

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

# Ensembling Neural Networks for Regression

João Miguel Mendes Ribeiro Agulha

DISSERTATION

**U.** PORTO

**FEUP** FACULDADE DE ENGENHARIA  
UNIVERSIDADE DO PORTO

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: João Mendes Moreira, PhD

Co-Supervisor: Tiago Mendes Neves, MSc

July 24, 2021



# **Ensembling Neural Networks for Regression**

**João Miguel Mendes Ribeiro Agulha**

Mestrado Integrado em Engenharia Informática e Computação

July 24, 2021



# Abstract

Machine Learning aims to find a model that searches through the hypothesis space to find a hopefully optimal hypothesis that best predicts a wanted outcome. Its generalization errors occur due to being affected, primarily by noise, bias, variance, and covariance.

Models may be formed by a variety of algorithms, namely, Neural Networks. Despite their accomplishments, these are generally referred to as a method with low bias and high variance, hence, unstable. Ensemble Learning, a thriving research area in terms of predictive performance, strives to tackle the variance issue by combining multiple models. In turn, it obtains more accurate, stable, and robust predictions. However, the ensemble's success is dependant on the data's nature, estimators' characteristics, estimators' generation strategy, and the predictions' integration mechanism for which there is no exact way to decide the most appropriate.

Some ensemble techniques are estimator agnostic, while others are not. The field of Decision Tree Ensembles has had many developments lately. Moreover, with recent years' Neural Network and Deep Learning success, Ensemble Learning also received new contributions. However, there is an evident lack of theorized proposals and systematic researches specifically on combining the most complementary existing strategies (paired strengths that most reduce each ensemble error component in conjunction). The few that exist are, in the majority, practical applications to concrete scenarios, tweaks to existing Decision-Tree specific frameworks, or very narrow and limited approaches. Given that Decision Tree Ensembles have demonstrably been proven to have excellent results in prediction accuracy improvement (very high performance), there is an expectation of adding value in the field of Neural Network Ensembles.

Stemming from the existing dissected literature, this thesis focuses on exploring and evolving the Neural Network Ensemble's state-of-the-art in the field of Supervised Learning, namely for Regression problems. The bottom line is to verify if combining the existing approaches leads to any performance improvement. Moreover, investigate which configurations and characteristics are the most appropriate.

As a result, we propose a hybrid Neural Network Ensemble combining framework that performs a complete error decomposition in bias, variance, and covariance on each algorithm to find the most promising ways to combine the most complementary existing strategies, aiming to take what is best from each strategy.

Creating multiple new and innovative hybrid ensemble algorithms involved a theoretical step of conceptualization and modularization. More concretely, and excluding varying training data ensemble techniques, more than 60 different algorithm combinations were developed.

Extensive experimental results with multiple datasets confirmed the theoretical assumptions that combining multiple already established Neural Network Ensemble strategies yields a clear and meaningful performance increase compared to the constituent base models and the respective original architectures they are based on. From the multitude of proposed new Neural Network Ensemble hybrid strategies, the best-performing ones decreased the global error, on average, from 12% to 17% versus their original architectures.

Particularly, combining Snapshot, Negative Correlation Learning, and Dropout offers the most notable results, but also, adding those three algorithms to other Ensemble Learning strategies improves their performance distinctly. However, if one searches for specific error component reduction on specific ensemble architectures, other combinations may be advisable to avoid some technique's pitfalls.

Therefore, and given that the experiments were considered statically valid and trustworthy, it is proven that combining different ensemble approaches is a plausible and consistent way of improving predictive capability with plenty of evolving potential.

**Keywords:** Machine Learning, Supervised Learning, Neural Networks, Ensemble Learning

# Resumo

O intuito de *Machine Learning* é encontrar um modelo que pesquisa através do espaço de hipóteses de forma a encontrar uma hipótese adequada que melhor prevê um resultado desejado. Os seus erros de generalização ocorrem devido a ser afetado, principalmente, por ruído, *bias*, variância e covariância.

Os modelos podem ser formados por uma variedade de algoritmos, a saber, Redes Neurais. Apesar das suas conquistas, estas são geralmente referidas como um método com baixo *bias* e alta variância, portanto, instáveis. *Ensemble Learning*, uma área de investigação próspera em termos de desempenho preditivo, tem como objetivo resolver o problema da variância combinando múltiplos modelos. Por sua vez, obtém-se previsões mais precisas, estáveis e robustas. No entanto, o sucesso do *Ensemble* depende da natureza dos dados, das características dos modelos de base, da estratégia de geração dos modelos de base e do mecanismo de integração de previsões para os quais não há uma maneira exata de decidir qual o mais apropriado.

Algumas técnicas de *Ensemble Learning* são agnósticas relativamente aos modelos de base, enquanto outras não. A área de *Ensemble Learning* para Árvores de Decisão teve muitos desenvolvimentos recentemente. Além disso, com o sucesso das Redes Neurais e *Deep Learning* nos últimos anos, *Ensemble Learning* também recebeu novas contribuições. No entanto, há uma evidente falta de propostas teorizadas e pesquisas sistemáticas especificamente sobre como combinar as estratégias existentes mais complementares (qualidades emparelhadas que oferecem uma maior redução de cada componente erro em conjunto). As poucas que existem são, na sua maioria, aplicações práticas a cenários em concreto, pequenos ajustes em estruturas específicas para Árvores de Decisão ou abordagens muito estreitas e limitadas. Dado que *Decision Tree Ensembles* apresentam demonstrativamente excelentes resultados na melhoria da exatidão das previsões (desempenho muito elevado), há a expectativa de poder adicionar valor na área de *Neural Network Ensembles*.

Partindo da literatura existente dissecada, esta tese centra-se em explorar e evoluir o estado da arte de *Neural Network Ensembles* no campo de Aprendizagem Supervisionada, nomeadamente para problemas de Regressão. Resumindo, pretende-se verificar se combinar abordagens existentes leva a alguma melhoria de desempenho. Além disso, investigar quais as configurações e características mais adequadas.

Como resultado, propomos uma *framework* híbrida de combinação de *Neural Network Ensembles* que realiza uma decomposição de erro de completa em *bias*, variância e covariância em cada algoritmo de forma a encontrar as maneiras mais promissoras de combinar as estratégias existentes mais complementares, com o objetivo de retirar o melhor de cada estratégia.

A criação de vários novos e inovadores algoritmos híbridos de *Ensemble* envolveu uma etapa teórica de conceptualização e modelação. Mais concretamente, e excluindo técnicas de variação de dados de treino para *Ensembles*, mais de 60 diferentes combinações de algoritmos foram desenvolvidas.

Os resultados de vastas experiências em múltiplos conjuntos de dados confirmaram os pressupostos teóricos de que combinar de múltiplas estratégias já estabelecidas de *Neural Network Ensembles* produz um aumento de desempenho claro e significativo em comparação com os modelos de base constituintes e as respectivas arquiteturas originais nas quais se baseiam. Da multiplicidade de novas estratégias híbridas de *Neural Network Ensembles* propostas, as de melhor desempenho diminuíram o erro global, em média, de 12% a 17% em comparação com as suas arquiteturas originais.

Em particular, combinar *Snapshot*, *Negative Correlation Learning* e *Dropout* oferece os resultados mais notáveis, mas também, adicionar estes três algoritmos a outras estratégias de *Ensemble Learning* melhora seu desempenho de forma distinta. No entanto, se se procurar por uma redução em específico de alguma componente de erro em certas arquiteturas de *Ensemble*, outras combinações podem ser aconselháveis de forma a evitar alçapões de algumas técnicas.

Portanto, e dado que as experiências foram consideradas estatisticamente válidas e confiáveis, prova-se que combinar diferentes abordagens de *Ensemble* é uma forma plausível e consistente de melhorar a capacidade preditiva tendo muito potencial de evolução.

**Keywords:** Machine Learning, Supervised Learning, Neural Networks, Ensemble Learning



# Acknowledgements

Two sincere kinds of acknowledgments are owed.

The first, individually, to my two supervisors. Professor João Pedro Moreira, thank you for having such a vibrant aura, from the get-go, around this project and for putting up my long, loud silences in those weekly meetings. Tiago Mendes Neves, thank you for clarifying numerous less than smart doubts. To both, a resounding thank you for awarding me the freedom required to determine the path to walk and for all your feedback and guidance throughout this journey.

The second, to all the people supporting me in these last few years, and, also, to some a little before that. No names are written so that I do not fall into the blunder of letting someone down.

Thanks to you all,

João



*“From a little spark  
may burst a flame.”*

Dante Alighieri



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context . . . . .	1
1.2	Problem . . . . .	2
1.3	Motivation . . . . .	2
1.4	Hypothesis . . . . .	3
1.5	Methodology . . . . .	3
1.6	Outline . . . . .	4
<b>2</b>	<b>Ensemble Learning</b>	<b>5</b>
2.1	Error Diagnosis . . . . .	7
2.1.1	Single Model . . . . .	7
2.1.2	Ensemble . . . . .	8
2.2	Background concepts on Ensemble Learning . . . . .	11
2.2.1	Base learner . . . . .	11
2.2.2	Homogeneous / Heterogeneous ensembles . . . . .	11
2.2.3	Meta-model . . . . .	12
2.2.4	Measure of Success . . . . .	12
2.2.5	Advantages . . . . .	13
2.2.6	Disadvantages . . . . .	14
2.3	Varying Training Data . . . . .	14
2.3.1	Random Splits . . . . .	14
2.3.2	K-fold Cross-training . . . . .	15
2.3.3	Pasting . . . . .	15
2.3.4	Bootstrapping . . . . .	15
2.3.5	Random Subspace . . . . .	16
2.3.6	Random Patches . . . . .	16
2.4	Varying Base Models' Characteristics . . . . .	16
2.5	Varying Base Models' Generation Strategy . . . . .	16
2.5.1	Bagging . . . . .	17
2.5.2	Boosting . . . . .	19
2.5.3	Bagging vs Boosting . . . . .	26
2.5.4	Stacking . . . . .	27
2.6	Prediction Integration Mechanism . . . . .	30
2.6.1	Average . . . . .	30
2.6.2	Weighted Average . . . . .	31
2.6.3	Meta-model . . . . .	32

<b>3</b>	<b>Neural Networks Ensembles</b>	<b>33</b>
3.1	Varying Training Data . . . . .	35
3.2	Varying Base Models' Characteristics . . . . .	36
3.2.1	Implicit Ensembles . . . . .	36
3.2.2	Explicit Ensembles . . . . .	37
3.2.3	Implicit and Explicit Ensembles Overview . . . . .	41
3.3	Varying Base Models' Generation Strategy . . . . .	41
3.3.1	Negative Correlation Learning . . . . .	41
3.3.2	Boosting Neural Network Ensemble . . . . .	46
3.3.3	Combining strategies for ensemble generation . . . . .	46
3.4	Applications . . . . .	47
<b>4</b>	<b>A framework to construct Neural Network Ensembles for regression</b>	<b>49</b>
4.1	Level-0 . . . . .	51
4.2	Level-1 . . . . .	58
4.2.1	Algorithm Hypothesis Formulation . . . . .	60
4.2.2	Results . . . . .	61
4.3	Statistical Validation . . . . .	70
4.4	Time analysis . . . . .	74
<b>5</b>	<b>Conclusions and Future Work</b>	<b>75</b>
5.1	Hypothesis Revisited . . . . .	75
5.2	Contributions . . . . .	76
5.3	Social Impact . . . . .	77
5.4	Future Work . . . . .	77
	<b>References</b>	<b>79</b>
<b>A</b>	<b>Neural Network</b>	<b>97</b>
A.1	Machine Representation . . . . .	97
A.1.1	Perceptron . . . . .	98
A.1.2	Hidden Layer . . . . .	98
A.2	Neuron mechanics . . . . .	99
A.2.1	Activation function . . . . .	99
A.2.2	Cost function . . . . .	100
A.3	Propagation . . . . .	100
A.3.1	Forward propagation . . . . .	100
A.3.2	Backpropagation . . . . .	100
A.4	NN Workflow . . . . .	102
A.5	Tuning . . . . .	103
A.5.1	Epochs . . . . .	103
A.5.2	Input variables range . . . . .	103
A.5.3	Learning Rate . . . . .	104
A.5.4	Regularization . . . . .	105
A.6	Disadvantages . . . . .	109
A.7	Deep Learning . . . . .	110
<b>B</b>	<b>Decision Tree</b>	<b>111</b>

<b>C</b>	<b>Kaggle Competitions</b>	<b>113</b>
C.1	Kaggle dominating algorithms . . . . .	113
C.2	Kaggle Top Competitor comment . . . . .	113
C.3	Netflix \$1M ML competition . . . . .	114
<b>D</b>	<b>Hybrid Algorithms' Pseudocode</b>	<b>115</b>
<b>E</b>	<b>Scientific Paper</b>	<b>119</b>





# List of Figures

2.1	Bias-variance trade-off . . . . .	7
2.2	Overfitting in Regression . . . . .	8
2.3	OK fit in Regression . . . . .	8
2.4	Underfitting in Regression . . . . .	8
2.5	Overfitting in Deep Learning . . . . .	8
2.6	OK fit in Deep Learning . . . . .	8
2.7	Underfitting in Deep Learning . . . . .	8
2.8	Low bias and Low variance . . . . .	10
2.9	Low bias and High variance . . . . .	10
2.10	High bias and Low variance . . . . .	11
2.11	High bias and High variance . . . . .	11
2.12	Bagging . . . . .	17
2.13	Boosting . . . . .	20
2.14	Pseudo-residual intuition . . . . .	22
2.15	Level-Wise approach . . . . .	24
2.16	Leaf-Wise approach . . . . .	26
2.17	Stacking . . . . .	28
2.18	Deep Stacking . . . . .	29
2.19	Super Learner . . . . .	30
3.1	Basic Ensemble architecture of NNs . . . . .	34
3.2	Cosine Annealing Cycles . . . . .	38
3.3	Comparison of Standard LR and Cyclic LR Schedules . . . . .	39
3.4	Polyak Averaging . . . . .	40
4.1	Individual ensemble results . . . . .	53
4.2	Comparison of different Random Split configurations . . . . .	56
4.3	Comparison of different Pasting, Random Subspace and Random Patches configurations . . . . .	56
4.4	Ensemble heatmap error results . . . . .	57
4.5	Comparison of Dropout empowered ensembles with different $p$ values . . . . .	59
4.6	Comparison of NCL empowered ensembles with different $\lambda$ values . . . . .	60
4.7	Proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' results . . . . .	61
4.8	Proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' error variation results . . . . .	62
4.9	Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' results . . . . .	62

4.10	Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' error variation results . . . . .	63
4.11	Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' bias variation results . . . . .	63
4.12	Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' variance variation results . . . . .	63
4.13	Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' covariance variation results . . . . .	64
4.14	Best-performing proposed hybrid NNE algorithms' results . . . . .	65
4.15	Best-performing proposed Dropout, NCL, Dropout+NCL empowered NNE algorithms' heatmap error results . . . . .	68
4.16	Best-performing proposed hybrid NNE algorithms' heatmap error results . . . . .	68
4.17	Individual ensemble Nemenyi test matrix . . . . .	71
4.18	Different configurations of Pasting, Random Subspace and Random Patches algorithms' Nemenyi test matrix . . . . .	71
4.19	Proposed Dropout, NCL, and Dropout+NCL empowered NNE Nemenyi algorithms' test matrix . . . . .	72
4.20	Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' Nemenyi test matrix . . . . .	72
4.21	Best-performing proposed hybrid NNE algorithms' Nemenyi test matrix . . . . .	73
A.1	Neural Network . . . . .	98
A.2	Perceptron . . . . .	98
A.3	Threshold activation function . . . . .	99
A.4	Sigmoid activation function . . . . .	99
A.5	Tanh activation function . . . . .	99
A.6	ReLU activation function . . . . .	99
A.7	Forward Propagation Model . . . . .	100
A.8	Backpropagation Model . . . . .	100
A.9	Gradient Descent . . . . .	101
A.10	NN workflow . . . . .	103
A.11	Comparison of multiple learning rate values . . . . .	104
A.12	Comparison of regular NN and Dropout NN . . . . .	106
A.13	Dropout: training vs testing . . . . .	106
A.14	Lasso . . . . .	107
A.15	Ridge . . . . .	107
A.16	Elastic Net . . . . .	107
A.17	Early Stopping . . . . .	109
C.1	Netflix \$1M machine competition rankings . . . . .	114

# List of Tables

2.1	Comparison of model fitting across the various domains . . . . .	8
2.2	Comparison of Bias-Variance values - 1 . . . . .	10
2.3	Comparison of Bias-Variance values - 2 . . . . .	11
2.4	Comparison of Bagging and Boosting algorithms' characteristics - 1 . . . . .	26
2.5	Comparison of Bagging and Boosting algorithms' characteristics - 2 . . . . .	27
3.1	Overview of Dropout-related researches . . . . .	37
3.2	Overview of Negative Correlation Learning related researches - 1 . . . . .	42
3.3	Overview of Negative Correlation Learning related researches - 2 . . . . .	43
3.4	Overview of Negative Correlation Learning related researches - 3 . . . . .	44
3.5	Overview of Negative Correlation Learning related researches - 4 . . . . .	45
3.6	Overview of Negative Correlation Learning related researches - 5 . . . . .	46
3.7	Overview of Practical Applications of NNEs - 1 . . . . .	47
3.8	Overview of Practical Applications of NNEs - 2 . . . . .	48
4.1	Overview of Generation Mode/Integration Methods - 1 . . . . .	51
4.2	Overview of Generation Mode/Integration Methods - 2 . . . . .	52
4.3	Level-0 datasets' information . . . . .	52
4.4	Individual ensemble numerical results . . . . .	55
4.5	Level-1 datasets' information - 1 . . . . .	58
4.6	Level-1 datasets' information - 2 . . . . .	59
4.7	Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' numerical results - 1 . . . . .	64
4.8	Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' numerical results - 2 . . . . .	65
4.9	Best-performing proposed hybrid NNE algorithms' numerical results . . . . .	67
4.10	Friedman tests . . . . .	70
4.11	Comparison of established and proposed algorithms time spent - 1 . . . . .	74
4.12	Comparison of established and proposed algorithms time spent - 2 . . . . .	74
A.1	Comparison of activation functions . . . . .	99
A.2	Comparison of standardization and normalization . . . . .	103
A.3	Comparison of adaptive learning rate methods - 1 . . . . .	104
A.4	Comparison of adaptive learning rate methods - 2 . . . . .	105
A.5	Comparison of weight regularization methods . . . . .	107
A.6	Comparison of activity regularization methods . . . . .	108
C.1	Overview of dominant algorithms in Kaggle competitions . . . . .	113



# Abbreviations

<b>AdaBoost</b>	Adaptive Boosting 20 – 21, 25, 41, 46 – 47, 51 – 52, 54 – 55, 74, 77
<b>Adagrad</b>	Adaptive Gradient Algorithm 49, 105
<b>AI</b>	Artificial Intelligence 1
<b>ANN</b>	Artificial Neural Network 97
<b>API</b>	Application Programming Interface 50
<b>Bagging</b>	Bootstrap Aggregating 17 – 18, 26, 41, 43 – 44, 46 – 47, 51, 53, 55, 74, 117
<b>BHNN</b>	Best Heterogeneous Neural Network 43
<b>BPNN</b>	Back-Propagation Neural Network 45
<b>CART</b>	Classification and Regression Tree 23, 111
<b>CatBoost</b>	Category Gradient Boosting 25, 51 – 52, 54 – 55, 74, 77
<b>CELS</b>	Cooperative Ensemble Learning System 42
<b>CNN</b>	Convolutional Neural Network 46, 113
<b>DART</b>	Dropouts meet Multiple Additive Regression Trees 25 – 26
<b>DL</b>	Deep Learning 2, 46, 50, 105, 110
<b>DNN</b>	Deep Neural Network 45, 110
<b>EDM</b>	Ensemble-based Decorrelation Method 45
<b>EENCL</b>	Evolutionary Ensembles with Negative Correlation Learning 43
<b>EFB</b>	Exclusive Feature Bundling 25 – 26
<b>EL</b>	Ensemble Learning 2, 4, 6, 14, 33, 46
<b>G-NCL</b>	Gated-Negative Correlation Learning 45
<b>GBDT</b>	Gradient Boosting Decision Trees 25 – 26
<b>GBM</b>	Gradient Boosting Machine 24, 113
<b>GD</b>	Gradient Descent 22, 101 – 102
<b>GOSS</b>	Gradient-based One-Side Sampling 25
<b>GPU</b>	Graphical Processing Unit 25
<b>HFC-PGA</b>	Hierarchical Pair Competition-based parallel Genetic Algorithm 44
<b>HNNE</b>	Heterogeneous Neural Network Ensemble 43
<b>LGBM</b>	Light Gradient Boosted Machine 25 – 26, 51 – 52, 54 – 55, 57, 74, 77
<b>LR</b>	Learning Rate 22 – 23, 36 – 41, 53, 58, 69, 101, 103 – 104, 107 – 108
<b>MART</b>	Multiple Additive Regression Trees 26

<b>ME</b>	Mixture of Experts <a href="#">45</a> , <a href="#">97</a>
<b>ML</b>	Machine Learning <a href="#">1</a> – <a href="#">2</a> , <a href="#">14</a> , <a href="#">16</a> , <a href="#">27</a> , <a href="#">32</a> , <a href="#">47</a> , <a href="#">105</a> , <a href="#">113</a> – <a href="#">114</a>
<b>MNCE</b>	Mixture of Negatively Correlated Experts <a href="#">45</a>
<b>MPPSO</b>	Multi-population particle swarm optimization <a href="#">43</a>
<b>MRNCL</b>	Multiobjective Regularized Negative Correlation Learning <a href="#">44</a>
<b>MSE</b>	Mean Squared Error <a href="#">41</a> , <a href="#">44</a>
<b>NC</b>	Negative Correlation <a href="#">41</a> , <a href="#">43</a>
<b>NCCD</b>	Negative Correlation Learning via Correlation-Corrected Data <a href="#">43</a>
<b>NCL</b>	Negative Correlation Learning <a href="#">41</a> – <a href="#">42</a> , <a href="#">44</a> – <a href="#">45</a> , <a href="#">51</a> , <a href="#">53</a> , <a href="#">55</a> , <a href="#">58</a> – <a href="#">69</a> , <a href="#">72</a> , <a href="#">74</a> , <a href="#">117</a> – <a href="#">118</a>
<b>NN</b>	Neural Network <a href="#">2</a> , <a href="#">4</a> , <a href="#">12</a> , <a href="#">22</a> , <a href="#">26</a> , <a href="#">34</a> – <a href="#">36</a> , <a href="#">40</a> – <a href="#">44</a> , <a href="#">45</a> , <a href="#">49</a> – <a href="#">54</a> , <a href="#">58</a> , <a href="#">60</a> , <a href="#">77</a> – <a href="#">78</a> , <a href="#">97</a> – <a href="#">103</a> , <a href="#">105</a> – <a href="#">110</a> , <a href="#">113</a> , <a href="#">115</a> – <a href="#">118</a>
<b>NNE</b>	Neural Network Ensemble <a href="#">2</a> – <a href="#">4</a> , <a href="#">33</a> – <a href="#">35</a> , <a href="#">42</a> , <a href="#">47</a> – <a href="#">49</a> , <a href="#">53</a> – <a href="#">54</a> , <a href="#">58</a> – <a href="#">65</a> , <a href="#">67</a> – <a href="#">69</a> , <a href="#">72</a> – <a href="#">73</a> , <a href="#">75</a> – <a href="#">77</a>
<b>OOB</b>	Out-of-Bag <a href="#">16</a> , <a href="#">44</a>
<b>P2P</b>	Peer-to-Peer <a href="#">47</a>
<b>ReLU</b>	Rectified Linear Unit <a href="#">49</a> , <a href="#">99</a>
<b>RMSprop</b>	Root Mean Square propagation <a href="#">105</a>
<b>RNCL</b>	Regularized Negative Correlation Learning <a href="#">44</a>
<b>RTQRT–NCL</b>	Root–Quartic Negative Correlation Learning <a href="#">44</a>
<b>RVFL</b>	Random Vector Functional Link <a href="#">37</a> , <a href="#">45</a>
<b>SGD</b>	Stochastic Gradient Descent <a href="#">23</a> , <a href="#">38</a> – <a href="#">40</a> , <a href="#">102</a> , <a href="#">104</a>
<b>SGDR</b>	Stochastic Gradient Descent with Warm Restarts <a href="#">38</a> – <a href="#">41</a> , <a href="#">53</a> , <a href="#">58</a> , <a href="#">61</a>
<b>Stacking</b>	Stacked Generalization <a href="#">27</a> – <a href="#">30</a> , <a href="#">41</a> , <a href="#">52</a> , <a href="#">54</a> – <a href="#">55</a> , <a href="#">57</a> , <a href="#">74</a>
<b>Tanh</b>	Hyperbolic Tangent <a href="#">99</a>
<b>XGBoost</b>	Extreme Gradient Boosting <a href="#">24</a> , <a href="#">51</a> – <a href="#">52</a> , <a href="#">54</a> – <a href="#">55</a> , <a href="#">74</a> , <a href="#">77</a> , <a href="#">113</a>

# Glossary

- Keras** is an open-source software library for Artificial Neural Networks [50](#), [113](#)  
**sklearn** is an open software library for Machine Learning [50](#), [113](#)  
**TensorFlow** is an end-to-end open-source platform for Machine Learning [50](#)





# Chapter 1

## Introduction

### Contents

---

<b>1.1 Context</b> . . . . .	<b>1</b>
<b>1.2 Problem</b> . . . . .	<b>2</b>
<b>1.3 Motivation</b> . . . . .	<b>2</b>
<b>1.4 Hypothesis</b> . . . . .	<b>3</b>
<b>1.5 Methodology</b> . . . . .	<b>3</b>
<b>1.6 Outline</b> . . . . .	<b>4</b>

---

### 1.1 Context

The emergence of data gathering propelled the need to derive insight from, usually, concealed meanings and patterns. A plausible way to achieve this goal is to use Machine Learning (ML) algorithms[1] which have an increasingly ubiquitous influence in everyone’s lives.

ML aims to study and develop computer algorithms that improve automatically through data to carry out specific tasks [2]. More concretely, ML strives, normally, to find a single model that searches through the hypothesis space to find a hopefully optimal hypothesis that expertly predicts a wanted outcome. It can be traced back to 1959 by Arthur Samuel <sup>1</sup>, an American IBMer <sup>2</sup> and pioneer in AI [3]. Since then, many advances have been made, and its use has widened. This widespread adoption can be seen from the top echelons of public quoted companies to the emerging startups and even single individuals due to the personal computer democratization.

Provided the task is simple, it is possible to explicitly program algorithms that guide the machine on how to execute all the required steps to solve the problem at hand, in which case, there

---

<sup>1</sup><https://history.computer.org/pioneers/samuel.html>

<sup>2</sup><https://www.ibm.com/pt-en>

is no learning needed on the computer's part [4]. For more advanced tasks, it is increasingly challenging for a human to craft the required algorithms manually. It might be more beneficial to let the machine develop its algorithm instead of having humans specifying every step needed [2]. This is a clear contrast from being explicitly programmed to do so [5].

ML employs numerous strategies to train computers on accomplishing tasks where no entirely satisfactory algorithm is available [4].

## 1.2 Problem

However, ML's benefits also carry risks [1]. One of which is that even if the hypothesis space contains very well-suited hypotheses for a single reality, it may be challenging to find a sole model that generalizes well for multiple realities. In other words, it may be affected by errors composed of noise, bias, and variance. Therefore, there is no guarantee that the chosen model properly performs on examples not observed during the training phase [1] [4].

Models may be formed by a variety of algorithms, namely, Neural Networks [6] (NNs). These are highly flexible non-linear methods capable of modeling complex data relationships [7]. However, they are highly sensitive to initial conditions, training algorithm's characteristics, Network characteristics, theoretical training dataset variance, and training cost [7]. Also, having low bias and high variance makes them be viewed as unstable [8].

Ensemble Learning (EL) is a successful research area in terms of predictive performance. By introducing the Ensemble technique, one can tackle the variance problem (lower predictions' overall variance) through "redundancy" introduction, specifically combining multiple models [9]. In turn, it is possible to obtain more accurate, stable, and robust predictions.

Nevertheless, NN allow for multiple configurations [8], and the Ensemble's success depends on the data's nature, estimators' characteristics, estimators' generation strategy, and the predictions' integration mechanism for which there is no exact way to decide the most appropriate. Also, Ensembles are highly dependant on their constituent estimators' diversity and accuracy. This means that each model may require a different knowledge of the hypothesis space to make distinct predictions/errors, thus providing slightly different forecasts for the Ensemble to aggregate over.

## 1.3 Motivation

Keeping in mind the above considerations, the Ensembles' field has had many developments over the years, with lots of published literature about Decision Tree Ensembles. Moreover, with recent years' NNs and Deep Learning (DL) success, EL also received new contributions.

However, the current Ensemble's landscape suffers from an evident lack of systematic researches, formulated hypotheses, theorized proposals, and implemented Neural Network Ensemble (NNE) approaches, specifically combining the most complementary existing and established strategies (paired strengths that most reduce each ensemble error component in conjunction). The

few that exist (section 3.3.3) are unlike what is put forward in this research, but further support the NNE's combination promising evolution path.

As a matter of fact, most studies are practical applications to concrete scenarios, tweaks to existing Decision-Tree specific frameworks, or very narrow and limited approaches rather than proposing new NNE hybrid frameworks from scratch that allow for multiple innovative combinations of algorithms with a complete error decomposition (bias, variance, and covariance), aiming to harvest each strategy's best characteristics. Given that Decision Tree Ensembles have demonstrably been proven to have excellent results in prediction accuracy improvement (very high performance), there is an expectation of adding value in the field of NNEs [8].

Consequentially, programmers would benefit from an increase in model performance and achieved results. More broadly, global societies would inherently profit from increased performance and reliability materialized in everyday devices.

So, there is a distinct need to evolve the current state-of-the-art.

## 1.4 Hypothesis

Being a research emphasis work, it focuses on exploring and evolving state-of-the-art techniques applied to NNEs in the field of Supervised Learning, namely for Regression problems, ideally by enhancing the performance of existing algorithms on tabular data. More concretely, this thesis proposes to test the following hypothesis:

Is it possible to improve the existing state-of-the-art  
Neural Network Ensemble approaches by combining them?

Knowing that Neural Network Ensembles allow for multiple  
configurations and have different characteristics,  
which are the most appropriate?

## 1.5 Methodology

It is a logical scheme with predefined steps to guide this thesis on testing the aforementioned hypothesis.

Before tackling the problem, it is necessary to have a profound and thorough knowledge basis of the ensemble's field of study current state-of-the-art. It is here that the need for a preliminary study arises. Only after that is it possible to ascertain the final results' trustworthiness and statistical relevance. Therefore, the following steps were followed:

- **Systematic Literature Review** - consisted of a quick, direct internet search, followed by a thorough exploration in recognized citation indexes (such as “Scopus”<sup>3</sup>, “Web of Science”<sup>4</sup>, and “CiteSeerX”<sup>5</sup>) of NN’s, EL’s, and, more profoundly, NNE’s related researches in the Supervised Learning field applied to regression modeling problems;
- **Critical Analysis** - after obtaining a multitude of articles and technical reviews, a quick analysis was performed to remove irrelevant documents. After that, an in-depth analysis was conducted to select documents that best apply to this thesis context;
- **Preliminary Study** - a meticulous knowledge apprehension of the current state-of-the-art trends and limitations to lay the foundations for the work to be developed. More specifically, fully understanding the referred field, the most common approaches, as well as some not so well recognized but leading the research line. Appendix 2, and 3 condense the required knowledge to understand this thesis, thus, act “almost” like a standalone document that can serve as an end-to-end structured survey.

Under the banner of the Preliminary Study, pursue a research line orientated towards overcoming those shortcomings by theorizing and implementing innovative state-of-the-art ensemble architectures. Then, systematically assess and conclude from the multiple proposed algorithms’ empirical and performance results if they constitute an improvement over the current techniques. Finally, elaborate on each of the new algorithms’ time requirements.

## 1.6 Outline

This thesis is divided into four other chapters:

- **Ensemble Learning** (Chapter 2, p. 5) - extensively describes the field of EL and its present status;
- **Neural Network Ensembles** (Chapter 3, p. 33) - comprehensively explains NNEs, as well as its current state-of-the-art. This chapter act as a direct “catalyst” for the work to be developed. More specifically, it introduces the basic theory of NNEs, its multiple approaches and established algorithms, and, in the end, analyzes various practical applications;
- **A framework to construct Neural Network Ensembles for regression** (Chapter 4, p. 49) - describes the proposed solution;
- **Conclusions and Future Work** (Chapter 5, p. 75) - focuses on making the work’s summary, synthesize its main ideas, and draw the final conclusions. In other words, it serves as the thesis wrap-up.

---

<sup>3</sup><https://www.scopus.com/home.uri>

<sup>4</sup><https://webofknowledge.com/>

<sup>5</sup><https://citeseerx.ist.psu.edu/index>

# Chapter 2

## Ensemble Learning

### Contents

---

<b>2.1</b>	<b>Error Diagnosis</b>	<b>7</b>
2.1.1	Single Model	7
2.1.2	Ensemble	8
<b>2.2</b>	<b>Background concepts on Ensemble Learning</b>	<b>11</b>
2.2.1	Base learner	11
2.2.2	Homogeneous / Heterogeneous ensembles	11
2.2.3	Meta-model	12
2.2.4	Measure of Success	12
2.2.5	Advantages	13
2.2.6	Disadvantages	14
<b>2.3</b>	<b>Varying Training Data</b>	<b>14</b>
2.3.1	Random Splits	14
2.3.2	K-fold Cross-training	15
2.3.3	Pasting	15
2.3.4	Bootstrapping	15
2.3.5	Random Subspace	16
2.3.6	Random Patches	16
<b>2.4</b>	<b>Varying Base Models' Characteristics</b>	<b>16</b>
<b>2.5</b>	<b>Varying Base Models' Generation Strategy</b>	<b>16</b>
2.5.1	Bagging	17
2.5.2	Boosting	19
2.5.3	Bagging vs Boosting	26
2.5.4	Stacking	27
<b>2.6</b>	<b>Prediction Integration Mechanism</b>	<b>30</b>
2.6.1	Average	30

2.6.2	Weighted Average . . . . .	31
2.6.3	Meta-model . . . . .	32

---

*Some EL techniques are estimator-specific (only work with particular base model algorithms), while others are not. This section explores general and Decision-Tree specific EL concepts and methods.*

One can “train many different candidate networks and then . . . select the best . . . and discard the rest. There are two disadvantages [to] such an approach. First, all of the effort involved in training the remaining networks is wasted. Second, . . . the network which had [the] best performance on the validation set might not be the one with the best performance on new test data” [6]. Furthermore, “it is generally impossible to know a priori which model will perform best for a given prediction problem and data set” [10].

Considering that there are already many different algorithms fitted on the dataset and each one has its strengths and weaknesses, why not, rather than using just a single model, use some or all group models to obtain an even more comprehensive predictive performance?

Ensemble’s principal hypothesis is that when a myriad (concrete finite set) of trained models on the same problem are rightly combined [11], they give rise to a strong learner [12] [13].

“Instead of looking for the best set of features and the best [estimator], now we look for the best set of [estimators] and then the best combination method. One can imagine that very soon, we will be looking for the best set of combination methods and then the best way to use them all” [13].

The intuition follows the “Unity is strength” saying. It simulates what is done when an expert panel in a human decision-making process combines their opinions. The fact is that the best single model “knows less” regarding the data compared to all other discarded models collectively combined. Combining models makes it possible to lower predictions’ overall variance while maintaining bias low simultaneously [14] [15].

Ambiguity decomposition has proven that a good ensemble guarantees lesser squared error than individual base learners [16] [17]. In this way, generalization error is reduced, and better performance is achieved [18] [6].

There are varying elements to build an ensemble [9], such as training data, estimators’ characteristics, estimators’ generation strategy, and predictions’ integration mechanism. Despite the intricacies of each step, varying the first three elements have the objective of promoting estimator diversity.

The ensemble was not proposed in a single research project, but rather a continuous and systematic process with a broad discussion among researchers [19] [20] [21] [22].

## 2.1 Error Diagnosis

Despite some concepts being often used interchangeably regarding single models and ensembles, they have different meanings.

### 2.1.1 Single Model

A model's intrinsic generalization error has its causes rooted in statistical noise, bias, and variance [15].

**Bias** is the contrast between the correct predictions and the model's predictions for the given data points. Low bias means the model's predictions are very close to the actual values (intricate model) and high bias (naive model) otherwise [15].

**Variance** is the model prediction's variability for a provided set of data points. Simply put, it describes how much a model changes when trained using different dataset sections due to small data sensitivity fluctuations (e.g., noise and specific observations). After being run several times, low variance occurs when the model's predictions do not vary much and high variance otherwise [15].

**Statistical noise**, also known as noise, denotes unexplained random variability in a data sample. Noisy data is meaningless data due to the presence of excessive variation. Often, and positively, the noisy data portion within a dataset is negligible.

The **Bias-Variance Trade-off** issue demonstrates that bias may be diminished to the detriment of augmented variance in any single Network and vice-versa (Fig. 2.1) [15]. These two usually change in opposing ways.

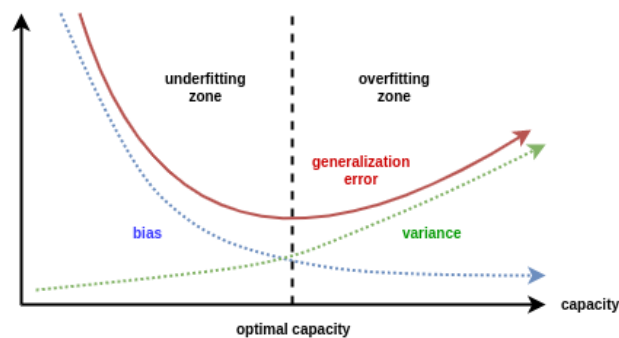


Figure 2.1: Bias-variance trade-off <sup>6</sup>

Focusing on a different subject, the model needs to have sufficient freedom degrees to “understand” the problem's data underlying complexities. Nonetheless, it is paramount to minimize the freedom degrees to circumvent high variance and, so, be further robust. Table 2.1 explores this topic thoroughly.

<sup>6</sup><https://www.programmersonsought.com/article/89934748226/>, last accessed on 2021-01-21

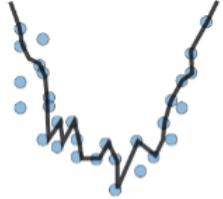
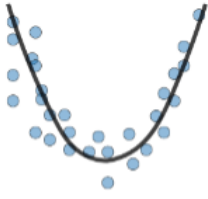
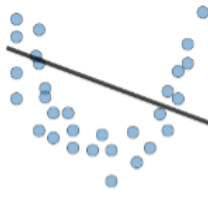

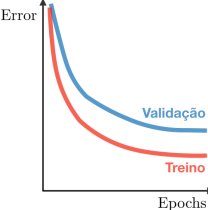

	<b>Overfitting</b>	<b>Just Right</b>	<b>Underfitting</b>
<b>Symptoms</b>	<ul style="list-style-type: none"> <li>- Training error much smaller than test error</li> <li>- Small training error</li> <li>- High variance</li> </ul>	<ul style="list-style-type: none"> <li>- Training error slightly smaller than the test error</li> </ul>	<ul style="list-style-type: none"> <li>- Training error close to the testing error</li> <li>- High training error</li> <li>- High bias</li> </ul>
<b>Regression</b>	 <p>Figure 2.2: Overfitting in Regression <sup>7</sup>.</p>	 <p>Figure 2.3: OK fit in Regression <sup>7</sup>.</p>	 <p>Figure 2.4: Underfitting in Regression <sup>7</sup>.</p>
<b>Deep Learning</b>	 <p>Figure 2.5: Overfitting in Deep Learning <sup>7</sup>.</p>	 <p>Figure 2.6: OK fit in Deep Learning <sup>7</sup>.</p>	 <p>Figure 2.7: Underfitting in Deep Learning <sup>7</sup>.</p>
<b>Solutions</b>	<ul style="list-style-type: none"> <li>- Regularize</li> <li>- Get more data</li> </ul>		<ul style="list-style-type: none"> <li>- Complexify model</li> <li>- Add more features</li> <li>- Train longer</li> </ul>

Table 2.1: Comparison of model fitting across the various domains.

Hence, the model choice is critical to obtain good results.

### 2.1.2 Ensemble

An ensemble's intrinsic generalization error has its causes rooted in statistical noise, bias, variance, and covariance [15] [23] [17] [24].

Apart from statistical noise in the single model context, which retains its original meaning in the ensemble context, the remaining error metrics have their definitions altered.

**Bias** is the average discrepancy between the ensemble's output and the base learner's outputs.

**Variance** is the average disagreement between the base learner's outputs. The lower it is, the more stable, robust, and reliable the respective estimators become, and vice-versa.

**Covariance**, also known as *Joint Variance*, measures the joint variability (degree of entanglement/interdependence) between two random variables [25] [26]. Simply put, the covariance sign

<sup>7</sup><https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-machine-learning-tip-s-and-tricks>, last accessed on 2021-01-21



displays the linear relationship tendency between variables [27]. Mathematically, it measures the mean pairwise difference between the different base learners. If two base learners' predictions vary in the same direction, then their covariance is positive [28]. Opposingly, if they differ in opposite directions, their covariance is negative [28]. If they vary independently one from the other, covariance is zero [28].

**Global Error**, composed of noise, bias, variance, and covariance, measures the average generalization error of each individual estimator. It can be analyzed from three different perspectives.

Exploring the mathematical side, the ensemble's estimator outputs for some input  $x$  are defined as the  $M$  estimators' predictions simple average for  $x$  after having been separately trained (Eqn. 2.1).

$$f_{ens}^{(M)}(x) = \frac{1}{M} \sum_{m=1}^M f_m(x; z_{(m)}^N) \quad (2.1)$$

where Let  $f_1, \dots, f_M$  denote  $M$  estimators, where the  $m$ th estimator is separately trained on  $z_{(m)}^N$ ,  $m = 1, \dots, M$

The training set's sample size is assumed to be uniformly  $N$ . Note that training set  $z_{(m)}^N$ , is a realization of a random sequence  $Z_{(m)}^N$  and that  $Z_{(m)}^N$ ,  $m = 1, \dots, M$ , have the same distribution  $p(x, y)$ ; however, they cannot always be assumed to be mutually independent. The ensemble output for some input  $x$  is defined as the Simple Average of  $M$  estimators' outputs for  $x$  after being separately trained.

Bias-variance-covariance decomposition has been analyzed through multiple **mathematical breakdowns** [15] [26] [23] [17] [24]. Although ensembles have been supported by numerous theories such as strength correlation [29], stochastic discrimination [30], bias-variance [16] [31] [32], and margin theory [33], these provide equivalent insight as the stated bias-variance-covariance relation (Eqn. 2.2) [34] [26].

The multiple error components are analyzed in Eqn. 2.2, and the Global Error is dissected in Eqn. 2.3.

$$GErr(f_{ens}^{(M)}) = E_{x_o} \left\{ \frac{1}{M} \overline{Var}(X_o) + \left(1 - \frac{1}{M}\right) \overline{Cov}(X_o) + \overline{Bias}(X_o)^2 \right\} + \sigma^2 \quad (2.2)$$

where  $\overline{Var}(X_o)$ ,  $\overline{Cov}(X_o)$ ,  $\overline{Bias}(X_o)$ , and  $\sigma^2$  are average conditional variance, conditional covariance, conditional bias averaged over  $M$  estimators, and noise, respectively:

$$\begin{cases} \overline{Var}(X_o) = \frac{1}{M} \sum_{m=1}^M Var\{f_m|X_o\} \\ \overline{Cov}(X_o) = \frac{1}{M(M-1)} \sum_m \sum_{m' \neq m} Cov\{f_m, f_{m'}|X_o\} \\ \overline{Bias}(X_o) = \frac{1}{M} \sum_{m=1}^M Bias\{f_m|X_o\} \end{cases}$$

Here  $f_m$  denotes  $f_m(X_0; Z_{(m)}^N)$ . Using the following average generalization error averaged over  $M$  estimators,

$$\overline{GErr} = \frac{1}{M} \sum_{m=1}^M (E_{X_0} \{Var\{f_m|X_0\} + Bias\{f_m|X_0\}^2\} + \sigma^2) \quad (2.3)$$

However, there are two caveats:

- the 2.2 equation cannot be directly applied to classification problems due to their categorical nature as it needs to be derived [31] [35] [36] [37] [38];
- the error decomposition equation assumes Simple Average (section 2.6.1) as the prediction integration mechanism [26]. This assumption might skew the bias, variance, and covariance distribution slightly when using Weighted Average (section 2.6.2). A possible future solution would be to derive the mathematical formula for Weighted Average, apply it to each base learner, and validate the referred hypothesis. Undoubtedly, the error value would not be affected. Still, it would be more challenging because the weights are calculated during the ensemble training, at which time the error decomposition is no longer useful.

