RVFL network termed as dRVFL. The parameters of the hidden layers of dRVFL were randomly generated within a suitable range and kept fixed while the output weights were computed using the closed form solution as in a standard RVFL network. The dRVFL network while extracting rich feature representations through several hidden layers also acts as a weighting network thereby, providing a weight to features from all the hidden layers including the original features obtained via direct links. We then proposed an ensemble deep network (edRVFL) that combined ensemble learning with deep learning. Unlike

traditional ensembling approaches that required training several models independently from scratch, the edRVFL was obtained by training a single dRVFL once. Both dRVFL and edRVFL frameworks are generic and can be used with any RVFL variant. To illustrate this, we integrated the deep learning networks with a recently proposed sparse-pretrained RVFL (SP-RVFL). Extensive experiments on benchmark datasets from diverse domains showed the efficacy of our proposed deep RVFL networks.

8.2 Future Work

In this section, we present the possible future research directions leading from the works of this thesis.

1. Gradient Boosting Decision Trees

Random Forest along with Gradient Boosted Trees are widely used to win several competitions. Popular gradient boosting frameworks such as XGBoost [151], LightGBM [152] typically use axis-aligned decision trees. In this thesis, we presented several oblique decision trees that generalize better than the axis-aligned decision trees. Thus, we would like to pursue in this direction to develop oblique decision trees based gradient boosting.

2. Representational (Deep) Learning using Decision Trees

The success of deep learning emanates from multi-layered representation. However, non-differentiable models such as decision trees are difficult to incorporate with representational learning ability. Because both deep learning and decision trees are powerful (in terms of their success) machine learning algorithms, it would be exciting to combine these two algorithms and reap the benefits from both these two worlds as in [108].

3. Deep RVFL ensemble

In this thesis, we presented an implicit ensembling technique for deep RVFL networks. In addition, techniques like Dropout, DropConnect, Stochastic Depth and Swapout could also be easily integrated with such deep RVFL networks to obtain ensembles of networks. A preliminary version of this idea was implemented with shallow RVFL networks in [153].

4. Application of deep RVFL networks in other learning tasks

Shallow RVFL network has been shown to perform better in other learning tasks beyond classification in areas such as regression [76] and forecasting [77, 78, 154]. It is left as our future work to assess the performance of deep RVFL networks in such tasks.

Author's Publications

Journal Papers

1. **Rakesh Katuwal**, P.N. Suganthan, and Le Zhang. An ensemble of decision trees with random vector functional link networks for multi-class classification. *Applied Soft Computing*, 70 : 1146–1153. doi:10.1016/j.asoc.2017.09.020
2. **Rakesh Katuwal**, P.N. Suganthan, and Le Zhang. Heterogeneous oblique random forest. *Pattern Recognition*, 99, 107078. doi:10.1016/j.patcog.2019.107078
3. **Rakesh Katuwal** and P.N. Suganthan. Stacked autoencoder based deep random vector functional link neural network for classification. *Applied Soft Computing*, 105854. doi:10.1016/j.asoc.2019.105854
4. P.N. Suganthan and **Rakesh Katuwal**. On the Origins of Randomization based Feedforward Neural Networks *Applied Soft Computing*. (submitted)
5. **Rakesh Katuwal** and P.N. Suganthan. Random vector functional link neural network based ensemble deep learning. doi:arxiv.org/abs/1907.00350

Conference Papers

1. **Rakesh Katuwal** and P. N. Suganthan. Enhancing multi-class classification of random forest using random vector functional neural network and oblique decision surfaces. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1 – 8, July 2018.
2. **Rakesh Katuwal** and P.N. Suganthan. An ensemble of kernel ridge regression for multi-class classification. *Procedia Computer Science*, 108:375 – 383, 2017. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.
3. **Rakesh Katuwal** and P. N. Suganthan. Dropout and dropconnect based ensemble of random vector functional link neural network. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1772 – 1778, Nov 2018.

4. W.X. Cheng, **Rakesh Katuwal**, P.N. Suganthan, and X. Qiu. A heterogeneous ensemble of trees. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1-6, Nov 2017.
5. W.X. Cheng, P.N. Suganthan, X. Qiu and **Rakesh Katuwal**. Classification of stock market trends with confidence-based selective predictions. In *International Conference on Fuzzy And Neural Computing Conference (FANCCO 2019)*. Springer.

