Stockholms universitet

Learning Concentric Circular Boundaries: A Simulation Study Comparing Neural Networks and Random Forests

Ellinor Lindkvist

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2025:13
Matematisk statistik
Juni 2025

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Stockholm University

Mathematical Statistics
Stockholm University
Bachelor Thesis **2025:13**
http://www.math.su.se

Learning Concentric Circular Boundaries: A Simulation Study Comparing Neural Networks and Random Forests

Ellinor Lindkvist*

June 2025

Abstract

As the amount of data generated by individuals continues to grow, the ability to extract information and make viable predictions from that data is increasingly important. This thesis aims to investigate the performance of two machine learning methods, namely random forest and neural network. For the neural network, we try two different activation functions, sigmoid and rectified linear unit (ReLU). The methods are used to solve a binary classification problem with simulated data, where each data point is labelled according to whether it falls within an even or odd numbered concentric circle. We evaluate the performances of the two methods for three different simulation scenarios: varying the number of input variables, varying the size of the training dataset and varying the label flipping probability. We measure the performance in classification error and mean squared prediction error (MSPE). The results show that the neural network with the ReLU activation achieves lower classification errors and MSPEs, overall, and that the neural network with a sigmoid activation function struggles to learn the signal in the data. This suggests that neural networks are better suited for circular boundaries than random forests but that the activation function has to be appropriately chosen for achieving good model performance.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: ellinor.lindkvist@gmail.com. Supervisor: Ola Hössjer and Johannes Heiny.

Learning Concentric Circular Boundaries: A Simulation Study Comparing Neural Networks and Random Forests

Ellinor Lindkvist

2025

SAMMANFATTNING

Allt eftersom data samlas in, blir förmågan att extrahera mönster och göra användbara prediktioner allt mer eftertraktad. I denna kandidatuppsats undersöker vi om det finns skillnad i prestanda mellan maskininlärningsmetoderna random forest och neuralt nätverk. I det neurala nätverket testar vi två olika aktiveringsfunktioner, sigmoidfunktionen och ReLU-funktionen. De båda metoderna används för att lösa ett binärt klassificeringsproblem med simulerad data. Den simulerade datan erhålls genom att varje punkt tilldelas en klass baserat på om den finns i en koncentrisk cirkel med udda eller jämnt nummer. Vi utvärderar prestanda för tre olika scenarier: genom att variera dimensionen på data, variera storleken på träningsdatamängden och variera sannolikheten att en punkt tillhör den andra klassen trots att den befinner sig i ett område för den första klassen. Vi mäter prestanda i klassificeringsfel och genomsnittligt kvadratiskt prediktionsfel (MSPE). Resultaten visar att det neurala nätverket med aktiveringsfunktionen ReLU generellt uppnår lägst klassificeringsfel och MSPE samt att det neurala nätverket med sigmoidfunktionen inte lär sig det underliggande mönstret i data. Detta antyder att neurala nätverk presterar bättre än random forest när mönstret är cirkulärt men att val av aktiveringsfunktion är väsentligt för att uppnå bra prestanda.

Acknowledgements

First and foremost, I would like to sincerely thank my supervisors, Ola Hössjer and Johannes Heiny, for their invaluable guidance, feedback and support throughout this thesis. I am also deeply grateful to my wonderful parents for always believing in me and for providing moments of respite throughout the bachelor's programme, including during the writing of this thesis.
ChatGPT has been used to correct Python code, ensure proper formatting in LaTeX, as well as a sounding board.

Contents

1 Introduction

1.1 Background and Motivation

In recent years, with the fast technological advances, almost everything we interact with generates data; from social media and viewing preferences on streaming services to financial transactions and tracking your health with apps. But data is of little use if we cannot find the patterns hidden behind the numbers and use those patterns to draw conclusions or make predictions.

One way to utilise data is through binary classification. Binary classification has many applications. For instance, binary classification makes it possible to predict if someone is a suitable blood donor from clinical data (Mostafa et al., 2021) and predict if someone will earn more than 50 000 USD per year based on social factors such as age, gender, education and marital status (Chen, 2021).

In this thesis, we use simulated data and implement the supervised machine learning methods random forest and neural network to learn the underlying pattern in the data and classify data points as belonging to either class 1 or class 0. A point of interest in previous research has been to use algorithms in order to apply the optimal weight to each tree in a random forest (Chen et al., 2024). As for neural networks, a comparative study investigated how the the choice of activation function affects performance as well as the impact of the dataset size (Boateng et al., 2023).

When we implement random forest in this simulation study, all trees will have equal weight. When we implement neural network, we consider the sigmoid activation function as well as the ReLU function. The focus is then on whether random forest or neural network perform better in terms of binary classification error. Comparison is done for three different scenarios, varying the dimension of the input data, varying the size of the training set and varying the probability that a label is flipped.

In order to compare the methods, both the classification error and the mean squared prediction error are computed. The main finding is that neural network with ReLU achieves lower errors overall, while using the sigmoid activation function yields a model that is essentially guessing which class a data point belongs to. This implies that neural networks are better suited for circular boundaries than random forests, however, selecting an appropriate activation function is essential for achieving good model performance.

1.2 Outline of Thesis

The structure of the thesis is as follows, Section 2 provides the reader with a brief introduction to machine learning as well as the theory behind the two methods implemented. Section 3 covers the specifics of the simulations. Section 4 swiftly presents the results, preceding a more thorough discussion of the results and future work in Section 5.

2 Theoretical Background

Unless otherwise stated, the content of this section is based on Lindholm et al. (2022).

2.1 Regression Models

Regression can be used to learn the relationship between input variables $\mathbf{X}$ and an output variable Y. For example, $\mathbf{X}$ could be a p-dimensional vector sampled from a multinormal distribution and Y a variable that is correlated with $\mathbf{X}$ through an input-output relation. This can be modelled through a function f such that,

$$\mathrm{Y} = f(\mathbf{X}; \boldsymbol{\theta}) + \varepsilon,$$

where $\boldsymbol{\theta}$ are the regression parameters of the model and ε is an independent error that cannot be explained by the model. Furthermore, the error term is assumed to be homoscedastic and normally distributed with mean 0 and variance σ^2. If the model is linear in its parameters $\boldsymbol{\theta}$, we have a linear regression model. But if the model is non-linear in its parameters, we have a non-linear regression model (Lindholm et al., 2022, pp. 37). Applying the linear operator conditional expected value on the model yields

$$\mathrm{E}[\mathrm{Y} \mid \mathbf{X} = \mathbf{x}] = \mathrm{E}[f(\mathbf{X}; \boldsymbol{\theta}) \mid \mathbf{X} = \mathbf{x})] + \mathrm{E}[\varepsilon \mid \mathbf{X} = \mathbf{x}],$$

which implies that

$$f(\mathbf{x}; \boldsymbol{\theta}) = \mathrm{E}[\mathrm{Y} \mid \mathbf{X} = \mathbf{x}],$$

since ε does not depend on $\mathbf{X}$ (Hastie et al., 2009, pp. 28). Regression models are fairly simple but they are important for more advanced methods such as neural networks (Lindholm et al., 2022, pp. 133).

2.1.1 Types of Machine Learning

Machine learning refers to a computer program with the ability to extract information from data and make predictions on previously unseen data. By presenting the problem as a mathematical model, the program is able to use training data to learn the parameters of the mathematical model and make accurate predictions. Since the program learns or adapts its predictions based on the available training data, the same program can be used for different problems.

Machine learning can be divided into three categories: reinforcement, supervised and unsupervised learning. In reinforcement learning, an agent interacts with an environment and learns what actions to take based on a reward system (Sutton et al., 2018, pp 1-2). In supervised learning, we have training data that contains input and output variables. We say that the input is labelled (Lindholm et al., 2022, pp. 13). In unsupervised learning, there are no labels, only the input is available (Lindholm et al., 2022, pp. 247). The focus of this thesis will be on supervised learning.

2.1.2 Supervised Learning and Binary Classification

Supervised learning uses training data $\mathcal{T} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, assumed to be observations of independent and identically distributed random variables with the same distribution as $(\mathbf{X}, \mathrm{Y})$, for the purpose of learning the regression function $f(\mathbf{x}; \boldsymbol{\theta}) = \mathrm{E}[\mathrm{Y} \mid \mathbf{X} = \mathbf{x}]$. Supervised machine learning problems are further categorised into regression and classification problems, meaning that the output Y is numerical and categorical, respectively. The focus of this thesis will be on binary classification problems.