The **combination of different bias-variance values** is summed up in Tables 2.2 and 2.3.

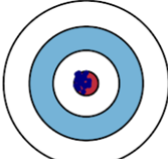
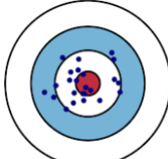
	Low Variance	High Variance
Low Bias	 <p>Figure 2.8: Low bias and Low variance (adapted from <sup>8</sup>).</p> <p>The ideal case. Best results are consistently obtained</p>	 <p>Figure 2.9: Low bias and High variance (adapted from <sup>8</sup>).</p> <p>On average, accurate but inconsistent results</p>

Table 2.2: Comparison of Bias-Variance values - 1.



<b>High Bias</b>		Figure 2.10: High bias and Low variance (adapted from <sup>8</sup> ).
	On average, consistent but not accurate results	
		Figure 2.11: High bias and High variance (adapted from <sup>8</sup> ).
	Neither accurate nor consistent results	

Table 2.3: Comparison of Bias-Variance values - 2.

## 2.2 Background concepts on Ensemble Learning

### 2.2.1 Base learner

Also known as *Base Model* or *Estimator*, it acts as the building block for designing more complex models. Each one extracts complex, often implicit, data relations. Ideally, they have skill in random ways, or, in other words, they have diversity among each other [16] [39].

More generally, the desire for somewhat under-optimized models is valid for ensemble members' selection. The "members of the committee should not individually be chosen to have an optimal trade-off between bias and variance. Instead, they should have a relatively smaller bias since the extra variance can be removed by averaging" [6]. Also, "[for] learning in large ensembles, it is advantageous to use under-regularized students, which actually over-fit the training data ... in particular if the individual students are subject to noise in the training process. Choosing students with a wide range of regularization parameters makes this improvement robust against changes in the unknown level of noise in the training data" [40].

### 2.2.2 Homogeneous / Heterogeneous ensembles

Homogeneous means every estimator originates from the equivalent algorithm. Oppositely, Heterogeneous means different algorithms induce some or all estimators.

Some may interpret that heterogeneous ensembles may provide better generalization performances considering their estimators come from "different families of models" thus have distinct data perspectives, internal workings, and decision fusion strategies. However, no concrete study corroborates this belief.

<sup>8</sup><https://medium.com/@itbodhi/bias-and-variance-trade-off-542b57ac7ff4>, last accessed on 2021-01-21

### 2.2.3 Meta-model

Also known as *Meta-algorithm*, it is an algorithm that learns other algorithms. In other words, it decides how to take a set of other (typically, but not necessarily non-meta) base models (level 0) to construct a Meta-model out of those (level 1). To achieve that, it adaptively combines or weights the component algorithms' outputs, commonly viewed as black-boxes (taking input and producing their outputs with the inner workings hidden) to make a final prediction [41].

In practice, Meta-model techniques' performances are dependant, in addition to the ensemble's usual requirements (section 2.2.4), on having a simple Meta-model that provides smooth prediction interpretations that offset individual models' deficiencies, customarily leading to better overall performance [1]. Hence, linear models are often used as the combiner but may be any algorithm such as NNs [42]. This combiner algorithm learns when to trust and how to best combine each base learner's predictions by conditionally deciding to weigh them differently [43]. In practice, this translates into determining how best to map the learner's determinations into an upgraded output [44].

In regression modeling tasks, the Meta-model is also referred to as Meta-regressor.

### 2.2.4 Measure of Success

The ensemble's success depends on the training data's nature, estimators' characteristics, estimators' generation strategy, and the predictions' integration mechanism [45].

If base learners make highly correlated predictions, the ensemble's accuracy will hardly improve over a single base learner. In addition to this, if base learners do not make correct predictions, the ensemble will also struggle to make correct predictions.

Deep down, the ensemble works appropriately because the member models yield a degree of accuracy (better than random guess) and diversity (disagree among each other with different output values) [46] [11] [47] [16]. Building a good ensemble is an exercise of carefully balancing the accuracy and correlation of base learners' predictions.

By not looking at the same input space (slightly different datasets) [46], it allows each unstable model to apprehend varying viewpoints of the input/hypothesis space, thus gaining different capabilities/knowledge, making different minor assumptions about solving a predictive modeling task, and promoting estimator disagreement. Consequentially, they make distinct predictions/errors and display low mutual prediction correlation (uncorrelated) [16]. Finally, the resulting ensemble has a more diversified estimator collection, which provides slightly different forecasts to combine. Accordingly, it better understands the whole training dataset and achieves more reliable performance [48].

Combining several base learners' outputs is only helpful if they are diverse and disagree on some inputs. Therefore, it is crucial to promote it among the base models even if at the expense of a moderate increase in their individual error rate values since it is expected to improve the ensemble's performance [9] [49]. On the other hand, keeping a small number of base models

allows for faster training (most computationally expensive step in the NN architecture) and may also improve results [50] [51].

Note, grid search or other tuning techniques, typically, should not be used in the base models. It would obtain highly tuned base learners with low or zero mutual variety, hampering the ensemble's effectiveness.

### 2.2.5 Advantages

The advantages of using ensembles are multiple:

- the final result is not decided by a single member but by a committee [52]. On the first hand, this makes the ensemble significantly more reliable and robust, meaning it has better adaptability to the true unknown function approximation and predictive performance, on average, compared to any single model [53] [8] as individual member weaknesses and errors are “averaged out” by other members' strengths [46];
- it improves stability since the final model is less susceptible to the training scheme choice, initial conditions (e.g., random initial weights), training data's specificities, or the single training run serendipity [7]. Improved stability means the ensemble is consistently more accurate [54];
- for the same increase in computation, the ensemble may be more efficient at improving overall accuracy because it uses that increase on two or more methods rather than increasing resources for a single method broadly;
- it is remarkably flexible. There is no limit on the number of models to use. This number may vary based on the complexity of the problem and model. However, as the number of ensemble models increases, performance typically improves monotonically, although with consecutive diminishing gains. Nonetheless, improvements are more pronounced with greater diversity;
- it is possible to add/remove estimators iteratively;
- not all base learners necessarily need to finish on time, thus introducing a kind of graceful degradation where a single model loss is not disastrous for conceiving reliable predictions [55];
- it “has been empirically observed that certain ensemble techniques often do not overfit the model, even when the ensemble contains thousands of [estimators]” [41];
- ensembles have a wide range of potential applications and validity.

These theoretical benefits and practical successes have established reasons in statistical, computational, and representational terms [48].

Therefore, at the time of writing, heterogeneous ensembles are one of the most robust approaches to improve ML algorithms, giving a 1%-3% increase in accuracy on top of that of the already tuned models. So, ensembles are the dominant approach, provided that predictive capability is the most crucial factor. In fact, they are the preeminent winning tactic for top ML competitions (appendix C). “Machine-learning competitions, particularly on Kaggle, [see] winners [using] huge ensembles of models that inevitably beat any single model, no matter how good” [56].

### 2.2.6 Disadvantages

There are a few disadvantages to using ensembles:

- it may be resource and time-intensive, depending on the model’s complexity and training data size, compared to a single model prediction [9]. In fact, EL algorithms are, without a doubt, more complex models if their number of parameters is taken as the complexity measure;
- it is even more challenging to characterize and explain predictions.

Not to forget that there is the likelihood that the ensemble’s performance is superior to that of the best-performing base model. Nevertheless, this is not guaranteed [9]. In that case, the best-performing base learner should be used instead, given its lower complexity, if looking for accuracy only. Note that the ensemble can still offer lower variance, thus, higher stability [9].

## 2.3 Varying Training Data

Varying the data choice used in each ensemble’s model provides each estimator with a different framing or view of the problem [9]. In practice, it is possible to generate training sets to promote an ensemble of diverse estimators by extracting different data (rows) or features (columns) from the original dataset.

These techniques assume individual models trained with higher diverse sets result in higher diverse models. However, it is essential to understand that a higher training dataset diversity is an insufficient condition on its own to generate individual models with higher diversity. In other words, training set diversity promotes but does not guarantee model diversity. Hence they are not guaranteed conductors to the ensemble’s overall performance increase.

Simultaneously, these methods only primarily focus on mutual model diversity and not individual accuracy in itself. In other words, they expect that promoting diversity at the expense of, sometimes, lowering individual performance may, in the end, improve the ensemble’s results.

### 2.3.1 Random Splits

Refers to repeatedly performing random splits on a dataset to build different train and test sets for each base learner [1] [57].

### 2.3.2 K-fold Cross-training

Similar to K-fold Cross-validation [4], it divides the dataset into  $k$  equally sized folds. All observations may be utilized during the model's training as each base learner is fitted on  $k - 1$  different folds and may be tested on the remaining  $k$  fold (holdout set) [16]. Remember that  $k$  is equal to the number of estimators.

### 2.3.3 Pasting

Also known as *Random Dataset Sampling*, the base learners are fitted on training dataset random samples without replacement, without necessarily being contiguous, and with a smaller size than the original set. Best employed when the memory constraints do not allow the training set to be fitted.

Pasting “takes small pieces of the data, grows a predictor on each small piece, and then pastes these predictors together. A version is given that scales up to terabyte data sets. The methods are also applicable to online learning” [58].

### 2.3.4 Bootstrapping

Refers to a robust and accurate statistical method to estimate and evaluate a model's skill when making predictions on unseen data, typically in small data samples [59]. In other words, it estimates statistical population scores in comparison to a sole original dataset estimation based on the realizations of the models [60]. These are presented as estimate statistics summary, including variance, mean, standard deviation, and empirical non-parametric confidence intervals of some statistical estimators based on multiple bootstrap samples [60] [61].

Despite Bootstrapping not being an Ensemble-specific strategy, it may be utilized as such. In this case, a **Bootstrap sample** is a randomly resampled (with replacement) subset of selected training points from the full training dataset [46]. In practice, each time an item is chosen from the initial data, it is not excluded. Any element has the same probability of being selected repeatedly; hence, some observations may be duplicated or not appear at all.

The models are fitted using slightly different training data perspectives as (almost) always a slightly different training dataset is generated following a Gaussian distribution (central tendency) [62]. This promotes base model diversity [4].

It also allows for good statistical “approximate properties”, such as representative and independent identically distributed (*i.i.d.*) samples extracted from a distribution [1]. Representative means the original dataset size  $N$  ought to be ample to apprehend the majority of the underlying distribution intricacy and ensure meaningful sample statistics can be calculated confidently [1]. Independence means that the size  $N$  ought to be sufficiently large when confronted with the bootstrap sample size  $B$  not to be over correlated [1].

Theoretically, the “averaging” action is performed across the (nearly) *i.i.d.* fitted base learners. Hence, it preserves the expected value and reduces variance (standard deviation/error) [1]. It is essential to keep in mind that this is only an approximation.

As a byproduct, the “samples not selected are usually referred to as the “out-of-bag” (OOB) samples. For a given iteration of bootstrap resampling, a model is built on the selected samples and is used to predict the out-of-bag samples” [4]. It allows one, without any additional data observations, to estimate the generalization performance.

Best employed when:

- there is not sufficient data to acquire an un-biased model performance estimation applying the train-test split evaluation;
- there is a requirement for a sound estimate of performance on unseen data. If not, then a single train-test split can be used;
- the computational cost of fitting two or more estimators on the training sample is not prohibitive. If not, all available resources should be put into fitting a single model.

### 2.3.5 Random Subspace

Also known as *Random Input Feature Subspace*, base learners are fitted on datasets built from randomly collected training data feature space subsets (feature selection) [49] [45]. Although the input subset’s random selection may induce a lower training precision, the models’ diversity gained improves the ensemble’s performance [49] [9] [4].

It may be used with any ML algorithm. However, it is better suited to models more susceptible to considerable changes in the input features, such as K-Nearest Neighbors and Decision Trees.

### 2.3.6 Random Patches

Combines both Pasting and Random Subspace ensembles. Base models are fitted on random training data patches (subsets of rows and columns, representing samples and features, respectively) [63].

## 2.4 Varying Base Models’ Characteristics

Varying the choice of base model comprises multiple possibilities as each algorithm has different strengths and weaknesses [9]. Additionally, each one has its respective definable parameters with pros and cons both on its own and in conjuncture with others. Thus, it creates an even greater possibility space with multiple different options from which to choose.

## 2.5 Varying Base Models’ Generation Strategy

Varying the estimator’s generation techniques aims to create multiple different models, each with predictions of its own that form the ensemble.



### 2.5.1 Bagging

Also known as *Bootstrap AGGregating* or *BAGGED Decision Trees*, it comprises Bootstrapping (section 2.3.4) and prediction integration mechanism concepts in a deterministic manner (Alg. 1) [60] [54]. The estimators are homogeneous (section 2.2.2) independent models, usually, but not mandatory, consisting of Decision Trees [9].

A “natural way to reduce the variance and hence increase the prediction accuracy of a statistical learning method is to take many training sets from the population, build a separate prediction model using each training set, and average the resulting predictions. . . . Of course, this is not practical because we generally do not have access to multiple training sets. Instead, we can bootstrap” [60].

---

#### Algorithm 1 Bagging

---

**Input:** training sample  $S$ , test sample  $T$ , number of base learners  $M$

**Output:** final prediction  $P$

- 1: **for**  $m = 1$  to  $M$  **do**
  - 2:      $S_m =$  bootstrap sample from  $S$
  - 3:     train base learner  $h_m$  on  $S_m$
  - 4: **end for**
  - 5:  $P = \sum_{m=1}^M \text{pred}(T)_{\text{base\_learner}_m} / M$
- 

Fitting truly independent models cannot, in practice, be achieved because it requires too much data. So, Bagging relies on good different bootstrap samples to fit “almost” independent base learners [64]. Each estimator is fitted with different bootstrap samples based on the original dataset (normally of the same size) [54] in **parallel** [4], independently from each other, exploiting the base learners' autonomy (Fig. 2.12). Due to its nature, it is considered a **variance reduction algorithm**. The final prediction depends on the problem at hand. The Simple Average of the members' predicted probabilities is performed (regression modeling problems). For small ensembles, there may be a lack of sample meaningfulness.

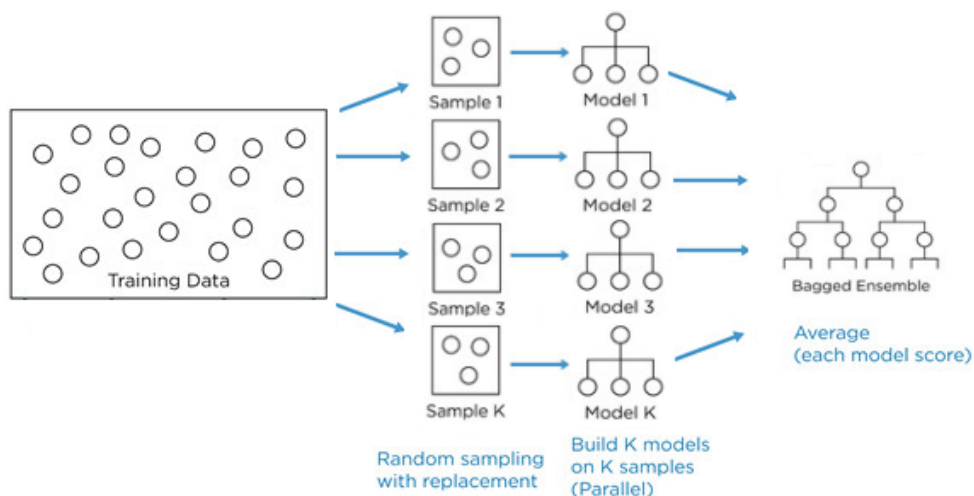


Figure 2.12: Bagging (adapted from <sup>9</sup>).

A significant advantage is that Bagging rarely overfits [9]. Also, it works best with small or medium-size datasets [45] and unstable base models [9]. In other words, models with a modest variance, thus somewhat conditioned on the training data specificities [9].

“If perturbing the learning set can cause significant changes in the predictor constructed, then bagging can improve accuracy” [54]. However, the “evidence, both experimental and theoretical, is that bagging can push a good but unstable procedure a significant step towards optimality. On the other hand, it can slightly degrade the performance of stable procedures” [54].

It is relevant to note that Bagging does not overfit as rapidly as AdaBoost in high noise scenarios [48].

### 2.5.1.1 Random Forest

It is a Random Patches variation (inherently, Bagging variation) and an **Estimator-specific** ensemble for Decision/Regression Trees. It brings together various concepts [29], offering high predictive performance and simple hyper-parameter tuning [64].

Multiple Random Forest characteristics contribute to its success:

- it leverages multiple unpruned Decision Trees’ power [29] instead of relying on a single Tree and expecting the correct conclusion was made at each split. Furthermore, by being unpruned, they exhibit the aforementioned benefits [4] [64];
- equal to Bagging [29], it fits, in parallel and independently,  $N$  Decision Trees on  $N$  randomly selected bootstrap samples from a larger dataset [4];
- Random Input Feature Subspace Selection refers to a “small tweak that decorrelates the trees” [60] [29]. It involves selecting a subset of input features based on the Random Selection algorithm on which the split is then performed. Diving deeper, instead of splitting at each node at similar features progressively, the features are reduced to a random subset that may be considered at each split point. This differentiation level makes the Trees more different, resulting in a greater, more diverse ensemble to aggregate over [60] [4];
- optimal split point greedy selection.

Subsequently, the hyperparameters are the number of Decision Trees, their depth, the number of random features to consider at each split point, and minimum improvement to loss [60].

As a side note, it may be used for imbalanced datasets [65].

### 2.5.1.2 Extra Trees

Also known as *Extremely Randomized Trees*, it is a Random Forest variation and an **Estimator-specific** ensemble for Decision/Regression Trees. The difference is that it uses a simpler algorithm to construct Decision Trees [66]. Nevertheless, it often achieves equal or even better performance than the Random Forest algorithm.

---

<sup>9</sup><https://www.kdnuggets.com/2019/09/ensemble-learning.html>, last accessed on 2021-01-21

It creates a vast amount of unpruned Decision Trees fitted on the whole dataset [66]. Also, each split point is chosen at random, and for each one, Random Input Feature Subspace Selection is employed instead of seeking the most discriminative ones [66]. This makes the Decision Trees in the ensemble more diverse [66].

There are only a few key hyperparameters to configure, more specifically, the number of Decision Trees as base learners and the amount of input features to select and consider at each split point, randomly.

The Simple Average of the members' predictions is performed (regression modeling problems).

### 2.5.2 Boosting

Also known as *Hypothesis Boosting*, it is built iteratively and incrementally to turn weak learners into strong learners (Alg. 2) [9]. The intuition is that to consult several doctors that base their diagnosis on a deterministic integration of prior diagnoses results.

Has his roots in the PAC learning model, in the enigma “Can a set of weak learners create a single strong learner?” [67], and in the subsequent hypothesis that it is possible for “an efficient algorithm for converting [(boosting)] relatively poor hypotheses into very good hypotheses” [67]. In other words, a weak learner (e.g., perform better than random chance) may be bettered by altering it. Later, it expanded to the concept of observation filtering, neglecting observations the weak learner can manage and concentrating on producing new models to tackle the problematic leftovers [68]. This led to its first proto development [69] and later a proper architecture [37].

---

#### Algorithm 2 Boosting

---

**Input:** training sample  $S$ , test sample  $T$ , number of weak learners  $M$

**Output:** final prediction  $P$

- 1: initialize weights  $w_n$
  - 2: **for**  $m = 1$  to  $M$  **do**
  - 3:     fit weak learner  $h_m$  on weighted data sample
  - 4:     aggregate weak learner to the ensemble
  - 5:     “update” the training dataset
  - 6: **end for**
  - 7:  $P = \sum_{m=1}^M weight_{base\_learner\_m} \times pred(T)_{base\_learner\_m}$
- 

In practical terms, it learns how to optimize each model's advantages by refocusing attention and building upon prior chain models to fix the current prediction errors and “updating” the training dataset [68] [69] [70]. Models are **sequentially** fit one at a time, hence, dependent on the fitted models in previous steps (Fig. 2.13) [71]. This is to exploit weak learners' dependence by considering the strengths and weaknesses of previous models when fitting the current one. Consequentially, subsequent models will perform better than their predecessors. Thus, theoretically, it is possible to achieve a flawless model. Due to its nature and not neglecting its capacity to reduce variance, it is considered a **bias reduction algorithm**.

In regression modeling problems, the final prediction is the Weighted Average of the members' predictions.

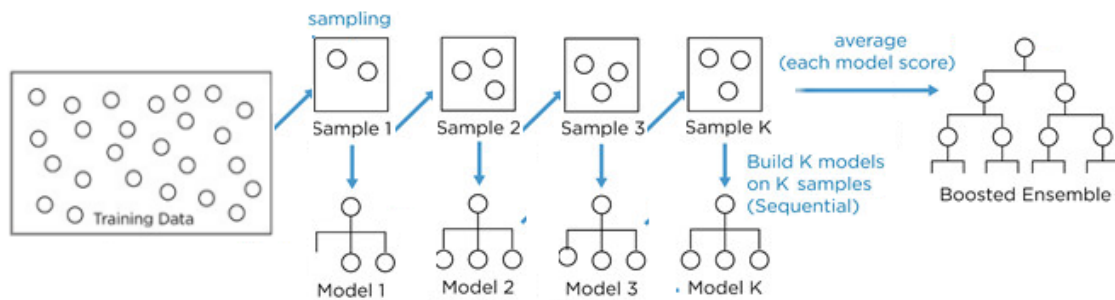


Figure 2.13: Boosting (adapted from <sup>10</sup>).

However, some information has to be defined as to how the weak learners will be sequentially fitted. More pointedly, which information from preceding learners needs to be considered when fitting the current model, how to aggregate to the previous models the current one, and the specific algorithms to use.

A “very surprising finding is that performing more boosting iterations can reduce the error on new data long after the error of the combined [estimators] on the training data has dropped to zero” [72].

It is also relevant to note that Boosting has a potential bias for some “difficult” samples. Therefore, this method is sometimes unstable.

Several Boosting variations and third-party library implementations co-exist in parallel, each with its unique characteristics.

### 2.5.2.1 AdaBoost

Also known as *Adaptive Boosting*, it is a Boosting-based ensemble algorithm and, perhaps, its first successful and most representative implementation (Alg. 3) [69]. It considers homogeneous (section 2.2.2) independent weak learners, normally, but not mandatory, Decision Trees, specifically Decision Stumps [60] [73]. It successfully finds the best base model at each iteration by solving the exact “local” optimization problem and having the original dataset sample’s and model’s weights varying proportionally to their ensemble error contribution and performance, respectively [74]. It is then added to the strong model [75].

The algorithm is designated “AdaBoost because, unlike previous algorithms, it adjusts adaptively to the errors of the weak hypotheses” [69].

More specifically, there are two sets of weights. The first set is assigned to each data point based on the current ensemble’s results meaning its distribution is over-represented by the patterns that earlier learners incorrectly recognize. More concretely, higher weight is given to incorrectly predicted data (boosted [74]) and less weight otherwise [4] [69]. This ensures they have a greater likelihood of being incorporated in the next model’s training set, making the sampled data geared

<sup>10</sup><https://www.kdnuggets.com/2019/09/ensemble-learning.html>, last accessed on 2021-01-21

towards increasingly hard-to-predict instances [74]. Subsequently, learners will focus their training intensities on harder and harder observations fitted up to that moment to correct prior models' prediction mistakes. The second set is assigned to each learner based on its scalar evaluation metric (coefficient) and indicates how much it ought to be considered in the ensemble. Each learner's error rate is calculated as the misclassified examples sum of the weights [60]. Higher weight is given to weak learners with better results and less weight otherwise [60] [64] [69]. Hence, previous sequence models weight the final prediction by their demonstrated accuracy [76] [77] [78].

---

**Algorithm 3** AdaBoost
 

---

**Input:** training sample  $S$ , test sample  $T$ , number of weak learners  $M$

**Output:** final prediction  $P$

- 1: initialize weights  $w_n$  to  $\frac{1}{\#S}$
  - 2: **for**  $m = 1$  to  $M$  **do**
  - 3:    $s_m =$  weighted sample from  $S$
  - 4:   fit a weak learner  $h_m(x)$  on  $s_m$  by minimizing weighted error function  $J_m$
  - 5:   compute  $J_M = \sum_{n=1}^{\#S} w_n [h_m(x_n) \neq t_n]$
  - 6:   compute  $\varepsilon_m = J_M / \sum_{n=1}^{\#S} w_n$
  - 7:   evaluate the update coefficient -  $\alpha_m = \log \frac{1-\varepsilon_m}{\varepsilon_m}$
  - 8:   add weak learner multiplied by the update coefficient to the ensemble
  - 9:   update the weights:  $w_n^{m+1} = w_n^m \exp(\alpha_m 1[h_m(x_n) \neq t_n])$
  - 10: **end for**
  - 11:  $P = \sum_{m=1}^M \text{weight}_{\text{base\_learner\_}m} \times \text{pred}(T)_{\text{base\_learner\_}m}$
- 

Despite being initially just one architecture for classification problems, currently, there have been regression proposed variations (AdaBoost.R2 [77], and L2Boost [79]). These derived versions essentially differ in their loss function preference and how each instance's weights, at each iteration, are redefined.

In regression modeling problems, the final prediction is the Weighted Average of the estimators' predictions.

Despite its achievements, it is computationally expensive. Often it is prolonged to train a model due to being a sequential algorithm, hence, difficult to parallelize. On top of that, it was demonstrated that having a very large number of outliers become performance detrimental. A possible solution is to employ Gentle AdaBoost [80].

### 2.5.2.2 Gradient Boosting

Also known as *Gradient Tree Boosting*, it is an **Estimator-specific** ensemble for Decision/Regression Trees and an AdaBoost variation formulated as a numerical optimization problem to a differentiable and arbitrary amount of loss functions (Alg. 4) [78]. It is one of the most robust methods to build predictive models [81] [82].

Some traits distinguish Gradient Boosting from AdaBoost:

- it does not change the observations' selection or distribution. Rather, it aims to solve the sequential conundrum by casting the Boosting procedure as a numerical optimization problem;
- for each distinct loss function, the model does not need to be inferred. The aim is to reduce the model's loss gradient as the model is fit (much like a NN) by combining base models through a GD like procedure, more specifically, “functional gradient descent” (or “gradient descent with functions”) [60] [1].

It is considered a stage-wise additive model. A “stagewise strategy is different from stepwise approaches that readjust previously entered terms when new ones are added” [83]. At a time, a fresh estimator is added, while the existing models are frosted and left unchanged. This weak learner is fitted in opposition to the current fitting error gradient of the ensemble observations. The resulted comparison is referred to as pseudo-residual (Fig. 2.14). For each sole observation, these designate in which course to update and “fix” the ensemble by lowering its prediction error. They can be seen as the weak learners' “targets”.

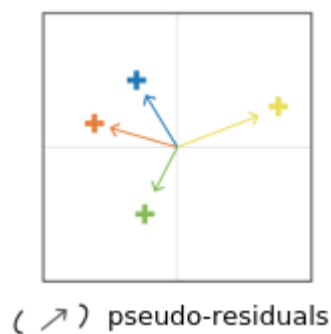


Figure 2.14: Pseudo-residual intuition (adapted from <sup>11</sup>).

Multiple Gradient Boosting characteristics contribute to its success. It involves four elements:

- an arbitrary differentiable loss function to be minimized depending on the sort of problem being tackled (regression problems, e.g., squared error);
- Regression Decision Stump as the weak learners that later evolve into larger Trees with 4-to-8 levels. They are greedily constructed, starting from the root, by taking the best split points according to the minimization loss or the purity rates (e.g., Gini index) [64];
- an additive model to append base models. The aforementioned model to minimize the residual loss employs a GD optimization procedure [78] when appending the greedily constructed Trees;
- shrinkage, also known as *Learning Rate* (LR), “reduces the influence of each individual tree and leaves space for future trees to improve the model” [84]. “Each update is simply scaled

<sup>11</sup><https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205>, last accessed on 2021-01-21

by the value of the “learning rate parameter  $\nu$ ”. ... Decreasing the value of  $\nu$  increases the best value for  $M$  [meaning the number of trees]. ... The  $\nu - M$  trade-off is clearly evident; smaller values of  $\nu$  give rise to larger optimal  $M$ -values. ... These results are fairly universal” [83].

---

**Algorithm 4** Gradient Boosting
 

---

**Input:** number of weak learners  $M$ , training sample  $S$ , test sample  $T$

**Output:** final prediction  $P$

```

1: initialize pseudo-residuals to the observation values
2: for  $m = 1$  to  $M$  do
3:   fit best possible weak learners by gradient opposite approximation on  $S$ 
4:   compute current weak learner optimal shrinkage  $s$ 
5:   compute new pseudo-residuals
6: end for
7:  $P = \sum_{m=1}^M weight_{base\_learner\_m} \times pred(T)_{base\_learner\_m}$ 

```

---

However, it may still be computationally expensive and prolonged to train a model, exasperated by massive datasets. Additionally, utilizing a naive Gradient Boosting procedure (greedy algorithm) can quickly overfit. Therefore, it might be helpful to employ regularization methods (section A.5.4). These penalize multiple algorithm sections and usually improve its performance. Also, bootstrap aggregation, Tree constraints (depth and number), and weighted updates (LR [83] [84]) are viable options.

### 2.5.2.3 Penalized Gradient Boosting

It is a Gradient Boosting variation that adds further constraints on Parameterized Trees. It does not use CART. Instead, it leverages a modified Regression Tree version with numeric values as the leaf node's constituents, allowing regularization functions (e.g., L1 and L2 (section A.5.4.4)).

The added “regularization term helps to smooth the final [learned] weights to avoid over-fitting. Intuitively, the regularized objective will tend to select a model employing simple and predictive functions” [85].

### 2.5.2.4 Stochastic Gradient Boosting

It is a Gradient Boosting variation that adopts a Stochastic Gradient Descent (SGD) method (section A.3.2.2).

A few alternatives to Stochastic Boosting can be employed. Before generating each Tree or, one may subsample rows or columns. Moreover, before contemplating doing a split, one may, even further, subsample columns. Normally, aggressively sub-sampling (e.g., selecting only half of the data) is positive in decreasing Tree mutual correlation. According to “user feedback, using column sub-sampling prevents over-fitting even more so than the traditional row sub-sampling” [84].

### 2.5.2.5 XGBoost

Also known as *Extreme Gradient Boosting*, it is an **Estimator-specific** ensemble library for Decision/Regression Trees and a Penalized Gradient Boosting/Stochastic Gradient Boosting variation that provides a very efficient/effective implementation. Typically expected to be significantly faster than other native approaches [85], it addresses the speed and performance problems of GBM by introducing many techniques that dramatically accelerate model training.

Compared to the GBM, XGBoost “refers to the engineering goal to push the limit of computations resources for Boosted Tree algorithms” [86].

Therefore, the three primary reasons to use XGBoost are superior model performance, memory occupation, and execution speed [87]. Having such qualities allows it to dominate regression predictive modeling problems on tabular datasets (appendix C) [85]. “Among the 29 challenge winning solutions, 3 published [on] Kaggle’s blog during 2015, 17 solutions used XGBoost. ... The success of the system was also witnessed in KDDCup 2015<sup>12</sup>, where XGBoost was used by every winning team in the top-10” [85].

Multiple XGBoost characteristics contribute to its success [85]:

- it harnesses regularization, in contrast to the standard Boosting implementations, to reduce overfitting (depth and values in leaves). Precisely, it predicts a loss function with an added regularization term;
- parallel and independent processing;
- it has high flexibility, which allows the definition of custom optimization objectives and evaluation criteria;
- it handles missing data;
- Tree pruning consists of splitting up to *max\_depth* and later backward prune the Tree by removing no positive gain splits;
- different training set bootstrap samples used to fit each Decision Tree;
- a random subset of input variables in each Tree split point is taken, ensuring each ensemble added Tree is skillful and more broadly diverse.

XGBoost may also be used with Random Forest in a level-wise approach (Fig. 2.15).

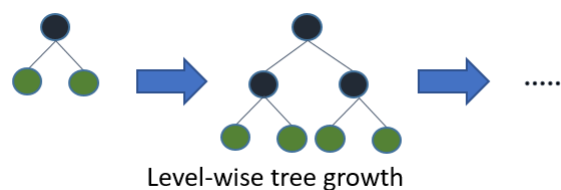


Figure 2.15: Level-Wise approach<sup>13</sup>

<sup>12</sup><https://www.kdd.org/kdd-cup>



It repurposes and harnesses the computational efficiencies implemented in the library for training Random Forest models [85]. Given that it derives from the Random Forest algorithm, it is necessary to define some parameters, specifically, the number of input features and weak learners to consider.

### 2.5.2.6 CatBoost

Also known as *Category Gradient Boosting*, developed at Yandex <sup>14</sup>, it is an AdaBoost variation and an **Estimator-specific** ensemble library for Decision/Regression Trees which provides a very efficient/effective Gradient Boosting implementation [88].

It offers several positives and features [88], such as:

- good quality with default parameters - less time spent on parameter tuning;
- categorical feature support - less time spent on data pre-processing;
- fast prediction;
- quick and scalable GPU version;
- enhanced accuracy - diminished model overfitting through novel Gradient Boosting scheme.

### 2.5.2.7 LGBM

Also known as *Light Gradient Boosted Machine* or , *LightGBM*, developed at Microsoft <sup>15</sup>, it is an **Estimator-specific** ensemble library for Decision/Regression Trees which provides a very efficient/effective Gradient Boosting implementation. It consists of Gradient Boosting Decision Trees (GBDT) with GOSS or DART and EFB. This results in a training algorithm with improved predictive performance and dramatically faster by up to 20x [89].

The training algorithm may be accelerated depending on the way the various Decision Trees are generated. Namely, the number of features (columns) and the number of examples (rows) in the train set, since split points for each feature and value have to be analyzed throughout development. “If we can reduce #data or #features, we will be able to substantially speed up the training of GBDT” [89].

Gradient-based One-Side Sampling (GOSS) is a modification to the Gradient Boosting method [89]. It focuses attention on training examples resulting in a larger gradient while excluding the remaining, because the first play a relevant part in the information gain computation process [89]. The model is then updated with those instances, and the rest are dropped. “GOSS can obtain [a] quite accurate estimation of the information gain with a much smaller data size” [89]. While other algorithms work in a level-wise approach pattern, GOSS works in a leaf-wise approach (Fig. 2.16). This speeds up learning and reduces the method’s computational complexity.

<sup>13</sup><https://medium.com/analytics-vidhya/gradient-boosting-lightgbm-xgboost-and-catboost-kaggle-challenge-santander-f3cf0cc56898>, last accessed on 2021-01-21

<sup>14</sup><https://yandex.com/dev/catboost/index/?turbo=true>

<sup>15</sup><https://www.microsoft.com/en-us/research/project/lightgbm/>

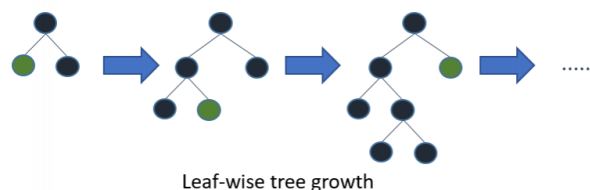


Figure 2.16: Leaf-Wise approach (adapted from <sup>16</sup>).

Exclusive Feature Bundling (EFB) is an approach for bundling sparse mutually exclusive features that infrequently use non-zero values concurrently to lessen the overall features number [89]. This resembles a type of automatic feature selection.

Multiple Additive Regression Trees (MART) is a GBDT precursor [89].

Dropouts meet Multiple Additive Regression Trees (DART) brings together the concept of NN's Dropout (section A.5.4.3) [90].

### 2.5.2.8 Histogram Based Gradient Boosting

It is an **Estimator-specific** ensemble for Regression Trees and an alternative method to implement Gradient Tree Boosting, sparked by LightGBM, which uses efficient data structures to optimize the building procedure.

“Instead of finding the split points on the sorted feature values, histogram-based algorithm buckets continuous feature values into discrete bins and uses these bins to construct feature histograms during training. Since the histogram-based algorithm is more efficient in both memory consumption and training speed, we will develop our work on its basis” [89].

### 2.5.3 Bagging vs Boosting

The base models' selection needs to be consistent with the way to aggregate them. Given that they have low bias and high variance, an aggregating scheme favoring reducing the ensemble's variance (e.g., Bagging) should be considered. On the other hand, if they have low variance and high bias, an aggregating scheme favoring reducing the ensemble's bias (e.g., Boosting) should be considered. Tables 2.4 and 2.5 compares Bagging and Boosting's attributes and characteristics.

<b>Bagging</b>	<ol style="list-style-type: none"> <li>1. Weights - equal</li> <li>2. Training set - independent and random</li> <li>3. Improves variance - “variance reduction” algorithm</li> <li>4. Parallel - intensive parallelization techniques can be used if required</li> <li>5. Overfitting - appears to be somewhat immune to training dataset overfitting given the learning algorithm's stochastic nature</li> <li>6. Stability - can improve models' performance</li> </ol>
----------------	--

Table 2.4: Comparison of Bagging and Boosting algorithms' characteristics - 1.

<sup>16</sup><https://medium.com/analytics-vidhya/gradient-boosting-lightgbm-xgboost-and-cat-boost-kaggle-challenge-santander-f3cf0cc56898>, last accessed on 2021-01-21

<b>Boosting</b>	<ol style="list-style-type: none"> <li>1. Weights - exist</li> <li>2. Training Set - conditioned on the prior step's results</li> <li>3. Improves bias - "bias reduction" algorithm</li> <li>4. Parallelization - not possible (sequential)</li> <li>5. Overfitting - may increase</li> <li>6. Stability - when effective is better than Bagging; when not can accelerate performance deterioration</li> </ol>
-----------------	--

Table 2.5: Comparison of Bagging and Boosting algorithms' characteristics - 2.

### 2.5.4 Stacking

Ensembles are usually static structures. This means the ensemble members' combining mechanism is input independent, hence, only dependant on the members' outputs. Given multiple ML models that are skillful on a problem, but in different ways, how to choose the most reliable models? Stipulating optimal models' weights manually or, indeed, heuristically is a complex process. One possible solution is to use further radical methods that learn how to best weight sub-model predictions [41]. The aim is to improve both bias and variance by discovering the optimal route to combine base learners [91].

Also known as *Model Stacking* or *Stacked Generalization*, it is a non-linear Meta-model combiner algorithm that evolved from the Weighted Average ensemble (Alg. 5) [43]. It replaces the linear weighted sum model used to combine the base learners' out-of-fold predictions made during K-fold Cross-validation [42] [64].

---

#### Algorithm 5 Stacking

---

**Input:** training data  $S$ , test sample  $T$ , number of base learners  $M$

**Output:** final prediction  $P$

- 1: Step 1: learn base-level base learners
  - 2: **for**  $m = 1$  to  $M$  **do**
  - 3:      $s_m$  = sample from  $S$  according to the distribution
  - 4:     fit a base learner  $h_m(x)$  on  $s_m$
  - 5: **end for**
  - 6: Step 2: construct new dataset of predictions
  - 7: **for**  $i = 1$  to  $\#S$  **do**
  - 8:      $S_h = x'_i, y_i$ , where  $x'_i = h_1(x_i), \dots, h_M(x_i)$
  - 9: **end for**
  - 10: Step 3: learn the Meta-model  $H$
  - 11:  $P = \sum_{m=1}^M \text{weight}_{\text{base\_learner\_}m} \times \text{pred}(T)_{\text{base\_learner\_}m}$
- 

Stacking "works by deducing the biases of the generalizers with respect to a provided learning set. This deduction proceeds by generalizing in a second space whose inputs are (for example) the guesses of the original generalizers when taught with part of the learning set and trying to guess the rest of it, and whose output is (for example) the correct guess" [43].

Looking at the algorithm from another perspective, it may be viewed as having two layers or levels. The first level, level 0, contains  $N$  heterogeneous (section 2.2.2), independent base learners [92] [43]. These are trained in parallel, independently, and deduce their biases concerning the provided learning set [43]. The second level, level 1, attempts to find the optimal combination of the first level models by training the combiner Meta-model algorithm on top of them [92] [43]. It takes as inputs the previous layers' models' predictions attempting to generalize in a secondary space [43]. The output is the final ensemble prediction (Fig. 2.17). The more each base learner has to say, the better it is, and the more it will contribute to the final Stacking result [43] [72].

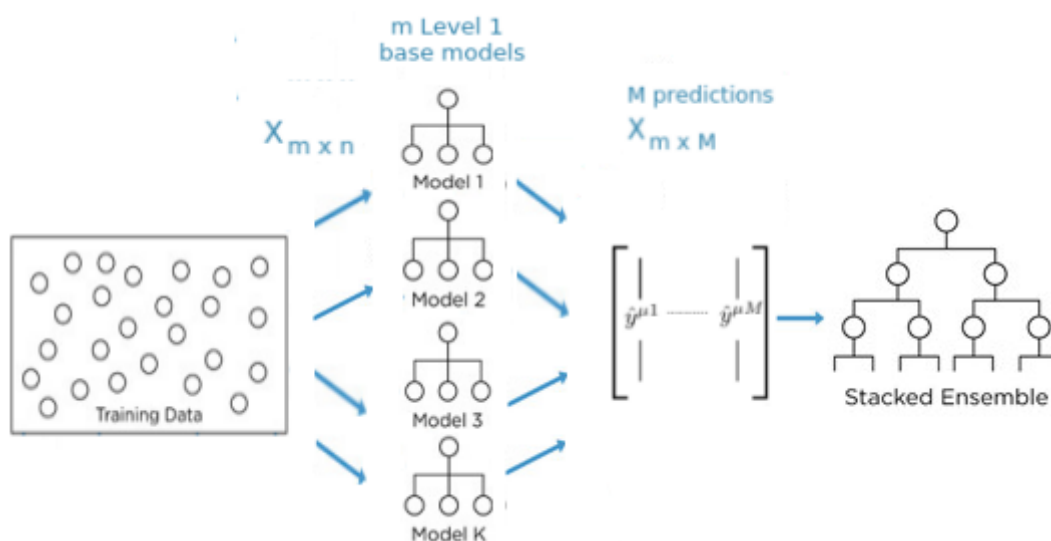


Figure 2.17: Stacking (adapted from <sup>17</sup>).

To overcome the limitation that the training data has to be split into two folds, **K-fold Cross-training** is employed 2.3.2.

On the other hand, no boundary is imposed on the amount of Stacking levels possible to implement [93]. In this case, the procedure can be referred to as **Multi-level Stacking**. For each different level Meta-model, a learning algorithm needs to be defined.

For a 3-level Stacking, the first layer fits  $N$  base learners on a dataset. Rather than fitting a sole Meta-model on the learners' predictions, the second layer fits  $M$  Meta-models trained to make predictions according to the previous layer's predictions. The third level fits one final Meta-model, taking the  $M$  Meta-models' predictions from the second layer as inputs and outputs predictions based on those (Fig. 2.18). "When used with multiple generalizers, stacked generalization can be seen as a more sophisticated version of cross-validation, exploiting a strategy more sophisticated than cross validation's crude winner-takes-all for combining the individual generalizers" [43].

Despite being possible to continue adding layers to the model, it may prove to be data expensive (if K-folds similar techniques are not employed) and time-consuming (if K-folds similar techniques are employed) without necessarily guaranteeing better results [43].

<sup>17</sup><https://www.kdnuggets.com/2019/09/ensemble-learning.html>, last accessed on 2021-01-21

Interestingly, notwithstanding Stacking being portrayed as a method with more than one level 0 base model, it is possible to use it when there is just one level 0 model. In which circumstance, the model in level 1 “is a scheme for estimating (and then correcting for) the error of a generalizer” [43].

