




2023

Statistical Intervals for Neural Network and its Relationship with Generalized Linear Model

Sheng Yuan

University of Kentucky, shengyuan229@gmail.com

Author ORCID Identifier:

 <https://orcid.org/0009-0001-5545-2085>

Digital Object Identifier: <https://doi.org/10.13023/etd.2023.301>

[Right click to open a feedback form in a new tab to let us know how this document benefits you.](#)

Recommended Citation

Yuan, Sheng, "Statistical Intervals for Neural Network and its Relationship with Generalized Linear Model" (2023). *Theses and Dissertations--Statistics*. 65.
https://uknowledge.uky.edu/statistics_etds/65

This Doctoral Dissertation is brought to you for free and open access by the Statistics at UKnowledge. It has been accepted for inclusion in Theses and Dissertations--Statistics by an authorized administrator of UKnowledge. For more information, please contact UKnowledge@lsv.uky.edu.

STUDENT AGREEMENT:

I represent that my thesis or dissertation and abstract are my original work. Proper attribution has been given to all outside sources. I understand that I am solely responsible for obtaining any needed copyright permissions. I have obtained needed written permission statement(s) from the owner(s) of each third-party copyrighted matter to be included in my work, allowing electronic distribution (if such use is not permitted by the fair use doctrine) which will be submitted to UKnowledge as Additional File.

I hereby grant to The University of Kentucky and its agents the irrevocable, non-exclusive, and royalty-free license to archive and make accessible my work in whole or in part in all forms of media, now or hereafter known. I agree that the document mentioned above may be made available immediately for worldwide access unless an embargo applies.

I retain all other ownership rights to the copyright of my work. I also retain the right to use in future works (such as articles or books) all or part of my work. I understand that I am free to register the copyright to my work.

REVIEW, APPROVAL AND ACCEPTANCE

The document mentioned above has been reviewed and accepted by the student's advisor, on behalf of the advisory committee, and by the Director of Graduate Studies (DGS), on behalf of the program; we verify that this is the final, approved version of the student's thesis including all changes required by the advisory committee. The undersigned agree to abide by the statements above.

Sheng Yuan, Student

Dr. Arnold J. Stromberg, Major Professor

Dr. Katherine L. Thompson, Director of Graduate Studies

DISSERTATION

Sheng Yuan

The Graduate School
University of Kentucky
2023

Statistical Intervals for Neural Network and its Relationship with Generalized
Linear Model

DISSERTATION

A dissertation submitted in partial
fulfillment of the requirements for
the degree of Doctor of Philosophy
in the College of Arts and Sciences
at the University of Kentucky

By

Sheng Yuan

Lexington, Kentucky

ORCID: <https://orcid.org/my-orcid?orcid=0009-0001-5545-2085>

Director: Dr. Arnold Stromberg, Professor of Statistics

Codirector: Dr. Qiang(Shawn) Cheng, Associate Professor of Bioinformatics
Lexington, Kentucky 2023

ABSTRACT OF DISSERTATION

Statistical Intervals for Neural Network and its Relationship with Generalized Linear Model

Neural networks have experienced widespread adoption and have become integral in cutting-edge domains like computer vision, natural language processing, and various contemporary fields. However, addressing the statistical aspects of neural networks has been a persistent challenge, with limited satisfactory results. In my research, I focused on exploring statistical intervals applied to neural networks, specifically confidence intervals and tolerance intervals. I employed variance estimation methods, such as direct estimation and resampling, to assess neural networks and their performance under outlier scenarios. Remarkably, when outliers were present, the resampling method with infinitesimal jackknife estimation yielded confidence intervals that closely aligned with nominal levels. To consider neural networks as nonparametric regression models, I employed tolerance intervals and observed that the coverage of these intervals approached the nominal level. Additionally, I conducted a comparative study between neural networks and generalized linear models. The results indicated that neural networks did not outperform linear models in low-dimensional settings. However, in high-dimensional models or multitask classification, neural networks exhibited significantly superior performance. Lastly, I proposed further research exploring advanced techniques in neural networks, as well as investigating statistical attributes of various deep learning methods. These future studies hold the potential to expand our understanding of neural networks and enhance their statistical properties.

KEYWORDS: Neural Network, Statistical Interval, Variance Estimation, Misspecification

Author's signature: _____ Sheng Yuan

Date: _____ July 20, 2023

Statistical Intervals for Neural Network and its Relationship with Generalized
Linear Model

By
Sheng Yuan

Director of Dissertation: Dr. Arnold Stromberg

Codirector of Dissertation: Dr. Qiang(Shawn) Cheng

Director of Graduate Studies: Dr. Katherine Thompson

Date: July 20, 2023

Dedicated to Ms Wenhan Jiang, for her help in the period of my dissertation writing.

ACKNOWLEDGMENTS

Acknowledge Dr. Arnold Stromberg, Dr. Qiang Cheng, Dr. Derek Young for their advice on my PhD graduation.

TABLE OF CONTENTS

Acknowledgments	iii
Table of Contents	iv
List of Figures	vi
List of Tables	viii
Chapter 1 Introduction	1
1.1 Machine Learning & Statistical Models	1
1.2 Neural Network	9
1.3 Statistical Interval	15
Chapter 2 Literature Review	18
2.1 Variance Estimation in Neural Network	18
2.2 Comparison of NN versus Linear model	19
2.3 Tolerance interval's application	19
Chapter 3 Confidence Interval For Neural Network (Direct Evaluation)	21
3.1 Direct Variance Estimation	21
3.2 Experiment	22
3.3 Discussion	26
3.4 Conclusion	29
Chapter 4 Confidence Interval For Neural Network (Resampling)	30
4.1 Variance Estimation Method using Resampling	30
4.2 Confidence Interval Building	34
4.3 Experiment	36
4.4 Extended Research: Effect of Outlier	37
4.5 Conclusion	39
Chapter 5 Tolerance interval	41
5.1 Definition	41
5.2 Experiment	42
5.3 Discussion	46
Chapter 6 Neural Network's Effect Comparing With Linear and Logistic Regression with misspecification	49
6.1 Basic Concept	49
6.2 Experiment	51
6.3 Conclusion	62

Chapter 7	Future Development	67
7.1	Problems in Neural Network	67
7.2	Advanced Neural Network	71
7.3	Advanced Variance Estimation Method	73
7.4	Neural Network and Logistic Regression	75
Chapter 8	Conclusion	76
Chapter 9	Appendix	79
9.1	Proof of unbiased estimator of jackknife	79
9.2	Proof of Jackknife as Approximation to Bootstrap	80
9.3	Proof of coefficient of variation in standard error estimation	81
9.4	Iteratively Reweighted Least Squares in Logistic Regression	81
Bibliography	82
Vita	105

LIST OF FIGURES

1.1	Perceptron	9
1.2	Deep Neural Network	10
3.1	True data, true parametric line and fitted line	23
3.2	True data, true parametric line and fitted line(zoom in)	23
3.3	True data, true parametric line and fitted line (x axis represent explanatory variable, blue dot represent true data with noise in simulation, and dashed line represent 95% confidence band)	24
3.4	sliding neighbourhood size from 0 to 1 and see coverage rate for 95% confidence interval in standard error	24
3.5	True data, true parametric line and fitted line with confidence band of using 95% confidence band make use of MAD approximation	25
3.6	sliding neighbourhood size from 0 to 1 and see coverage rate for 95% confidence interval in MAD	26
3.7	From left to right, we have sample size n change from 100, 500 to 1000 .	26
3.8	From left to right, we have the fitting of neural network's performance, 95% CI with 97.4% coverage percentage with sliding window size 0.05, and the growth of sliding window size's effect on coverage rate.	28
3.9	From left to right, we have the fitting of neural network's performance, 95% CI with 99.6% coverage percentage with sliding window size 0.05, and the growth of sliding window size's effect on coverage rate.	28
4.1	CV vs B when sample size fixed to 500	36
4.2	95% Confidence Interval using Infinitesimal Jackknife Variance Estimation	37
4.3	True data, true parametric line and fitted line(delete outlier)	38
4.4	Huber's M Estimator for k=1	38
4.5	True data, true parametric line and fitted line(delete outlier)	39
5.1	The coverage probability at n=500 sample size of unique x value for 90×95 , 95×95 and 99×95 tolerance intervals.	43
5.2	Fitting linear curve with neural network.	44
5.3	The coverage probability at n=500 sample size of unique x value for 90×95 , 95×95 and 99×95 tolerance intervals, fitting linear model with neural network.	45
5.4	Left figure describe for fixed confidence limit α , the behavior of K versus coverage percent P; Right figure talks about for fixed coverage percent P, the behavior of K versus confidence limit α	46
5.5	The coverage probability at n=500 sample size 95×95 tolerance intervals, fitting linear model with MAD as variance estimation on neural network.	47
5.6	The coverage probability at n=500 sample size of unique x value for 90×95 , 95×95 and 99×95 tolerance intervals, fitting nonlinear model with neural network.	47

6.1	Sigmoid Function	55
6.2	ROC curve example	56
6.3	Model Significance	59
6.4	Logistic Regression Confusion Matrix	61
6.5	Neural Network Confusion Matrix	61
7.1	Neuron Structure CNN	72
7.2	Neuron Structure RNN	72

LIST OF TABLES

4.1	Compare coverage rate of multiple smoothing method with 95% CI . . .	39
5.1	Compare coverage rate of multiple smoothing method with 95% CI . . .	43
6.1	paper performance	63
6.2	paper performance cont.	64
6.3	paper performance cont.	65
6.4	paper performance cont.	66

Chapter 1 Introduction

Over the years, data analysis methods have evolved and now, researchers tend to categorize them based on their objectives. Traditional statistical analysis focuses on parameter estimation and interpretation, aiming to provide more mathematically or statistically explainable results rather than just outputting a result. On the other hand, machine learning methods aim to predict outcomes on new data based on training data, with metrics such as accuracy and precision receiving more attention. Researchers in this field tend to focus on improving the speed and performance of models, rather than delving into the methods used to discover probability insights behind the model.

In my dissertation thesis, I plan to bridge the gap between these two methods by introducing statistical insights into machine learning models. I aim to investigate the reasons behind the impressive performance of machine learning models and find ways to improve them using mathematical methods.

During the early days of machine learning, computation resources were limited, and the learning algorithms were not advanced enough to provide predictions for large output layers. However, since 2010, the development of deep neural network algorithms, as well as parallel computing, has enabled the use of multiple GPUs instead of CPUs, resulting in faster computations. The training of deep neural networks with numerous hidden layers and large output outcomes has become considerably faster, leading to the expansion and deepening of the field of machine learning.

As a result of these advancements, we now have traditional machine learning methods, such as neural networks, which are widely used in big data analysis and highly complex machine learning tasks.

1.1 Machine Learning & Statistical Models

The original definition of statistics involves using a sample to make inferences about the behavior of a population. In this context, machine learning models can be considered a specific case of statistical models. The distinction between machine learning and statistical models is based on whether statistical inference is used in parameter estimation and model fitting.

To further explore this distinction, we will introduce several models, including generalized linear models, ensemble models, and neural networks, and discuss their relationships.

1.1.1 Generalized Linear Model

The generalized linear model is a fundamental structure of statistical models. In this discussion, we will introduce the basic structure of the model, its underlying assumptions, and its potential limitations.

Model Structure

Statistics, by definition, involves using a sample to make inferences about the behavior of a larger population. Machine learning models can be seen as a subset of statistical models, where the focus is primarily on making predictions based on data. One key difference between machine learning and statistical models lies in their approach to parameter estimation and model fitting. While statistical models typically use statistical inference methods to estimate model parameters, machine learning models often rely on optimization algorithms to minimize a loss function and fit the model to the data.

To delve deeper into this distinction, we will discuss several models, including generalized linear models, ensemble models, and neural networks, and explore their relationships.

The term "generalized" in generalized linear models comes from the fact that they are part of a larger class of models introduced in [1]. In these models, the response variables $\{y_i\}$ are assumed to follow an exponential family distribution with mean value $\{\mu_i\}$, and to be a function of $X\beta$, which may be nonlinear. To construct a generalized linear model, three fundamental components are used:

- Randomized components: refer to the assumption that the response variable follows a specific distribution, such as normal, binomial, or Poisson. It's important to note that there is no requirement for an independent error term in this case.
- The linear component of a generalized linear model consists of p variables in the model, denoted by x_1, \dots, x_p , which represent p known characteristics of each observation. The prediction for the response variable is modeled as a linear combination of these variables, in the form of $\beta_0 + \beta_1 x_1 + \dots + \beta_p x_p$.
- The link function is a crucial component of a generalized linear model, as it specifies the relationship between the randomized outcome and the linear component. It is represented by a function $g(\cdot)$, which relates the mean response variable Y to the linear predictor $X\beta$, such that $g(Y) = X\beta$.

To specify the exponential family[2] in the context of generalized linear models, we refer to a group of parameterized probability distributions. The density function of a random variable x can be factorized into a specific form, given by:

$$f_X(x|\theta) = h(x)g(\theta)\exp(\eta(\theta) \cdot T(x)) \tag{1.1}$$

Overall, the three fundamental elements of a generalized linear model are the randomized distribution of the output variable, the linear form, and the link function.

When it comes to fitting the model, there are different loss functions for generalized linear models that need to be satisfied. For regression problems, the sum of squared errors is commonly used, while for classification tasks, cross-entropy loss is often preferred. To estimate the model's parameters, maximum likelihood estimation or Bayesian methods can be employed.

Model Assumptions

These assumptions below are assumptions for generalized linear models. We have some specific criteria for more specified linear models.

- The observations Y_1, \dots, Y_n should be independent of each other.
- The response variable Y_i is typically assumed to follow a distribution from the exponential family (e.g., binomial, Poisson, multinomial, normal, etc.).
- While a linear relationship between the response variable and explanatory variables is not assumed in GLMs, a linear relationship is expected between the transformed expected response (using the link function) and the explanatory variables, i.e., $g(Y) = X\beta$.
- The explanatory variables can include elementary transformations of the original variables, such as linear, power, exponential, logarithmic, trigonometric, inverse trigonometric, and hyperbolic functions.
- Homogeneity of variance is not required in GLMs, unlike in linear regression where it is assumed.
- The errors must be independently distributed.
- Maximum likelihood estimation (MLE) is used to estimate the model parameters in GLMs instead of ordinary least squares (OLS), mainly because there is no explicit solution for the parameters using OLS as the loss function.

Potential Weakness

There are multiple disadvantages for generalized linear regressions, which has been shown below:

- Feature selection can be challenging in generalized linear regression, and regularization methods are often necessary to improve model performance.
- GLMs assume that the outcomes follow a specific type of distribution, which can limit their applicability in some cases.

- There should be no correlation among the predictors, which can be difficult to achieve in practice.
- GLMs can be sensitive to outliers, and regularization methods can help mitigate their impact on the model.
- GLMs may have lower predictive power compared to some machine learning models, particularly for complex and high-dimensional data.

When fitting GLMs using the maximum likelihood estimation (MLE) method, we need to use Newton's method to iteratively update the parameters. This method is sensitive to the initial value and may involve the Hessian matrix or score matrix to update parameters, which can sometimes result in failure to achieve the global minimum or convergence. When using Bayesian methods, choosing the prior distribution to approximate the posterior can be challenging, and it may not be possible to achieve an analytical solution.

Neural networks are closely related to generalized linear models, although they share some of the same disadvantages. However, neural networks typically have stronger predictive power and fewer restrictions on the assumptions compared to GLMs. This dissertation mainly discusses several statistical methods for analyzing neural networks. Before delving deeper into neural networks, it is important to introduce the basic structure of machine learning models and statistical methods used to analyze them.

1.1.2 Ensemble Model

This section discusses the use of ensemble methods to combine multiple base estimators in order to improve model performance over using a single algorithm.[3, 4, 5]

The goal of a statistical model is to infer the behavior of the response variable y from a dataset of predictor variables \mathbf{x}_i , where i ranges from 1 to n . One approach to modeling is to use a set of basic learners denoted by L , and a function $\phi(x, L)$ to make predictions of y from x . This function $\phi(x, L)$ represents the ensemble method. One example of an ensemble method is as follows:

Suppose we have a dataset y, x_1, \dots, x_n and we want to understand the relationship between y and the x 's. We apply a resampling method (such as bootstrap) to both y and x k times, creating k sub-datasets. For each sub-dataset, we fit a basic learner (such as a linear regression model), resulting in k different models.

Then, we use the k models to make predictions for a special case of x_{11}, \dots, x_{n1} , resulting in k different \hat{y} values. Because each model was trained on a different sub-dataset, the coefficients for the same variable in the k models may be different. Finally, we aggregate the k \hat{y} values using a method such as averaging to obtain our

final prediction for the specific case x_{11}, \dots, x_{n1} . This entire process is called ensemble.

Ensemble learning is a technique that involves combining the predictions of multiple models to improve overall performance. The key step in ensemble learning is to derive a new sample from the existing sample (usually done through resampling) and then aggregate the predictions of the base models. This approach is often used to compensate for poor performance of individual models and can be more effective than relying on a single method alone.

One commonly used base learner in ensemble learning is the decision tree[6], which is a fast algorithm that can be used to generate predictions quickly. There are various types of ensemble methods, but one common approach is called bagging, which involves resampling the training data to create multiple subsets and then training a base learner on each subset. The predictions of the individual learners are then aggregated to produce a final prediction.

Another popular ensemble method is the random forest, which is a type of bagged learner that uses decision trees as the base models. Random forest models differ from standard decision trees by introducing randomness in the feature selection and resampling processes, which helps to reduce overfitting and improve the generalization performance of the model.

In this context, we can compare the behavior of our ensemble neural network with that of a random forest model.

Decision Tree

The decision tree[6] is one of the base learners used in the random forest ensemble method. Decision trees are a type of machine learning method that can be applied in various fields, such as pattern recognition and image processing[7]. The development of decision trees has been summarized in recent research [8].

A decision tree is a tree-based technique that uses a series of basic tests to efficiently separate a dataset into categories. Each test is conducted by comparing a numeric value with a fixed threshold[9], and the outcome at the leaf is a Boolean value[10, 11, 12, 13]. Decision trees are mainly used for grouping purposes, but they can also be used as a classification model in data mining. The nodes and branches of the tree represent features and their associated subsets, respectively.[14, 15]

In machine learning, the goal is to achieve the best value for the loss function[16]. The loss function most commonly used for each node in the decision tree is entropy[17]. Entropy is a value between 0 and 1, with values closer to 0 indicating better classification. The entropy of classifying a set S [18, 19] into different c classes is shown

below:

$$H(S) = \sum_{i=1}^c p_i \log_2 p_i \quad (1.2)$$

p_i represent the proportion of the classified sample to the true subset.

The concept of information gain, also referred to as mutual information, is commonly used in decision trees to measure the amount of information that a random variable provides about the target variable.[20, 21] Information gain, denoted as $IG(S,a)$, is calculated based on the definition of entropy[22, 23] and evaluates the impact of splitting a dataset S using a random variable a . The formula for information gain is shown below:

$$IG(S, a) = H(S) - H(S|a) \quad (1.3)$$

$H(S|a)$ is the conditional entropy of dataset S given random variable a , and defined as follow:

$$H(S|a) = \sum_{v \in V(a)} \frac{|S_v|}{|V|} H(S_v) \quad (1.4)$$

The range attribute of a is $V(a)$, S_v is a subset of S equal to the attribute v 's size[21].

Decision trees have several advantages, including their ease of interpretation, transferability, and lack of need for prior assumptions. However, decision trees can suffer from issues such as incorrect decision-making and overfitting when there are too many features included in the model. Additionally, the computation time may be slow due to the complexity of the tree. Another limitation is that binary splitting of attributes in each sub-node can make regression predictions difficult to follow.[24, 25, 26].

Bagged Learners

Ensemble methods that combine multiple learning algorithms can typically achieve better predictive performance compared to using a single base learner.[3, 4, 27] There are several ways to ensemble base learners, including:

- Bayes optimal classifier[28]: This method combines all hypotheses in the prior parameter space to obtain the best probability for classification.
- Boosting[29]: This method iteratively assigns weights to weak learners that were misclassified to reduce bias and variance in each iteration.
- Bayesian Model Averaging (BMA)[30]: BMA uses the average with weights from several models, given the posterior probability for each model given the data.
- Bayesian Model Combination[31]: An update to BMA, this method samples weights from all possible ensembles instead of sampling each model individually.

- Bucket of models[32]: This method uses a selection algorithm with a train-test split to choose the best model to average.
- Stacking[33]: This method trains an algorithm to combine predictions from several other learning algorithms.

Bagging, or bootstrapping, is a successful method for improving uncertainty or classification accuracy. It is particularly useful in large, high-dimensional datasets where using a simple learner can make it difficult to find the optimal way to fit the model.

If we define the dataset as $D_i = (Y_i, X_i)$, where Y_i represents the response variable and X_i represents a p -dimensional explanatory variable for the i -th instance, we can use the function $\hat{\theta}(x) = h(D_1, \dots, D_n)(x)$ to estimate $E(Y|X = x)$ for a new explanatory random variable x . Typically, decision trees such as CART[34] or MARS[35] can be used as the function $h(x)$.

Definition 1.1.1 (Bagging). Bagging is defined theoretically as follow:

1. Get a bootstrap sample $D_i^* = (Y_i^*, X_i^*)$ from the original sample space D_i .
2. Compute bootstrapped predictor $\hat{\theta}^*(x) = h(D_1^*, \dots, D_n^*)(x)$, note that we may use different $h(x)$ for estimation on each bootstrap sample.
3. The bagged predictor is $\hat{\theta}_B(x) = E^*[\hat{\theta}^*(x)]$

The expectation in 3. above most of the time will be implemented by Monte Carlo simulation:

$$\hat{\theta}_B(x) \approx B^{-1} \sum_{b=1}^B \hat{\theta}_{(b)}(x) \quad (1.5)$$

Referring to [36], the bootstrap size B is often to be chosen at range of 50.

There have been numerous theoretical studies conducted on bagging. One such study [36, 37] suggests that bagging can improve the mean square error through a bias-variance trade-off, particularly when the base learners $h(x)$ are unstable. Other studies have also investigated bagging, such as [38], which decomposed smooth estimators into linear and higher-order terms, and [39], which used U-statistics to determine the leading effects of variance, bias, and MSE on bagging, even highlighting the potential for increased second-order MSE terms[40]. To reduce the computational costs associated with bootstrap, [41] analyzed the variance and MSE reduction of subbagging, and proposed techniques such as subbagging and half subbagging, which are computationally efficient and provide similar accuracy to bagging.

In addition to random forest[42], bagging has been extensively applied to many other machine learning algorithms. SVM[43], multilayer perceptron[44], regularization[45],

neural networks[46], stack denoising autoencoders[47], and even survival trees[48] have all been subject to bagging to showcase its effectiveness on these types of machine learning and statistical learning algorithms. For instance, to address unbalanced cases, an asymmetric bagging algorithm[49] has been developed, as well as Roughly Balanced Bagging[50] and Neighbourhood Balanced Bagging[51] algorithm to demonstrate the classification effectiveness of bagging. Furthermore, mini-batch bagging algorithms combining bagging and boosting[52] have been proposed to speed up computation.

Random Forest

Random forest is one of the most widely used machine learning methods in business and big data analysis. It has been applied in various fields, including chemoinformatics[53], ecology[54, 55], 3D object recognition[56], bioinformatics[57], and econometrics[58]. In fact, Howard and Bowles (2012) suggest that *"random forests" have been the most successful general-purpose algorithm in modern times.*

The term "random forest"[42] usually refers to a collection of B randomized decision trees. As defined in 1.1.1, if we take $\hat{\theta}^*(x)$ to be a decision tree and apply the same Monte Carlo method as described in 1.5, we can build the basic structure of a random forest.

We notice that there are 3 parameters relate to random forest are important:

1. $a_n \in 1, 2, \dots, n$ is the number of sampled data points in each tree.
2. $m_{try} \in 1, 2, \dots, p$ is the number of possible features in each tree each node.
3. $nodesize \in 1, 2, \dots, a_n$ is if the each sample in each cell below that the cell will not split.

Randomness is a crucial element of random forest, which is introduced in two ways. First, we bootstrap a sample of size less than or equal to N to subset samples for each tree. Second, we subset a small sample of coordinates from the entire range of p . Additionally, each individual tree is not pruned, further adding to the randomness of the model.

A considerable amount of literature([57, 59, 60]) has analyzed parameter tuning in random forests, including B , m_{try} , and a_n . Some theory and extensions have also been developed to explain Breiman's forest[42]. For instance, [61, 62, 63] proposed the idea of choosing $a_n < n$ examples without replacement from the initial sample. [64] introduced the median forest, which is an ensemble of decision trees with each individual tree split point being in the median when choosing a coordinate at each node, using the subsample method instead of bootstrap. [65, 66] explored the bagging principle and application of nearest neighbors. Additionally, discussions on how to split decision trees, consistency, asymptotic properties, and variable importance are

also ongoing.[67]

There are several extensions of random forests, such as weighted forest[68, 69], online forest[70, 71, 72, 73, 74], survival trees[75, 76, 77], ranking forest[78], clustering forest[79], quantile forest[80], etc. Similar methods for variance estimation for random forests have also been developed for neural networks.

1.2 Neural Network

Now we talk about the idea in the definition of neural networks.

Basic Structure

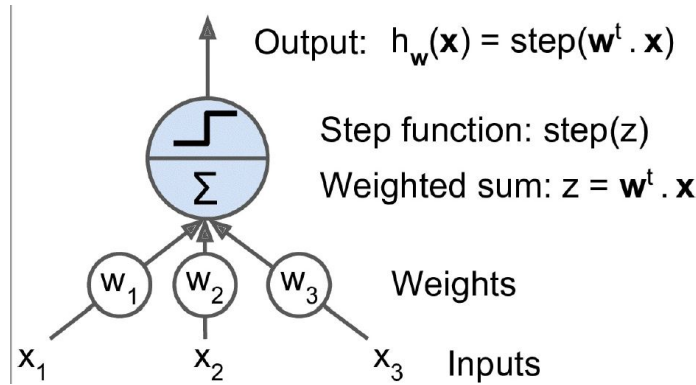


Figure 1.1: Perceptron

To begin, we'll explore the perceptron, which is the simplest type of neural network. As depicted in Figure(1.2), the perceptron works similarly to generalized linear regression. The inputs, denoted by x_i , represent the different features in the dataset, while the corresponding weights, w_i , reflect their importance. Combining these weights with the inputs in a linear form ($z = \sum_i w_i x_i$) yields a linear predictor. After applying a step function (or link function in the case of GLM), the perceptron outputs a value of $Y = \sigma(XW')$, where $X = x_1, \dots, x_p$ and $W = w_1, \dots, w_p$. The perceptron serves as the basic building block for neural networks.

Moving on to more complex neural networks, we have the deep neural network, which takes the form of a multi-layer perceptron (as illustrated in Figure(1.2)).

Similar to linear regression, we construct an input matrix for each input as $X = [1|x_1 \dots x_p]$, where the column vector of all ones represents the intercept term. In this setup, our input layer has $p+1$ features or neurons. By applying a linear transform using a weight matrix and activation function, we can obtain one of the neurons in the next hidden layer, which is of dimension $n \times 1$. By using different weights with the same input matrix X , we can obtain different neurons in the next hidden layer (where

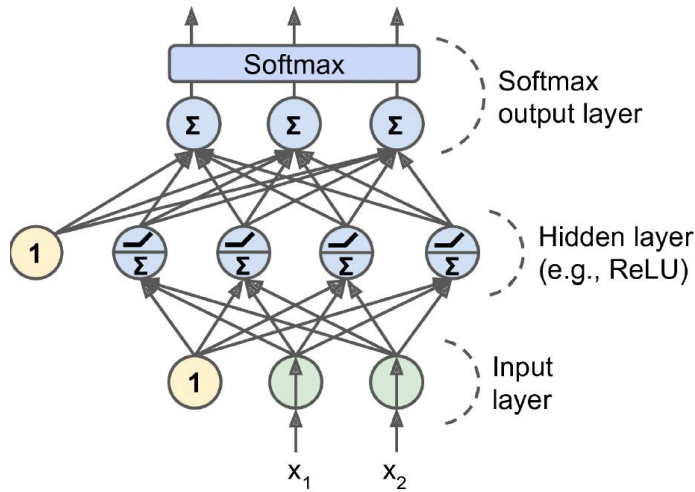


Figure 1.2: Deep Neural Network

L1 represents the number of neurons in the first hidden layer), and each neuron has a size of $n \times 1$. We can then consider the neurons in the first hidden layer as inputs and use the same linear transform with activation function to obtain neurons in the next layer, which leads to the final output layer through iteration. To achieve different goals, we can apply different activation functions and output dimensions.