Assuming that the data points $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$ are observations of independent random variables, and that Y belongs to class 1 or class 0, we have that the probability that Y belongs to class 1 is equal to the expected value of Y conditioned on $\mathbf{X}$,

$$\mathrm{E}[\mathrm{Y} \mid \mathbf{X} = \mathbf{x}] = P(\mathrm{Y} = 1 \mid \mathbf{X} = \mathbf{x}) = f(\mathbf{x}; \boldsymbol{\theta}).$$

We can also calculate the probability that Y is in class 0 conditioned on $\mathbf{X}$,

$$P(\mathrm{Y} = 0 \mid \mathbf{X} = \mathbf{x}) = 1 - f(\mathbf{x}; \boldsymbol{\theta}).$$

Since the probabilities sum to one, we note that Y conditioned on $\mathbf{X}$ is a Bernoulli distributed variable. Furthermore, while the expected value of ε is still 0, ε is not homoscedastic in the binary classification case, which we will now show. We have that the variance of Y conditioned on $\mathbf{X} = \mathbf{x}$ is

$$\mathrm{Var}(\mathrm{Y} \mid \mathbf{X} = \mathbf{x}) = \mathrm{Var}(f(\mathbf{X}; \boldsymbol{\theta}) + \varepsilon \mid \mathbf{X} = \mathbf{x}).$$

Since the variance of $f(\mathbf{X}; \boldsymbol{\theta})$ is 0 when conditioned on $\mathbf{X} = \mathbf{x}$, we can simplify the expression to

$$\mathrm{Var}(\mathrm{Y} \mid \mathbf{X} = \mathbf{x}) = \mathrm{Var}(\varepsilon \mid \mathbf{X} = \mathbf{x}).$$

Using the fact that Y conditioned on $\mathbf{X}$ is a Bernoulli distributed variable we finally get

$$\mathrm{Var}(\mathrm{Y} \mid \mathbf{X} = \mathbf{x}) = \mathrm{Var}(\varepsilon \mid \mathbf{X} = \mathbf{x}) = f(\mathbf{x}; \boldsymbol{\theta})(1 - f(\mathbf{x}; \boldsymbol{\theta}))$$

(Hastie et al., 2009, Section 2.6.1).

2.2 Using Trees for Classification

This section is based on Section 2.3 in Lindholm et al. (2022), unless otherwise stated.

Tree-based methods aim to partition the input space into disjoint regions. In the two-dimensional case, the regions are rectangle shaped as shown in Figure 1. The regions are then labelled as class 1 or class 0, visualised as blue circles and red triangles in the figure.

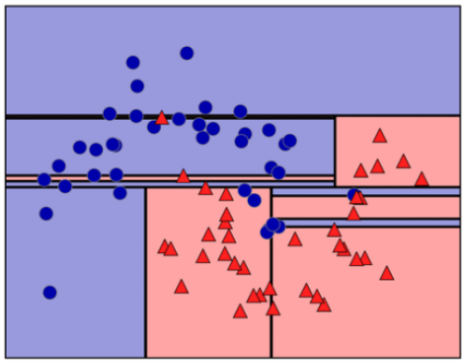

Figure 1: Cropped figure from Müller (2020). Partition of the input space for a binary classification tree with two input variables.

2.2.1 Deciding which Class with Binary Trees

A binary tree is a tree structure in which every inner node has at most two children. Decision trees can then be described as binary trees with a splitting criterion on each inner node. The condition is of the form $x_{\star j} < s_k$, where $x_{\star j}$ is one of the input variables from previously unseen input $\mathbf{x}_\star = [x_{\star 1}, \ldots, x_{\star p}]^\top$ and s_k is the numeric threshold on inner node k, with the root node being labelled as $k = 0$. If the inequality is true, the rule is to traverse down the left branch. In the false case, we instead traverse down the right branch. Repeating this procedure, we eventually end up in a leaf node. During training, the leaf node is assigned with a class and thus all data points that end up in a given leaf node belong to that class. Ending up in a leaf node with label y corresponds to a prediction $\hat{f}(\mathbf{x}_\star) = y$ for all input vectors $\mathbf{x}_\star$ that belong to this leaf node.

The decision tree partitions the input space into disjoint regions $R_1, R_2, ..., R_L$ with the number of regions corresponding to the number of leaf nodes. In the case where $\mathbf{x}_\star = [x_{\star 1}, x_{\star 2}]^\top$, the boundaries in the region partition are parallel with the x_1-axis or the x_2-axis and thus the regions are rectangles. For higher dimensions we have p-dimensional rectangles where p is the number of input variables.

2.2.2 Learning a Classification Tree

The prediction $\hat{f}(\mathbf{x}_\star)$ is a piecewise constant function that can be written as

$$\hat{f}(\mathbf{x}_\star) = \sum_{l=1}^{L} \hat{f}_l \mathbb{I}\{\mathbf{x}_\star \in R_l\}, \tag{1}$$

where L is the total number of leaf nodes, R_l is the lth region and $\hat{f}_l$ is the constant prediction in said region. The indicator function $\mathbb{I}$ takes the value 1 if $\mathbf{x}_\star \in R_l$, 0 otherwise. During learning, the aim is that the tree ends up with satisfactory values for the parameters in Equation (1). Firstly, the tree has to

have optimal splits which ultimately define the regions R_l. Secondly, it has to decide $\hat{f}_l$ for each region. In the binary classification case, it is typically decided by majority vote of the training data i.e., the most frequent class of each region decides the outcome. Lastly, the algorithm has to learn the size of the tree, defined through L, as opposed to the total number of nodes. This can also be interpreted as the algorithm having to learn when to stop splitting. Allowing a tree to continue splitting, until there are no data points left on which to base a new split, can result in the classifier overfitting to the training data which in turn leads to it not being able to generalise to new unseen data. However, we still want the regions to be selected so that the tree fits the training data to some extent.

The great number of possible ways to partition the input space makes exploration of all possible splits and construction of the resulting binary trees computationally impossible. Instead, we use the heuristic algorithm called recursive binary splitting. This algorithm is greedy; it will determine a splitting rule with the objective to obtain a model that explains the training data as well as possible while only taking the current split into consideration.

The split at any internal node is computed by solving an optimisation problem of the form

$$\min_{j,s} \left(n_1 Q_1 + n_2 Q_2 \right), \tag{2}$$

with n_1 and n_2 denoting the number of training data points in the left and right child of the current split, whereas Q_1 and Q_2 are the costs associated with the respective child node. Variables j and s denote, as previously mentioned, the index of the splitting variable, x_j (the jth component of $\mathbf{x}$), and the numeric threshold of an inner node. The variables in Equation (2) all depend on j and s, but this explicit dependence is not necessary for explaining how the algorithm works, and therefore they are dropped from our notation for cleaner expressions.

The proportion of training observations in region R_l that belong to class m is defined as

$$\hat{\pi}_{lm} = \frac{1}{n_l} \sum_{i:\mathbf{x}_i \in R_l} \mathbb{I}\{y_i = m\}, \tag{3}$$

with n_l the number of data points in R_l. For simplicity, we set $\hat{\pi}_{l1} = r$ in future expressions. In a binary classification problem where we have classes 0 and 1, Equation (3) then gives us

$$\hat{\pi}_{l1} = \frac{1}{n_l} \sum_{i:\mathbf{x}_i \in R_l} \mathbb{I}\{y_i = 1\} = r$$
$$\hat{\pi}_{l0} = \frac{1}{n_l} \sum_{i:\mathbf{x}_i \in R_l} \mathbb{I}\{y_i = 0\} = 1 - r.$$

If $r = 0$ or $r = 1$, we have node purity in leaf node l, meaning that all data points belong to the same class.

We now define three different splitting criteria, Q, called the misclassification rate (Eq. (4a)), Gini index (Eq. (4b)) and the entropy criterion (Eq. (4c)),

$$Q_{M,l} = 1 - \max\{r, 1-r\}, \tag{4a}$$
$$Q_{G,l} = 2r(1-r), \tag{4b}$$
$$Q_{E,l} = -r \ln r - (1-r)\ln(1-r). \tag{4c}$$

As illustrated in Figure 2, all three criteria provide zero loss if all data points belong to the same class, $r \in \{0, 1\}$, and maximal loss if the points are equally divided between the two classes, that is, $r = \frac{1}{2}$. For other values of r, the Gini index and the entropy both have higher loss than the misclassification rate. This means that they prioritize node purity and by extension, work well with the recursive binary splitting algorithm, that seeks to minimize loss. All three criteria can be used in binary splitting.

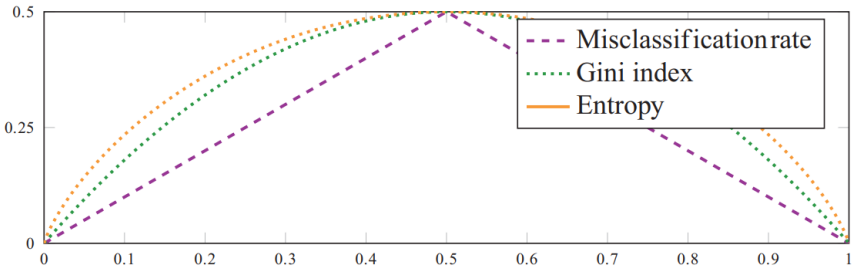

Figure 2: Figure from Lindholm et al. (2022). The three splitting criteria as functions of r. (The entropy criterion has been scaled such that it passes through $(0.5, 0.5)$.)

2.2.3 Overfitted Trees and How to Prevent it

If a tree model's depth is not restricted, it will continue to grow until each leaf node is pure, a so called fully grown tree. A fully grown tree will correctly classify all training data points, as per its definition, but it will not achieve optimal performance when presented with new, unseen data. Such a tree is said to be overfitted. To prevent overfitting, one can set a stopping criterion to stop splitting if the number of data points in a leaf node is less than some integer. Another way is to limit the depth of the tree. Lastly, one can grow the full tree and then prune it until a satisfactory depth is achieved - this method is appropriately named pruning.