Bibliography

- [1] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems 30*, pages 971–980. Curran Associates, Inc., 2017.
- [2] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15:3133–3181, 2014.
- [3] James Surowiecki. *The wisdom of crowds*. Anchor, 2005.
- [4] Louis B Rosenberg. Human swarms, a real-time paradigm for collective intelligence. *Collective Intelligence*, 2015.
- [5] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.
- [6] Miguel A. Carreira-Perpinan and Pooya Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. In *Advances in Neural Information Processing Systems 31*, pages 1211–1221. Curran Associates, Inc., 2018.
- [7] Ira Shavitt and Eran Segal. Regularization learning networks: Deep learning for tabular datasets. In *Advances in Neural Information Processing Systems 31*, pages 1379–1389. Curran Associates, Inc., 2018.
- [8] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.
- [9] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *The annals of statistics*, 28(2):337–407, 2000.
- [10] L. Zhang and P. N. Suganthan. Benchmarking ensemble classifiers with novel co-trained kernel ridge regression and random vector functional link ensembles [research frontier]. *IEEE Computational Intelligence Magazine*, 12(4):61–72, Nov 2017.
- [11] Y. H. Pao and Y. Takefuji. Functional-link net computing: theory, system architecture, and functionalities. *IEEE Computer*, 25(5):76–79, May 1992.
- [12] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: Theory and applications. *Neurocomputing*, 70(1):489 – 501, 2006. Neural Networks.

- [13] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [14] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85 – 117, 2015.
- [15] Matthew Olson, Abraham Wyner, and Richard Berk. Modern neural networks generalize on small data sets. In *Advances in Neural Information Processing Systems 31*, pages 3619–3628. Curran Associates, Inc., 2018.
- [16] M. Lichman. UCI machine learning repository, 2013.
- [17] Fernando Berzal, Juan-Carlos Cubero, Nicolás Marín, and Daniel Sánchez. Building multi-way decision trees with numerical attributes. *Information Sciences*, 165(1):73 – 90, 2004.
- [18] David Biggs, Barry De Ville, and Ed Suen. A method of choosing multiway partitions for classification and decision trees. *Journal of Applied Statistics*, 18(1):49–62, 1991.
- [19] L. Zhang and P. N. Suganthan. Oblique decision tree ensemble via multisurface proximal support vector machine. *IEEE Transactions on Cybernetics*, 45(10):2165–2176, Oct 2015.
- [20] Sreerama K Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *Journal of artificial intelligence research*, 2:1–32, 1994.
- [21] Ponnuthurai Nagaratnam Suganthan. On non-iterative learning algorithms with closed-form solution. *Applied Soft Computing*, 70:1078 – 1082, 2018.
- [22] Ping Guo, C.L.P. Chen, and Y.G. Sun. An exact supervised learning for a three-layer supervised neural network. In *Proceedings of the International Conference on neural Information Processing (ICONIP’95)*, pages 1041–1044, 1995.
- [23] Ping Guo. A vest of the pseudoinverse learning algorithm. In <https://arxiv.org/pdf/1805.07828>, pages 1–5, 2018.
- [24] Hugues Berry and Mathias Quoy. Structure and dynamics of random recurrent neural networks. *Adaptive Behavior*, 14(2):129–137, 2006.
- [25] R. Katuwal and P. N. Suganthan. Enhancing multi-class classification of random forest using random vector functional neural network and oblique decision surfaces. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2018.
- [26] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, pages 550–558, 2016.