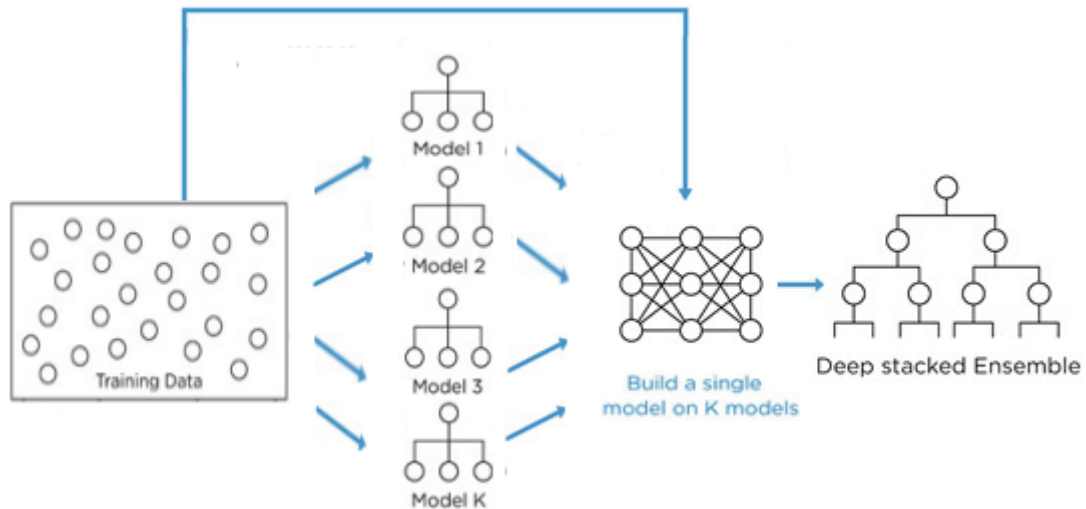


Figure 2.18: Deep Stacking (adapted from <sup>18</sup>).

Optionally, in addition to the previous layers' models' predictions, the base models' inputs may also be added to the level 1 model, which some refer to as **Deep Stacking** [94]. It “allows the model to select the right sub-model weights based on specific input variables ... to boost performance even further” [94]. This is done to uncover more rooted relationships linking sub-models and offer higher accuracy than an ensemble of simple linear “stacked” models.

Stacking's performance is dependent on the specifics of the Meta-model techniques (section 2.2.3).

#### 2.5.4.1 Blending

It is a Stacking variation. Although both terms are often seen used interchangeably [95], they represent different concepts. Blending has explicit rules for creating a Stacking model [96]. Rather than having the Meta-model fitted on out-of-fold base model predictions, it is fitted on base model predictions made on a small hold-out set (e.g., 10% of the training set).

Blending's performance is dependent on the specifics of the Meta-model techniques (section 2.2.3).

Comparing to Stacking might be more advantageous to use considering less data is used overall, and it wards against information leaks because the base models and the stacker both use different data.

<sup>18</sup><https://www.kdnuggets.com/2019/09/ensemble-learning.html>, last accessed on 2021-01-21

### 2.5.4.2 Super Learner

Also known as *Cross-validation Ensemble*, it is a Stacking-specific configuration to K-fold Cross-validation [10] [97] [98]. In this case, every estimator uses the equivalent K-fold data splits, and the Meta-model is fitted on each estimator's out-of-fold predictions to learn how to combine them best [99] [100]. More specifically, every level 0 base learner is trained with different  $k - 1$  folds and tested on the remaining  $k$  fold to obtain predictions used to train the Meta-model (Fig. 2.19).

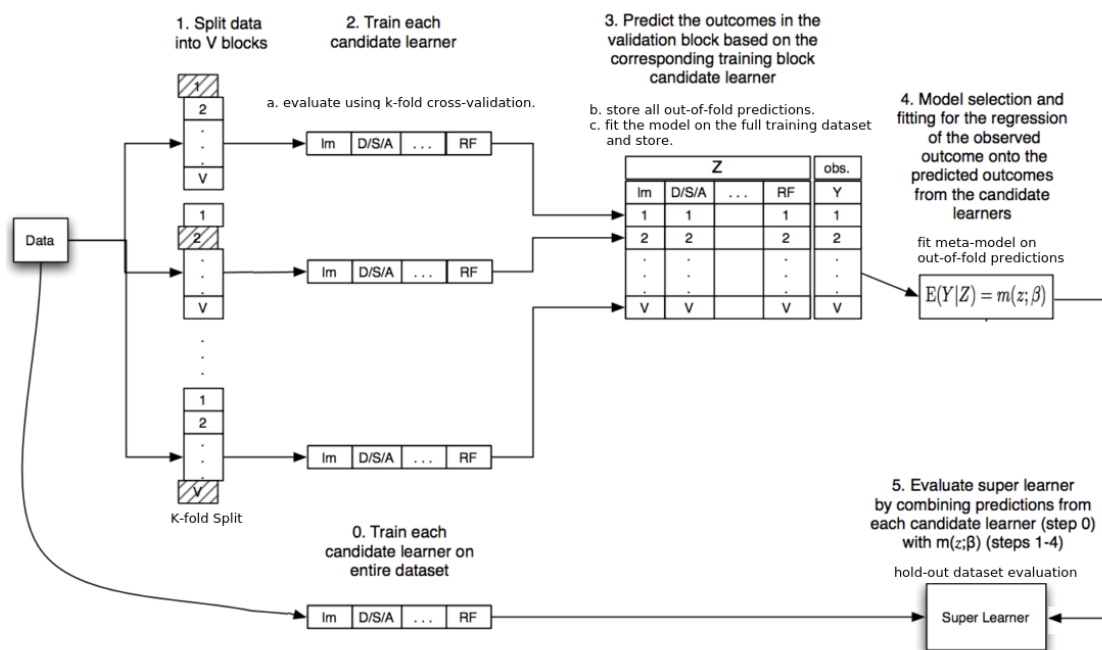


Figure 2.19: Super Learner (adapted from [10]).

Basically, it employs cross-validation to decide the base learners' optimal weights.

This model's results are, almost guaranteed, better than the best-performing sole estimator, with the possibility of performing better than any single model.

## 2.6 Prediction Integration Mechanism

There are multiple approaches to combine ensemble members' outcomes [9]. Given that this thesis revolves around regression, classification integration mechanisms will be left aside.

### 2.6.1 Average

Also known as *Committee of Networks*, *Simple Average*, or *Raw Average*, it is a non-trainable integration mechanism where each models' predictions equally contribute (equally distributed weights) to the final ensemble prediction (Eqn. 2.4) [12] [69] [101] [53] [7].

$$\tilde{y}(x) = \sum_{n=1}^N y_n(x)/N \quad (2.4)$$

where  $\tilde{y}(x)$  is the Average ensemble prediction,  $N$  is the number of ensemble members,  $y_n$  is an ensemble member, and  $x$  is the input observation

Starts by generating  $N$  independent models with possibly different hyperparameters (e.g., initial synaptic weight values are customarily taken from a distribution according to domain knowledge or at random). After that, each model is trained and its predictions combined at test time using Simple Average (regression modeling problems).

“The reason model averaging works is that different models will usually not make all the same errors on the test set” [8]. “Owing to its simplicity and effectiveness, simple averaging is among the most popularly used methods and represents the first choice in many real applications” [9].

It is sensible if the base learners’ performance is comparable or their predictions’ distributions are (nearly) Gaussian [102] [103] [104] [72].

The main disadvantages are threefold:

- the average operation may cause a loss of information;
- for small ensembles, the sample of predictions may not be sufficiently large for the mode to be meaningful;
- each model equally contributes to the final prediction, despite how expertly it actually behaved.

Also, some specific situations have been pointed out:

- if the ensemble includes heterogeneous base learners, then Simple Average may provide suboptimal results. The reason is that it is affected by the weak learners’ and the overconfident learners’ performances;
- it has been stated that Simple Average is better than Weighted Average as the latter promotes overfitting [105];
- Simple Average should be used for large ensembles since the global optimal generalization error may be attained by individual models training set size optimization [40].

### 2.6.2 Weighted Average

It is a trainable integration mechanism often denoted as an extension of the Average ensemble, aiming to tackle its inefficiencies [101].

The idea is that some estimators are expected to be more precise than others, and these should be awarded a higher contribution share, whereas less skillful models should be awarded a lesser

contribution share (Eqn. 2.5) [45] [9]. In practice, it uses the sub-models expect performance to weigh each of their contributions for an improved combined ensemble prediction [106]. “The weight of each [estimator] can be set proportional to its accuracy performance on a validation set” [41]. Despite not performing so well, some estimators may still be useful, so they are kept around [53]. In fact, some estimators may have poor overall performance but good results in certain input subspaces, causing better overall performance.

Some base learners “will typically make better predictions than other members . . . expect to be able to reduce the error still further if we give greater weight to some committee members than to others . . . weighted combination of the predictions of the members” [6]. Concerning “smaller ensembles, optimization of the ensemble weights can yield significant improvements in ensemble generalization performance” [40].

$$\tilde{y}(x; \alpha) = \sum_{n=1}^N \alpha_n \cdot y_n(x) / \text{sum}(\alpha) \quad (2.5)$$

where  $\tilde{y}(x; \alpha)$  is the Weighted Average ensemble prediction,  $N$  is the number of ensemble members,  $y_n$  is an ensemble member,  $\alpha$  represents all base models’ weights,  $\alpha_n$  is the ensemble member’s weight, and  $x$  is the input observation.

The weights are small positive values that sum up to a value that may be different from 1, calculated by extracting the ensemble model’s, hopefully, optimal weights. Each one reflects a base learner’s final predictive contribution or “trust” percentage [40]. They are neither fixed nor previously imposed and, in truth, can be optimized during the training process. To add more weight to a given model without calculating explicit weight coefficients, one can add a specific model more than once to the ensemble.

Optimizing weights for smaller ensembles can bear considerably more solid generalization performance compared to an optimal individual estimator employing all data [40]. Additionally, to find the optimal weights, one can perform a weight grid search, use an optimization algorithm on a holdout dataset, a linear model, or another ML model [45].

The final prediction is the estimators’ integrated predictions, where each one’s synaptic weights are applied to each estimators’ output [6].

The Average ensemble described in section 2.6.1 is a special case of the Weighted Average ensemble when all  $\alpha$  from Eqn. 2.5 are equal to  $1/M$ , with  $M$  being the ensemble size.

### 2.6.3 Meta-model

Refer to section 2.2.3.



# Chapter 3

## Neural Networks Ensembles

### Contents

---

<b>3.1</b>	<b>Varying Training Data . . . . .</b>	<b>35</b>
<b>3.2</b>	<b>Varying Base Models' Characteristics . . . . .</b>	<b>36</b>
3.2.1	Implicit Ensembles . . . . .	36
3.2.2	Explicit Ensembles . . . . .	37
3.2.3	Implicit and Explicit Ensembles Overview . . . . .	41
<b>3.3</b>	<b>Varying Base Models' Generation Strategy . . . . .</b>	<b>41</b>
3.3.1	Negative Correlation Learning . . . . .	41
3.3.2	Boosting Neural Network Ensemble . . . . .	46
3.3.3	Combining strategies for ensemble generation . . . . .	46
<b>3.4</b>	<b>Applications . . . . .</b>	<b>47</b>

---

*Some EL techniques are estimator-specific (only work with particular base model algorithms), while others are not. This section explores Neural-Network specific EL concepts and methods.*

NNEs gained attention as soon as they were put forward [46]. For that reason, since 1990, many researchers have focused on the aforementioned domain by putting forward multiple papers proposing new methods for NNEs. At 1993's Neural Information Processing Conference, a whole discussion, particularly for NNEs (Fig. 3.1), entitled "put it together" was taken.

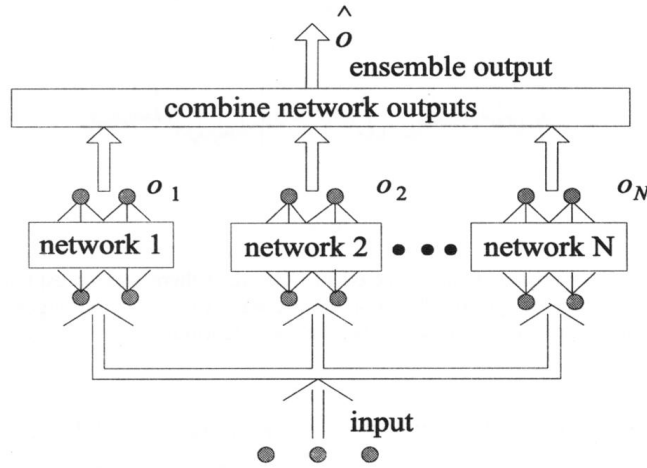


Figure 3.1: Basic Ensemble architecture of NNs [107].

“Most Neural Network algorithms achieve sub-optimal performance specifically due to the existence of an overwhelming number of sub-optimal local minima” [101]. A NNE combines a limited number of NNs [40], and it is, in practice, one reliable approach to resolve NN’s variance and drastically improve predictions and generalization capability [46] [108]. Moreover, it can predict unknown samples with high adaptability [46].

Diversity means they have reached different local minima, with a different set of final weights, thus making distinct prediction errors [53]. It is proved that if the ensemble’s NNs are independent of one another, then as the amount of combined Networks tends to infinity, it is possible to reduce the expected error to zero. However, such assumptions rarely hold a place in practice since independence cannot be wholly guaranteed [46].

Provided input  $x \in R^m$  follows distribution  $p(x)$ , if the output corresponding to  $x$  is  $d(x)$ , the output corresponding to individual NN  $f_i (i = 1, 2, \dots, N)$  is  $f_i(x)$ , the weigh corresponding to  $f_i(x)$  is  $\omega_i$ , so the NNE’s output  $f_{ensemble}(x)$  corresponding to  $x$  is equal to Eqn. 3.1.

$$f_{ensemble}(x) = \sum_{i=1}^N \omega_i f_i(x) \quad (3.1)$$

and, the NNE’s generalization error is given by Eqn. 3.2.

$$E_{ensemble} = \int P(x) (f_{ensemble}(x) - d(x))^2 dx \quad (3.2)$$

Through theoretical analysis of NNE’s computational formula [40], Eqn. 3.3 analyzes the relation between the ensemble’s generalization error and diversity.

$$E_{ensemble} = E_{average} - A_{ensemble} \quad (3.3)$$

where  $E_{ensemble}$  is the ensemble's generalization error,  $E_{average}$  is the weighted average of the individual NN's generalization error, and  $A_{ensemble}$  is the ensemble's diversity.

Eqn. 3.4 provides an inference on Eqn. 3.3.

$$E_{ensemble} \leq E_{average} \quad (3.4)$$

Moreover, increasing  $A_{ensemble}$  shows it is effectively conceivable to decrease the NNE's generalization error.

However, the NNE algorithm is conditioned on both NN's and Ensemble's disadvantages (section A.6 and 2.2.6).

In order to compare NNEs, it is useful to define a comparison threshold. More specifically, this could be a baseline method, a single NN with the same architecture as ensemble's NNs. A more audacious option would be to confront the NNE's performance with a sole NN whose weights represent the ensemble base model's total average weight amount. More specifically, authenticate if averaging an abundance of shallow models performs stabler than an equivalent single deep NN (section A.7).

At present, reviews on NNEs are uncommon. Relevant ones include an introduction to ensemble models [109], a survey on the experts' mixture [110], a survey on modern trends in single and multiple ensemble models [111], and a survey on ensemble approaches for regression [112].

As a side note, Appendix A presents a profound explanation of the NNs' workings and how they can be optimized.

NNE introduces multiple specific steps on top of those provided by general ensembles (section 2.3, 2.4, and 2.5) to tackle the various disadvantages of NNs (section A.6): **training data, base models' characteristics, and base models' generation strategy.**

### 3.1 Varying Training Data

It is possible to explore various training techniques (section 2.3) to promote mutual model diversity. K-fold Cross-training (section 2.3.2) and Bootstrapping (section 2.3.4) can be used in small Networks. However, it may take longer with deeper Networks.

It is also possible to incorporate other algorithms' characteristics in the NN's training procedure. In addition to training the NNs with different data points (rows in a dataset) (section 2.3.3 and 2.3.1), one could train the NNs with different features (columns in the dataset), as seen in the Random Subspace 2.3.5, Random Patches 2.3.6, and Random Forest's Random Input Feature Selection (section 2.5.1.1). The objective is to avoid overfitting particular features and provide an additional performance boost. It was found that an ensemble of identical NNs fitted on distinct

feature subsets is slightly more fruitful than an ensemble of distinct NNs fitted on the equivalent subset [113].

One study from 2007 reviewed the “relationship between the ensemble and its component Neural Networks ... with the goal of creating a set of nets for an ensemble with the use of a sampling technique. ... [Each] net in the ensemble is trained on a different sub-sample of the training data” [114].

## 3.2 Varying Base Models’ Characteristics

Refers to varying the models’ characteristics (section 2.4) to get more considerable mutual diversity. Particularly, differences in Network type, topology [115] [116], “random initialization, random selection of mini-batches, hyperparameters, or outcomes of non-deterministic implementations of Neural Networks are enough to cause different members of the ensemble to make partially independent errors” [8]. Changing training specificities (e.g., regularization or LR) and properties (e.g., number of layers in NN, learning algorithm, initial weights, or loss function) [117] [46] also are enough to promote diversity (section A.5).

### 3.2.1 Implicit Ensembles

Implicit ensembles train a single model, which behaves like an ensemble of multiple models. Also, the model parameters (weights) are shared, and the original Network approximates the ensemble models’ averaging.

The advantage is to keep the ensemble’s training time equal to that of a single model by maintaining the additional cost as low as possible while also promoting diversity.

#### 3.2.1.1 Neural Network Weight Sharing

In this case, “each individual model in ... [the] ensemble layer corresponds to weights in the ensemble layer optimized in different directions. ... By adopting [a] weight sharing approach, the results show ... [it] can notably improve the accuracy and stability of the original Neural Networks with ignorable extra time and space overhead” [118].

#### 3.2.1.2 Dropout

Despite not being an Ensemble-specific strategy, it can be considered an implicit ensemble variation that allows for overfitting reduction and, more importantly, intuitively, simulates having a large number of slightly different Network architectures (section A.5.4.3). This promotes diversity during the ensemble’s NN estimators learning process and leads to better generalization capacity by randomly dropping a given percentage of Nodes.

Table 3.1 presents a couple of ensemble researches where Dropout is applied.

Paper	Summary
Dropout and DropConnect based Ensemble of Random Vector Functional Link Neural Network (2019)	“Random Vector Functional Link (RVFL) Neural Network with closed form solution is a randomized Neural Network suitable to use as [estimators] of the ensemble because of its extremely fast training time, good generalization and innate randomization in its architecture. . . . This study proposes an ensemble of RVFL Neural Networks by introducing additional regularization / randomization . . . via . . . Dropout and DropConnect. . . . Experiments . . . [show the] ensemble performs better than other RVFL based ensembles” [119].
Uncertainty-aware Deep Learning Forecast using Dropout-based Ensemble Method (2019)	A “probabilistic forecast technique . . . [which] focuses on deep learning . . . achieves generality by using dropout. . . . [Implementation] shows how the dropout technique is utilized to realize an ensemble method” [120].

Table 3.1: Overview of Dropout-related researches.

### 3.2.2 Explicit Ensembles

Like implicit ensembles, explicit ensembles train a single model, which behaves like an ensemble of multiple estimators. Still, explicit ensemble estimator parameters' (weights) are not shared, and the output is the ensemble model's prediction integration.

#### 3.2.2.1 Horizontal Integration

It builds a sequence of multiple models from contiguous training epochs and integrates their predictions with the Simple Average mechanism (sec. 2.6.1) [11]. In practice, it averages the Network's state over the last several iterations.

Each subsequent model can be saved during the referred procedure, and a subset of the best learners may be taken for the ensemble construction by analyzing their learning curves.

“Slightly inferiorly trained networks are a free by-product of most tuning algorithms; it is desirable to use such extra copies even when their performance is significantly worse than the best performance found. Better performance yet can be achieved through careful planning for an ensemble prediction by using the best available parameters and training different copies on different subsets of the available database” [46].

This “smoothed” weights version over the last steps can be seen, intuitively, as a bowl-shaped objective with the Network bouncing near the mode so that the average has a greater probability of occurring close to it somewhere.

It is a cheap way of obtaining an extra 1%-2% performance. However, Horizontal Integration suffers from a lack of estimator variety.

### 3.2.2.2 Snapshot

It provides an ensemble with accurate and diverse learners, which can be collected from a single training run, not necessarily from contiguous training epochs, with each one providing predictions without incurring additional training costs [46] [121].

It “is an optimization process [that] visits multiple local minima before converging to a final solution. . . . [It] takes model snapshots at these various minima and averages their predictions” [121]. Pointedly, it “lowers the learning rate at a very fast pace, encouraging the model to converge towards its first local minimum. . . . [By updating] the learning rate at each iteration rather than at every epoch [it] improves the convergence of short cycles, even when a large initial learning rate is used” [121].

A major benefit is that finding “much flatter solutions [than] a single model . . . leads to better generalization than conventional training . . . achieves notable improvement in test accuracy . . . improves generalization” [122].

Normally used when a model demands ample computational resources in its training stage, namely, weeks or months because it is composed of very deep models or huge datasets. It is “extremely easy to implement . . . [and] has almost no computational overhead” [122].

### 3.2.2.3 Stochastic Gradient Descent with Warm Restarts

Also known as *SGDR*, it is a Snapshot extension that takes a step forward in promoting model diversity and providing faster learning in an SGD environment (section A.3.2.2).

SGDR falls under the umbrella of **Aggressive LR schedules**. Starting with a generous LR value, it implements a simulated learning process restart policy that aggressively, swiftly, and systematically resets it to a near-zero value before suddenly increasing it anew to the maximum [123]. In other words, it forces the optimization procedure to cycle during a sole training run [124].

“Converge  $M$  times to [multiple] local minima along its optimization path, . . . then restart the optimization with a large learning rate to escape the current local minimum . . . [and] attempt to find another possibly better local minima” (Fig. 3.2) [121].

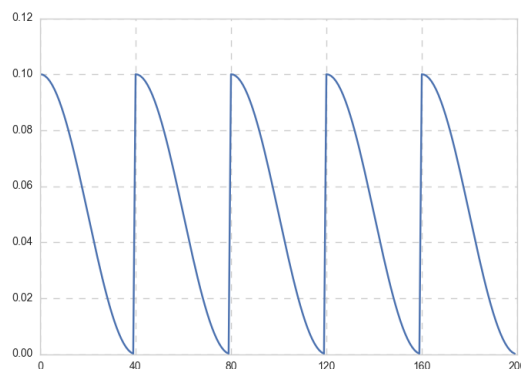


Figure 3.2: Cosine Annealing Cycles. X-axis denotes the LR and the y-axis denotes the epochs (adapted from <sup>19</sup>)

Eqn. 3.5 represents the **Cosine Annealing LR Schedule**.

$$\alpha(t) = \frac{\alpha_0}{2} \left( \cos\left(\frac{\pi \cdot \text{mod}(t-1, \lceil T/M \rceil)}{\lceil T/M \rceil}\right) + 1 \right) \quad (3.5)$$

where  $\alpha(t)$  is the LR at the epoch  $t$ ,  $\alpha_0$  is the max LR,  $M$  is the amount of cycles,  $T$  is the total amount of epochs,  $\text{mod}$  is the module function,  $\pi$  is the pi numeric value,  $\text{cos}$  is the cosine function, and  $\lceil \cdot \rceil$  is the square brackets is the floor function.

Fig. 3.3 compares the commonly used LR schedule with Snapshot's LR schedule.

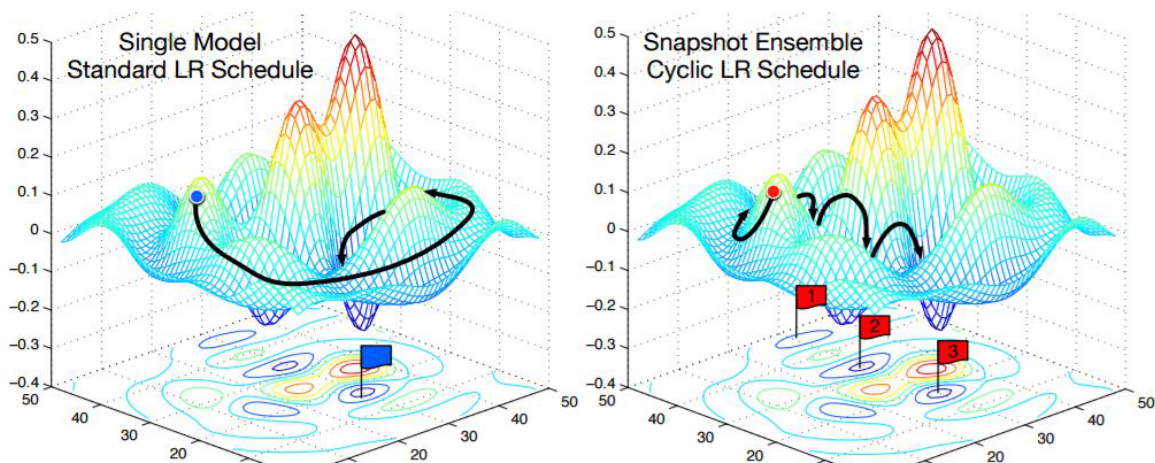


Figure 3.3: Comparison of Standard LR and Cyclic LR Schedules. The left image follows a constant LR optimization path towards a single local minima (blue color). The right image aggressively varies the LR forcing irregular LR optimization paths to visit multiple local minima to more successfully find the global minima (orange color) and induce diversity<sup>20</sup>.

This subjects the model weights to dramatic changes converging to very different values with different respective Network performances each, thus “practically eliminating the need to experimentally find the best values and schedule for the global learning rates” [125]. The best-performing weights can be saved as “checkpoints” for the ensemble.

Obtained “good results with SGDR, mainly by using ... ensembles of snapshots from [the] SGDR’s trajectory” [121]. Also, “Warm restarts ... [schedule] the learning rate to achieve competitive results ... roughly two to four times faster” [124].

Alternatively, it may inject an oscillating amount of noise through multiple epochs to avoid getting stuck in the same local minima.

<sup>19</sup><https://medium.com/udacity-pytorch-challengers/ideas-on-how-to-fine-tune-a-pre-trained-model-in-pytorch-184c47185a20>, last accessed on 2021-01-21

<sup>20</sup><https://towardsdatascience.com/advanced-topics-in-neural-networks-f27fbcc638ae>, last accessed on 2021-01-21

Another aspect is that it also provides re-use of small random “good weights” that work as initiating points in the following LR rounds.

### 3.2.2.4 Polyak averaging

Also known as *Polyak-Ruppert Averaging* or *Temporal Averaging*, it “has been shown to improve the convergence of standard SGD” [126].

It aims to save multiple sets of noisy weights from various models seen close to the training phase’s end and then average those into a single NN (Fig. 3.4) [127] [128]. This translates into finding the best-performing set of weights [129]. Also, attempts to calm down the noisy optimization process due to hyperparameters choice (e.g., LR) [8]. In other words, it aims to average an abundance of shallow models so that the single NN performs stabler.

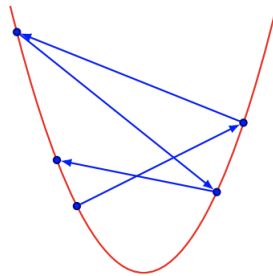


Figure 3.4: Polyak Averaging (adapted from <sup>21</sup>).

Eqn. 3.6 represents **Polyak Averaging**.

$$\theta_n = \frac{1}{N} \sum_n^N \theta_n \quad (3.6)$$

where  $\theta_n$  is the weight at iteration  $n$ , and  $N$  is the number of iterations.

An optimization algorithm may “leap back and forth across a valley several times without ever visiting a point near [its] bottom” [8]. The fundamental breakthrough was reached on “the paradoxical idea [that] a slow algorithm having less than optimal convergence rate must be averaged” [127]. If one takes a Neural Network group that has “converged to local minima and [applies] averaging, [one] can construct an improved estimate. One way to understand this fact is to consider that, in general, networks [that] have fallen into different local minima will perform poorly in different regions of feature space, and thus their error terms will not be strongly correlated” [101]. Averaging of all locations “on either side should be close to the bottom of the valley” [8].

“Alternatively, an exponential moving average over the parameters can be used, giving higher weight to more recent parameter values” [126].

<sup>21</sup><https://towardsdatascience.com/advanced-topics-in-neural-networks-f27fbcc638ae>, last accessed on 2021-01-21



Going further, it is possible to define a Weighted Average and even a **Weighted Average with exponential decay**. This last approach linearly increases the models' weights the more recent they are.

### 3.2.3 Implicit and Explicit Ensembles Overview

Both implicit and explicit ensembles produce base learners from a sole NN at the cost of mutual diversity since low-level estimator features are expected to be similar [130]. To tackle the, somewhat, lack of diverseness, one may employ branching-based deep models [131] or SGWR with Cosine Annealing LRs [124] [121].

## 3.3 Varying Base Models' Generation Strategy

The following techniques refer to base model generation approaches (section 2.5). However, only some are compatible with NNs. These comprise Bagging (section 2.5.1) and Boosting (section 2.5.2), particularly AdaBoost (section 2.5.2.1). Furthermore, Stacking (section 2.5.4) and its variations, Blending (section 2.5.4.1) and Super Learner (section 2.5.4.2), are also applicable.

### 3.3.1 Negative Correlation Learning

Also known as *NCL*, it seems to have the highest potential as it has recently gained momentum and been improved upon with multiple studies.

The training process trains all the Networks synchronously, interactively<sup>22</sup>, and dependently<sup>23</sup>. Consequentially it makes each model locally coupled with the others.

More importantly, it minimizes the ensemble's empirical risk function by adjusting the NN's objective function (mean squared error (MSE)) with a penalty term, expressing the ensemble NNs' correlation [132]. In practice, a "bias" is attached to the error function. When generating a model for the ensemble, the added penalty term to the NN's objective function promotes a negative correlation (NC) between the new model and the previously generated models [133] [134]. Note, do not mistake NCL's bias with statistical bias.

Basically, it promotes interaction, mutual model diversity, and lower predictions' overall correlation [133] [135] [136] [137] [138]. Therefore, it can be considered a synchronous observation method.

Eqn. 3.7 symbolizes the NCL's error function and Eqn. 3.8 expresses the NCL's penalty term.

$$e_i = \frac{1}{2}(f_i - d)^2 + \lambda p_i \quad (3.7)$$

where  $e_i$  is the adjusted error function,  $f_i$  is the model's predicted

<sup>22</sup>Interactively means the NCL algorithm analyses the base learner's training.

<sup>23</sup>Dependently means the NCL algorithm mutually exchanges information across base learners.

output,  $d$  is the models' target output,  $\lambda$  is the neuron's bias coefficient, and  $p_i$  is the penalty term

$$p_i = (f_i - \bar{f}) \sum_{j \neq i} (f_j - \bar{f}) \quad (3.8)$$

where  $p_i$  is the penalty term,  $f_i$  is the model's predicted output, and  $\bar{f}$  is the ensemble's average output.

Adjusting the penalty term grants the possibility to tune the NNs' diversity. However, the detriment is that the  $\lambda$  variable, which is particularly relevant for the NNE's performance, has issues [139].

Since NCL is very much concerned with the NN's internal characteristics, particularly the objective training function, it may only exclusively be employed with NNs, hence, not applicable to other architectures.

Tables 3.2, 3.3, 3.4, 3.5, and 3.6 show multiple relevant selected papers. The first one acts as the precursor of the following researches.

Paper	Summary
Ensemble Learning Using Decorrelated Neural Networks (1996)	"[Decorrelates the] network training method for improving the quality of regression learning. ... Individual networks are trained by backpropagation ... to have their errors linearly decorrelated. ... Outputs are then linearly combined. ... Results show ... lower MSE" [133].
A cooperative ensemble learning system (1998)	A "new cooperative Ensemble Learning system (CELS)... encourages different individual networks ... to learn different parts... of the training data so ... the ensemble [learns] training data better. ... [This] tends to create negatively correlated networks ... trained simultaneously ... [with an] opportunity for [cooperation and specialization]. ... Experiments ... demonstrate ... CELS produces NNEs with good generalisation ability." [140].
Ensemble learning via negative correlation(1999)	"[NCL] attempts to train individual networks in an ensemble and combines them in the same learning process. ... All the individual networks in the ensemble are trained simultaneously and interactively through correlation penalty terms. ... [This] can create negatively correlated networks to encourage specialisation and cooperation. ... Results show ... [it] can produce Neural Network Ensembles with good generalisation ability" [134].
Simultaneous training of negatively correlated neural networks in an ensemble (1999)	"Cooperative Ensemble Learning system (CELS) ... encourages ... individual networks in an ensemble to learn different parts ... of a training data ... so ... the ensemble can learn the ... data better. ... Networks are trained simultaneously rather than independently or sequentially ... providing opportunity for ... interaction with each other and to specialize using a correlation penalty term in the error function. ... Results show that CELS can produce NNEs with good generalization ability" [136].

Table 3.2: Overview of Negative Correlation Learning related researches - 1.

Evolutionary ensembles with negative correlation learning (2000)	“Evolutionary ensembles with negative correlation learning (EENCL) ... addresses the issues of automatic determination of the number of individual Neural Networks (NNs) in an ensemble and the exploitation of the interaction between individual NN design and combination. ... EENCL encourages ... individual NNs ... to learn different parts ... of the training data so that the ensemble can learn better the entire training data. ... Cooperation and specialization among different individual NNs ... is considered during the individual NN design. ... [It also] provides an opportunity for different NNs to interact with each other and to specialize. Experiments ... demonstrate ... [it] can produce NN ensembles with good generalization ability” [141].
Negative correlation learning and the ambiguity family of ensemble methods (2003)	By “removing an assumption made in the original work, NC can be shown to be a derivative technique of the Ambiguity decomposition by Krogh and Vedelsby. ... From this formalisation ... (calculate parameter bounds), [it] shows significant improvements. ... Success lies in rescaling an estimate of ensemble covariance ... [showing] that during this rescaling, NC [term] varies smoothly between a single neural network and an ensemble system” [142].
Method of incremental construction of heterogeneous neural network ensemble with negative correlation(2004)	A “new method for [incrementally] constructing a heterogeneous neural network ensemble (HNNE) based on heterogeneous Neural Network with negative correlation. ... [The] proposed method adjusts both the ... architecture of the individual networks and the ... weights. ... Improves the accuracy of the member Neural Networks while increasing diversity ... and decreasing the generalization error of [the] ensemble. ... [It] consists of two parts ... [dynamically] constructing the best heterogeneous Neural Networks (BHNN) [based on negative learning] and constructing [the] HNNE. ... Results ... show the error rate can be improved by HNNE ... better than Boosting, Bagging and other network ensemble methods.” [143].
Fast Neural Network Ensemble Learning via negative-correlation data correction (2005)	A “new negative correlation (NC) learning method. ... [The] advantages are ... 1) [requiring] much less communication overhead than the standard NC method ... 2) ... applicable to ensembles of heterogeneous networks” [144].
Negatively Correlated Neural Network Ensemble with Multi-population Particle Swarm Optimization (2005)	“Multi-population particle swarm optimization (MPPSO) algorithm trains negatively correlated Neural Network Ensembles. ... Each sub-swarm is responsible for training a ... network ... [with its] architecture ... automatically configured ... and simultaneously / successively [exchanging] information among them. ... Performance of ensemble is improved. ... Results show that MPPSO ... is an effective and practical method.” [145].
A preliminary study on negative correlation learning via Correlation-Corrected Data (NCCD) (2005)	A “cooperative Neural Network Ensemble Learning method based on Negative Correlation learning. ... [It] enables easy integration of various network models and reduces communication bandwidth significantly for effective parallel speedup. ... Comparison with the best Negative Correlation learning method ... demonstrates comparable performance at significantly reduced communication overhead” [146].

Table 3.3: Overview of Negative Correlation Learning related researches - 2.

Evolutionary random neural ensembles based on negative correlation learning (2007)	“Incorporates bootstrap of data, random feature subspace and evolutionary algorithm with negative correlation learning ... [promoting] diversity ... of individual NNs ... [to] learn different parts ... of the training data so that the ensemble can learn better the entire training data. ... [It] emphasizes ... cooperation among different individual NNs ... thus improves the generalization ... out-of-bag (OOB) estimation. ... Another benefit of this algorithm ... [is that] performance ... is better than the performance of other ensemble algorithms” [147].
Analyzing Anti–correlation in Ensemble Learning (2007)	“Proposes an alternative anti–correlation measure, RTQRT–NCL, which shows significant improvements ... particularly with larger ensembles” [148].
A comparative study of data sampling techniques for constructing neural network ensembles (2009)	A “negative correlation learning that implicitly encourages different networks to [learn] different training spaces is shown as better or at least comparable to Bagging and Boosting that explicitly create different training spaces” [149].
Regularized negative correlation learning for Neural Network ensembles (2009)	“NCL ... introduces a correlation penalty term to the cost function of each individual network so that each ... minimizes its ... MSE together with the correlation of the ensemble. ... When $\lambda=1$ ... [it] only minimizes the MSE without regularization. ... [That is] why NCL is prone to overfitting. ... Tuning the correlation parameter $\lambda$ [when $\lambda=1$ ] ... cannot overcome the overfitting problem. ... [This research] proposes ... regularized negative correlation learning (RNCL) ... which incorporates an additional regularization term ... [which] decomposes the ensemble’s training objectives, including MSE and regularization, into a set of sub-objectives ... implemented by an individual Neural Network. ... Also ... [it] provides an automatic algorithm to optimize regularization parameters. ... Experiments demonstrate that RNCL achieves better performance than NCL, especially when the noise level is nontrivial in the data set” [150].
Multiobjective neural network ensembles based on regularized negative correlation learning (2010)	“Multiobjective regularized negative correlation learning (MRNCL) ... incorporates an additional regularization term [in] the ensemble and uses the evolutionary multiobjective algorithm to design ensembles. ... [It also] defines ... crossover and mutation operators and adopts nondominated sorting algorithm with fitness sharing and rank-based fitness assignment. ... Experiments ... demonstrate that MRNCL achieves better performance than NCL, especially when the noise level is nontrivial in the data set” [151].
A new selective Neural Network ensemble with negative correlation (2012)	“Hierarchical pair competition-based parallel genetic algorithm (HFC–PGA) ... aim ... to achieve ... the best Neural Network ... also a diversity of potential Neural Networks ... selected to build an ensemble such that the generalization error is minimized and the negative correlation is maximized” [152].

Table 3.4: Overview of Negative Correlation Learning related researches - 3.

<p>Combining features of negative correlation learning with mixture of experts in proposed ensemble methods (2012)</p>	<p>“NCL and mixture of experts (ME) ... employ different special error functions for the simultaneous training of NNs to produce negatively correlated NNs. ... [They] have different but complementary features, so ... a hybrid system ... may be better than each of its basis approaches. ... Two approaches are proposed ... (1) G-NCL, a dynamic combiner of ME ... combines the outputs of base experts in the NCL method ... [with] weights estimated dynamically from the inputs ... (2) [mixture of negatively correlated experts], MNCE, ... [has the] capability of parameter [controlling] for NCL ... in the error function of ME, ... [enabling] efficiently adjustment between experts ... to establish better balance in bias-variance covariance trade-offs. ... Thus, [it] improves the generalization ability. ... Results show [this] ... method ... improved performance over the original methods” [153].</p>
<p>Fast decorrelated Neural Network ensembles with random weights (2014)</p>	<p>“NCL aims to produce ensembles with sound generalization capability through controlling the disagreement among base learners’ outputs ... usually implemented by using feed-forward Neural Networks with error back-propagation algorithms (BPNNs). However, it suffers from slow convergence, [due to] local minima problem and model uncertainties caused by the initial weights and the setting of learning parameters. To achieve a better solution ... [this papers] employs ... random vector functional link (RVFL) networks as ... base models. [These] are generated randomly ... [with a] cost function defined for NCL. ... Results indicate ... [this] approach outperforms other ensembling techniques” [154].</p>
<p>A niching evolutionary algorithm with adaptive negative correlation learning for Neural Network ensemble (2017)</p>	<p>Proposes an “evolutionary algorithm with adaptive negative correlation learning ... in which the penalty coefficient <math>\lambda</math> is set to dynamically change during training ... with the purpose of appropriately controlling the trade-off between the diversity and accuracy in the ensemble. Further, a modified dynamical fitness sharing method is applied to preserve the diversity of population during training. ... Results show that [the] method can be used to design a satisfactory NN ensemble and outperform related works” [155].</p>
<p>Regularizing deep Neural Networks with an ensemble-based decorrelation method (2018)</p>	<p>The “Ensemble-based Decorrelation Method (EDM) ... is motivated by the idea to ... improve generalization capacity of DNNs. EDM can be applied to hidden layers in fully connected Neural Networks ... [by treating] each hidden layer as an ensemble of several base learners [and] dividing all the hidden units into several non-overlap groups ... viewed as ... base learners. EDM encourages DNNs to learn more diverse representations by minimizing covariance between all base learners during the training. ... Results ... demonstrate that EDM can ... reduce the overfitting and improve the generalization capacity of DNNs” [156].</p>
<p>Crowd Counting with Deep Negative Correlation Learning (2018)</p>	<p>“Deep convolutional networks (ConvNets) have achieved unprecedented performances. ... However, their adaptations to crowd counting on single images are still in their infancy and suffer from severe over-fitting. ... [A] new learning strategy to produce generalizable features [through] deep negative correlation learning ... deeply learns a pool of decorrelated regressors ... through managing their intrinsic diversities. ... [It] is [an] end-to-end-trainable [algorithm]. ... Extensive experiments ... indicate the superiority of D-ConvNet ... compared ... [to] state-of-the-art methods.” [157].</p>

Table 3.5: Overview of Negative Correlation Learning related researches - 4.

Nonlinear Regression via Deep Negative Correlation Learning (2021)	“[In] deep learning, two common solutions exist [for nonlinear regression problems] i) [employ] a robust loss function ... jointly optimizable with the deep [CNN], ii) ... ensemble of deep networks. ... [The former] improves performance [but] lacks due to ... limitation of choosing a single hypothesis ... [and the] latter suffers from much larger computational complexity... [This paper] proposes to regress via an efficient “divide and conquer” manner ... [viewed as a] generalization of negative correlation learning. ... Without extra parameters ... [the] proposed method controls bias-variance-covariance trade-off systematically and usually ... each base model is ... “accurate” and “diversified”. ... Each sub-problem ... has less ... complexity and thus is easier to optimize. ... Experiments demonstrate superiority over challenging baselines [and] versatility” [157].
Generalized Negative Correlation Learning for Deep Ensembling (2021)	“Numerous works on decomposing ... loss functions [but] exact mathematical connection is rarely exploited. ... For ensembling ... formulate ... bias-variance decomposition for arbitrary ... differentiable loss functions ... [and] derive a Generalized Negative Correlation Learning (GNCL) algorithm which offers explicit control over the ensemble’s diversity and smoothly interpolates between ... independent training and joint training. ... Discusses under which circumstances training of an ensemble of Neural Networks might fail and what ensembling method should be favored depending on the choice of the individual networks.” [158].

Table 3.6: Overview of Negative Correlation Learning related researches - 5.

### 3.3.2 Boosting Neural Network Ensemble

One interesting yet far from completely researched option is combining multiple Boosting variations and implementations (section 2.5.2.2, section 2.5.2.6, section 2.5.2.7, section 2.5.2.5) with DL (section A.7).

### 3.3.3 Combining strategies for ensemble generation

There are multiple proposed approaches to combine strategies for generating ensembles, each offering different breakthroughs from those presented in this work.

MultiBoosting combines AdaBoost with wagging, a variant of Bagging using C4.5 as the base learners achieving better results and execution time than the constituent algorithms [159].

Multistrategy Ensemble Learning investigates the hypothesis that accuracy improvement is due to base learners’ increased diversity. So three new multistrategy EL techniques were developed with results showing they are, on average, more accurate than their base strategies [160].

Cocktail ensemble uses a hybrid mechanism for combining multiple individual ensembles via pairwise combination with a regression error-ambiguity decomposition. In other words, it resembles an ensemble of ensembles. Results show that the proposed approach outperforms the individual ensembles, two other methods of ensemble combination, and two state-of-the-art regression approaches [161].



### 3.4 Applications

Since the NNE was first proposed, multiple researchers have focused their efforts and attention on further development. In that spirit, they produced an open-ended stream of in-depth studies and various approaches. At the present day, NNE techniques have been comprehensively and successfully adopted for broad practical purposes and grew into a relevant ML research field. Tables 3.7 and 3.8 present some known applications.