2.2.4 The Bias-Variance Decomposition

This section is based on Section 4.4 in Lindholm et al. (2022).
Consider z_0 being the bullseye on an archery target. We then have an arrow, the random variable Z, which represents attempts to estimate z_0. Then a grouping

off-centre represents the systematic bias, while a scattering all over the target represents the variance. Mathematically, we define bias as

$$\mathrm{E}[Z] - z_0,$$

and variance as

$$\mathrm{E}[(Z - \mathrm{E}[Z])^2] = \mathrm{E}[Z^2] - \mathrm{E}[Z]^2.$$

In order to measure how good of an estimator Z is, we introduce the expected squared error between z_0 and Z, which can be defined in terms of squared bias and variance as

$$\begin{aligned}
\mathrm{E}[(Z - z_0)^2] &= \mathrm{E}[((Z - \mathrm{E}[Z]) + (\mathrm{E}[Z] - z_0))^2] \\
&= \mathrm{E}\left[(Z - \mathrm{E}[Z])^2\right] + 2\underbrace{(\mathrm{E}[Z] - \mathrm{E}[Z])}_{0}(\mathrm{E}[Z] - z_0) + (\mathrm{E}[Z] - z_0)^2 \\
&= \underbrace{\mathrm{E}\left[(Z - \mathrm{E}[Z])^2\right]}_{\text{Variance}} + \underbrace{(\mathrm{E}[Z] - z_0)^2}_{\text{Bias}^2}.
\end{aligned}$$

Noting that the expected squared error can be decomposed into the sum of squared bias and variance, we realise that in order to minimize the error, one has to consider both bias and variance; this is what is known as the bias-variance trade-off.

In the supervised learning setting, if expectation is conditional on $\mathbf{X}$, z_0 would be $\mathrm{E}[\mathrm{Y} \mid \mathbf{X}] = f(\mathbf{X}; \boldsymbol{\theta})$, while Z corresponds to predictions made from the fitted model $\hat{f}(\mathbf{X}, \boldsymbol{\theta}) = f(\mathbf{X}; \hat{\boldsymbol{\theta}}(\mathcal{T}))$, learned from training data $\mathcal{T}$. This implies that the mean squared estimation error $\mathrm{E}[(Z - z_0)^2]$ corresponds to the expected squared estimation error for estimating $f(\mathbf{X}; \boldsymbol{\theta})$ for a fixed $\mathbf{X}$. Furthermore, $\mathrm{E}[Z]$ would be the so called average training model $\mathrm{E}[f(\mathbf{X}; \hat{\boldsymbol{\theta}}(\mathcal{T}))] = \bar{f}(\mathbf{X})$, averaged over training data. The average training model is more of a concept rather than an actual model, as it would require re-training the model using an infinite amount of training sets of the same size and then computing the average. As previously mentioned, Z are predictions made by the learned model, but one can also view $Z = \hat{f}(\mathbf{X}; \boldsymbol{\theta})$ as a prediction of a future observation $f(\mathbf{X}; \boldsymbol{\theta}) + \varepsilon$ with a fixed response vector $\mathbf{X}$ and error term ε. In analogy to the estimation error case, one can then derive the mean squared prediction error in terms of variance and squared bias,

$$
\begin{aligned}
&\text{MSPE}(\mathbf{X}) \\
=&\text{E}[(f(\mathbf{X};\boldsymbol{\theta}) + \varepsilon - \hat{f}(\mathbf{X};\boldsymbol{\theta}))^2 \mid \mathbf{X}] \\
=&\text{E}[(\varepsilon + f(\mathbf{X};\boldsymbol{\theta}) - \hat{f}(\mathbf{X};\boldsymbol{\theta}))^2 \mid \mathbf{X}] \\
=&\text{E}[\varepsilon^2 + 2\varepsilon(f(\mathbf{X};\boldsymbol{\theta}) - \hat{f}(\mathbf{X};\boldsymbol{\theta})) + (f(\mathbf{X};\boldsymbol{\theta}) - \hat{f}(\mathbf{X};\boldsymbol{\theta}))^2 \mid \mathbf{X}] \\
=&\text{E}[\varepsilon^2 \mid \mathbf{X}] + \text{E}[(f(\mathbf{X};\boldsymbol{\theta}) - \hat{f}(\mathbf{X};\boldsymbol{\theta}))^2 \mid \mathbf{X}] \qquad (5) \\
=&\text{Var}(\varepsilon \mid \mathbf{X}) + \text{E}[(f(\mathbf{X};\boldsymbol{\theta}) - \bar{f}(\mathbf{X}) + \bar{f}(\mathbf{X}) - \hat{f}(\mathbf{X};\boldsymbol{\theta}))^2 \mid \mathbf{X}] \\
=&\text{Var}(\varepsilon \mid \mathbf{X}) + \text{E}[(f(\mathbf{X};\boldsymbol{\theta}) - \bar{f}(\mathbf{X}))^2 \mid \mathbf{X}] + \text{E}[(\bar{f}(\mathbf{X}) - \hat{f}(\mathbf{X};\boldsymbol{\theta}))^2 \mid \mathbf{X}] \\
=&\text{Var}(\varepsilon \mid \mathbf{X}) + \underbrace{(f(\mathbf{X};\boldsymbol{\theta}) - \bar{f}(\mathbf{X}))^2}_{\text{Bias}^2} + \underbrace{\text{E}[(\bar{f}(\mathbf{X}) - \hat{f}(\mathbf{X};\boldsymbol{\theta}))^2 \mid \mathbf{X}]}_{\text{Variance}}.
\end{aligned}
$$

In Equation (5), the expectation is over ε and $\mathcal{T}$ but this nested expectation is summarised in E for cleaner notation. We have also used that the expected value of ε is 0 in the third equality and the fact that the cross-term vanishes (as $\text{E}[\bar{f}(\mathbf{X}) - \hat{f}(\mathbf{X};\boldsymbol{\theta}) \mid \mathbf{X}] = 0$) in the fifth equality.

2.2.5 Bagging to Reduce Variance

This section is based on Section 7.1 in Lindholm et al. (2022). Bootstrap aggregation, or bagging for short, is an ensemble method. Ensemble methods reflect the saying "There is strength in numbers.", as they learn multiple models and then transform their individual outputs into one aggregated prediction. When bagging, we train the different models with independent samples $\tilde{\mathcal{T}}_1, ..., \tilde{\mathcal{T}}_B$, bootstrapped from the original training set $\mathcal{T}$. The bootstrapped samples are of the same size as the original training set. After training, we end up with B models, whose predictions $\hat{f}_b(\mathbf{x})$ are averaged to obtain one prediction,

$$
\hat{f}_{bag}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(\mathbf{x}). \qquad (6)
$$

In the binary classification problem, the average can be seen as an estimate of the conditional probability of class 1 given $\mathbf{x}$, or one can define a threshold function where class 1 is chosen if this conditional probability exceeds 0.5 and class 0 otherwise. The latter is equivalent to using the majority vote.

We can now use Equation (6) to derive the bias of the prediction,

$$\begin{aligned}
\text{Bias}(\hat{f}_{bag}(\mathbf{x})) &= \text{E}[\hat{f}_{bag}(\mathbf{x})] - f(\mathbf{x};\boldsymbol{\theta}) \\
&= \text{E}\left[\frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(\mathbf{x})\right] - f(\mathbf{x};\boldsymbol{\theta}) \\
&= \frac{1}{B}\text{E}\left[\sum_{b=1}^{B}\hat{f}_b(x)\right] - f(\mathbf{x};\boldsymbol{\theta}) \\
&= \left[\text{E}[\hat{f}_b(\mathbf{x})] \approx \text{E}[f(\mathbf{x};\hat{\boldsymbol{\theta}}(\mathcal{T}))] = \bar{f}(\mathbf{x})\right] \\
&\approx \frac{1}{B}B\bar{f}(\mathbf{x}) - f(\mathbf{x};\boldsymbol{\theta}) \\
&= \bar{f}(\text{x}) - f(\mathbf{x};\boldsymbol{\theta}).
\end{aligned}$$

So, bagging neither reduces nor increases the bias! We now derive the variance,

$$\begin{aligned}
\text{Var}(\hat{f}_{bag}(\mathbf{x})) &= \frac{1}{B^2}\text{Var}\left(\sum_{b=1}^{B}\hat{f}_b(\mathbf{x})\right) \\
&= \frac{1}{B^2}\left(\sum_{b=1}^{B}\text{Var}\left(\hat{f}_b(\mathbf{x})\right) + \sum_{b\neq b'}^{B}\text{Cov}\left(\hat{f}_b(\mathbf{x}), \hat{f}_{b'}(\mathbf{x})\right)\right) \\
&= \frac{\text{Var}(\hat{f}_b(\mathbf{x}))}{B} + \frac{1}{B^2}\underbrace{\sum_{b\neq b'}^{B}\text{Cov}\left(\hat{f}_b(\mathbf{x}), \hat{f}_{b'}(\mathbf{x})\right)}_{(*)}. \qquad (7)
\end{aligned}$$