Adam Optimizer

When we construct neural network model into parametric version, which is

$$\hat{Y} = \sigma(\sigma \dots (\sigma(XW'_0 + b_0)W'_1 + b_1)W'_L + b_L) \quad (1.6)$$

The total number of times the activation function is applied is equal to the total number of layers (L) plus one. In each layer, W'_j is an $L_j \times L_k$ matrix where $j = k + 1$, and the dimension of W'_0 is $p \times L_1$. Additionally, b_i in each layer is an $n \times L_k$ matrix.

In the case of the parameteric model described above (for simplicity, we denote $\hat{Y} = h_W(X)$, where h represents a complex nonlinear function), we typically optimize a loss function to find the parameter set. If the loss function $L(w)$ is well-defined and the error term has an explicit distribution, we may directly obtain a maximum likelihood estimator by setting the first derivative of the log likelihood to be 0 and the second derivative to be greater than 0.

The structure of a neural network is complex, and it can be challenging to find the global minimum of the loss function by directly assigning values to the model parameters. As a solution, we introduce gradient descent as our optimization method.

Gradient descent, also known as steepest descent, was first proposed by Cauchy in 1847 and studied by [81] in 1907. It utilizes gradient information to find the direction

of descent and updates the parameters using the descent information and a learning rate.

Definition 1.2.1. If $d \in \mathbf{R}^n$ is a vector, and $f : \mathbf{R}^n \rightarrow \mathbf{R}$, the smooth direction is defined as d if $\nabla f(x)d < 0$.

If d is the descent direction, then the value of f will decrease as we move along d from x . The gradient $\nabla f(x)$ offers the direction of steepest ascent of the function at a single point x , while $-\nabla f(x)$ represents the direction of steepest descent from point x . Therefore, on each update for our data point $x_i \in x$, we update our parameters using the formula $x_i - \eta \frac{\partial f(x_i)}{\partial x_i}$, where η is the learning rate.

To summarize, the gradient descent method can be described as follows:

1. Initialize the weights $W = W_1, \dots, W_L$ in the neural network, and assign 0 as the initial value of the intercept b . There are several methods for weight initialization, such as constant initialization, random normal initialization, He initialization[1], and Xavier initialization[2]. Here, we will apply Xavier initialization, which draws a random variable from a uniform distribution with $[-1/L_i, 1/L_i]$ in each layer i .
2. For each $w_{ij} \in W_i$, set $d_j = -\frac{\partial L(w_{ij})}{\partial w_{ij}}$.
3. If $\|d_j\| \leq \epsilon$, stop; otherwise, continue.
4. Find an approximation to the problem: minimize $L(w_{ij} + \eta d_j)$. The learning rate η is set to be 0.01, but there are several methods for setting η as a function and tuning it each iteration.
5. Set $w_{ij, new} = w_{ij} + \eta d_j$ and go back to step 2.

While there are several advanced optimization methods like Stochastic Gradient Descent[82], Mini-Batch Gradient Descent[83], and Adagrad[84], we will introduce the Adam (Adaptive Moment Estimation) method[85]. In Adam, we insert some updates for steps 4 and 5. Instead of directly using the gradient in step 2, we introduce two raw moment estimates m_j and v_j by applying a weighted combination of the previous moment m_{j-1} and gradient information d_j . This allows us to achieve bias-corrected moment estimates for updating the parameters using the learning rate η multiplied by $m_j/(\sqrt{v_j} + \epsilon)$.

The computation of the Adam method is pretty straightforward and easy to apply. It effectively avoids the problem of gradient vanishing. In fact, as suggested by Ruder in [86], "Adam might be the best overall choice.

Autodiff

Given that our loss function related to weight is complex and involves many variables, we can simplify the process of calculating its derivatives using a technique

called automatic differentiation, or autodiff for short. This method is based on the concept that most computer programs, no matter how intricate, can be expressed as a composition of finitely many elementary functions, such as exponentials, logarithms, powers, and inverses. By repeatedly applying the chain rule to these operations, we can obtain the derivatives in a specific direction with much less computational and time complexity.

There are two traditional methods for autodiff: forward accumulation and reverse accumulation. In the context of neural networks, reverse mode automatic differentiation is commonly used to calculate gradients with respect to the initialized weights. Here, we will discuss reverse mode autodiff in more detail.

To start, suppose we have a target function G that depends on n variables x_1, \dots, x_n , and a set of elementary arithmetic operations and functions that were applied to these variables. We can decompose G into N smaller parts, each of which involves a subset of the variables and elementary operations. Let $x_i = g_i(x_{\text{par}(i)})$ for $i = n + 1, \dots, N$, where $\text{par}(i)$ represents the parent of variable i .

Next, we can calculate the derivatives of G with respect to each of the x_i 's using the chain rule. We begin by defining the derivative of G with respect to the last variable x_N as:

$$\frac{df}{dx_N} = 1 \tag{1.7}$$

Then, we can calculate the derivatives with respect to each of the variables x_i in reverse order ($i = N - 1, N - 2, \dots, 1$) using the following formula:

$$\frac{df}{dx_i} = \sum_{j:i \in \text{par}(j)} \frac{df}{dx_j} \frac{dx_j}{dx_i} \tag{1.8}$$

Here, each x_j can be expressed as a composition of elementary functions or elementary functions of multiple x_i 's. Finally, we can update each of the parameters using gradient descent, since we have calculated the derivatives with respect to each of the variables.

Let's explore the basic structure of a neural network as an example. Suppose we have an input $X = \{x_1 | x_2 | \dots | x_p\}$, where $v_{1, \dots, p} = x_1, \dots, x_p$ denote the input vertices and hidden layer vertices. Suppose we have a total of P neurons in the hidden and output layers, where $i = p + 1, p + 2, \dots, P$. Then, we can express each vertex v_i as $v_i = \sigma_i(w_i \cdot v_{\text{par}(i)})$, where σ_i is an activation function and w_i is the weight parameter associated with vertex v_i . Here, $\text{par}(i)$ denotes the parent vertices of v_i .

To calculate the gradient of the target function with respect to each v_j where j is in the set of $\text{par}(i)$, we can use the chain rule and obtain the expression:

$$\sigma'_i(w_i \cdot v_{par(i)})w_{i,j}$$

where $\sigma'_i(x)$ is the derivative of the activation function with respect to its input, evaluated at $w_i \cdot v_{par(i)}$. If we define $v'_i = \sigma'_i(w_i \cdot v_{par(i)})$, then the update of the weight in the gradient is given by:

$$\frac{df}{dw_i} = \frac{df}{dv_i} v'_i w_i$$

In particular, if we use the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, we have $\sigma'(x) = \sigma(x)(1 - \sigma(x))$, which is still in the form of a sigmoid function. This makes the application of reverse auto-differentiation in backpropagation easier to conduct.

When training a neural network using backpropagation, we can use a technique called "early stopping". This involves monitoring the validation loss of the model during training, and stopping the training process when the validation loss starts to increase. This is done in order to prevent overfitting, which occurs when the model is too complex and starts to memorize the training data instead of learning general patterns.

Specifically, we divide our dataset into training and validation sets, and monitor the validation loss during training. If the validation loss stops decreasing and starts to increase, we stop the training process and select the weights that gave the lowest validation loss as our final weights. This is done to ensure that our model generalizes well to new, unseen data.

It's worth noting that sometimes the validation loss can fluctuate due to noise in the data or other factors, which is why we introduce a stopping criterion that waits for t consecutive increases in validation loss before stopping the training process. This helps ensure that we're not stopping prematurely due to random fluctuations in the validation loss.

Fine Tuning Parameters

Neural networks are often considered black-box models, and as such, tuning hyperparameters is a critical part of the model fitting process. There are many parameters that can be adjusted, including the number of layers, the number of neurons per layer, the activation function used in each layer, the weight initialization method, and more.

To develop an effective neural network model, it is common practice to split the available data into three parts: one for training, one for validation, and one for testing. Typically, 70% of the data is used for training, 20% for validation, and 10% for testing. During the training phase, the model is fitted using different parameter sets, and the validation data is used to assess how well the model is performing based on a

specific metric. Finally, the test data is used to evaluate the performance of the fully trained model. It's important to note that the allocation of data into these three groups should be done randomly to avoid bias.

There are two common techniques used for searching over parameter sets in neural network models: grid search and random search.

Grid search involves exhaustively enumerating all possible combinations of parameter values within a defined range, and evaluating the performance of the model on a validation set or through k-fold cross-validation on a combined training-validation dataset. Once the best set of parameters is identified, it is used to fit the test set. While grid search guarantees that all possible parameter combinations are explored, it suffers from the curse of dimensionality and can be difficult to parallelize since each setting is independent of the others.

On the other hand, random search selects a random combination of parameter values from a defined range, and evaluates the performance of the model. This process is repeated for a set number of iterations or until a satisfactory level of performance is achieved. Compared to grid search, random search can be more efficient in exploring high-dimensional parameter spaces, but it may not guarantee that the best set of parameters is found.

Both techniques have their advantages and disadvantages, and the choice between them will depend on the specific problem at hand and available computational resources.

To reduce the computational cost of hyperparameter tuning with grid search, one strategy is to sample a random value for each parameter at every iteration, rather than exhaustively testing all possible combinations. It is important to note that each iteration should generate a unique set of parameters. In practice, using a small number of hyperparameters in the final tuning can often lead to better performance compared to grid search.

Hyperparameter tuning can be performed using various advanced methods, such as Bayesian optimization[87], gradient-based optimization[88], evolutionary algorithms[89], early stopping[90, 91, 92], and spectral approaches[93].

Since our research focuses specifically on studying statistical intervals for neural networks, we have limited our hyperparameter tuning to adjusting only the number of neurons in each layer. Therefore, we have opted to use a grid search approach with a range of $[2,30] \times [2,30]$. This means that the number of neurons in the first layer is varied from 2 to 30, as well as the number of neurons in the second layer. While more advanced methods can be useful for more complex hyperparameter tuning tasks, the grid search approach is sufficient for our current research goals.

1.2.1 Relationship of NN and Generalized Linear Model

We can express a generalized linear model using the following formula:

$$Y = g(X\beta) \tag{1.9}$$

In this formula, Y represents the target variable, X represents the independent variables, and β represents the coefficients for each independent variable. The function g is the activation function (or link function), which is chosen based on the distribution of the target variable.

Linear regression and logistic regression are two commonly used methods for fitting generalized linear models. In linear regression, we assume that the target variable has a normal distribution, and the activation function g is the identity function. In logistic regression, we assume that the target variable follows a Bernoulli distribution, and the activation function g is the logit function.

If we compare formula (1.6) for a neural network with formula (1.9) for a generalized linear model, we can observe that when the neural network has only one layer and one neuron in the output layer, it reduces to a generalized linear model.

To specify our neural network within the recursive Canonical GLMs framework, we define the following properties:

1. We can specify multiple GLMs in parallel with each other instead of specifying a single GLM. We refer to these parallel groupings of GLMs as "layers".
2. By stacking multiple layers and using the output of the previous layer as input for the next layer, we construct the neural network.

1.3 Statistical Interval

Because samples actually comes from a population or process, the statistical intervals explains uncertainty exist in the data. We here introduce two types of statistical intervals commonly used in statistical inference.

1.3.1 Constructive Method

As samples are drawn from a population or process, there is inherent uncertainty in the data. Statistical intervals are used to quantify this uncertainty. In statistical inference, there are two commonly used types of statistical intervals that we will introduce here.

There are several traditional methods that involve multiple stages in constructing statistical intervals. The first method is the Delta method, which involves using a nonlinear regression model to obtain the prediction interval. The second method is

the Bayesian method, which uses Bayes' theorem to optimize the weights. The Mean-Variance Estimation method (MVEM) is another traditional technique that has been used to construct statistical intervals. Like other traditional methods, MVEM assumes normally distributed errors around the average of the target, and statistical intervals can be easily formed if the mean and variance are known. The fourth method is the Bootstrap method, which applies resampling to the data to construct a reliable statistical interval. These methods will be further discussed in the literature review.

There are multiple papers talks about statistical intervals directly from neural networks, which are specifically used in neural network model. These methods will be illustrated in literature review. Here we talks not only about interval construction method, but also intervals used in real-world uncertainty estimation of predictions.

1.3.2 Confidence Interval

A confidence interval is a statistical range of plausible values for an unknown parameter based on a sample of data. It is calculated at a specified level of confidence, denoted by α , which represents the probability that the interval will contain the true population parameter if the sampling were repeated many times.

To construct a confidence interval for a parameter θ based on a random sample X from a probability distribution P , we seek a range of values $[L, U]$ such that the probability of θ being in the range is at least $(1 - \alpha)$. That is,

$$P(L < \theta < U) \geq (1 - \alpha) \text{ (for every } \theta) \quad (1.10)$$

There are different methods for constructing confidence intervals, but two common approaches are the empirical quantile method and the use of normal cutoffs with an estimated variance.

1.3.3 Tolerance Interval

A tolerance interval is a statistical range that is commonly used in manufacturing and engineering to ensure that a specified proportion (P) of a population is covered with a certain level of confidence (α). The endpoints of a tolerance interval are known as tolerance limits.

The main difference between a confidence interval and a tolerance interval is that the former is used to detect limits on a given population parameter, while the latter is used to identify the limits within which we expect a stated proportion of the population to lie. To put it simply, the population parameter in a tolerance interval is the P percent of the population distribution.

To understand the definition of a $[100 \times (1 - \alpha)\%]/[100 \times P\%]$ tolerance interval, we can define the coverage of an interval $[L, U]$ as:

$$C(L, U) = F(U) - F(L) \tag{1.11}$$

where F represents the cumulative distribution function of the probability allocated to the population. The interpretation of a $[100 \times (1 - \alpha)\%]/[100 \times P\%]$ tolerance interval is that, within a $[100 \times (1 - \alpha)\%]$ confidence level, at least $[100 \times P\%]$ of the population will fall between L and U . To obtain L and U , we need to have:

$$\tau = Pr[C(L, U) \geq P] \geq 1 - \alpha \tag{1.12}$$

where τ represents the probability that the interval $[L, U]$ will cover at least P proportion of the population.

I have done multiple experiments on statistical building and statistical attributes post-hoc analyzed from neural network, these codes has been saved in the following path: https://github.com/VPdota2/Dissertation_SY. All codes related to direct estimate with neighborhood method is saved in under the document *CI_using_sliding_window.ipynb*, all codes related to resampling with infinitesimal jackknife is saved at *Infinitesimal_Confidence_Intervalxxx.ipynb*, all tolerance interval is saved at *Tolerance Interval Neural Network.ipynb* and all model related to GLM is saved at *Misspecify xxxxx.ipynb*.

Chapter 2 Literature Review

Here we review literature about extensions of neural networks, variance estimation methods in neural network and other ideas in statistical intervals.

2.1 Variance Estimation in Neural Network

As introduced in the paper [94], uncertainty estimation was fully discussed and methods are all fully introduced. Deterministic neural networks, Bayesian neural networks, ensemble of neural networks, and test-time data augmentation approaches is discussed for the latest developments, and practical application in order to measure uncertainty, as well as approaches for the calibration of neural networks is also discussed. The paper also and give an overview of existing baselines and available implementations. I will also discuss these existing method and their latest development.

At first, we need to talk about the necessity of uncertainty estimation, which mainly due to some drawback of neural network: deep neural network is not transparent enough, makes the outcome not trustworthy[95]; in-domain and out-of-domain samples are hard to distinguish[96, 97]; overfitting occurs [98, 99] and uncertainty estimates for DNN decision[100] is not reliable; and neural network hard to fight against adversarial attacks [101, 102, 103].

Then we delve into the concept of predictive uncertainty in machine learning models, which encompasses data uncertainty, model uncertainty[104], and distribution uncertainty[105].

Regarding the input data domain, we can identify different forms of data uncertainty: in domain uncertainty (innate from variation of the data)[106]; domain shift uncertainty (different distribution of prediction data then observed data)[107] and out of domain uncertainty (data point significantly deviate from the training distribution)[108, 109, 110, 111].

In general, there are 4 types of method used to determine uncertainty for deep neural networks: Single determinist method directly output neural network's prediction uncertainty ([105, 112, 113, 114, 115, 116, 117, 118, 119]); Bayesian Neural Network method to assume data belongs to some type of distribution,using Variational Inference([120, 121]) and Sampling Approach([122]); test time augmentation used to augment the input data at test time in order to generate the certainty of prediction([100, 123, 124, 125]) and ensemble networks.

Especially when we talking about ensemble neural networks, my research talks about using this method for uncertainty estimation as well. [126] introduced an ensemble training pipeline to quantify predictive uncertainty within DNNs. Using

bagging, in [127] it found bagging to worsen the predictive accuracy of ensemble methods on the investigated tasks. [128] introduced a framework for the comparison of uncertainty quantification methods with a specific focus on real life applications. [129] found ensemble methods to deliver more accurate and better calibrated predictions on active learning tasks than Monte Carlo Dropout. [130] presented an ensemble method for the improved detection of out-of-distribution samples.

2.2 Comparison of NN versus Linear model

The theoretical comparison between neural networks and generalized linear models (GLMs) has been extensively studied in the literature. Early work by [131] conducted a detailed analysis and comparison of neural networks with traditional regression models. Later, [132] proposed that neural networks are a flexible class of nonlinear regression models. [133] further elaborated on the overlap between neural networks and GLMs, suggesting that neural networks are a type of overlapping GLM. These findings are similar to what has been discussed in Chapter 1. [134] also explored the advantages of using neural network models instead of parametric regression models.

Regarding the relationship between neural networks and logistic regression, [135] has conducted an extensive analysis.

In [136], a review of articles comparing multilayered feedforward neural networks with standard statistical techniques such as regression analysis and logistic regression was conducted. The paper summarizes the relevant articles in various areas of applications and highlights an important advantage of neural networks. Specifically, neural networks can approximate any nonlinear mathematical function approximately. This aspect of neural networks is particularly useful when the relationship between the variables is unknown or complex and difficult to handle statistically. However, the determination of various parameters is not straightforward, and finding the optimal configuration of neural networks is time-consuming.

Another feature of neural networks is the lack of interpretability of the weights obtained during the model building process. In contrast, statistical models, such as GLMs, allow for the interpretation of coefficients of individual variables in a statistical manner. Furthermore, the parametric assumptions of these models can be used to infer the significance of certain variables in prediction or classification problems.

2.3 Tolerance interval's application

Tolerance intervals are a useful tool in statistical analysis for establishing bounds around a population distribution or regression model. Several researchers have extensively studied the theory and applications of tolerance intervals, including [137], [138], [139] and [140]. Initially, the study of tolerance intervals under the assumption of normality was initiated by [141] and [142], with further research on the normal

assumption topic using different approaches, as discussed in [143] and [144]. Later, theoretical analysis on nonnormal tolerance intervals was also studied, with [145] and [146] providing studies for exponential families and gamma distributions for continuous distributions, and [147] and [148] providing initial studies for concrete distributions such as discrete exponential families, binomial and Poisson distributions. Nonparametric tolerance intervals have also been studied by [149] and [150], while Bayesian tolerance intervals have been examined in [151] and [152]. Recently, approximation methods for tolerance intervals have also been examined, such as in [153], which compared tolerance factor approximations and the exact tolerance factors for normal populations.

When extending tolerance intervals to regression models, [154] was the first to do so by using the multiplication factor k from [137] for linear regression. The general form proposed by [155] utilizes transformation and/or weighting to extend linear regression to nonlinear. In a discussion about confidence and prediction intervals for nonparametric regression, [156] pointed out that [157] used [141] tolerance interval for univariate data. In this paper, we will be using neural networks as a nonparametric regression model.

There are multiple ways to construct tolerance intervals, each focusing on different aspects. For example, [158] considered simultaneous tolerance intervals in regression, while later articles ([159],[160]) talked about calibration intervals using simultaneous construction, which are bounds for the predictors when a response is given. [159] calculates tolerance intervals that "control the center" by requiring that the tolerance interval be constructed around an estimate of the center of the distribution, while [160] calculates tolerance intervals that "control the tails" by ensuring that the proportion of the distribution falling outside the interval is below a specified amount in each tail. The regression tolerance intervals [157] we will construct will be formulated to control the center, as discussed in [143], [137], and [153].

Chapter 3 Confidence Interval For Neural Network (Direct Evaluation)

We aim to investigate a method that incorporates prediction information from neighboring predictors to observe the coverage and behavior of this approach.

3.1 Direct Variance Estimation

3.1.1 Neighbourhood

In the field of topology, a neighborhood of a point refers to a set of points that includes the initial point and allows movement in any direction within the set without leaving it. In one-dimensional space, we define the neighborhood as follows:

Definition 3.1.1. If x is any single point, a and δ are both positive number, then, the neighborhood of a is defined as

$$B_\delta(a) = \{x \in R : |x - a| < \delta\}$$

In our approach, we aim to utilize the information from $B_\alpha(x_i)$ for all $i \in 1, 2, \dots, n$, where α can vary within a chosen range, and x_i represents any point in the predictors.

We want to incorporate the predictions \hat{y}_j for j in $B_\alpha(x_i)$ and examine how this information can be used for variance estimation.

3.1.2 MAD Smoothing

Median Absolute Deviation (MAD) is a robust measure of variability for a univariate sample of quantitative data. Here, we provide a brief introduction to this method.

If we have a set of discrete numbers x_1, x_2, \dots, x_n , the sample median is defined as the middle value of the ordered statistics when the total number of values, denoted by n , is odd. When n is even, the sample median is calculated as the average of the ordered statistics with ranks $n/2$ and $(n/2) + 1$. [161] prove that the median has a breakdown point of 50%, indicating that it remains robust even in the presence of outliers. Additionally, the influence function of the median is bounded, providing a measure of the effect of removing outliers.

When it comes to robust estimation of scale, there are several statistical methods to consider. One approach involves calculating the average deviation from the median, denoted as $ave_i|x_i - med_j x_j|$. This estimator has a breakdown point of 0, which means it is sensitive to outliers. Alternatively, we can compute the median absolute deviation of the mean, denoted as $med_i|x_i - ave_j x_j|$. This estimator exhibits the best possible breakdown point of 50%, meaning it remains robust even when up to half of the data points are outliers. Furthermore, the influence function of this estimator

has the sharpest possible bound among all scale estimators[162].

In our analysis, we are interested in evaluating the scale factor for the standard deviation. To accomplish this, we can proceed as follows:

$$\frac{1}{2} = P(|X - \mu| \leq MAD) = P\left(\left|\frac{X - \mu}{\sigma}\right| \leq \frac{MAD}{\sigma}\right) = P(|Z| \leq \frac{MAD}{\sigma}) \quad (3.1)$$

Here Z represent a random variable follow standard normal distribution.

Using the fact $\Phi(-MAD/\sigma) = 1 - \Phi(MAD/\sigma)$ (symmetric property of normal distribution) and $\Phi(MAD/\sigma) - \Phi(-MAD/\sigma) = 1/2$ (From previous formula), we have $MAD/\sigma = \Phi^{-1}(3/4) = 0.67449$, hence $\hat{\sigma} = k \cdot MAD = 1/0.67449 \cdot MAD \approx 1.483 \cdot MAD$.

3.2 Experiment

First we build a dataset simulated by the following formula:

$$y_i = 3\cos(x_i) - 5(x_i/15)^2 + \epsilon_i \quad (3.2)$$

The error term ϵ_i comes from a t-distribution with 2 degrees of freedom.

We aim to examine which parameter in the base learner, specifically a 2-layer neural network, can effectively capture the underlying structure of the dataset. In this study, we impose constraints on the network structure by using the rectified linear unit (ReLU) as the activation function and the mean square loss as the optimization goal. To evaluate the performance of our model, we split the data into a training set (70% of the data) and a validation set (30% of the data).

To determine the optimal number of neurons in each layer, we conduct a grid search. For both layers, we explore neuron sizes ranging from 2 to 30 with an increment of 1. Through this search, we find that the first layer achieves the best performance with 8 neurons, while the second layer performs optimally with 29 neurons. In our optimization process, we employ the Adam optimizer with a learning rate of 0.01.

In the visualization, we plot the simulated data as blue dots, the true parameter values are represented by a solid line, and the dashed line corresponds to the fitted response variable learned by our neural network.

Upon initial observation, it appears that there might be an outlier around $x=7$, and the fitted line seems to closely resemble the true parameter line. However, there are still some potential outliers around 2 and 8. To gain a clearer understanding, we zoom in on the response variable, limiting the range from -10 to 10.

Upon closer inspection, we observe outliers near 2, indicating that the fitted line tends to deviate towards the left of the true parametric model. On the other hand,

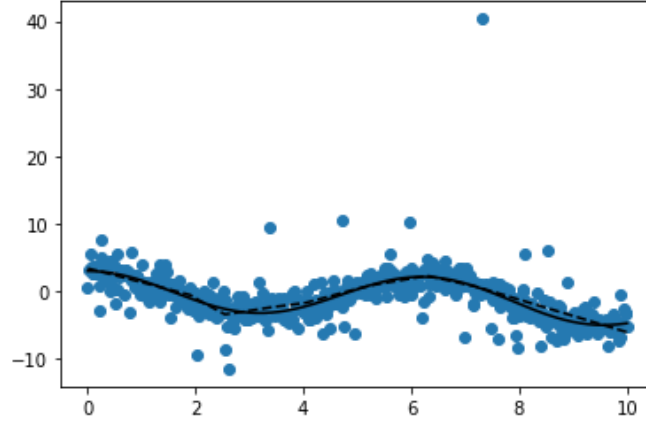


Figure 3.1: True data, true parametric line and fitted line

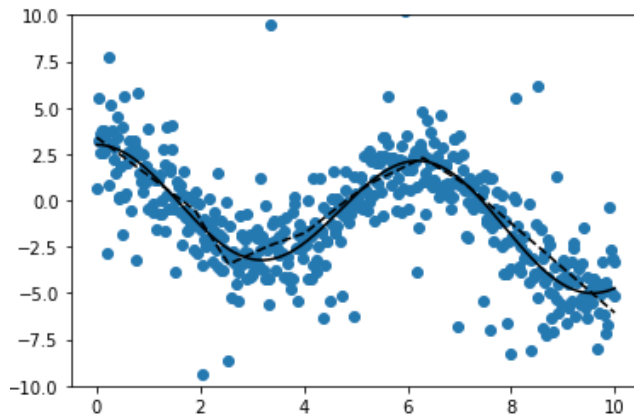


Figure 3.2: True data, true parametric line and fitted line(zoom in)

as the number of outliers increases towards the right side, our neural network may tend to learn a straight line rather than capturing the curvature present in the true parametric model.

3.2.1 Standard Deviation

For the sliding window method, our initial step is to estimate the prediction standard deviation, denoted as σ , using the standard error \hat{s} of the predictions within our neighborhood. It's important to note that we need to divide the number of cases, denoted as c , in the neighborhood to make this adjustment.

To define our neighborhood size, we utilize $B_{0.05}(x_i)$. In this case, we employ the normal assumption to construct a 95% confidence band, employing a multiplier of $z_{0.975} = 1.96$.

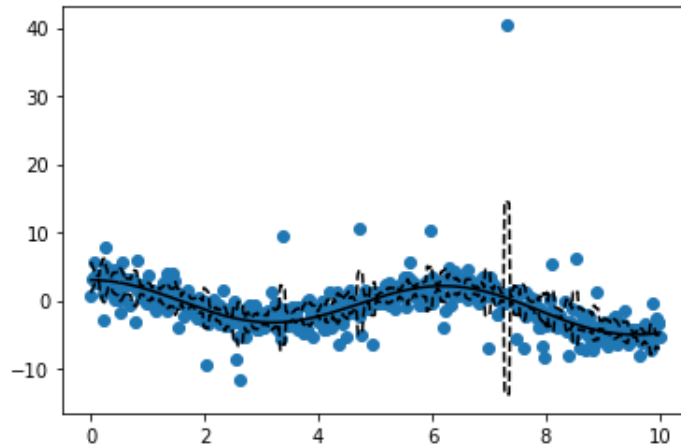


Figure 3.3: True data, true parametric line and fitted line (x axis represent explanatory variable, blue dot represent true data with noise in simulation, and dashed line represent 95% confidence band)

Upon observing the plot, we can determine that the coverage rate is 95.4%, which appears to be quite good. However, the confidence interval lacks smoothness and is heavily influenced by extreme values.

When we slide our neighborhood size from 0 to 1, we observe the development of the coverage rate in relation to the neighborhood size.

