Stockholms universitet

A comparative study of binary classification performance with Logistic Regression, Support Vector Machines and Artificial Neural Networks

Liliya Trila

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2024:2
Matematisk statistik
Januari 2024

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Mathematical Statistics
Stockholm University
Bachelor Thesis **2024:2**
http://www.math.su.se

A comparative study of binary classification performance with Logistic Regression, Support Vector Machines and Artificial Neural Networks

Liliya Trila*

January 2024

Abstract

Binary classification is a common task in machine learning, where the goal is to categorize data into one of two classes. In this thesis we compare three methods for binary classification: logistic regression, support vector machines and artificial neural networks. The aim of this project is to understand the similarities and differences between these methods.

In the first part of the project, we present the theory related to each method. We also do a theoretical comparison of the methods.

In the second part of this project we compare these methods on both simulated and real-world data. To evaluate the performance of each method we use the performance metrics accuracy and AUC. The results show that logistic regression and support vector classifiers have very similar performance when the data is linearly separable. The performance of artificial neural networks tend to be slightly lower for small and high-dimensional datasets. When the data is not linearly separable artificial neural network models tend to perform slightly better on lower-dimensional data while support vector machines with an RBF kernel perform slightly better on higher-dimensional data. But overall, their performance is quite comparable.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: liliatrila@gmail.com. Supervisor: Taras Bodnar and Johannes Heiny.

Acknowledgement

I would like to thank my supervisors Taras Bodnar and Johannes Heiny for their valuable feedback during the writing of this thesis.

In this project, ChatGPT was used to gain a better understanding of certain topics and, in some cases, to improve grammar.

Contents

1 Introduction

Machine Learning is a branch of computer science and artificial intelligence that focuses on developing algorithms that are capable of learning meaningful patterns from data and making predictions when presented with new, unseen data. One common task in ML is classification, which involves categorizing data into different classes or categories based on some features. In binary classification, the goal is to categorize data into one of two classes.

In this thesis, we compare three methods for binary classification: Logistic Regression, Support Vector Machines, and Artificial Neural Networks.

Logistic regression (LR) originated in classical statistics. When using LR, the goal is not only to obtain a good predictive model but also to explain the relationship between variables [9].

Support vector machine (SVM) is a method that originated in the field of machine learning. It takes a more geometrical approach to data classification [13].

Artificial neural networks (ANN) also originated in the field of machine learning. These models were developed to mimic the learning process of biological neurons [6].

Numerous studies have compared these methods, suggesting that artificial neural network models tend to outperform both logistic regression and support vector machines. However, there are also studies where this is not the case [3] [12] [11] [8] [2].

The aim of this project is to explore the similarities and differences between these methods and understand why some methods perform better than others.

In Section 2, we will describe the theory related to all methods and make a theoretical comparison. In Section 3, we will outline how the simulation study was conducted and present the real data used in this project. In Section 4, we will present the results, and in Section 5, we will discuss these results.

2 Theory

2.1 Logistic regression

The theory that describes logistic regression has been taken from [7] and [9].

2.1.1 Description of the model

Logistic regression is a widely used statistical method for classification that models the probability of the response variable based on one or more predictor variables. In binary classification, the response variable Y follows a Bernoulli distribution, taking on binary values 0 and 1. The predictor variables $\mathbf{x} = (x_1, x_2, ..., x_p)^T$ can be continuous or categorical. The conditional probability $P(Y = 1|\mathbf{x})$ is modeled by applying the *sigmoid function*, also known as the *logistic function*, to the linear combination of the explanatory variables. Mathematically it can be written as

$$\begin{aligned} P(Y = 1|\mathbf{x}) &= \frac{e^{\beta_0+\beta_1 x_1+...+\beta_p x_p}}{1 + e^{\beta_0+\beta_1 x_1+...+\beta_p x_p}} = \frac{e^{\beta_0+\boldsymbol{\beta}^T\mathbf{x}}}{1 + e^{\beta_0+\boldsymbol{\beta}^T\mathbf{x}}} \\ &= \frac{1}{1 + e^{-(\beta_0+\boldsymbol{\beta}^T\mathbf{x})}}, \end{aligned} \tag{1}$$

where β_0 is the intercept and $\boldsymbol{\beta} = (\beta_1, ..., \beta_p)^T$ is the vector of coefficients.