Before continuing, we remind ourselves that the predictions $\hat{f}_b(\mathbf{x})$ are identically distributed. We also note that the total number of covariance terms is $B(B-1)$ and that correlation ρ is defined as

$$\rho = \frac{\text{Cov}\left(\hat{f}_b(\mathbf{x}), \hat{f}_{b'}(\mathbf{x})\right)}{\text{Var}(\hat{f}_b(\mathbf{x}))}$$

since all resampled estimates $\hat{f}_b(\mathbf{x})$ have the same variance. We can then rewrite $(*)$ as

$$\begin{aligned}
(*) &= B(B-1)\cdot\text{Cov}\left(\hat{f}_b(\mathbf{x}), \hat{f}'_b(\mathbf{x})\right) \\
&= B(B-1)\cdot\rho\cdot\text{Var}\left(\hat{f}_b(\mathbf{x})\right).
\end{aligned}$$

Substituting $(*)$ back into Equation (7) gives us

$$\begin{aligned}
\text{Var}\left(\hat{f}_{bag}(\mathbf{x})\right) &= \frac{\text{Var}(\hat{f}_b(\mathbf{x}))}{B} + \frac{1}{B^2}B(B-1)\cdot\rho\cdot\text{Var}\left(\hat{f}_b(\mathbf{x})\right) \\
&= \frac{\text{Var}(\hat{f}_b(\mathbf{x}))}{B} + \left(1-\frac{1}{B}\right)\cdot\rho\cdot\text{Var}\left(\hat{f}_b(\mathbf{x})\right). \qquad (8)
\end{aligned}$$

If the estimates $\hat{f}_b(\mathbf{x})$ are independent, the correlation term vanishes and we would get

$$\begin{aligned} \mathrm{Var}(\hat{f}_{bag}(\mathbf{x})) &= \mathrm{Var}\left(\frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(\mathbf{x})\right) \\ &= \frac{1}{B^2}B \cdot \mathrm{Var}(\hat{f}_b(\mathbf{x})) \\ &= \frac{\mathrm{Var}(\hat{f}_b(\mathbf{x}))}{B}. \end{aligned} \tag{9}$$

From Equation (9) we see that the variance of $\hat{f}_{bag}(\mathbf{x})$ will decrease as B approaches infinity. From Equation (8) we also see that when B approaches infinity, the variance can only decrease to a certain point, namely $\rho \cdot \mathrm{Var}\left(\hat{f}_b(\mathbf{x})\right)$, where ρ is the correlation. In summary, bagging is a method that reduces variance without increasing bias. To further reduce variance, one can consider a method called random forests, which aims at decorrelating the involved trees.

2.2.6 Random Forests

In this section, although random forests can be used for both regression and classification, random forests will be presented as a method to solve binary classification problems. This section is based on Section 7.2 in Lindholm et al. (2022).

Random forests is a modification of bagging and it involves building a large collection of de-correlated trees and then making a prediction by majority vote. Similar to bagging, the first step is to bootstrap independent samples $\tilde{\mathcal{T}}_1, ..., \tilde{\mathcal{T}}_B$ from the training set $\mathcal{T}$. We then grow the trees, but at each node we uniformly at random select m variables from the total of p variables. Then the optimisation problem, presented in Section 2.2.2, is solved but only while considering the m variables chosen as possible splits. This is repeated until the stopping criterion is reached. For random forests, we stop splitting if the number of data points in a node is less than $n_{\min}$. The result is again B estimates of f, who all cast their vote. The final prediction is the majority vote of the ensemble's votes (Hastie et al., 2009, Section 15).

It is the act of randomly selecting which variables be considered for splits that de-correlates the trees. If one input variable is particularly favourable for the greedy recursive binary splitting, chances are high that all the trees will have that split. But by randomly selecting which splitting variables are possible, we force the greedy algorithm to choose another split, resulting in weaker correlation between trees. It is worth noting that this also results in higher variance of the estimate $\hat{f}_b(\mathbf{x})$, computed from each individual tree, since none of these trees have access to the full dataset. However, experience has shown that the decrease in correlation outweighs the increase in variance and the net effect is often a reduction in the averaged prediction variance.

2.3 Using Neural Networks for Classification

Neural networks aim to learn the weights $\mathbf{w}$ in the network, which are then used to parametrize a function that takes input $\mathbf{x}$ and predicts the output y (Lindholm et al,. 2022, pp. 133).

2.3.1 Choosing a Loss Function

A loss function $\mathcal{L}(\hat{f}(\mathbf{x}_i), y_i)$ measures how close a model's prediction $\hat{f}(\mathbf{x}_i)$ is to the observed response variable y_i. If the model fits the data well, the value of the loss function is small.

When considering binary classification problems, one might intuitively reach for the misclassification loss:

$$\mathcal{L}(\hat{f}(\mathbf{x}_i), y_i) = \mathbb{I}\{\hat{f}(\mathbf{x}_i) \neq y_i\} = \begin{cases} 0 \text{ if } \hat{f}(\mathbf{x}_i) = y_i \\ 1 \text{ if } \hat{f}(\mathbf{x}_i) \neq y_i. \end{cases} \tag{10}$$

However, this loss function is sparsely used in actual model training since it is discrete and as a consequence, not differentiable with respect to the parameter vector $\boldsymbol{\theta}$. This vector $\boldsymbol{\theta}$ contains the elements of all weight matrices and offset vectors of the neural network, introduced in Section 2.3.3. Instead, we consider the cross-entropy loss for neural networks:

$$\mathcal{L}(\hat{f}(\mathbf{x}_i), y_i) = \begin{cases} \ln \hat{f}(\mathbf{x}_i) & \text{if } y_i = 1 \\ \ln(1 - \hat{f}(\mathbf{x}_i)) & \text{if } y_i = 0 \end{cases} \tag{11}$$

(Lindholm et al., 2022, pp. 98-99). The loss function for the whole training dataset can then be written as

$$\begin{aligned} J(\boldsymbol{\theta}) &= \mathcal{L}\left(\left\{\hat{f}(\mathbf{x}_i)\right\}_{i=1}^{n},\ \{y_i\}_{i=1}^{n}\right) \\ &= \frac{1}{n}\sum_{i=1}^{n}\left[y_i \ln \hat{f}(\mathbf{x}_i) + (1 - y_i)\ln(1 - \hat{f}(\mathbf{x}_i))\right] \end{aligned} \tag{12}$$

which is quite similar to the splitting criterion in Equation (4c). However, the difference is that the loss function in Equation (12) is averaging over the size of the dataset and has both the estimated label and the true label as parameters.

2.3.2 Gradient-Based Optimizers

Stochastic Gradient Descent From a differentiable loss function, one can calculate its gradient,

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\hat{f}(\mathbf{x}_i), y_i),$$

However, this proves to be computationally disadvantageous as summing takes time and using all the data points takes up a lot of memory. Instead, we

approximate the gradient by using a mini-batch, a subset of the entire dataset, of size $\tilde{n} < n$, so that

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{\tilde{n}} \sum_{i=1}^{\tilde{n}} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\hat{f}(\mathbf{x}_i), y_i).$$

This is called a stochastic gradient since the data points chosen for the mini-batch are random. The descent part simply refers to the fact that we update parameters in the negative direction of the gradient (Lindholm et al., 2022, pp. 124). The updating rule is as follows,

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}),$$

where γ is the learning rate. The learning rate tells us how much in the direction of the negative gradient we should update and its choice can be an optimisation problem in itself (Lindholm et al., 2022, pp. 117). However, in this thesis we use a constant learning rate.

Adaptive Moment Estimation An extension of stochastic gradient descents (SGD) is adaptive moment estimation (Adam) which is preferable when the loss function has many saddle points. Stochastic gradient descent risks getting stuck in saddle points since the gradients are zero there. Adam avoids this by calculating learning rates $\boldsymbol{\gamma}_t$ and search directions $\mathbf{d}_t$. The updating rule is then

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \boldsymbol{\gamma}_t \mathbf{d}_t,$$

where t is the iteration number and $\boldsymbol{\gamma}_t$ and $\mathbf{d}_t$ are outputs from functions $\gamma\left(\nabla_{\boldsymbol{\theta}} J_t, \ldots, \nabla_{\boldsymbol{\theta}} J_0\right)$ and $d\left(\nabla_{\boldsymbol{\theta}} J_t, \ldots, \nabla_{\boldsymbol{\theta}} J_0\right)$. The Adam optimizer updates the learning rates according to

$$\boldsymbol{\gamma}_t = \frac{\eta}{\sqrt{t}} \left((1-\beta_1) \mathrm{diag} \left(\sum_{i=1}^{t} \beta_1^{t-i} \|\nabla_{\boldsymbol{\theta}} J_i\|^2 \right) \right)^{\frac{1}{2}}$$

and the search direction according to

$$\mathbf{d}_t = (1-\beta_2) \sum_{i=1}^{t} \beta_1^{t-i} \nabla_{\boldsymbol{\theta}} J_i.$$

The parameters β_1 and β_2 are typically set to $\beta_1 = 0.999$ and $\beta_2 = 0.9$ while η is the initial learning rate (Lindholm et al., 2022, pp. 128).