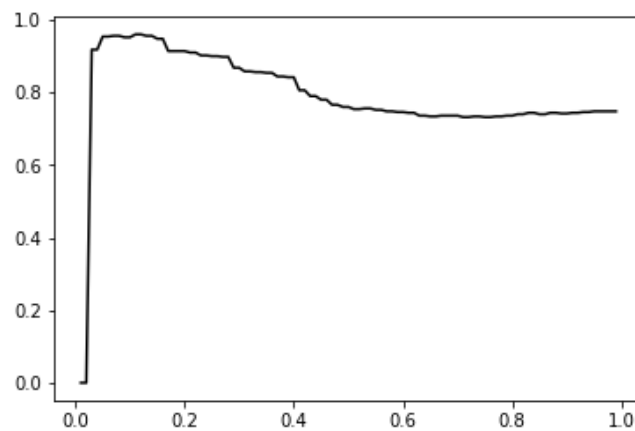


Figure 3.4: sliding neighbourhood size from 0 to 1 and see coverage rate for 95% confidence interval in standard error

Initially, our coverage rate is approximately 95%, which aligns with our confidence band. As we increase the window size, the coverage rate gradually decreases and stabilizes when $\delta > 0.4$. It's worth noting that in the early stages, the standard

deviation of a prediction in our small window (around 0.01) is 0, resulting in a coverage rate of 0 initially.

3.2.2 MAD neighbourhood

We want to use the MAD in the neighborhood $B_{0.05}(x_i)$ of x_i 's in all element from interval $x \in [0, 10]$. We want to use $\hat{\sigma} = 1.483MAD$, where $MAD = med|\hat{y}_j - y_i|$, where y_i are true value in x_i , and \hat{y}_j is the prediction of x_j in neighbourhood of x_i where $\delta = 0.05$.

We construct a 95% confidence band utilizing the normal assumption and divide $\hat{\sigma}$ by c , which represents the number of cases within the neighborhood. Upon examining the zoomed-in plot shown in Figure (3.5), we observe that although the confidence band is not perfectly smooth, it effectively encompasses the majority of the true parameter line (solid line) in the model. Furthermore, the confidence band appears to encapsulate the data points with random noise within its boundaries. Notably, the presence of outliers does not exert a significant influence on the confidence band, which achieves a high coverage rate of 97.4%.

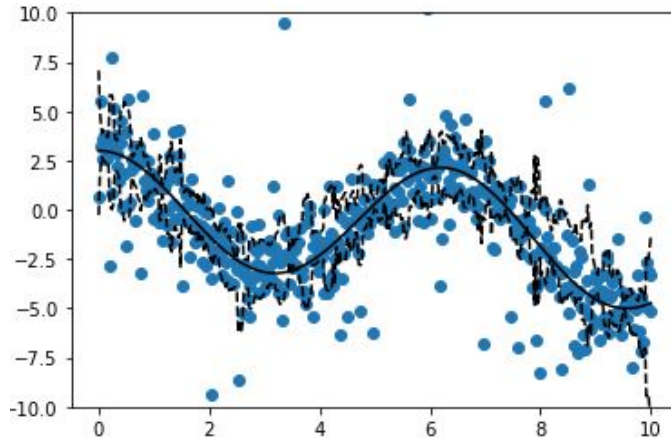


Figure 3.5: True data, true parametric line and fitted line with confidence band of using 95% confidence band make use of MAD approximation

Due to the application of the median, the impact of outliers has been mitigated, resulting in a smoother confidence band compared to using the standard error directly. However, to examine the progression of the coverage rate, we slide the neighborhood size from 0 to 1, as depicted in Figure (3.6). This analysis allows us to observe how the coverage rate evolves as the neighborhood size changes.

Examining Figure (3.6), we can analyze the coverage rate's behavior when adjusting the window size. Notably, the coverage rate appears to be quite satisfactory within the window size range of 0 to 0.1. However, as the window size exceeds this range, the coverage rate experiences a rapid decay before stabilizing at a window size

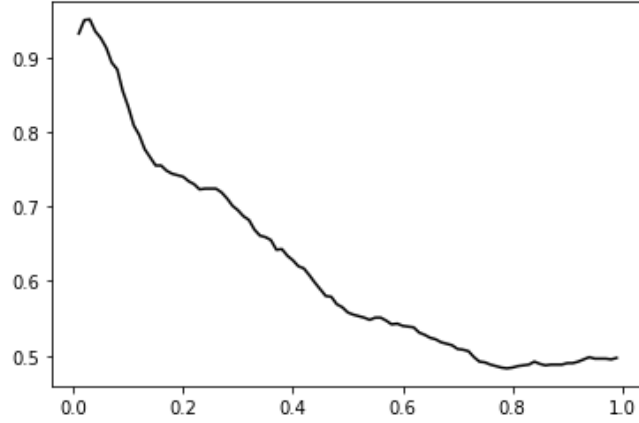


Figure 3.6: sliding neighbourhood size from 0 to 1 and see coverage rate for 95% confidence interval in MAD

of 0.8. It is also worth noting that the initial decay in the coverage rate is relatively smooth compared to using the standard error directly on the predictions within the window.

3.3 Discussion

Here we can see several research extension of using MAD as robust estimator for variance.

3.3.1 Sample Size n

We want to see sample size's effect on sliding window using MAD estimate for 95% confidence band coverage.

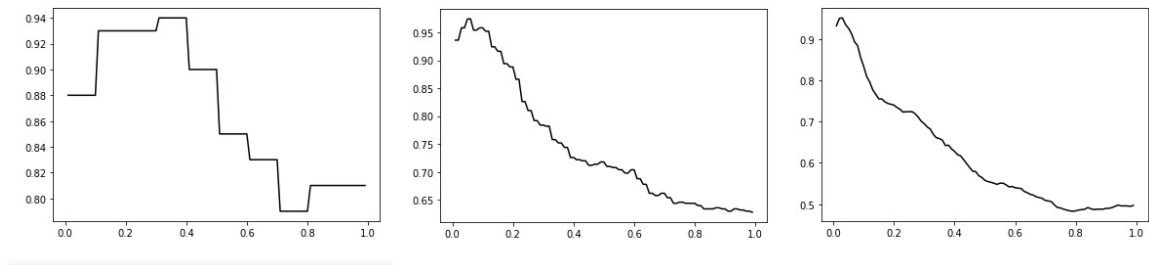


Figure 3.7: From left to right, we have sample size n change from 100, 500 to 1000

From Figure (3.7), we can observe certain patterns regarding the coverage rate as the sample size increases. When the window size approaches 1, the coverage rate tends to decrease. However, when the window size is around 0.05, the coverage rate

remains close to 0.95, corresponding to a 95% confidence interval.

There are two notable observations. Firstly, as the sample size increases, the plot of coverage rate versus window size becomes smoother. Nevertheless, the overall trend remains consistent: an initial increase in coverage rate followed by a subsequent decrease. This suggests that the coverage rate may be more influenced by the window size rather than the sample size. It is possible that the coverage rate is associated with a function representing the relationship between window size and the learning ability of a base learner.

Secondly, after a considerable period of decay in the coverage rate as the window size increases, the coverage rate eventually reaches a stable state. This implies that using a small window size (approximately 1/200 of the range) appears to be a reasonable choice when there is sufficient information available for the development of the curve.

3.3.2 Random Error

If we generate $n=500$ random errors, denoted as ϵ , from a distribution F with zero mean and any type of variance, we can examine how well a neural network fits these errors. To assess the coverage rate of the neural network, we employ a sliding window approach combined with the Median Absolute Deviation (MAD) method for robust variance estimation.

Initially, we choose F to be a t-distribution with 2 degrees of freedom and set the random seed to 42. We train a neural network with 2 layers: the first layer consists of 14 neurons, while the second layer contains 15 neurons. We then evaluate the neural network's performance by examining its fitting to the generated errors.

To quantify the uncertainty in the performance estimates, we construct a 95% confidence interval using a sliding window of size 0.05. Additionally, we investigate the effect of changing the sliding window size, denoted as m , on the performance and confidence interval. We vary the window size from 0 to 1, increasing it by 0.02 in each iteration.

Examining the plot on the left, we observe that the fitting of the neural network displays fluctuation towards the left side. However, in the middle of the dataset, the neural network demonstrates a strong ability to fit the random errors accurately.

Furthermore, when the window size is fixed at 0.05, the confidence interval for the performance of the neural network fitter appears reliable. The interval maintains a smooth distribution around the true parameters, and the coverage percentage does not exhibit excessive confidence or conservativeness.

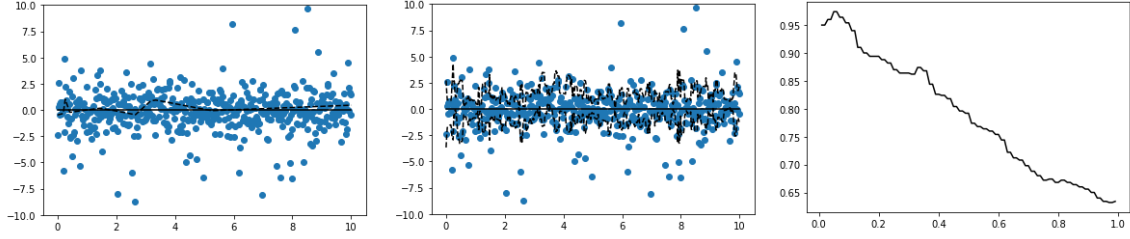


Figure 3.8: From left to right, we have the fitting of neural network’s performance, 95% CI with 97.4% coverage percentage with sliding window size 0.05, and the growth of sliding window size’s effect on coverage rate.

Nevertheless, as the window size increases while keeping the sample size fixed at 500, we observe a gradual reduction in coverage. This suggests that the larger window sizes may lead to decreased accuracy in capturing the true performance of the neural network.

The plot below shows if we change the random error from a t distribution to a normal distribution with mean 0 and variance 1, by applying MAD method for variance estimation.

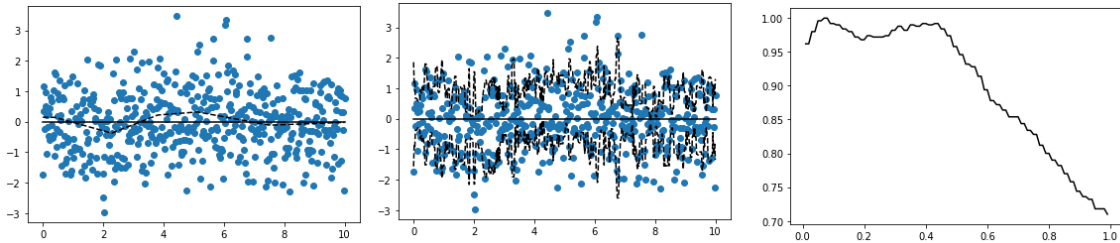


Figure 3.9: From left to right, we have the fitting of neural network’s performance, 95% CI with 99.6% coverage percentage with sliding window size 0.05, and the growth of sliding window size’s effect on coverage rate.

Analyzing the plot on the left, we observe that the fitted line generally captures the true regression line ($y = 0$), although it exhibits fluctuations around the true parameter. However, there are no distinct patterns apparent in the graph.

The 95% confidence interval with a window size of 0.05 successfully captures the true parameters, but the interval itself appears to be less smooth and demonstrates overconfidence.

Upon varying the sliding window size from 0 to 1 with an increment of 0.02, we note that the coverage probability initially increases, then fluctuates around 0.975 be-

fore gradually decreasing. Prior to the 0.5 mark, the coverage probability experiences an upward trend, followed by a decline and subsequent fluctuation.

3.4 Conclusion

When using a neural network, confidence intervals can be employed to estimate the uncertainty surrounding the predicted output of the model. To calculate these intervals, variance is used as a measure of the spread of values, allowing for the quantification of uncertainty.

Directly evaluating variance using residuals and the neighborhood of observations can be computationally expensive. By utilizing the full model information present in the data, the model fitting process becomes less susceptible to model bias.

Instead of assuming a constant variance throughout the entire population and using the entire population to estimate variance, it is crucial to consider that predictions may deviate from the true parameters. In such cases, the neighborhood method introduced in this chapter proves to be a superior option for estimating variance at specific points. Comparing the Median Absolute Deviation (MAD) method to the standard deviation method in the neighborhood, it becomes evident that, for small neighborhood sizes, the MAD method yields confidence intervals closer to the nominal level. This is particularly important as the number of variables increases, making it challenging to identify the neighborhood in high-dimensional space. Additionally, the currently employed neighborhood method may face difficulties when used with standard error variance estimation methods.

In conducting sensitivity analysis with varying sample sizes and neighborhood window sizes in the presence of random errors, we conclude that sample size has minimal impact on neighborhood size selection. Furthermore, when random errors are present, neural networks demonstrate effective model fitting. The decay rate of the coverage percentage is slower for smaller window sizes and becomes faster for larger window sizes.

In future studies, it would be valuable to explore the optimal neighborhood size required to achieve confidence intervals at the nominal level. Additionally, investigating multivariate regression confidence intervals for neural networks, which can be compared to traditional KL-Divergence-based Bayes estimation([163]) for neural network variance estimates, would be beneficial.

Chapter 4 Confidence Interval For Neural Network (Resampling)

4.1 Variance Estimation Method using Resampling

In this study, we employed bootstrap, jackknife, and extension methods to estimate the variance in a fixed-two layer neural network, aiming to explore the coverage properties of confidence intervals.

4.1.1 Jackknife

The Jackknife method, initially introduced and studied by Maurice Quenouille[164, 165], and later expanded upon by John Tukey[166], represents a straightforward and versatile approach for estimating variance in various problem domains.

If there exist a dataset $\mathbf{x} = (x_1, \dots, x_n)$ as our dataset, the i th *jackknife sample* $\mathbf{x}_{(i)}$, which is delete i th element from \mathbf{x} ,

$$\mathbf{x} = (x_1, x_2, \dots, x_{i-1}, x_{i+1}, \dots, x_n) \quad (4.1)$$

and for the population parameter $\hat{\theta} = g(\mathbf{x})$, each *jackknife replication* $\hat{\theta}_{(i)}$ will also be achieved from $\mathbf{x}_{(i)}$, say

$$\hat{\theta}_{(i)} = g(\mathbf{x}_{(i)}) \quad (4.2)$$

In [167], the estimate of bias in bootstrap is defined by

$$bias_{jack} = (n - 1)(\hat{\theta}_{(\cdot)} - \hat{\theta}) \quad (4.3)$$

where

$$\hat{\theta}_{(\cdot)} = \sum_{i=1}^n \hat{\theta}_{(i)} / n \quad (4.4)$$

Hence, the bias-correct version of jackknife estimator is in the form

$$\hat{\theta}_{U-jack} = \hat{\theta} - bias_{jack} = n\hat{\theta} - (n - 1)\hat{\theta}_{(\cdot)} \quad (4.5)$$

It is mathematically established that $bias_{jack}$ often serves as an unbiased estimator for the bias of the overall sample statistic. In fact, it has been demonstrated in 9.1 that $bias_{jack}$ is a quadratic Taylor series approximation to $bias_F = bias_F(\hat{\theta}, \theta) = E_F[g(\mathbf{x})] - \theta$, where F represents the theoretical probability distribution.

4.1.2 Pseudo-values

As per the jackknife method, a pseudoestimate is obtained by excluding X_i from a set of X_n and utilizing the remaining elements to estimate the population parameter θ . We denote this estimate as $\hat{\theta}(i)$. Additionally, the pseudo value $p(i) = n\hat{\theta} - (n - 1)\hat{\theta}_{(i)}$

is introduced as a measure[168]. The pseudo-value for the average of all jackknife estimators is defined as follows:

$$\begin{aligned} p_{(\cdot)} &= \frac{1}{n} \sum_{i=1}^n (n\hat{\theta} - (n-1)\hat{\theta}_{(i)}) \\ &= \sum_{i=1}^n \hat{\theta} + (n-1) \left(\frac{1}{n} \sum_{i=1}^n \hat{\theta}_{(i)} \right) = n\hat{\theta} - (n-1)\hat{\theta}_{(\cdot)} \end{aligned} \quad (4.6)$$

Which is actually the bias-corrected jackknife estimate. The natural estimator for the variance of $\hat{\theta}_{U-jack}$ is s_{U-jack}^2/n and s_{U-jack}^2 is sample variance of n pseudo-values:

$$\begin{aligned} \hat{var}(\hat{\theta}_{U-jack}) &= \frac{s_{U-jack}^2}{n} = \frac{1}{n} \frac{1}{n-1} \sum_i (p_{(i)} - p_{(\cdot)})^2 \\ &= \frac{n-1}{n} \sum_i (\hat{\theta}_{(i)} - \hat{\theta}_{(\cdot)})^2 \end{aligned} \quad (4.7)$$

4.1.3 Infinitesimal Jackknife

If weight is attached to each observation (using projection expression in jackknife), instead of assigning weight 0 for the omitted variable in jackknife, we should assign weight slightly less weight than others, and consider limiting case that the weight approaches zero. Notice that the $\sum w_i$ not necessary to be 1. If we consider T as discrete probability distributions, that is concentrate of the mass on a finite number of points. We can define T as a function of $2n$ variables:

$$T(X_1, \dots, X_n; w_1, \dots, w_n) \quad (4.8)$$

If G is any probability distribution for which T is defined, and c is a positive constant, then we let $T(cG) = T(G)$.

If we reduce w_i by ϵ and leave other weight at $1/n$, we have

$$\hat{\theta}_{(i)}(\epsilon) = T(X_1, \dots, X_n; \frac{1}{n}, \dots, \frac{1}{n} - \epsilon, \frac{1}{n}) \quad (4.9)$$

Assume we can different T with respect to w_i , and define

$$\begin{aligned} \hat{D}_i &= \frac{\partial T}{\partial w_i} \Big|_{x_j=X_j, w_j=\frac{1}{n}, j=1, \dots, n} \\ \hat{D}_{ii} &= \frac{\partial^2 T}{\partial w_i^2} \Big|_{x_j=X_j, w_j=\frac{1}{n}, j=1, \dots, n} \end{aligned} \quad (4.10)$$

We can form the Taylor series expansion

$$\begin{aligned} \hat{\theta}_{(i)}(\epsilon) - \hat{\theta} &= T(\dots, \frac{1}{n} - \epsilon, \dots) - T(\dots, \frac{1}{n}, \dots) \\ &= -\epsilon \hat{D}_i + \frac{\epsilon^2}{2} \hat{D}_{ii} - \dots \end{aligned} \quad (4.11)$$

The variance estimate $\hat{V}(\epsilon)$ is defined as

$$n^2 \epsilon^2 \hat{V}(\epsilon) = (1 - \epsilon) \sum [\hat{\theta}_{(i)}(\epsilon) - \hat{\theta}_{(\cdot)}(\epsilon)]^2 \quad (4.12)$$

If $\epsilon = 1/n$, then this degenerate go jackknife variance estimator. We can see that

$$\hat{\theta}_{(i)}(\epsilon) \cong \hat{\theta} - \epsilon \hat{D}_i \quad (4.13)$$

Lemma 1. *Let G as discrete distribution and $P(X = z_i) = g_i$ and w_i be weight attached to value z_i , and assume z_i are distinct. If the derivatives are exist, then*

$$\sum_i g_i D_i^G = 0 \text{ and } \sum_i \sum_j g_i g_j D_{ji}^G = 0$$

where D_i^G is derivative of T with respect to w_i evaluated at $W=G$.

Proof.

$$0 = \frac{dT(cg_1, \dots, cg_I)}{dc} = \sum \frac{\partial T}{\partial w_i} \frac{dw_i}{dc} = \sum_i g_i \frac{\partial T}{\partial w_i}$$

Similarly,

$$\sum_i g_i \sum_j g_j \frac{\partial T}{\partial w_j \partial w_i} = 0$$

If we evaluate at $c=1$ for both equation, the proof is completed. \square

Hence

$$\hat{\theta}_{(\cdot)}(\epsilon) \cong \hat{\theta} - \frac{\epsilon}{n} \sum \hat{D}_i = \hat{\theta} \quad (4.14)$$

since c in lemma can be chosen by ϵ/n We will have

$$\hat{\theta}_{(i)}(\epsilon) - \hat{\theta}_{(\cdot)}(\epsilon) \cong -\epsilon \hat{D}_i \quad (4.15)$$

Using equation (4.12), we can see that

$$n^2 \epsilon^2 \hat{V}(\epsilon) \cong (1 - \epsilon) \sum \epsilon^2 \hat{D}_i^2 \quad (4.16)$$

Letting $\epsilon \rightarrow 0$, we have

$$n \hat{V}(0) = \frac{1}{n} \sum \hat{D}_i^2 \quad (4.17)$$

$\hat{V}(0)$ is the IJK[169] variance estimate for $\hat{\theta}$.

Also, the bias term can be expressed as $\hat{B}(\epsilon)$

$$n^2 \epsilon^2 \hat{B}(\epsilon) = n(1 - \epsilon)(\hat{\theta}_{(\cdot)}(\epsilon) - \hat{\theta}) \quad (4.18)$$

Writing in Taylor expansion, will be

$$\hat{\theta}_{(i)}(\epsilon) \cong \hat{\theta} - \epsilon \hat{D}_i + \frac{\epsilon^2}{2} \hat{D}_{ii} \quad (4.19)$$

and using $\sum \hat{D}_i = 0$ and $\frac{1}{n} \sum \hat{\theta}_{(i)}(\epsilon) = \hat{\theta}_{(\cdot)}(\epsilon)$, we have

$$\begin{aligned}\hat{\theta}_{(\cdot)}(\epsilon) &\cong \hat{\theta} + \frac{\epsilon^2}{2} \hat{D}_{ii} \\ n^2 \epsilon^2 \hat{B}(\epsilon) &= n(1 - \epsilon) \left(\frac{\epsilon^2}{2n} \sum \hat{D}_{ii} \right)\end{aligned}\tag{4.20}$$

Let $\epsilon \rightarrow 0$, we have

$$n \hat{B}(0) = \frac{1}{2n} \sum \hat{D}_{ii}\tag{4.21}$$

This $\hat{B}(0)$ is the IJK bias in $\hat{\theta}$.

From previous introduction, we know that the variance of Infinitesimal Jackknife estimate is

$$\hat{V}_{IJ}^\infty = \sum_{i=1}^n Cov_*[N_i^*, t^*(x)]^2\tag{4.22}$$

where $t^*(x)$ represent a base learner on bootstrap sample, and N_i^* is the i th training example appears in a bootstrap sample.

But we can only work with finite number of B bootstrap replicates, here we will have

$$\hat{V}_{IJ}^B = \sum_{i=1}^n \hat{Cov}_i, \text{ with } \hat{Cov}_i = \frac{\sum_b (N_{bi}^* - 1)(t_b^*(x) - \bar{t}^*(x))}{B}\tag{4.23}$$

N_{bi}^* represent the i th observation appear times in bootstrap example b .

4.1.4 Jackknives' relationship with bootstrap

Jackknife can be actually explained as an approximation to the bootstrap. If we consider linear statistic

$$\hat{\theta} = s(x) = \mu + \frac{1}{n} \sum \alpha(x_i)\tag{4.24}$$

We know that jackknife standard error

$$\hat{s}e_{Boot} = \frac{n-1}{n} \sum_i (\hat{\theta}_{(i)} - \hat{\theta}_{(\cdot)})^2\tag{4.25}$$

and bootstrap standard error

$$\hat{s}e_B = \left\{ \sum_{b=1}^B [\hat{\theta}^*(b) - \hat{\theta}^*(\cdot)]^2 / (B-1) \right\}^{1/2}\tag{4.26}$$

where $\hat{\theta}^*(\cdot)$ is the average of bootstrap statistics. And the bootstrap standard error multiply $\{(n-1)/n\}^{1/2}$ is the jackknife standard error.

4.2 Confidence Interval Building

Here we want to see performance of different variance estimation method's performance on building confidence intervals for fixe 2-layer with fixed neurons neural networks.

4.2.1 Normal Assumption

We know that if sample size n grows large, the distribution of $\hat{\theta}$ becomes approximately normal, and mean will be around θ with variance near $\hat{s}e$. We will use $z_{(\alpha/2)}$ represent the $100*\alpha/2$ percentile point as cut-off point, then, since $\frac{\hat{\theta}-\theta}{\hat{s}e}$ is approximate normal distribution, we will have:

$$P(z_{\alpha/2} \leq \frac{\hat{\theta} - \theta}{\hat{s}e} \leq z_{1-\alpha/2}) = 1 - \alpha \quad (4.27)$$

Can also be change into form

$$[\hat{\theta} - z_{1-\alpha/2} * \hat{s}e, \hat{\theta} - z_{\alpha/2} * \hat{s}e] \quad (4.28)$$

Or we can write in a form

$$\hat{\theta} \pm z_{1-\alpha/2} * \hat{s}e \quad (4.29)$$

4.2.2 Percentile Method

We can let \hat{G} be the empirical distribution of bootstrap statistics $\hat{\theta}^* = s(\mathbf{x}^*)$. The $1 - \alpha$ percentile confidence interval can be defined by $\alpha/2$ and $1 - \alpha/2$ percentiles of \hat{G} :

$$[\hat{\theta}_{\%,low}, \hat{\theta}_{\%,upper}] = [\hat{G}^{-1}(\alpha/2), \hat{G}^{-1}(1 - \alpha/2)] \quad (4.30)$$

And we define the $\hat{G}^{-1}(\alpha)$ to be the $100 * \alpha$ percentile of the bootstrap distribution.

We can consider B independent bootstrap datasets $\mathbf{x}^{*1}, \mathbf{x}^{*2}, \dots, \mathbf{x}^{*B}$ and compute the bootstrap statistics $\hat{\theta}^*(b) = s(\mathbf{x}^{*b})$. The $\hat{\theta}_B^{(\alpha)}$ will be the $100 * \alpha$ th percentile of bootstrap statistic $\hat{\theta}^*$, then the approximate $1 - \alpha$ percentile interval is

$$[\hat{\theta}_{\%,low}, \hat{\theta}_{\%,upper}] \approx [\hat{\theta}_B^{*(\alpha/2)}, \hat{\theta}_B^{*(1-\alpha/2)}] \quad (4.31)$$

If the sampling distribution of $\hat{\theta}^*$ is roughly normal, the percentile interval may agrees on the standard normal interval. But if the distribution of $(\hat{\theta})^*$ is not normally distributed, the percentile interval seemed to perform better than normal approximation, unless the transform of θ fits normal distribution in the following lemma:

Lemma 2. *If transformation $\hat{\phi} = f(\hat{\theta})$ have a perfectly normal approximation*

$$\hat{\phi} \sim N(\phi, a^2) \quad (4.32)$$

Then the percentile interval of $\hat{\theta}$ is $[f^{-1}(\hat{\phi} - z_{1-\alpha/2}a), f^{-1}(\hat{\phi} - z_{\alpha/2}a)]$

Proof. The percentile interval of $\hat{\phi}$ will be

$$[\hat{\phi}^{(\alpha)}, \hat{\phi}^{1-(\alpha)}] \quad (4.33)$$

Since $\hat{\phi}$ is a normal distributed random variable, the normal interval for $\hat{\phi}$ will be

$$[\hat{\phi} - z_{1-\alpha/2} * a, \hat{\phi} - z_{\alpha/2} * a] \quad (4.34)$$

Since $\hat{\phi} = f(\hat{\theta})$ then $\hat{\theta} = f^{(-1)}(\hat{\phi})$. The percentile interval will be

$$[f^{-1}(\hat{\phi} - z_{1-\alpha/2} * a), f^{-1}(\hat{\phi} - z_{\alpha/2} * a)]$$

□

4.2.3 Bootstrap Size's Effect

We want to study the feature of the distribution of θ belongs to the population, noted by $\hat{\gamma}_B$. The variance of $\hat{\gamma}_B$ depend on sample size n and bootstrap sample size B. In fact, if the variance of bootstrap estimate standard error of \bar{x} is our research target, the variance of standard error has the form

$$var(\hat{s}e_B) = \frac{c_1}{n^2} + \frac{c_2}{nB} \quad (4.35)$$

where c_1 and c_2 are constants depend on population distribution F. The c_1/n^2 represents sampling variation, which comes from sample size n is smaller than whole population. While the second part is c_2/nB represent the resampling variation, and it comes from bootstrap in resample.

Since n, B change will lead to change of $E(\hat{s}e_B)$, in hear we will consider the coefficient of variation of $\hat{s}e_B$

$$cv(\hat{s}e_B) = \frac{\sqrt{var(\hat{s}e_B)}}{E(\hat{s}e_B)} \quad (4.36)$$

Using simplification in Appendix(9.3) we can see that

$$cv(\hat{s}e_B) = \left\{ cv(\hat{s}e_\infty)^2 + \frac{E(\hat{\Delta}) + 2}{4B} \right\}^{1/2} \quad (4.37)$$

If the population parameter is using sample mean as estimation, while each sample are iid normally distributed, then

$$cv(\hat{s}e_B) = \left[\frac{1}{2n} + \frac{1}{2B} \right]^{1/2} \quad (4.38)$$

We choose n=500 and range B from 1 to 2000 to see how the coefficient of variation behaves, notice the dashed line is $\hat{s}e_\infty$.