Paper	Summary
Fast license plate character recognition based on AdaBoost (2010)	“Used dynamic adaptive weight cutting method to fast train AdaBoost, and obtained good recognition effect” [162]
Research of P2P traffic identification based on neural network ensemble (2010)	“Used dynamic weighted ensemble methods to P2P traffic identification” [163]
AdaBoost based ensemble of neural networks in analog circuit fault diagnosis (2010)	“Proposed a new NNE method based on AdaBoost, and applied it to analog circuit fault diagnosis” [164]
Performance of global-local hybrid ensemble versus Boosting and Bagging ensembles (2013)	“Presented a global-local hybrid ensemble, which employs both local and global learners, and compares its performance against Bagging and Boosting” [165]
Hebbian ensemble Neural Network for robot movement control (2013)	“Proposed Hebbian NNE with large information capacity for complex maneuver representations applied in robot movement control” [166]
Fast decorrelated neural network ensembles with random weights (2014)	“Combined random vector functional link networks with the least square method and the Negative correlation learning strategy for fast building of decorrelated NNE” [154]
A novel decorrelated Neural Network ensemble algorithm for face recognition (2015)	“Described a NNE based on two-dimensional Neural Networks with random weights incorporated with the negative correlation ensemble learning strategy for building the final system for face recognition” [167]

Table 3.7: Overview of Practical Applications of NNEs - 1.

Applying the ensemble artificial neural network-based hybrid data-driven model to daily total load forecasting (2015)	“Created a hybrid data-driven NNE model, calibrated via multi-objective optimization algorithm, that combines partial mutual information-based input variable selection with NNE-based output estimation and k-nearest neighbor regression-based output error estimation for short-term electricity load forecasting” [168]
---	---

Table 3.8: Overview of Practical Applications of NNEs - 2.

Other applications include multi-target regression (Hadavandi et al. [169]), distributed training (Zhang and Zhong [170]), time series (Kourentzes et al. [171] and Smith and Jin [172]), evolutionary techniques (Tian et al. [173], Soares et al. [174], and Zhao et al. [175]), and reinforcement learning (Faußer and Schwenker [176]).



## Chapter 4

# A framework to construct Neural Network Ensembles for regression

### Contents

---

<b>4.1</b>	<b>Level-0</b>	<b>51</b>
<b>4.2</b>	<b>Level-1</b>	<b>58</b>
4.2.1	Algorithm Hypothesis Formulation	60
4.2.2	Results	61
<b>4.3</b>	<b>Statistical Validation</b>	<b>70</b>
<b>4.4</b>	<b>Time analysis</b>	<b>74</b>

---

A thorough testing infrastructure was created to confirm how best to combine current state-of-the-art NNE strategies.

This framework has three steps. It starts by evaluating the error decomposition of Tables 4.1 and 4.2 ensemble techniques in bias, variance, and covariance [26]. Then, it identifies the most complementary existing strategies (most reduce each ensemble error component in conjunction). Finally, it combines these strategies to create a multitude of new ensemble algorithms.

The following list presents this thesis' premises:

- the NN estimator uses a *ReLU* activation function in the input layer with a number of neurons according to the number of train set's features, *MAE* as the loss function, and *AdaGrad* as the optimizer;
- the estimator's structure and parameters are consistent throughout all ensemble algorithms and test levels for computational simplicity and comparison trustworthiness. In other words, the Network architecture is not a variable affecting the final results;

- each ensemble has five estimators, and has been averaged five times except for Snapshot (and derived architectures) which have been averaged 15 times for improved results stability;
- different regression datasets for each of the two levels of the framework;
- holdout with 80% for training and 20% for testing as resampling method;
- datasets' categorical values are one-hot encoded using *scikit – learn* <sup>24</sup>;
- missing values are removed by dropping their respective instances;
- features are scaled not to skew the algorithm's internal workings;
- two baselines: (1) a single NN with the same architecture as the ensemble's base estimators, and (2) the Simple Average of the base learners results;
- Level-1's tests are corroborated by Level-0's results evaluated through error decomposition;
- ensemble error decomposition in bias, variance, and covariance serves as an insight into understanding which ensemble strategies most reduce each individual error component, thus elucidating on which strategies should be merged to reduce all components the most throughout;
- each component on a stacked bar graph refers to the average of the normalized sum of that component for each of its Level's dataset;
- the right y-axis of heatmap graphs refers to the percentage difference between a given line's algorithm on a particular dataset and the NN single model baseline (first row). -1.0 means the error has diminished 100%, and 3.0 means the error increased at least 300%;
- stacked bar graphs with multiple versions of the same algorithm use the individual, non-merged, default version of that same algorithm as the comparison metric;
- the framework is powered by *sklearn* <sup>24</sup>, *Keras* <sup>25</sup>, and *TensorFlow* <sup>26</sup>. It is essential to understand that *Keras* is a high-level API, that runs on top of *Tensorflow*, designed to be fast and easy to use, hence perfect for quick implementations. However, more complex designs are often impossible to create with *Keras* alone, thus requiring *Tensorflow*. This is a high and low-level framework used for high-performance models and large datasets, thus ideal for DL (section A.7) researches;
- given that every collected dataset is entirely and freely available/maintained online, confidentiality issues are not a thing. Nonetheless, they should be used responsibly and ethically;
- note that the provided URLs were used to gather the datasets, but they may also be collected from various other sources. Nonetheless, the referred URLs are stable, meaning they are expected to host the data indefinitely;
- datasets are licensed under the [CC0](#) license; I bear the responsibility for rights violations or infringements regarding the datasets and adherence to the data license;

- csv file format is used regarding the datasets. However, some of them were in different formats, namely, txt, so they had to be converted;
- although AdaBoost.R2, Gradient Boosting (Histogram), LGBM, CatBoost, and XGBoost use Decision Trees instead of NNs as base learners, they have only been added in the Level-0 because some of their characteristics can add value in other algorithms, so they are used for performance comparison.

## 4.1 Level-0

This level’s tests essentially consist of plainly implementing the current most recognized and established Neural-Network Ensemble specific and general algorithms (Tables 4.1 and 4.2). Every ensemble algorithm has its particular characteristics that aim to vary the training data (section 3.1), estimators’ characteristics (section 3.2), estimators’ generation strategy (section 3.3), and the prediction integration mechanism (section 2.6). So, it is essential to analyze which ensemble architectures most diminish each error component and, hopefully, discover which of their characteristics are most responsible for that reduction.

Ensemble Algorithm	Tests Designation	GEN	INT	BL	SEC	REF
Simple Average	average	Par	SA	NN	2.6.1	[12]
Random Splits	rand_split0.X	Par	SA	NN	2.3.1	
K-fold Cross-training	cross_training	Par	SA	NN	2.3.2	[16]
Random Subspace	rand_subspace0.X	Par	SA	NN	2.3.5	[49]
Pasting	pasting0.X	Par	SA	NN	2.3.3	[58]
Random Patches	pasting0.X +rand_subspace0.X	Par	SA	NN	2.3.6	[63]
Horizontal Averaging	horizontal_avg	Str	SA	NN	3.2.2.1	[11]
Polyak Averaging	polyak_avg	Str	SA	NN	3.2.2.2	[127]
Snapshot with Cosine Annealing Learning Rates	snapshot	Str	SA	NN	3.2.2.2 3.2.2.3	[121]
NCL	ncl0.X	Seq	SA	NN	3.3.1	[133]
Dropout	dropoutX	Par	SA	NN	3.2.1.2	[177]
Bagging	bagging	Par	SA	NN	2.5.1	[54]
AdaBoost(.R2)	adaboost_scratch	Seq	WA	NN	2.5.2.1	[77]
AdaBoost(.R2)	adaboost_nn	Seq	WA	NN	2.5.2.1	[77]

Table 4.1: Overview of Generation Mode (GEN), Integration Method (INT), framework’s Base Learner (BL), section (SEC), and main reference (REF) for the main ensemble methods for regression. Par stands for Parallel, Str for Stream (base models generated from the same original base learner), Seq for Sequential, SA for Simple Average, WA for Weighted Average, DT for Decision Trees, and NN for Neural Networks. The suffix *X* indicates the hyperparameter(s) of the method in the experiments - 1.

<sup>24</sup><https://scikit-learn.org/stable/>

<sup>25</sup><https://keras.io/>

<sup>26</sup><https://www.tensorflow.org/>

Ensemble Algorithm	Tests Designation	GEN	INT	BL	SEC	REF
Super Learner	super_learner	Par	WA	NN	<a href="#">2.5.4.2</a>	[10]
Blending	blending	Par	WA	NN	<a href="#">2.5.4.1</a>	[96]
Stacking	stacking_nn	Par	WA	NN	<a href="#">2.5.4</a>	[43]
AdaBoost(.R2)	adaboost_default	Seq	WA	DT	<a href="#">2.5.2.1</a>	[77]
Gradient Boosting	gradient_boosting	Seq	WA	DT	<a href="#">2.5.2.2</a>	[78] [178]
Gradient Boosting Histogram	gradient_boosting_hist	Seq	WA	DT	<a href="#">2.5.2.8</a>	[179]
LGBM	lgbm	Seq	WA	DT	<a href="#">2.5.2.7</a>	[89]
CatBoost	catboost	Seq	WA	DT	<a href="#">2.5.2.6</a>	[88]
XGBoost	xgboost	Seq	WA	DT	<a href="#">2.5.2.5</a>	[85]
Random Forest	rand_forest	Par	SA	DT	<a href="#">2.5.1.1</a>	[29]
Extra Trees	extra_trees	Par	SA	DT	<a href="#">2.5.1.2</a>	[66]

Table 4.2: Overview of Generation Mode (GEN), Integration Method (INT), framework’s Base Learner (BL), section (SEC), and main reference (REF) for the main ensemble methods for regression. Par stands for Parallel, Str for Stream (base models generated from the same original base learner), Seq for Sequential, SA for Simple Average, WA for Weighted Average, DT for Decision Trees, and NN for Neural Networks. The suffix  $X$  indicates the hyperparameter(s) of the method in the experiments - 2.

Ten different datasets listed in Table 4.3 are used. These give statistical confidence as the empirical results meet the theoretical assumptions.

name	no. of samples	no. of features	categorical features	missing data	origin
fried delve	40768	10	no	no	27
energydata complete	19735	29	yes	no	28
bike sharing/hour	17389	16	yes	no	28
student	395	31	no	no	28
friedman	1200	5	no	no	29
mv	40768	10	no	no	29
atlttime1004a	17812	7	yes	no	30
triazines	186	61	no	no	31
fruitfly	125	5	no	no	31
add10	9792	11	no	no	32

Table 4.3: Level-0 datasets’ information.

Looking at Fig. 4.1, and broadly speaking, it is clear to see that ensembles, almost as a whole, lower predictions’ overall variance while maintaining bias low simultaneously. In this way, the generalization error is lowered, and better performance is achieved.

<sup>27</sup><https://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html>

<sup>28</sup><https://archive.ics.uci.edu/ml/datasets.php>

<sup>29</sup><https://sci2s.ugr.es/keel/category.php?cat=reg#sub2>

<sup>30</sup><http://users.stat.ufl.edu/~winner/data/atlttime1004a.dat>

<sup>31</sup>[https://www.openml.org/search?sort=runs&order=desc&type=task&from=100&q=+tasktype.tt\\_id%3A2](https://www.openml.org/search?sort=runs&order=desc&type=task&from=100&q=+tasktype.tt_id%3A2)

<sup>32</sup><https://www.cs.toronto.edu/~delve/data/>

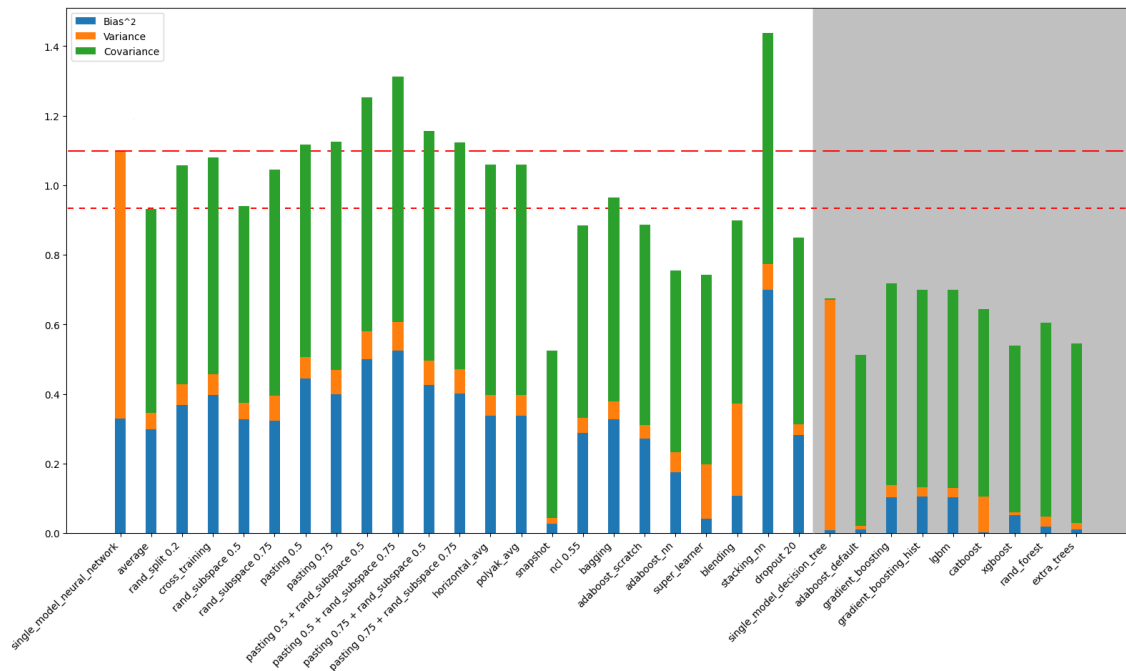


Figure 4.1: Individual ensemble results. NNE results have a white background, while Decision Tree Ensemble results have a grey background. The wider red dashed line and narrower red dashed line depict the Single NN baseline error threshold and the Simple Averaging ensemble baseline error threshold, respectively.

As expected, Simple Average achieved better results than the single model baseline. Due to their characteristics, Horizontal Averaging and Polyak Averaging performed poorly since contiguous epoch estimators suffer from a lack of diversity. Surprisingly Snapshot with SGDR (Cosine Annealing LR) performed exceptionally well in reducing covariance, variance, and specifically, bias. So it is possible to infer that aggressively varying the LR of NNs is a successful way of promoting estimators' diversity. Other notable results were Dropout ( $p = 0.2\%$ ), and NCL ( $\lambda = 0.55$ ), which slightly lowered the error compared to the Average ensemble.

The base models' selection needs to be consistent with the way to aggregate them. If the base models have low bias and high variance, an aggregating scheme that favors reducing the ensemble's variance (e.g., Bagging) should be considered. Otherwise, an aggregating scheme favoring reducing the ensemble's bias (e.g., Boosting) should be considered when high bias and low variance base models exist. Given that the NN single model exhibits both a relatively high bias and variance, both mentioned ensemble strategies are worth exploring.

Surprisingly, Bagging failed to improve results compared to Simple Average. One can conclude that bootstrap sampling faltered in promoting the expected estimator diversity in these specific datasets. Regarding Bagging variants, despite Random Forest performing pretty well by reducing variance, covariance, and most noticeably bias, Extra Trees came out on top. Given that these two variants performed positively, Random Subspace and other techniques that vary the training data are worth exploring for NNs. The rest either (1) were already explored (models fitted on the full dataset), (2) had bad empirical results (Bootstrapping), or (3) do not directly apply to

NNs (optimal split point selection algorithms).

On the other hand, Boosting achieved the best results throughout almost all different implementations. These results clearly show that refocusing subsequent models' attention on remaining difficult observations and using a Weighted Average final prediction according to each estimator's respective performance are exceptional ensemble strategies. AdaBoost.R2 manifested the best global error value closely followed by XGBoost. Curiously, CatBoost smashed bias but increased variance, while XGBoost smashed variance but failed to reduce bias as much as CatBoost.

Almost all Decision Tree Boosting methods reduced covariance and, very appreciable, bias and variance while NN Boosting Methods lowered bias and covariance but increased variance. Comparing Decision Tree AdaBoost.R2 (default) with NN AdaBoost.R2, the latter achieved worse results than the former. A possible avenue to improve the latter's performance is to vary its estimators' characteristics. Given that AdaBoost.R2, XGBoost, and CatBoost performed the best, but also, Gradient Boosting and LGBM manifested positive results, adding regularization terms to the NNs (e.g., Dropout), Random Subspace, and Weighted Average are worth exploring. The rest either (1) do not directly apply to NNs (default parameters, split points, and individual NN pruning), (2) do not apply to the specific baseline (does not suffer from overfitting), (3) do not offer advantages due to the dataset characteristics (no categorical features and no missing data), (4) have already been explored (parallel/independent processing) or (5) perform poorly on NNs (Bootstrapping).

Note that the Decision Tree single model and Decision Tree ensembles' results are better than the NN baseline and NNEs' results, respectively, because (1) it is easier for Decision Trees to perform well on default hyperparameters, (2) the adopted NN estimator architecture is very simple and trained on few epochs, and (3) some datasets may have too few samples.

Meta-model techniques depend on having a simple model that provides smooth prediction interpretations that offset individual models' deficiencies for better performance. For that reason, Super Learner and Blending, which use a linear Meta-model, achieved the best results, with the former edging out the latter. However, both obtained the worst variance results, particularly Blending. Stacking, which uses a non-linear Meta-model (NN), had disappointing results. Hence, linear Meta-models are, in this case, superior to non-linear Meta-models. Also, one might wonder that Blending's characteristics (fit a linear Meta-model on estimators' holdout set predictions) lower bias but promote variance, and Super Learner's characteristics (out-of-fold predictions amid K-fold Cross-validation) lower bias and increase variance, but more positively. Consequentially, Super Learner's results indicate a better global error reduction when compared with Blending and Stacking.

Regarding the specific numerical results, Table 4.4 depicts every error component thoroughly.

Algorithm	$Bias^2$	Variance	Covariance	Error
single_model_neural_network	0.33	0.766	0.004	1.099
average	0.298	0.049	0.584	0.931
rand_split 0.2	0.368	0.06	0.628	1.057
cross_training	0.398	0.06	0.623	1.08
rand_subspace 0.5	0.326	0.047	0.566	0.939
rand_subspace 0.75	0.324	0.071	0.65	1.044
pasting 0.5	0.444	0.062	0.611	1.117
pasting 0.75	0.4	0.07	0.654	1.124
pasting 0.5 + rand_subspace 0.5	0.5	0.081	0.672	1.253
pasting 0.5 + rand_subspace 0.75	0.525	0.082	0.705	1.313
pasting 0.75 + rand_subspace 0.5	0.425	0.07	0.66	1.156
pasting 0.75 + rand_subspace 0.75	0.401	0.07	0.652	1.123
horizontal_avg	0.337	0.059	0.663	1.059
polyak_avg	0.337	0.059	0.664	1.06
snapshot	0.027	0.017	0.481	0.525
ncl 0.55	0.288	0.043	0.554	0.885
bagging	0.326	0.051	0.587	0.965
adaboost_scratch	0.272	0.039	0.577	0.887
adaboost_nn	0.175	0.058	0.522	0.756
super_learner	0.042	0.155	0.546	0.742
blending	0.107	0.266	0.527	0.9
stacking_nn	0.7	0.074	0.665	1.439
dropout 20	0.283	0.029	0.538	0.85
single_model_decision_tree	0.008	0.663	0.004	0.675
adaboost_default	0.01	0.011	0.491	0.512
gradient_boosting	0.104	0.033	0.58	0.717
gradient_boosting_hist	0.104	0.027	0.569	0.7
lgbm	0.104	0.027	0.57	0.7
catboost	0.003	0.103	0.539	0.645
xgboost	0.052	0.008	0.479	0.54
rand_forest	0.019	0.027	0.557	0.604
extra_trees	0.009	0.019	0.517	0.545

Table 4.4: Individual ensemble numerical results.

Some varying training data ensemble techniques offer positive characteristics, except for Bootstrapping (Bagging) and K-fold Cross-training. Also, Random Split (section 2.3.1) result's increased estimator diversity did not compensate for the loss of available training data (Fig. 4.2).

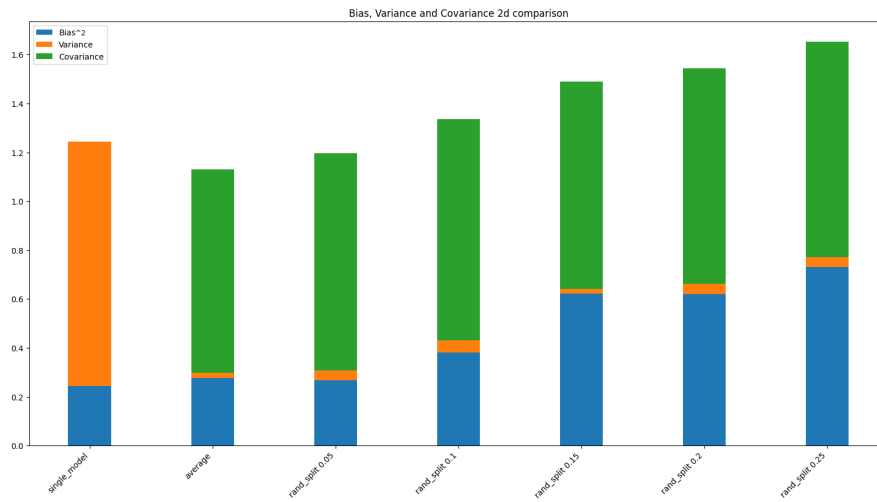


Figure 4.2: Comparison of different Random Split configurations.

Pasting, Random Subspace, and Random Patches suffered from a lack of samples and features to select different subsets or combinations. Therefore, more thorough testing with selected datasets with enough samples/features was performed to obtain more insightful results. Results, as seen in Fig. 4.3, show that these techniques alone do not promote better performance.

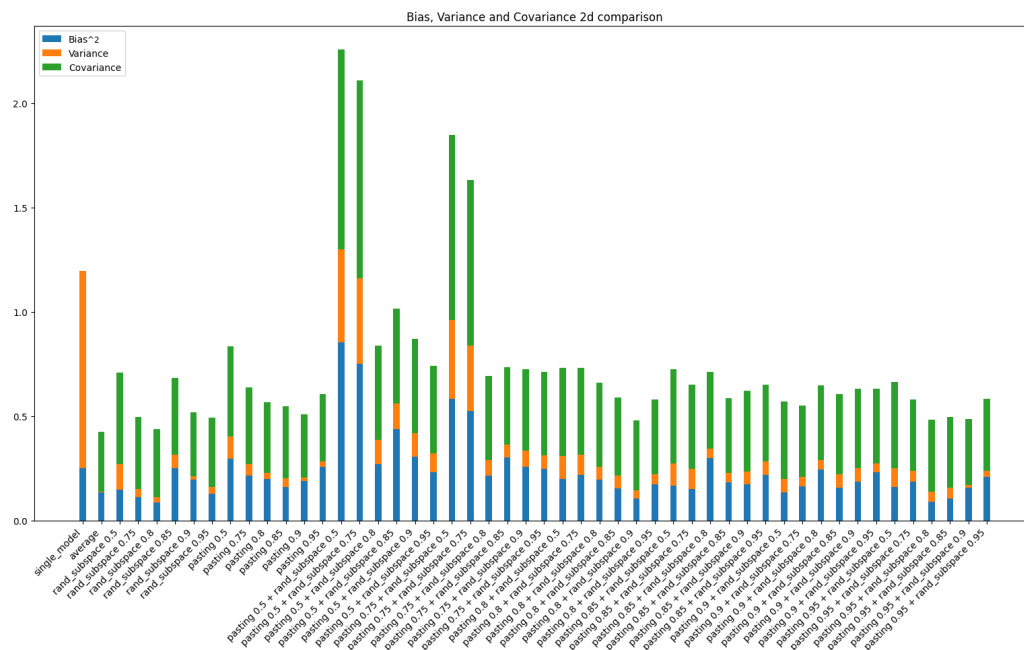


Figure 4.3: Comparison of different Pasting, Random Subspace and Random Patches configurations.

In actuality, varying the number of features beyond a certain threshold is slightly better than varying the number of samples for individual estimators. Not surprisingly, when combined, these techniques harm performance.



There is a caveat. Performance improvements using these techniques are very dependant (1) on the dataset and (2) on the randomness of the selected variables. So these techniques may still be helpful in particular circumstances.

Analyzing the individual dataset error results (Fig. 4.4), it is clear that most algorithms consistently perform across the datasets, either good or bad, with few outliers (except Gradient Boosting and LGBM).

Two findings are worth noting:

- with the existing configuration, Stacking performs poorly across all datasets, and on some occasions, it behaves pretty disastrously;
- LGBM and Gradient Boosting algorithms have similar error values across all datasets since the former stems from the latter.

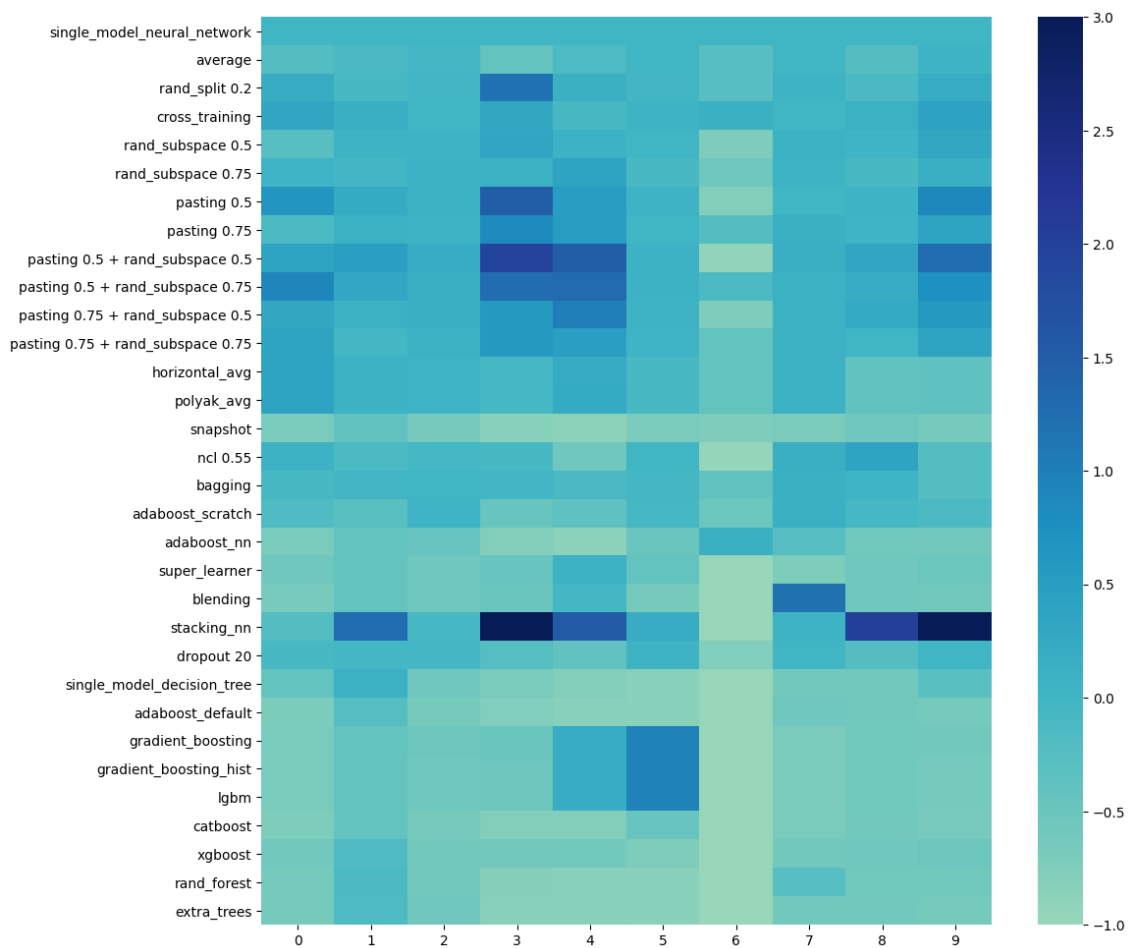


Figure 4.4: Ensemble heatmap error results. X-axis reflects each dataset utilized.

On the whole, this level's experiments revealed that:

- most ensemble algorithms surpass the Single Model baseline, but only some consistently improve on the Simple Average Ensemble baseline bias-variance-covariance components;

- some techniques offer marginal improvements, while others significantly and complementarily reduce one or more error components;
- varying the ensemble NNs' characteristics, particularly aggressively varying the LR, offers exceptional results;
- NCL and Dropout consistently top the Simple Average ensemble;
- Meta-model techniques also manifest very good results.

Bottom line, the best-performing NNE algorithms were Snapshot with SGDR, Super Learner, and Blending ensembles. Additionally, since Dropout and NCL achieved better performance than the Simple Average ensemble and can combine with other algorithms, they are considered plausible and promising candidates for the next level's experiments.

## 4.2 Level-1

This level starts by (1) pair-wise merging every previous level's NNEs with the referred most promising approaches, and (2) dive deep into merging all possible combinations of the previous level's best-performing NNE algorithms with the referred most promising approaches. More specifically, the aim is to discover if it is possible and beneficial to combine multiple ensemble methods or their individual characteristics to build a single ensemble architecture that exhibits performance increases.

Twenty different datasets, listed in Tables 4.5 and 4.6, and distinct from those used in the previous level, are used to ensure independent results.

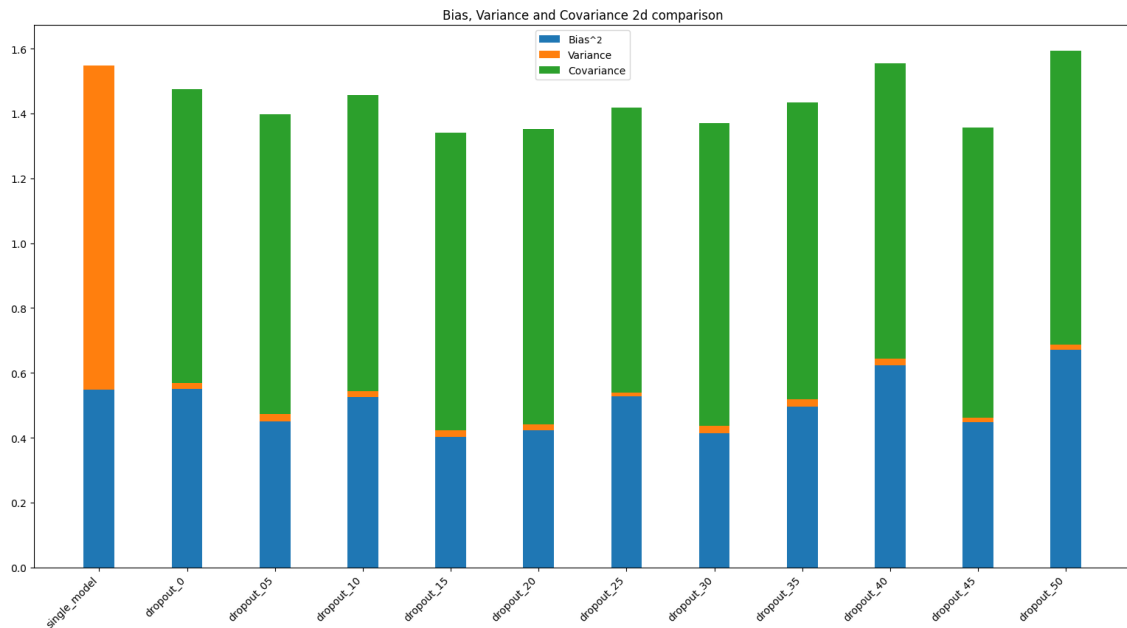
name	no. of samples	no. of features	categorical features	missing data	origin
auto price	159	16	no	no	33
auto mpg	398	8	no	yes	33
cpu act	8192	22	no	no	33
cpu small	8192	13	no	no	33
housing boston	506	14	no	no	33
housing california	20460	9	no	no	33
machine cpu	209	7	no	no	33
pole telecomm	15000	49	no	no	33
stock airplane companies	950	10	no	no	33
wisconsin breast cancer	194	33	no	no	33
aileron	13750	41	no	no	33
airfoil self-noise	1503	6	no	no	34
combined cycle power plant	9568	5	no	no	34

Table 4.5: Level-1 datasets' information - 1.

name	no. of samples	no. of features	categorical features	missing data	origin
real estate valuation	414	8	no	no	34
yacht hydrodynamics	308	7	no	no	34
insurance concrete	1338	7	yes	no	35
compressive strength	1030	9	no	no	36
electrical maintenance	1056	5	no	no	36
house 16h	22784	17	no	no	36
pole telecommunications	14998	27	no	no	36

Table 4.6: Level-1 datasets' information - 2.

However, first, it is required to discover the best  $p$  and  $\lambda$  hyperparameter values, respectively, for Dropout and NCL. After extensive testing on Dropout ensemble and Dropout empowered NNEs,  $p = 0.15$  was found to be the best value (error equal to 1.34), with  $p = 0.2$  and  $p = 0.45$  following right behind (error equal to 1.352 and 1.356 respectively) (Fig. 4.5).

Figure 4.5: Comparison of Dropout empowered ensembles with different  $p$  values.

Analogously, after extensive testing on NCL ensemble and NCL empowered NNEs,  $\lambda = 0.55$  was found to be the best value (Fig. 4.6).

<sup>33</sup><https://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html>

<sup>34</sup><https://archive.ics.uci.edu/ml/datasets.php>

<sup>35</sup><https://www.kaggle.com/mirichoi0218/insurance>

<sup>36</sup><https://sci2s.ugr.es/keel/category.php?cat=reg#sub2>

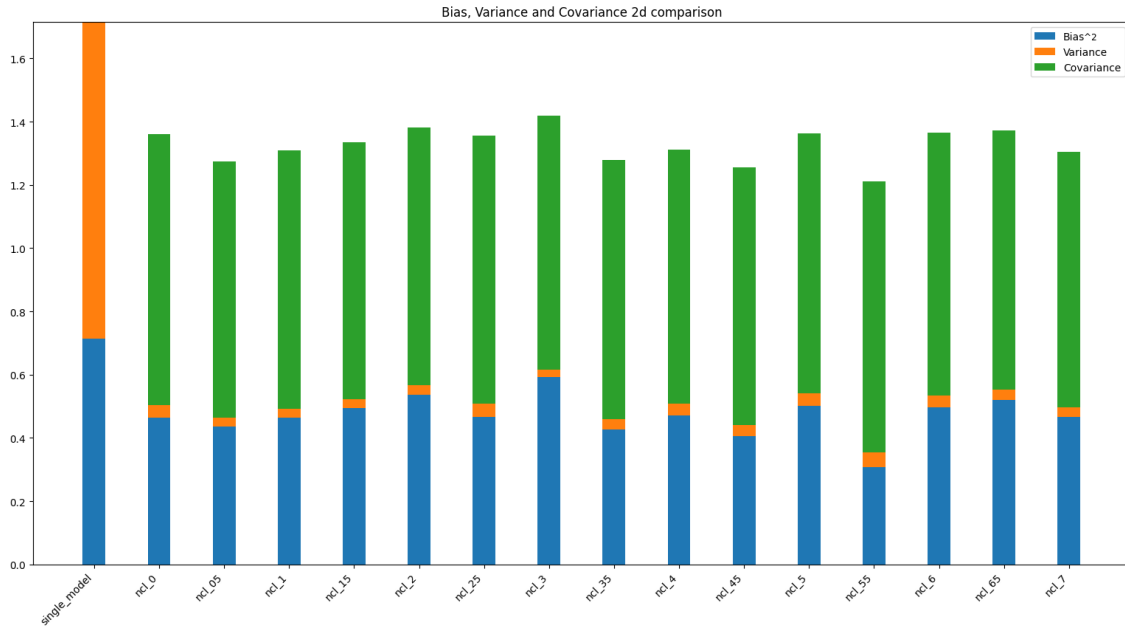


Figure 4.6: Comparison of NCL empowered ensembles with different  $\lambda$  values.

#### 4.2.1 Algorithm Hypothesis Formulation

This is the theoretical step to idealize, modulate, and adjust the proposed innovative algorithms.

These are divided between (1) integrating the previous level's most promising approaches with every other NNE method, including the referred best-performing NNE algorithms (Alg. 6, 8, 9, 10, 11, 12, 13, 14, and 15), and (2) integrating the previous level's most promising approaches with the previous level's best-performing NNE algorithms mutually combined (Alg. 7, 16, and 17).

*Average\_NCL\_Dropout* and *Snapshot\_NCL\_Dropout* are thoroughly explained in this section (Alg. 6 and 7). The remaining proposed Level-1's hybrid algorithms are depicted in appendix D.

---

##### Algorithm 6 Average\_NCL\_Dropout

---

**Input:** number of estimators  $M$ , Dropout percentage  $p$

**Output:** final prediction  $p$

- 1: create  $M$  NN estimators with  $p$  value
  - 2: **for**  $m = 1$  to  $M$  **do**
  - 3:     perform NCL procedure between estimator  $m$  and  $L[:m]$  estimators
  - 4:     on a train\_set
  - 5: **end for**
  - 6: use Simple Average to integrate saved estimators' test\_set predictions
-

**Algorithm 7** Snapshot\_NCL\_Dropout**Input:** number of estimators  $M$ , number of training epochs  $N$ , Dropout percentage  $p$ **Output:** result  $Y_M(x)$ 

```

1: create  $M$  NN estimators with  $p$  value
2: for  $n = 1$  to  $N$  do
3:   if  $(N - n) < M$  then
4:     save copy of current original model
5:     perform one epoch of NCL procedure between original model and
6:       saved estimators on a train_set with a SGDR policy
7:   else
8:     train original model one epoch on a train_set with a SGDR policy
9:   end if
10: end for
11: use Simple Averaging to integrate saved estimators' test_set predictions

```

**4.2.2 Results**

Fig. 4.7, and, more concretely, Fig. 4.8 compare the effects of pair-wise merging every previous level's NNEs with the referred most promising approaches. By looking at the proposed algorithms, it is clear that, almost always, joining NCL and Dropout to each default version improves results the most, and in the case which it does not, joining only NCL has the number one spot. Therefore, exploring the previous level's best-performing NNE algorithms and most promising approaches is a reasonable option.

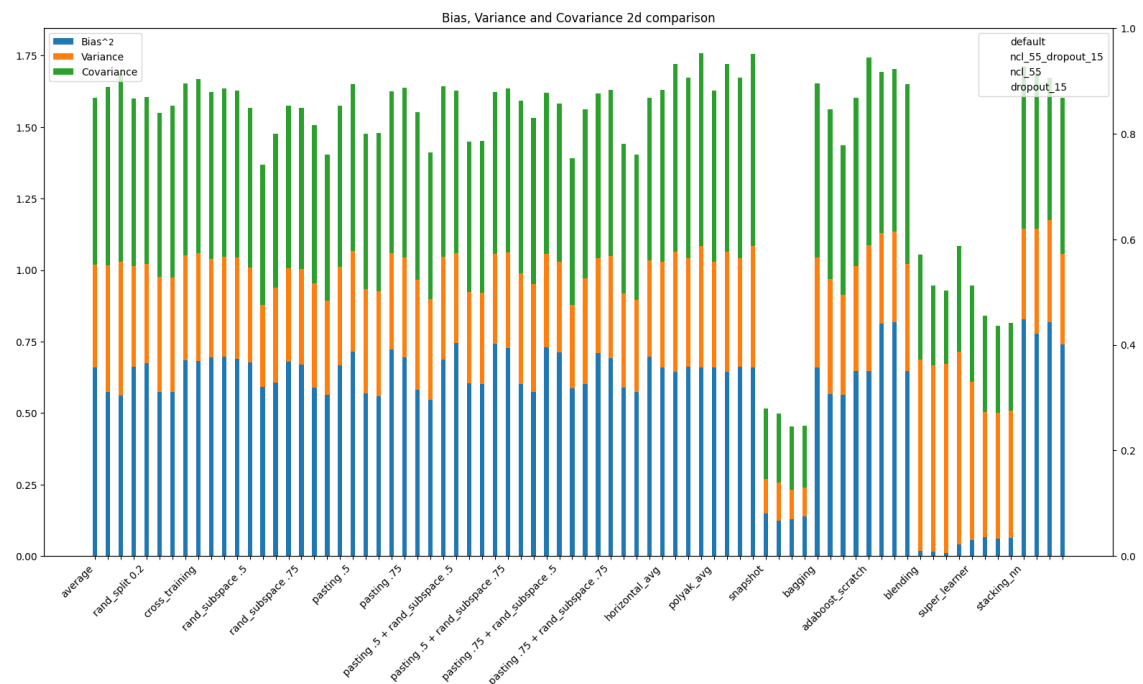


Figure 4.7: Proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' results.

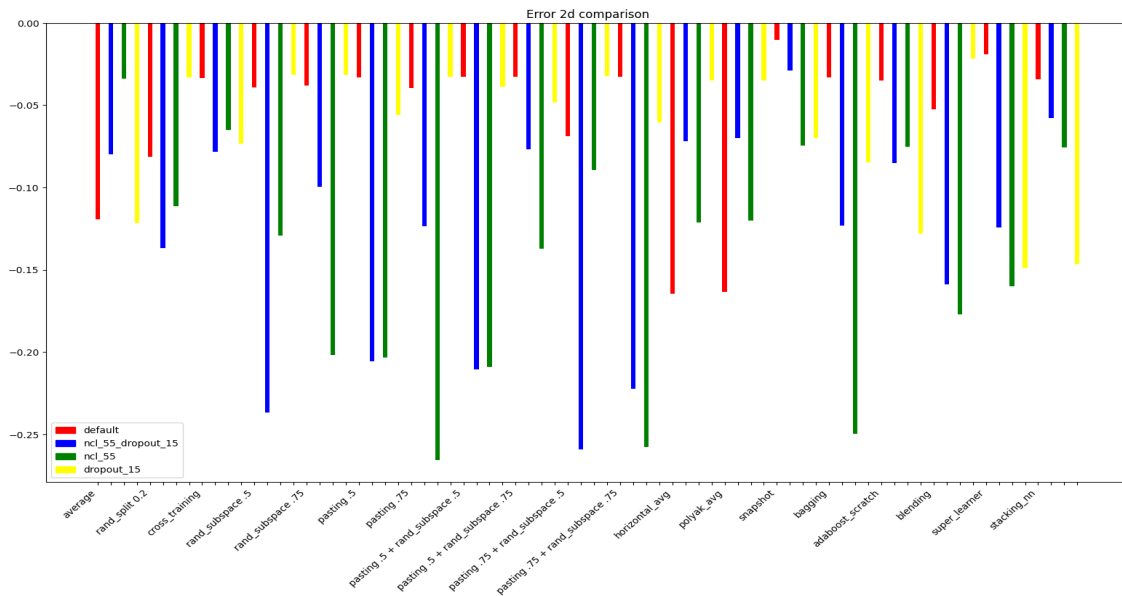


Figure 4.8: Proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' error variation results.

At this point, and according to Fig. 4.7, there have been created 56 different hybrid algorithms, which, almost all, improve on their respective default version's performance. Nonetheless, many more algorithms could be generated by merging different training data varying techniques and estimators' characteristics. However, the first did not show promising results, and the second diverges from this thesis scope.

Fig. 4.9 (a subset of Fig. 4.7) and Fig. 4.10 (a subset of Fig. 4.8) compare the effects of joining techniques of the previous level's best-performing NNE algorithms. Fig. 4.11, 4.12, and 4.13 compare the referred effects on bias, variance, and covariance, respectively.

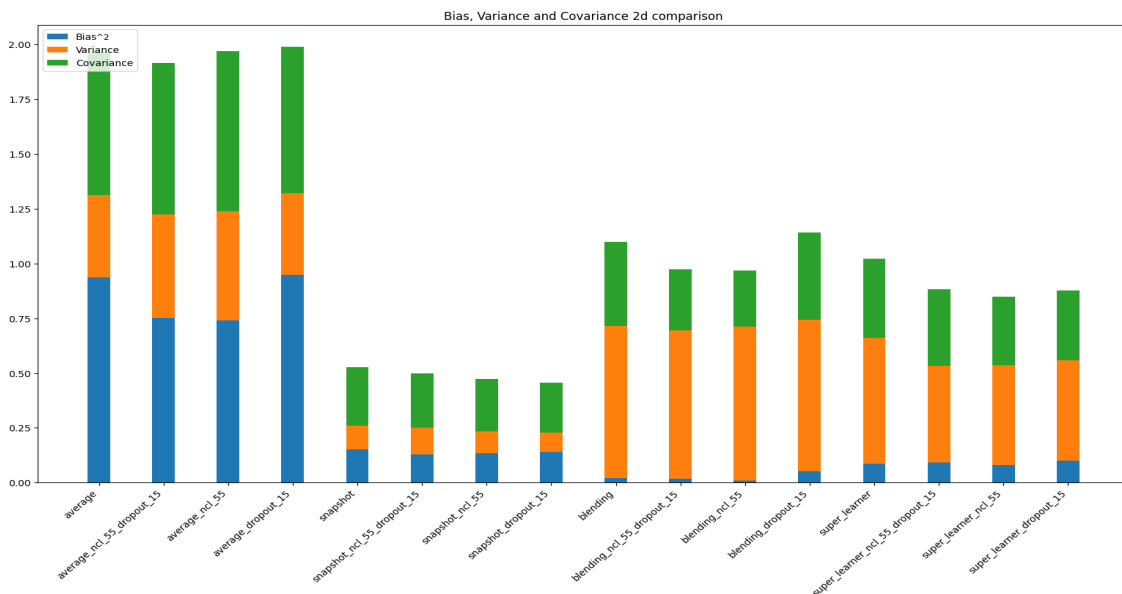


Figure 4.9: Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' results.