2.3.3 Neural Networks

Artificial neural networks, or simply neural networks, are networks of interconnected units divided into input-, output- and hidden layers. Figure 3 illustrates a feedforward neural network with four input units (input layer), two hidden

layers and two output units (output layer). In a feedforward neural network, there are no loops, meaning that no unit can affect its own input. Each connection (represented by arrows between nodes in Figure 3) is associated with a real-valued weight that decides how much a unit will be affected by each unit of the previous layer. The unit is semi-linear as it computes the weighted sum of its input (a linear operation) and then applies a non-linear activation function to the result. If the activation function was linear, we would simply have a linear transformation of the input over and over again, which we can summarize as a single linear function, resulting in a neural network equivalent to a network without hidden layers.

A neural network typically learns through a combination of backpropagation and optimization. The backpropagation algorithm is used after a forward pass. While making a backward pass, the partial derivative for each weight is computed as,

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial h} \frac{\partial h}{\partial z} \frac{\partial z}{\partial w},$$

where $\mathcal{L}$ is the loss function, z is the value of a node and $h(z)$ is the activation function. The loss can then be minimized by using stochastic gradient descent for example (Sutton et al., 2018, Section 9.7).

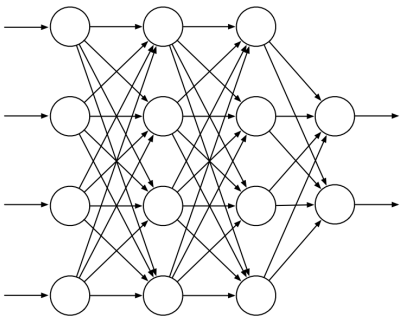

Figure 3: Figure from Sutton et al. (2018). A feedforward neural network with one input layer, two hidden layers and one ouput layer.

Suppose we have a neural network for a binary classification problem, with $L = 4$ layers: one input layer, two hidden layers and one output unit. We can then define the output of the input layer as

$$\mathbf{q}_1 = \mathbf{x}.$$

In layer two and three, the input is the output of the previous layer and we get

$$\mathbf{q}_2 = h(\mathbf{w}_1 \mathbf{q}_1 + \mathbf{b}_1)$$
$$\mathbf{q}_3 = h(\mathbf{w}_2 \mathbf{q}_2 + \mathbf{b}_2),$$

where h is the sigmoid activation function defined as

$$h(z) = \frac{1}{1 + e^{-z}},$$

and $\mathbf{b}_l$ is the offset vector of layer $l+1$. The matrix $\mathbf{w}_l$ contains the weights of the connections between layer l and layer $l+1$, $l \in \{1, ..., L-1\}$.

Another choice for the activation function is the rectified linear unit (ReLU) defined as

$$h(z) = \max(0, z).$$

For the final layer, the output is a scalar which represents the probability that a data point belongs to class 1,

$$q_4 = h(\mathbf{w}_3\mathbf{q}_3 + b_3) \in [0, 1].$$

So, in a neural network with a total of L layers, the final output q_L is not a prediction, but the probability that the output variable Y belongs to class 1. The threshold to decide the prediction is set to 0.5 and we get

$$\hat{f}(\mathbf{x}) = \begin{cases} 0, & \text{if } q_L \leq 0.5 \\ 1, & \text{otherwise} \end{cases}$$

(Lindholm et al., 2022, Section 6.1).

2.4 Classification Error in Binary Classification Problems

In Equation (5) of Section 2.2.4, we defined the expected squared prediction error MSPE($\mathbf{X}$) conditioned on $\mathbf{X}$. In the simulation study however, we should use the integrated expected mean squared prediction error (MSPE) which can be decomposed into a variance term and a mean squared estimation error term (MSE) as

$$\begin{aligned} \text{MSPE} &= \text{E}[\text{MSPE}(\mathbf{X})] \\ &= \text{E}[(f(\mathbf{X};\boldsymbol{\theta}) + \varepsilon - \hat{f}(\mathbf{X};\boldsymbol{\theta}))^2] \\ &= \text{Var}(\varepsilon) + \underbrace{\text{E}[(f(\mathbf{X};\boldsymbol{\theta}) - \bar{f}(\mathbf{X}))^2]}_{\text{Bias}^2} + \underbrace{\text{E}[(\bar{f}(\mathbf{X}) - \hat{f}(\mathbf{X};\boldsymbol{\theta}))^2]}_{\text{Variance}} \\ &= \text{Var}(\varepsilon) + \text{MSE}, \end{aligned} \tag{13}$$

since $\mathbf{X}$ will vary. At this point, the reader has noticed that MSPE can be used both when Y is a continuous variable and when it is a discrete, binary variable. This of course begs the question: is there a more adequate error for the binary classification problem?

To find such an error, we first introduce the Bayes classifier,

$$\hat{\text{Y}} = \mathbb{1}(\hat{f}(\mathbf{x}) > 0.5), \tag{14}$$

which is a prediction of the response variable Y for an input vector $\mathbf{x}$. The classification error is then the probability that the prediction is wrong. In the ideal case, when the regression function is known, i.e. $\hat{f}(\mathbf{x}) = f(\mathbf{x})$, we would get the classification error

$$\begin{aligned}\text{Err}(\mathbf{x}) &= P(\text{Y} \neq \hat{\text{Y}} \mid \mathbf{X} = \mathbf{x}) \\ &= 1 - \max(1 - f(\mathbf{x}; \boldsymbol{\theta}), f(\mathbf{x}; \boldsymbol{\theta})).\end{aligned}$$

We then have the unconditioned error

$$\begin{aligned}\text{Err} &= \text{E}[\text{Err}(\mathbf{X})] \\ &= 1 - \text{E}[\max(1 - f(\mathbf{X}; \boldsymbol{\theta}), f(\mathbf{X}; \boldsymbol{\theta}))],\end{aligned}$$

(James et al., 2023, Section 2.2).

In reality, we rarely, if ever, have training data such that $\hat{f}(\mathbf{x}) = f(\mathbf{x})$ so we condition on the training data and get

$$\text{Err}(\mathbf{x} \mid \mathcal{T}) = \begin{cases} 1 - \text{Y}, & \text{if } \hat{f}(\mathbf{x}) > 0.5 \\ \text{Y}, & \text{if } \hat{f}(\mathbf{x}) < 0.5. \end{cases}$$

Taking the expectation with respect to Y and the training data $\mathcal{T}$, and using the law of total expectation, yields

$$\begin{aligned}\text{Err}(\mathbf{x}) &= \text{E}[\text{Err}(\mathbf{x} \mid \mathcal{T})] \\ &= (1 - f(\mathbf{x}))P(\hat{f}(\mathbf{x}) > 0.5) + f(\mathbf{x})P(\hat{f}(\mathbf{x}) < 0.5). \qquad (15)\end{aligned}$$

Taking the expectation of Equation (15) with respect to $\mathbf{X}$ gives us

$$\begin{aligned}\text{Err} &= \text{E}[\text{Err}(\mathbf{X})] \\ &= \text{E}[(1 - f(\mathbf{X}))P(\hat{f}(\mathbf{X}) > 0.5 \mid \mathbf{X})] + \text{E}[f(\mathbf{X})P(\hat{f}(\mathbf{X}) < 0.5 \mid \mathbf{X})]. \qquad (16)\end{aligned}$$

We have by now introduced four different ways to quantify errors, MSPE($\mathbf{x}$), MSPE, Err($\mathbf{x}$) and Err (Equation (5), (13), (15) and (16) respectively).

3 Data Generation and Simulation Scenarios

In this thesis we conduct three different simulations in order to compare the performances of the chosen methods with respect to different aspects of binary classification. In the first simulation we vary the dimension of the input data while in the second simulation we vary the size of the training data. Lastly, in the third simulation we compare the methods' performances by varying the probability that a data point has a flipped label. The measurements for performance are both MSPE and Err.

The simulations are done in Python with packages Numpy and Sklearn while Matplotlib.pyplot was used for visualization. The technique for achieving a non-linear relationship between $\mathbf{X}$ and Y was inspired by Markus Söderqvist's bachelor's thesis (Söderqvist, 2024).

3.1 Generating the Data

We introduce δ as the probability that a label is flipped. For some $0 \leq \delta < 0.5$, it means that a perfect classifier, an oracle with knowledge of $f(\mathbf{x})$ for all $\mathbf{x}$, will still misclassify a fraction δ of the data.

We start by generating $\mathbf{X}$ from the multivariate normal distribution with mean vector $\mathbf{0}$ ($\dim p$) and the covariance matrix being the identity matrix ($\dim p \times p$). To achieve a non-linear relationship between $\mathbf{X}$ and Y we define an "archery target" which is a shape with concentric circles. We classify observations in even numbered rings as class 0 and observations in the odd numbered rings as class 1. We then flip the label of each observation with probability δ to obtain Y. See Figure 4 for an example of how Y is generated from $\mathbf{X} = (x_1, x_2)$ with ten concentric circles. For $\mathbf{X}$ with dimension $p > 2$, the archery rings become p-dimensional spheres.