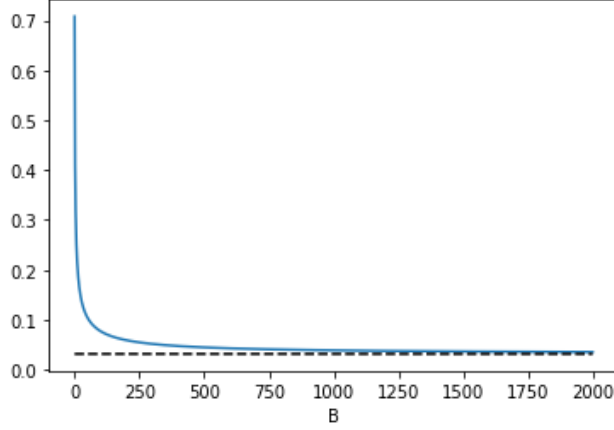


Figure 4.1: CV vs B when sample size fixed to 500

Notice that there seems to be a singular point around 200, actually it comes from

$$\begin{aligned}
 \frac{d^2 \text{cv}(\hat{s}e_B)}{d B^2} &= \frac{d - \frac{1}{4\sqrt{\frac{1}{2B} + \frac{1}{2n}B^2}}}{d B} \\
 &= \frac{4B + 3n}{16n(\frac{1}{2B} + \frac{1}{2n})^{\frac{3}{2}}B^4} \\
 &\stackrel{\text{set to } 0}{=} B = 3n/4 = 375
 \end{aligned}
 \tag{4.39}$$

We know a random forest's performance cannot be worsened by adding more trees to it, comparing it to [170], I choose the B larger than 500, I will use 500, 2000, 5000 to see coverage rate of 95% confidence interval.

4.3 Experiment

We want to compare coverage probability of two different method of confidence interval coverage probability and coverage performance using bootstrap, jackknife and infinitesimal jackknife method.

We still use the synthetic data generated by (5.6), and see ensemble other method's performance. When the bootstrap size is fixed at 500, and applying the multiplier z^* to construct prediction confidence interval, the coverage plot is shown as follow:

We observed that these confidence intervals for the whole dataset is highly correlated with outliers - when there is effect of outlier on both side of the line, the confidence interval is seemingly covered more for these outliers. However, overall, the percentage coverage for these models are close to the nominal level - α .

When comparing Infinitesimal Jackknife with Bootstrap, Jackknife, the overall coverage comparison is shown as follow:

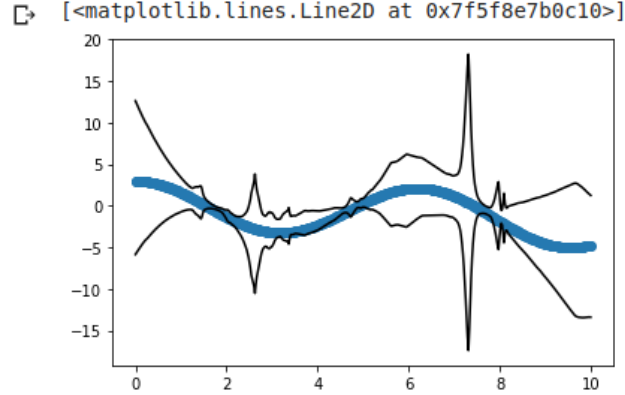


Figure 4.2: 95% Confidence Interval using Infinitesimal Jackknife Variance Estimation

	Infinitesimal Jackknife	Jackknife	Bootstrap Quantile
95% CI	96.2%	88.6%	93%

When comparing with these three methods, infinitesimal jackknife method is the closest to nominal level $1 - \alpha = 95\%$. We consider infinitesimal jackknife is more sensitive to nonlinear models. However, the IJ confidence interval is extremely sensitive to outliers, we need to introduce robust estimator to enhance confidence interval coverage.

4.4 Extended Research: Effect of Outlier

Since there will be a spike around our confidence interval around outlier's position (around $x=7$), I want to try several method to smooth the confidence interval excluding outlier's effect. I only present the confidence interval build by Infinitesimal Jackknife variance estimation for comparison.

4.4.1 Delete

If we arbitrarily delete the data seems to be the outlier($x=7.314$ precisely), then we will fit our data like the plot below:

Seems like our fitted line is more skewed to the left than not delete the outliers, seems like delete outlier is making our model fitting become worse.

4.4.2 Huber's M Estimator

If our loss function for regression is a sample average, then a class of extremum estimators is called M-estimator[171]. Generally, if $\hat{e}_i = y_i - \hat{y}_i$ represents residuals in model fitting, then we want to minimize $\sum \rho(\hat{e}_i)$ as a loss function where $\rho(\hat{e}_i)$ is a function of residuals.

The specific Huber M-estimator(Or Huber Loss)[172] is a compromise between e^2 and $|e|$. Although LAD(Least Absolute Deviations) estimates over LS(Least Square)

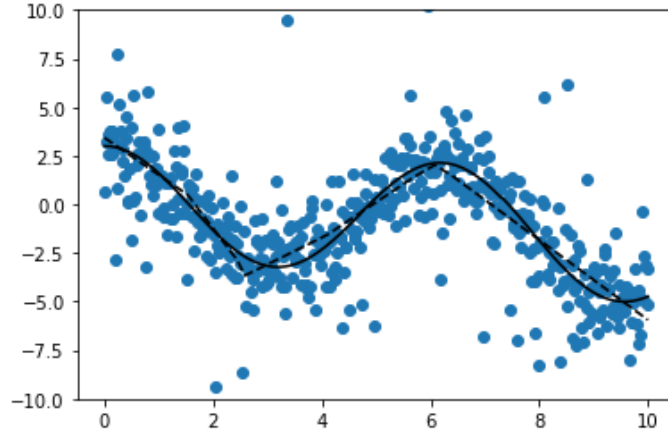


Figure 4.3: True data, true parametric line and fitted line(delete outlier)

estimates on not sensitive to outliers, but LS are more accurate to evaluate regression performance. Huber M-estimates is a method to combine advantage of both method and alleviate the effect of outliers. Huber's M Statistics is defined as:

$$\rho(e) = \begin{cases} e^2 & \text{if } -k \leq e \leq k \\ 2k|e| - k^2 & \text{if } e < -k \text{ or } k < e \end{cases} \quad (4.40)$$

Where $k = 1.5\hat{\sigma}$ and $\hat{\sigma}$ is the estimate of population standard deviation σ .

We actually choose $\hat{\sigma} = 1.483MAD[173]$ and MAD is the median of absolute deviation $|\hat{e}_i|$. The algorithm we are going to apply here is we fit the model iteratively to get residual from our result, then update k in each iteration to achieve a stable point in regression.

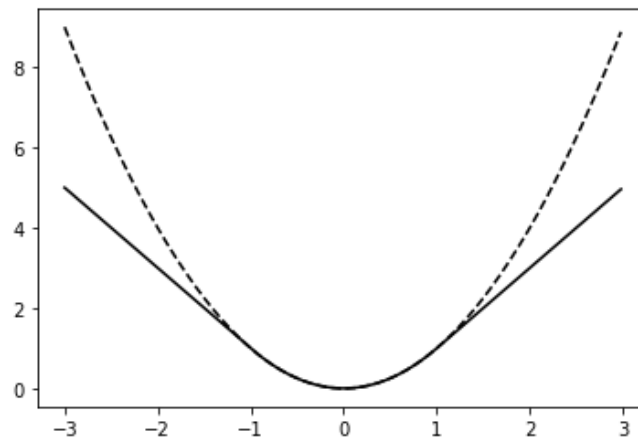


Figure 4.4: Huber's M Estimator for k=1

The Figure 4.4.2 represent a Huber's estimator behavior if residuals increment from -3 to 3. The solid line is what Huber's M Estimator behavior and dashed line

is just the least square estimates. We can see that by using Huber’s M statistic, the outlier’s effect has been alleviated outlier’s effect from second order power to linear.

After applying Huber’s M statistics with updated k in each iteration, we see how our neural network fit for the theoretical line in the Fig(5.2.1)

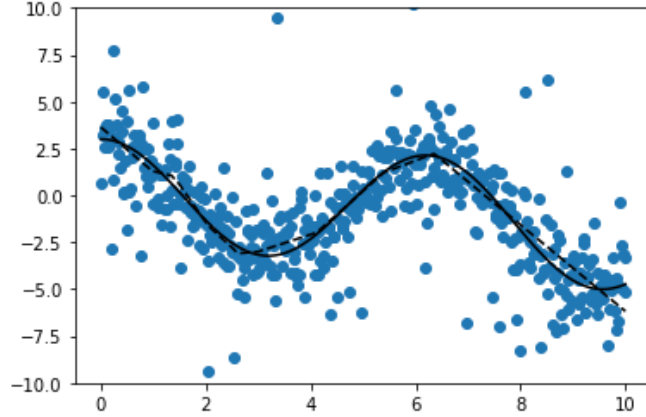


Figure 4.5: True data, true parametric line and fitted line(delete outlier)

We can see from the plot that the fitted line can depict true parametric line very well, except for some edge on the curvature point can’t depict the true line very smoothly.

4.4.3 Discussion

Method	Delete	Huber’s M
Coverage Rate	66.6%	93.4%

Table 4.1: Compare coverage rate of multiple smoothing method with 95% CI

We can combine those method and see coverage rate in the table below. We can see that, under the Huber’s M estimator, the percentage coverage for these models are close to the nominal level - α . If we only delete one outlier, it is hard to determine which is outlier and which is not. Hence, the smoothing method are theoretically better in coverage percentage than just ignore outlier.

4.5 Conclusion

I have explored an alternative approach to constructing confidence intervals for neural network regression. This approach involves using resampling methods to either calculate variance or construct a sampling distribution for neural network predictions, thereby enabling the construction of confidence intervals.

For resampling methods, various techniques can be employed, such as bootstrap, jackknife, and infinitesimal jackknife. While jackknife and bootstrap methods have previously been applied for variance estimation in neural networks, the infinitesimal jackknife is a novel addition. The infinitesimal jackknife method exhibits lower variance compared to the traditional jackknife method when the model fitting extent is the same, making it more sensitive to the coverage of the confidence interval.

After analyzing the impact of bootstrap size and considering that 200 bootstrap samples provide consistent and stable standard errors, a similar nonlinear model is fitted using a neural network. Among the different resampling methods, the infinitesimal jackknife method with the multiplier of the normal distribution achieves the closest nominal coverage to the neural network prediction, indicating its superiority in variance estimation.

Since the simulated model includes some observations that deviate significantly from the true model line, robust regression techniques were introduced to mitigate the effect of outliers on confidence interval construction. Comparing the approach of solely deleting outliers with the Huber's M estimator, which incorporates the Median Absolute Deviation (MAD) as a measure of variance, the latter exhibits much better coverage within the confidence interval setting. This demonstrates that instead of deleting outliers, down-weighting them is more appropriate when making predictions for the underlying data trend.

Based on my research findings, resampling methods provide more accurate uncertainty estimation for neural network regression compared to direct estimation. However, it is worth noting that both resampling and neural network approaches are time-consuming. Therefore, there is room for further improvements. For example, combining resampling methods with Bayesian estimation techniques (as cited in the referenced paper) can enhance the understanding of uncertainty estimation for neural networks. Additionally, ensemble methods, which involve combining multiple neural networks, have the potential to reduce model variance and may lead to better model fitting than a complex multi-layer perceptron model. Further analysis can be conducted to explore this avenue as well.

Chapter 5 Tolerance interval

In addition to the traditional setting of confidence intervals, we also aim to explore other types of statistical analysis in our research. Specifically, we will investigate the use of tolerance intervals in conjunction with neural networks as a nonparametric regression model. This research direction will focus on establishing bounds around the population distribution using a neural network as the predictor, rather than making point estimates or constructing confidence intervals.

5.1 Definition

In this part, we introduce the method of how to achieve tolerance interval for nonparameteric regression and its theory basis.

5.1.1 Nonparametric Regression Tolerance Intervals

Regression analysis is a statistical method used to estimate the relationship between a quantitative response variable and one or more explanatory variables. Nonparametric regression is a variation of this method that models the response variable as a function of the explanatory variable(s) without making assumptions about the shape of the relationship.

Nonparametric regression can be expressed as:

$$\mathbf{Y} = f(\mathbf{X}) + \epsilon \quad (5.1)$$

where $f(\cdot)$ represents the regression function and ϵ is a random error, typically assumed to be generated from a theoretical distribution. Unlike parametric regression, nonparametric regression does not assume constant variance of the error term ($Var(\epsilon_i) = \sigma^2(x_i)$).

Neural network is a complex, nonlinear regression method that can also be considered a nonparametric approach to analyzing regression models.

To determine if the fitted model, $\hat{f}(\cdot)$, provides a good and smooth fit to the data, there are many subjective methods that can be used. However, statistical interval estimation can be challenging for nonparametric regression because inherent bias in the method can result in higher error at peaks and lower error at valleys in the regression curve.

Tolerance intervals can be used to provide a nonparametric measure of interval estimation. Suppose a random sample $Z_1 = z_1, \dots, Z_n = z_n$ comes from a distribution function F_Z , the upper and lower tolerance bounds of $[100 \times (1 - \alpha)\%]/[100 \times P\%]$ interval are $L = z_{(r)}$ and $U = z_{(n-r+1)}$, where $z_{(j)}$ represents the j th value of the

ordered statistics of the random samples and $r < (n - 1)/2$. The coverage probability of the tolerance region $[Z_{(r)}, Z_{(n-r+1)}]$ is beta distributed with shape parameters $n - 2r + 1$ and $2r$. For a two-sided tolerance interval, r is determined by minimizing the probability that the beta distribution is greater than or equal to P :

$$\arg \min_{r:r < (n+1)/2} Pr(Beta \geq P) \geq 1 - \alpha \quad (5.2)$$

and

$$\sum_{i=0}^{n-r} \binom{n}{i} P^i (1 - P)^{(n-i)} \geq 1 - \alpha \quad (5.3)$$

An alternative method for achieving $[100 \times (1 - \alpha)\%]/[100 \times P\%]$ regression tolerance bounds is to use the order statistics of all residuals, i.e., $e_{(1)}, \dots, e_{(n)}$. For each prediction point in the regression, the tolerance bounds can be calculated as:

$$L = \hat{y}_j + e_{(r)} \quad (5.4)$$

and

$$U = \hat{y}_j + e_{(n-r+1)} \quad (5.5)$$

5.2 Experiment

In the first and second chapters, we explored the definition of tolerance intervals and their applications to various statistical models. In this chapter, we will investigate the behavior of tolerance intervals when applied to a nonparametric model, specifically a neural network.

5.2.1 Nonlinear model tolerance interval

We are still applying the previous simulation function

$$y_i = 3\cos(x_i) - 5(x_i/15)^2 + \epsilon_i \quad (5.6)$$

where ϵ_i obeys t distribution with 2 degrees of freedom.

We define our x variable from the interval $[0,10]$ with increment of 0.02 and want to see the coverage probability for each point in x axis related to nominal confidence level.

Based on the coverage probability plot 5.2.1, we can observe that when x is in the range of $[0.5, 5]$ and $[7, 9.5]$, the coverage probability exceeds the nominal level. However, similar to the findings in [174], when x has a large value, the coverage probability appears to be low. In our analysis, we observed that the decay of coverage probability is much faster than that reported in [174] when x becomes larger in our experiment.

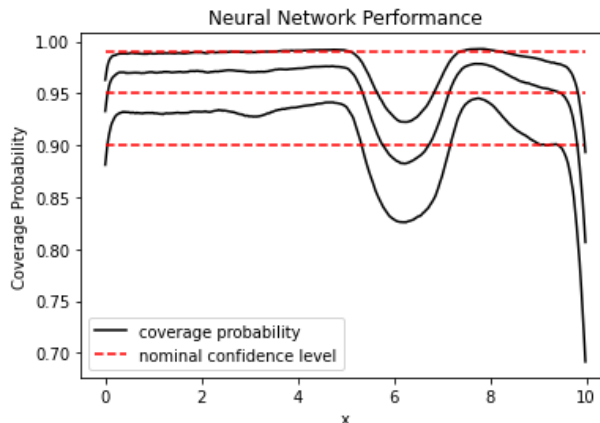


Figure 5.1: The coverage probability at $n=500$ sample size of unique x value for 90×95 , 95×95 and 99×95 tolerance intervals.

$1 - \alpha$	n	P		
		0.9	0.95	0.99
0.90	50	0.9477496	0.7948052	0.141384
	500	0.8893424	0.9084754	0.95710896
0.95	50	0.9477496	0.7948052	0.95710896
	500	0.91052844	0.95447828	0.95710896
0.99	50	0.9477496	0.7948052	0.141384
	500	0.95003264	0.97866392	0.95710896

Table 5.1: Compare coverage rate of multiple smoothing method with 95% CI

Additionally, we noticed an important factor that affects the performance of the neural network, which is the variance estimation in outliers when x is around the outliers. The coverage probability drops into a convex curve around $x=6$, whereas the effect of outliers on [174]’s paper is not significant.

Table (5.1) presents the estimated coverage from our simulations. It is apparent that when the sample size is small, the coverage probability is significantly below the nominal level ($1-\alpha$). This occurs because when the sample size is small, a small increment in the order statistics i has a large effect on the cumulative probability of a Beta distribution, leading to an underestimation of the true coverage probability in the case of small sample sizes. The required sample size has been studied by [175], who specified $n \geq f(\alpha, P)$, where f is a specified distribution.

We observe that as the proportion of the population increases, the coverage probability also increases when the sample size and confidence level are fixed, while the confidence level determines the nominal level of coverage. Moreover, as the sample size becomes larger, the coverage probability becomes closer to the nominal level.

However, when compared with the results from [174], the coverage probability for neural network is lower than that for LOESS[176].

There still exist some issue in fitting neural network to achieve tolerance interval for model, here we do several extension works on tolerance interval to see neural network’s performance.

5.2.2 Experiment using Linear Model

We aim to examine whether a linear curve with normal error, fitted using a neural network, can achieve similar coverage to the nominal level without being affected by outliers. This is because our theoretical curve is non-linear and has non-normal distribution errors.

In the experiment, the following formula is firstly used:

$$y = 3x + \epsilon \tag{5.7}$$

and X belongs to the interval $[0, 10]$ and with equal increment and of size 500. ϵ is a random variable belongs to a standard normal distribution.

After fitting the 2-layer neural network with first layer size 3 and second layer size 29, we can see the fitting of the linear curve in the solid black line in Figure(5.2).

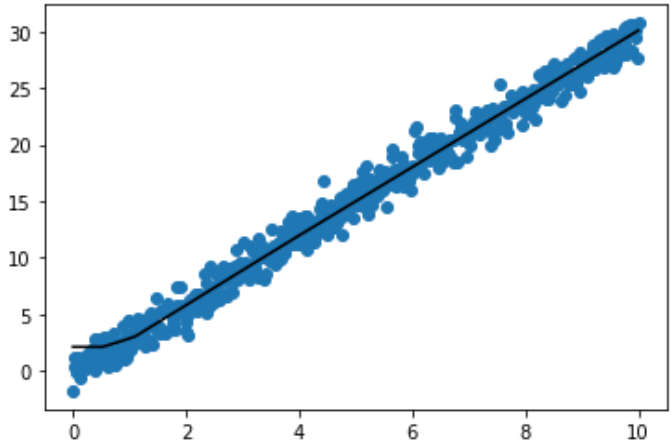


Figure 5.2: Fitting linear curve with neural network.

Upon examining the fitted curve, we observe that it is curvilinear for x in the range of $[0,1]$, but becomes linear after $x > 1$. Next, we investigate the performance of different values of $P = 0.95$ and $\alpha = 0.1, 0.05, 0.01$ by examining the distribution of coverage for y values at each x in the interval $[0,10]$ in Figure (5.3).

The coverage plot above reveals that the coverage percentage drops drastically to a very low level when x is less than 1 (this may be due to the vanishing gradient issue).

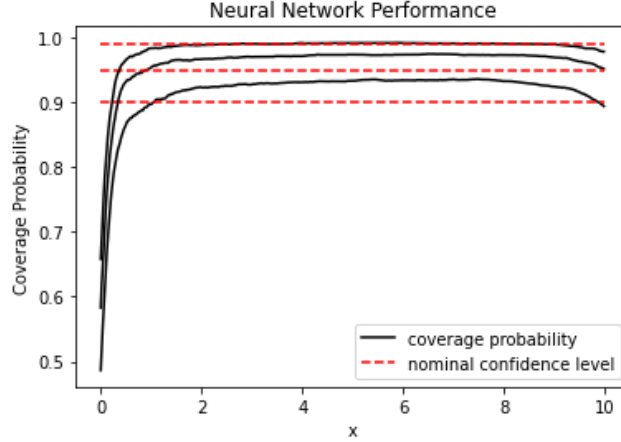


Figure 5.3: The coverage probability at $n=500$ sample size of unique x value for 90×95 , 95×95 and 99×95 tolerance intervals, fitting linear model with neural network.

However, as x increases, the coverage tends to become closer to the nominal level, or even exceed it. This may be because normal random errors are less likely to create outliers, and fitting a complicated neural network on a simple linear function tends to have a large effect on certain parts if there is overfitting. Therefore, our robustness issue of tolerance interval is resolved. However, if the neural network cannot fit the true curve well, there may still be some effect on certain parts of the model.

5.2.3 MAD as Deviation Estimator

In Section(3.1.2) we discussed the method of using MAD with sliding window as an estimator for standard deviation, i.e., $\hat{\sigma} = 1.483 \text{ MAD}$. We now introduce the method of using variance with normal assumption to build tolerance interval, and see the performance of neural networks.

When using normal assumptions on variance, the bounds for a $[100 \times (1-\alpha)\%]/[100 \times P\%]$ two-side regression tolerance interval will be expressed in the form as

$$L = \hat{y}_j - k * s \quad (5.8)$$

And

$$R = \hat{y}_j + k * s \quad (5.9)$$

We might want to assume s/s_j for each j is approximately 1, then we can use $\hat{\sigma}$ at each point y_j as estimator of s_j . We also use [154] to estimate k -factor in our tolerance interval. Let $f=n-p$, where n is how many cases in one sample, and p is how many features in your data. We will have

$$k = \sqrt{\frac{f \chi_{1;P}^2(1/n_h^*)}{\chi_{f,\alpha}^2}} \quad (5.10)$$

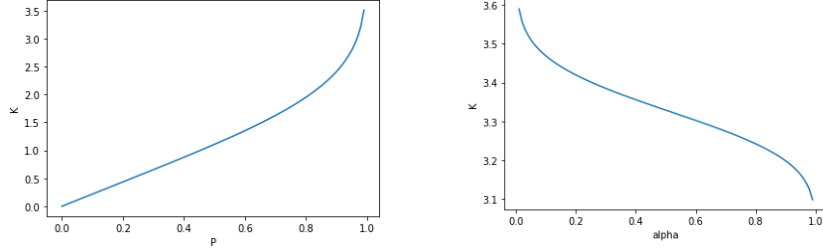


Figure 5.4: Left figure describe for fixed confidence limit α , the behavior of K versus coverage percent P; Right figure talks about for fixed coverage percent P, the behavior of K versus confidence limit α .

where $\chi_{c,\alpha}^2(\delta)$ will be the the $\alpha\%$ quantile of a noncentral chi-square distribution with c degrees of freedom and noncentrality parameter δ .

Figure(5.4) describe for a fixed $\alpha = 0.05$, the multiplier k's behaviour on coverage probability P; It is observed that for a fixed P=0.95, the multiplier k's behaviour on confidence level α :

When α is fixed, an increase in coverage probability P leads to a larger value of K. Additionally, as P approaches 1, the derivative of P with respect to K becomes higher. When P is held fixed, K exhibits smoother decrements for α values in the middle of [0.2,0.8], while for α values at the edges (0 to 0.2 or 0.8 to 1), the decrements of K become faster. Generally, P is the primary factor influencing the composite values of K, while α is the secondary factor.

To simplify the equations, we assume $n_h^* = 1$ since there are too many parameters in the model. Equation (5.6) is still used with the error term following a standard normal distribution. The 95×95 tolerance interval for the simulated data can be seen in Figure (5.5).

From the tolerance interval figure we can see that tolerance interval can capture most the data under normal distribution error, but not overconfidence. In fact, for 95×90 , 95×95 and 95×99 tolerance interval coverage on different x compare to nominal level, while we see that in fig(5.6) for each coverage line, there exist little difference. This is perhaps for different choice of coverage percent P in the set $\{90\%, 95\%, 99\%\}$, the multiplier K has little difference.

5.3 Discussion

This chapter explores the use of "tolerance interval" in the context of neural network models. We provide an overview of the background, development, and applicability of tolerance intervals in neural networks. We apply three methods to construct tolerance intervals: the quantile method, the variance estimation method

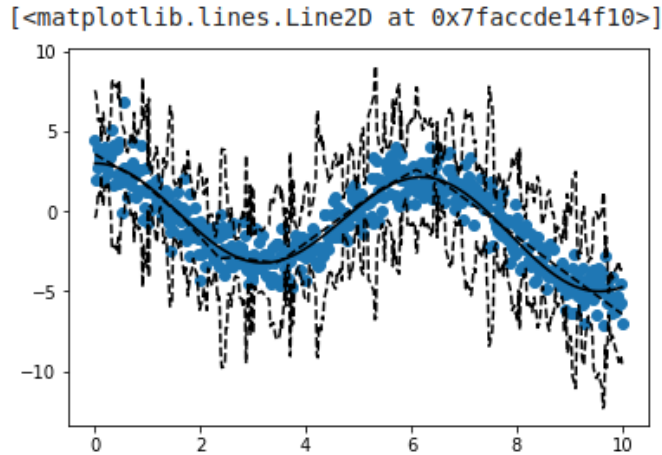


Figure 5.5: The coverage probability at $n=500$ sample size 95×95 tolerance intervals, fitting linear model with MAD as variance estimation on neural network.

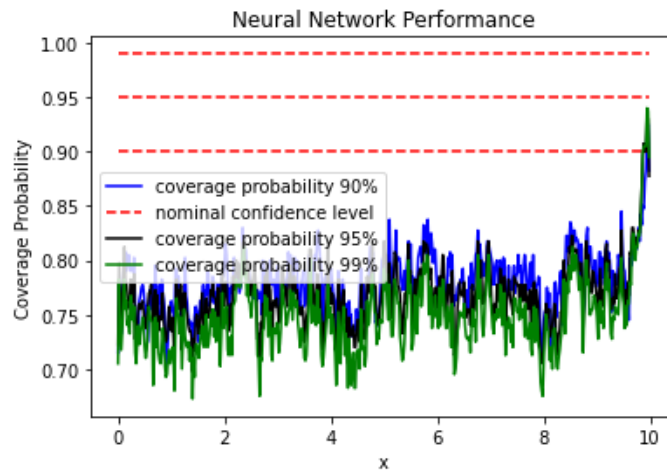


Figure 5.6: The coverage probability at $n=500$ sample size of unique x value for 90×95 , 95×95 and 99×95 tolerance intervals, fitting nonlinear model with neural network.

with normal simulation, and a method similar to the R package tolerance ([177]). To implement these methods, we utilize the Keras API in Python, as neural networks can be challenging to work with in the R environment.

To compare the advantages and disadvantages of using neural networks in building tolerance intervals, we applied a nonparametric regression tolerance interval method as discussed in [174]. We compared our results with those of [174] and found that the neural network model showed similar coverage probabilities as the traditional nonparametric model (LOESS) when residual and order statistics of the neural network were used to build the tolerance interval.

Furthermore, to establish a baseline model for the neural network, we used the tolerance interval building method for linear regression when fitting the neural network model to data under a linear regression setting. As discussed in Chapter 7, the prediction power of neural networks is almost the same as linear regression when the underlying model is linear. Thus, we expect that neural networks will show similar or better coverage performance than linear regression. We found that the nonparametric tolerance interval coverage ability is similar to that of the linear regression tolerance interval under statistical inference.

We also attempted to use the MAD method to approximate the variance and build the tolerance interval under this setting. However, we found that the coverage performance was not as good as the nonparametric regression method. Nevertheless, the theoretical tolerance interval setting under the neural network model can be further explored.

Chapter 6 Neural Network's Effect Comparing With Linear and Logistic Regression with misspecification

In Section 1.1, we mentioned that neural networks are essentially recursive generalized linear models, built by stacking GLMs in horizontal and vertical ways. In this section, we will delve into the relationship between neural networks and linear/logistic regression models, specifically for linear structures. We will explore how model fitting performance is affected when misspecification occurs.

6.1 Basic Concept

Here, we will introduce the concepts of linear regression and logistic regression, which are two common special cases in generalized linear models. Additionally, we will discuss the concept of misspecification in statistical models.