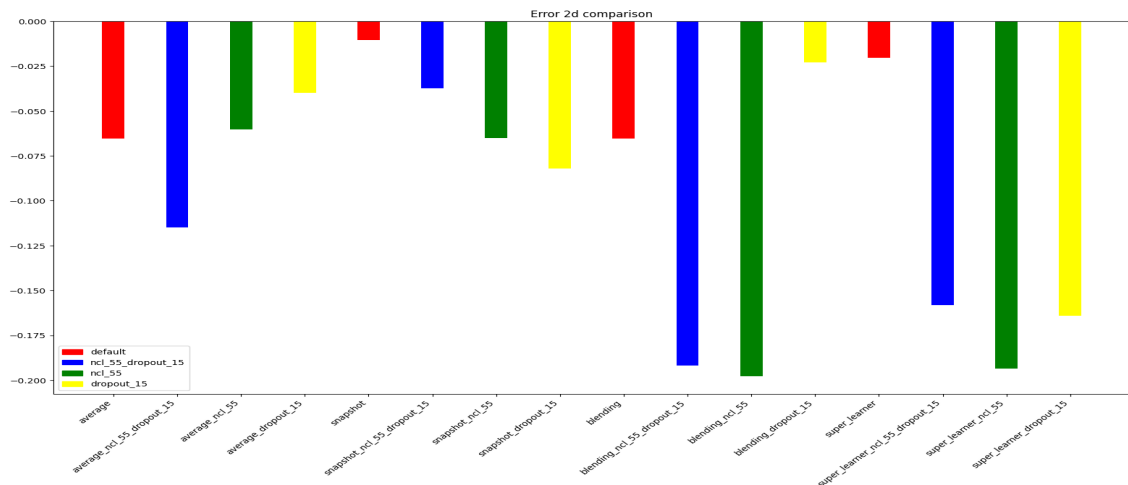


Figure 4.10: Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' **error** variation results.

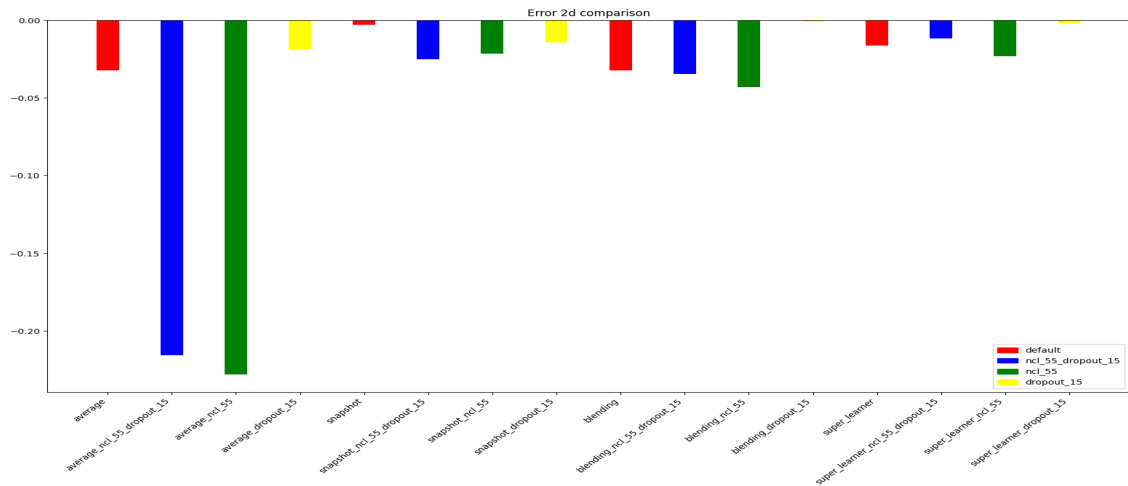


Figure 4.11: Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' **bias** variation results.

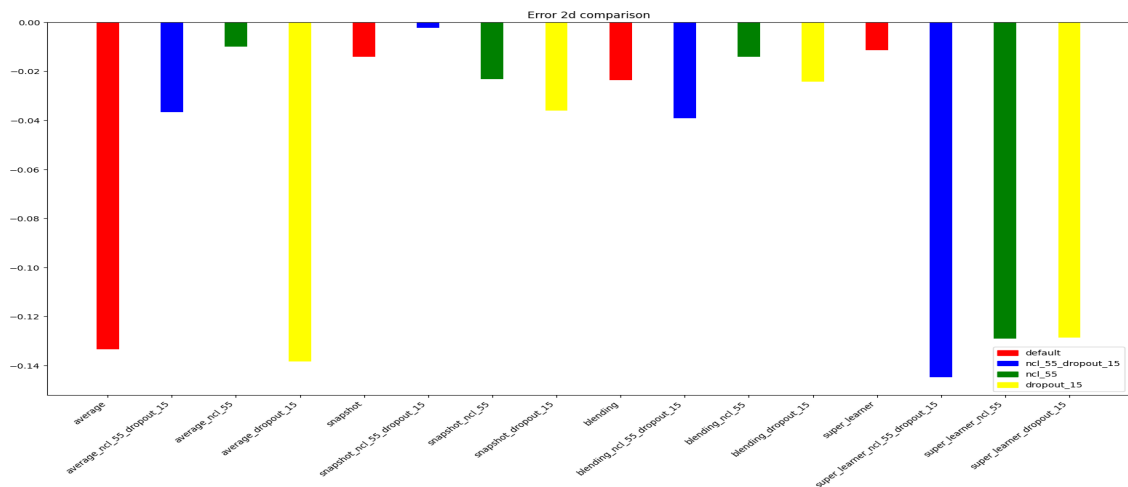


Figure 4.12: Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' **variance** variation results.

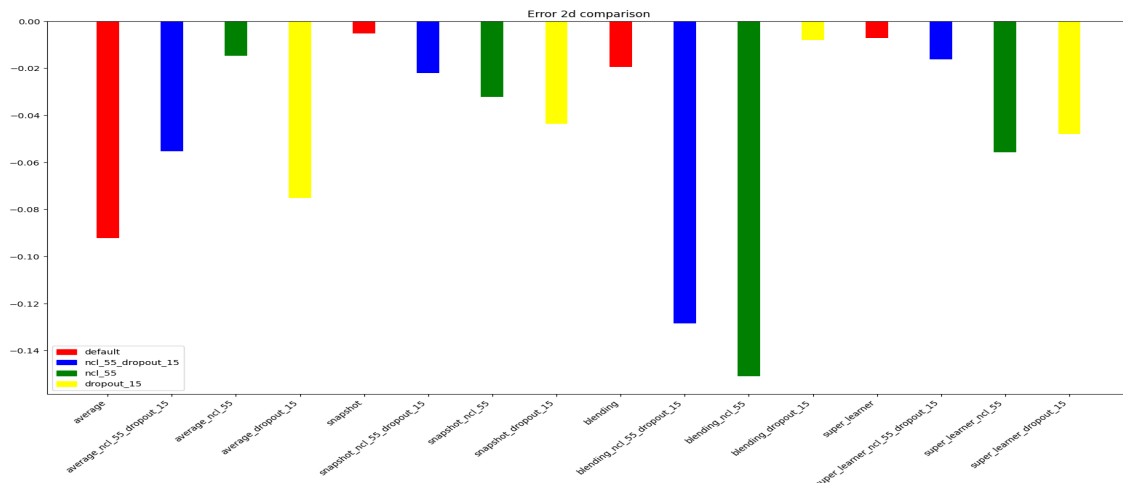


Figure 4.13: Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' **covariance** variation results.

It is clear to see that Snapshot, Blending, and Super Learner benefit, typically, from being empowered with both NCL and Dropout, both individually and in conjunction. Also, if NCL has better results than Dropout, then NCL+Dropout usually has worse performance than that of NCL but better than Dropout, and vice-versa. Regarding the Snapshot ensemble, joining it with Dropout offers the best results. Oppositely, Meta-model strategies tend to benefit from NCL alone, and when Dropout is employed, it worsens their performance. As a side note, Simple Average stays in the middle ground, meaning that it is improved the most by using NCL with Dropout.

Diving deeper into each error component, NCL alone achieves the best results in bias reduction comparing to Dropout, either alone or in combination. The case in which it did not was on Snapshot ensemble, ending in second place and closely behind NCL+Dropout. Looking at variance, typically, Dropout reduces it the most in Simple Average and Snapshot ensembles, while Meta-model approaches prefer the NCL+Dropout combination. Finally, covariance in Meta-model strategies is reduced the most by NCL, whether in Snapshot methods Dropout gives the best results.

Regarding the specific numerical results (Tables 4.7 and 4.8), *snapshot\_dropout\_15* compared to *snapshot* reduced error by 13.5%, *blending\_dropout\_15* compared to *blending* reduced error by 12.1%, and *super\_learner\_dropout\_15* compared to *super\_learner* reduced error by 16.9%.

Algorithm	$Bias^2$	Variance	Covariance	Error
average	0.936	0.376	0.653	1.965
average_ncl_55_dropout_15	0.753	0.472	0.69	1.915
average_ncl_55	0.74	0.499	0.731	1.97
average_dropout_15	0.949	0.371	0.67	1.99

Table 4.7: Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' error numerical results - 1.



Algorithm	$Bias^2$	Variance	Covariance	Error
snapshot	0.152	0.109	0.266	0.526
snapshot_ncl_55_dropout_15	0.13	0.12	0.249	0.499
snapshot_ncl_55	0.134	0.1	0.239	0.472
snapshot_dropout_15	0.141	0.087	0.227	0.455
blending	0.02	0.693	0.387	1.101
blending_ncl_55_dropout_15	0.018	0.678	0.278	0.974
blending_ncl_55	0.01	0.703	0.255	0.968
blending_dropout_15	0.052	0.693	0.398	1.143
super_learner	0.086	0.574	0.361	1.022
super_learner_ncl_55_dropout_15	0.091	0.441	0.352	0.884
super_learner_ncl_55	0.079	0.457	0.313	0.849
super_learner_dropout_15	0.101	0.457	0.32	0.878

Table 4.8: Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' error numerical results - 2.

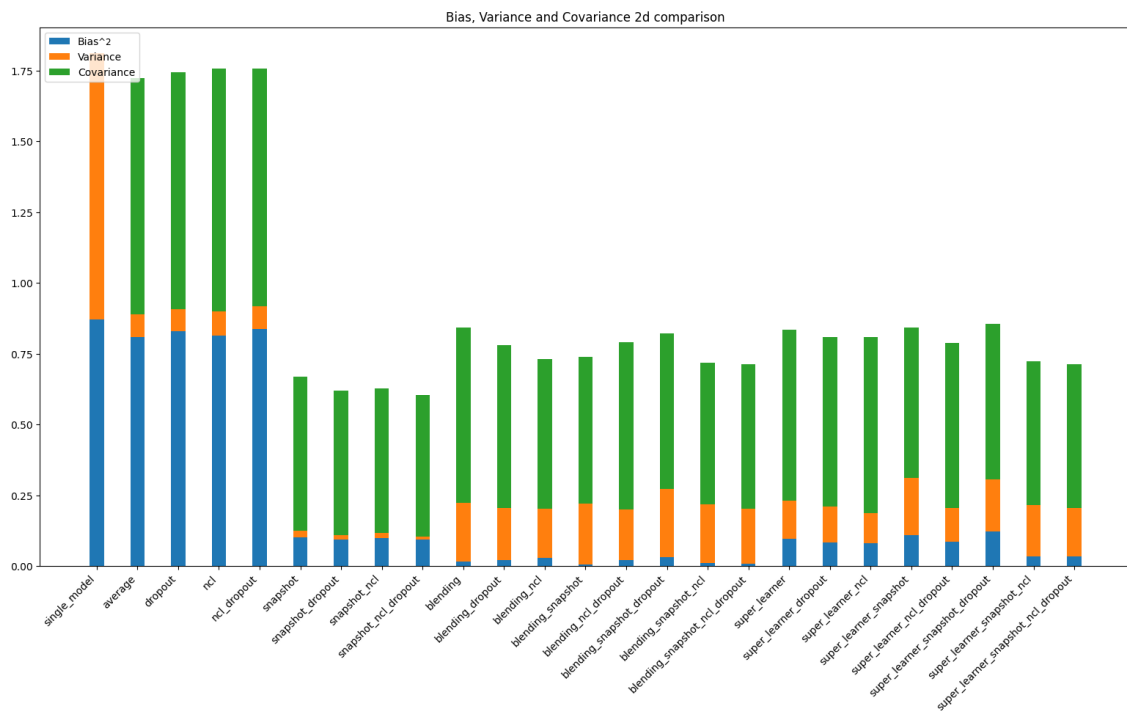


Figure 4.14: Best-performing proposed hybrid NNE algorithms' results.

Fig. 4.14 shows the results of combining all compatible previous level's best-performing NNE algorithms and the most promising approaches. Results show that Snapshot (stream models) benefit the most from employing Dropout since it acts from the beginning of their training while NCL has too few epochs, close to the training run's end, to promote mutual model diversity and lower predictions overall correlation. On the other hand, independent models benefit the most from NCL because it promotes mutual model diversity from the ground up.

Looking at individual methods, Dropout and NCL had similar performance outcomes as both finished last. The latter decreased bias the most while the former reduced variance and covariance

slightly more. Snapshot performs the best by reducing covariance ponderously and squashing bias and variance, thus acting as a bias/variance reduction algorithm. Blending, which ended second performance-wise, reduced covariance, but more importantly, smashed bias at the expense of a more pronounced increase in variance hence categorized as a bias reduction algorithm. Super Learner acts as the middle ground between Snapshot and Blending by reducing bias more than the former but less than the latter and variance otherwise. Its covariance reduction was similar to that of Blending.

Examining the framework's combination mechanism, Dropout adds to each algorithm's estimators, Dropout layers with a specific  $p$  value. NCL trains the algorithm's estimators according to the initially proposed method, and Snapshot generates many stream estimators. Optionally, once harvested, the stream estimators can be trained with an NCL policy to promote diversity regarding the original model. NCL can also be used without Snapshot. In this case, the base learners are entirely independent. Dropout, Snapshot, and NCL (combined) algorithms' estimators may be mutually integrated with the Simple Average approach or merged with a linear Meta-model algorithm. If Blending is used, the estimators make holdout set predictions that fit the Meta-model. On the other hand, if Super Learner is adopted, the estimators make out-of-fold predictions amid K-fold Cross-validation that fit the Meta-model.

Turning attention to the combined algorithms results, *snapshot\_ncl\_dropout* was the best across the board, performance-wise, since (1) Snapshot already possessed the best individual model performance and (2) NCL and Dropout further helped in lowering every error component. Regarding Blending, merging it with Dropout or NCL reduces variance and covariance but raises bias, especially with NCL.

On the other hand, merging Blending with Snapshot lowers bias and covariance but increases variance. So, the various Blending combination possibilities typically exhibit the referred combined characteristics that contribute to *blending\_snapshot\_ncl\_dropout* having the best performance in the subset of Blending techniques.

Combining Super Learner with Dropout lowers bias and variance, and combining it with NCL lowers both components even further, in particular, variance at the expense of slightly raising covariance. Merging Super Learner with Snapshot strongly reduced covariance but increased bias and, especially, variance. Consequentially, the multiple Super Learner combination possibilities typically exhibit the referred combined characteristics that contribute to *super\_learner\_snapshot\_ncl\_dropout* having the best performance in the subset of Super Learner techniques.

Regarding the specific numerical results (Table 4.9), *snapshot\_ncl\_dropout* compared to *snapshot* reduced error by 10.1%, *blending\_snapshot\_ncl\_dropout* compared to *blending* reduced error by 15.6%, and *super\_learner\_snapshot\_ncl\_dropout* compared to *super\_learner* reduced error by 14.6%.

Algorithm	$Bias^2$	Variance	Covariance	Error
single_model	0.87	0.941	0.0	1.812
dropout	0.83	0.077	0.841	1.748
ncl	0.814	0.085	0.863	1.762
ncl_dropout	0.837	0.081	0.843	1.761
snapshot	0.101	0.025	0.549	0.674
snapshot_dropout	0.093	0.017	0.511	0.621
snapshot_ncl	0.1	0.017	0.514	0.631
snapshot_ncl_dropout	0.093	0.012	0.501	0.606
blending	0.016	0.208	0.622	0.846
blending_dropout	0.021	0.184	0.579	0.784
blending_ncl	0.03	0.172	0.533	0.735
blending_snapshot	0.006	0.215	0.52	0.741
blending_ncl_dropout	0.02	0.18	0.592	0.792
blending_snapshot_dropout	0.032	0.241	0.55	0.824
blending_snapshot_ncl	0.012	0.207	0.502	0.72
blending_snapshot_ncl_dropout	0.007	0.196	0.511	0.714
super_learner	0.098	0.133	0.608	0.839
super_learner_dropout	0.084	0.127	0.6	0.811
super_learner_ncl	0.082	0.107	0.624	0.813
super_learner_snapshot	0.11	0.201	0.535	0.845
super_learner_ncl_dropout	0.087	0.117	0.587	0.791
super_learner_snapshot_dropout	0.122	0.185	0.552	0.859
super_learner_snapshot_ncl	0.035	0.182	0.509	0.726
super_learner_snapshot_ncl_dropout	0.034	0.173	0.51	0.716

Table 4.9: Best-performing proposed hybrid NNE algorithms' numerical results.

Analyzing the individual dataset error results (Fig. 4.15 and 4.16), it is clear that most algorithms consistently perform across the datasets, either good or bad.

It is important to note that there was a case in which almost every proposed algorithm performed worse than the baselines but since this was an isolated case, it is considered an outlier.

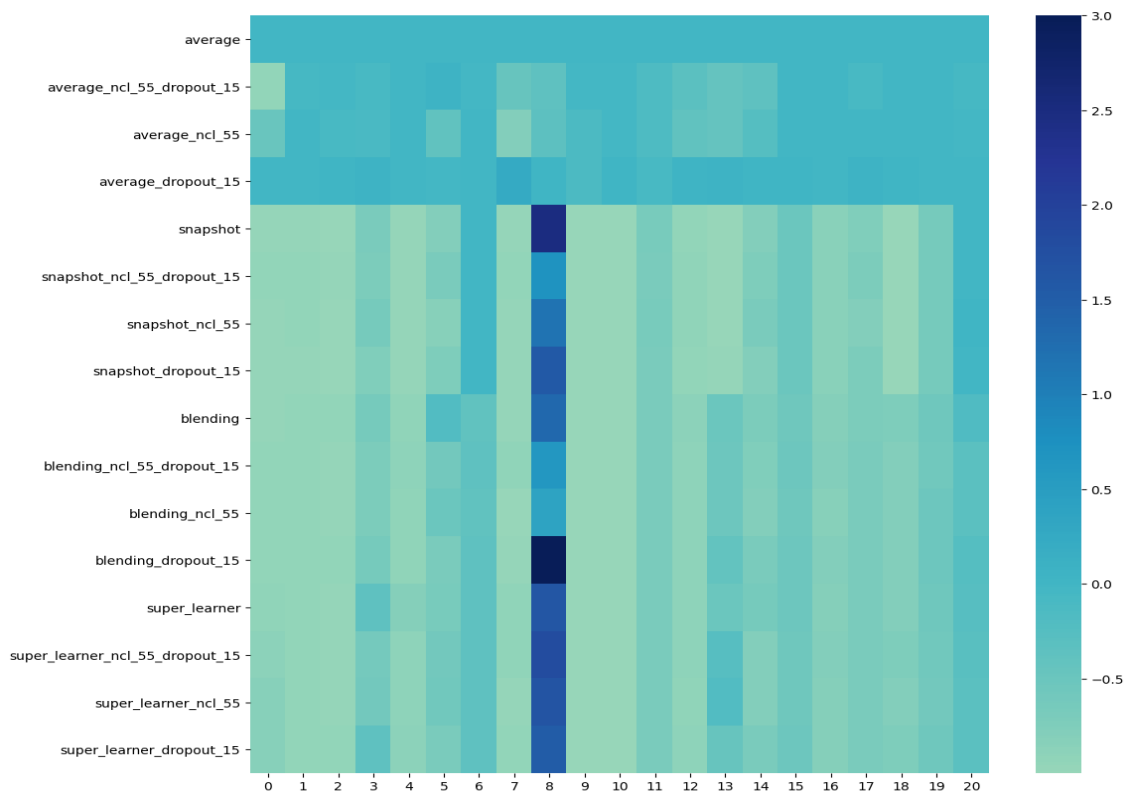


Figure 4.15: Best-performing proposed Dropout, NCL, Dropout+NCL empowered NNE algorithms' heatmap error results.

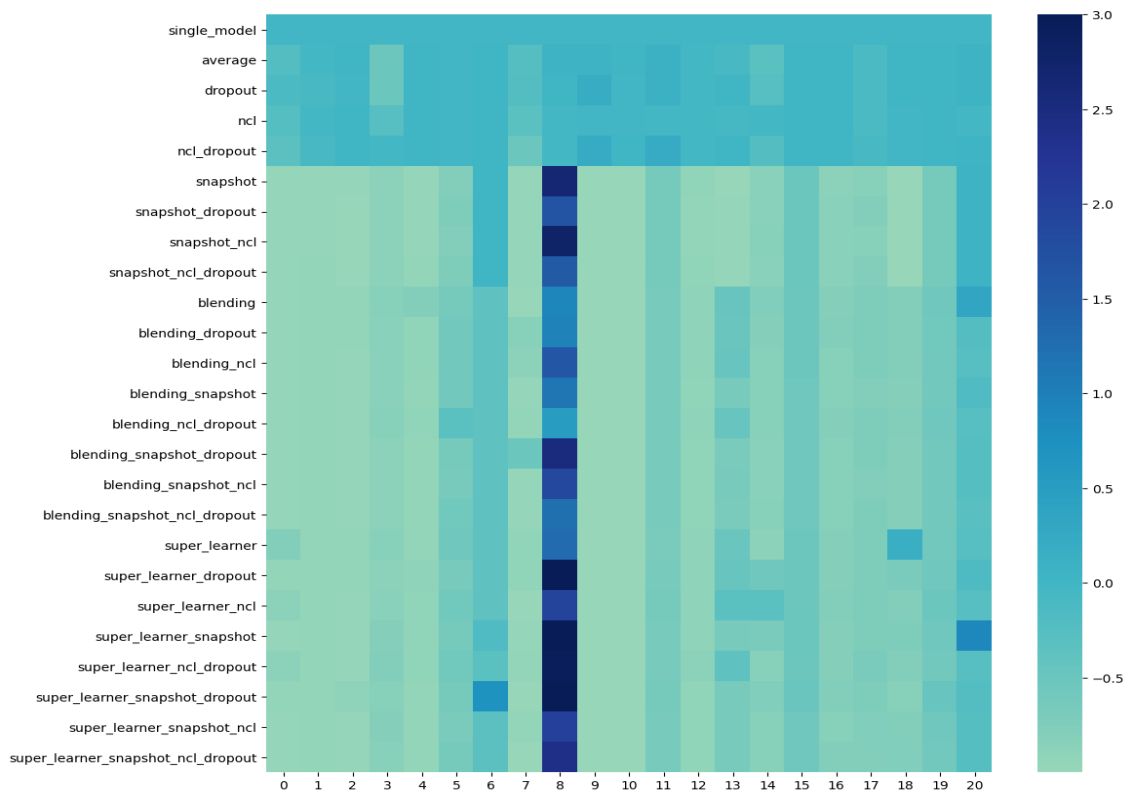


Figure 4.16: Best-performing proposed hybrid NNE algorithms' heatmap error results.

On the whole, having explored and discussed the merits of combining the previous layer's best-performing NNE algorithms (Snapshot, Blending, and Super Learner) and the most promising approaches (NCL and Dropout), it is possible to state that adding to the existing ensemble strategies:

- NCL/Dropout, both individually and in conjunction, improves results;
- NCL+Dropout works best most of the time, and when it doesn't, NCL alone has a slight edge, except for Snapshot ensembles where Dropout alone works best;
- NCL gives the best bias results, except in Snapshot and Blending, where NCL+Dropout works best;
- Dropout gives the best variance and covariance results, except in Meta-model strategies, where NCL works best at the expense of, occasionally, lightly increasing covariance.

Also:

- Snapshot reduces bias and especially variance and covariance, while Blending reduces bias more pronouncedly at the expense of increased variance. Super Learner stays balanced, as each error component reduction stays between that of Snapshot and Blending;
- Snapshot displays the best results with Dropout since it acts from the beginning of their training while NCL has too few epochs, close to the training run's end, to promote mutual model diversity. Independent models benefit the most from NCL because it promotes mutual model diversity from the ground up;
- *snapshot\_ncl\_dropout* is the best algorithm since (1) Snapshot already possessed the best individual model performance, and (2) NCL+Dropout further helped lower complementary error components. *blending\_snapshot\_ncl\_dropout* was the best blending subset algorithm, and *super\_learner\_snapshot\_ncl\_dropout* the best Super Learner subset algorithm;
- every Snapshot combination outperformed all other Blending and Super Learner proposed architectures. Nonetheless, it does not make Snapshot the absolute winner in all circumstances. Since Blending and Super Learner offer a second space generalization, improving their current Meta-model or adding subsequent generalization levels is a possible avenue into improving results. Furthermore, given that the results difference between Snapshot and Blending/Super Learner best-performing combination algorithms are not significant, it is reasonable to assume that subsequent improvements to these Meta-model variants would likely outperform Snapshot. A counterargument is to harvest Snapshot's models at detached epochs and improve the existing Snapshot's varying LR policy.

Bottom line, experimental results confirm performance increases from combining multiple established algorithms as the developed ensembles showed better performance than the original constituent counterparts. Merging the best-performing methods resulted in a lower global error value, as each technique was complementary in lowering one or more ensemble error components.

### 4.3 Statistical Validation

Friedman test is employed to ascertain if repeated and related measurements consistently follow the same distribution [180]. Specifically, explore the statistically significant difference between the means of more than two groups with the same subject observation to ensure random event independent observations.

Its null hypothesis is that the multiple paired samples follow the same distribution. This assumption's rejection means that at least one paired sample follows a different distribution.

Looking at Table 4.10, particularly the  $p$  value, every experiment rejected the null hypothesis by surpassing the defined type I error of 0.05. So, it proved that the observation values (ensemble architecture's errors) from multiple runs (different datasets) have different means, follow different distributions, and, inherently, are statically valid. In other words, there is enough proof to conclude that different types of ensemble algorithms lead to statistically significant differences in their global error values.

Experience	<i>stat</i>	<i>p</i>
Fig. 4.1	206.254	8.845e-28
Fig. 4.3	112.976	5.798e-07
Fig. 4.7	1028.134	2.369-167
Fig. 4.9	144.651	2.773e-23
Fig. 4.14	215.686	3.001e-33

Table 4.10: Friedman tests

Given that the experiments'  $p$ -values are statistically significant, the Nemenyi posthoc test, which returns the  $p$ -values for each pairwise mean comparison, can be executed to determine the specific groups with different means [181]. Fig. 4.17, 4.18, 4.19, 4.20, and 4.21 clearly show that many algorithms have a high pairwise  $p$  value denoting they have statistically similar means. The reason is that most algorithms' global error values are clustered around two groups, high and low values, without many in between. Naturally, algorithms inside those clusters will have a high  $p$  value between one another, while comparing algorithms that belong to different clusters show a low mutual  $p$  value. Note that high and low values mean better performing and worse performing algorithms, respectively.

Note that the right  $y$ -axis of heatmap graphs refers to the mean similarity percentage between a given line's and column's algorithm. 0 means they are statistically non-similar, and 1.0 means they are statistically equal.



Figure 4.17: Individual ensemble Nemenyi test matrix.

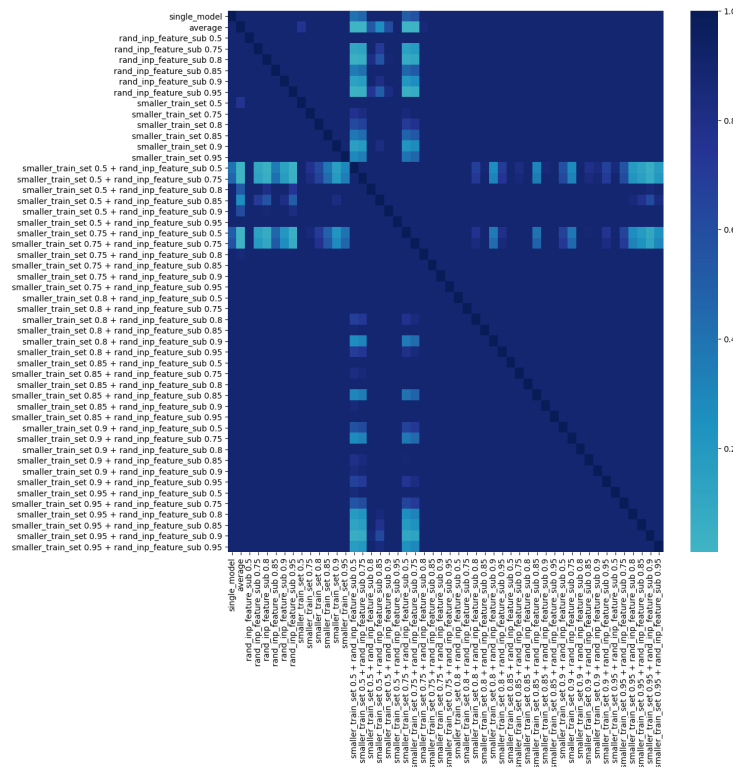


Figure 4.18: Different configurations of Pasting, Random Subspace and Random Patches algorithms' Nemenyi test matrix.

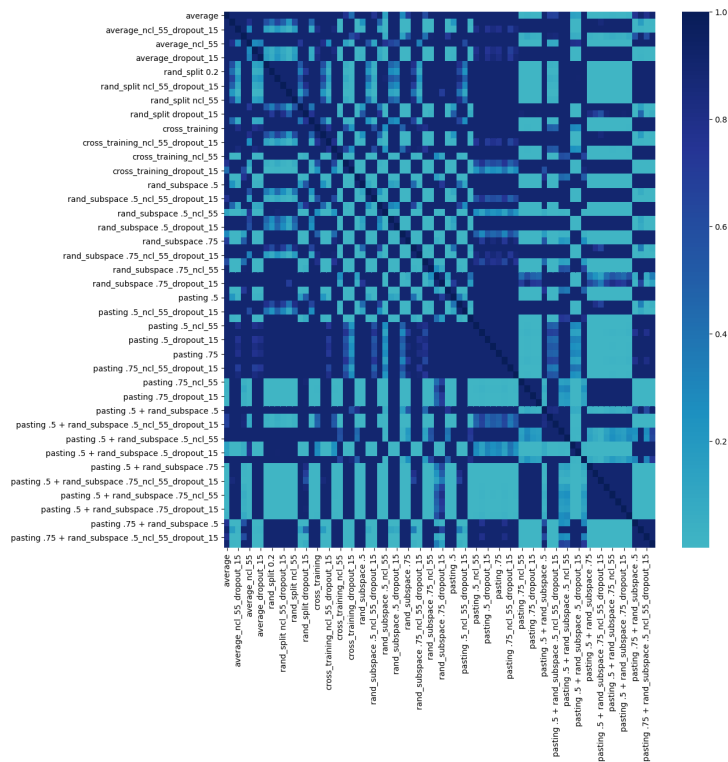


Figure 4.19: Proposed Dropout, NCL, and Dropout+NCL empowered NNE Nemenyi algorithms' test matrix.



Figure 4.20: Best-performing proposed Dropout, NCL, and Dropout+NCL empowered NNE algorithms' Nemenyi test matrix.



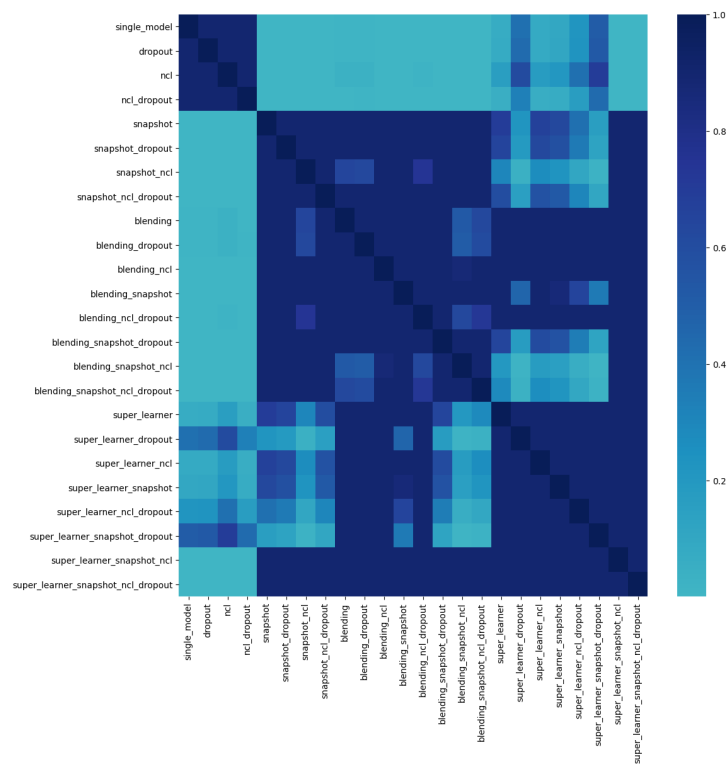


Figure 4.21: Best-performing proposed hybrid NNE algorithms’ Nemenyi test matrix.

## 4.4 Time analysis

This section provides a time analysis of the proposed models. Note that a performance increase is worth considering the application domain associated costs. For example, spending another hour training to forecast tomorrow's weather better is quite useful. On the other hand, spending the same hour training a predictive model on what a stock is worth in 15 minutes is entirely useless.

Looking at Tables 4.11 and 4.12, it is clear that ensembles increase the time spent compared to a single model. In addition to this, one can derive other conclusions. Super Learner, NCL, AdaBoost\_nn, and Stacking\_nn take the longest time to execute, in this precise order. In fact, the first takes almost twice as much as the last. Besides, NCL tends to take, on average, three to five times longer to train compared to the same algorithm without the NCL segment.

Given that performance increases are only worthwhile if the respective time increase is not prohibitory, combining algorithms that include Super Learner, NCL, AdaBoost\_nn, or Stacking\_nn may be considered not advantageous comparing to other similar performing algorithms which execute faster (Tables 4.4, 4.7, 4.8 4.9, 4.11, and 4.12). However, this is just a rough guide as only specific dataset performance, time results, and personal judgment give the final decision.

name	time (sec)
single_model_neural_network	0.966
rand_split 0.2	4.799
cross_training	4.811
rand_subspace .5	5.079
rand_subspace .75	5.079
pasting .5	4.585
pasting .75	5.101
pasting .5 + rand_subspace .5	4.994
pasting .5 + rand_subspace .75	4.555

Table 4.11: Comparison of established and proposed algorithms time spent - 1.

name	time (sec)
pasting .75 + rand_subspace .5	4.830
pasting .75 + rand_subspace .75	4.807
horizontal_avg	2.301
polyak_avg	3.461
snapshot	1.720
snapshot_dropout	1.781
snapshot_ncl	2.015
snapshot_ncl_dropout	1.975
average	6.126
dropout_20	4.224
ncl	13.227
ncl_dropout	13.176
bagging	5.036
adaboost_nn_scratch	20.391
adaboost_nn	18.852
blending	2.805
blending_dropout	2.868
blending_ncl	12.918
blending_snapshot	2.315
blending_ncl_dropout	13.222
blending_snapshot_dropout	2.007
blending_snapshot_ncl	2.132
blending_snapshot_ncl_dropout	2.306
super_learner	10.810
super_learner_dropout	9.956
super_learner_ncl	77.420
super_learner_snapshot	7.544
super_learner_ncl_dropout	78.247
super_learner_snapshot_dropout	7.401
super_learner_snapshot_ncl	96.507
super_learner_snapshot_ncl_dropout	95.435
stacking_nn	10.187
single_model_decision_tree	0.104
adaboost_default	0.076
gradient_boosting	0.043
gradient_boosting_hist	0.137
lgbm	2.626
catboost	0.174
xgboost	1.482
rand_forest	0.084
extra_trees	0.07

Table 4.12: Comparison of established and proposed algorithms time spent - 2.

# Chapter 5

## Conclusions and Future Work

### Contents

---

<b>5.1 Hypothesis Revisited</b> . . . . .	<b>75</b>
<b>5.2 Contributions</b> . . . . .	<b>76</b>
<b>5.3 Social Impact</b> . . . . .	<b>77</b>
<b>5.4 Future Work</b> . . . . .	<b>77</b>

---

This thesis treated a multitude of topics and their respective intricacies. This chapter revisits the initial formulated hypothesis, presents the final drawn results' conclusions, enumerates the main contributions, and elaborates on future work ideas.

Initially, the primary identified problem was a lack of systematic researches, formulated hypotheses, and theorized proposals about NNEs. Concretely, how to best combine the most complementary existing strategies (paired strengths that most reduce each ensemble error component in conjunction) by decomposing the ensemble error in bias, variance, and covariance to harvest each strategies' best qualities.

So, the fundamental idea was to prove the advantage of combining already established NNE strategies in Regression problems that improve the performances of its constituent models and the respective original architectures they are based on.

### 5.1 Hypothesis Revisited

Looking back at the original formulated hypothesis:

Is it possible to improve the existing state-of-the-art  
Neural Network Ensemble approaches by combining them?

## Knowing that Neural Network Ensembles allow for multiple configurations and have different characteristics, which are the most appropriate?

Regarding the first hypothesis, comprehensive experimental results with numerous datasets confirmed the theoretical assumptions by revealing a clear and meaningful performance increase from combining various algorithms, as each technique method is complementary in lowering the various ensemble error components. Furthermore, from the multitude of proposed new NNE hybrid strategies, the best-performing ones decreased the global error and increased performance, on average, from 12% to 17% versus their original constituent algorithms.

Regarding the second hypothesis, having explored and discussed the merits of the current NNE approaches, we learned that combining Snapshot, NCL, and Dropout offers the most notable overall error reduction, hence giving the best results. Additionally, adding those three algorithms to other ensemble strategies improves their performance the most. However, if one searches for specific error component reduction on specific ensemble architectures, other combinations may be advisable to avoid some technique pitfalls. e.g., NCL reduces bias more pronouncedly while Dropout reduces variance and covariance more.

Therefore, and given that the experiments were considered statically valid and trustworthy according to Friedman and Nemenyi's test results, it is proven that combining different ensemble approaches is a plausible and consistent way of improving predictive capability.

## 5.2 Contributions

All told, the main contributions are:

### **Preliminary study**

Following this contextualization, state-of-the-art generic Ensemble and NNE techniques and trends were thoroughly explored, as well as their limitations, summarized in a Preliminary Study which stands as a structured survey on its own.

### **Complete Ensemble Error Decomposition**

Performs an ensemble prediction error decomposition in bias, variance, and covariance to find the most promising ways of combining the most complementary existing strategies.

### **Innovative Algorithms**

Multiple new and innovative hybrid ensemble algorithms were theorized, modulated, adjusted, and implemented for this thesis-specific context. More concretely, and excluding varying training data techniques, more than 60 different algorithm combinations were developed.

### **Hybrid Ensemble Combining Framework**

Automatically constructs new NN Regression Ensembles that improve the performances of its constituent models and the respective original architectures they are based on. It also performs the complete ensemble error decomposition and a time analysis.

### 5.3 Social Impact

Regarding this work's broader impact, the advances presented in this thesis are not socially good or bad inherently. The proposed new methods are complementary to the currently existing ones. Likewise, their social impact is dependant on their use rather than on themselves.

### 5.4 Future Work

This thesis's results are exciting but constitute only a first step into understanding NNE algorithms and how to combine them best. Some topics can be considered for future work as we believe they have plenty of evolving potential and are sensible/possible to be explored given all that was learned, in particular:

#### **Boosting NNs**

Pursue research efforts into converting Decision-Tree specific Ensemble architectures into NNEs. Namely, this avenue refers to Boosting strategies since Random Forest (and its variants) (1) already has its NN compatible characteristics harvested, and (2) its whole training strategy is designed for Decision Trees. On the other hand, Boosting gave exceptional results and its training strategy is entirely compatible with NNs, as seen with AdaBoost. However, other Boosting implementations such as Gradient Boosting, LightGBM, CatBoost, and XGBoost have not yet been integrated with NNs. Altering these libraries from the ground up would create hybrid NN Boosting Ensemble algorithms with improved performance, e.g. *XGBoost\_Snapshot\_NCL\_Dropout*.

#### **Parallelize NCL**

Despite NCL having better results than the Simple Average Ensemble, it took longer to execute. A parallel strategy could be employed to scale up the algorithm, splitting it into several threads and changing the NN's training strategy. Although it could improve overall performance, it would undoubtedly improve significantly the time consumed.

#### **Extending to other estimators**

All the work presented in this thesis revolved around NNs. However, all the theory and rationale holds for other types of ensemble estimators, e.g., Decision Trees or KNN. Therefore extending this work to other base models, namely unstable ones, is a promising avenue for developing other research endeavors.

#### **Classification extension**

Despite the developed framework being devised for regression modeling problems, it may also be used as a proxy to generate ensembles for classification problems. This would require a few alterations specifically in the error decomposition formula, dataset treatment, and NN specifications.

# References

- [1] Kevin P Murphy. *Machine learning: a probabilistic perspective*, volume 621485037. The MIT Press, First edition, 2012. ISBN 0262018029.
- [2] Tom Mitchell. *Machine Learning*. McGraw-Hill Education, First edition, 1997. ISBN 0070428072.
- [3] A.L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, jul 1959. ISSN 0018-8646. doi: 10.1147/rd.33.0210.
- [4] Max Kuhn and Kjell Johnson. *Applied predictive modeling*. Springer, First edition, jan 2013. ISBN 1461468493. doi: 10.1007/978-1-4614-6849-3.
- [5] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, Third edition, 2009. ISBN 0136042594.
- [6] Christopher M Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, First edition, 1996. ISBN 0198538646.
- [7] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, Second edition, 1998. ISBN 0132733501.
- [8] Ian Goodfellow. *Deep learning*. The MIT Press, First edition, 2016. ISBN 0262035613.
- [9] Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman and Hall/CRC, First edition, 2012. ISBN 1439830031.
- [10] Mark J. Van Der Laan, Eric C. Polley, and Alan E. Hubbard. Super learner. *Statistical Applications in Genetics and Molecular Biology*, 6(1), sep 2007. ISSN 15446115. doi: 10.2202/1544-6115.1309.
- [11] Jingjing Xie, Bing Xu, and Zhang Chuang. Horizontal and vertical ensemble with deep representation for classification, jun 2013.
- [12] Robert T Clemen. Combining forecasts: A review and annotated bibliography. *International Journal of Forecasting*, 5(4):559–583, 1989. ISSN 0169-2070. doi: [https://doi.org/10.1016/0169-2070\(89\)90012-5](https://doi.org/10.1016/0169-2070(89)90012-5).
- [13] Tin Kam Ho. Multiple classifier combination: Lessons and next steps. In *Machine Perception and Artificial Intelligence*, volume 47, pages 171–198. World Scientific, may 2002. doi: 10.1142/9789812778147\_0007.