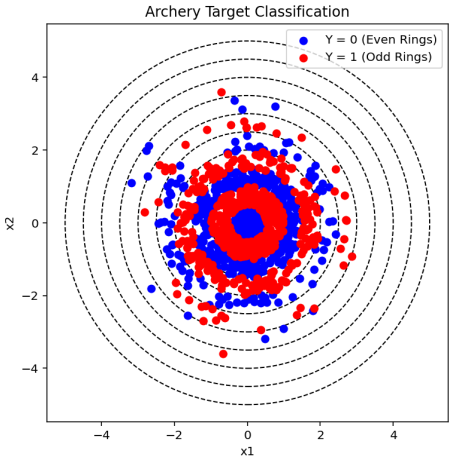

Figure 4: Simulated data with 1000 data points, $p = 2$, $\delta = 0$ and the radius increasing by 0.5 per ring (the innermost ring has radius 0.5).

3.2 Expressing Errors with the Label Flipping Probability

In simulations with hyperparameter δ, both Err and MSPE have explicit expressions. Let us call the entire archery target Ω where the even rings make up the region Ω_{even} and the odd rings make up the region Ω_{odd}, i.e. $\Omega_{even} \cup \Omega_{odd} = \Omega$. Then after flipping labels with probability δ, the conditional probability that $\mathrm{Y} = 1$ for an input vector $\mathbf{x}$ is

$$f(\mathbf{x}; \boldsymbol{\theta}) = \begin{cases} 1 - \delta, \text{ if } \mathbf{x} \in \Omega_{odd} \\ \delta, \text{ if } \mathbf{x} \notin \Omega_{odd}. \end{cases}$$

From the training data, we get $\hat{\Omega}_{odd} = \{\mathbf{x}; \hat{f}(\mathbf{x}) > 0.5\}$ and $\hat{\Omega}_{even} = \{\mathbf{x}; \hat{f}(\mathbf{x}) < 0.5\}$.

We then define the region $D = \Omega \triangle \hat{\Omega}$, which is the symmetric difference between sets Ω and $\hat{\Omega}$. We can then show that $\text{Err}(\mathbf{x})$ is

$$\text{Err}(\mathbf{x}) = \begin{cases} \delta, \text{ if } \mathbf{x} \notin D \\ 1 - \delta, \text{ if } \mathbf{x} \in D. \end{cases}$$

Taking the expectation of $\text{Err}(\mathbf{x})$ gives us

$$\begin{aligned} \text{Err} &= \text{E}[\text{Err}(\mathbf{X})] \\ &= \delta(1 - P(\mathbf{X} \in D)) + (1 - \delta)P(\mathbf{X} \in D) \\ &= \delta + (1 - 2\delta)P(\mathbf{X} \in D). \end{aligned}$$

We note that an oracle (with $D = \varnothing$) will never be able to achieve an Err smaller than δ.

For the MSPE, we first calculate the variance of ε,

$$\begin{aligned} \text{Var}(\varepsilon) &= \text{E}[\text{Var}(\varepsilon \mid \mathbf{X})] \\ &= \text{E}[\text{Var}(\text{Y} - f(\mathbf{X}; \boldsymbol{\theta})) \mid \mathbf{X}] \\ &= \text{E}[\text{Var}(\text{Y} \mid \mathbf{X})] \\ &= \text{E}[f(\mathbf{X}; \boldsymbol{\theta})(1 - f(\mathbf{X}; \boldsymbol{\theta}))] \\ &= \text{E}\left[\begin{cases} \delta(1 - \delta), \text{ if } \mathbf{X} \notin D \\ (1 - \delta)(1 - (1 - \delta)), \text{ if } \mathbf{X} \in D \end{cases}\right] \\ &= \text{E}[\delta(1 - \delta)] \\ &= \delta(1 - \delta). \end{aligned}$$

We then get the following expression for MSPE from Equation (13),

$$\text{MSPE} = \delta(1 - \delta) + \text{MSE}.$$

As such, the smallest possible MSPE will be equal to $\delta(1-\delta)$. For example, if δ is set to 0.05, the smallest errors achievable are $\text{Err}_{oracle} = 0.05$ and $\text{MSPE}_{oracle} = 0.0475$.

3.3 Simulations: Increasing Input Dimension, Training Set Size and Label Flipping Probability

In the first simulation, we simulate 100 samples of size 1000 with dimension of the input data $p = 2, 3, 5$, where p thus denotes the number of parameters. Each sample of size 1000 is divided into 80% training data (800 observations) and 20% test data (200 observations). We also have a radius increase of 0.5 per ring, ten rings and label flipping probability $\delta = 0.05$.

For random forest, we have 200 trees, a max depth of seven, the minimum number of samples required for a split is four and the number of features taken into consideration before a split is $m = \lfloor \sqrt{p} \rfloor$ (Hastie et al., 2009, pp. 592).

For neural network, we have nine hidden layers, the first seven are of size 64 and the last two are of size 32. The batch size is set to 32. For activation function we use the sigmoid function and ReLU function paired with stochastic gradient descent and Adam, respectively.

In the second simulation, we use one dataset from the first simulation as well as the same hyperparameter values. What we vary is the size of the training dataset. First, we randomly split the set into subsets containing 80% training data and 20% test data. This is done 50 times. Then we train the models using different proportions of the training data, the lowest being 10%, 80 data points and the highest being 100%, 800 data points. The classification error is then calculated by averaging over 50 splits.

For the third simulation we simplify the boundaries by reducing the number of rings from ten to four by changing the radius increase to 1 per ring. We are interested in comparing the models' performances for different values of the label flipping probability δ and only consider the 2-dimensional input data.

The simulations were conducted using functions from an existing package, with all unspecified hyperparameters set to their default values. The classification error and mean squared prediction error were calculated for each sample.

4 Results

4.1 Simulation 1: Increasing the Input Dimension

In Figure 5 we see that for $p = 2$ input variables, the neural network with ReLU and Adam has the smallest classification error and mean squared prediction error. It also achieves the lowest errors out of the three methods with $p = 3$ input variables, as shown in Figure 6. For $p = 5$ input variables, visualised in Figure 7, it performs only slightly better when looking at the classification error but much worse than the other two methods if comparing their MSPEs. It is also the method with the highest MSPE variance throughout the different number of input variables.

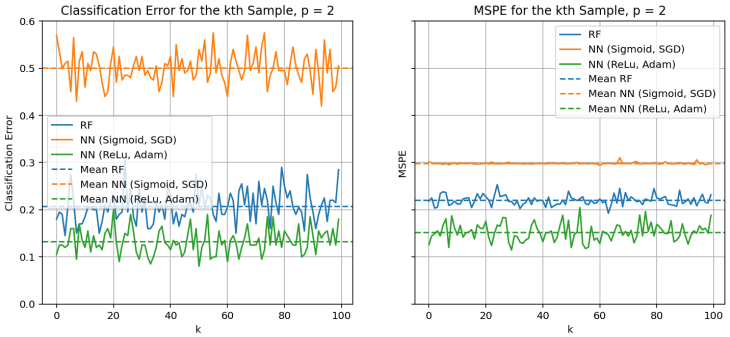

Figure 5: Classification errors and mean square prediction errors for 100 samples with $p = 2$ input variables and ten concentric circles to define the two classes.

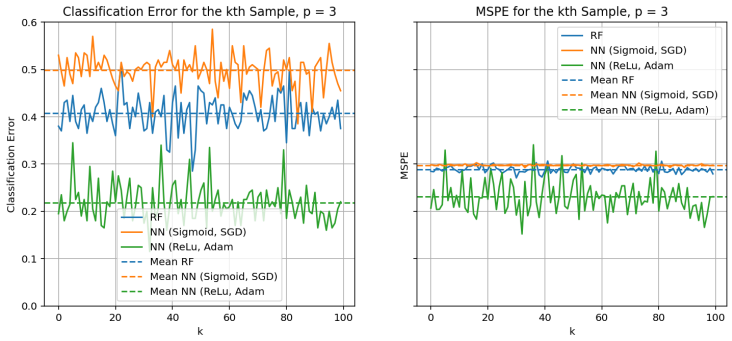

Figure 6: Classification errors and mean square prediction errors for 100 samples with $p = 3$ input variables and ten concentric spheres to define the two classes.

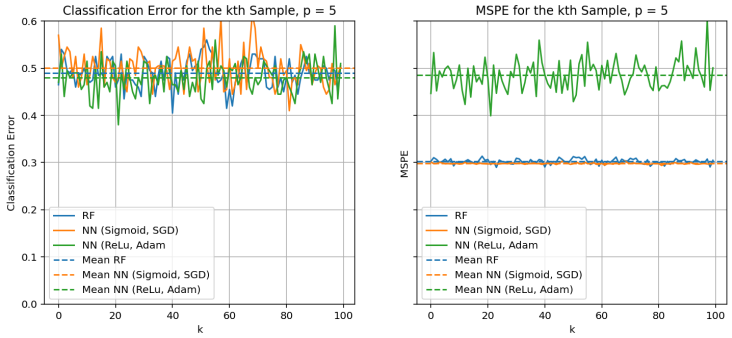

Figure 7: Classification errors and mean square prediction errors for 100 samples with $p = 5$ input variables and ten concentric five-dimensional spheres to define the two classes.