6.1.1 Linear Regression

In the context of generalized linear models, linear regression assumes that the outcome variable is normally distributed and the link function is the identity function. The basic model structure can be expressed as:

$$E(Y) = X\beta \quad (6.1)$$

Based on the property that if a new variable Y is constructed by adding a constant $X\beta$ to a normally distributed variable ϵ , the distribution of Y is still normal. Therefore, linear regression is traditionally expressed in the following form:

$$Y = X\beta + \epsilon \quad (6.2)$$

where $\epsilon \sim N(0, \sigma^2)$, representing a normal distribution with 0 mean and σ as the standard deviation.

When discussing model assumptions, two specific requirements are designed for linear regression. The first is the assumption of homogeneity of variance. Additionally, the loss function of linear regression is the sum of squared error. To minimize this error, we can differentiate the loss function to obtain:

$$Loss = (Y - X\beta)'(Y - X\beta) \frac{\partial L}{\partial \beta} = \beta' X' X \beta - 2X\beta Y \quad (6.3)$$

Since the sum of square loss is a convex function, the $\hat{\beta}$ that makes the equation equal to 0 will be the parameter that solves linear regression. Thus, we have:

$$\hat{\beta} = (X'X)^{-1}X'Y \quad (6.4)$$

Furthermore, we know that $E(MSE) = \sigma^2$, since

$$SSE = (Y - X\hat{\beta})^T(Y - X\hat{\beta})$$

P is the projection matrix on the column space of X. The fact is that $PY = X\hat{\beta}$. Since $Y \sim N(X\beta, \sigma^2 I_n)$, and

$$\frac{SSE}{\sigma^2} = \frac{Y^T(I - P)Y}{\sigma^2} \sim \chi_{(n-k)}^2$$

because I-P is the projection matrix of rank n-k.

When Mean Squared Error (MSE) is used as a performance metric, it is considered the best (minimum variance) unbiased estimator for linear model structures[178]. Therefore, MSE is often used as a performance metric to evaluate how well an artificial neural network can learn from a linear model.

6.1.2 Logistic Regression

Under GLM structure, when the outcome is binary, the target variable is assumed to be Bernoulli distributed and link function will be logit function. The basic model structure will follow the form:

$$\text{logit}(Y) = X\beta$$

Where logit function is $\text{logit}(Y) = \log\left(\frac{\text{Pr}(Y=1)}{\text{Pr}(Y=0)}\right)$, which is the log of the odds of this case belongs to class "1".

For maximum likelihood estimate of β , we want to maximize the likelihood

$$L(\beta) = \prod_{s \text{ in } y_i=1} p(x_i) \times \prod_{s \text{ in } y_i=0} (1 - p(x_i))$$

where s is each case, when we take the log of this likelihood, we have

$$l(\beta) = \sum_{i=1}^n (y_i \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i)))$$

The cross-entropy loss is obtained by taking the negative logarithm of the likelihood above. In order to minimize this loss, an optimization method such as Newton's method is typically used to find the local minimum. However, due to the inclusion of probability P in the Hessian matrix, there is no closed-form solution for logistic regression to achieve the global minimum, as explained in detail in Section (9.4).

In the case of multiple outcomes in classification, a baseline outcome is typically specified, and the probabilities of the other classes are predicted in parallel. The variable is then classified into the group with the highest predicted probability. This approach is often used in logistic regression.

6.1.3 Misspecification

Statistical models are defined as a set of data drawn from a specific type of distribution. This distribution is referred to as the statistical model. When misspecification is introduced in linear models [179], it means that the true linear model is specified as follows:

$$f(y_i) = \beta_0 + \beta_1 x_i + \beta_2 q_i + \epsilon_i$$

and we have the relationship of x_i with q_i : $x_i = g(q_i) + u_i$ where u_i is some unique variance inside x .

A linear model is called misspecified if we build following model if true model is specified in the function of $f(y_i)$:

$$y_i = \hat{\pi}_0 + \hat{\pi}_1 x_i + \hat{v}_i$$

Where \hat{v} represent the residual of the misspecified model, since actual relationship of q_i is actually omitted in the wrongly presented model.

This chapter primarily focuses on the misspecification of the interaction term in linear models, where the true model includes an interaction term, but the misspecified model does not. We will compare the performance of generalized linear models and neural networks in this context.

6.2 Experiment

6.2.1 Compare linear model for NN and OLS for different sample size

To begin, we will compare linear regression with a neural network that has 3 layers, uses the Adam optimizer, ReLU activation functions in the hidden layers, and a linear function in the output layer. We will use a theoretical sample with sizes of $n=75, 150,$ and 500 , defined as follows:

$$y = 3x + \epsilon \tag{6.5}$$

where x is within the range of $[0, 10]$ and ϵ is drawn from a normal distribution with mean 0 and standard deviation 1. Since linear regression with the OLS (Ordinary Least Squares) estimator is the BLUE (Best Linear Unbiased Estimator) for the parameter b (in this case, b equals 3), it can learn the linear model best with the formula $\beta = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$. By using 67% of the sample as the training data and 33% of the sample as the testing data, we can evaluate the performance of MLE on the test set, which is:

method	training size	LR	NN
MSE	50	0.9241	1.0127
MSE	100	0.8667	0.8790
MSE	333	0.9829	0.9836

Next, we will extend the dimension of the data from 1 to 5, where x_1 is generated from a Uniform(0,10) distribution, x_2 is generated from a Uniform(5,15) distribution, x_3 is generated from a Uniform(10,20) distribution, x_4 is generated from a Uniform(15,25) distribution, and x_5 is generated from a Uniform(20,30) distribution. We will create another theoretical sample with sizes of $n=75, 150, \text{ and } 500$, defined as follows:

$$y = x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 + \epsilon \quad (6.6)$$

where ϵ is drawn from a normal distribution with mean 0 and standard deviation 1. In this case, we fit the OLS model and a neural network with 500 iterations to this sample, and we obtain:

method	training size	LR	NN
MSE	50	0.9518	27.6432
MSE	100	0.8800	16.5946
MSE	333	0.9842	2.2070

We can observe that linear regression performs much better than the neural network in the linear simulation models, especially when the sample size is small. Additionally, linear regression can learn the variance very well even when the sample size is small, while the neural network tends to underfit in this situation. However, as the sample size increases to around 10^3 , the neural network with 3 layers exhibits a similar effect on linear model fitting as linear regression.

6.2.2 Compare linear model misspecification with interaction using NN and OLS method for different sample size

Next we developed some interaction model to see NN's effect on misspecification on linear models, the theoretical data looks like:

$$y = x_1 + 2 * x_2 + 3 * x_3 + 4 * x_4 + 5 * x_5 + 2.5 * x_1x_2 + 4.5 * x_3x_4 + \epsilon \quad (6.7)$$

We are trying to compare LR and NN's effect on reduced model, that is, only fit models on independent variables with no interaction term, then test those only independent included variable's prediction performance on the test set. Linear Regression still seems to outfit neural network, by setting test set 25, 50, 167, 1677, the MSE on test set be:

method	training size	LR	NN
MSE	50	46.2052	255.9778
MSE	100	49.5002	225.3776
MSE	333	44.8011	140.2441
MSE	3333	43.1964	43.5922

MSE on Neural network is not better than linear regression on linear model with misspecification.

While fitting the reduced model on training data (without test set), when sample size becomes larger, we have the performance of NN better than LR in the measurement of MSE.

method	training size	LR	NN
MSE	75	44.5415	264.4734
MSE	500	42.4133	105.2864
MSE	1000	42.6387	45.1027
MSE	5000	42.8269	33.3050

Notice that batch size is important for converge, I adjusted batch size to 1.

To see the effect more clearly, we make the individual effect of interaction variables insignificant, and have a significant slope for interaction of variables, i.e., in the formula below, we generate $n = 75, 500, 1000, 5000, 10000$ random variables, and only fit LR and NN model on $x_1, x_2, x_3, x_4,$ and x_5 but no interaction term included:

$$y = x_1 + 2 * x_2 + 3 * x_3 + 0 * x_4 + 0 * x_5 + 2.5 * x_4x_5 + \epsilon \quad (6.8)$$

I changed $\epsilon \sim N(0,10)$ to see further variance reduction, as model performance below, after sample size increase, the loss of NN is getting smaller and better than linear regression's behavior.

method	training size	LR	NN
MSE	75	24.2514	144.2779
MSE	500	21.9112	80.9711
MSE	1000	23.3927	25.0894
MSE	5000	23.1192	23.5615
MSE	10000	23.3390	20.9650

When sample size becomes larger and larger, the fitting of NN is smaller and finally better than linear regression.

6.2.3 Compare linear model misspecification with interaction using NN and OLS method for different sample error

Next we see the effect of change the variance of error to $\sigma = \{50, 25, 12.5, 5\}$ and fix sample size to 1000, while the following model is used to create samples that amplify the interaction effect:

$$y = x_1 + 2 * x_2 + 0 * x_3 + 0 * x_4 + 3 * x_5 + 50(x_3x_4) + \epsilon \quad (6.9)$$

The performance of neural network and linear regression using reduced model (without interaction term) is shown below:

method	σ size	LR	NN
MSE	50	412.6013	261.0349
MSE	25	410.9903	242.5660
MSE	12.5	410.7536	233.9843
MSE	5	410.7942	237.1565

MSE on NN outperforms LR in this specific setting.

We then create a dataset based on $x_1, x_2, x_3, x_4 * x_5$ in the formula below and use sample size as 1000,

$$y = x_1 + 2 * x_2 + 3 * x_3 + 0 * x_4 + 0 * x_5 + 25 * \log(x_4 x_5) + \epsilon \quad (6.10)$$

where $x_1 \sim U[0, 10]$, $x_2 \sim U[2, 12]$, $x_3 \sim U[4, 14]$, $x_4 \sim U[6, 16]$, $x_5 \sim U[8, 10]$, and afterwards, we fit simulated dataset on reduced model (only single variable x_1 through x_5 without interaction term), the performance is like

method	σ size	LR	NN
MSE	50	50.01	50.00
MSE	25	25.00	26.19
MSE	12.5	12.50	16.11
MSE	5	5.01	5.23

But after creation of dataset based on $x_5, x_1 * x_2, x_3 * x_4$ in the formula below and use sample size as 1000,

$$y = 0 * x_1 + 0 * x_2 + 0 * x_3 + 0 * x_4 + 2 * x_5 + 3(x_1 x_2) + 5(x_3 x_4) + \epsilon \quad (6.11)$$

where $x_1 \sim U[0, 10]$, $x_2 \sim U[2, 12]$, $x_3 \sim U[4, 14]$, $x_4 \sim U[6, 16]$, $x_5 \sim U[8, 10]$ and afterwards, we fit simulated dataset on reduced model (only single variable x_1 through x_5 without interaction term), the performance is like

method	σ size	LR(Reduced)	NN(Reduced)	LR(Full)	NN(Full)
MSE	50	69.0035	60.0925	49.9854	52.6746
MSE	25	54.0656	44.7310	24.9927	27.3412
MSE	12.5	49.7273	39.9236	12.4963	14.9813
MSE	5	48.5045	38.3074	4.9985	5.7446

When it comes to misspecification of both interaction and missing term, NN can detect better MSE than linear regression when error is large enough, the data is built using the following formula:

$$y = 1.5 * x_1 + 2.5 * x_2 - 2 * x_3 - 3 * x_4 + 3(x_1 x_2) + 5(x_3 x_4) - 3x_5 + \epsilon \quad (6.12)$$

The sample size is 1000, I used 67% of the data as training 33% of the data as testing. We have $x_1 \sim U[0, 10]$, $x_2 \sim U[2, 12]$, $x_3 \sim U[4, 14]$, $x_4 \sim U[6, 16]$, $x_5 \sim U[8, 10]$ afterward, we fit simulated dataset on reduced model (only single variable x_1 through x_4 without x_5 and interaction term), the performance is like

method	σ size	LR(Reduced)	NN(Reduced)
MSE	50	105.5080	95.7167
MSE	25	61.4098	58.8847
MSE	12.5	46.1043	45.2764
MSE	5	42.3015	40.0126

Overall, NN has better fitting then linear regression when misspecification include both interaction term and missing term, we can also observe that when error term is large in variance, NN has better fitting in test set than linear regression.

6.2.4 Compare NN and Logistic Regression for categorical classification

Let's begin by discussing binary outcomes. Simulating logistic regression outcomes can be challenging because it requires using a linear model to obtain the logarithm of the odds, followed by an inverse transformation to convert the probability into a binary class.

We need to look at the inverse of the logit function - sigmoid function. Sigmoid function has been provided below:

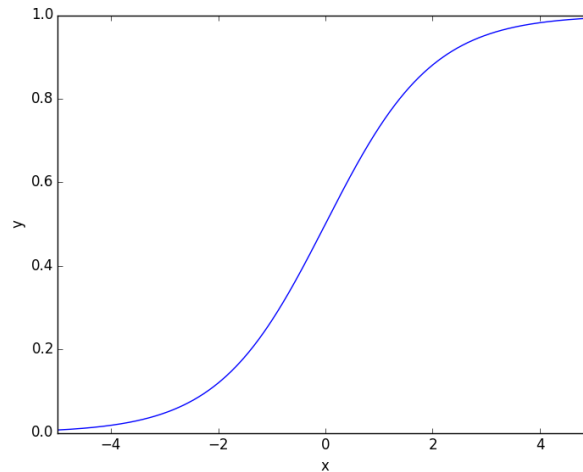


Figure 6.1: Sigmoid Function

The simulation process is shown as follow:

- Make a certain X and β , then fit $X\beta$ as $\text{logit}(y)$.
- Transfer $\text{logit}(y)$ into probability (p) using sigmoid function ($\sigma(x) = \frac{1}{1+e^{-x}}$).
- For case of $X\beta$, generate each target variable based on a Bernoulli distribution with probability p .

To ensure the successful fitting of logistic regression, it is important to simulate the range of $X\beta$ within the interval of $[-5, 5]$. Additionally, it is necessary to have at least 5% of the total data resulting in $X\beta$ being less than 0. Failing to meet these conditions could lead to an imbalance where the target variable is either all "1" or all "0," rendering the logistic regression unfit for analysis.

We have implemented a new evaluation metric, the Receiver Operating Characteristic (ROC) curve and the corresponding Area Under the Curve (AUC) score, to assess the performance of our classification model. The ROC curve is a graphical representation of the model's ability to differentiate between the classes.

The ROC curve is generated by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR). TPR is the ratio of true positives to the sum of true positives and false negatives, while FPR is the ratio of false positives to the sum of true negatives and false positives. By varying the decision threshold for classifying probabilities from 0 to 1 and applying it to the predicted output, we can plot the TPR against the FPR, resulting in the ROC curve. The AUC score represents the total area under this curve.

Interpreting the ROC/AUC score, a higher AUC indicates that the model has a greater ability to correctly predict instances of class 0 as 0 and class 1 as 1. Essentially, a higher AUC suggests that the model exhibits improved separability between the classes.

To illustrate, please refer to Figure (6.2), which showcases an example of an ROC/AUC curve.

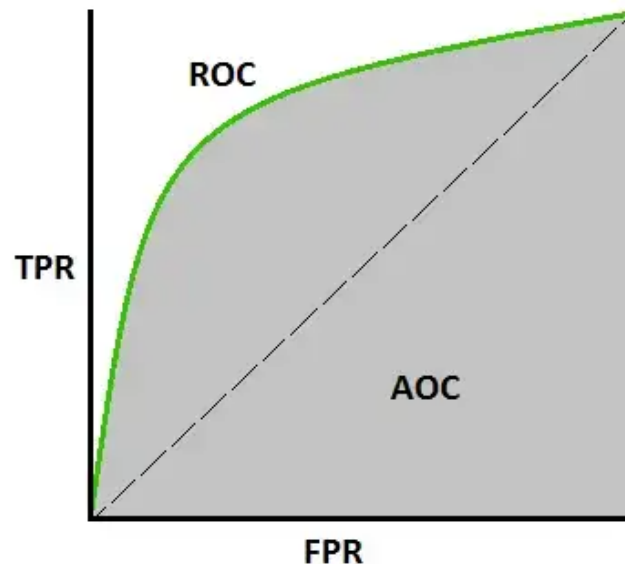


Figure 6.2: ROC curve example

I simulated a logistic regression model by simulating logit:

$$\text{logit}(y) = -9 + 3.5x_1 + 0.2 * x_2$$

For each probability in p , y represents a random sample obtained from a binomial distribution. We conducted a logistic regression analysis with the formula $y \sim x_1 + x_2$, where x_1 and x_2 are the predictor variables. When we increased the sample size from 100 to 1000, the coefficients estimated by the logistic regression model approached the true values of $[0.2, 3.5]$ more closely.

Furthermore, we evaluated the performance of the logistic regression model and a neural network (NN) by calculating their ROC scores. The logistic regression model achieved a ROC score of 0.9578, indicating a high level of predictive accuracy. Similarly, the neural network attained a ROC score of 0.9574, demonstrating comparable performance to the logistic regression model.

When I introduced an interaction term

$$\text{logit}(y) = -9 + 3.5x_1 + 0.2 * x_2 + 1.5x_1 * x_2$$

then if fit logistic regression and NN only on reduced model, ROC scores are 0.9730 for logistic regression and 0.9851 for neural network. This seems to prove that neural network can't perform better than logistic regression.

When more complicated model is introduced, we used bootstrap to construct 95% bootstrap intervals for the ROC-AUC scores for both logistic regression and neural network method. The model is expressed in the following form:

$$\text{logit}(y) = 2 + 0 * x_1 + 0 * x_2 - 0.2 * x_3 - 0.3 * x_4 + 0 * x_2 * x_1 + 0 * x_3 * x_4$$

Where x_1 is a random choice from 0 and 1, x_2 is drawn from a normal distribution with 0 mean 1 variance, x_3 and x_4 are drawn from uniform $[0,10]$ distribution and uniform $[10,20]$ distribution.

When only $\{x_1, x_2, x_3, x_4\}$ included in the model, the 95% confidence interval for logistic regression ROC-AUC score is $[0.7088, 0.8324]$, while the same level confidence interval for 2-layer NN ROC-AUC score is $[0.7080, 0.8326]$, which means neural network are similar confident to logistic regression when model is correctly specified.

Then we tried to include interaction term in the following model:

$$\text{logit}(y) = 2 + 0 * x_1 + 0 * x_2 - 0.2 * x_3 - 0.3 * x_4 + 3 * x_2 * x_1 + 0.05 * x_3 * x_4$$

When model is misspecified, i.e., only $\{x_1, x_2, x_3, x_4\}$ are in the model, we have logistic regression 95% confidence interval on the test set as $[0.7741, 0.8758]$, while the neural network 95% confidence interval on the test set as $[0.7730, 0.8749]$. The difference is not significant at all.

6.2.5 True Dataset

California Housing data[180]

The California housing dataset encompasses data gathered from all block groups in California during the 1990 Census. Geographical area significantly influences population density, thus information regarding distances between centroids of each block group is included, measured in terms of latitude and longitude. Block groups that lack entries for either independent or dependent variables have been excluded from the dataset.

The California Housing dataset comprises of 10 variables, encompassing 8 quantitative variables (longitude of houses, latitude of houses, median house age, total rooms, total bedrooms, population in the area, households, median income, median house value), and 1 categorical variable (ocean proximity) with 5 distinct classes.

There are 207 missing values for 'total bedrooms', I used median imputer for this variable; Then I applied label encoder to the categorical variable with 5 classes, which converts these string classes into $\{0, 1, 2, 3, 4\}$, five classes. In order to achieve the same scale for all predictors, I applied a standard scaler, which is

$$z = \frac{x - u}{s} \tag{6.13}$$

where u is the mean of this variable, and s is the standard deviation of this variable with $(n-1)$ as denominator. n is how many cases we have in this dataset. Then I applied a 80/20 train-test split on the whole dataset, fit models on the 80% training data and find performance on those 20% testing data.

Those models I used for model fitting is using linear regression as base model, and a 2-layer neural network, with 64 neurons in each layer and relu as active function in each layer, using Adam optimizer as optimization method and minimize mse as loss function. A patience of 10 has been applied as early stopping criteria.

The comparison of linear regression with and without interaction term is shown below, neural network is been used as another modeling method:

RMSE (on test)	Reduced Model	Full Model (with 2nd order interaction)	Full Model (with 3rd order interaction)
Linear Regression	71447.87	67932.09	112478.66
2-layer NN	67261.91	63560.67	62637.14

Upon analysis, it becomes evident that the model exhibits misspecification concerning the second-order term. The full model demonstrates superior evaluation metrics when tested. Comparatively, the neural network outperforms linear regression in terms of fitting, even after incorporating the interaction term.

Introducing a third-order interaction term into the full model reveals that linear regression tends to overfit, as evidenced by an increase in RMSE on the test set. However, the neural network retains the ability to capture certain patterns within the dataset. Thus, we can deduce that the neural network possesses greater capacity than the linear regression model, particularly when dealing with high-dimensional data.

When we talk about dimension reduction, we first want to see how neural network's fitting performance compared with linear regression if we remove some insignificant variables from reduced model. I first achieved model significance on those variables:

```

=====
                        OLS Regression Results
=====
Dep. Variable:    median_house_value    R-squared:    0.640
Model:           OLS                    Adj. R-squared: 0.640
Method:         Least Squares          F-statistic:   3261.
Date:           Sat, 05 Nov 2022       Prob (F-statistic): 0.00
Time:           03:25:37               Log-Likelihood: -2.0749e+05
No. Observations: 16512                AIC:           4.150e+05
Df Residuals:   16502                 BIC:           4.151e+05
Df Model:       9
Covariance Type: nonrobust
=====
                    coef    std err          t      P>|t|    [0.025    0.975]
-----
const             2.072e+05    539.942     383.735    0.000    2.06e+05    2.08e+05
x1                -8.585e+04    1654.029    -51.907    0.000    -8.91e+04    -8.26e+04
x2               -9.095e+04    1632.840    -55.698    0.000    -9.41e+04    -8.77e+04
x3                1.492e+04     605.976     24.629    0.000     1.37e+04     1.61e+04
x4               -1.769e+04    1938.780     -9.126    0.000    -2.15e+04    -1.39e+04
x5                4.877e+04    3254.220     14.986    0.000     4.24e+04     5.51e+04
x6               -4.388e+04    1368.521    -32.067    0.000    -4.66e+04    -4.12e+04
x7                1.76e+04     3232.476     5.445    0.000     1.13e+04     2.39e+04
x8                7.714e+04     715.405    107.833    0.000     7.57e+04     7.85e+04
x9               -451.5202     580.846     -0.777    0.437    -1590.041     687.001
=====
Omnibus:         4032.853    Durbin-Watson:    1.965
Prob(Omnibus):   0.000    Jarque-Bera (JB): 16157.977
Skew:            1.163    Prob(JB):         0.00
Kurtosis:        7.251    Cond. No.         16.3
=====

```

Figure 6.3: Model Significance

Variables x_1, \dots, x_9 represent ['longitude', 'latitude', 'housing_median_age', 'total_rooms', 'total_bedrooms', 'population', 'households', 'median_income', 'median_house_value', 'ocean_proximity']. When variables are reduced based on p-value, the smaller the t-statistics, the earlier variable will be excluded.

When considering only the 5 most significant variables, the Neural Network achieves an RMSE of 76133 on the test set, whereas linear regression yields an RMSE of 124421.84. On the other hand, when focusing solely on the 3 most significant variables, the Neural Network achieves an RMSE of 72688.58, while linear regression produces an RMSE of 1404194.86. These results suggest that the Neural Network exhibits more consistent performance on the test set when the number of variables is reduced and the model is underfitting.

Outliers persist within the model, and the distribution of the output exhibits right-skewness. Housing prices were truncated at 500,001 if they exceeded this threshold.

Moreover, multiple high-priced houses are present in California, with a light and non-normal distribution. To address these concerns, two remedial actions were implemented. Firstly, outliers were deleted, and secondly, a log transformation was applied to those outliers. The performance of the two models is outlined below:

RMSE (on test)	Delete outliers	log transform target variable
Linear Regression	62934.35	0.3501
2-layer NN	69884.50	0.3315

We noticed that by removing outliers, linear regression demonstrates superior model fitting performance compared to a 2-layer neural network. This suggests a possible presence of overfitting in the 2-layer neural network. Furthermore, when employing a remediation technique such as the log transformation of the target variable, the linear regression model exhibits a higher root mean square error (RMSE) than the 2-layer neural network in the train-test split evaluation.

MNIST

The MNIST dataset[181] is derived from the National Institute of Standards and Technology database and serves as a benchmark for evaluating the performance of various models. It comprises 70,000 grayscale images of handwritten digits, each represented by a 28×28 pixel grid. The pixel values range from 0 to 255, with higher values indicating brighter pixels. The objective is to classify these images into one of ten classes, corresponding to integer values from 0 to 9.

The dataset consists of 60,000 training samples and 10,000 testing samples, with each observation having 10 classes and 784 variables. It's worth noting that the data is sparse, as there is often empty space around the edges of each image, resulting in many variables being set to zero. Consequently, the assumption of linearity in logistic regression is violated when applied to this dataset.

When comparing logistic regression and a 2-layer neural network on the MNIST dataset, I also aimed to assess their predictive capabilities. To start, I designated the handwritten digit '0' as the target variable, while categorizing all other digits as 'others.' For logistic regression, the model achieved an impressive ROC-AUC score of 0.9961, with a precision of 0.96 for classifying the digit '0.' Given the highly imbalanced nature of the data, the regression output demonstrated considerable precision.

Next, I applied the 2-layer neural network, which yielded a slightly lower ROC-AUC score of 0.9946 and a precision of 0.83 for classifying the digit '0.' At first glance, it may appear that the neural network performed worse than logistic regression when using these evaluation metrics.

However, when considering all ten classes for classification, the neural network outperformed multivariate logistic regression in terms of overall accuracy. Additionally, the precision of the neural network was significantly higher when conducting multi-class classification. It is important to note that logistic regression in the scikit-learn package employs a one-vs-rest schema, creating a ten-dimensional outcome where each dimension represents a specific class with the target variable set to 1 and the others set to 0. In contrast, the neural network utilizes the softmax function ($\frac{\exp(z_i)}{\sum \exp(z_j)}$) to output the probability of a case belonging to each class. The softmax function is a generalized extension of the sigmoid function.

The confusion matrix of the multinomial logistic regression and the neural network is shown below. We observed that the macro average for the neural network is significantly higher than that of logistic regression. Additionally, the neural network exhibits higher precision for predicting each digit class compared to logistic regression as well.

	precision	recall	f1-score	support
0	0.96	0.96	0.96	996
1	0.96	0.97	0.96	1141
2	0.90	0.88	0.89	1040
3	0.90	0.87	0.89	1013
4	0.91	0.92	0.91	962
5	0.87	0.86	0.87	863
6	0.93	0.95	0.94	989
7	0.92	0.91	0.92	1064
8	0.87	0.89	0.88	963
9	0.87	0.89	0.88	969
accuracy			0.91	10000
macro avg	0.91	0.91	0.91	10000
weighted avg	0.91	0.91	0.91	10000

Figure 6.4: Logistic Regression Confusion Matrix

	precision	recall	f1-score	support
0	0.98	0.96	0.97	996
1	0.98	0.98	0.98	1141
2	0.96	0.95	0.95	1040
3	0.94	0.92	0.93	1013
4	0.94	0.95	0.95	962
5	0.93	0.94	0.93	863
6	0.96	0.97	0.96	989
7	0.95	0.96	0.96	1064
8	0.92	0.94	0.93	963
9	0.93	0.94	0.93	969
accuracy			0.95	10000
macro avg	0.95	0.95	0.95	10000
weighted avg	0.95	0.95	0.95	10000

Figure 6.5: Neural Network Confusion Matrix

6.3 Conclusion

When applying linear regression, assuming the model is not misspecified, it generally exhibits superior performance compared to neural networks when evaluating the Mean Squared Error (MSE) as the performance metric. As the sample size increases, the performance of neural networks becomes closer to that of linear regression. However, when the linear model is misspecified, neural networks tend to provide better prediction performance than linear regression. In such cases, larger sample sizes contribute to improved prediction performance, while minimizing the variance in error also leads to better predictions.

Regarding bi-variate classification tasks, logistic regression tends to outperform two-layer artificial neural networks (ANNs). We examined 28 papers specified in [182] and found that in 10 cases (36%), ANNs outperformed logistic regression, while in 4 cases (14%), logistic regression outperformed ANNs. The two methods exhibited similar performance in the remaining 14 cases (50%). In scenarios where the output is not binary, Cox regression is typically employed for this type of data. A detailed comparison is provided below:

It can be concluded that logistic regression performs better than neural networks in binary classification tasks. However, when conducting multi-class classification, logistic regression exhibits inferior performance compared to two-layer neural networks.