The sigmoid function is a so-called link function, and it is defined as

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

It is a continuous function that maps real-valued input values to an output in the interval $(0, 1)$. The curve of the sigmoid function is known for its characteristic S-shape, which increases smoothly from 0 to 1 as the input value ranges from negative infinity to positive infinity. This allows us to interpret the model's output as the probability that $Y = 1$, given the input variables. The shape of the sigmoid function is shown in Figure 1.

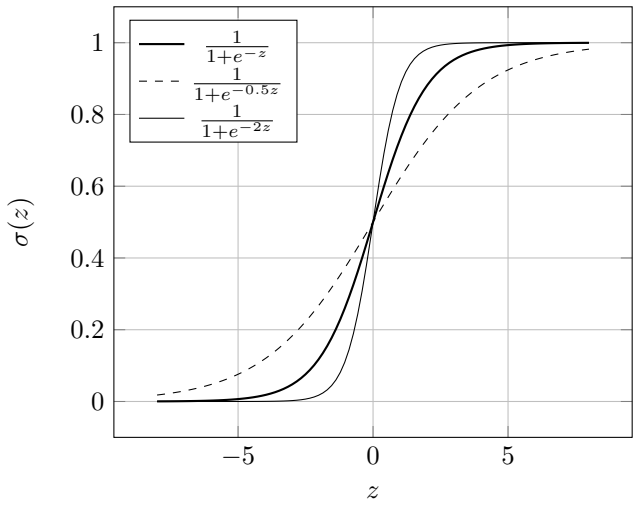


Figure 1: *Plot of the sigmoid function* $\sigma(z) = \frac{1}{1+e^{-vz}}$ *when* $v = 1$ *(black curve), when* $v = 0.5$ *(dashed curve) and when* $v = 2$ *(grey curve). The figure is from [7]*

The logistic regression model can also be expressed in terms of the log odds of the probability $P(Y = 1|\mathbf{x})$

$$\begin{aligned} \log\left(\frac{P(Y=1|\mathbf{x})}{1-P(Y=1|\mathbf{x})}\right) &= \beta_0 + \beta_1 x_1 + ... + \beta_p x_p \\ &= \beta_0 + \boldsymbol{\beta}^T \mathbf{x}. \end{aligned} \tag{3}$$

Unlike probabilities, which are constrained to the interval $[0, 1]$, the log odds is unbounded and can be estimated by the linear function $\beta_0 + \boldsymbol{\beta}^T\mathbf{x}$. This implies that binary logistic regression is a linear model in the log odds of the probability $P(Y = 1|\mathbf{x})$. Consequently, each parameter β_j $(j = 1, ...p)$ can be interpreted as the change in log odds of the conditional probability for a one-unit change in x_j, assuming x_j does not interact with other variables and that all other variables are held constant.

To use logistic regression as a classification method, we must choose a threshold c, and assign the class based on whether the predicted probability is above or below this threshold. In other words, to classify a new observation $\mathbf{x}^\star$, we begin by calculating the predicted conditional probability $P(Y^\star = 1|\mathbf{x}^\star)$ by plugging $\mathbf{x}^\star$ in the estimated logistic function. If the predicted probability is greater than or equal to c, the point is classified as belonging to class 1, and if it is less than c, the point is classified as belonging to class 0. The threshold is often set to be 0.5 and geometrically it corresponds to a hyperplane $\beta_0 + \boldsymbol{\beta}^T\mathbf{x} = 0$, which separates the feature space into two regions. Points lying on one side of the hyperplane have a predicted probability of belonging to one class, while points lying on the other side have a predicted probability of belonging to the other class.

2.1.2 Estimation of the parameters

The coefficients β_0 and $\boldsymbol{\beta}$ are estimated using the *Maximum Likelihood Estimation (MLE) method*, which aims to find the parameter values that maximize the likelihood of observing the given data.