4.2 Simulation 2: Increasing the Training Set Size

In Figure 8 we see that the neural network with sigmoid activation function and stochastic gradient descent is not learning anything for this problem. Both random forest and neural network with ReLU and Adam are generalising to some degree as the blue curves are non-zero and the orange curves are not at 0.5. In Figure 9 we observe that when presented with a higher dimensional problem, the neural network with ReLU and Adam performs better than the other methods as seen in the decrease of the test score as the training set gets larger. While the test score for random forest is also decreasing, the incline is not as steep. In Figure 10, all methods fail to generalise as all three test score curves have flatlined at 0.5.

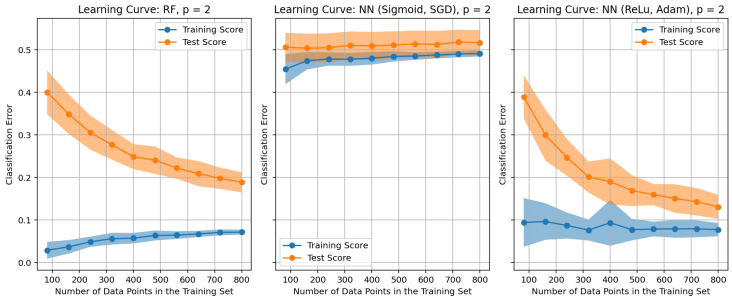

Figure 8: Learning curves with regard to the size of the training set ($p = 2$ input variables and ten concentric circles to define the two classes). Shaded error bands represent ± 1 standard deviation.

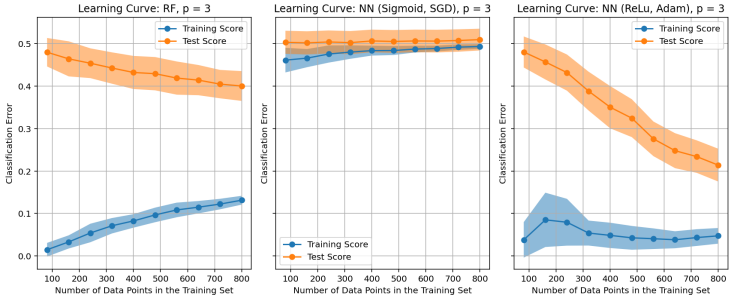

Figure 9: Learning curves with regard to the size of the training set ($p = 3$ input variables and ten concentric spheres to define the two classes). Shaded error bands represent ± 1 standard deviation.

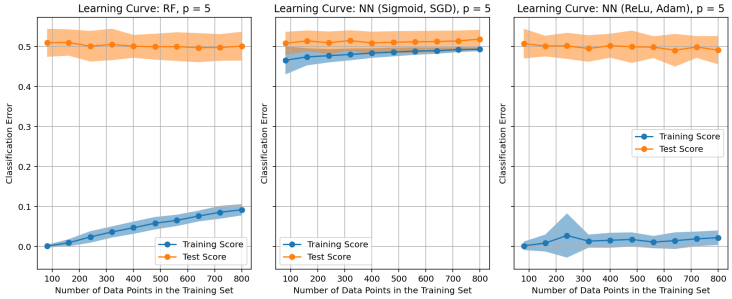

Figure 10: Learning curves with regard to the size of the training set ($p = 5$ input variables and ten concentric five-dimensional spheres to define the two classes). Shaded error bands represent ± 1 standard deviation.

4.3 Simulation 3: Increasing the Label Flipping Probability

In Figure 11, we see once again that neural network with sigmoid activation function and optimizer SGD is not learning anything meaningful. As for the random forest, its classification error is only slightly larger than the neural network with ReLU and Adam's, up until the flipping probability is around 0.3, after which the two methods perform almost equally. The MSPEs are relatively similar as well.

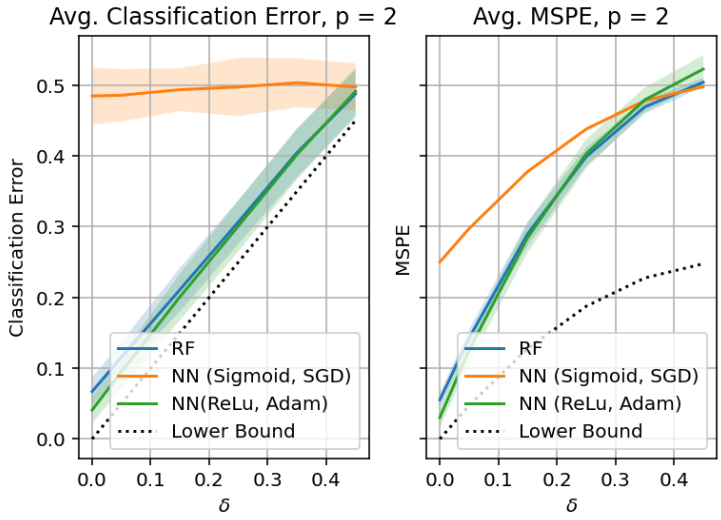

Figure 11: Classification error and MSPE for $p = 2$ input variables, four concentric circles to define the two classes and different values of the label flipping probability δ, averaged over 100 simulations.

5 Discussion

5.1 Analysing the Differences in Performance

If we define the best classifier as the one achieving the smallest classification errors, one could say that the neural network with ReLU activation function and Adam optimizer is the best method for classifying data points in alternating concentric circles since it has the lowest classification errors in Figures 5-9. For five input variables, it has a higher MSPE than the other two methods (Figure 10), but we keep in mind that the methods all had sufficiently well-tuned hyperparameters for two input variables so a decrease in performance in higher dimensions is to be expected. An interesting aspect of the neural network with ReLU and Adam, is that its MSPE has the highest variance throughout simulation 1. However, this has little to do with the method itself and is probably a

consequence of the hyperparameter settings chosen for this task. The high variance of the MSPE in Figures 8-10 is an indicator that the neural network with ReLU and Adam is, to some extent, overfitting. The reason for the high MSPE variance across simulations for the neural network classifier (with ReLU and Adam) could therefore be that many parameters are estimated for this method, but not to the extent that a large degree of overfitting takes place. A reason for this could be that the test data is sampled from the same distribution as the training data.

The worst performing model is the one learned through the neural network with sigmoid and stochastic gradient descent. In all three simulations, it fails to achieve a classification error below 0.45, which means it is basically guessing which class a data point belongs to. The cause for this poor performance could be the choice of optimizer, activation function or the combination. The sigmoid function's derivative with regard to z is

$$\frac{\partial h}{\partial z} = \frac{e^{-z}}{(1+e^{-z})^2}.$$

The derivative takes values between 0 and 0.25 but more importantly, it is 0 for both large negative numbers and positive numbers. This might be causing the updates to stagnate since the gradient is 0 too often in the gradient with partial derivatives

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial h}\frac{\partial h}{\partial z}\frac{\partial z}{\partial w}.$$

It is possible that Adam would perform better than SGD in combination with the sigmoid function since it has the property of adapting the learning rate for each weight.

In simulation 2, we focus more on which method is better at effectively using the training data by varying the size of the training set. In Figure 8 we see that random forest has a relatively low training score although it is slightly increasing with the number of samples in the training set. At the same time, the error band on the blue curve implies that the training score's variance is decreasing which is not that surprising since larger training set makes for a better fit. It seems from simulation 2 that both random forest and neural network (with ReLU and Adam) are generalising to some degree, although not very well for small training sets. As the number of data points increase, the two curves are closing the gap between them which is a sign of better generalisation for larger training sets. We also interpret from the training score curves that random forest is struggling to find a pattern when presented with more training data points while the neural network is consistent in fitting the training data no matter the size. This indicates that random forest has more problems with overfitting when the size of the training data set is small.

Increasing the number of input variables from two to five rends all methods useless; as seen in Figure 10, none of them are learning anything from the data. As previously mentioned, this is not strange at all since the hyperparameters are chosen to work well in the two-dimensional case. However, throughout simulation 2, the training scores for random forest and neural network with ReLU and

Adam are behaving similarly, despite the increase in dimension. This suggests that both random forests and neural networks are good at fitting training data, no matter its dimension, and the difference between the methods must then lie in the amount of overfitting on said training data. When comparing Figure 8 and Figure 9, we note that the neural network (with ReLU and Adam) has only a slightly steeper training curve than random forest in the two-dimensional case, while the difference in incline is more pronounced in the three-dimensional case. As a consequence, the gap between the training and test curves closes more quickly for the neural network (with ReLU and Adam) than for random forest. These observations imply that not only is the neural network better at finding patterns in smaller sets than random forest, it also generalises better to higher dimensions.

In simulation 3, the left plot in Figure 11 implies that no method is better than the others when the label flipping probability is higher. Perhaps even more surprising is that none of the two models that learned (neural network with ReLU and random forest), significantly outperformed the other when we did not mix the classes at all ($\delta = 0$). The neural network, represented by the green graph, only slightly outperformed the random forest. This could be interpreted as both methods being suitable for this task. As the label flipping probability increases the classification errors approach 0.5. This is not surprising, as the data points have basically lost any pattern of separation between the two classes if there is a 50% chance of flipping class. This also explains the methods' poor MSPEs shown on the right hand side in Figure 11. As δ increases, the two classifiers struggle to find a pattern for the two classes, and therefore the classifiers are essentially guessing the labels.