In a related study mentioned in [135], specifically Schumacher (1996), it is noted that logistic regression demonstrates better model understanding than neural networks when the dataset involves less than 400 observations and/or more than five

¹On ROC there is no difference, but Hosmer-Lemeshow test for logistic reg is 0.34 and NN is 0.08, which means NN is lack of fit

²fraction of patients predicted to Survive who died (fpsd) as a function of the fraction of patients predicted to survive (fps))

³Author estimates the number of patient cases needed to show statistically significant differences in the fpsd values between LR and NN, the sample size in each level is in the extend of 10e5, also, fps is in the treatment level {0.1, 0.2, 0.3, 0.4, 0.5, 0.6}, then reported similar final result for both NN and logistic regression method on each treatment level.

⁴Although NN has better ROC, but NN had poorer calibration(TP vs predict percentage).

⁵ROC-AUC score for a neural network model was significantly larger than that for logistic regression in the training set ($p = 0.04$). However, the performance was statistically equivalent in the test set ($p = 0.45$).

⁶Neural Networks are able to distinguish patients at high and low risk from their DNA flow cytometry histograms and that they interpret the histograms differently than conventional techniques.

⁷The neural networks successfully estimated perioperative cardiac risk with better calibration than comparable logistic regression models.

⁸This is a book using multiple dataset, we only present diabete dataset result here.

⁹Logistic regression being the most unstable

¹⁰This model used to determine variable importance.

¹¹The ANN improvement over the Cox model on nonlinear data was very small (0.015), but is significant at $\alpha=0.05$

¹²On train-test split, the ROC for both methods are equal likely performed.

Table 6.1: paper performance

Paper	Evaluation Metric	Regression	NN	total size	validation size	Outcome
Lippmann [183]	ROC-AUC Score	76.20%	76.10%	80600	50% train	equi
Ennis [184]	ROC-AUC Score	81.8% (Model with full interaction)	81.6% (Variables only)	32092	66% train	equi
Warner [134]	ROC-AUC Score	71.62%	71.65%	32092	66% train	regression ¹
Cooper [185]	fpsd versus fps ²	-	-	14199	70% train	equi ³
Burke [186]	ROC-AUC Score	77.6% (Model with cubic spines of age)	78.4% (Variables only)	5773	60% train	NN
Selker [187]	ROC-AUC Score	90.5%	92.3%	8271	63% train	NN ⁴
Rowland [188]	ROC-AUC Score	68%	69%	1674	50% train not random	equi
Duh [189]	ROC-AUC Score	90.9%	90.9%	5626	66% train	equi ⁵
Ravdin [190]	Goodness of fit score (Goodness of fit)	81.5	81.7	1373	66% train	equi
Ravdin [190]	low risk	5.50 %	5.60%	1590	50% train	equi ⁶

Table 6.2: paper performance cont.

Paper	Evaluation Metric	Regression	NN	total size	validation size	Outcome
Eisenstein [191]	ROC-AUC Score	no missing, no interact: 0.645; no missing, interact: 0.663; missing, no interact: 0.5; missing, interact: 0.5	no missing, 0 hid: 0.662; no missing, 1 hid: 0.600; missing, 0 hid: 0.568; missing, int: 0.598	1139	56% train	no conclusion
Lapuerta [192]	ROC and Hosmer-Lemeshow Chi-SQ	68.3% and 18.6	67.5% and 45	1081	52% train	neural network
Virtanen [193]	Sensitivity and specificity	87% and 41%	85% and 26%	974	5-fold	regression
Zirnikov [194]	ROC-AUC score	91.7%	95.4%	890	50% train	nn
Zirnikov [195]	ROC-AUC score	88.4%	93.5%	865	50% train	nn
Michie [196]	error rate	22.3%	24.8%	768	12-fold cv	reg ⁷
Jefferson [197]	accuracy on test	-	-	620	leave-one-out	nn ⁸
Ohno-Machado [195]	Not comparable	Test Statistic	ROC score -	588	10-fold cv	equi ⁹

Table 6.3: paper performance cont.

Paper	Evaluation Metric	Regression	NN	total size	validation size	Outcome
Buchman [198]	Accuracy, Specificity, Sensitivity, R-square	0.65, 0.62, 0.72 and 0.26	0.88, 0.83, 0.97, 0.57	491	66% train	neural network
Faraggi [199]	Concordance Index "C"	first order ph: 0.607; second order ph: 0.580	2 hidden layer : 0.600; 3hidden layer: 0.582	475	50% train	equi
Kattan [200]	Concordance Index "C"	0.79	0.8	424	66% train	nn ¹⁰
Doig [201]	sensitivity, specificity, ppv, npv, roc	0.133, 0.976, 0.4, 0.902, 0.8230	0.267, 0.976, 0.571, 0.916, 0.8178	422	66% train	nn ¹¹
Dybowski [202]	ROC and Brier Score	0.753, 0.2323	0.85, 0.16	258	65% train	neural network
Lette [203]	sensitivity, specificity, ppv, npv	0.67, 0.82, 0.18, 0.98	0.67, 0.96, 0.5, 0.98	360	55% train	neural network
Marchevsky [203]	accuracy	66%	89%	279	55% test in NN, 39% test in LR	neural network
Rae [204]	accuracy	20-item: 0.728, 5-item: 0.6	20-item: 0.83, 5-item: 0.731	274	180 train	neural network

Table 6.4: paper performance cont.

Hammad [205]	Sensitivity and PPV	66% and 59%	83% and 63%	251	80% train	neural network
Biagiotti [206]	sensitivity, speci- ficity, brier score	0.84, 0.966, 0.051	0.96, 0.977, 0.031	226	5-fold	neural network

covariates. Additionally, even with large sample sizes, unless the underlying function can only be captured by a single hidden layer neural network and cannot be expressed using logistic regression with interaction or quadratic terms, neural networks may provide a better understanding of the true data. In other cases, logistic regression is not inferior to neural networks and provides more interpretability to the model parameters. These findings align precisely with our own results.

Chapter 7 Future Development

During the course of my research, I have encountered several challenges and pitfalls associated with neural networks. As a result, I have identified various methods for enhancing and calibrating neural networks. Furthermore, advanced variance estimation techniques and conducting further comparisons between neural networks and linear models represent promising areas for future investigation. These aspects present valuable opportunities for expanding our understanding and refining the application of neural networks in research.

7.1 Problems in Neural Network

Neural networks are commonly employed for analyzing high-dimensional data, where the network structure tends to be intricate in terms of both dimensionality and layer sizes. During the process of fitting neural network models, I encountered three typical challenges:

- **Vanishing gradient:** This phenomenon, discussed in [207], refers to the issue of gradients becoming extremely small during the training process, leading to slow convergence or even stagnation. Dealing with vanishing gradients is crucial to ensure effective learning in neural networks.
- **Model oscillation:** Model oscillation occurs when the neural network's parameters fluctuate excessively during training, leading to unstable and inconsistent predictions. Overcoming model oscillation is essential for achieving reliable and robust neural network models.
- **Local optima:** Neural networks are prone to getting trapped in local optima, where the model converges to suboptimal solutions instead of finding the global optimum. Exploring strategies to escape local optima and guide neural networks towards better solutions is an ongoing challenge.

These pitfalls highlight the need for careful consideration and implementation of techniques to mitigate their impact and improve the overall performance and reliability of neural network models.

7.1.1 Vanish/Exploding Gradient

Backpropagation, which calculates gradients from the upper layer to the lower layer in neural networks, can lead to diminishing gradients. This means that the gradient values calculated for the lower layers become smaller and smaller, and in some cases, they may even vanish to zero. Conversely, in certain situations where the gradient in the upper layer is large, it can cause the gradient in the lower layer to

explode and disrupt the algorithm, leading to divergence.

Vanishing and exploding gradients are common obstacles encountered during the training and optimization of neural networks, presenting substantial challenges. Addressing these challenges requires careful attention and the utilization of specific techniques like weight initialization, gradient clipping, and normalization methods. These strategies play a vital role in alleviating these issues and promoting greater stability and efficacy in the learning process of deep neural networks.

If the initial weights come from a normal distribution, the variance of the weight value, denoted as w_{ij} , will depend on the specific initialization method used. In the case of Xavier initialization (also known as Glorot initialization)[208], the variance of w_{ij} is typically set to:

$$var(w_{ij}) = \frac{2}{n_{in} + n_{out}} \quad (7.1)$$

where n_{in} represents the number of input connections to the weight and n_{out} represents the number of output connections. This formula ensures that the variance of the weight values is adjusted according to the network's architecture. If initial weight come from uniform distribution $-[a,a]$, and a defined as

$$a = \sqrt{\frac{6}{n_{in} + n_{out}}} \quad (7.2)$$

On the other hand, He initialization[209] is defined similar as Xavier initialization, instead of using output information, He initialization only consider input connections for weights are being initialized.

In [208] Glorot and Bengio mentioned that vanishing gradient problem can also be due to choosing activation function badly. Here we define saturating function f as

Definition 7.1.1. For a function f , it is not saturating if and only if $|\lim_{z \rightarrow \infty} f(x) = +\infty|$ or $|\lim_{z \rightarrow -\infty} f(x) = +\infty|$

From the definition above, it is evident that the vanishing gradient problem can be alleviated if a function is not saturating. However, the choice of activation function plays a crucial role in determining whether saturation occurs. For instance, the Sigmoid function is commonly used but is saturating, meaning that its gradient approaches zero as the input values become very large or very small; On the other hand, the Rectified Linear Unit (ReLU)[210] activation function is not saturating when $x > 0$ since does not suffer from saturation when the input x is greater than zero, as its gradient remains constant. This characteristic of ReLU makes it more resilient to the vanishing gradient problem in such cases. ReLU activation function is shown as follow:

$$ReLU(x) = \max(0, x) \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases} \quad (7.3)$$

There are some extensions of ReLU functions as activation function, basic rule is when $x < 0$, we define $\sigma(x) = ax$, where a can be either a fixed value or random. Leaky ReLU[211], Parametric ReLU[209] and random ReLU[212] are all this type of extension.

There exists a smoother alternative to the non-saturating activation function called the Exponential Linear Unit (ELU)[213]. ELU is defined as follows:

$$ELU(x) = \begin{cases} a(\exp(x) - 1), & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases} \quad (7.4)$$

While ELU is slower compared to the ReLU activation function, it effectively avoids issues such as the dying gradient problem and non-smooth gradients. In a study by [214], it was demonstrated that by modifying the function for negative inputs to $ELU(x) = a(\exp(bx) - 1)$, the convergence of ReLU can be accelerated.

In addition to the choice of initialization and activation functions, Ioffe and Szegedy[215] proposed a method called batch normalization, which normalizes each input in each layer. This technique considerably reduces the vanishing/exploding gradient problem. Further research, such as the work by[216], has shown that combining dropout ([217]) with batch normalization can yield promising results.

For the purpose of continuously adjusting the learning rate in neural networks, especially in the context of Recurrent Neural Networks (RNNs)[218],introduced a typical approach known as gradient clipping.

7.1.2 Instability and Oscillation

In the earlier discussion on optimization methods in Chapter 1, we introduced the concept of the "learning rate." When utilizing a fixed learning rate, setting it too small can result in slow convergence of the gradient descent method. Conversely, choosing a learning rate that is too large can lead to the model converging to multiple points, causing oscillations and instability in the training process. It is essential to strike a balance and select an appropriate learning rate to ensure effective and efficient convergence of the optimization algorithm.

A commonly employed approach to adjust the learning rate is through the use of a learning rate schedule. Since the learning rate is a hyperparameter that requires tuning, various strategies can be applied to adapt it. These include tuning the global learning rate (as discussed in [219]), layer-wise learning rate, neuron-wise learning rate, or even a parameter-wise learning rate (resembling a diagonal Newton method, as described in [220]).

In the context of neural networks, [221] and [222] propose the utilization of a second-order diagonal Newton approximation, where each parameter is assigned a

different learning rate. This approach ensures that the learning rate adapts to individual parameters, promoting effective optimization.

Several adaptive learning rate techniques have gained considerable attention in the neural network community. For instance, Adagrad ([223]) dynamically adjusts the learning rate for each parameter based on the historical gradient information, allowing for efficient learning rate adaptation. Another notable method is the adaptive learning rate approach introduced by [224], which claims to eliminate the need for manual tuning of the learning rate hyperparameter entirely.

These adaptive learning rate procedures offer alternatives to manually tuning the learning rate and can contribute to improved optimization performance in neural networks.

Another approach to mitigate oscillation in the training of neural networks is referred to as momentum. In the context of Stochastic Gradient Descent (SGD), where a random parameter is chosen to update all parameters using its gradient, researchers proposed a method to smooth out these gradients by taking the average of previously applied gradients. This smoothing technique, described in [225] and [226], involves updating a gradient average as $\bar{g} \leftarrow (1 - \beta)\bar{g} + \beta g$. By doing so, the idea is to reduce noise and oscillations that can occur during the gradient descent process.

Polyak averaging, introduced in [227] and based on the work by [228], and other methods such as those proposed by [229] and [230], also incorporate some form of momentum to accelerate convex optimization [231]. Additionally, researchers like [232], [233], and [234] have suggested using second-order derivative information to further enhance the momentum method's speed.

These techniques for adjusting the learning rate in gradient descent can generally alleviate the effects of oscillation and instability during neural network model training. They create a more stable learning environment compared to traditional neural network approaches. By applying these adjustments to the learning rate, we can stabilize the fitting process of neural networks and gain deeper insights into the relationship between neural networks and statistical non-linear models.

7.1.3 Local Optima

Training deeper neural networks poses a greater challenge compared to shallow ones due to an increased likelihood of missing out on superior minima when starting from random initialization. Extensive research has demonstrated this phenomenon in both supervised deep learning techniques ([235], [236], and [237]) and unsupervised deep learning techniques ([238]). Consequently, the application of appropriate initialization schemes can substantially enhance performance.

The study presented in [239] reveals that different initializations consistently lead to distinct effective local minima. By employing unsupervised pretraining, it becomes possible to discover minima that outperform those obtained through random initialization. Another approach known as "curriculum learning" has been proposed ([240], [241], and [242]). This method involves commencing the learning process with an easier optimization task, such as a convex problem, and gradually transitioning to more challenging tasks that closely align with the actual objective of interest. In addition, [243] suggests an initialization technique that initializes weights to ensure that the Jacobian of each layer has singular values close to 1. This initialization strategy helps establish appropriate initial values within the correct range.

These three types of difficulties in neural network and some solutions/remedy plans has been discussed above, however, how to statistically interpret these advanced method to better fit neural network can be studied in the future.

7.2 Advanced Neural Network

In contrast to traditional artificial neural network (ANN) methods, neural networks have evolved and diversified in multiple ways. In this context, we will introduce two commonly employed advanced neural network frameworks that facilitate the construction of future intervals and enable the learning of statistical characteristics within these models. These frameworks serve as fundamental structures for building the majority of neural networks utilized today.

7.2.1 Convolutional Neural Network

Drawing inspiration from the organization of the visual cortex and the connectivity patterns of neurons in the human brain, the Convolutional Neural Network (ConvNet) has been developed based on this neuron connection method. ConvNets are primarily utilized in computer vision and image-based data domains.

The core component of ConvNets lies in their ability to process image data, which is distinct and presents unique challenges. Images exhibit spatial and temporal dependencies that are inherently difficult to handle using traditional methods of data processing. ConvNets have proven to be highly effective in addressing these challenges and extracting meaningful features from image data.

ConvNets incorporate two essential types of layers: kernels (filters) and pooling layers. A kernel, also known as a filter, is a small matrix that is applied to an image to extract specific features. On the other hand, pooling layers perform dimensionality reduction and statistical extraction on the image data.

Following the dimension processing performed by the kernel and pooling layers, a feed-forward neural network is employed to accomplish classification or regression

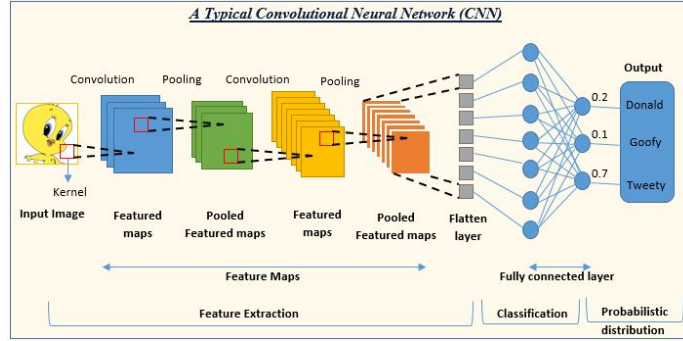


Figure 7.1: Neuron Structure CNN

tasks. For further in-depth information about ConvNets, please refer to [244]. This resource provides detailed insights into ConvNet architectures and their applications.

7.2.2 Recurrent Neural Network

Recurrent neural networks (RNNs)[245] are widely used in the analysis of sequential data. Unlike traditional neural networks, which assume independence among samples, RNNs are designed to incorporate dependencies between data points.

In sequential data, all input observations inherently exhibit auto-correlation. To account for this, we can unfold the input matrix X into a sequence of individual observations: $[x_1, x_2, \dots, x_t]$, where the subscript t represents the time point at which a specific observation was fed into the model. Instead of solely outputting y_t at each time point, we also introduce another hidden variable a_t as a hidden output. This hidden variable is utilized in the model fitting of the subsequent observation. Consequently, at each time step, both the information from the current input and the historical information captured by a_t can be leveraged.

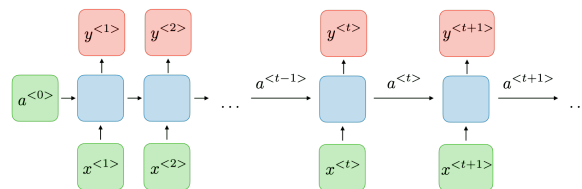


Figure 7.2: Neuron Structure RNN

The figure depicted above illustrates a single neuron within a recurrent neural network (RNN). By combining multiple neurons, we can form a layer, and stacking multiple layers results in a feed-forward neural network. This serves as the fundamental structure of a recurrent neural network.

When historical data is incorporated into an RNN using the aforementioned structure, we can only obtain one-step-ahead information. Consequently, alternative methods have been developed to facilitate long-term memory and selectively "forget" irrelevant information. Among these methods, two widely used approaches in practical applications are Long Short-term Memory (LSTM) [246] and Gated Recurrent Unit (GRU) [247]. By employing variance estimations in sequential data prediction using these advanced neural networks, we can achieve confidence intervals or tolerance intervals similar to those employed in traditional statistical methods.

7.3 Advanced Variance Estimation Method

We can delve into more advanced techniques for estimating variance in order to construct intervals. As discussed earlier in Chapter 2, there are several alternative methods available that can be utilized for this objective.

7.3.1 Variational Inference

In my research, I encounter numerous intractable distributions. To address this challenge, I employ variational inference, a method for approximating empirical distributions based on theoretical parametric distributions.

The primary objective of variational inference in my research is to provide an analytical approximation to the posterior probability of certain unobserved variables, enabling statistical inference on these variables. Variational inference can be viewed as an alternative to Monte Carlo sampling and an extension of the Expectation-Maximization (EM) algorithm.

The distribution of \mathbf{X} given some unobserved data \mathbf{Z} is approximated by $Q(\mathbf{X})$, which is selected from a family of distributions of simpler form than $P(\mathbf{X} | \mathbf{Z})$ (e.g., a family of Gaussian distributions). The goal is to make $Q(\mathbf{Z})$ similar to the true posterior distribution $P(\mathbf{Z} | \mathbf{X})$.

Similar to logistic regression, we utilize the Kullback-Leibler (KL) divergence as the loss function for variational inference. KL divergence measures the difference between two probability distributions. The formula for calculating continuous KL divergence is as follows:

$$D_{KL}(P||Q) = \int_{-\infty}^{\infty} P(x) \log \frac{P(x)}{Q(x)} dx \quad (7.5)$$

Here, P represents the proposed probability distribution, and Q is the reference distribution.

KL divergence is closely related to the Evidence Lower Bound (ELBO), which explains the term "ELBO" as follows:

$$\begin{aligned}
 \log(p(x)) &= \log \int \frac{q(\theta)p(x|\theta)}{q(\theta)} \\
 &= \log E_q \left[\frac{p(x, \theta)}{q(\theta)} \right] \\
 &\geq E_q \left[\log \frac{p(x, \theta)}{q(\theta)} \right] = ELBO
 \end{aligned}
 \tag{7.6}$$

The inequality arises from the Markov Inequality, and ELBO serves as a lower bound for the logarithm of the probability. In fact, the difference between $\log(p(x))$ and ELBO is the KL divergence.

Minimizing KL divergence minimizes the discrepancy between the theoretical distribution $q(x|\theta)$ and the empirical distribution $p(x)$ by updating the parameter θ using inference methods. Consequently, we can assume that the true population distribution of predictions is $p(y)$. By employing a theoretical distribution $q(y|\theta, w)$ to estimate the true population distribution, and assuming the theoretical distribution has parametric forms for statistical intervals, we can obtain statistical intervals for the predictions of neural networks.

The method described above was introduced in [248]. Variational inference has also been applied in RNN and CNN frameworks. Along this line, numerous discussions can be conducted in the future regarding different parameter settings for variational inference and other variance estimation methods based on variational inference.

7.3.2 Delete-d Jackknife

In Chapter 4, we introduced the concept of delete-1 jackknife, where one observation is omitted at a time to build jackknife samples. However, an alternative approach called delete-d jackknife allows for the omission of d observations to create jackknife samples. By excluding d observations in each iteration, we can generate $\binom{n}{d}$ jackknife samples, with each sample having a size of n-d.

The formula for the delete-d jackknife estimator of variance is as follows:

$$\frac{n-d}{d * \binom{n}{d}} \sum (\hat{\theta}_{(z)} - \hat{\theta}_.)^2$$

When d is close to n/2, the number of combinations can become extremely large, which can lead to computational challenges. Therefore, it is preferable to choose d closer to n. However, even with a smaller d, the computational resources required for delete-d jackknife can still be substantial. As a result, delete-d jackknife may not be the most suitable method for constructing confidence intervals in neural network applications.

7.4 Neural Network and Logistic Regression

In [135] and [136], there was also discussion in logistic regression. When applying sum of squares error or KL - distance as loss function, the back propagation update of parameters should give neural network and logistic regression similar output.

In fact, when a single layer neural network was applied in the [249] dataset. The logistic regression coefficient are the same as either using KL distance as loss function or sum of square errors in [135]. However, when I did a simulation on the following model:

$$\text{logit}(y) = 4 - 0.2x_1 - 0.3 * x_2$$

When x_1 is a random number from 0 or 1 with 0.5 probability, and x_2 is a random number from standard normal distribution. Using the *LogisticRegression* in *sklearn*, the optimization method used in this model is 'lbfgs'[250], I achieved 3.895 as intercept, and (0.2358, -0.2772) as regressed coefficient.

However, when I used *keras* to build a single neuron single layer neural network, and initialized all coefficient using a normal distribution with mean 0 and standard deviation 0.05, when Adam optimization is used, the coefficient achieved are 1.335 for intercept, (0.733, 1.343) as regressed coefficient. When I checked the ROC-score, logistic regression returned 0.764, while NN returned 0.502. The difference is large, and logistic regression seems to fit an more optimal situation.

It seems that logistic regression in Keras structure are more sensitive to intercept, which means when normalize those variables, the model may have a better fitting. Moreover, if learning rate was tuned in Adam optimizer, the coefficient could be achieved in a better stage. Although neural network has so many parameters to tune, people like to use neural network than logistic regression is because, people sacrifice model complexity to achieve better prediction.

In future research, if neural network's coefficient can be replicated as logistic regression, maybe we can have better understanding in logistic regression misspecification as well.

Chapter 8 Conclusion

Although neural network has been applied to various fields, distinct characteristics and requirements innate from neural network makes neural network differentiate from traditional statistical methods. As a result, their exploration in statistical analyses has been relatively limited, but there are ongoing efforts to bridge the gap between these two approaches. In my research, I focused on connecting neural networks with traditional linear models and incorporating statistical intervals into neural networks. Based on this research, several conclusions were drawn:

- Evaluation of Statistical Intervals: To assess the application of statistical intervals in neural networks, coverage analysis was employed. Parametric and nonparametric methods were explored to construct statistical intervals on neural networks, and their coverage was analyzed.
- Parametric Statistical Intervals: For the application of parametric statistical intervals in neural networks, variance estimation methods were investigated for neural network predictions. In the study of direct variance estimation, it was observed that robust estimation with neighborhood information outperformed the direct use of standard error on the neighborhood. The former method exhibited greater stability and closer adherence to the nominal coverage when the neighborhood size was small. In the case of resampling estimation on variance, the infinitesimal jackknife method demonstrated coverage levels closer to the nominal level compared to other traditional methods.
- Nonparametric Statistical Interval: The bootstrap resampling method was employed for constructing statistical intervals in neural networks. When compared to the parametric method, the bootstrap method is more computationally complex and may be challenging to apply in real-world scenarios. Although the coverage of the bootstrap method was found to be closer to the nominal level when only resampling under output was applied in prediction variance estimation, the computational complexity and practical considerations associated with the bootstrap method should be taken into account.
- Tolerance Interval: In the context of considering neural networks as nonparametric models, nonparametric regression was applied to neural networks under a one-dimensional nonlinear underlying model. The results demonstrated that the nonparametric regression approach outperformed the assumption of normal errors in the parametric tolerance interval. Even in cases where the underlying dataset exhibited a linear relationship, neural networks still exhibited better performance when treated as a nonparametric regression model, as evidenced by coverage levels closer to the nominal level.

- Comparison between Neural Network and Generalized Linear Model: In situations where the dimensionality is low, linear regression models tend to provide better fitting even in the presence of misspecification. This observation also holds true for cases where logistic regression is required. However, as the dimensionality increases or when multiple outcomes need to be classified, neural networks consistently outperform generalized linear models in terms of evaluation metrics. Neural networks exhibit superior performance in high-dimensional scenarios or when handling multi-class classification tasks.

After conducting preliminary research on neural networks, we discovered that while certain statistical intervals can be directly applied to neural networks, we encountered several challenges during our investigation. These challenges are outlined below:

- Since neural network are complex and highly non-linear models, it is really challenging to directly interpret the inner workings and understand the relationships between input features and output predictions through neural network models. Based on the reason above, post hoc analysis is oftenly used to understand the relationships between input features and output predictions and determine the confidence level of the network's predictions and evaluate its reliability. However, ad-hoc analysis are seldom used in neural network inference, making neural network hard to be explained.
- While Chapter 7 presented several approaches to mitigate suboptimal convergence and address gradient explosion or vanishing issues in neural network model fitting, it is important to note that certain challenges may still arise. Specifically, when the dimensionality of the variables is not large and the sample size is small, it is quite common to observe frequent oscillation around the "0" point.
- When the sample size is relatively small, it has been observed in multiple research studies, including my own, that neural networks may not perform as effectively as traditional linear regression or logistic regression models when the model is misspecified. Specifically, in the context of medical data, logistic regression has demonstrated comparable predictive power to linear regression models.

Based on the experience above, some further analysis can be studied:

- There is a burgeoning interest in bridging the gap between statistical methodologies and neural network techniques. The exploration of statistical properties and methodologies within the realm of neural networks represents a compelling and continuously evolving area of research. For instance, well-established techniques such as dropout have been scrutinized as potential resampling methods in the context of statistical inference. Acquiring a deeper understanding of the underlying mechanisms not only facilitates better comprehension of neural networks but also enhances their application across diverse domains.

- Advanced deep learning methods, including neural networks such as Generative Adversarial Networks (GANs), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs), pose challenges when it comes to estimating their variance. These challenges primarily arise from concerns related to interpretability and the substantial computational requirements involved. However, by leveraging resampling techniques and direct estimation methods, researchers can construct statistical intervals for these advanced neural networks and interpret their results in a more statistically meaningful manner.
- Advanced variance estimation method, such as variational inference, has been applied to neural network prediction already. By leveraging variational inference and Bayesian estimation in nonparametric regression, researchers can obtain more robust and interpretable results, gain insights into the uncertainty of the estimated relationships, and make informed decisions based on a more comprehensive understanding of the data.

This paper explores the application of various statistical methods to neural networks, although the depth of analysis in these studies is relatively limited. There is significant potential for further exploration and research in this field, making it highly worthwhile to pursue. By delving deeper into statistical analyses applied to neural networks, researchers can uncover valuable insights and advance our understanding of the complex interactions between statistical methods and deep learning. With numerous unexplored areas awaiting investigation, this field holds promising opportunities for novel discoveries and advancements.