- [14] Michael P. Perrone. *Improving regression estimation: Averaging methods for variance reduction with extensions to general convex measure optimization*. PhD thesis, Brown University, Institute for Brain and Neural Systems, USA, 1993. UMI Order No. GAX94-07007.
- [15] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, jan 1992. ISSN 0899-7667. doi: 10.1162/neco.1992.4.1.1.
- [16] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. *Advances in Neural Information Processing Systems*, 7:231–238, 1995.
- [17] Gavin Brown, Jeremy Wyatt, Rachel Harris, and Xin Yao. Diversity creation methods: A survey and categorisation. *Information Fusion*, 6(1):5–20, 2005. ISSN 15662535. doi: 10.1016/j.inffus.2004.04.004.
- [18] Ury Naftaly, Nathan Intrator, and David Horn. Optimal ensemble averaging of neural networks. *Network: Computation in Neural Systems*, 8(3):283–296, jan 1997. ISSN 0954-898X. doi: 10.1088/0954-898x\_8\_3\_004.
- [19] G.S. Sebestyen. *Learning machines foundations of trainable pattern-classifying systems*. Macmillan, First edition, 1962. ISBN B0006AY384.
- [20] L. Kanal. Patterns in pattern recognition. *IEEE Transactions on Information Theory*, 20(6): 697–722, 1974. doi: 10.1109/TIT.1974.1055306.
- [21] Marvin Minsky. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI Magazine*, 12(2):34–34, jun 1991. ISSN 2371-9621. doi: 10.1609/AIMAG.V12I2.894.
- [22] Thomas G. Dietterich. Machine-learning research; four current directions. *AI Magazine*, 18(4):97–137, dec 1997. ISSN 07384602.
- [23] Pedro Domingos. A unified bias-variance decomposition and its applications. *IN PROC. 17TH International Conference on Machine Learning*, pages 231—238, 2000.
- [24] Gavin Brown, Jeremy L. Wyatt, and Peter Tiño. Managing diversity in regression ensembles. *Journal of Machine Learning Research*, 6(55):1621–1650, 2005.
- [25] Department of mathematical sciences da university of alabama in huntsville: Covariance and correlation, 2020. <https://www.randomservices.org/random/expect/Covariance.html>, last accessed on 2020-12-02.
- [26] Naonori Ueda and Ryohei Nakano. Generalization error of ensemble estimators. In *IEEE International Conference on Neural Networks - Conference Proceedings*, volume 1, pages 90–95. IEEE, 1996. doi: 10.1109/icnn.1996.548872.
- [27] Department of mathematical sciences da university of alabama in huntsville: Sample correlation and regression, 2020. <https://www.randomservices.org/random/sample/Covariance.html>, last accessed on 2020-12-02.
- [28] Wolfram mathworld: Covariance, 2020. <https://www.quora.com/What-is-the-difference-between-the-R-gbm-gradient-boosting-machine-and-xgboost-extreme-gradient-boosting/answer/Tianqi-Chen-1>, last accessed on 2020-12-02.



- [29] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, oct 2001. ISSN 08856125. doi: 10.1023/A:1010933404324.
- [30] E. M. Kleinberg. Stochastic discrimination. *Annals of Mathematics and Artificial Intelligence*, 1(1-4):207–239, sep 1990. ISSN 10122443. doi: 10.1007/BF01531079.
- [31] R. Kohavi and D. Wolpert. Bias plus variance decomposition for zero-one loss functions. In *ICML*, volume 96, pages 275–83, 1996.
- [32] David H. Wolpert. On bias plus variance. *Neural Computation*, 9(6):1211–1243, aug 1997. ISSN 08997667. doi: 10.1162/neco.1997.9.6.1211.
- [33] Peter Bartlett, Yoav Freund, Wee Sun Lee, and Robert E. Schapire. Boosting the margin: a new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5): 1651–1686, oct 1998. ISSN 0090-5364. doi: 10.1214/aos/1024691352.
- [34] Vincent Pisetta. *New Insights into Decision Trees Ensembles*. PhD thesis, Lyon, 2, 2012.
- [35] Eun Bae Kong and Thomas G. Dietterich. Error-correcting output coding corrects bias and variance. In *Machine Learning Proceedings 1995*, pages 313–321. Elsevier, jan 1995. doi: 10.1016/b978-1-55860-377-6.50046-3.
- [36] Jerome H. Friedman. On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997. ISSN 13845810. doi: 10.1023/A:1009778005914.
- [37] L. Breiman. Arcing classifier (with discussion and a rejoinder by the author). *Annals of Statistics*, 26(3):801–849, 1998.
- [38] Gareth M. James. Variance and bias for general loss functions. *Machine Learning*, 51(2): 115–135, may 2003. ISSN 08856125. doi: 10.1023/A:1022899518027.
- [39] P.M. Granitto, P.F. Verdes, and H.A. Ceccatto. Neural network ensembles: Evaluation of aggregation algorithms. *Artificial Intelligence*, 163(2):139–162, apr 2005. ISSN 00043702. doi: 10.1016/j.artint.2004.09.006.
- [40] Peter Sollich, Peter Sollich, and Anders Krogh. Learning with ensembles: How over-fitting can be useful. *Proceedings of the 8th International Conference on Neural Information Processing Systems*, pages 190–196, 1995.
- [41] Lior Rokach. *Pattern Classification Using Ensemble Methods (Series in Machine Perception and Artificial Intelligence)*. World Scientific Publishing Company, First edition, 2009. ISBN 0262527014.
- [42] Kai Ming Ting and Ian H. Witten. Stacked generalization: when does it work? *International Joint Conference on Artificial Intelligence*, pages 866–871, 1997. ISSN 1170-487X.
- [43] David H Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992. ISSN 0893-6080. doi: [https://doi.org/10.1016/S0893-6080\(05\)80023-1](https://doi.org/10.1016/S0893-6080(05)80023-1).
- [44] Georgios Sigletos, Georgios Paliouras, and Constantine D. Spyropoulos. Combining information extraction systems using voting and stacked generalization. *Journal of Machine Learning Research*, 6:1751–1782, 2005. doi: 10.5555/1046920.1194903.

- [45] Cha Zhang and Yunqian Ma. *Ensemble Machine Learning: Methods and Applications*. Springer, First edition, 2012. ISBN 1441993258.
- [46] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990. ISSN 01628828. doi: 10.1109/34.58871.
- [47] Sean Tao. Deep neural network ensembles. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11943 LNCS:1–12, apr 2019.
- [48] Thomas G. Dietterich. Ensemble methods in machine learning. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1857 LNCS, pages 1–15. Springer Verlag, 2000. ISBN 3540677046. doi: 10.1007/3-540-45014-9\_1.
- [49] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998. ISSN 01628828. doi: 10.1109/34.709601.
- [50] Bart Bakker and Tom Heskes. Clustering ensembles of neural network models. *Neural Networks*, 16(2):261–269, mar 2003. ISSN 08936080. doi: 10.1016/S0893-6080(02)00187-9.
- [51] Qiang Fu, Shang Xu Hu, and Sheng Ying Zhao. Clustering-based selective neural network ensemble. *Journal of Zhejiang University: Science*, 6 A(5):387–392, may 2005. ISSN 10093095. doi: 10.1631/jzus.2005.A0387.
- [52] Barak A. Pearlmutter and Ronald Rosenfeld. Chaitin-kolmogorov complexity and generalization in neural networks. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 3, pages 925–931. Morgan-Kaufmann, 1990.
- [53] Sherif Hashem. Optimal linear combinations of neural networks. *Neural Networks*, 10(4): 599–614, jun 1997. ISSN 08936080. doi: 10.1016/S0893-6080(96)00098-6.
- [54] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996. ISSN 08856125. doi: 10.1007/bf00058655.
- [55] Kaggle ensembling guide, 2020. <https://mlwave.com/kaggle-ensembling-guide/>, last accessed on 2020-11-20.
- [56] Francois Chollet. *Deep Learning with Python*. Manning Publications, First edition, 2017. ISBN 1617294433.
- [57] P. Granitto, P. Verdes, H. Ceccatto, and H. Navone. Selecting diverse members of neural network ensembles. *Neural Networks, Brazilian Symposium on*, pages 255–260, jan 2000. ISSN 1522-4899. doi: 10.1109/SBRN.2000.889748.
- [58] Leo Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36(1):85–103, 1999. ISSN 08856125. doi: 10.1023/a:1007563306331.
- [59] Paul Cohen. *Empirical methods for artificial intelligence*. MIT Press, Cambridge Mass., First edition, 1995. ISBN 0262032254.

- [60] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to Statistical Learning*, volume 7. Springer, Second edition, 2000. ISBN 1-4614-7137-0. doi: 10.1007/978-1-4614-7138-7.
- [61] Thomas J. DiCiccio and Bradley Efron. Bootstrap confidence intervals. *Statistical Science*, 11(3):189–212, 1996. ISSN 08834237. doi: 10.1214/ss/1032280214.
- [62] Bradley Efron and Robert Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall/CRC, First edition, 1994. ISBN 0412042317.
- [63] Gilles Louppe and Pierre Geurts. Ensembles on random patches. In *Machine Learning and Knowledge Discovery in Databases*, volume 7523 LNAI, pages 346–361. Springer, Berlin, Heidelberg, 2012. ISBN 3642334597. doi: 10.1007/978-3-642-33460-3\_28.
- [64] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, New York, NY, Second edition, 2009. ISBN 0-387-84857-0. doi: 10.1007/978-0-387-84858-7.
- [65] Chao Chen, Andy Liaw, and Leo Breiman. Using random forest to learn imbalanced data. *University of California, Berkeley*, 2004.
- [66] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, apr 2006. ISSN 08856125. doi: 10.1007/s10994-006-6226-1.
- [67] M. Kearns. Thoughts on hypothesis boosting. Unpublished, dec 1988.
- [68] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, jun 1990. ISSN 0885-6125. doi: 10.1007/bf00116037.
- [69] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 904, pages 23–37. Springer Verlag, 1995. ISBN 3540591191. doi: 10.1007/3-540-59119-2\_166.
- [70] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, sep 1995. ISSN 10902651. doi: 10.1006/inco.1995.1136.
- [71] L. Valiant. Probably approximately correct: Nature’s algorithms for learning and prospering in a complex world. *Common Knowledge*, pages 1–1, 2015.
- [72] Francisco Azuaje. Data mining: Practical machine learning tools and techniques. *BioMedical Engineering OnLine*, 5(1):1–2, dec 2006. doi: 10.1186/1475-925x-5-51.
- [73] Robert E. Schapire. Explaining adaboost. In *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, pages 37–52. Springer Berlin Heidelberg, jan 2013. ISBN 3642411366. doi: 10.1007/978-3-642-41136-6\_5.
- [74] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and Its Interface*, 2(3):349–360, 2009. ISSN 19387989. doi: 10.4310/sii.2009.v2.n3.a8.
- [75] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, dec 1999. ISSN 08856125. doi: 10.1023/A:1007614523901.

- [76] H. Drucker and C. Cortes. Boosting decision trees. In *Advances in Neural Information Processing Systems*, pages 479–485, 1995.
- [77] H. Drucker. Improving regressors using boosting techniques. *ICML*, 1997.
- [78] Llew Mason, Jonathan Baxter, Peter Bartlett, and Marcus Freen. Boosting algorithms as gradient descent. *Advances in Neural Information Processing Systems*, pages 512–518, 1999. ISSN 10495258.
- [79] H. Ehrlinger and H. Ishwaran. Characterizing l2boosting. *Ann. Statist.*, 40(2):1074–1101, 2012.
- [80] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: A statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, apr 2000. ISSN 00905364. doi: 10.1214/aos/1016218223.
- [81] Leo Breiman. Arcing the edge. In *Technical Report 486*, volume 4, pages 1–14, 1997.
- [82] Leo Breiman. Prediction games and arcing algorithms. *Neural Computation*, 11(7):1493–1517, oct 1999. ISSN 0899-7667. doi: 10.1162/089976699300016106.
- [83] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001. ISSN 00905364. doi: 10.1214/aos/1013203451.
- [84] Jerome H. Friedman. Stochastic gradient boosting. *Computational Statistics and Data Analysis*, 38(4):367–378, feb 2002. ISSN 01679473. doi: 10.1016/S0167-9473(01)00065-2.
- [85] 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*, volume 13, pages 785–794. Association for Computing Machinery, aug 2016. ISBN 1450342322. doi: 10.1145/2939672.2939785.
- [86] Quora: Gbm vs xgboost, 2020. <https://www.quora.com/What-is-the-difference-between-the-R-gbm-gradient-boosting-machine-and-xgboost-extreme-gradient-boosting/answer/Tianqi-Chen-1>, last accessed on 2020-12-02.
- [87] Benchmarking random forest implementations, 2021. <http://datascience.la/benchmarking-random-forest-implementations/>, last accessed on 2021-05-03.
- [88] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support, oct 2018.
- [89] 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*, volume 30, pages 3146–3154, 2017.
- [90] K.V. Rashmi and Ran Gilad-Bachrach. Dart: Dropouts meet multiple additive regression trees. *Journal of Machine Learning Research*, 38:489–497, may 2015.
- [91] Mohsen Shahhosseini, Guiping Hu, and Hieu Pham. Optimizing ensemble weights and hyperparameters of machine learning models for regression problems, aug 2019.

- [92] Ian Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Third edition, 2011. ISBN 0123748560.
- [93] Kai Ming Ting and Ian H. Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10:271–289, may 1999. ISSN 10769757. doi: 10.1613/jair.594.
- [94] Many heads are better than one: The case for ensemble learning, 2020. <https://www.kdnuggets.com/2019/09/ensemble-learning.html>, last accessed on 2020-12-27.
- [95] Joseph Sill, Gabor Takacs, Lester Mackey, and David Lin. Feature-weighted linear stacking, nov 2009.
- [96] R. Bell, Y. Koren, and C. Volinsky. The bellkor solution to the netflix prize, 2007.
- [97] Eric Polley and Mark van der Laan. Super learner in prediction. *U.C. Berkeley Division of Biostatistics Working Paper Series*, may 2010.
- [98] Eric C. Polley, Sherri Rose, and Mark J. van der Laan. Super learning. *U.C. Berkeley Division of Biostatistics Working Paper Series*, pages 43–66, 2011. doi: 10.1007/978-1-4419-9782-1\_3.
- [99] Mark J. van der Laan and Sherri Rose. *Targeted Learning*. Springer Series in Statistics. Springer, New York, NY, First edition, 2011. ISBN 1-4419-9781-4. doi: 10.1007/978-1-4419-9782-1.
- [100] Mark J. van der Laan and Sherri Rose. *Targeted Learning in Data Science*. Springer Series in Statistics. Springer International Publishing, Cham, First edition, 2018. ISBN 3-319-65303-7. doi: 10.1007/978-3-319-65304-4.
- [101] Michael P. Perrone and Leon N. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. *Neural networks for speech and image processing*, pages 342–358, sep 1995. doi: 10.1142/9789812795885\_0025.
- [102] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-December, pages 770–778. IEEE Computer Society, dec 2016. ISBN 1467388504. doi: 10.1109/CVPR.2016.90.
- [103] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.
- [104] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 06 2015. doi: 10.1109/CVPR.2015.7298594.
- [105] David W. Opitz, David W. Opitz, and Jude W. Shavlik. Actively searching for an effective neural-network ensemble. *Connection Science*, 8(3):337–353, 1996.
- [106] J. Lin and B. Zhu. Neural network ensemble based on forecasting effective measure and its application. *J Computation Information Systems*, 1(4):781–787, 2005.

- [107] Ying Zhao, Jun Gao, and Xuezhi Yang. A survey of neural network ensembles. In *Proceedings of 2005 International Conference on Neural Networks and Brain Proceedings, ICNNB'05*, volume 1, pages 438–442, 2005. ISBN 0780394224. doi: 10.1109/icnnb.2005.1614650.
- [108] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, jun 2017. ISSN 15577317. doi: 10.1145/3065386.
- [109] Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39, feb 2010. ISSN 02692821. doi: 10.1007/s10462-009-9124-7.
- [110] Saeed Masoudnia and Reza Ebrahimpour. Mixture of experts: A literature survey. *Artificial Intelligence Review*, 42(2):275–293, may 2014. ISSN 02692821. doi: 10.1007/s10462-012-9338-y.
- [111] Michał Woźniak, Manuel Graña, and Emilio Corchado. A survey of multiple classifier systems as hybrid systems. *Information Fusion*, 16(1):3–17, 2014. ISSN 15662535. doi: 10.1016/j.inffus.2013.04.006.
- [112] João Mendes-Moreira, Carlos Soares, Alípio Mário Jorge, and Jorge Freire De Sousa. Ensemble approaches for regression: A survey. *ACM Computing Surveys*, 45(1), nov 2012. ISSN 03600300. doi: 10.1145/2379776.2379786.
- [113] Robert P.W. Duin and David M.J. Tax. Experiments with classifier combining rules. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1857:16–29, 2000. ISSN 16113349. doi: 10.1007/3-540-45014-9\_2.
- [114] Miguel Lopez, Patricia Melin, and Oscar Castillo. A method for creating ensemble neural networks using a sampling data approach. *Advances in Soft Computing*, 41:355–364, 2007. ISSN 16153871. doi: 10.1007/978-3-540-72432-2\_36.
- [115] D. Partridge and W.B. Yates. Engineering multiversion neural-net systems. *Neural Computation*, 8(4):869–893, may 1996. ISSN 08997667. doi: 10.1162/neco.1996.8.4.869.
- [116] D. Partridge. Network generalization differences quantified. *Neural Networks*, 9(2):263–271, 1996. ISSN 08936080.
- [117] R. Maclin and J. Shavlik. Combining the predictions of multiple classifiers: Using competitive learning to initialize neural networks. In *IJCAI*, 1995.
- [118] Shaofeng Zhang, Meng Liu, and Junchi Yan. The diversified ensemble neural network. *NeurIPS 2020*, 2020.
- [119] Rakesh Katuwal and P.N. Suganthan. Dropout and dropconnect based ensemble of random vector functional link neural network. In *Proceedings of the 2018 IEEE Symposium Series on Computational Intelligence, SSCI 2018*, pages 1772–1778. Institute of Electrical and Electronics Engineers Inc., jan 2019. ISBN 1538692769. doi: 10.1109/SSCI.2018.8628640.

- [120] Hyun Yong Lee, Nac Woo Kim, Jun Gi Lee, and Byung Tak Lee. Uncertainty-aware deep learning forecast using dropout-based ensemble method. In *ICTC 2019 - 10th International Conference on ICT Convergence: ICT Convergence Leading the Autonomous Future*, pages 1120–1125. Institute of Electrical and Electronics Engineers Inc., oct 2019. ISBN 1728108926. doi: 10.1109/ICTC46691.2019.8939695.
- [121] Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q. Weinberger. Snapshot ensembles: Train 1, get m for free. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, mar 2017.
- [122] Pavel Izmailov, Dmitrii Podoprikin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*, 2:876–885, mar 2018.
- [123] The cyclical learning rate technique, 2020. <http://teleported.in/posts/cyclic-learning-rate/>, last accessed on 2020-12-27.
- [124] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, aug 2016.
- [125] Leslie N. Smith. Cyclical learning rates for training neural networks. *Proceedings - 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017*, pages 464–472, jun 2015.
- [126] Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, dec 2015.
- [127] B.T. Polyak and A.B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, jul 1992. ISSN 03630129. doi: 10.1137/0330046.
- [128] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pages 2818–2826. IEEE Computer Society, dec 2016. ISBN 1467388504. doi: 10.1109/CVPR.2016.308.
- [129] D. Ruppert. Efficient estimations from a slowly convergent robbins-monro process. Technical report, Cornell University Operations Research and Industrial Engineering, 1988.
- [130] Yue Cao, Thomas Andrew Geddes, Jean Yee Hwa Yang, and Pengyi Yang. Ensemble deep learning in bioinformatics. In *Nature Machine Intelligence*, volume 2, pages 500–508. Nature Research, sep 2020. doi: 10.1038/s42256-020-0217-y.
- [131] Bohyung Han, Jack Sim, and Hartwig Adam. Branchout: Regularization for online ensemble tracking with convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 521–530, 2017. doi: 10.1109/CVPR.2017.63.
- [132] Min Jang and Sungzoon Cho. Observational learning algorithm for an ensemble of neural networks. *Pattern Analysis and Applications*, 5(2):154–167, 2002. ISSN 14337541. doi: 10.1007/s100440200014.

- [133] Bruce E. Rosen. Ensemble learning using decorrelated neural networks. *Connection Science*, 8(3-4):373–384, 1996. doi: 10.1080/095400996116820.
- [134] Y. Liu and X. Yao. Ensemble learning via negative correlation. *Neural Networks*, 12(10): 1399–1404, dec 1999. ISSN 08936080. doi: 10.1016/S0893-6080(99)00073-8.
- [135] Y. Liu. *Negative Correlation Learning and Evolutionary Neural Network Ensembles*. PhD thesis, University College, The University of New South Wales, Australian Defence Force Academy, Canberra, Australia, oct 1998.
- [136] Yong Liu and Xin Yao. Simultaneous training of negatively correlated neural networks in an ensemble. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 29(6):716–725, 1999. ISSN 10834419. doi: 10.1109/3477.809027.
- [137] R. I. McKay and H. Abbass. Analyzing anti-correlation in ensemble learning, dec 2001.
- [138] H. Abbass. Anti-correlation measures in genetic programming. In Nikola Kasabov and Peter Whigham", editors, *In Australasia-Japan Workshop on Intelligent and Evolutionary Systems*, pages 45–51, 2001.
- [139] Gavin Brown. Diversity in neural network ensembles. *Colorado Technical University*, 2004.
- [140] Yong Liu and Xin Yao. A cooperative ensemble learning system. *IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*, 3:2202–2207, 1998. doi: 10.1109/IJCNN.1998.687202.
- [141] Yong Liu, Xin Yao, and Tetsuya Higuchi. Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation*, 4(4):380–387, nov 2000. ISSN 1089778X. doi: 10.1109/4235.887237.
- [142] Gavin Brown and Jeremy Wyatt. Negative correlation learning and the ambiguity family of ensemble methods. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2709:266–275, 2003. ISSN 16113349. doi: 10.1007/3-540-44938-8\_27.
- [143] Xianghua Fu, Boqin Feng, Zhaofeng Ma, and Ming He. Method of incremental construction of heterogeneous neural network ensemble with negative correlation. *Xi'an Jiaotong University*, pages 51–63, 2004.
- [144] Zeke S.H. Chan and Nik Kasabov. Fast neural network ensemble learning via negative-correlation data correction. *IEEE Transactions on Neural Networks*, 16(6):1707–1710, nov 2005. ISSN 10459227. doi: 10.1109/TNN.2005.852859.
- [145] Zheng Qin, Yu Liu, Xingchen Heng, and Xianhui Wang. Negatively correlated neural network ensemble with multi-population particle swarm optimization. In *Lecture Notes in Computer Science*, pages 520–525. Springer Verlag, 2005. doi: 10.1007/11427391\_83.
- [146] Zeke S.H. Chan and Nikola Kasabov. A preliminary study on negative correlation learning via correlation-corrected data (nccd). *Neural Processing Letters*, 21(3):207–214, jun 2005. ISSN 13704621. doi: 10.1007/s11063-005-1084-6.



- [147] Huanhuan Chen and Xin Yao. Evolutionary random neural ensembles based on negative correlation learning. In *IEEE Congress on Evolutionary Computation, CEC 2007*, pages 1468–1474, 2007. ISBN 1424413400. doi: 10.1109/CEC.2007.4424645.
- [148] R. McKay and H. Abbass. Analyzing anti-correlation in ensemble learning. *Proceedings of 2001 Conference on Artificial Neural Networks and Expert SystemsAt: Otago, NZ*, 2007.
- [149] M.A.H. Akhand, Md Monirul Islam, and Kazuyuki Murase. A comparative study of data sampling techniques for constructing neural network ensembles. *International Journal of Neural Systems*, 19(2):67–89, apr 2009. ISSN 01290657. doi: 10.1142/S0129065709001859.
- [150] Huanhuan Chen and Xin Yao. Regularized negative correlation learning for neural network ensembles. *IEEE Transactions on Neural Networks*, 20(12):1962–1979, dec 2009. ISSN 10459227. doi: 10.1109/TNN.2009.2034144.
- [151] Huanhuan Chen and Xin Yao. Multiobjective neural network ensembles based on regularized negative correlation learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(12):1738–1751, 2010. ISSN 10414347. doi: 10.1109/TKDE.2010.26.
- [152] Heesung Lee, Euntai Kim, and Witold Pedrycz. A new selective neural network ensemble with negative correlation. *Applied Intelligence*, 37(4):488–498, dec 2012. ISSN 15737497. doi: 10.1007/s10489-012-0342-3.
- [153] Saeed Masoudnia, Reza Ebrahimpour, and Seyed Ali Asghar Abbaszadeh Arani. Combining features of negative correlation learning with mixture of experts in proposed ensemble methods. *Applied Soft Computing Journal*, 12(11):3539–3551, 2012. ISSN 15684946. doi: 10.1016/j.asoc.2012.07.022.
- [154] Monther Alhamdoosh and Dianhui Wang. Fast decorrelated neural network ensembles with random weights. *Information Sciences*, 264:104–117, 2014. ISSN 00200255. doi: 10.1016/j.ins.2013.12.016.
- [155] Weiguo Sheng, Pengxiao Shan, Shengyong Chen, Yurong Liu, and Fuad E. Alsaadi. A niching evolutionary algorithm with adaptive negative correlation learning for neural network ensemble. *Neurocomputing*, 247:173–182, jul 2017. ISSN 18728286. doi: 10.1016/j.neucom.2017.03.055.
- [156] Shuqin Gu, Yuexian Hou, Lipeng Zhang, and Yazhou Zhang. Regularizing deep neural networks with an ensemble-based decorrelation method. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 2018-July, pages 2177–2183. International Joint Conferences on Artificial Intelligence, 2018. ISBN 0999241127. doi: 10.24963/ijcai.2018/301.
- [157] Zenglin Shi, Le Zhang, Yun Liu, Xiaofeng Cao, Yangdong Ye, Ming-Ming Cheng, and Guoyan Zheng. Crowd counting with deep negative correlation learning. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5382–5390, 2018. doi: 10.1109/CVPR.2018.00564.
- [158] Sebastian Buschjäger, Lukas Pfahler, and Katharina Morik. Generalized negative correlation learning for deep ensembling, nov 2020.

- [159] Geoffrey I. Webb. Multiboosting: a technique for combining boosting and wagging. *Machine Learning*, 40(2):159–196, aug 2000. ISSN 08856125. doi: 10.1023/A:1007659514849.
- [160] Geoffrey I. Webb and Zijian Zheng. Multistrategy ensemble learning: Reducing error by combining ensemble learning techniques. *IEEE Transactions on Knowledge and Data Engineering*, 16(8):980–991, aug 2004. ISSN 10414347. doi: 10.1109/TKDE.2004.29.
- [161] Yang Yu, Zhi Hua Zhou, and Kai Ming Ting. Cocktail ensemble for regression. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 721–726, 2007. ISBN 0769530184. doi: 10.1109/ICDM.2007.60.
- [162] C Hang and J Gao. Fast license plate character recognition based on adaboost. *Computer Modeling in Engineering Sciences*, 9:140–143, 2010.
- [163] H Xu, S Wang, and R Wang. Research of p2p traffic identification based on neural network ensemble. *Journal of Nonjing University of Posts and Telecommunications*, 3:79–83, 2010.
- [164] H Liu, G Chen, and G Song. Adaboost based ensemble of neural networks in analog circuit fault diagnosis. *Chinese Journal of Scientific Instrument*, 4:851–856, 2010.
- [165] D Baumgartner and G Serpen. Performance of global-local hybrid ensemble versus boosting and bagging ensembles. *International Journal of Machine Learning and Cybernetics*, 4(4):301–317, 2013.
- [166] D Calderon, T Baidyk, and E Kussul. Hebbian ensemble neural network for robot movement control. *Optical Memory and Neural Networks*, 22(3):166–183, 2013.
- [167] K Dai, J Zhao, and F Cao. A novel decorrelated neural network ensemble algorithm for face recognition. *Knowledge Based Systems*, 89:541–552, 2015.
- [168] J Dong, C Zheng, G Kan, M Zhao, J Wen, and J Yu. Applying the ensemble artificial neural network-based hybrid data-driven model to daily total load forecasting. *Neural Computing and Applications*, 26:603–611, 2015.
- [169] E Hadavandi, J Shahrabi, and S Shamshirband. A novel boosted-neural network ensemble for modeling multi-target regression problems. *Engineering Applications of Artificial Intelligence*, 45:204–219, 2015.
- [170] Y Zhang and S Zhong. A privacy-preserving algorithm for distributed training of neural network ensembles. *Neural Computing and Applications*, 22:S269–S282, 2013.
- [171] N Kourentzes, DK Barrow, and SF Crone. Neural network ensemble operators for time series forecasting. *Expert Systems with Applications*, 41(9):4235–4244, 2014.
- [172] C Smith and Y Jin. Evolutionary multi-objective generation of recurrent neural network ensembles for time series prediction. *Neurocomputing*, 143:302–311, 2014.
- [173] J Tian, M Li, F Chen, and J Kou. Coevolutionary learning of neural network ensemble for complex classification tasks. *Pattern Recognition*, 45(4):1373–1385, 2012.
- [174] SG Soares, CH Antunes, and R Araújo. Comparison of a genetic algorithm and simulated annealing for automatic neural network ensemble development. *Neurocomputing*, 121:498–511, 2013.

- [175] Z-S Zhao, X Feng, Y-Y Lin, F Wei, S-K Wang, T-L Xiao, M-Y Cao, and Z-G Hou. Evolved neural network ensemble by multiple heterogeneous swarm intelligence. *Neurocomputing*, 49:29–38, 2015.
- [176] S Faußer and F Schwenker. Neural network ensembles in reinforcement learning. *Neural Processing Letters*, 41:55–69, 2015.
- [177] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, jun 2014.
- [178] S. Mason, Jonathan Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent in function space. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, may 1999.
- [179] scikit learn. Histogram-based gradient boosting regression tree., 2021. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingRegressor.html>, last accessed on 2021-05-23.
- [180] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937. ISSN 01621459. doi: 10.1080/01621459.1937.10503522.
- [181] P.B Nemenyi. *Distribution-free Multiple Comparisons*. PhD thesis, Princeton University, USA, 1963.
- [182] Balázs Csanád Csáji. Approximation with artificial neural networks. Master’s thesis, Faculty of Sciences, Eötvös Loránd University, Hungary, 2001.
- [183] Frank Rosenblatt. The perceptron—a perceiving and recognizing automaton. Technical report, Cornell Aeronautical Laboratory, 1957.
- [184] Avrim L. Blum and Ronald L. Rivest. Training a 3-node neural network is np-complete. *Neural Networks*, 5(1):117–127, jan 1992. ISSN 08936080. doi: 10.1016/S0893-6080(05)80010-3.
- [185] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, jul 2015. PMLR.
- [186] Russell Reed. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. Bradford Books, First edition, 1999. ISBN 9814271063.
- [187] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, nov 2016.
- [188] Regularizers difference, 2021. <https://stats.stackexchange.com/questions/383310/what-is-the-difference-between-kernel-bias-and-activity-regulizers-and-when-t/383326#383326>, last accessed on 2021-02-04.

- [189] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, jul 2012.
- [190] N Sharkey, J Neary, and A Sharkey. Searching weight space for back propagation solution types. *Current trends in connectionism: Proceedings of the 1995 Swedish conference on connectionism*, pages 103–120, 1995.
- [191] Bambang Parmanto, Paul W Munro, and Howard R Doyle. Improving committee diagnosis with resampling techniques. In *Advances in Neural Information Processing Systems*, volume 8, pages 882–888, 1995.
- [192] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *Advances in Neural Information Processing Systems Deep Learning and Representation Learning Workshop*, 2015.
- [193] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, 5(1):15–17, may 1976. ISSN 00200190. doi: 10.1016/0020-0190(76)90095-8.
- [194] Leo Breiman, Jerome Friedman, Charles J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Chapman and Hall/CRC, First edition, 1984. ISBN 0412048418.





# Appendices





# Appendix A

## Neural Network

**(Artificial) Neural Networks** ((A)NN) belong to the soft computing techniques field and try to represent computing intelligence by seeking to emulate the human brain's mechanisms [8].

They are also referred to as universal approximators since they follow the universal approximation theorem [182]. Consequentially, they are highly flexible non-linear methods capable of learning complex relationships between variables and mapping a near-infinite number of functions (given enough resources and time). Independently of the functions' complexity, a finite NN can approximate any boundary, theoretically, with any required accuracy. So, they can be considered a ME.

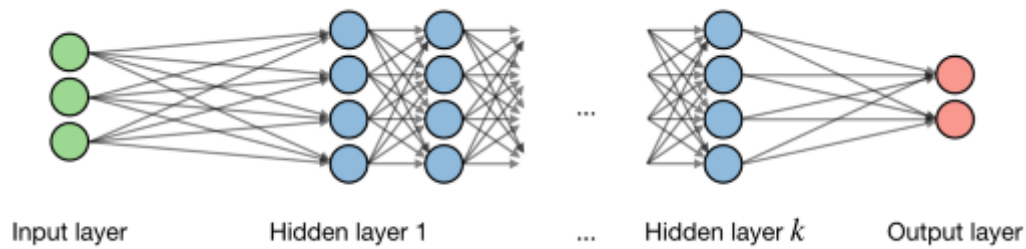
Also, NNs are not explicitly programmed to perform a specific task (e.g., rule-based) [1]. Instead of having a fixed architecture, the NN learns how to perform a task independently by adjusting its parameters [8].

Owing to numerous recent technological advantages such as distributed information storage, massive parallel processing, self-learning, self-adapted abilities, and several others, NNs have been widely used throughout.

### A.1 Machine Representation

In computer science, the **Neuron**, also known as *Node*, gets its input values from either the input layer or any previous hidden layers' neurons. It then processes them and, finally, generates an output signal to the output layer or any following hidden layers' neurons (Fig. A.1) [8].

The external inputs are analogous to the human body's sensors (eyes, nose, ears, or tongue). In a NN, they are independent variables  $X_1, X_2, \dots, X_i$ . Each set of them corresponds to a single dataset row.

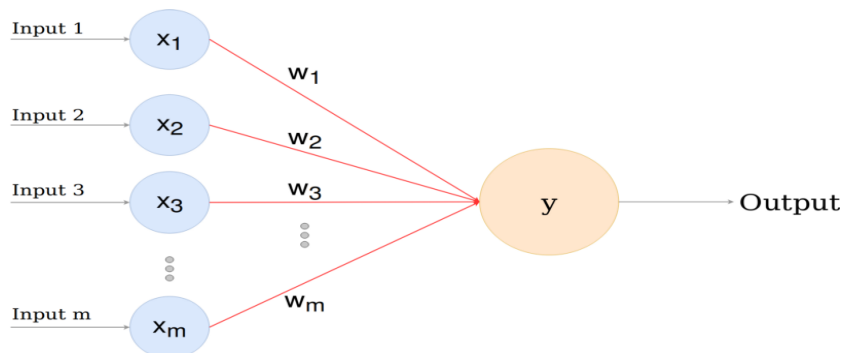
Figure A.1: Neural Network <sup>37</sup>

**Synapses** represent the interconnections between neurons in different layers [8]. Each connection is assigned a set of weights representing the NN's learning. These are adjusted during the Networks' training and are crucial in determining which values get passed along.

**Output Value** is the outcome of a neuron [8]. More specifically, the predicted value according to a particular set of input of features. This value can be **qualitative** (e.g., price of a house) or **categorical**.

### A.1.1 Perceptron

First proposed in 1957 [183], it is the most straightforward type of NN able to learn how to perform a task. It consists of a single NN input layer with  $X_1, X_2, \dots, X_i$  as entry points, a single hidden layer with one neuron, and, lastly, an output layer that predicts a final result (Fig. A.2).

Figure A.2: Perceptron <sup>38</sup>

### A.1.2 Hidden Layer

It is utilized to add further complexity and building flexibility, thus, predictive power to the NNs [8].

In practice, it allows the model to transform the input features into a different set of features and present an apprehended representation of low input data resolution [1]. In other words, hidden

<sup>37</sup><https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning>, last accessed on 2021-01-21

<sup>38</sup>[https://miro.medium.com/max/810/1\\*7pwA1DjBw6JDkwZQecUNiw.png](https://miro.medium.com/max/810/1*7pwA1DjBw6JDkwZQecUNiw.png), last accessed on 2021-01-21

layers' neurons are used to pick up hidden feature associations at multiple abstraction levels, which may prove helpful in making better final predictions.

Additionally, they can output their internal representations directly [1].

## A.2 Neuron mechanics

The neuron takes the weighted sum of all inputs [4]. By noting  $i$  the  $i$ th Network's layer and  $j$  the  $j$ th hidden layer unit, each Node's mathematical operation is represented by Eqn. A.1.

$$z^{[ij]} = \phi\left(\sum_{k=1}^K w_k^{[ij]} x_k^{[ij]} + b^{[ij]}\right) \quad (\text{A.1})$$

where  $z$  is the neuron's output,  $K$  is the number of input synapses,  $w$  is the synapse's weight,  $x$  is the synapse's input value,  $b$  is the neuron's bias coefficient, and  $\phi$  is the activation function.

### A.2.1 Activation function

It is a mathematical operation used by each NN's neuron to produce a final decision based on the  $z$  value [8]. There are several different kinds, but each maps the weighted sum to an output value. The final decision is then passed to the next layer's Nodes (Table A.1).

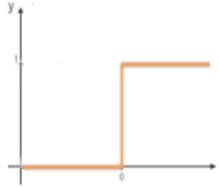
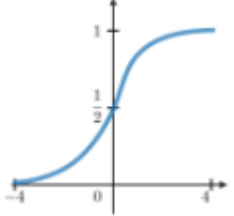
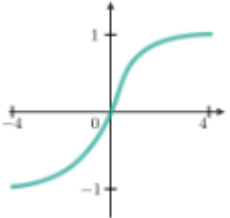
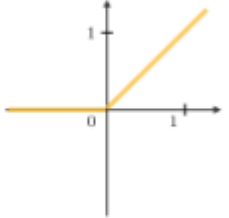
Threshold	Sigmoid	Tanh	ReLU
 <p>Figure A.3: Threshold activation function (adapted from <sup>39</sup>).</p>	 <p>Figure A.4: Sigmoid activation function <sup>40</sup></p>	 <p>Figure A.5: Tanh activation function <sup>40</sup></p>	 <p>Figure A.6: ReLU activation function <sup>40</sup></p>
$f(z) = 0$ if $x < 0$ , and $f(z) = 1$ if $x \geq 0$	$f(z) = 0.5$ if $z = 0$	$f(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}}$	$f(z) = \max(0, z)$

Table A.1: Comparison of activation functions.

<sup>39</sup><https://www.andreaperlato.com/aipost/the-activation-function/>, last accessed on 2021-01-21

<sup>40</sup><https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-deep-learning>, last accessed on 2021-01-21

**Tanh** is also known as *Hyperbolic Tangent*, and **ReLU** is also known as *Rectified Linear Unit*. A standard workflow uses the **rectifier** function for the hidden layers [8].

### A.2.2 Cost function

Commonly used to assess model performance, it represents a numerical judgment of how incorrect the NN predictions are or to what extent the model outputs correctly predict the actual outputs [4]. If the cost function is reduced, it means the NN's mistakes are also diminishing and, in turn, making more accurate predictions.

In practice, the NN's predicted output  $y$  is compared to the target output  $\bar{y}$ . Their differences are quantified, and the error is computed.

## A.3 Propagation

It is divided between Forward and Backpropagation.

### A.3.1 Forward propagation

Data is entered into the NN's input layer, passed through the hidden layers (activate neurons based on their weights and disseminate their activations), and, finally, the output layer produces a prediction  $f(x)$  or  $y$  (Fig. A.7, section A.2).

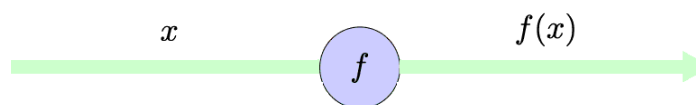


Figure A.7: Forward Propagation Model (adapted from <sup>41</sup>).

### A.3.2 Backpropagation

Also known as *Backward Propagation*, the **cost function's** loss is minimized by passing it back through the NN, starting from the output and ending in the input layer (Fig. A.8) [4].

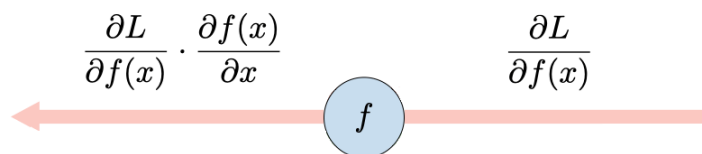


Figure A.8: Backpropagation Model (adapted from <sup>42</sup>).

<sup>41</sup><https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks>, last accessed on 2021-01-21

<sup>42</sup><https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks>, last accessed on 2021-01-21

### A.3.2.1 Gradient Descent

Also known as *Batch Gradient Descent* or *GD*, it is a global search algorithm (brute force approach) that discovers the best set of weights [8]. However, it does not harness gradient calculation's computational advantages, thus requiring many evaluation functions compared to gradient-based approaches [8].

GD computes the loss function's gradient concerning the weights, checks if the slope is either positive or negative (down or uphill), and, using the chain rule, optimally and concurrently updates the weights in the opposite direction. In other words, the weights are renewed according to their error responsibility and in the course that offers the biggest loss error (or cost function) decrease considering the targets and predictions (Fig. A.9). However, it involves trying out multiple input feature combinations, recording their costs, and identifying the minimal cost combination.

The **derivative concerning the weight**  $w$  is calculated by employing the chain rule (Eqn. A.2).

$$\frac{\partial L(z,y)}{\partial w} = \frac{\partial L(z,y)}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w} \quad (\text{A.2})$$

where  $L$  is the loss,  $\alpha$  is the LR,  $z$  is the neuron's weighted inputs,  $y$  is the neuron's output, and  $w$  is the synapse's weight.

Inherently, the GD's update rule (concerning the cost function) follows the Eqn. A.3.

$$w \longleftarrow w - \alpha \times \frac{\partial L(z,y)}{\partial w} \quad (\text{A.3})$$

The LR determines by how much the weights are updated.

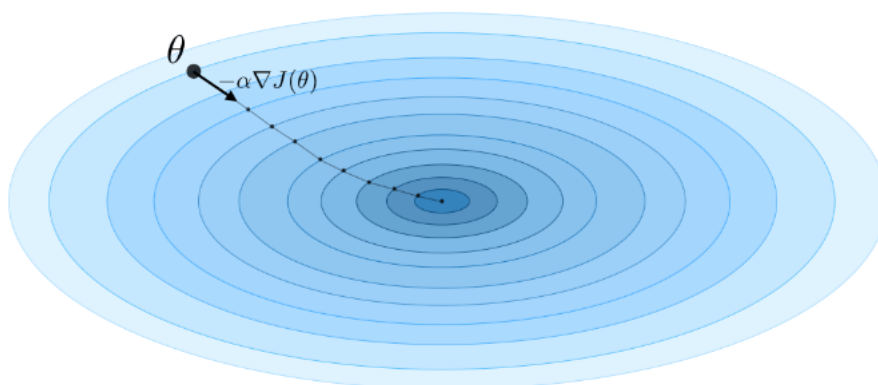


Figure A.9: Gradient Descent <sup>43</sup>

<sup>43</sup><https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-supervised-learning>, last accessed on 2021-01-21

This approach may be viable when working with convex cost functions, relatively smooth error manifolds, perceptrons with a handful of features, and datasets unaffected by erroneous data or outliers. In this case, the algorithm moves somewhat directly towards an optimum solution, either local or global [8].

However, updating weights during training is customarily not according to the entire training data at once because of computation complexities and noise issues. Also, NNs have an enormous number of local minima but only a subset of which will be global [101]. Since GD is deterministic, the cost function's optimizer will tend to discover local minimum rather than global. This means the algorithm's computed weights will not accurately represent the optimal values for maximum prediction accuracy as the cost could be lower. Also, GD has to go through all the training instances every time an update is performed. As the number of instances increases, it becomes too inefficient and time-consuming. Being an NP-complete problem [184], GD is not for practical NNs.

Fortunately, there are more efficient ways to compute optimal weight values for a NN to minimize its cost, namely, **SGD**.

### A.3.2.2 Stochastic Gradient Descent

Also known as *SGD*, it feeds individual dataset rows into the Network, and their weights are updated after each iteration on an observation basis [8]. Hence, SGD assists in evading the obstacle of converging to local rather than global minima.

In practical terms, it does not require the function to be convex. This is because the SGD algorithm has much higher fluctuations stirred up. After all, it analyses one row at a time.

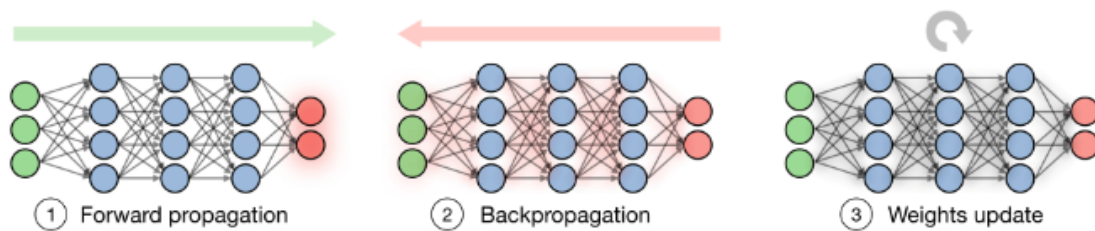
It is also faster since it does not have to memory load all data before computing weights.