Suppose we have N independent datapoints $\{Y_i, \mathbf{x}_i\}_{i=1}^N$. The joint likelihood can be expressed as a product of individual likelihoods

$$\mathrm{L}(\beta_0, \boldsymbol{\beta}) = \prod_{i=1}^{N} p_i^{y_i}(1-p_i)^{1-y_i}, \tag{4}$$

where y_i denotes the outcome of the *i-th* response variable and p_i denotes the probability that $\mathrm{Y}_i = 1$ given $\mathbf{x}_i$.

It is often easier to maximize the log-likelihood, which is obtained by taking the log of the likelihood function

$$\ell(\beta_0, \boldsymbol{\beta}) = \sum_{i=1}^{N} \Big\{ y_i \log(p_i) + (1-y_i)\log(1-p_i) \Big\}. \tag{5}$$

By substitituting $\frac{1}{1+e^{-(\beta_0+\boldsymbol{\beta}^T\mathbf{x}_i)}}$ for p_i and simplifying, the expression becomes

$$\ell(\beta_0, \boldsymbol{\beta}) = \sum_{i=1}^{N} \Big\{ y_i(\beta_0 + \boldsymbol{\beta}^T\mathbf{x}_i) - \log\left(1 + e^{\beta_0+\boldsymbol{\beta}^T\mathbf{x}_i}\right) \Big\}. \tag{6}$$

Optimizing this function requires the use of numerical methods and one common choice is the Newton-Raphson algorithm. To apply this method, we need to compute the gradient vector and the Hessian matrix of the log-likelihood function $\ell(\beta_0, \boldsymbol{\beta})$ with respect to the parameters β_0 and $\boldsymbol{\beta} = (\beta_1, ..., \beta_p)$. For convenience, we will include the intercept into the vector of coefficients $\boldsymbol{\beta}$.

The gradient vector and the Hessian matrix are obtained by taking the first and second partial derivatives of the log-likelihood function. They are given by the formulas

$$\frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = \sum_{i=1}^{N} (y_i - p_i)\mathbf{x}_i, \tag{7}$$

$$\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}^2} = -\sum_{i=1}^{N} p_i(1-p_i)\mathbf{x}_i\mathbf{x}_i^T. \tag{8}$$

The Newton-Raphson algorithm updates the parameters $\boldsymbol{\beta}$ iteratively by using the following formula

$$\boldsymbol{\beta}^{t+1} = \boldsymbol{\beta}^t - \left(\frac{\partial^2 \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}^2}\right)^{-1} \frac{\partial \ell(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}, \tag{9}$$

where the derivatives are evaluated at $\boldsymbol{\beta}^t$.

2.1.3 Common issues during the training process

Training a logistic regression model can present various challenges, including overfitting, collinearity, and the handling of linearly non-separable classes.

Overfitting occurs when a model learns the patterns of the training data too well and captures noise. This leads to the model failing to generalize to new, unseen data.

Collinearity is a term used to describe pairwise linear dependence among some or all predictor variables, which can result in unstable estimates of the coefficients. When variables are highly correlated, it might be difficult for the model to identify the individual impact of each predictor variable on the response variable.

Lastly, as was described in section 2.1.1, logistic regression is inherently a linear model and therefore can only produce linear decision boundaries.

Common ways to adress the first two problems is to use regularization, i.e adding a penalty term to the loss function. Two popular types of regularization are Lasso (L1 regularization) and Ridge (L2 regularization)

L1 regularization

$$\ell(\beta_0, \boldsymbol{\beta}) = \sum_{i=1}^{N} \left\{ y_i(\beta_0 + \boldsymbol{\beta}^T \mathbf{x}_i) - \log\left(1 + e^{\beta_0 + \boldsymbol{\beta}^T \mathbf{x}_i}\right) \right\} - \lambda \sum_{j=1}^{p} |\beta_j|, \tag{10}$$