Chapter 9 Appendix

9.1 Proof of unbiased estimator of jackknife

We define $\mathbf{P}^* = (P_1^*, \dots, P_n^*)$ be vector of probabilities satisfying $0 < P_i^* < 1$ and $\sum_1^n P_i^* = 1$, also $\hat{F}^* = \hat{F}(\mathbf{P}^*)$ putting mass P_i^* on x_i . Hence we define a $\hat{\theta}^*$

$$\hat{\theta}^* = T(\mathbf{P}^*) \equiv t(\hat{F}^*(\mathbf{P}^*)) \quad (9.1)$$

Our statistics can be explained as a set of vectors \mathbf{P}^* satisfying $0 \leq P_i^* \leq 1$ and $\sum_1^n P_i^* = 1$, this set of vector is considered as an simplex and denoted by S_n . (Can be consider as assign weight in sample statistics)

If we define

$$\mathbf{P}^0 = \left(\frac{1}{n}, \frac{1}{n}, \dots, \frac{1}{n}\right)^T \quad (9.2)$$

then $T(\mathbf{P}^0)$ is the observed value of statistics. The jackknife value of statistic are

$$\hat{\theta}_{(i)} = T(\mathbf{P}_{(i)}) \quad (9.3)$$

where

$$\mathbf{P}_{(i)} = \left(\frac{1}{n-1}, \dots, 0, \frac{1}{n-1}, \dots, \frac{1}{n-1}\right)^T, \text{ where } 0 \text{ is on the } i^{\text{th}} \text{ place} \quad (9.4)$$

We define linear statistic $T(\mathbf{P}^*)$ be

$$T(\mathbf{P}^*) = c_0 + (\mathbf{P}^* - \mathbf{P}^0)^T \mathbf{U} \quad (9.5)$$

when c_0 is constant and $\mathbf{U} = (U_1, \dots, U_n)^T$ satisfying $\sum_1^n U_i = 0$

Theorem 3. *For a linear statistic, the jackknife estimates of bias is identically 0*

Proof. We know that T passes through all points, $(\mathbf{P}_{(i)}, T(\mathbf{P}_{(i)}))$ for $i=1,2,\dots,n$, as well as $(\mathbf{P}^0, T(\mathbf{P}^0))$

$$\begin{aligned} c_0 &= T(\mathbf{P}^0) \\ \hat{\theta}_{(i)} &= c_0 + (\mathbf{P}_{(i)} - \mathbf{P}^0)^T \mathbf{U} \end{aligned} \quad (9.6)$$

Hence, $\widehat{bias}_{jack} = (n-1)(\hat{\theta}_{(\cdot)} - \hat{\theta})$, and

$$\begin{aligned}
\hat{\theta}_{(\cdot)} - \hat{\theta} &= \sum \left(\frac{\theta_{(i)}}{n} - \hat{\theta} \right) \\
&= \sum (\mathbf{P}_{(i)} - \mathbf{P}^0)^T \mathbf{U} \\
&= \sum_i \left\{ \left(\frac{1}{n-1} - \frac{1}{n} \right) U_1 + \dots - \frac{1}{n} U_i + \dots + \left(\frac{1}{n-1} - \frac{1}{n} \right) U_n \right\} \\
&= \sum_i \left\{ \sum_{j \neq i} \left(\frac{1}{n-1} - \frac{1}{n} \right) U_j - \frac{1}{n} U_i \right\} \\
&= \sum_i \left\{ - \left(\frac{1}{n-1} - \frac{1}{n} \right) U_i - \frac{1}{n} U_i \right\} \\
&= \sum_i - \left(\frac{1}{n-1} U_i \right) \\
&= 0
\end{aligned} \tag{9.7}$$

□

9.2 Proof of Jackknife as Approximation to Bootstrap

For any statistic, jackknife estimate for $T(\mathbf{P}^*)$ is almost the same as the bootstrap variance for a certain linear approximation to $T(\mathbf{P}^*)$.

Theorem 4. *Suppose linear statistic $T^{(LIN)}$ is a unique hyperplane passing through the jackknife points $(\mathbf{P}_{(i)}, T(\mathbf{P}_{(i)}))$ for $i = 1, 2, \dots, n$, Then*

$$var_* T^{(LIN)} = \frac{n-1}{n} var_{jack} \hat{\theta} \tag{9.8}$$

where $var_{jack} \hat{\theta}$ is the jackknife estimate of variance for $\hat{\theta}$. In other words, the jackknife estimate of variance for $\hat{\theta} = t(\hat{F})$ equals $n/n-1$ times bootstrap estimate of variance for $T^{(LIN)}$.

Proof. To solve a set of n linear equations

$$\hat{\theta}_{(i)} = T^{(LIN)}(\mathbf{P}_{(i)}) \tag{9.9}$$

we can obtain

$$c_0 = \hat{\theta}; U_i = (n-1)(\hat{\theta}_{(i)} - \hat{\theta}_{(\cdot)}) \tag{9.10}$$

and

$$\begin{aligned}
var_* T^{(LIN)}(\mathbf{P}^*) &= U^T (Var_* \mathbf{P}^*) U + 2U^T Cov_*(P^*) \\
&= \frac{1}{n^2} U^T U \\
&= \frac{n-1}{n} \left\{ \frac{n-1}{n} \sum (\hat{\theta}_{(i)} - \hat{\theta}_{(\cdot)})^2 \right\}
\end{aligned} \tag{9.11}$$

□

9.3 Proof of coefficient of variation in standard error estimation

9.4 Iteratively Reweighted Least Squares in Logistic Regression

First we derive log likelihood in a new form, here $\log p - \log(1 - p) = \beta^T x_i$:

$$\begin{aligned} l(\beta) &= \sum (y_i \log p + (1 - y_i) \log(1 - p)) \\ &= \sum (y_i \log(p) - y_i \log(1 - p) + \log(1 - p)) \\ &= \sum (y_i \beta^T x_i - \log(1 + e^{\beta^T x_i})) \end{aligned} \tag{9.12}$$

The first order derivative is

$$\begin{aligned} \frac{\partial l(\beta)}{\partial \beta} &= \sum x_i y_i - \frac{x_i e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} \\ &= \sum (x_i (y_i - p)) = \mathbf{X}^T (\mathbf{y} - p) \end{aligned} \tag{9.13}$$

To solve this equation to be 0, we have to use Newton-Raphson algorithm, which requires Hessian matrix:

$$\frac{\partial^2 l(\beta)}{\partial \beta \partial \beta^T} = \sum x_i x_i^T p(1 - p) = -\mathbf{X}^T \mathbf{W} \mathbf{X} \tag{9.14}$$

The Newton step will be

$$\begin{aligned} \beta^{new} &= \beta^{old} + (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T (\mathbf{y} - p) \\ &= (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} (\mathbf{X} \beta^{old} \mathbf{W}^{-1} (\mathbf{y} - p)) \\ &= (\mathbf{X}^T \mathbf{W} \mathbf{X})^{-1} \mathbf{X}^T \mathbf{W} \mathbf{z} \end{aligned} \tag{9.15}$$

since each iteration p changes, so is \mathbf{W} , \mathbf{z} . Hence global minimum may not be achieved by the property of Newton-Raphson algorithm, and this is called iteratively reweighted least square in the algorithm.

Bibliography

- [1] Peter McCullagh and John A Nelder. *Generalized linear models*. Routledge, 2019.
- [2] Morton Kupperman. Probabilities of hypotheses and information-statistics in sampling from exponential-class populations. *The Annals of Mathematical Statistics*, 29(2):571–575, 1958. ISSN 00034851. URL <http://www.jstor.org/stable/2237349>.
- [3] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *J. Artif. Int. Res.*, 11(1):169–198, July 1999. ISSN 1076-9757.
- [4] R. Polikar. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, 2006. doi: 10.1109/MCAS.2006.1688199.
- [5] Lior Rokach. Ensemble-based classifiers. *Artif. Intell. Rev.*, 33(1–2):1–39, February 2010. ISSN 0269-2821. doi: 10.1007/s10462-009-9124-7. URL <https://doi.org/10.1007/s10462-009-9124-7>.
- [6] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.
- [7] G. Stein, B. Chen, A. Wu, and K. Hua. Decision tree classifier for network intrusion detection with ga-based feature selection. In *ACM-SE 43*, 2005.
- [8] Bahzad Jijo and Adnan Mohsin Abdulazeez. Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends*, 2:20–28, 01 2021.
- [9] Irfan Damanik, Agus Windarto, Anjar Wanto, Poningsih, Sundari Andani, and Widodo Saputra. Decision tree optimization in c4.5 algorithm using genetic algorithm. *Journal of Physics: Conference Series*, 1255:012012, 08 2019. doi: 10.1088/1742-6596/1255/1/012012.
- [10] Feng-Jen Yang. An extended idea about decision trees. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 349–354, 2019. doi: 10.1109/CSCI49370.2019.00068.
- [11] Jinwen Liang, Zheng Qin, Sheng Xiao, Lu Ou, and Xiaodong Lin. Efficient and secure decision tree classification for cloud-assisted online diagnosis services. *IEEE Transactions on Dependable and Secure Computing*, 18(4):1632–1644, 2021. doi: 10.1109/TDSC.2019.2922958.
- [12] Adel Eesa and Adnan Mohsin Abdulazeez. Intrusion detection and attack classifier based on three techniques: A comparative study. *Engineering and Technology journal, University of Technology, Baghdad*, 29:386, 01 2011.

- [13] Adel Sabry Eesa, Zeynep Orman, and Adnan Mohsin Abdulazeez Brifceni. A novel feature-selection approach based on the cuttlefish optimization algorithm for intrusion detection systems. *Expert Systems with Applications*, 42(5):2670–2679, 2015. ISSN 0957-4174. doi: <https://doi.org/10.1016/j.eswa.2014.11.009>. URL <https://www.sciencedirect.com/science/article/pii/S0957417414006952>.
- [14] Philip H. Swain and Hans Hauska. The decision tree classifier: Design and potential. *IEEE Transactions on Geoscience Electronics*, 15(3):142–147, 1977. doi: 10.1109/TGE.1977.6498972.
- [15] Batta Mahesh. Machine learning algorithms -a review, 01 2019.
- [16] A. Wald. Statistical Decision Functions. *The Annals of Mathematical Statistics*, 20(2):165–205, 1949.
- [17] V. Cheushev, D.A. Simovici, V. Shmerko, and S. Yanushkevich. Functional entropy and decision trees. In *Proceedings. 1998 28th IEEE International Symposium on Multiple- Valued Logic (Cat. No.98CB36138)*, pages 257–262, 1998. doi: 10.1109/ISMVL.1998.679467.
- [18] Xihui Chen, Zenan Yang, and Wei Lou. Fault diagnosis of rolling bearing based on the permutation entropy of vmd and decision tree. In *2019 3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE)*, pages 1911–1915, 2019. doi: 10.1109/EITCE47263.2019.9095187.
- [19] Changxing Shang, Min Li, Shengzhong Feng, Qingshan Jiang, and Jianping Fan. Feature selection via maximizing global information gain for text classification. *Knowledge-Based Systems*, 54:298–309, 2013. ISSN 0950-7051. doi: <https://doi.org/10.1016/j.knosys.2013.09.019>. URL <https://www.sciencedirect.com/science/article/pii/S0950705113003067>.
- [20] Laura Elena Raileanu and Kilian Stoffel. Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41(1):77–93, May 2004. ISSN 1573-7470. doi: 10.1023/B:AMAI.0000018580.96245.c6. URL <https://doi.org/10.1023/B:AMAI.0000018580.96245.c6>.
- [21] Yue Liu, Lingjie Hu, Fei Yan, and Bofeng Zhang. Information gain with weight based decision tree for the employment forecasting of undergraduates. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 2210–2213, 2013. doi: 10.1109/GreenCom-iThings-CPSCom.2013.417.
- [22] R. López De Mántaras. A distance-based attribute selection measure for decision tree induction. *Machine Learning*, 6(1):81–92, Jan 1991. ISSN 1573-0565. doi: 10.1023/A:1022694001379. URL <https://doi.org/10.1023/A:1022694001379>.

- [23] Shweta Taneja, Charu Gupta, Kratika Goyal, and Dharna Gureja. An enhanced k-nearest neighbor algorithm using information gain and clustering. In *2014 Fourth International Conference on Advanced Computing Communication Technologies*, pages 325–329, 2014. doi: 10.1109/ACCT.2014.22.
- [24] Yongheng Zhao and Yanxia Zhang. Comparison of decision tree methods for finding active objects. *Advances in Space Research*, 41(12):1955–1959, 2008. ISSN 0273-1177. doi: <https://doi.org/10.1016/j.asr.2007.07.020>. URL <https://www.sciencedirect.com/science/article/pii/S027311770700796X>.
- [25] Puran Tewari, Kapil Mittal, and Dinesh Khanduja. An insight into “decision tree analysis”. *World Wide Journal of Multidisciplinary Research and Development*, 3:111–115, 12 2017.
- [26] Priyanka and Dharmender Kumar. Decision tree classifier: a detailed survey. *International Journal of Information and Decision Sciences*, 12(3):246–269, 2020. URL <https://ideas.repec.org/a/ids/ijidsc/v12y2020i3p246-269.html>.
- [27] Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1): 1–39, Feb 2010. ISSN 1573-7462. doi: 10.1007/s10462-009-9124-7. URL <https://doi.org/10.1007/s10462-009-9124-7>.
- [28] Luc Devroye, László Györfi, and Gábor Lugosi. *A Probabilistic Theory of Pattern Recognition*, volume 31. 01 1996. ISBN 978-1-4612-6877-2. doi: 10.1007/978-1-4612-0711-5.
- [29] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, Jun 1990. ISSN 1573-0565. doi: 10.1007/BF00116037. URL <https://doi.org/10.1007/BF00116037>.
- [30] Jennifer A. Hoeting, David Madigan, Adrian E. Raftery, and Chris T. Volinsky. Bayesian model averaging: a tutorial (with comments by M. Clyde, David Draper and E. I. George, and a rejoinder by the authors). *Statistical Science*, 14 (4):382 – 417, 1999. doi: 10.1214/ss/1009212519. URL <https://doi.org/10.1214/ss/1009212519>.
- [31] Kristine Monteith, James L. Carroll, Kevin Seppi, and Tony Martinez. Turning bayesian model averaging into bayesian model combination. In *The 2011 International Joint Conference on Neural Networks*, pages 2657–2663, 2011. doi: 10.1109/IJCNN.2011.6033566.
- [32] Guangzhi Qu and Hui Wu. Bucket learning: Improving model quality through enhancing local patterns. In *2009 IEEE International Conference on Data Mining Workshops*, pages 539–544, 2009. doi: 10.1109/ICDMW.2009.66.
- [33] 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). URL <https://www.sciencedirect.com/science/article/pii/S0893608005800231>.

- [34] Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. *Classification And Regression Trees*. 10 2017. ISBN 9781315139470. doi: 10.1201/9781315139470.
- [35] Jerome H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67, 1991. ISSN 00905364. URL <http://www.jstor.org/stable/2241837>.
- [36] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, Aug 1996. ISSN 1573-0565. doi: 10.1007/BF00058655. URL <https://doi.org/10.1007/BF00058655>.
- [37] Leo Breiman. Heuristics of instability and stabilization in model selection. *The Annals of Statistics*, 24(6):2350 – 2383, 1996. doi: 10.1214/aos/1032181158. URL <https://doi.org/10.1214/aos/1032181158>.
- [38] Jerome H. Friedman and Peter Hall. On bagging and nonlinear estimation. *Journal of Statistical Planning and Inference*, 137(3):669–683, 2007. ISSN 0378-3758. doi: <https://doi.org/10.1016/j.jspi.2006.06.002>. URL <https://www.sciencedirect.com/science/article/pii/S0378375806001339>. Special Issue on Nonparametric Statistics and Related Topics: In honor of M.L. Puri.
- [39] Andreas Buja and Werner Stuetzle. Observations on bagging. *Statistica Sinica*, 16(2):323–351, 2006. ISSN 10170405, 19968507. URL <http://www.jstor.org/stable/24307547>.
- [40] Andreas Buja and Werner Stuetzle. Smoothing effects of bagging, 2000.
- [41] Peter Bühlmann and Bin Yu. Analyzing bagging. *The Annals of Statistics*, 30(4):927 – 961, 2002. doi: 10.1214/aos/1031689014. URL <https://doi.org/10.1214/aos/1031689014>.
- [42] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL <https://doi.org/10.1023/A:1010933404324>.
- [43] Hyun-Chul Kim, Shaoning Pang, Hong-Mo Je, Daijin Kim, and Sung-Yang Bang. Support vector machine ensemble with bagging. In Seong-Whan Lee and Alessandro Verri, editors, *Pattern Recognition with Support Vector Machines*, pages 397–408, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45665-0.
- [44] Kyoungnam Ha, Sungzoon Cho, and Douglas MacLachlan. Response models based on bagging neural networks. *Journal of Interactive Marketing*, 19(1):17–30, 2005. ISSN 1094-9968. doi: <https://doi.org/10.1002/dir.20028>. URL <https://www.sciencedirect.com/science/article/pii/S1094996805700591>.

- [45] R. Gencay and Min Qi. Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging. *IEEE Transactions on Neural Networks*, 12(4):726–734, 2001. doi: 10.1109/72.935086.
- [46] A.S. Khwaja, M. Naeem, A. Anpalagan, A. Venetsanopoulos, and B. Venkatesh. Improved short-term load forecasting using bagged neural networks. *Electric Power Systems Research*, 125:109–115, 2015. ISSN 0378-7796. doi: <https://doi.org/10.1016/j.epsr.2015.03.027>. URL <https://www.sciencedirect.com/science/article/pii/S0378779615000942>.
- [47] Ricardo F. Alvear-Sandoval and Aníbal R. Figueiras-Vidal. On building ensembles of stacked denoising auto-encoding classifiers and their further improvement. *Information Fusion*, 39:41–52, 2018. ISSN 1566-2535. doi: <https://doi.org/10.1016/j.inffus.2017.03.008>. URL <https://www.sciencedirect.com/science/article/pii/S1566253517300726>.
- [48] Torsten Hothorn, Berthold Lausen, Axel Benner, and Martin Radespiel-Tröger. Bagging survival trees. *Statistics in Medicine*, 23(1):77–91, 2004. doi: <https://doi.org/10.1002/sim.1593>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.1593>.
- [49] Dacheng Tao, Xiaou Tang, Xuelong Li, and Xindong Wu. Asymmetric bagging and random subspace for support vector machines-based relevance feedback in image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(7):1088–1099, 2006. doi: 10.1109/TPAMI.2006.134.
- [50] Shohei Hido, Hisashi Kashima, and Yutaka Takahashi. Roughly balanced bagging for imbalanced data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2(5-6):412–426, 2009. doi: <https://doi.org/10.1002/sam.10061>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/sam.10061>.
- [51] Jerzy Błaszczyński and Jerzy Stefanowski. Neighbourhood sampling in bagging for imbalanced data. *Neurocomputing*, 150:529–542, 2015. ISSN 0925-2312. doi: <https://doi.org/10.1016/j.neucom.2014.07.064>. URL <https://www.sciencedirect.com/science/article/pii/S0925231214012296>. Special Issue on Information Processing and Machine Learning for Applications of Engineering Solving Complex Machine Learning Problems with Ensemble Methods Visual Analytics using Multidimensional Projections.
- [52] Nikunj C. Oza and Stuart J. Russell. Online bagging and boosting. In Thomas S. Richardson and Tommi S. Jaakkola, editors, *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics*, volume R3 of *Proceedings of Machine Learning Research*, pages 229–236. PMLR, 04–07 Jan 2001. URL <https://proceedings.mlr.press/r3/oza01a.html>. Reissued by PMLR on 31 March 2021.

- [53] Vladimir Svetnik, Andy Liaw, Christopher Tong, J. Christopher Culberson, Robert P. Sheridan, and Bradley P. Feuston. Random forest: a classification and regression tool for compound classification and qsar modeling. *Journal of Chemical Information and Computer Sciences*, 43(6):1947–1958, Nov 2003. ISSN 0095-2338. doi: 10.1021/ci034160g. URL <https://doi.org/10.1021/ci034160g>.
- [54] Anantha M. Prasad, Louis R. Iverson, and Andy Liaw. Newer classification and regression tree techniques: Bagging and random forests for ecological prediction. *Ecosystems*, 9(2):181–199, Mar 2006. ISSN 1435-0629. doi: 10.1007/s10021-005-0054-1. URL <https://doi.org/10.1007/s10021-005-0054-1>.
- [55] D. Richard Cutler, Thomas C. Edwards Jr., Karen H. Beard, Adele Cutler, Kyle T. Hess, Jacob Gibson, and Joshua J. Lawler. Random forests for classification in ecology. *Ecology*, 88(11):2783–2792, 2007. doi: <https://doi.org/10.1890/07-0539.1>. URL <https://esajournals.onlinelibrary.wiley.com/doi/abs/10.1890/07-0539.1>.
- [56] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from single depth images. In *CVPR 2011*, pages 1297–1304, 2011. doi: 10.1109/CVPR.2011.5995316.
- [57] Ramón Díaz-Uriarte and Sara Alvarez de Andrés. Gene selection and classification of microarray data using random forest. *BMC Bioinformatics*, 7(1): 3, Jan 2006. ISSN 1471-2105. doi: 10.1186/1471-2105-7-3. URL <https://doi.org/10.1186/1471-2105-7-3>.
- [58] Hal R. Varian. Big data: New tricks for econometrics. *Journal of Economic Perspectives*, 28(2):3–28, May 2014. doi: 10.1257/jep.28.2.3. URL <https://www.aeaweb.org/articles?id=10.1257/jep.28.2.3>.
- [59] Simon Bernard, Laurent Heutte, and Sébastien Adam. Forest-rk: A new random forest induction method. In De-Shuang Huang, Donald C. Wunsch, Daniel S. Levine, and Kang-Hyun Jo, editors, *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence*, pages 430–437, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-85984-0.
- [60] Robin Genuer, Jean-Michel Poggi, and Christine Tuleau-Malot. Variable selection using random forests. *Pattern Recognition Letters*, 31(14): 2225–2236, 2010. ISSN 0167-8655. doi: <https://doi.org/10.1016/j.patrec.2010.03.014>. URL <https://www.sciencedirect.com/science/article/pii/S0167865510000954>.
- [61] Stefan Wager. Asymptotic theory for random forests, 2016.

- [62] Erwan Scornet, Gérard Biau, and Jean-Philippe Vert. Consistency of random forests. *The Annals of Statistics*, 43(4):1716 – 1741, 2015. doi: 10.1214/15-AOS1321. URL <https://doi.org/10.1214/15-AOS1321>.
- [63] Lucas Mentch and Giles Hooker. Quantifying uncertainty in random forests via confidence intervals and hypothesis tests, 2015.
- [64] Erwan Scornet. On the asymptotics of random forests. *Journal of Multivariate Analysis*, 146:72–83, 2016. ISSN 0047-259X. doi: <https://doi.org/10.1016/j.jmva.2015.06.009>. URL <https://www.sciencedirect.com/science/article/pii/S0047259X15001542>. Special Issue on Statistical Models and Methods for High or Infinite Dimensional Spaces.
- [65] Gérard Biau and Luc Devroye. On the layered nearest neighbour estimate, the bagged nearest neighbour estimate and the random forest method in regression and classification. *Journal of Multivariate Analysis*, 101(10): 2499–2518, 2010. ISSN 0047-259X. doi: <https://doi.org/10.1016/j.jmva.2010.06.019>. URL <https://www.sciencedirect.com/science/article/pii/S0047259X10001387>.
- [66] Gérard Biau, Frédéric Cérou, and Arnaud Guyader. On the rate of convergence of the bagged nearest neighbor estimate. *Journal of Machine Learning Research*, 11(22):687–712, 2010. URL <http://jmlr.org/papers/v11/biau10a.html>.
- [67] Gérard Biau and Erwan Scornet. A random forest guided tour. *TEST*, 25(2): 197–227, Jun 2016. ISSN 1863-8260. doi: 10.1007/s11749-016-0481-7. URL <https://doi.org/10.1007/s11749-016-0481-7>.
- [68] Simon Bernard, Sébastien Adam, and Laurent Heutte. Dynamic random forests. *Pattern Recognition Letters*, 33(12):1580–1586, 2012. ISSN 0167-8655. doi: <https://doi.org/10.1016/j.patrec.2012.04.003>. URL <https://www.sciencedirect.com/science/article/pii/S0167865512001274>.
- [69] Stacey J. Winham, Robert R. Freimuth, and Joanna M. Biernacka. A weighted random forests approach to improve predictive performance. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 6(6):496–505, 2013. doi: <https://doi.org/10.1002/sam.11196>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/sam.11196>.
- [70] Amir Saffari, Christian Leistner, Jakob Santner, Martin Godec, and Horst Bischof. On-line random forests. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1393–1400, 2009. doi: 10.1109/ICCVW.2009.5457447.
- [71] Misha Denil, David Matheson, and Nando De Freitas. Consistency of on-line random forests. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML’13*, page III–1256–III–1264. JMLR.org, 2013.

- [72] Balaji Lakshminarayanan, Daniel M. Roy, and Yee Whye Teh. Mondrian forests: Efficient online random forests, 2015.
- [73] Zhao Yi, Stefano Soatto, Maneesh Dewan, and Yiqiang Zhan. Information forests. In *2012 Information Theory and Applications Workshop*, pages 143–146, 2012. doi: 10.1109/ITA.2012.6181810.
- [74] Gérard Biau and Luc Devroye. Cellular tree classifiers. In Peter Auer, Alexander Clark, Thomas Zeugmann, and Sandra Zilles, editors, *Algorithmic Learning Theory*, pages 8–17, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11662-4.
- [75] Hemant Ishwaran, Udaya B. Kogalur, Eugene H. Blackstone, and Michael S. Lauer. Random survival forests. *The Annals of Applied Statistics*, 2(3):841 – 860, 2008. doi: 10.1214/08-AOAS169. URL <https://doi.org/10.1214/08-AOAS169>.
- [76] Fang Yang, Jiheng Wang, and Guangzhe Fan. Kernel induced random survival forests, 2010.
- [77] Hemant Ishwaran, Udaya B. Kogalur, Xi Chen, and Andy J. Minn. Random survival forests for high-dimensional data. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 4(1):115–132, 2011. doi: <https://doi.org/10.1002/sam.10103>. URL <https://onlinelibrary.wiley.com/doi/abs/10.1002/sam.10103>.
- [78] Stéphan Cléménçon, Marine Depecker, and Nicolas Vayatis. Ranking forests. *J. Mach. Learn. Res.*, 14:39–73, 2013. ISSN 1532-4435.
- [79] Donghui Yan, Aiyou Chen, and Michael I. Jordan. Cluster forests. *Computational Statistics & Data Analysis*, 66:178–192, 2013. ISSN 0167-9473. doi: <https://doi.org/10.1016/j.csda.2013.04.010>. URL <https://www.sciencedirect.com/science/article/pii/S0167947313001400>.
- [80] Nicolai Meinshausen. Quantile regression forests. *Journal of Machine Learning Research*, 7(35):983–999, 2006. URL <http://jmlr.org/papers/v7/meinshausen06a.html>.
- [81] R. Courant. Variational methods for the solution of problems of equilibrium and vibrations. *Bulletin of the American Mathematical Society*, 49(1):1 – 23, 1943. doi: [bams/1183504922](https://doi.org/10.1090/bams/1183504922). URL <https://doi.org/>.
- [82] Léon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. URL <http://leon.bottou.org/papers/bottou-98x>. revised, oct 2012.

- [83] Sarit Khirirat, Hamid Reza Feyzmahdavian, and Mikael Johansson. Mini-batch gradient descent: Faster convergence under data sparsity. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 2880–2887, 2017. doi: 10.1109/CDC.2017.8264077.
- [84] Maya R. Gupta, Samy Bengio, and Jason Weston. Training highly multiclass classifiers. *J. Mach. Learn. Res.*, 15(1):1461–1492, jan 2014. ISSN 1532-4435.
- [85] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.
- [86] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL <http://arxiv.org/abs/1609.04747>.
- [87] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL <https://proceedings.neurips.cc/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf>.
- [88] Justin Domke. Generic methods for optimization-based modeling. In Neil D. Lawrence and Mark Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 318–326, La Palma, Canary Islands, 21–23 Apr 2012. PMLR. URL <https://proceedings.mlr.press/v22/domke12.html>.
- [89] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL <https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf>.
- [90] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In Arthur Gretton and Christian C. Robert, editors, *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 240–248, Cadiz, Spain, 09–11 May 2016. PMLR. URL <https://proceedings.mlr.press/v51/jamieson16.html>.
- [91] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Ben-tzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 230–246, 2020. URL <https://proceedings.mlsys.org/paper/2020/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf>.