- [27] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. Snapshot ensembles: Train 1, get m for free. <https://arxiv.org/abs/1704.00109>, 2017.
- [28] Y. Ren, L. Zhang, and P. N. Suganthan. Ensemble classification and regression-recent developments, applications and future directions [review article]. *IEEE Computational Intelligence Magazine*, 11(1):41–53, Feb 2016.
- [29] Robert E Schapire, Yoav Freund, Peter Bartlett, Wee Sun Lee, et al. Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5):1651–1686, 1998.
- [30] Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1):1–39, 2010.
- [31] Thomas G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, pages 1–15, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
- [32] Avrim Blum and Ronald L Rivest. Training a 3-node neural network is np-complete. In *Proceedings of the 1st International Conference on Neural Information Processing Systems*, pages 494–501. MIT Press, 1988.
- [33] Laurent Hyafil and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17, 1976.
- [34] Ron Kohavi, David H Wolpert, et al. Bias plus variance decomposition for zero-one loss functions. In *ICML*, volume 96, pages 275–83, 1996.
- [35] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.
- [36] Eun Bae Kong and Thomas G. Dietterich. Error-correcting output coding corrects bias and variance. In Armand Frieditis and Stuart Russell, editors, *Machine Learning Proceedings 1995*, pages 313 – 321. Morgan Kaufmann, San Francisco (CA), 1995.
- [37] Jerome H. Friedman. On bias, variance, 0/1—loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, Mar 1997.
- [38] Gareth M James. Variance and bias for general loss functions. *Machine learning*, 51(2):115–135, 2003.
- [39] Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends® in Computer Graphics and Vision*, 7(2–3):81–227, 2012.
- [40] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

- [41] Leo Breiman. Bias, variance, and arcing classifiers. *Tech. Rep. 460, Statistics Department, University of California, Berkeley, CA, USA*, 1996.
- [42] Carla E. Brodley and Paul E. Utgoff. Multivariate decision trees. *Machine Learning*, 19(1):45–77, Apr 1995.
- [43] M. Ristin, M. Guillaumin, J. Gall, and L. Van Gool. Incremental learning of random forests for large-scale image classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(3):490–503, March 2016.
- [44] P. Kotschieder, M. Fiterau, A. Criminisi, and S. R. Buló. Deep neural decision forests. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1467–1475, Dec 2015.
- [45] J. Gall and V. Lempitsky. Class-specific hough forests for object detection. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1022–1029, June 2009.
- [46] S. Schuster, P. Wohlhart, C. Leistner, A. Saffari, P. M. Roth, and H. Bischof. Alternating decision forests. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 508–515, June 2013.
- [47] G. Garcia-Hernando and T. K. Kim. Transition forests: Learning discriminative temporal transitions for action recognition and detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 407–415, July 2017.
- [48] Chunyu Hu, Yiqiang Chen, Lisha Hu, and Xiaohui Peng. A novel random forests based class incremental learning method for activity recognition. *Pattern Recognition*, 78:277 – 290, 2018.
- [49] S. Buló and P. Kotschieder. Neural decision forests for semantic image labelling. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 81–88, June 2014.
- [50] Björn Fröhlich, Erik Rodner, and Joachim Denzler. *Semantic Segmentation with Millions of Features: Integrating Multiple Cues in a Combined Random Forest Approach*, pages 218–231. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- [51] L. Zhang, J. Varadarajan, P. N. Suganthan, N. Ahuja, and P. Moulin. Robust visual tracking using oblique random forests. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5825–5834, July 2017.
- [52] Yuanyuan Liu, Xiaohui Yuan, Xi Gong, Zhong Xie, Fang Fang, and Zhongwen Luo. Conditional convolution neural network enhanced random forest for facial expression recognition. *Pattern Recognition*, 84:251 – 261, 2018.
- [53] A. Criminisi, D. Robertson, E. Konukoglu, J. Shotton, S. Pathak, S. White, and K. Siddiqui. Regression forests for efficient anatomy detection and local-

- ization in computed tomography scans. *Medical Image Analysis*, 17(8):1293 – 1303, 2013.
- [54] Xueheng Qiu, Le Zhang, Ponnuthurai Nagaratnam Suganthan, and Gehan A.J. Amaratunga. Oblique random forest ensemble via least square estimation for time series forecasting. *Information Sciences*, 420:249 – 262, 2017.
- [55] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, Aug 1998.
- [56] Bjoern H Menze, B Michael Kelm, Daniel N Splitthoff, Ullrich Koethe, and Fred A Hamprecht. On oblique random forests. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 453–469. Springer, 2011.
- [57] W. Pedrycz and Z. A. Sosnowski. Genetically optimized fuzzy decision trees. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35(3):633–641, June 2005.
- [58] Sung-Hyuk Cha and Charles Tappert. A genetic algorithm for constructing compact binary decision trees. *Journal of Pattern Recognition Research*, 4(1):1–13, 2009.
- [59] Li Zhang, Wei-Da Zhou, Tian-Tian Su, and Li-Cheng Jiao. Decision tree support vector machine. *International Journal on Artificial Intelligence Tools*, 16(01):1–15, 2007.
- [60] Tracy D Lemmond, Barry Y Chen, Andrew O Hatch, and William G Hanley. An extended study of the discriminant random forest. In *Data Mining*, pages 123–146. Springer, 2010.
- [61] Alfred Kar Yin Truong. *Fast growing and interpretable oblique trees via logistic regression models*. PhD thesis, University of Oxford, 2009.
- [62] Ronald A Fisher. The statistical utilization of multiple measurements. *Annals of eugenics*, 8(4):376–386, 1938.
- [63] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.
- [64] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.
- [65] O. L. Mangasarian and E. W. Wild. Multisurface proximal support vector machine classification via generalized eigenvalues. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(1):69–74, Jan 2006.
- [66] Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