5.2 How the Data is Distributed in the Rings

In the simulations, we sampled vector $\mathbf{X}$ of dimension p from the multivariate normal distribution $\mathcal{N}(\mathbf{0}, \mathbb{1})$. Furthermore, the magnitude $||\mathbf{X}||$ is independent of the direction $\frac{\mathbf{X}}{||\mathbf{X}||}$ due to the rotational invariance of the Gaussian distribution. This rotational invariance also leads to all directions being uniformly distributed on the p-dimensional unit sphere. It follows from the multivariate distribution that the squared Euclidean distance $||\mathbf{X}||^2$ between the data point and the midpoint of the concentric circles, is chisquare distributed with p degrees of freedom. This makes it possible to calculate the probability that a data point ends up in a specific ring as well as the probability density function of the square root of the chisquare distribution using the Python package Scipy.stats.

Table 1: The probabilities of a data point in $\mathbb{R}^p$ lying between two concentric p-dimensional spheres, rounded to two decimals, with a total of ten concentric spheres. In bold is the highest probability in dimension p.

	$\mathbf{X} \in \mathbb{R}^2$	$\mathbf{X} \in \mathbb{R}^3$	$\mathbf{X} \in \mathbb{R}^5$
$P(0 < \|\|\mathbf{X}\|\| < 0.5)$	0.12	0.03	0.00
$P(0.5 < \|\|\mathbf{X}\|\| < 1)$	**0.28**	0.17	0.04
$P(1 < \|\|\mathbf{X}\|\| < 1.5)$	**0.28**	**0.28**	0.15
$P(1.5 < \|\|\mathbf{X}\|\| < 2)$	0.19	0.26	0.26
$P(2 < \|\|\mathbf{X}\|\| < 2.5)$	0.09	0.16	**0.27**
$P(2.5 < \|\|\mathbf{X}\|\| < 3)$	0.03	0.07	0.17
$P(3 < \|\|\mathbf{X}\|\| < 3.5)$	0.01	0.02	0.08
$P(3.5 < \|\|\mathbf{X}\|\| < 4)$	0.00	0.01	0.02
$P(4 < \|\|\mathbf{X}\|\| < 4.5)$	0.00	0.00	0.01
$P(4.5 < \|\|\mathbf{X}\|\| < 5)$	0.00	0.00	0.00
$P(5 < \|\|\mathbf{X}\|\|)$	0.00	0.00	0.00

Table 2: The probabilities of a data point in $\mathbb{R}^p$ lying between two concentric p-dimensional spheres, rounded to two decimals, with a total of four concentric spheres. In bold is the highest probability in dimension p.

	$\mathbf{X} \in \mathbb{R}^2$	$\mathbf{X} \in \mathbb{R}^3$	$\mathbf{X} \in \mathbb{R}^5$
$P(0 < \|\|\mathbf{X}\|\| < 1)$	0.39	0.20	0.04
$P(1 < \|\|\mathbf{X}\|\| < 2)$	**0.47**	**0.54**	0.41
$P(2 < \|\|\mathbf{X}\|\| < 3)$	0.12	0.23	**0.44**
$P(3 < \|\|\mathbf{X}\|\| < 4)$	0.01	0.03	0.10
$P(4 < \|\|\mathbf{X}\|\|)$	0.00	0.00	0.01

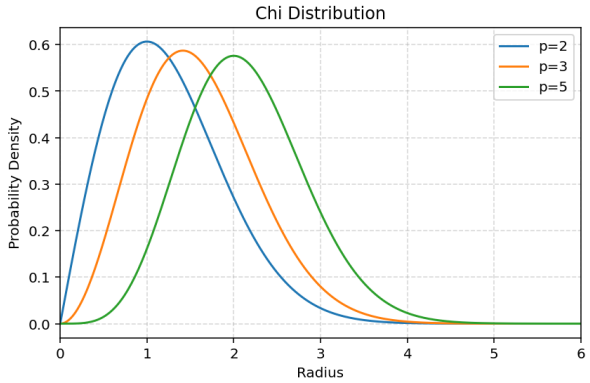

Figure 12: Probability density function of the square root of a chisquare distribution with degrees of freedom $p = 2, 3, 5$.

Intuitively, as the dimension of a vector increases, so does its distance to the origin as we sum more components. This intuition is backed by the probabilities in Table 1 and 2 as well as in Figure 12 that show a trend of the data points "migrating" to the outer regions, leaving our archery target completely for a sufficiently large dimension p. The fact that our data points behave like this enforces the previous statement that the sigmoid function is unsuitable for this task. As p increases, the derivative's value approaches 0, leading to SGD being unable to update the weights of the network.

5.3 Improvements and Further Work

In Figures 8-10, we notice a "bump" in the training curves of the neural network with ReLU activation and Adam optimizer. Since the classification error is fairly low as the number of data points increases, it seems unreasonable that the bias is causing the bump. Instead, since the neural network with ReLU has many parameters to estimate, it is probably the variance that is the dominating factor, which decreases as the model has more training data to go on. Nevertheless, further investigation is required to confirm this hypothesis.

As previously mentioned in Section 5.1, the neural network (with ReLU and Adam) achieves the lowest MSPE in simulation 2 for input dimensions $p = 2$ and $p = 3$, but the highest MSPE for $p = 5$. Specifically, the neural network's MSPE increases from approximately 0.15 to 0.48. In comparison, random forest exhibits a slower increase, as its MSPE goes from approximately 0.22 to 0.3. As such, it would be of interest to investigate why the MSPE of the neural network increases faster than the MSPE of random forest as the dimension of the input increases.

The classifiers' performances with a label flipping probability $\delta = 0.05$ are underwhelming. But most likely this is not due to a fault in the methods random forest and neural network themselves but rather in the hyperparameter tuning. If time had allowed, a more thorough method for tuning would perhaps have made it possible to change the hyperparameters between each scenario (i.e. when adding more parameters and increasing the label flipping probability) and compare the methods when at their best. One time-consuming method for finding better hyperparameters is grid-search. Grid-search trains the model with all different kinds of combinations of hyperparameters and returns the best settings. However, it is not guaranteed that grid-search results in the lowest classification errors since the set of hyperparameter combinations to choose from are decided by the programmer.

Another aspect that could potentially improve the performance is the addition of a "radial distance"-feature. This one-dimensional input variable would simply be a data point's distance to the centre point of the concentric circles and it is enough to figure out the pattern of the concentric circles. On the other hand, this distance makes the remaining $p - 1$ input variables of the original input vector superfluous as they themselves do not provide any further information about the location of the two classes. The simulations in this thesis do not provide an answer to if any of the methods used in this thesis is better at

learning that $p-1$ degrees of freedom of the input vector are of no use. Another point of analysis would then be to investigate which method is best at ignoring unnecessary aspects of data. However, it is unlikely that the performance of the neural network with sigmoid activation function and SGD would improve if provided with the radial distance. As shown in Table 1, for $p = 2$, most input vectors will be found around one unit of distance from the origin and the derivative of the sigmoid function assumes the rather small value $h'(||\mathbf{x}||) \approx 0.2$. As the distance increases, the derivative then approaches zero, causing weight updates to stagnate during training. Nevertheless, further investigation is needed to substantiate this claim.

Lastly, while this thesis' focus was on the comparison of two supervised learning methods, it would be interesting to only focus on classifying data points defined by the alternating concentric circle pattern and find the method best suited for finding these boundaries. Another possibility would be to try support vector machines. This method, much like neural networks, projects the data points from $\mathbf{R}^p$ to a space of higher dimension in which the boundary between the two classes is easier to find.

6 References

Boateng, A., Aidoo, E. N., Maposa, D., Odoom, C., & Owusu, S. A. (2024). *Optimizing binary classification performance in neural networks through simulation: A comparative study of activation functions.* Retrieved May 2, 2025, from https://researchportal.hw.ac.uk/en/publications/optimizing-binary-classification-performance-in-neural-networks-t.

Chen, L. (2021). *Supervised learning for binary classification on US adult income.* Retrieved May 2, 2025, from https://www.researchgate.net/publication/357058950.

Chen, X., & Zhang, X. (2024). *Optimal weighted random forests.* Retrieved May 2, 2025, from https://dl-acm-org.ezp.sub.su.se/doi/10.5555/3722577.3722897.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer.

James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). *An introduction to statistical learning: With applications in Python* (1st ed.). Springer.

Lindholm, A., Wahlström, N., Lindsten, F., & Schön, T. B. (2022). *Machine learning: A first course for engineers and scientists.* Cambridge University Press.

Mostafa, F. B., & Hasan, M. E. (2021). *Machine learning approaches for binary classification to discover liver diseases using clinical data.* Retrieved May 2, 2025, from https://arxiv.org/abs/2104.12055.

Müller, A. C. (2020). *Random forests — Applied machine learning in Python.* Retrieved April 24, 2025, from https://amueller.github.io/aml/02-supervised-learning/09-random-forests.html.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). Bradford Books.

Söderqvist, M. (2024). *Predictive performance of AdaBoost and random forest in binary classification tasks* [Bachelor's thesis, Stockholm University]. File name: 2024_5_report.pdf.
Retrieved from https://kurser.math.su.se/mod/folder/view.php?id=13969.