L2 regularization

$$\ell(\beta_0, \boldsymbol{\beta}) = \sum_{i=1}^{N} \left\{ y_i(\beta_0 + \boldsymbol{\beta}^T \mathbf{x}_i) - \log\left(1 + e^{\beta_0 + \boldsymbol{\beta}^T \mathbf{x}_i}\right) \right\} - \lambda \sum_{j=1}^{p} \beta_j^2. \tag{11}$$

The intercept is usually not regularized because penalizing it often leads to underfitting. The regularization parameter λ is used to control the strength of the penalty term, with larger values of λ leading to stronger regularization. In L2 regularization, increasing λ asymptotically shrinks the weights towards zero, while in L1 regularization, increasing λ can shrink the weights all the way to zero. Thus the L1 method also works as a selection method, which means that it can remove non-significant variables. It is also possible to combine these two regularization types, and the resulting method is called elastic net.

To adress linearly non-separable classes, we can use basis expansion. Basis expansion is a term that describes the process of transforming the vector of independent variables $\mathbf{x}$ into a higher-dimensional space where classes can be separated by a linear boundary. This is possible because linear models only need to be linear in their coefficients. However, doing so often leads to overfitting, and therefore, basis expansion is often used with one of the regularization methods described above.

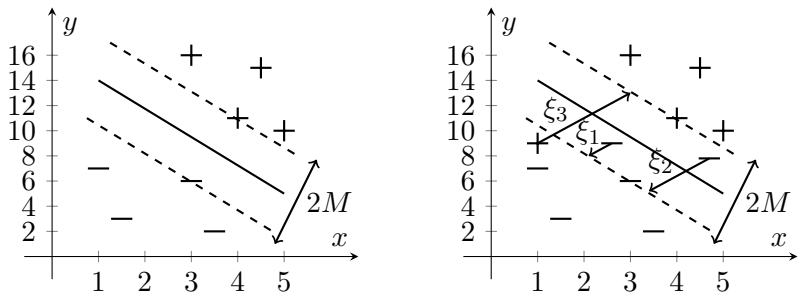
2.2 Support vector machines

The theory that describes support vector machines has been taken from [7] and [13].

2.2.1 Support vector classifiers

The support vector classifier is another popular classification method. The main idea of this method is to find a hyperplane that optimally separates the data into two classes. This is achieved by maximizing the margin, denoted as M, which is defined as the distance between the hyperplane and the nearest data point(s) from each class (referred to as support vectors). The target variable Y takes on values $\{-1, 1\}$ and $\mathbf{x} = (x_1, x_2, ..., x_p)^T$ is a set of independent variables. An illustration of how the support vector classifier works is shown in Figure 2.

Mathematically, a hyperplane is described by the equation $\boldsymbol{\beta}^T\mathbf{x} + \beta_0 = 0$. The vector $\boldsymbol{\beta}$ is called the normal vector because it is orthogonal to the surface of the hyperplane and β_0 is the bias term. For any point $\mathbf{x}_0$ lying on the hyperplane, the dot product $\boldsymbol{\beta}^T\mathbf{x}_0$ is equal to $-\beta_0$. This implies that the distance from the origin to the hyperplane is given by $\frac{\beta_0}{||\boldsymbol{\beta}||}$, where $||\boldsymbol{\beta}||$ denotes the magnitude of the vector $\boldsymbol{\beta}$. The signed distance from any point $\mathbf{x}$ to the hyperplane is given by the formula $\frac{1}{||\boldsymbol{\beta}||}(\boldsymbol{\beta}^T\mathbf{x} + \beta_0)$.



(a) *The support vector classifier in the case of perfect separability.*

(b) *The support vector classifier in the case of overlapping classes.*

Figure 2: *The support vector classifier when classes are perfectly separable and when classes overlap.*

Hard margin

Suppose there are N datapoints $\{y_i, \mathbf{x}_i\}_{i=1}^N$. In the case of perfect separability between classes, the problem of maximizing the margin M can be formulated as follows

$$\begin{aligned} &\max_{\beta_0, \boldsymbol{\beta}} M \\ &\text{Subject to } \frac{1}{||\boldsymbol{\beta}||} y_i(\boldsymbol{\beta}^T\mathbf{x}_i + \beta_0) \geq M, \qquad i = 1, ..., N. \end{aligned} \tag{12}$$

The constraints guarantee that each data point is at least a signed distance of M

from the hyperplane, resulting in an empty margin around the boundary with a width of $2M$.