- [67] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.
- [68] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1994.
- [69] W. F. Schmidt, M. A. Kraaijveld, and R. P. W. Duin. Feedforward neural networks with random weights. In *Proceedings., 11th IAPR International Conference on Pattern Recognition. Vol.II. Conference B: Pattern Recognition Methodology and Systems*, pages 1–4, Aug 1992.
- [70] Hubert A.B. Te Braake and Gerrit Van Straten. Random activation weight neural net (rawn) for fast non-iterative training. *Engineering Applications of Artificial Intelligence*, 8(1):71 – 80, 1995.
- [71] Le Zhang and Ponnuthurai Nagaratnam Suganthan. A survey of randomized algorithms for training neural networks. *Information Sciences*, 364:146–155, Oct, 2016.
- [72] Bernard Widrow, Aaron Greenblatt, Youngsik Kim, and Dookun Park. The no-prop algorithm: A new learning algorithm for multilayer neural networks. *Neural Networks*, 37:182 – 188, 2013. Twenty-fifth Anniversay Commemorative Issue.
- [73] Halbert White. Chapter 9 approximate nonlinear forecasting methods. volume 1 of *Handbook of Economic Forecasting*, pages 459 – 512. Elsevier, 2006.
- [74] L. Zhang and P. N. Suganthan. Visual tracking with convolutional random vector functional link network. *IEEE Transactions on Cybernetics*, 47(10):3243–3253, Oct 2017.
- [75] Rakesh Katuwal, P.N. Suganthan, and Le Zhang. An ensemble of decision trees with random vector functional link networks for multi-class classification. *Applied Soft Computing*, 70:1146 – 1153, 2018.
- [76] Najdan Vuković, Milica Petrović, and Zoran Miljković. A comprehensive experimental evaluation of orthogonal polynomial expanded random vector functional link neural networks for regression. *Applied Soft Computing*, 70:1083 – 1096, 2018.
- [77] Ling Tang, Yao Wu, and Lean Yu. A non-iterative decomposition-ensemble learning paradigm using rvfl network for crude oil price forecasting. *Applied Soft Computing*, 70:1097 – 1108, 2018.
- [78] Yajnaseni Dash, Saroj Kanta Mishra, Sandeep Sahany, and Bijaya Ketan Panigrahi. Indian summer monsoon rainfall prediction: A comparison of iterative and non-iterative approaches. *Applied Soft Computing*, 70:1122 – 1134, 2018.

- [79] Yoh-Han Pao, Gwang-Hoon Park, and Dejan J. Sobajic. Learning and generalization characteristics of the random vector functional-link net. *Neuro-computing*, 6(2):163 – 180, 1994. Backpropagation, Part IV.
- [80] B. K. Verma and J. J. Mulawka. A modified backpropagation algorithm. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 2, pages 840–844 vol.2, June 1994.
- [81] B. Igel and. Stochastic choice of basis functions in adaptive function approximation and the functional-link net. *IEEE Transactions on Neural Networks*, 6(6):1320–1329, Nov 1995.
- [82] Le Zhang and Ponnuthurai Nagaratnam Suganthan. A comprehensive evaluation of random vector functional link networks. *Information Sciences*, 367-368:1094 – 1105, 2016.
- [83] Ye Ren, Ponnuthurai Nagaratnam Suganthan, N. Srikanth, and Gehan Amaratunga. Random vector functional link network for short-term electricity load demand forecasting. *Information Sciences*, 367-368:1078 – 1093, 2016.
- [84] Michael J Kearns and UV Vazirani. An introduction to computational learning theory the mit press. *Cambridge, Massachusetts*, 1994.
- [85] G. Huang, H. Zhou, X. Ding, and R. Zhang. Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):513–529, April 2012.
- [86] Jyotirmayee Naik, Prachitara Satapathy, and P.K. Dash. Short-term wind speed and wind power prediction using hybrid empirical mode decomposition and kernel ridge regression. *Applied Soft Computing*, 70:1167 – 1188, 2018.
- [87] B Schölkopf, Ralf Herbrich, and A Smola. A generalized representer theorem. computational learning theory. *Lecture Notes in Computer Science. 2001a*, 2111:416–426, 2001.
- [88] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.
- [89] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.
- [90] Yongshan Zhang, Jia Wu, Zhihua Cai, Bo Du, and Philip S. Yu. An unsupervised parameter learning model for RVFL neural network. *Neural Networks*, 112:85 – 97, 2019.
- [91] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