### A.3.2.3 Mini-batch gradient descent

Acts as a middle point between GD and SGD [8]. The updating step is performed on mini-batches. In practice, the weights are computed using batches of rows instead of all dataset rows. The NN's weights are updated as an aggregate after processing all batch observations.

## A.4 NN Workflow

After initializing the NN's weights, Fig. A.10's steps are followed sequentially, from left to right.

Figure A.10: NN workflow (adapted from <sup>44</sup>).

Once the entire training data set has passed through the Network or the loss error stops reducing (early stopping), an epoch (from 2. to 6.) is complete.

## A.5 Tuning

### A.5.1 Epochs

It is the number of cycles in the NN's training process [56]. The epoch is repeated as many times as necessary to achieve a sufficiently low-cost function output. After the last epoch, the training is over.

### A.5.2 Input variables range

Input variables need to be **standardized** or **normalized** before they are fed to the NN [1]. Table A.2 compares these two methods.

Standardization	Normalization
$\frac{x - \mu_B}{\sigma_B}$ <p>where <math>x</math> is the input observation, <math>\mu_B</math> is the batch's mean, and <math>\sigma_B</math> is the batch's standard deviation</p>	$\frac{x - \min(x)}{\max(x) - \min(x)}$ <p>where <math>x</math> is the input observation, <math>\min()</math> is the minimum function, and <math>\max()</math> is the maximum function</p>

Table A.2: Comparison of standardization and normalization.

Both techniques ensure that all input variables occupy roughly the same range of values [1]. In turn, an outlier for one feature does not skew the results of the NN.

**Batch Normalization** is a particular case of Normalization. The authors refer to it as “internal covariate shift”, which changes the inputs' distribution during training by scaling the previous layer's outputs (Eqn. A.4) [185]. Customarily done before a non-linear layer or after a fully connected layer, it strives to correct a batch by providing more significant LR, diminishing an influential dependence on the initialization, and accelerating the Network's training process.

<sup>44</sup><https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks>, last accessed on 2021-01-21

$$x'_i = \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} + \beta \quad (\text{A.4})$$

where  $x'_i$  is the normalized input observation,  $x_i$  is the input observation,  $\gamma$  and  $\beta$  are parameters to be learned,  $\mu_B$  is the batch's mean,  $\sigma_B^2$  is the batch's variance, and  $\varepsilon$  is a numerical stability constant.

### A.5.3 Learning Rate

Also known as  $\alpha$  or *LR*, it means at which velocity are the weights updated [8].

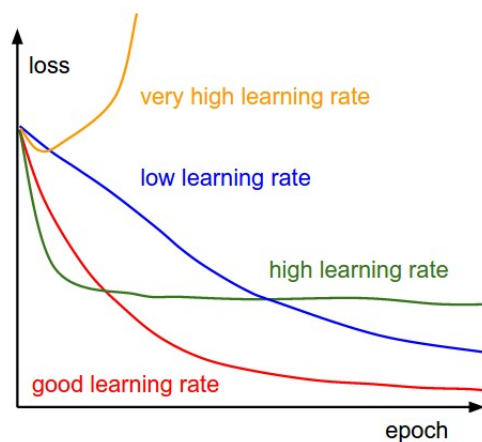


Figure A.11: Comparison of multiple learning rate values (adapted from <sup>45</sup>).

Choosing the initial LR can be challenging and require careful experimentation (Fig. A.11). An **LR schedule**, an update mechanism to decay it over time, is also difficult to set in advance as it does not adapt to data dynamics [56]. In fact, the same LR is applied to all parameters, which might be learning at different rates, making it very hard to get out of a saddle point [56].

It may either be static or adaptively adjusted. **Adaptive LR** allows the LR to fluctuate while training a model, thus, “fine-tuning” the model during training [8]. This may diminish its elapsed time and enhance the optimal solution. Despite Adam optimizer being commonly usually used, others may also be serviceable (Tables A.3 and A.4).

Method	Explanation	Update of $w$	Update of $b$
Momentum	- 2 parameters to tune - Improvement on SGD - Dampens oscillations	$w - \alpha v_{dw}$	$b - \alpha v_{db}$

Table A.3: Comparison of adaptive learning rate methods - 1.

<sup>45</sup><https://medium.com/analytics-vidhya/cyclical-learning-rates-a922a60e8c04>, last accessed on 2021-01-21



RMSprop	- Root Mean Square propagation - Speeds up the learning algorithm by controlling oscillations	$w - \alpha \frac{dw}{\sqrt{s_{dw}}}$	$b - \alpha \frac{db}{\sqrt{s_{db}}}$
Adam	- Adaptive Moment estimation - 4 parameters to tune - Most popular method	$w - \alpha \frac{v_{dw}}{\sqrt{s_{dw} + \epsilon}}$	$b - \alpha \frac{v_{db}}{\sqrt{s_{db} + \epsilon}}$

Table A.4: Comparison of adaptive learning rate methods - 2.

Remark: other methods include *SGD*, *Adagrad*, and *Adadelta* [123].

### A.5.4 Regularization

It is a technique that usually causes slight alterations in the learning algorithm that leads to different local minima promoting diversity and reducing overfitting [4]. Instead of discovering the best model parameters for a particular circumstance, the “best” model is a larger model appropriately regularized. In turn, the resulting model is simpler, more robust against training data noise, has more stable predictions, therefore, exhibiting better generalization capabilities [56].

It is an “ill-posed [problem] if small changes in [its] information cause large changes in the solution. This instability ... makes solutions unreliable because small measurement errors or uncertainties in parameters may be greatly magnified and lead to wildly different responses. ... The idea behind regularization is to use supplementary information to restate an ill-posed problem in a stable form” [186].

From the field of ML, the concept of regularization consists of penalizing coefficients [4]. In DL, it actually penalizes the Nodes’ weight matrices (among others) [8].

The NN has a broad spectrum of regularization parameters. A proper guideline is to design an under-constrained NN structure and apply regularization to lessen the overfitting likelihood.

#### A.5.4.1 Data Augmentation

Increases the data amount by adding new synthetic data based on existing data or lightly mutated replicas of existing data [8]. There is a close relationship with oversampling/undersampling in data analysis [4].

#### A.5.4.2 Limit the Number of Units

The amount of neuron units in a NN’s layer  $M$  is an adjustable parameter to get the best predictive performance. A possible avenue is to limit  $M$  hence giving rise to a simpler model.

However, due to local minima, the generalization error is not a mere function of  $M$ . Accordingly, there is the need to control it to avoid overfitting.

### A.5.4.3 Dropout

It is one of the most exciting types of regularization techniques. Since its release in 2014 [177], it has risen to become one of the most commonly adopted DL (section A.7) methods to introduce randomness.

As a NN learns, neighboring neurons start to rely on each other's contexts somewhat, leading to overfitting obstacles. This is referred to as co-adaptation [8]. Dropout prevents it by making a neuron's particular unit presence unreliable by randomly dropping it with probability  $p > 0$ . If a Node is dropped, there are no incoming/outgoing weights to/from this Node (Fig. A.12) [177]. This implies that their contribution to neurons' downstream activation is temporally dismissed, and any neuron's weight updates are not applied, resulting in diverse internal independent learned representations. In turn, there is a reduction in specific weight dependency and better generalization capacity [8].

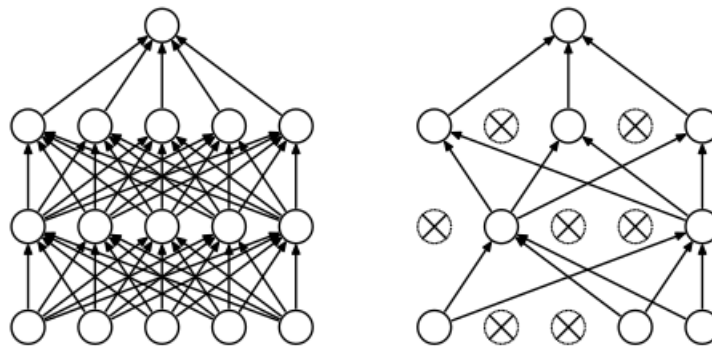


Figure A.12: Comparison of regular NN and Dropout NN [177].

When training the Network, Dropout forces perturbations to circumvent excessively relying on particular feature sets [8] by implementing two tricks:

- it shares parameters across all Networks;
- for each training instance, it samples a distinct NN.

So each iteration has a diverse set of Nodes resulting in a diverse set of outputs [56].

It is relevant to note that some Nodes may be switched off more than others, but since this is executed repeatedly, on average, each Node will get equal treatment [56]. Also, note that Dropout is only performed during training time. During test time, every Node is always present (Fig. A.13).

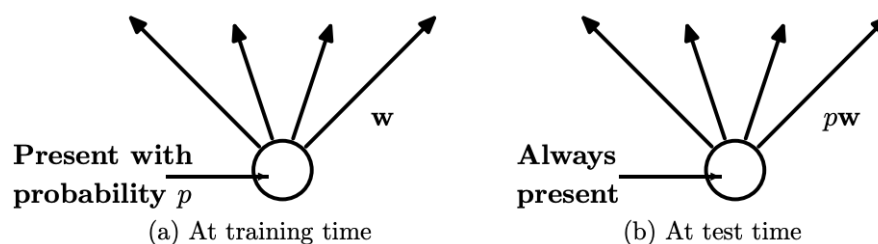


Figure A.13: Dropout: training vs testing (adapted from [177]).

In addition to reducing overfitting, it also introduces sparsity in the output vectors.

### A.5.4.4 Weight Regularization

It is a generic approach that tries to limit the model’s capacity and encourages it to be less complicated (Eqn. A.5). Assuming that smaller weight matrices’ values lead to simpler models, the loss function is updated with a regularization term  $\lambda$ . This value penalizes the model according to the weights’ magnitude, making sure they are not extremely large [56].

$$w_{t+1} = w_t - \alpha \nabla_w J - \lambda w_t \tag{A.5}$$

where  $w_t$  is the weight at step  $t + 1$ ,  $w_t$  is the weight at step  $t$ ,  
 $\alpha$  is the LR,  $\nabla_w J$  is the gradient with respect to  $w$ ,  
 and  $\lambda$  is the regularization term.

Weight regularization may be applied to a previously trained Network without any regularization. Table A.5 sums up these methods.

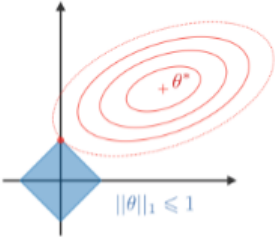
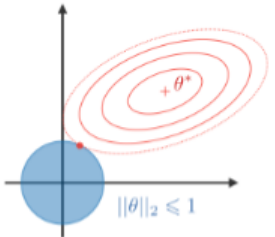
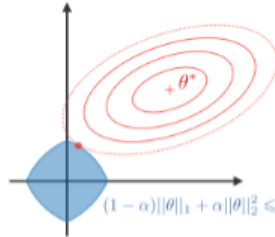
Lasso / L1	Ridge / L2 / Tikhonov	Elastic Net
<ul style="list-style-type: none"> <li>- Expects the model to use some inputs more than others</li> <li>- Good for variable selection and model compression</li> <li>- Penalize weights absolute value</li> <li>- Shrinks coefficients to 0</li> </ul>	<ul style="list-style-type: none"> <li>- Model encouraged to use all inputs without leaning too heavily on any specific one</li> <li>- Weight decaying</li> <li>- Makes coefficients smaller</li> </ul>	<ul style="list-style-type: none"> <li>- Lasso and Ridge combination</li> <li>- Trade-off between variable selection and small coefficients</li> <li>- Beneficial if the amount of predictors is greater than the amount of observations</li> </ul>
 <p>Figure A.14: Lasso <sup>46</sup></p>	 <p>Figure A.15: Ridge <sup>46</sup></p>	 <p>Figure A.16: Elastic Net <sup>46</sup></p>
$\dots + \lambda \ \theta\ _1$ $\lambda \in \mathbb{R}$	$\dots + \lambda \ \theta\ _2^2$ $\lambda \in \mathbb{R}$	$\dots + \lambda [(1 - \alpha)\ \theta\ _1 + \alpha\ \theta\ _2^2]$ $\lambda \in \mathbb{R}, \alpha \in [0, 1]$

Table A.5: Comparison of weight regularization methods.

L1 and L2 are considered a type of “explicit regularization” [187].

#### A.5.4.5 Bias Regularization

From an intuition standpoint, it is similar to weight regularization. However, it tries to reduce the model’s bias [188]. It is commonly used if there is a requirement for the output function to pass through or close to the origin [188].

#### A.5.4.6 Activity Regularization

Also known as *Representation Regularization* or *Sparse Feature Learning*, it is a generic approach similar to weight regularization because both add penalties to the NN. However, activity regularization “places a penalty on the activations of the units in a Neural Network [output layer], encouraging [them] to be small and sparse” [8]. This activation penalty is a real value employed per layer or only at the output layer.

Given that the algorithm penalizes the output layer’s results, it tries to reduce the model’s weights and bias [188]. Table A.6 examines multiple approaches.

L1 vector norm	L2 vector norm
<ul style="list-style-type: none"> <li>- Sum of the absolute activation values</li> <li>- Encourages sparsity (some observation values may become zero)</li> </ul>	<ul style="list-style-type: none"> <li>- Sum of the squared activation values</li> <li>- Encourages small activations values</li> </ul>

Table A.6: Comparison of activity regularization methods.

Commonly used when there is a requirement for the output to be smaller (or closer to 0) [188].

#### A.5.4.7 Weight Constraint Regularization

Weight regularization and activity regularization add penalties that **encourage** the NN to have smaller weights. Oppositely, weight constraint is a generic method that **forces** NNs to have smaller weights. In practice, it checks the weights’ magnitudes and scales them to a value below a pre-defined threshold.

Also, weight constraint allows for more aggressive Network configurations, such as very high LR, which allow the Network to produce more comprehensive weight updates [189].

Some constraint examples include:

- force the vector norm to be 1 (e.g., the unit norm);
- limit the vector norm’s maximum size (e.g., the maximum norm);
- limit the vector norm’s minimum and maximum size (e.g., the min\_max norm).

<sup>46</sup><https://stanford.edu/~shervine/teaching/cs-229/cheatsheet-machine-learning-tips-and-tricks>, last accessed on 2021-01-21

Note that Maximum Norm, also known as *MaxNorm* or *Max-Norm*, is less aggressive than other norms such as the unit norm. “It typically improves the performance of stochastic gradient descent training of Deep Neural Nets” [177].

A common example is to use weight constraint regularization with Dropout regularization [177].

#### A.5.4.8 Early Stopping

A major difficulty regarding NN training is how long to train them. Too little means the model will underfit the training and the test sets [4]. On the other hand, too much training means the model will overfit the training dataset and have reduced test set performance [4]. The solution is to halt the training method once a plateau is reached from a validation loss standpoint (sufficiently small reduction) or starts to increase over several epochs on a validation set to improve the generalization capability [8]. In this case, before the final epoch reaches, the training ends since there is no point in further training the model (Fig. A.17).

It may be considered a type of “implicit regularization” [187], much like using a smaller Network with less capacity.

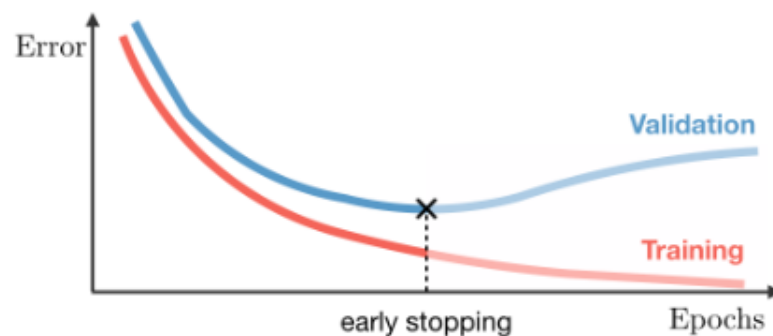


Figure A.17: Early Stopping <sup>47</sup>

## A.6 Disadvantages

Despite the NN’s benefits, there are also drawbacks such as:

- susceptible to their original conditions, such as random initial weights and training data’s statistical noise;
- guaranteeing diverse (even random) weights does not automatically guarantee higher diversity or noticeable differences due to the Network’s optimization problem difficulty, the stochastic learning algorithm characteristic, and the sheer amount of different input-output mapping possibilities [190] [191] [115] [7];

<sup>47</sup><https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks>, last accessed on 2021-01-21

- no direct way to find the best architecture;
- learn via a stochastic learning algorithm, a tough optimization method that may miss convergence, in which case, it does not find the best-performing set of weights;
- in the case of getting stuck in a local minima, there is no optimal convergence at all;
- high computational cost in the training phase, thus, time-consuming and slow convergence speed;
- favor requiring vast training data amounts to make reasonable predictions;
- low flexibility for complex problem-solving means that its computational cost grows ever larger as the problem's size increases, hindering proper scaling and further development;
- “black box” distribution of knowledge is essentially manifested in an abstract digital manner, causing difficulty while understanding the predictions;
- **perhaps more important**, generally have low bias and high variance, hence, unstable.

## A.7 Deep Learning

Represents computational models comprised of multiple hidden layers (section A.1.2) [56]. This is a more compact version of a Network with the same number of units but being shallow. In fact, every single Node may be visualized as an individual model. So, Deep Neural Networks (DNN), in addition to “stacking” predictions, also “stack” data feature representations [8].

In practice, stacking layers allows the model to wreak the benefits of multiple hidden layers combined. Even though this is not theoretically proved, as there is neither a good theory for why extensive Networks work nor what is going on inside, it is “somewhat” empirically demonstrated [56].

However, “first, they are hard to interpret, and second, they suffer from overfitting” since they may overcomplicate a relatively simple problem [8]. Also, as mentioned in section A.6, the NN's training phase is, typically, a costly procedure [7]. This is exacerbated in DL because a vast amount of weights correspond to different layers needing tuning [192].

## Appendix B

# Decision Tree

It is a fast algorithm, present in ensemble methods as base learners, which employs a greedy local optimization approach [193] and uses a sequence of interrogations to make predictions. Several factors must be considered, such as what features to make decisions upon and the decision-making threshold.

High entropy implies low homogeneity and low entropy otherwise. The main objective is to minimize entropy. So, the splitting stops when  $parent\_entropy - child\_entropy < threshold$ , which indicates homogeneity [1].

Trees may be categorized based on their depth:

- **Pruned Tree**, also known as *Shallow Tree* or *General Tree*, is a Decision Tree with few depths [1]. So, it manifests a high bias and low variance;
  - **Decision Stump** is a single split, one-level Pruned Tree variation with an aggressive sub-sampling policy [1]. It is commonly associated with being a weak learner.
- **Unpruned Decision Tree**, also known as *Deep Tree* or *Specific Tree*, is a Decision Tree with lots of depths, if not fully grown [1]. So, it manifests a low bias and high variance. This is accomplished by slightly overfitting the training data, making each Tree further altered with less correlated predictions between one another.

**Classification And Regression Trees (CART)** are Decision Trees for classification and regression predictive modeling problems [60] [4] [72].

It is composed of Binary Trees [194]. A sole input variable  $x$  depicts the root node, and its split point (considering the variable is numeric) arises leaf nodes. These comprise an output variable  $y$  applied to produce an input variable selection prediction.

The splitting procedure consists of a recursive binary numerical method that divides the input space using input variable selection until a suitable Tree is constructed. The specific split points are chosen with a greedy algorithm that minimizes the cost function.

The squared error sum over every training data point within the box is the cost function (regression predictive modeling problems, Eqn. B.1).

$$\text{sum}(y - \bar{y})^2 \tag{B.1}$$

where  $y$  is the model's predicted output,  $\bar{y}$  is the model's target output, and  $\text{sum}()$  is the sum function.

The stopping criterion lets the Decision Tree grasp the moment to quit splitting as it strives its way down. It, obviously, strongly influences the performance of the Tree. More specifically, a minimum count of each leaf node assigned training instances or a dataset tunned training members count. Lastly, a pruning method is employed to establish the final Decision Tree complexity, namely, the number of Tree splits.

Simpler Trees are often favored. It is so since they are less prone to overfitting the data and more straightforward to understand.



## Appendix C

# Kaggle Competitions

### C.1 Kaggle dominating algorithms

Gradient Boosted Machines and NNs have dominated recent Kaggle competitions, as seen in Table C.1.

Competition	Type	Winning ML Library / Algorithm
Liberty Mutual	Regression	<b><u>XGBoost</u></b>
Caterpillar Tubes	Regression	<b><u>Keras + XGBoost</u></b> + Reg. Forest
Diabetic Retinopathy	Image	SparseConvNet + RF
Avito	CTR	<b><u>XGBoost</u></b>
Taxi Trajectory 2	Geostats	Classic NN
Grasp and Lift	EFG	<b><u>Keras + XGBoost</u></b> + other CNN
Otto Group	Classification	Stacked ensemble of 35 models
Facebook IV	Classification	sklearn GBM

Table C.1: Overview of dominant algorithms in Kaggle competitions

### C.2 Kaggle Top Competitor comment

Giuliano Janson, a top Kaggle competitor, at Quora (2016),<sup>48</sup> commented that “there is really no way you can win most Kaggle competitions without a very strong ensemble. As good as your best individual model is, it won’t match a good ensemble”.

---

<sup>48</sup>available at: <https://www.quora.com/What-machine-learning-approaches-have-won-\most-Kaggle-competitions>

### C.3 Netflix \$1M ML competition

Netflix hosted a \$1M ML competition<sup>49</sup>. The prize included teams attempting to improve on Netflix's native algorithm recommendation predictions. The team that obtained a 10% performance (Fig. C.1) increase was awarded a \$1M prize.

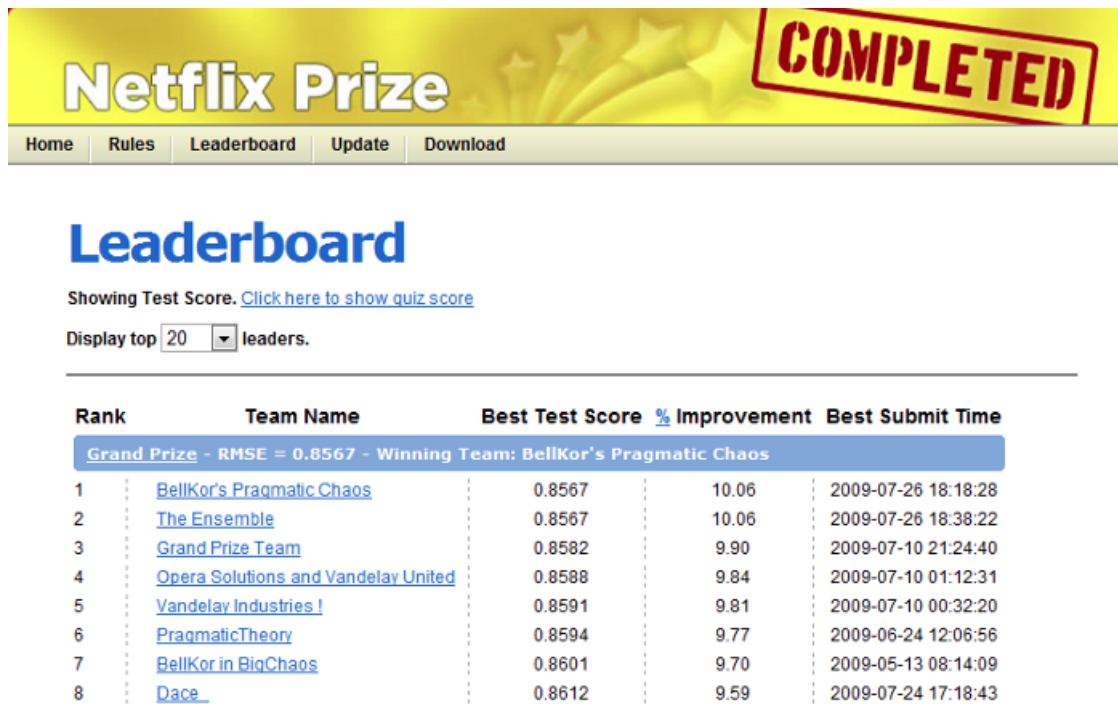


Figure C.1: Netflix \$1M machine competition rankings<sup>50</sup>

<sup>49</sup>available at: <https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429>

<sup>50</sup><https://netflixtechblog.com/netflix-recommendations-beyond-the-5-stars-part-1-55838468f429>, last accessed on 2021-01-21

## Appendix D

# Hybrid Algorithms' Pseudocode

---

**Algorithm 8** Random\_Split\_NCL\_Dropout

---

**Input:** size of random split training dataset  $r$ , number of estimators  $M$ , Dropout percentage  $p$

**Output:** final prediction  $p$

- 1: create  $M$  NN estimators with  $p$  value
  - 2: **for**  $m = 1$  to  $M$  **do**
  - 3:     perform NCL procedure between estimator  $m$  and  $L[: m]$
  - 4:     estimators on a random train\_subset with contiguous data samples of size  $r$
  - 5: **end for**
  - 6: use Simple Averaging to integrate saved estimators' test\_set predictions
- 

---

**Algorithm 9** Cross\_Training\_NCL\_Dropout

---

**Input:** number of estimators  $M$ , Dropout percentage  $p$

**Output:** result  $Y_M(x)$

- 1: create  $M$  NN estimators with  $p$  value
  - 2: divide training dataset in  $M$  folds
  - 3: **for**  $m = 1$  to  $M$  **do**
  - 4:     train estimator  $m$  on  $M - 1$  folds using NCL procedure between
  - 5:         estimator  $m$  and  $L[: m]$  estimators
  - 6: **end for**
  - 7: use Simple Averaging to integrate saved estimators' test\_set predictions
- 

---

**Algorithm 10** Random\_Subspace\_NCL\_Dropout

---

**Input:** number of random selected features  $f$ , number of estimators  $M$ , Dropout percentage  $p$

**Output:** final prediction  $p$

- 1: create  $M$  NN estimators with  $p$  value
  - 2: **for**  $m = 1$  to  $M$  **do**
  - 3:     perform NCL procedure between estimator  $m$  and  $L[: m]$  estimators
  - 4:     on a train\_set with  $f$  features
  - 5: **end for**
  - 6: use Simple Averaging to integrate saved estimators' test\_set predictions
-

---

**Algorithm 11** Pasting\_NCL\_Dropout

---

**Input:** size of pasting training dataset  $p$ , number of estimators  $M$ , Dropout percentage  $p$ **Output:** final prediction  $p$ 

- 1: create  $M$  NN estimators with  $p$  value
  - 2: **for**  $m = 1$  to  $M$  **do**
  - 3:     perform NCL procedure between estimator  $m$  and  $L[: m]$
  - 4:     estimators on a random train\_subset of size  $p$
  - 5: **end for**
  - 6: use Simple Averaging to integrate saved estimators' test\_set predictions
- 

---

**Algorithm 12** Random\_Patches\_NCL\_Dropout

---

**Input:** number of random selected features  $f$ , size of pasting training dataset  $p$ , number of estimators  $M$ , Dropout percentage  $p$ **Output:** final prediction  $p$ 

- 1: create  $M$  NN estimators with  $p$  value
  - 2: **for**  $m = 1$  to  $M$  **do**
  - 3:     perform NCL procedure between estimator  $m$  and  $L[: m]$
  - 4:     estimators on a random train\_subset of size  $p$  with  $f$  features
  - 5: **end for**
  - 6: use Simple Averaging to integrate saved estimators' test\_set predictions
- 

---

**Algorithm 13** Horizontal\_Averaging\_NCL\_Dropout

---

**Input:** number of estimators  $M$ , number of training epochs  $N$ , Dropout percentage  $p$ **Output:** result  $Y_M(x)$ 

- 1: create  $M$  NN estimators with  $p$  value
  - 2: **for**  $n = 1$  to  $N$  **do**
  - 3:     **if**  $(N - n) < M$  **then**
  - 4:         save copy of current original model
  - 5:         perform one epoch of NCL procedure between original model and
  - 6:         saved estimators on a train\_set
  - 7:     **else**
  - 8:         train original model one epoch on a train\_set
  - 9:     **end if**
  - 10: **end for**
  - 11: use Simple Averaging to integrate saved estimators' test\_set predictions
-

---

**Algorithm 14** Polyak\_Averaging\_NCL\_Dropout

---

**Input:** number of estimators  $M$ , number of training epochs  $N$ , Dropout percentage  $p$ **Output:** result  $Y_M(x)$ 

```

1: create  $M$  NN estimators with  $p$  value
2: for  $n = 1$  to  $N$  do
3:   if  $(N - n) < M$  then
4:     save copy of current original model
5:     perform one epoch of NCL procedure between original model and
6:       saved estimators on a train_set
7:   else
8:     train original model one epoch on a train_set
9:   end if
10: end for
11: average saved estimators into a single estimator
12: use single estimator to make test_set predictions

```

---



---

**Algorithm 15** Bagging\_NCL\_Dropout

---

**Input:** training sample  $S$ , number of estimators  $M$ , Dropout percentage  $p$ **Output:** final prediction  $p$ 

```

1: create  $M$  NN estimators with  $p$  value
2: for  $n = 1$  to  $N$  do
3:    $S_m =$  bootstrap sample from  $S$ 
4:   if  $(N - n) < M$  then
5:     perform one epoch of NCL procedure between original model and
6:       saved estimators on a  $S_m$  set
7:     save copy of current original model
8:   else
9:     train original model one epoch on a  $S_m$ 
10:  end if
11: end for
12: use Simple Averaging to integrate saved estimators' test_set predictions

```

---

---

**Algorithm 16** Blending\_Snapshot\_NCL\_Dropout

---

**Input:** number of estimators  $M$ , number of training epochs  $N$ , Dropout percentage  $p$ **Output:** result  $Y_M(x)$ 

```

1: create  $M$  NN estimators with  $p$  value
2: for  $n = 1$  to  $N$  do
3:   if  $(N - n) < M$  then
4:     save copy of current original model
5:     perform one epoch of NCL procedure between original model and
6:       saved estimators on a train_set
7:   else
8:     train original model one epoch on a train_set
9:   end if
10: end for
11: fit Blending's linear meta-model with saved estimators' val_set predictions
12: use Blending's meta-model to perform test_set predictions

```

---



---

**Algorithm 17** Super\_Learner\_Snapshot\_NCL\_Dropout

---

**Input:** number of estimators  $M$ , number of training epochs  $N$ , Dropout percentage  $p$ , number of folds  $F$ **Output:** result  $Y_M(x)$ 

```

1: create  $M$  NN estimators with  $p$  value
2: divide training dataset in training and validation datasets
3: for  $n = 1$  to  $N$  do
4:   if  $(N - n) < M$  then
5:     save copy of current original model
6:     perform one epoch of NCL procedure between original model and
7:       saved estimators on a train_set
8:   else
9:     train original model one epoch on a train_set
10:  end if
11: end for
12: divide validation dataset in  $F$  folds
13: for  $f = 1$  to  $F$  do
14:   save  $k$  fold test set target
15:   for  $m = 1$  to  $M$  do
16:     train estimator  $m$  on  $f - 1$  folds using NCL procedure between
17:       estimator  $m$  and  $L[: m]$  estimators
18:     save  $k$  fold test set estimator  $m$  prediction
19:   end for
20: end for
21: fit meta-model on previously saved estimators predictions and targets

```

---

## **Appendix E**

# **Scientific Paper**

---

# Ensembling Neural Networks for Regression

---

**João Aguilha**  
Faculty of Engineering,  
University of Porto,  
Portugal  
up201607930@fe.up.pt

João Mendes-Moreira  
Faculty of Engineering,  
University of Porto,  
Portugal  
LIAAD-INESC TEC,  
Portugal  
jmoreira@fe.up.pt

Tiago Mendes-Neves  
Faculty of Engineering,  
University of Porto,  
Portugal  
LIAAD-INESC TEC,  
Portugal  
up201406104@fe.up.pt

## Abstract

Ensemble Learning is a thriving research area in terms of predictive performance. Its algorithms use strategies that vary in complexity to combine models, leading to an increase in performance. Also, some techniques are base learner agnostic while others are not. For example, Negative Correlation Learning relies on Neural Networks. The proposed framework blends different strategies to generate Neural Network ensemble methods for regression, aiming to take the best of each strategy, hence boosting the ensemble performance. More specifically, it implements established ensemble algorithms, analyses their capabilities through bias-variance-covariance error decomposition, and creates many new techniques by combining different strategies with complementary strengths. Extensive experiments with multiple groups of different datasets showed that combining multiple strategies yields better results. The best-performing algorithms decreased model error, on average, from 12% to 17% versus their constituent strategies.

## 1 Introduction

Ensemble Learning is a successful research area in terms of predictive performance. The goal is to combine multiple base learners predictions in a single, more robust, prediction. Some of the methods that had initial success used simple yet effective approaches to generate ensembles (e.g., Bagging [2] and AdaBoost [9]). Later, the combination of some of these strategies, sometimes with slight modifications, resulted in new methods also very well succeeded (e.g., Random Forest [4] as a combination of Bagging and Randomized Trees, or MultiBoosting [31] as a combination of wagging, a variant of Bagging and AdaBoost).

Some of these methods were designed for unstable base learners, as is the case of Decision Trees and Neural Networks. Some other methods for a specific base learner, for instance, Random Forests were explicitly designed for Decision Trees, and Negative Correlation Learning [25] for Neural Networks. With the recent success of Neural Networks and Deep Learning, Ensemble Learning also received new contributions to the already existing ones. It is the case of Snapshot [14] and Dropout [28] as strategies used to generate Neural Network ensembles.

Motivated by the works of Geoffrey Webb [31], and Geoffrey Webb & Zijian Zheng [32] we present a hybrid framework to generate Neural Network Ensembles for regression.

The main contributions are:

- to use an ensemble error decomposition to find the most promising ways of combining the most complementary strategies in a single ensemble method;
- to construct new Neural Network regression ensembles that improve its constituent models' performances.



## 2 Revising Ensemble Learning for Regression

Ideally, base learners yield a degree of accuracy and diversity to make uncorrelated predictions so that the ensemble works appropriately. Specifically for Neural Networks, it means they converged to different local minima, making different prediction errors. If base learners make highly correlated predictions, the accuracy of the ensemble will hardly improve over a single base learner. If base learners do not make correct predictions, the ensemble will also struggle to make correct predictions. Building a good ensemble is an exercise of carefully balancing the correlation and accuracy of base learners' predictions.

There are different approaches to build an ensemble, such as training data, base model characteristics, base model generation strategy, and prediction integration mechanism. Varying the first three elements have the objective of promoting estimator diversity. It is possible to vary the base models' characteristics, as each definable parameter has its pros and cons both independently and in conjunction with others. There is a vast possibility space with multiple different options from which to choose. More specifically, network type, topology, different random initialization, random selection of mini-batches, training specificities, and different outcomes of non-deterministic implementations of Neural Networks are enough to promote diversity.

### 2.1 General Ensemble techniques

Varying each estimator's training data provides a different framing of the problem. Despite Bootstrapping not being an ensemble-specific strategy, it may be utilized by repeatedly sampling with replacement dataset instances. Also, Pasting uses a smaller dataset, without demanding contiguous data samples, for each estimator [3], Random Subspace employs a random input feature subspace policy on each estimator's dataset [13], and Random Patches merges Random Subspace with Pasting [20]. Random Splits repeatedly samples from a dataset with a random data split both train and test sets, with contiguous data samples, for each weak learner. K-fold Cross-Training splits the dataset into  $k$  equally sized folds, feeds each estimator a different set  $k - 1$  folds, and may test it on the remaining holdout fold [17]. Note that  $k$  is the number of estimators.

Bagging is a variance reduction ensemble algorithm that uses bootstrap samples to fit independent parallel base learners [2]. There are multiple Bagging-based architectures. For example, Random Forest [4] leverages unpruned Decision Trees fitted on bootstrap samples, random input feature selection at each split point, and a greedy algorithm for optimal split point selection. Also, Extra Trees harnesses unpruned Decision Trees fitted on the entire dataset, random input feature selection at each split point, and randomly selected split points [12].

Boosting is, primarily, a bias reduction ensemble algorithm that builds upon prior chain models, fixes current prediction errors through attention refocusing (updates the dataset), and learns how to optimize each model's advantages. As a result, it turns weak learners into strong learners. There are multiple Boosting-based approaches such as AdaBoost, which at each iteration solves a "local" optimization problem and assigns weights to the data points and estimators based on their shown ensemble error contribution and performance respectively [10]. Despite being developed for classification problems, Drucker [8] proposed a regression version named AdaBoost.R2.

The literature is prolific in Boosting techniques for Decision Trees. Gradient Boosting employs a numerical optimization problem to a differentiable arbitrary amount of loss functions, an additive model to append base learners, and a shrinkage term to control each estimator's influence [21] [22]. LightGBM (LGBM) leverages attention focusing on training examples resulting in a larger gradient while excluding the remaining, a type of automatic feature selection, and a similar concept to Dropout [15]. CatBoost leverages good default parameters, categorical feature support, and a Gradient Boosting scheme that tackles overfitting [7]. XGBoost harnesses a loss function with an added regularization term for overfitting reduction, custom optimization objectives and evaluation criteria for high flexibility, Decision Trees fitted on bootstrap samples, and random input feature selection at each split point [5]. Furthermore, it adds missing data handling, parallel/independent processing, and pruning to remove no positive gain splits.

However, optimal model weights and predictions are difficult to find; thus, meta-model approaches that generalize in a second space to discover the optimal route for estimator combination are worthy of exploring. Stacking fits a meta-model on the base models' out-of-fold predictions [33]. Blending fits

a linear meta-model on the base models' holdout set predictions [1]. Super Learner is an application to k-fold cross-validation where all models use the same k-fold data splits and a meta-model is fit on estimators' out-of-fold predictions [30].

## 2.2 Neural Network Specific Ensemble Techniques

Despite Dropout not being an ensemble-specific strategy, we can use it as such. It promotes diversity during the learning process of each Neural Network in the ensemble by randomly dropping a given percentage of nodes [28].

Snapshot generates an ensemble with estimators that visited multiple local minima but not necessarily from contiguous training epochs [14]. Stochastic Gradient Descent with Warm Restarts (SGDR) promotes even greater Snapshot diversity. This approach aggressively cycles the learning rate, thus avoiding individual estimators getting stuck in the same local minima [19] [27]. Another alternative that stems from Snapshot is Polyak Averaging, which averages into a single Network multiple sets of noisy weights from contiguous training epochs close to the end of a training run [24].

Negative Correlation Learning, motivated by the works of Naonori Ueda and Ryohei Nakano [29], is a reliable strategy to promote mutual model diversity and lower the correlation between base learners predictions. When generating a model for the ensemble, the added penalty term to the Neural Network's objective function promotes a negative correlation between the new model and the previously generated models [25] [18].

There are various options for varying the base models' prediction integration strategy. Simple Averaging combines independent base learner's predictions with equally distributed weights [6]. Simple Averaging of models extracted from contiguous training epochs close to the end of a training run is defined as Horizontal Averaging [34]. Still, it might be helpful to consider each respective base learners' demonstrated accuracy in determining the final result, thus using Weighted Average.

Adopted prediction generation and integration methods are summed up in Table 1.

## 2.3 Combining strategies for ensemble generation

There are multiple proposed approaches to combine strategies for generating ensembles, each offering different breakthroughs from those presented in this work.

MultiBoosting combines AdaBoost with wagging, a variant of Bagging using C4.5 as the base learners achieving better results and execution time than the constituent algorithms [31].

Multistrategy Ensemble Learning investigates the hypothesis that accuracy improvement is due to base learners' increased diversity. So three new multistrategy Ensemble Learning techniques were developed with results showing they are, on average, more accurate than their base strategies [32].

Cocktail ensemble uses a hybrid mechanism for combining multiple individual ensembles via pairwise combination with a regression error-ambiguity decomposition. In other words, it resembles an ensemble of ensembles. Results show the proposed approach outperforms the individual ensembles, two other methods of ensemble combination, and two state-of-the-art regression approaches [35].

## 2.4 Ensemble Error Decomposition

There are two main formulas for ensemble generalization error decomposition in regression. First, Krogh [17] decomposes the error metric in bias and variance, and the second by Ueda [29] decomposes it in bias, variance, and covariance. This last approach is employed since it decomposes the error in greater detail.

**Bias** measures the average difference between the ensemble's output and the base learner's outputs. **Variance** indicates the average disagreement between the base learner's outputs. The lower it is, the more stable, robust, and reliable the respective estimators become, and vice-versa. **Covariance** measures the pairwise difference between different base learners.

However, there are two caveats. First, the above error generalization decomposition equations cannot be directly applied to classification problems due to their categorical nature [16]. Second, the error decomposition equation assumes the use of Simple Averaging as the prediction integration strategy

Table 1: Overview of Generation Mode (GEN), Integration Method (INT), Base Learner used in our framework (BL), and main reference (REF) for the main ensemble methods for regression. Par stands for Parallel, Str for Stream (means that we generate the base models from the same original base learner), Seq for Sequential, SA for Simple Average, WA for Weighted Average, DT for Decision Trees, and NN for Neural Networks. The suffix  $X$  indicates the hyperparameter(s) of the method in the experiments.

Ensemble Algorithm	Tests Designation	GEN	INT	BL	REF
Simple Averaging	average	Par	SA	NN	[6]
Random Splits	rand_split0.X	Par	SA	NN	sec. 2.1
K-fold Cross-training	cross_training	Par	SA	NN	[17]
Random Subspace	rand_subspace0.X	Par	SA	NN	[13]
Pasting	pasting0.X	Par	SA	NN	[3]
Random Patches	pasting0.X +rand_subspace0.X	Par	SA	NN	[20]
Horizontal Averaging	horizontal_avg	Str	SA	NN	[34]
Polyak Averaging	polyak_avg	Str	SA	NN	[24]
Snapshot with Cosine Annealing Learning Rates	snapshot	Str	SA	NN	[14]
Negative Correlation Learning	ncl0.X	Seq	SA	NN	[25]
Dropout	dropoutX	Par	SA	NN	[28]
Bagging	bagging	Par	SA	NN	[2]
AdaBoost(.R2)	adaboost_scratch	Seq	WA	NN	[8]
AdaBoost(.R2)	adaboost_nn	Seq	WA	NN	[8]
Super Learner	super_learner	Par	WA	NN	[30]
Blending	blending	Par	WA	NN	[1]
Stacking	stacking_nn	Par	WA	NN	[33]
AdaBoost(.R2)	adaboost_default	Seq	WA	DT	[8]
Gradient Boosting	gradient_boosting	Seq	WA	DT	[21] [22]
Gradient Boosting Histogram	gradient_boosting_hist	Seq	WA	DT	[26]
LGBM	lgbm	Seq	WA	DT	[15]
CatBoost	catboost	Seq	WA	DT	[7]
XGBoost	xgboost	Seq	WA	DT	[5]
Random Forest	rand_forest	Par	SA	DT	[4]
Extra Trees	extra_trees	Par	SA	DT	[12]

[29]. This assumption might skew the bias, variance, and covariance distribution slightly when applying Weighted Averaging. Deriving this decomposition for Weighted Averaging would be more challenging due to the estimation of the ensemble error decomposition before the learning process, where the ensemble learns the weights.

### 3 A framework to construct Neural Network ensembles for regression

This framework has three steps. First, it evaluates the error decomposition of the Table 1 ensemble techniques in bias, variance, and covariance [29]. Second, it identifies the most complementary pairs of strategies in terms of ensemble error component reduction. Third, combines these pairs of strategies to create a multitude of new ensemble algorithms.

The following list presents the premises of our work:

- the Neural Network estimator uses a *ReLU* activation function in the input layer with a number of neurons according to the number of train set’s features, *MAE* as the loss function, and *AdaGrad* as the optimizer;
- the estimator’s structure and parameters are consistent throughout all ensemble algorithms and test levels for computational simplicity and comparison trustworthiness. In other words, the Network architecture is not a variable affecting the final results;
- each ensemble has five estimators;
- each ensemble algorithm has been averaged five times except for Snapshot (and derived architectures) which have been averaged 15 times for improved results stability;

- different datasets for each of the two levels of the framework;
- holdout with 80% for training and 20% for testing as resampling method;
- datasets' categorical values are one-hot encoded using *scikit - learn*<sup>1</sup>;
- missing values are removed by dropping their respective instances;
- we present two baselines: (1) a single Neural Network with the same architecture as the ensemble's base estimators, and (2) the simple average of the base learners results;
- each component on a stacked bar graph refers to the average of the normalized sum of that component for each dataset;
- the framework is powered by *Keras*<sup>2</sup> and *TensorFlow*<sup>3</sup>;
- although AdaBoost.R2, Gradient Boosting (Histogram), LGBM, CatBoost, and XGBoost use Decision Trees instead of Neural Networks as base learners, they have only been added in the Level-0 because some of their characteristics can add value in other algorithms, so they are used for performance comparison.