- [92] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018.
- [93] Elad Hazan, Adam Klivans, and Yang Yuan. Hyperparameter optimization: A spectral approach, 2018.
- [94] Jakob Gawlikowski, Cedric Rouvère Njéutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, et al. A survey of uncertainty in deep neural networks. *arXiv preprint arXiv:2107.03342*, 2021.
- [95] Abhijit Guha Roy, Sailesh Conjeti, Nassir Navab, Christian Wachinger, Alzheimer’s Disease Neuroimaging Initiative, et al. Bayesian quicknat: Model uncertainty in deep whole-brain segmentation for structure-wise quality control. *NeuroImage*, 195:11–22, 2019.
- [96] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. Training confidence-calibrated classifiers for detecting out-of-distribution samples. *arXiv preprint arXiv:1711.09325*, 2017.
- [97] John Mitros and Brian Mac Namee. On the validity of bayesian neural networks for uncertainty estimation. *arXiv preprint arXiv:1912.01530*, 2019.
- [98] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.
- [99] Andrew G Wilson and Pavel Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *Advances in neural information processing systems*, 33:4697–4708, 2020.
- [100] Murat Seckin Ayhan and Philipp Berens. Test-time data augmentation for estimation of heteroscedastic aleatoric uncertainty in deep neural networks. In *Medical Imaging with Deep Learning*, 2018.
- [101] Amrith Rawat, Martin Wistuba, and Maria-Irina Nicolae. Harnessing model uncertainty for detecting adversarial examples. In *NIPS Workshop on Bayesian Deep Learning*, 2017.
- [102] Alexandru Constantin Serban, Erik Poll, and Joost Visser. Adversarial examples—a complete characterisation of the phenomenon. *arXiv preprint arXiv:1810.01185*, 2018.
- [103] Lewis Smith and Yarin Gal. Understanding measures of uncertainty for adversarial example detection. *arXiv preprint arXiv:1803.08533*, 2018.
- [104] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, 110:457–506, 2021.

- [105] Andrey Malinin and Mark Gales. Predictive uncertainty estimation via prior networks. *Advances in neural information processing systems*, 31, 2018.
- [106] Arsenii Ashukha, Alexander Lyzhov, Dmitry Molchanov, and Dmitry Vetrov. Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. *arXiv preprint arXiv:2002.06470*, 2020.
- [107] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model’s uncertainty? evaluating predictive uncertainty under dataset shift. *Advances in neural information processing systems*, 32, 2019.
- [108] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.
- [109] Shiyu Liang, Yixuan Li, and Rayadurgam Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*, 2017.
- [110] Alireza Shafaei, Mark Schmidt, and James J Little. A less biased evaluation of out-of-distribution sample detectors. *arXiv preprint arXiv:1809.04729*, 2018.
- [111] Martin Mundt, Iuliia Pliushch, Sagnik Majumder, and Visvanathan Ramesh. Open set recognition through deep neural network uncertainty: Does out-of-distribution detection require generative classifiers? In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.
- [112] Tiago Ramalho and Miguel Miranda. Density estimation in representation space to predict model uncertainty. In *Engineering Dependable and Secure Machine Learning Systems: Third International Workshop, EDSMLS 2020, New York City, NY, USA, February 7, 2020, Revised Selected Papers 3*, pages 84–96. Springer, 2020.
- [113] Murat Sensoy, Lance Kaplan, and Melih Kandemir. Evidential deep learning to quantify classification uncertainty. *Advances in neural information processing systems*, 31, 2018.
- [114] Maithra Raghu, Katy Blumer, Rory Sayres, Ziad Obermeyer, Bobby Kleinberg, Sendhil Mullainathan, and Jon Kleinberg. Direct uncertainty prediction for medical second opinions. In *International Conference on Machine Learning*, pages 5281–5290. PMLR, 2019.
- [115] Jay Nandy, Wynne Hsu, and Mong Li Lee. Towards maximizing the representation gap between in-domain & out-of-distribution examples. *Advances in Neural Information Processing Systems*, 33:9239–9250, 2020.

- [116] Philipp Oberdiek, Matthias Rottmann, and Hanno Gottschalk. Classification uncertainty of deep neural networks based on gradient information. In *Artificial Neural Networks in Pattern Recognition: 8th IAPR TC3 Workshop, AN-NPR 2018, Siena, Italy, September 19–21, 2018, Proceedings 8*, pages 113–125. Springer, 2018.
- [117] Jinsol Lee and Ghassan AlRegib. Gradients as a measure of uncertainty in neural networks. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 2416–2420. IEEE, 2020.
- [118] Marcin Możejko, Mateusz Susik, and Rafał Karczewski. Inhibited softmax for uncertainty estimation in neural networks. *arXiv preprint arXiv:1810.01861*, 2018.
- [119] Luis Oala, Cosmas Heiß, Jan Macdonald, Maximilian März, Wojciech Samek, and Gitta Kutyniok. Interval neural networks: Uncertainty scores. *arXiv preprint arXiv:2003.11566*, 2020.
- [120] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13, 1993.
- [121] David Barber and Christopher M Bishop. Ensemble learning in bayesian neural networks. *Nato ASI Series F Computer and Systems Sciences*, 168:215–238, 1998.
- [122] Radford M Neal. Bayesian training of backpropagation networks by the hybrid monte carlo method. Technical report, Citeseer, 1992.
- [123] Guotai Wang, Wenqi Li, Sébastien Ourselin, and Tom Vercauteren. Automatic brain tumor segmentation using convolutional neural networks with test-time augmentation. In *Brainlesion: Glioma, Multiple Sclerosis, Stroke and Traumatic Brain Injuries: 4th International Workshop, BrainLes 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 16, 2018, Revised Selected Papers, Part II 4*, pages 61–72. Springer, 2019.
- [124] Guotai Wang, Wenqi Li, Michael Aertsen, Jan Deprest, Sébastien Ourselin, and Tom Vercauteren. Aleatoric uncertainty estimation with test-time augmentation for medical image segmentation with convolutional neural networks. *Neurocomputing*, 338:34–45, 2019.
- [125] Nikita Moshkov, Botond Mathe, Attila Kertesz-Farkas, Reka Hollandi, and Peter Horvath. Test-time augmentation for deep learning-based cell segmentation on microscopy images. *Scientific reports*, 10(1):1–7, 2020.
- [126] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

- [127] Stefan Lee, Senthil Purushwalkam, Michael Cogswell, David Crandall, and Dhruv Batra. Why m heads are better than one: Training a diverse ensemble of deep networks. *arXiv preprint arXiv:1511.06314*, 2015.
- [128] Fredrik K Gustafsson, Martin Danelljan, and Thomas B Schon. Evaluating scalable bayesian deep learning methods for robust computer vision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 318–319, 2020.
- [129] William H Beluch, Tim Genewein, Andreas Nürnberger, and Jan M Köhler. The power of ensembles for active learning in image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9368–9377, 2018.
- [130] Apoorv Vyas, Nataraj Jammalamadaka, Xia Zhu, Dipankar Das, Bharat Kaul, and Theodore L Willke. Out-of-distribution detection using an ensemble of self supervised leave-out classifiers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 550–564, 2018.
- [131] Bing Cheng and D Michael Titterton. Neural networks: A review from a statistical perspective. *Statistical science*, pages 2–30, 1994.
- [132] Brian D Ripley. Neural networks and related methods for classification. *Journal of the Royal Statistical Society: Series B (Methodological)*, 56(3):409–437, 1994.
- [133] Warren S Sarle. Neural networks and statistical models. 1994.
- [134] Brad Warner and Manavendra Misra. Understanding neural networks as statistical tools. *The american statistician*, 50(4):284–293, 1996.
- [135] Martin Schumacher, Reinhard Roßner, and Werner Vach. Neural networks and logistic regression: Part i. *Computational Statistics & Data Analysis*, 21(6):661–682, 1996.
- [136] Mukta Paliwal and Usha A Kumar. Neural networks and statistical techniques: A review of applications. *Expert systems with applications*, 36(1):2–17, 2009.
- [137] Kalimuthu Krishnamoorthy and Thomas Mathew. *Statistical tolerance regions: theory, applications, and computation*. John Wiley & Sons, 2009.
- [138] D. V. Lindley. Statistical tolerance regions: Classical and bayesian. *Journal of the Royal Statistical Society: Series A (General)*, 134(4):682–682, 1971. doi: <https://doi.org/10.2307/2343667>. URL <https://rss.onlinelibrary.wiley.com/doi/abs/10.2307/2343667>.
- [139] Mark G Vangel. Tolerance interval. *Wiley StatsRef: Statistics Reference Online*, pages 1–7, 2014.
- [140] William Q Meeker, Gerald J Hahn, and Luis A Escobar. *Statistical intervals: a guide for practitioners and researchers*, volume 541. John Wiley & Sons, 2017.

- [141] S. S. Wilks. Determination of sample sizes for setting tolerance limits. *The Annals of Mathematical Statistics*, 12(1):91–96, 1941. ISSN 00034851. URL <http://www.jstor.org/stable/2235627>.
- [142] S. S. Wilks. Statistical Prediction with Special Reference to the Problem of Tolerance Limits. *The Annals of Mathematical Statistics*, 13(4):400 – 409, 1942. doi: 10.1214/aoms/1177731537. URL <https://doi.org/10.1214/aoms/1177731537>.
- [143] JK Patel. Tolerance limits-a review. *Communications in statistics-Theory and Methods*, 15(9):2719–2762, 1986.
- [144] Miloš Jílek and Hanns Ackermann. A bibliography of statistical tolerance regions, ii. *Statistics: A Journal of Theoretical and Applied Statistics*, 20(1): 165–172, 1989.
- [145] DT Shirke, RR Kumbhar, and D Kundu. Tolerance intervals for exponentiated scale family of distributions. *Journal of Applied Statistics*, 32(10):1067–1074, 2005.
- [146] Piao Chen and Zhi-Sheng Ye. Approximate statistical limits for a gamma distribution. *Journal of Quality Technology*, 49(1):64–77, 2017.
- [147] Tianwen Tony Cai and Hsiuying Wang. Tolerance intervals for discrete distributions in exponential families. *Statistica Sinica*, pages 905–923, 2009.
- [148] Hsiuying Wang and Fugee Tsung. Tolerance intervals with improved coverage probabilities for binomial and poisson variables. *Technometrics*, 51(1):25–33, 2009.
- [149] Jafar Ahmadi and Nasser Reza Arghami. Nonparametric confidence and tolerance intervals from record values data. *Statistical Papers*, 44(4):455–468, 2003.
- [150] Derek S Young and Thomas Mathew. Improved nonparametric tolerance intervals based on interpolated and extrapolated order statistics. *Journal of Nonparametric Statistics*, 26(3):415–432, 2014.
- [151] John Aitchison. Two papers on the comparison of bayesian and frequentist approaches to statistical problems of prediction: Bayesian tolerance regions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):161–175, 1964.
- [152] Michael Hamada, Valen Johnson, Leslie M Moore, and Joanne Wendelberger. Bayesian prediction intervals and their relationship to tolerance intervals. *Technometrics*, 46(4):452–459, 2004.
- [153] Willis A Jensen. Approximations of tolerance intervals for normally distributed data. *Quality and Reliability Engineering International*, 25(5):571–580, 2009.

- [154] W Allen Wallis. Tolerance intervals for linear regression. In *Proceedings of the second Berkeley symposium on mathematical statistics and probability*, pages 43–51. University of California Press, 1951.
- [155] Raymond J Carroll and David Ruppert. Prediction and tolerance intervals with transformation and/or weighting. *Technometrics*, 33(2):197–210, 1991.
- [156] David Ruppert, Matt P Wand, and Raymond J Carroll. *Semiparametric regression*. Number 12. Cambridge university press, 2003.
- [157] Derek S Young. Regression tolerance intervals. *Communications in Statistics-Simulation and Computation*, 42(9):2040–2055, 2013.
- [158] Gerald J Lieberman and Rupert G Miller Jr. Simultaneous tolerance intervals in regression. *Biometrika*, 50(1-2):155–168, 1963.
- [159] Robert W Mee, Keith R Eberhardt, and Charles P Reeve. Calibration and simultaneous tolerance intervals for regression. *Technometrics*, 33(2):211–219, 1991.
- [160] Henry Scheffe. A statistical theory of calibration. *The Annals of Statistics*, pages 1–37, 1973.
- [161] F.R. Hampel, E.M. Ronchetti, P.J. Rousseeuw, and W.A. Stahel. *Robust Statistics: The Approach Based on Influence Functions*. Wiley Series in Probability and Statistics. Wiley, 2011. ISBN 9781118150689. URL <https://books.google.com/books?id=XK3uhrVefXQC>.
- [162] Frank R. Hampel. The influence curve and its role in robust estimation. *Journal of the American Statistical Association*, 69(346):383–393, 1974. doi: 10.1080/01621459.1974.10482962. URL <https://www.tandfonline.com/doi/abs/10.1080/01621459.1974.10482962>.
- [163] Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 0387947248.
- [164] M. H. Quenouille. Problems in Plane Sampling. *The Annals of Mathematical Statistics*, 20(3):355 – 375, 1949. doi: 10.1214/aoms/1177729989. URL <https://doi.org/10.1214/aoms/1177729989>.
- [165] M. H. Quenouille. Notes on bias in estimation. *Biometrika*, 43(3/4):353–360, 1956. ISSN 00063444. URL <http://www.jstor.org/stable/2332914>.
- [166] Abstracts of Papers. *The Annals of Mathematical Statistics*, 29(2):614 – 623, 1958. doi: 10.1214/aoms/1177706647. URL <https://doi.org/10.1214/aoms/1177706647>.
- [167] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Number 57 in Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, Boca Raton, Florida, USA, 1993.

- [168] Rupert G. Miller. A trustworthy jackknife. *The Annals of Mathematical Statistics*, 35(4):1594–1605, 1964. ISSN 00034851. URL <http://www.jstor.org/stable/2238296>.
- [169] RUPERT G. MILLER. The jackknife-a review. *Biometrika*, 61(1):1–15, 04 1974. ISSN 0006-3444. doi: 10.1093/biomet/61.1.1. URL <https://doi.org/10.1093/biomet/61.1.1>.
- [170] Thais Mayumi Oshiro, Pedro Santoro Perez, and José Augusto Baranauskas. How many trees in a random forest? In Petra Perner, editor, *Machine Learning and Data Mining in Pattern Recognition*, pages 154–168, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-31537-4.
- [171] In Choi. Econometrics: by fumio hayashi, princeton university press, 2000. *Econometric Theory*, 18(4):1000–1006, 2002. doi: 10.1017/S0266466602004115.
- [172] Peter J. Huber. *Robust Estimation of a Location Parameter*, pages 492–518. Springer New York, New York, NY, 1992. ISBN 978-1-4612-4380-9. doi: 10.1007/978-1-4612-4380-9_35. URL https://doi.org/10.1007/978-1-4612-4380-9_35.
- [173] David S. Birkes and Yadolah Dodge. Alternative methods of regression: Birkes/alternative. 1993.
- [174] Derek S. Young. Regression tolerance intervals. *Communications in Statistics - Simulation and Computation*, 42(9):2040–2055, 2013. doi: 10.1080/03610918.2012.689064. URL <https://doi.org/10.1080/03610918.2012.689064>.
- [175] G. David Faulkenberry and David L. Weeks. Sample size determination for tolerance limits. *Technometrics*, 10(2):343–348, 1968. ISSN 00401706. URL <http://www.jstor.org/stable/1267049>.
- [176] William S. Cleveland, Susan J. Devlin, and Eric Grosse. Regression by local fitting: Methods, properties, and computational algorithms. *Journal of Econometrics*, 37(1):87–114, 1988. ISSN 0304-4076. doi: [https://doi.org/10.1016/0304-4076\(88\)90077-2](https://doi.org/10.1016/0304-4076(88)90077-2). URL <https://www.sciencedirect.com/science/article/pii/0304407688900772>.
- [177] Derek S. Young. tolerance: An r package for estimating tolerance intervals. *Journal of Statistical Software*, 36(5):1–39, 2010. doi: 10.18637/jss.v036.i05. URL <https://www.jstatsoft.org/index.php/jss/article/view/v036i05>.
- [178] Richard A Johnson, Dean W Wichern, et al. Applied multivariate statistical analysis. *New Jersey*, 405, 1992.
- [179] R. Wilms, E. Mäthner, L. Winnen, and R. Lanwehr. Omitted variable bias: A threat to estimating causal relationships. *Methods in Psychology*, 5:100075, 2021. ISSN 2590-2601. doi: <https://doi.org/10.1016/j.metip>.

2021.100075. URL <https://www.sciencedirect.com/science/article/pii/S2590260121000321>.

- [180] R. Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, May 1997. URL <http://www.sciencedirect.com/science/article/B6V1D-3WMMX3B-B/1/4baafcb4328934d470158b0233c44102>.
- [181] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.
- [182] Daniel J. Sargent. Comparison of artificial neural networks with other statistical approaches. *Cancer*, 91(S8):1636–1642, 2001. doi: [https://doi.org/10.1002/1097-0142\(20010415\)91:8+\(1636::AID-CNCR1176\)3.0.CO;2-D](https://doi.org/10.1002/1097-0142(20010415)91:8+(1636::AID-CNCR1176)3.0.CO;2-D). URL <https://acsjournals.onlinelibrary.wiley.com/doi/abs/10.1002/1097-0142%2820010415%2991%3A8%2B%3C1636%3A%3AAID-CNCR1176%3E3.0.CO%3B2-D>.
- [183] Richard P Lippmann and David M Shahian. Coronary artery bypass risk prediction using neural networks. *The Annals of thoracic surgery*, 63(6):1635–1643, 1997.
- [184] Marguerite Ennis, Geoffrey Hinton, David Naylor, Mike Revow, and Robert Tibshirani. A comparison of statistical learning methods on the gusto database. *Statistics in medicine*, 17(21):2501–2508, 1998.
- [185] Gregory F Cooper, Constantin F Aliferis, Richard Ambrosino, John Aronis, Bruce G Buchanan, Richard Caruana, Michael J Fine, Clark Glymour, Geoffrey Gordon, Barbara H Hanusa, et al. An evaluation of machine-learning methods for predicting pneumonia mortality. *Artificial intelligence in medicine*, 9(2): 107–138, 1997.
- [186] H Burke, D Rosen, and P Goodman. Advances in neural information processing systems 7, 1995.
- [187] Harry P Selker, John L Griffith, Sanjay Patil, William J Long, and RB d’Agostino. A comparison of performance of mathematical predictive methods for medical diagnosis: identifying acute cardiac ischemia among emergency department patients. *Journal of investigative medicine: the official publication of the American Federation for Clinical Research*, 43(5):468–476, 1995.
- [188] Todd Rowland, Lucila Ohno-Machado, and A Ohrn. Comparison of multiple prediction models for ambulation following spinal cord injury. In *Proceedings of the AMIA Symposium*, page 528. American Medical Informatics Association, 1998.

- [189] Mei-Sheng Duh, Alexander M Walker, Marcello Pagano, and Kenneth Kronlund. Prediction and cross-validation of neural networks versus logistic regression: using hepatic disorders as an example. *American journal of epidemiology*, 147(4):407–413, 1998.
- [190] Peter M Ravdin and Gary M Clark. A practical application of neural network analysis for predicting outcome of individual breast cancer patients. *Breast cancer research and treatment*, 22(3):285–293, 1992.
- [191] EL Eisenstein and F Alemi. A comparison of three techniques for rapid model development: an application in patient risk-stratification. In *Proceedings of the AMIA Annual Fall Symposium*, page 443. American Medical Informatics Association, 1996.
- [192] Pablo Lapuerta, Gilbert J L’Italien, Sumita Paul, Robert C Hendel, Jeffrey A Leppo, Lee A Fleisher, Mylan C Cohen, Kim A Eagle, and Robert P Giugliano. Neural network assessment of perioperative cardiac risk in vascular surgery patients. *Medical decision making*, 18(1):70–75, 1998.
- [193] Arja Virtanen, Mehran Gomari, Ries Kranse, and Ulf-Hakan Stenman. Estimation of prostate cancer probability by logistic regression: free and total prostate-specific antigen, digital rectal examination, and heredity are significant variables. *Clinical chemistry*, 45(7):987–994, 1999.
- [194] B Zernikow, K Holtmannspoetter, E Michel, W Pielemeier, F Hornschuh, A Westermann, and KH Hennecke. Artificial neural network for risk assessment in preterm neonates. *Archives of Disease in Childhood-Fetal and Neonatal Edition*, 79(2):F129–F134, 1998.
- [195] B Zernikow, K Holtmannspoetter, E Michel, M Theilhaber, W Pielemeier, and KH Hennecke. Artificial neural network for predicting intracranial haemorrhage in preterm neonates. *Acta Paediatrica*, 87(9):969–975, 1998.
- [196] Donald Michie, David J Spiegelhalter, and Charles C Taylor. Machine learning, neural and statistical classification. 1994.
- [197] MF Jefferson, N Pendleton, CP Lucas, SB Lucas, and MA Horan. Evolution of artificial neural network architecture: prediction of depression after mania. *Methods of information in medicine*, 37(03):220–225, 1998.
- [198] Timothy G Buchman, Kenneth L Kubos, Alexander J Seidler, and Michael J Siegforth. A comparison of statistical and connectionist models for the prediction of chronicity in a surgical intensive care unit. *Critical care medicine*, 22(5):750–762, 1994.
- [199] David Faraggi and Richard Simon. A neural network model for survival data. *Statistics in medicine*, 14(1):73–82, 1995.

- [200] Michael W Kattan, Kenneth R Hess, and J Robert Beck. Experiments to determine whether recursive partitioning (cart) or an artificial neural network overcomes theoretical limitations of cox proportional hazards regression. *Computers and biomedical research*, 31(5):363–373, 1998.
- [201] Gordon S Doig, Kevin J Inman, William J Sibbald, CM Martin, and JM Robertson. Modeling mortality in the intensive care unit: comparing the performance of a back-propagation, associative-learning neural network with multivariate logistic regression. In *Proceedings of the annual symposium on computer application in medical care*, page 361. American Medical Informatics Association, 1993.
- [202] Richard Dybowski, Vanya Gant, P Weller, and R Chang. Prediction of outcome in critically ill patients using artificial neural network synthesised by genetic algorithm. *The Lancet*, 347(9009):1146–1150, 1996.
- [203] Jean Lette, Bruce W Colletti, Michel Cerino, Daniel Mcnamara, Marie-Claire Eybalin, André Levasseur, and Stanley Natnel. Artificial intelligence versus logistic regression statistical modelling to predict cardiac complications after noncardiac surgery. *Clinical cardiology*, 17(11):609–614, 1994.
- [204] Sarah A Rae, Wen Jia Wang, and Derek Partridge. Artificial neural networks: a potential role in osteoporosis. *Journal of the Royal Society of Medicine*, 92(3):119–122, 1999.
- [205] TA Hammad, MF Abdel-Wahab, N DeClaris, A El-Sahly, N El-Kady, and GT Strickland. Comparative evaluation of the use of artificial neural networks for modelling the epidemiology of schistosomiasis mansoni. *Transactions of the Royal Society of Tropical Medicine and Hygiene*, 90(4):372–376, 1996.
- [206] Roberto Biagiotti, Cristina Desii, Ermanno Vanzi, and Guido Gacci. Predicting ovarian malignancy: application of artificial neural networks to transvaginal and color doppler flow us. *Radiology*, 210(2):399–403, 1999.
- [207] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.
- [208] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterton, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL <https://proceedings.mlr.press/v9/glorot10a.html>.
- [209] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

- [210] P. Lennie. The cost of cortical computation. *Curr Biol*, 13(6):493–497, Mar 2003.
- [211] Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. 2013.
- [212] Muhammad Khalid, Junaid Baber, Mumraiz Khan Kasi, Maheen Bakhtyar, Varsha Devi, and Naveed Sheikh. Empirical evaluation of activation functions in deep convolution neural network for facial expression recognition. In *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*, pages 204–207, 2020. doi: 10.1109/TSP49548.2020.9163446.
- [213] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2016.
- [214] Yang Li, Chunxiao Fan, Yong Li, Qiong Wu, and Yue Ming. Improving deep neural network with multiple parametric exponential linear units. *Neurocomputing*, 301:11–24, 2018. ISSN 0925-2312. doi: <https://doi.org/10.1016/j.neucom.2018.01.084>. URL <https://www.sciencedirect.com/science/article/pii/S0925231218301255>.
- [215] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.
- [216] Xiang Li, Shuo Chen, Xiaolin Hu, and Jian Yang. Understanding the disharmony between dropout and batch normalization by variance shift. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2682–2690, 2019.
- [217] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
- [218] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. *arXiv preprint arXiv:1905.11881*, 2019.
- [219] KyungHyun Cho, Tapani Raiko, and Alexander T Ihler. Enhanced gradient and adaptive learning rate for training restricted boltzmann machines. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 105–112. Citeseer, 2011.
- [220] Antoine Bordes, Xavier Glorot, Jason Weston, and Yoshua Bengio. Joint learning of words and meaning representations for open-text semantic parsing. In *Artificial intelligence and statistics*, pages 127–135. PMLR, 2012.
- [221] Yann Le Cun and Françoise Fogelman-Soulié. Modèles connexionnistes de l’apprentissage. *Intellectica*, 2(1):114–143, 1987.

- [222] Yann LeCun et al. Generalization and network design strategies. *Connectionism in perspective*, 19(143-155):18, 1989.
- [223] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.
- [224] Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In *International conference on machine learning*, pages 343–351. PMLR, 2013.
- [225] Geoffrey E Hinton. Relaxation and its role in vision. 1977.
- [226] Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.
- [227] Kevin Swersky, Bo Chen, Ben Marlin, and Nando De Freitas. A tutorial on stochastic approximation algorithms for training restricted boltzmann machines and deep belief nets. In *2010 Information Theory and Applications Workshop (ITA)*, pages 1–10. IEEE, 2010.
- [228] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.
- [229] Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical programming*, 120(1):221–259, 2009.
- [230] Nicolas Roux, Mark Schmidt, and Francis Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. *Advances in neural information processing systems*, 25, 2012.
- [231] Arkadij Semenovič Nemirovskij and David Borisovich Yudin. Problem complexity and method efficiency in optimization. 1983.
- [232] L Bottou. Large-scale learning with stochastic gradient descent. *Neural Networks: Tricks of the Trade, Reloaded*. Springer, 2013.
- [233] Léon Bottou and Yann Cun. Large scale online learning. *Advances in neural information processing systems*, 16, 2003.
- [234] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. *Advances in neural information processing systems*, 20, 2007.
- [235] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.
- [236] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19, 2006.

- [237] Marc’Aurelio Ranzato, Christopher Poultney, Sumit Chopra, and Yann Cun. Efficient learning of sparse representations with an energy-based model. *Advances in neural information processing systems*, 19, 2006.
- [238] Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In David van Dyk and Max Welling, editors, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 448–455, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA, 16–18 Apr 2009. PMLR. URL <https://proceedings.mlr.press/v5/salakhutdinov09a.html>.
- [239] Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. Why does unsupervised pre-training help deep learning? In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 201–208. JMLR Workshop and Conference Proceedings, 2010.
- [240] Jeffrey L Elman. Learning and development in neural networks: The importance of starting small. *Cognition*, 48(1):71–99, 1993.
- [241] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.
- [242] Kai A Krueger and Peter Dayan. Flexible shaping: How learning in small steps helps. *Cognition*, 110(3):380–394, 2009.
- [243] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
- [244] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [245] Larry R Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 5:64–67, 2001.
- [246] Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pages 37–45, 2012.
- [247] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.
- [248] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

- [249] W Sauerbrei, H Madjar, and HJ Prömpeler. Differentiation of benign and malignant breast tumors by logistic regression and a classification tree using doppler flow signals. *Methods of information in medicine*, 37(03):226–234, 1998.
- [250] DC Liu and J Nocedal. On the limited memory method for large scale optimization: Mathematical programming b. 1989.

Vita

Sheng Yuan was born in Guilin, China, renowned for its breathtaking scenery. He spent his formative years in this enchanting town, often described as "the best vista among the whole world."

After graduating from Guilin's esteemed high school at the age of 18, Sheng embarked on his undergraduate journey at Nankai University, where he pursued a Bachelor of Science degree in Mathematics and Statistics. During his studies in mathematics, Sheng discovered the remarkable ability of data to capture and describe the intricacies of the world. This realization fueled his desire to delve deeper into the field, leading him to pursue a Ph.D. in Statistics at the University of Kentucky, beginning in the fall semester of 2016. As he delved further into the realm of statistics and machine learning, Sheng recognized a gap between the popular machine learning methods and statistical foundations. To bridge this gap, he collaborated with Dr. Arnold Stromberg to conduct research, connecting machine learning methods with statistical models.

Upon successfully completing his Ph.D. in Statistics, Sheng embarked on a career as a model validator at Citigroup. In this role, he applied his knowledge and expertise in both statistics and machine learning to validate industry models developed by Citibank. Sheng's aim was to contribute his learnings to the practical landscape of the industry and make a meaningful impact.