- [92] Liyanaarachchi Lekamalage Chamara Kasun, Hongming Zhou, Guang-Bin Huang, and Chi Man Vong. Representational learning with extreme learning machine for big data. *IEEE intelligent systems*, 28(6):31–34, 2013.
- [93] H. Li, H. Zhao, and H. Li. Neural-response-based extreme learning machine for image classification. *IEEE Transactions on Neural Networks and Learning Systems*, 30(2):539–552, Feb 2019.
- [94] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.
- [95] C. M. Wong, C. M. Vong, P. K. Wong, and J. Cao. Kernel-based multilayer extreme learning machines for representation learning. *IEEE Transactions on Neural Networks and Learning Systems*, 29(3):757–762, March 2018.
- [96] J. Tang, C. Deng, and G. Huang. Extreme learning machine for multilayer perceptron. *IEEE Transactions on Neural Networks and Learning Systems*, 27(4):809–821, April 2016.
- [97] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004.
- [98] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
- [99] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.
- [100] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *Computer Vision – ECCV 2016*, pages 646–661, Cham, 2016. Springer International Publishing.
- [101] Saurabh Singh, Derek Hoiem, and David Forsyth. Swapout: Learning an ensemble of deep architectures. In *Advances in Neural Information Processing Systems 29*, pages 28–36. Curran Associates, Inc., 2016.
- [102] Rakesh Katuwal, P.N. Suganthan, and Le Zhang. An ensemble of decision trees with random vector functional link networks for multi-class classification. *Applied Soft Computing*, 70:1146 – 1153, 2018.
- [103] Rakesh Katuwal and P.N. Suganthan. An ensemble of kernel ridge regression for multi-class classification. *Procedia Computer Science*, 108:375 – 383, 2017. International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.

- [104] Rakesh Katuwal and P. N. Suganthan. Enhancing multi-class classification of random forest using random vector functional neural network and oblique decision surfaces. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, July 2018.
- [105] I. K. Sethi. Entropy nets: from decision trees to neural networks. *Proceedings of the IEEE*, 78(10):1605–1613, Oct 1990.
- [106] David L Richmond, Dagmar Kainmueller, Michael Y Yang, Eugene W Myers, and Carsten Rother. Relating cascaded random forests to deep convolutional neural networks for semantic segmentation. *arXiv preprint arXiv:1507.07583*, 2015.
- [107] José M Jerez-Aragónés, José A Gómez-Ruiz, Gonzalo Ramos-Jiménez, José Muñoz-Pérez, and Emilio Alba-Conejo. A combined neural network and decision trees model for prognosis of breast cancer relapse. *Artificial intelligence in medicine*, 27(1):45–63, 2003.
- [108] Ji Feng, Yang Yu, and Zhi-Hua Zhou. Multi-layered gradient boosting decision trees. In *Advances in Neural Information Processing Systems 31*, pages 3551–3561. Curran Associates, Inc., 2018.
- [109] Hadi Sadoghi Yazdi, Nima Salehi Moghaddami, and Hanieh Poostchi Mohammadabadi. Correlation based splitting criterion in multi branch decision tree. *Central European Journal of Computer Science*, 1, 2011.
- [110] Venkatesh N Murthy, Vivek Singh, Terrence Chen, R Manmatha, and Dorin Comaniciu. Deep decision network for multi-class image classification. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, pages 2240–2248. IEEE, 2016.
- [111] Lawrence Mitchell, Terence M. Sloan, Muriel Mewissen, Peter Ghazal, Thorsten Forster, Michal Piotrowski, and Arthur S. Trew. A parallel random forest classifier for r. In *Proceedings of the Second International Workshop on Emerging Computational Methods for the Life Sciences*, ECMLS '11, pages 1–6, New York, NY, USA, 2011. ACM.
- [112] H. Grahm, N. Lavesson, M. H. Lapajne, and D. Slat. Cudarf: A cuda-based implementation of random forests. In *2011 9th IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, pages 95–101, Dec 2011.
- [113] Toby Sharp. Implementing decision trees and forests on a gpu. In *Computer Vision – ECCV 2008*, pages 595–608, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [114] T. M. Khoshgoftaar, M. Golawala, and J. V. Hulse. An empirical study of learning from imbalanced data using random forest. In *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*, volume 2, pages 310–317, Oct 2007.