### 3.1 Level-0

Since this level's tests essentially consist of plainly implementing established ensemble algorithms (Table 1), we use ten datasets, which gives statistical confidence as the empirical results meet the theoretical assumptions.

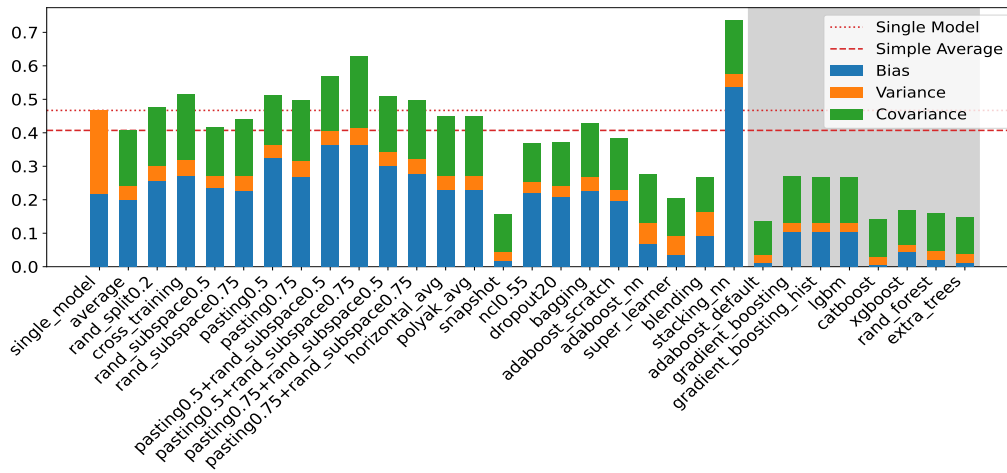


Figure 1: Level-0 results. Neural Network ensemble results have a white background, while decision tree ensemble results have a grey background.

Looking at Figure 1, and broadly speaking, it is clear to see that ensembles, almost as a whole, lower predictions' overall variance while keeping bias low at the same time. In this way, we lower generalization error and achieve better performance.

As expected, Simple Averaging achieved better results than the baseline. Due to their characteristics, Horizontal Averaging and Polyak Averaging performed poorly since contiguous epoch estimators suffer from a lack of diversity. Surprisingly Snapshot with SGDR (Cosine Annealing Learning Rates) performed exceptionally well in reducing covariance, variance, and specifically, bias. So it is possible to infer that aggressively varying the learning rate of Neural Networks is a successful way of promoting estimators' diversity. Other notable results were Dropout ( $p = 20\%$ ), and Negative Correlation Learning ( $lambda = 0.55$ ) that managed to lower error slightly comparing to the Average ensemble.

The base models' selection needs to be consistent with the way to aggregate them. If the base models have low bias and high variance, we should consider an aggregating scheme favoring reducing the

<sup>1</sup><https://scikit-learn.org/stable/>

<sup>2</sup><https://keras.io/>

<sup>3</sup><https://www.tensorflow.org/>

ensemble's variance (e.g., Bagging). We should consider an aggregating scheme favoring reducing the ensemble's bias (e.g., Boosting) when we have high bias and low variance base models. Given that the single model exhibits both a relatively high bias and variance, both mentioned ensemble strategies are worth exploring.

Surprisingly, Bagging failed to improve results comparing to Simple Averaging. One can conclude that bootstrap sampling faltered in promoting the expected estimator diversity in these specific datasets. Regarding Bagging variants, despite Random Forest performing pretty well by reducing variance, covariance, and most noticeably bias, Extra Trees came out on top. Nonetheless, given that these two variants performed positively, Random Subspace and other techniques that vary the training data are worth exploring for Neural Networks. The rest either (1) were already explored (models fitted on the full dataset), (2) had bad empirical results (bootstrapping), or (3) do not directly apply to Neural Networks (optimal split point selection algorithms).

On the other hand, Boosting achieved the best results throughout almost all different implementations. These results clearly show that refocusing subsequent models' attention on remaining difficult observations and using a weighted average final prediction according to each estimator's respective performance are exceptional ensemble strategies. AdaBoost.R2 manifested the best global error value closely followed by XGBoost. Curiously, CatBoost smashed bias but increased variance, while XGBoost smashed variance but failed to reduce bias as much as CatBoost.

Almost all Decision Tree Boosting methods reduced covariance and, very appreciable, bias and variance while Neural Network Boosting Methods lowered bias and covariance but increased variance. Comparing Decision Tree AdaBoost.R2 (default) with Neural Network AdaBoost.R2, the latter achieved worse results than the former. A possible avenue to improve the latter's performance is to vary its estimators' characteristics. Given that AdaBoost.R2, XGBoost, and CatBoost performed the best, adding regularization terms to the Neural Networks (e.g., Dropout) and Random Subspace are worth exploring. The rest either (1) do not directly apply to Neural Networks (default parameters, split points, and individual Neural Network pruning), (2) do not apply to the specific baseline (does not suffer from overfitting), (3) do not offer advantages due to the dataset characteristics (no categorical features and no missing data), (4) have already been explored (parallel/independent processing) or (5) perform poorly on Neural Networks (bootstrapping).

Note that tree-based ensembles' results are better than the Neural Network ensembles' results because (1) it is easier for Decision Trees to perform well on default hyperparameters, (2) the adopted Neural Network estimator architecture is very simple and trained on few epochs, and (3) some datasets may have too few samples.

Meta-model techniques depend on having a simple model that provides smooth prediction interpretations that offset individual models' deficiencies for better performance. For that reason, Super Learner and Blending, which use a linear meta-model, achieved the best results, with the former edging out the latter. However, both obtained the worst variance results. Stacking, which uses a non-linear meta-model (Neural Network), had disappointing results. Hence, linear meta-models are, in this case, superior to non-linear meta-models. Also, one might wonder that Blending's characteristics (fit a linear meta-model on estimators' holdout set predictions) lower bias but promote variance. Super Learner (out-of-fold predictions during k-fold cross-validation) also lowers bias and increases variance but at a larger scale than Blending. However, Super Learner's results still indicate a better global error reduction when compared with Stacking and Blending (sec. 2.1).

Some varying training data ensemble techniques offer positive characteristics, except for Bootstrapping (Bagging) and K-fold Cross-training. Also, Random Splits' (sec. 2.1) increased estimator diversity did not compensate for the loss of available training data. Pasting, Random Subspace, and Random Patches suffered from a lack of samples and features to select different subsets or combinations. Therefore, we performed more thorough testing with selected datasets with enough samples/features to obtain more insightful results. Results show that these techniques alone do not promote better performance.

Nevertheless, varying the number of features above a certain threshold is slightly better than varying the number of samples for individual estimators. When combined, these techniques harm performance. However, there is a caveat. Performance improvements using these techniques are very dependant (1) on the dataset and (2) on the randomness of the selected variables. So these techniques may still be helpful in particular circumstances.

Bottom line, this level’s best-performing methods were Snapshot with SGDR, Super Learner, and Blending ensembles. Additionally, since Dropout and NCL achieved better performance than the Simple Averaging ensemble and could combine with other algorithms, we considered them plausible and promising candidates for the next level’s experiments.

### 3.2 Level-1

In this level, we merge the previous level’s best-performing methods and promising approaches. We use twenty datasets different from those used in the previous level to ensure independent results.

However, first, it is required to discover the best  $p$  and  $lambda$  hyperparameter values, respectively, for Dropout and NCL. After extensive testing on Dropout ensemble and Dropout empowered Neural Network ensembles, we found  $p = 15$  to be the best value, with  $p = 20$  following right behind. Analogously, after extensive testing on NCL ensemble and NCL empowered Neural Network ensembles, we found  $lambda = 0.55$  to be the best value.

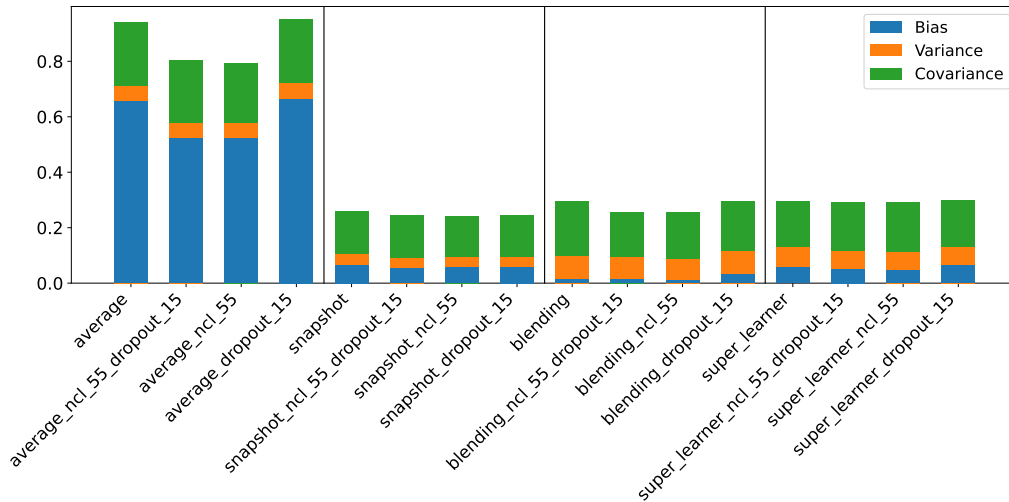


Figure 2: Level-1 results. Dropout, NCL, and Dropout+NCL empowered Level-0 ensembles.

Figure 2 compares the effects of joining techniques of the previous level’s best-performing methods. Individual (non-merged) strategies serve as the comparison metric. It is clear to see that Snapshot, Blending, and Super Learner benefit, typically, from being empowered with both NCL and Dropout, both individually and in conjunction. Also, if NCL has better results than Dropout, then NCL+Dropout usually has worse performance than that of NCL but better than Dropout, and vice-versa. Regarding the Snapshot ensemble, joining it with Dropout offers the best results. Oppositely, meta-model strategies tend to benefit from NCL alone, and when Dropout is employed, it worsens their performance. As a side note, Simple Averaging stays in the middle ground, meaning that it is improved the most by using NCL with Dropout.

Diving deeper into each error component, NCL alone achieves the best results in bias reduction comparing to Dropout, either alone or in combination. The case in which it did not was on Snapshot ensemble, ending in second place and closely behind NCL+Dropout. Looking at variance, typically, Dropout reduces it the most in Simple Averaging and Snapshot ensembles, while meta-model approaches prefer the NCL+Dropout combination. Finally, covariance in meta-model strategies is reduced the most by NCL, whether in Snapshot methods Dropout gives the best results.

Regarding the specific numerical results, *snapshot\_dropout\_15* compared to *snapshot* reduced error by 13.5%, *blending\_dropout\_15* compared to *blending* reduced error by 12.1%, and *super\_learner\_dropout\_15* compared to *super\_learner* reduced error by 16.9%.

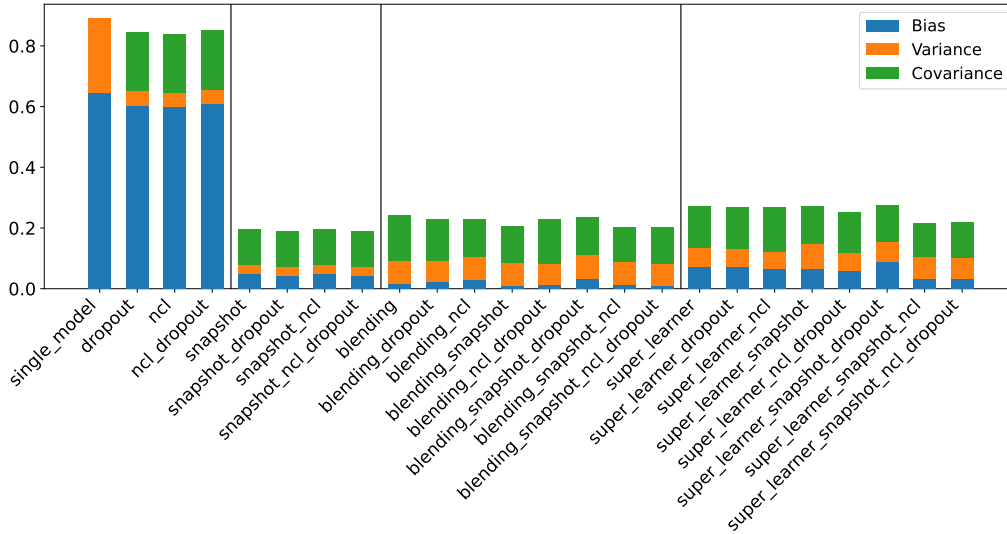


Figure 3: Level-1 results. Dropout, NCL, Snapshot, Blending, and Super Learner algorithm combination.

Figure 3 shows the results of combining all compatible previous level’s best-performing and promising methods. Results show that stream models benefit the most from employing Dropout since it acts from the beginning of their training while NCL has too few epochs, close to the training run’s end, to promote mutual model diversity. On the other hand, independent models benefit the most from NCL because it promotes mutual model diversity from the ground up.

Looking at individual methods, Dropout and NCL had similar performance outcomes as both finished last. The latter decreased bias the most while the former reduced variance and covariance slightly more. Snapshot performs the best by reducing covariance ponderously and squashing bias and variance, thus acting as a bias/variance reduction algorithm. Blending, which ended second performance-wise, reduced covariance, but more importantly, smashed bias at the expense of a more pronounced increase in variance hence categorized as a bias reduction algorithm. Super\_Learner acts as the middle ground between Snapshot and Blending by reducing bias more than the former but less than the latter and variance otherwise. Its covariance reduction was similar to that of Blending.

Examining the framework’s combination mechanism, Dropout adds to each algorithm’s estimators, Dropout layers with a specific  $p$  value. NCL trains the algorithm’s estimators following the initially proposed method, and Snapshot generates many stream estimators. Optionally, once harvested, we can train these stream estimators with an NCL policy to promote diversity. We can also use NCL without Snapshot. In this case, the base learners are entirely independent. Dropout, Snapshot, and NCL (combined) algorithms’ estimators may be mutually integrated with a simple averaging approach or merged with a linear meta-model algorithm. If we use Blending, the estimators make holdout set predictions that fit the meta-algorithm. If Super Learner is adopted, the estimators make out-of-fold predictions during k-fold cross-validation that fit the meta-algorithm.

Turning attention to the combined algorithms results, *snapshot\_ncl\_dropout* was the best across the board, performance-wise, since (1) Snapshot already possessed the best individual model performance and (2) NCL and Dropout further helped in lowering every error component. Regarding Blending, merging it with Dropout or NCL reduces variance and covariance but raises bias, especially with NCL. On the other hand, merging Blending with Snapshot lowers bias and covariance but increases variance. So, the various Blending combination possibilities typically exhibit the referred combined characteristics that contribute to *blending\_snapshot\_ncl\_dropout* having the best performance in the subset of Blending techniques.

Combining Super\_Learner with Dropout lowers bias and variance, and combining it with NCL lowers both metrics even further, in particular, variance at the expense of slightly raising covariance. Merging Super\_Learner with Snapshot strongly reduced covariance but increased bias and, especially, variance. Consequentially, the multiple Super Learner combination possibilities typically exhibit the referred

combined characteristics that contribute to *super\_learner\_snapshot\_ncl\_dropout* having the best performance in the subset of Super Learner techniques.

Regarding the specific numerical results, *snapshot\_ncl\_dropout* compared to *snapshot* reduced error by 10.1%, *blending\_snapshot\_ncl\_dropout* compared to *blending* reduced error by 15.6%, and *super\_learner\_snapshot\_ncl\_dropout* compared to *super\_learner* reduced error by 13.5%.

On the whole, every Snapshot combination outperformed all other Blending and Super Learner proposed architectures. Nonetheless, it does not make Snapshot the absolute winner in all circumstances. Since Blending and Super Learner offer a second space generalization, improving their current meta-model or adding subsequent generalization levels is possible. Given that the results difference between Snapshot and Blending/Super Learner best-performing combination models are not that dissimilar, it is reasonable to assume that subsequent improvements to these meta-model variants would likely outperform Snapshot. A counterargument is to harvest Snapshot's models at detached epochs and improve the existing Snapshot's varying learning rate policy.

### 3.3 Statistical Validation

Friedman test is employed to ascertain if repeated, and related measurements consistently follow the same distribution [11]. Its null hypothesis is that the multiple paired samples follow the same distribution. This assumption's rejection means that at least one paired sample follows a different distribution.

Every experiment rejected the null hypothesis by surpassing the defined type I error of 0.05. So, it proved that the observation values (ensemble architecture's errors) from multiple runs (different datasets) have different means, follow different distributions, and, inherently, are statically valid. In other words, there is enough proof to conclude that different types of ensemble algorithms lead to statistically significant differences in their global error values.

Given that the experiments' p-values are statistically significant, Nemenyi posthoc test, which returns the p-values for each pairwise mean comparison, can be executed to determine the specific groups with different means [23]. Results revealed that many algorithms have a high pairwise  $p$  value denoting they have statistically similar means. The reason is that most algorithms' global error values are clustered around two groups, high and low values, without many in between. Naturally, algorithms inside those clusters will have a high  $p$  value between one another while comparing algorithms that belong to different clusters show a low mutual  $p$  value. Note that high and low values mean better performing and worse performing algorithms, respectively.

## 4 Conclusions and future work

The fundamental idea in this paper was to prove the advantage of combining already established Neural Network ensemble strategies in regression problems to improve their performance. So, we created multiple new architectures with the help of a hybrid ensemble combining framework.

We discuss each technique's merits, with only some consistently improving the Simple Averaging ensemble's bias-variance-covariance metrics. Experimental results confirm performance increases from combining multiple algorithms as the developed ensembles showed better performance than the original constituent counterparts. Merging the best-performing methods resulted in a lower global error value, as each technique was complementary in lowering one or more ensemble error components.

Finally, although we dealt with Neural Networks, all the theory and rationale holds for other types of ensemble estimators, e.g., Decision Trees or KNN. In addition to this, the developed framework can also be used as a proxy to generate ensembles for classification despite the ensemble error decomposition being specific for regression.

The code and datasets used to produce this work are available in the supplementary material. Regarding the broader impact of this work: the advances presented in this paper are not socially good or bad inherently; we propose new methods that are complementary to the currently existing ones. Likewise, the social impact depends on the use case rather than the method itself.



## A Code

The developed code is hosted at <https://anonymous.4open.science/r/DISS-26C3/>, along with all the required datasets and evaluation procedures for as long as it is needed.

## B Error decomposition

This section explains in detail the implemented ensemble error decomposition formula as depicted in [29].

$$f_{ens}^{(M)}(x) = \frac{1}{M} \sum_{m=1}^M f_m(x; z_{(m)}^N)$$

where Let  $f_1, \dots, f_M$  denote  $M$  estimators, where the  $m$ th estimator is separately trained on  $z_{(m)}^N$ ,  $m = 1, \dots, M$

The training set's sample size is assumed to be uniformly  $N$ . Note that the training set  $z_{(m)}^N$ , is a realization of a random sequence  $Z_{(m)}^N$  and that  $Z_{(m)}^N$ ,  $m = 1, \dots, M$ , have the same distribution  $p(x, y)$ ; however, we cannot always assume them to be mutually independent. The ensemble output for some input  $x$  is the linear average of  $M$  estimators' outputs for  $x$  after being separately trained.

So, we have the following decomposition for the ensemble generalization error [29]:

$$f_{ens}^{(M)}(x) = \frac{1}{M} \sum_{m=1}^M f_m(x; z_{(m)}^N)$$

$$GErr(f_{ens}^{(M)}) = E_{x_o} \left\{ \frac{1}{M} \overline{Var}(X_o) + \left(1 - \frac{1}{M}\right) \overline{Cov}(X_o) + \overline{Bias}(X_o)^2 \right\} + \sigma^2$$

where  $\overline{Var}(X_o)$ ,  $\overline{Cov}(X_o)$ ,  $\overline{Bias}(X_o)$ , and  $\sigma^2$  are average conditional variance, conditional covariance, conditional bias averaged over  $M$  estimators and noise, respectively:

$$\begin{cases} \overline{Var}(X_o) = \frac{1}{M} \sum_{m=1}^M Var\{f_m|X_o\} \\ \overline{Cov}(X_o) = \frac{1}{M(M-1)} \sum_m \sum_{m' \neq m} Cov\{f_m, f_{m'}|X_o\} \\ \overline{Bias}(X_o) = \frac{1}{M} \sum_{m=1}^M Bias\{f_m|X_o\} \end{cases}$$

## C Datasets

This section thoroughly lists every dataset used in the Scientific paper's Level-0 (Table 2) and Level-1 (Tables 3 and 4) experiments, their respective characteristics, and changes made to them.

Given that every collected dataset is entirely and freely available/maintained online, confidentiality issues are not a thing. As such, they are licensed under the CC0 license. Nonetheless, they should be used responsibly and ethically. Therefore, we bear the responsibility for rights violations or infringements regarding the datasets and adherence to the data license.

Regarding data organization, we use the csv file format regarding the datasets. However, some of them were in different formats, namely, txt, so they had to be converted. They fall under the category of regression modeling tasks. Categorical values are one-hot encoded, missing values are removed by dropping their respective instances, and features are scaled not to skew the algorithm's internal workings.

Note that we utilized the provided URLs to gather the datasets, but they may also be collected from various other sources. Despite that, the referred URLs are stable, meaning they are expected to host the data indefinitely. Also, the data is available/maintained in the datasets folder accessible by the URL from Appendix A.

Table 2: Level-0 Datasets' Information

name	no. of samples	no. of features	categorical features	missing data	origin
fried delve	40768	10	no	no	1
energydata complete	19735	29	yes	no	2
bike sharing/hour	17389	16	yes	no	2
friedman	1200	5	no	no	3
mv	40768	10	no	no	3
atlname1004a	17812	7	yes	no	4
triazines	186	61	no	no	5
fruitfly	125	5	no	no	5
student	395	31	no	no	2
add10	9792	11	no	no	6

Table 3: Level-1 Datasets' Information - 1

name	no. of samples	no. of features	categorical features	missing data	origin
auto price	159	16	no	no	7
auto mpg	398	8	no	yes	7
cpu act	8192	22	no	no	7
cpu small	8192	13	no	no	7
housing boston	506	14	no	no	7
housing california	20460	9	no	no	7
machine cpu	209	7	no	no	7
pole telecomm	15000	49	no	no	7
stock airplane companies	950	10	no	no	7

<sup>1</sup><https://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html>

<sup>2</sup><https://archive.ics.uci.edu/ml/datasets.php>

<sup>3</sup><https://sci2s.ugr.es/keel/category.php?cat=reg#sub2>

<sup>4</sup><http://users.stat.ufl.edu/~winner/data/atlname1004a.dat>

<sup>5</sup>[https://www.openml.org/search?sort=runs&order=desc&ctype=task&from=100&q=+tasktype.tt\\_id%3A2](https://www.openml.org/search?sort=runs&order=desc&ctype=task&from=100&q=+tasktype.tt_id%3A2)

<sup>6</sup><https://www.cs.toronto.edu/~delve/data/>

<sup>7</sup><https://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html>

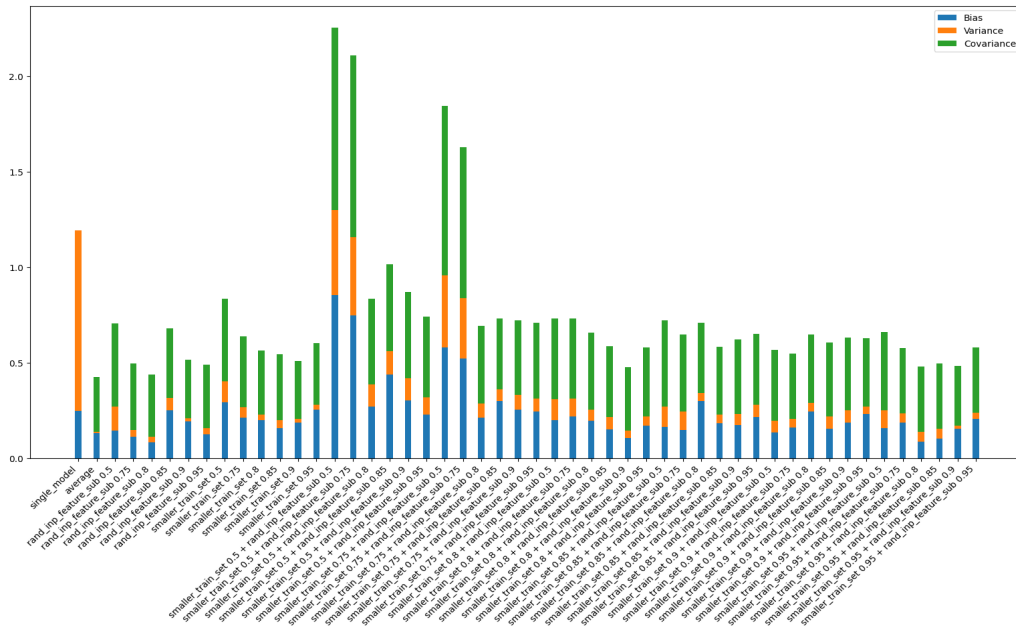
Table 4: Level-1 Datasets' Information - 2

name	no. of samples	no. of features	categorical features	missing data	origin
wisconsin	194	33	no	no	7
breast cancer					
ailerons	13750	41	no	no	7
airfoil self-noise	1503	6	no	no	8
combined cycle					
power plant	9568	5	no	no	8
concrete					
compressive strength	1030	9	no	no	10
real estate valuation	414	8	no	no	8
yacht hydrodynamics	308	7	no	no	8
insurance	1338	7	yes	no	9
electrical maintenance	1056	5	no	no	10
house 16h	22784	17	no	no	10
pole telecommunications	14998	27	no	no	10

## D Vary training data

This section presents the comparison results of bias, variance, and covariance for the Scientific paper's Level-0's data varying techniques (Figure 4).

Figure 4: Level-0 Pasting, Random Subspace and Random Patches results



<sup>8</sup><https://archive.ics.uci.edu/ml/datasets.php>

<sup>9</sup><https://www.kaggle.com/mirichoi0218/insurance>

<sup>10</sup><https://sci2s.ugr.es/keel/category.php?cat=reg#sub2>

## E Hybrid algorithms

This section depicts the pseudocode of the Scientific paper's Level-1's developed hybrid algorithms.

---

**Algorithm 1** Snapshot\_NCL\_Dropout

---

**Input:** number of estimators  $M$ , number of training epochs  $N$ , Dropout percentage  $d$

**Output:** final prediction  $p$

- 1: create  $M$  neural network estimators with  $d$  value
  - 2: **for**  $n = 1$  to  $N$  **do**
  - 3:     **if**  $(N - n) < M$  **then**
  - 4:         save copy of current original model
  - 5:         perform one epoch of NCL procedure between original model and
  - 6:         saved estimators on a train\_set with a SGDR policy
  - 7:     **else**
  - 8:         train original model one epoch on a train\_set with a SGDR policy
  - 9:     **end if**
  - 10: **end for**
  - 11: use simple averaging to integrate saved estimators' test\_set predictions
- 

---

**Algorithm 2** Super\_Learner\_Snapshot\_NCL\_Dropout

---

**Input:** number of estimators  $M$ , number of training epochs  $N$ , Dropout percentage  $d$ , number of folds  $F$

**Output:** final prediction  $p$

- 1: create  $M$  neural network estimators with  $d$  value
  - 2: divide training dataset in training and validation datasets
  - 3: **for**  $n = 1$  to  $N$  **do**
  - 4:     **if**  $(N - n) < M$  **then**
  - 5:         save copy of current original model
  - 6:         perform one epoch of NCL procedure between original model and
  - 7:         saved estimators on a train\_set
  - 8:     **else**
  - 9:         train original model one epoch on a train\_set
  - 10:     **end if**
  - 11: **end for**
  - 12: divide validation dataset in  $F$  folds
  - 13: **for**  $f = 1$  to  $F$  **do**
  - 14:     save  $k$  fold test set target
  - 15:     **for**  $m = 1$  to  $M$  **do**
  - 16:         train estimator  $m$  on  $f - 1$  folds using NCL procedure between
  - 17:         estimator  $m$  and  $L[:m]$  estimators
  - 18:         save  $k$  fold test set estimator  $m$  prediction
  - 19:     **end for**
  - 20: **end for**
  - 21: fit meta-model on previously saved estimators predictions and targets
-

---

**Algorithm 3** Blending\_Snapshot\_NCL\_Dropout

---

**Input:** number of estimators  $M$ , number of training epochs  $N$ , Dropout percentage  $d$ **Output:** final prediction  $p$ 

```
1: create  $M$  neural network estimators with  $d$  value
2: for  $n = 1$  to  $N$  do
3:   if  $(N - n) < M$  then
4:     save copy of current original model
5:     perform one epoch of NCL procedure between original model and
6:       saved estimators on a train_set
7:   else
8:     train original model one epoch on a train_set
9:   end if
10: end for
11: fit Blending's linear meta-model with saved estimators' val_set predictions
12: use Blending's meta-model to perform test_set predictions
```

---

## F Numerical results

This section thoroughly lists every error component value of the Scientific paper's Level-0 (Table 5) and Level-1 (Tables 6 and 7) experiments.

Table 5: Scientific paper Figure 1 numerical results.

Algorithm	$Bias^2$	Variance	Covariance	Error
single_model	0.401	0.896	0.004	1.229
average	0.336	0.051	0.584	0.933
rand_split 0.2	0.413	0.062	0.628	1.059
cross_training	0.458	0.062	0.623	1.083
rand_subspace 0.5	0.353	0.049	0.566	0.941
rand_subspace 0.75	0.38	0.074	0.65	1.047
pasting 0.5	0.464	0.064	0.611	1.119
pasting 0.75	0.438	0.071	0.654	1.126
pasting 0.5 + rand_subspace 0.5	0.529	0.084	0.672	1.256
pasting 0.5 + rand_subspace 0.75	0.589	0.084	0.705	1.314
pasting 0.75 + rand_subspace 0.5	0.46	0.074	0.66	1.159
pasting 0.75 + rand_subspace 0.75	0.442	0.073	0.652	1.126
horizontal_avg	0.39	0.064	0.663	1.064
polyak_avg	0.39	0.064	0.664	1.065
snapshot	0.044	0.019	0.481	0.526
ncl 0.55	0.303	0.048	0.554	0.89
bagging	0.363	0.054	0.587	0.967
adaboost_scratch	0.311	0.041	0.577	0.89
adaboost_nn	0.173	0.061	0.522	0.758
super_learner	0.1	0.157	0.546	0.745
blending	0.173	0.274	0.527	0.907
stacking_nn	0.701	0.078	0.665	1.442
dropout 20	0.302	0.031	0.538	0.851
adaboost_default	0.025	0.018	0.491	0.519
gradient_boosting	0.17	0.034	0.58	0.717
gradient_boosting_hist	0.166	0.027	0.569	0.7
lgbm	0.166	0.027	0.57	0.7
catboost	0.029	0.105	0.539	0.647
xgboost	0.062	0.013	0.479	0.544
rand_forest	0.057	0.042	0.557	0.619
extra_trees	0.042	0.028	0.517	0.554

Table 6: Scientific paper Figure 2 numerical results.

Algorithm	$Bias^2$	Variance	Covariance	Error
average	0.936	0.376	0.653	1.965
average_ncl_55_dropout_15	0.753	0.472	0.69	1.915
average_ncl_55	0.74	0.499	0.731	1.97
average_dropout_15	0.949	0.371	0.67	1.99
snapshot	0.152	0.109	0.266	0.526
snapshot_ncl_55_dropout_15	0.13	0.12	0.249	0.499
snapshot_ncl_55	0.134	0.1	0.239	0.472
snapshot_dropout_15	0.141	0.087	0.227	0.455
blending	0.02	0.693	0.387	1.101
blending_ncl_55_dropout_15	0.018	0.678	0.278	0.974
blending_ncl_55	0.01	0.703	0.255	0.968
blending_dropout_15	0.052	0.693	0.398	1.143
super_learner	0.086	0.574	0.361	1.022
super_learner_ncl_55_dropout_15	0.091	0.441	0.352	0.884
super_learner_ncl_55	0.079	0.457	0.313	0.849
super_learner_dropout_15	0.101	0.457	0.32	0.878

Table 7: Scientific paper Figure 3 numerical results.

Algorithm	$Bias^2$	Variance	Covariance	Error
single_model	0.87	0.941	0.0	1.812
dropout	0.83	0.077	0.841	1.748
ncl	0.814	0.085	0.863	1.762
ncl_dropout	0.837	0.081	0.843	1.761
snapshot	0.101	0.025	0.549	0.674
snapshot_dropout	0.093	0.017	0.511	0.621
snapshot_ncl	0.1	0.017	0.514	0.631
snapshot_ncl_dropout	0.093	0.012	0.501	0.606
blending	0.016	0.208	0.622	0.846
blending_dropout	0.021	0.184	0.579	0.784
blending_ncl	0.03	0.172	0.533	0.735
blending_snapshot	0.006	0.215	0.52	0.741
blending_ncl_dropout	0.02	0.18	0.592	0.792
blending_snapshot_dropout	0.032	0.241	0.55	0.824
blending_snapshot_ncl	0.012	0.207	0.502	0.72
blending_snapshot_ncl_dropout	0.007	0.196	0.511	0.714
super_learner	0.098	0.133	0.608	0.839
super_learner_dropout	0.084	0.127	0.6	0.811
super_learner_ncl	0.082	0.107	0.624	0.813
super_learner_snapshot	0.11	0.201	0.535	0.845
super_learner_ncl_dropout	0.087	0.117	0.587	0.791
super_learner_snapshot_dropout	0.122	0.185	0.552	0.859
super_learner_snapshot_ncl	0.035	0.182	0.509	0.726
super_learner_snapshot_ncl_dropout	0.034	0.173	0.51	0.716

## G Detailed dataset results

This section presents for both the Scientific paper’s Level-0 and Level-1, each datasets individual error results for every algorithm through the use of a heatmap.

Note that the right y-axis of heatmap graphs refers to the percentage difference between a given line’s algorithm on a particular dataset and the single model. -1.0 means the error has diminished 100%, and 3.0 means the error increased at least 300%.

Analyzing the individual dataset error results (Figure 5), it is clear that most algorithms consistently perform across the datasets, either good or bad, with few outliers (except Gradient Boosting and LGBM).

Two findings are worth noting. First, with the existing configuration, stacking performs poorly across all datasets, and on some occasions, it behaves pretty disastrously. Second, LGBM and Gradient Boosting algorithms have similar error values across all datasets since the former stems from the latter.

Figure 5: Scientific paper Figure 1 heatmap error results.



Analyzing the individual dataset error results (Figures 6 and 7), it is clear that most algorithms consistently perform across the datasets, either good or bad.

It is important to note that there was a case in which every proposed algorithm performed worse than the baselines but since this was an isolated case, it is considered an outlier.

Figure 6: Scientific paper Figure 2 heatmap error results.

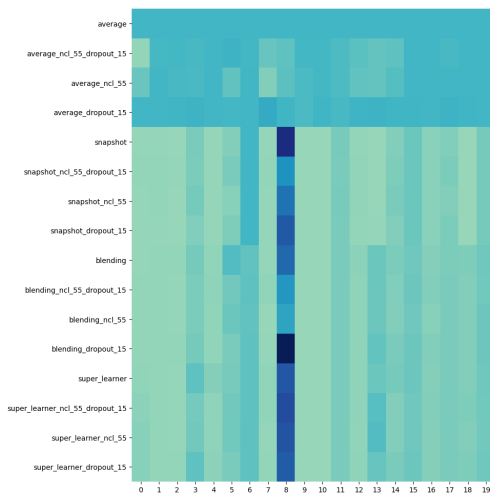
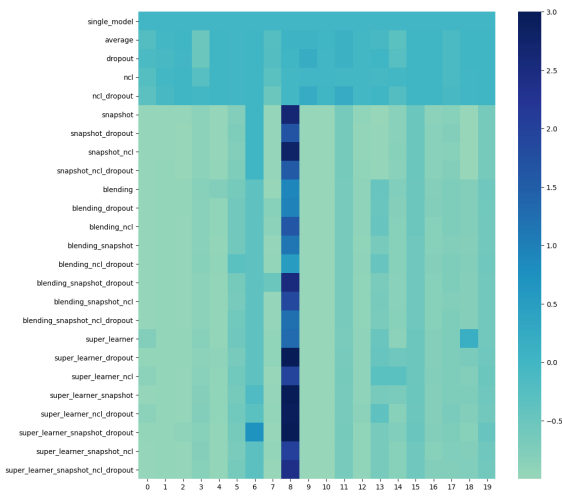


Figure 7: Scientific paper Figure 3 heatmap error results.



## H Statistical tests

This section thoroughly depicts every Friedman (Table 8) and Nemenyi (Figures 8, 9, 10, and 11) test employed in the Scientific paper's Level-0 and Level-1 experiments.

Note that the right y-axis of heatmap graphs refers to the mean similarity percentage between a given line's and column's algorithm. 0 means they are statistically non-similar, and 1.0 means they are statistically equal.

Table 8: Friedman tests

Experience	<i>stat</i>	<i>p</i>
Scientific paper Figure 1	199.348	6.559e-27
Figure 4	112.976	5.798e-07
Scientific paper Figure 2	144.651	2.773e-23
Scientific paper Figure 3	215.686	3.001e-33

Figure 9: Figure 4 Nemenyi test matrix.

Figure 8: Scientific paper Figure 1 Nemenyi test matrix.

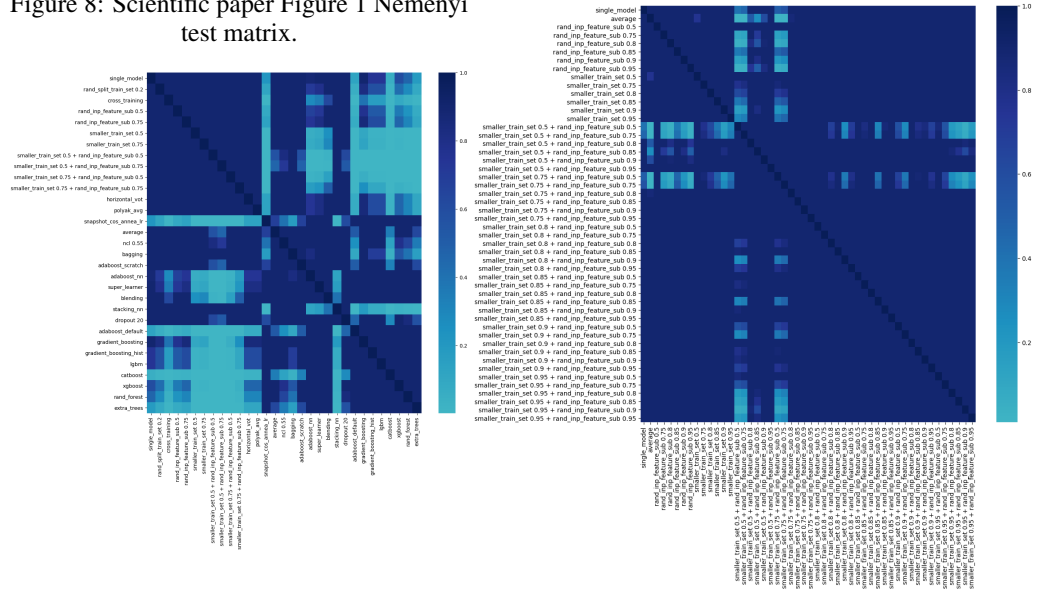


Figure 10: Scientific paper Figure 2 Nemenyi test matrix. Figure 11: Scientific paper Figure 3 Nemenyi test matrix.





## References

- [1] R. Bell, Y. Koren, and C. Volinsky. The BellKor solution to the Netflix Prize, 2007.
- [2] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- [3] L. Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36(1):85–103, 1999.
- [4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, oct 2001.
- [5] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD'16*, volume 13, pages 785–794. Association for Computing Machinery, aug 2016.
- [6] R. T. Clemen. Combining forecasts: A review and annotated bibliography. *International Journal of Forecasting*, 5(4):559–583, 1989.
- [7] A. V. Dorogush, V. Ershov, and A. Gulin. CatBoost: gradient boosting with categorical features support, oct 2018.
- [8] H. Drucker. Improving Regressors using Boosting Techniques. *ICML*, 1997.
- [9] H. Drucker and C. Cortes. Boosting Decision Trees. In *Advances in Neural Information Processing Systems*, pages 479–485, 1995.
- [10] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 904, pages 23–37. Springer Verlag, 1995.
- [11] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.
- [12] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, apr 2006.
- [13] T. K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.
- [14] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot Ensembles: Train 1, get M for free. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, mar 2017.
- [15] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*, volume 30, pages 3146–3154, 2017.
- [16] R. Kohavi and D. Wolpert. Bias Plus Variance Decomposition for Zero-One Loss Functions. In *ICML*, volume 96, pages 275–83, 1996.
- [17] A. Krogh and J. Vedelsby. Neural Network Ensembles, Cross Validation, and Active Learning. *Advances in Neural Information Processing Systems*, 7:231–238, 1995.
- [18] Y. Liu and X. Yao. Ensemble learning via negative correlation. *Neural Networks*, 12(10):1399–1404, dec 1999.
- [19] I. Loshchilov and F. Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, aug 2016.
- [20] G. Louppe and P. Geurts. Ensembles on random patches. In *Machine Learning and Knowledge Discovery in Databases*, volume 7523 LNAI, pages 346–361. Springer, Berlin, Heidelberg, 2012.

- [21] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent. *Advances in Neural Information Processing Systems*, pages 512–518, 1999.
- [22] S. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting Algorithms as Gradient Descent in Function. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, 1999.
- [23] P. Nemenyi. *Distribution-free Multiple Comparisons*. PhD thesis, Princeton University, USA, 1963.
- [24] B. Polyak and A. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, jul 1992.
- [25] B. E. Rosen. Ensemble Learning Using Decorrelated Neural Networks. *Connection Science*, 8(3-4):373–384, 1996.
- [26] scikit learn. Histogram-based Gradient Boosting Regression Tree., 2021. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingRegressor.html>, last accessed on 2021-05-23.
- [27] L. N. Smith. Cyclical Learning Rates for Training Neural Networks. *Proceedings - 2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017*, pages 464–472, jun 2015.
- [28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 06 2014.
- [29] N. Ueda and R. Nakano. Generalization error of ensemble estimators. In *IEEE International Conference on Neural Networks - Conference Proceedings*, volume 1, pages 90–95. IEEE, 1996.
- [30] M. J. Van Der Laan, E. C. Polley, and A. E. Hubbard. Super learner. *Statistical Applications in Genetics and Molecular Biology*, 6(1), sep 2007.
- [31] G. I. Webb. MultiBoosting: a technique for combining boosting and wagging. *Machine Learning*, 40(2):159–196, aug 2000.
- [32] G. I. Webb and Z. Zheng. Multistrategy ensemble learning: Reducing error by combining ensemble learning techniques. *IEEE Transactions on Knowledge and Data Engineering*, 16(8):980–991, aug 2004.
- [33] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.
- [34] J. Xie, B. Xu, and Z. Chuang. Horizontal and Vertical Ensemble with Deep Representation for Classification, jun 2013.
- [35] Y. Yu, Z. H. Zhou, and K. M. Ting. Cocktail ensemble for regression. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 721–726, 2007.

## Checklist

1. For all authors...
  - (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
  - (b) Did you describe the limitations of your work? [Yes]
  - (c) Did you discuss any potential negative societal impacts of your work? [Yes]
  - (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
  - (a) Did you state the full set of assumptions of all theoretical results? [Yes]
  - (b) Did you include complete proofs of all theoretical results? [Yes]

3. If you ran experiments...
  - (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [\[Yes\]](#)
  - (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [\[Yes\]](#)
  - (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [\[Yes\]](#)
  - (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [\[No\]](#)
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
  - (a) If your work uses existing assets, did you cite the creators? [\[Yes\]](#)
  - (b) Did you mention the license of the assets? [\[N/A\]](#)
  - (c) Did you include any new assets either in the supplemental material or as a URL? [\[N/A\]](#)
  - (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [\[N/A\]](#)
  - (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [\[N/A\]](#)
5. If you used crowdsourcing or conducted research with human subjects...
  - (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [\[N/A\]](#)
  - (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [\[N/A\]](#)
  - (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [\[N/A\]](#)