- [115] Mohammed Khalilia, Sounak Chakraborty, and Mihail Popescu. Predicting disease risks from highly imbalanced data using random forest. *BMC medical informatics and decision making*, 11(1):51, 2011.
- [116] Iain Brown and Christophe Mues. An experimental comparison of classification algorithms for imbalanced credit scoring data sets. *Expert Systems with Applications*, 39(3):3446 – 3453, 2012.
- [117] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan):1–30, 2006.
- [118] Peter Nemenyi. Distribution-free multiple comparisons. In *Biometrics*, volume 18, page 263. International Biometric SOC 1441 I ST, NW, SUITE 700 Washington, DC, 20005-2210, 1962.
- [119] Rakesh Katuwal, P.N. Suganthan, and Le Zhang. Heterogeneous oblique random forest. *Pattern Recognition*, 99:107078, 2020.
- [120] N. Manwani and P. S. Sastry. Geometric decision tree. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(1):181–192, Feb 2012.
- [121] Giuliano Armano and Emanuele Tamponi. Building forests of local trees. *Pattern Recognition*, 76:380 – 390, 2018.
- [122] David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.
- [123] F. Takahashi and S. Abe. Decision-tree-based multiclass support vector machines. In *Neural Information Processing, 2002. ICONIP '02. Proceedings of the 9th International Conference on*, volume 3, pages 1418–1422 vol.3, Nov 2002.
- [124] Mohammad Norouzi, Maxwell D Collins, David J Fleet, and Pushmeet Kohli. Co2 forest: Improved random forest by continuous optimization of oblique splits. *arXiv:1506.06155*, 2015.
- [125] Mohammad Norouzi, Maxwell Collins, Matthew A Johnson, David J Fleet, and Pushmeet Kohli. Efficient non-greedy optimization of decision trees. In *Advances in Neural Information Processing Systems*, pages 1729–1737, 2015.
- [126] Ian Millington and John Funge. *Artificial intelligence for games*. CRC Press, 2009.
- [127] J. C.Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, 1974.
- [128] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.
- [129] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, Apr 2006.

- [130] Dragos D Margineantu and Thomas G Dietterich. Pruning adaptive boosting. In *ICML*, volume 97, pages 211–218, 1997.
- [131] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso. Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1619–1630, Oct 2006.
- [132] Le Zhang and Ponnuthurai Nagaratnam Suganthan. Random forests with ensemble of feature spaces. *Pattern Recognition*, 47(10):3429 – 3437, 2014.
- [133] Cheng-Tao Chu, Sang K Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Kunle Olukotun, and Andrew Y Ng. Map-reduce for machine learning on multicore. In *Advances in neural information processing systems*, pages 281–288, 2007.
- [134] Lizhong Ding and Shizhong Liao. Approximate model selection for large scale lssvm. In *Asian Conference on Machine Learning*, pages 165–180, 2011.
- [135] Michael Wainberg, Babak Alipanahi, and Brendan J. Frey. Are random forests truly the best classifiers? *Journal of Machine Learning Research*, 17(110):1–5, 2016.
- [136] Alessio Benavoli, Giorgio Corani, and Francesca Mangili. Should we really use post-hoc tests based on mean-ranks? *The Journal of Machine Learning Research*, 17(1):152–161, 2016.
- [137] Rakesh Katuwal and P.N. Suganthan. Stacked autoencoder based deep random vector functional link neural network for classification. *Applied Soft Computing*, page 105854, 2019.
- [138] Kai Sun, Jianshe Zhang, Chunxia Zhang, and Junying Hu. Generalized extreme learning machine autoencoder and a new deep neural network. *Neurocomputing*, 230:374 – 381, 2017.
- [139] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.
- [140] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.
- [141] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.
- [142] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

- [143] Yuchen Zhang, John Duchi, and Martin Wainwright. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates. *The Journal of Machine Learning Research*, 16(1):3299–3340, 2015.
- [144] Moshe Lichman. Uci machine learning repository, 2013.
- [145] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases*, pages 498–515. Springer International Publishing, 2015.
- [146] Robert J Lyon, BW Stappers, S Cooper, JM Brooke, and JD Knowles. Fifty years of pulsar candidate selection: from simple filters to a new principled real-time classification approach. *Monthly Notices of the Royal Astronomical Society*, 459(1):1104–1123, 2016.
- [147] Rakesh Katuwal, PN Suganthan, and M Tanveer. Random vector functional link neural network based ensemble deep learning. *arXiv preprint arXiv:1907.00350*, 2019.
- [148] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. Deep reservoir computing: A critical experimental analysis. *Neurocomputing*, 268:87 – 99, 2017.
- [149] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006. PMID: 16764513.
- [150] Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *Artificial intelligence and statistics*, pages 448–455, 2009.
- [151] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.
- [152] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc., 2017.
- [153] Rakesh Katuwal and P. N. Suganthan. Dropout and dropconnect based ensemble of random vector functional link neural network. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1772–1778, Nov 2018.
- [154] Xueheng Qiu, Ponnuthurai Nagaratnam Suganthan, and Gehan A.J. Amaratunga. Ensemble incremental learning random vector functional link network for short-term electric load forecasting. *Knowledge-Based Systems*, 145:182 – 196, 2018.

Appendix

Details of the 121 UCI datasets

Table A1: Details of 121 UCI Datasets Used in this thesis

Dataset	# Patterns	# Features	# Classes
abalone*	4177	8	3
ac-inflam	120	6	2
acute-nephritis	120	6	2
adult	48842	14	2
annealing*	798	38	6
arrhythmia*	452	262	13
audiology-std*	226	59	18
balance-scale*	625	4	3
balloons	16	4	2
bank	45211	17	2
blood	748	4	2
breast-cancer	286	9	2
bc-wisc	699	9	2
bc-wisc-diag	569	30	2
bc-wisc-prog	198	33	2
breast-tissue*	106	9	6
car*	1728	6	4
ctg-10classes*	2126	21	10
ctg-3classes	2126	21	3
chess-krvk*	28056	6	18
chess-krvkp	3196	36	2
congress-voting	435	16	2
conn-bench-solar	208	60	2
conn-bench-vowel*	528	11	11
connect-4	67557	42	2
contrac*	1473	9	3
credit-approval	690	15	2
cylinder-bands	512	35	2
dermatology*	366	34	6
echocardiogram	131	10	2
ecoli*	336	7	8
energy-y1*	768	8	3
energy-y2*	768	8	3
fertility	100	9	2
flags*	194	28	8
glass*	214	9	6
haberman-survival	306	3	2
hayes-roth*	132	3	3
heart-cleveland*	303	13	5
heart-hungarian	294	12	2
heart-switzerland	123	12	2
heart-va*	200	12	5
hepatitis	155	19	2
hill-valley	606	100	2
horse-colic	300	25	2
ilpd-indian-liver	583	9	2
image-segmentation*	210	19	7
ionosphere	351	33	2
iris*	150	4	3
<i>(continued on next page)</i>			

Dataset	# Patterns	# Features	# Classes
led-display*	1000	7	10
lenses*	24	4	3
letter*	20000	16	26
libras*	360	90	15
low-res-spect*	531	100	9
lung cancer*	32	56	3
lymphography*	148	18	4
magic	19020	10	2
mammographic	961	5	2
miniboone	130064	50	2
molec-biol-promoter	106	57	2
molec-biol-splice*	3190	60	3
monks-1	124	6	2
monks-2	169	6	2
monks-3	3190	6	2
mushroom	8124	21	2
musk-1	476	166	2
musk-2	6598	166	2
nursery*	12960	8	5
oocMer12F*	1022	25	3
oocMer14D*	1022	25	3
oocIris2F	912	25	2
oocIris5B*	912	32	3
optical*	3823	62	10
ozone	2536	72	2
page-blocks*	5473	10	5
parkinsons	195	22	2
pendigits*	7494	16	10
pima	768	8	2
pb-MATERIAL*	106	4	3
pb-REL-L*	103	4	3
pb-SPAN*	92	4	3
pb-T-OR-D	102	4	2
pb-TYPE*	105	4	6
planning	182	12	2
plant-margin*	1600	64	100
plant-shape*	1600	64	100
plant-texture*	1600	64	100
post-operative*	90	8	3
primary-tumor*	330	17	15
ringnorm	7400	20	2
seeds*	210	7	3
semeion*	1593	256	10
soybean*	307	35	18
spambase	4601	57	2
spect	80	22	2
spectf	80	44	2
st-aus-credit	690	14	2
st-german-credit	1000	24	2
st-heart	270	13	2
st-image*	2310	18	7
st-landstat*	4435	36	6
st-shuttle*	43500	9	7
st-vehicle*	846	18	4
steel plates*	1941	27	7
synthetic-control*	600	60	6
teaching*	151	5	3
thyroid*	3772	21	3
tic-tac-toe	958	9	2
titanic	2201	3	2
trains	10	28	2
twonorm	7400	20	2
vc-2classes	310	6	2
vc-3classes*	310	6	3
wall-following	5456	24	4
waveform*	5000	21	3
waveform-noise*	5000	40	3
wine*	179	13	3
w-qua-red*	1599	11	6
w-qua-white*	4898	11	7
yeast*	1484	8	10
zoo*	101	16	7

* denotes the multi-class datasets used in Chapter 3.

Details of the other top 13 ranked classifiers out of top 20

In this section, we provide a brief overview of the classifiers listed in Table 4.8 in Chapter 4.

- MPRaF: MPSVM based random forest with axis-parallel regularization with $\text{ntree}=500$ and tuning mtry in the range $1 : \#inputs$.
- CoKRR-max: Max result of the CoKRR using Gaussian kernels, tuning C and the kernel spread with the values $2^{-14}, \dots, 2^5$ and $2^{-16}, \dots, 2^8$ respectively.
- MPRRoF: MPSVM based random rotation forest with axis-parallel regularization.
- parRF_t: uses a parallel implementation of random forest using the *randomForest* package with $\text{mtry}=2:2:8$.
- RaF_t: creates a random forest using the *caret* interface to the function *randomForest* in the *randomForest* package, with number of trees = 500 and tuning the parameter mtry with values $2:3:29$.
- svm_C: is the support vector machine, implemented in C using LibSVM with Gaussian kernel. The regularization parameter C and kernel spread γ are tuned in the ranges $2^{-5}, \dots, 2^{14}$ and $2^{-16}, \dots, 2^8$ respectively. LibSVM uses the one-vs-one approach for multi-class datasets.
- svmPoly_t: uses the *kernlab* package with linear, quadratic and cubic kernels using scale $s = \{0.001, 0.01, 0.1\}$, offset $o = 1$, degree $d = \{1, 2, 3\}$ and $C = \{0.25, 0.5, 1\}$.
- KRR: uses Gaussian kernel, tuning the regularization parameter and the kernel spread with values $2^{-5}, \dots, 2^{14}$ and $2^{-16}, \dots, 2^8$ respectively.
- rForest_R: creates a random forest ensemble, using the R function *randomForest* in the *randomForest* package, with parameters $\text{ntree} = 500$ and $\text{mtry} = \sqrt{\#inputs}$.
- svmRadialCost_t: uses (*kernlab* package) where only the cost C is tuned and kernel spread is calculated automatically.
- svmRadial_t: is the SVM with Gaussian kernel (in the *kernlab* package) tuning C and kernel spread with values $2^{-2}, \dots, 2^2$ and $10^{-2}, \dots, 10^2$ respectively.
- RVFL ensemble: an ensemble of 500 RVFL base models.
- C5.0_t: creates a Boosting ensemble of C5.0 decision trees and rule models, with and without winnow (feature selection), tuning the number of boosting trails in $\{1, 10, 20\}$.