This problem can be simplified by multiplying both sides in the constraints by $||\boldsymbol{\beta}||$ and choosing $||\boldsymbol{\beta}|| = \frac{1}{M}$. The result is that the right side of the constraints becomes equal to 1 and it allows us to rewrite the problem as

$$\begin{aligned} &\min_{\beta_0, \boldsymbol{\beta}} \frac{1}{2}||\boldsymbol{\beta}||^2 \\ &\text{Subject to } y_i(\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0) \geq 1, \qquad i = 1, ..., N. \end{aligned} \tag{13}$$

This is a convex minimization problem (quadratic criterion with linear inequality constraints) that can be solved using the *method of Lagrange multipliers*. The function to be minimized with respect to the parameters $\boldsymbol{\beta}$ and β_0 is

$$L_P = \frac{1}{2}||\boldsymbol{\beta}||^2 - \sum_{i=1}^{N} \alpha_i [y_i(\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0) - 1]. \tag{14}$$

This function is called the primal Lagrange function. However, in the case of support vector classifiers, it is easier to optimize the dual Lagrange function, and it allows us to use the kernel trick (which will be described in the next section).

To obtain the dual Lagrange function, we start by computing and setting the partial derivatives of the primal function to 0. The resulting equations are

$$\boldsymbol{\beta} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i, \tag{15}$$

$$0 = \sum_{i=1}^{N} \alpha_i y_i. \tag{16}$$

By substituting these equations back into the primal function, we obtain the dual Lagrange function

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j. \tag{17}$$

This function is maximized with respect to the parameters α_i $(i = 1, ..., N)$ under the constraints $(15), (16)$ and

$$\begin{aligned} &\alpha_i \geq 0, \qquad i = 1, ..., N, \\ &\alpha_i [y_i(\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0) - 1] = 0, \qquad i = 1, ..., N. \end{aligned} \tag{18}$$

From these constraints, we can see that if $\alpha_i > 0$, the corresponding point $\mathbf{x}_i$ lies on the margin and is a support vector. Furthermore, $\alpha_i = 0$ for all other points.

The vector of parameters $\boldsymbol{\beta}$ is estimated by plugging in the estimated α values into the expression

$$\boldsymbol{\beta} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i. \tag{19}$$

Because the α values for points that do not lie on the margin are 0, the vector $\boldsymbol{\beta}$ is estimated through a linear combination of the support vectors.

The bias term is estimated by plugging in one of the support vectors $\{y_s, \mathbf{x}_s\}$ into

$$\alpha[y(\boldsymbol{\beta}^T \mathbf{x} + \beta_0) - 1] = 0 \tag{20}$$

and solving for β_0.

To classify a new observation $\mathbf{x}^\star$, we plug it into the estimated function

$$f(\mathbf{x}^\star) = \hat{\boldsymbol{\beta}}^T \mathbf{x}^\star + \hat{\beta}_0. \tag{21}$$

If the outcome is positive, then the point is classified as belonging to class 1, and if it is negative, the point is classified as belonging to class -1.

Soft margin

If there are overlapping points, the problem of maximizing the margin M becomes more complicated and we have to allow some points to be on the wrong side of the margin. This is achieved by introducing slack variables $\boldsymbol{\xi} = (\xi_1, ..., \xi_N)$ and modifying the constraints as follows

$$\frac{1}{||\boldsymbol{\beta}||} y_i(\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0) \geq M(1 - \xi_i), \qquad i = 1, ..., N. \tag{22}$$

The value ξ_i represents the proportional deviation from the margin, and all $\xi_i \geq 0$. A data point $\mathbf{x}_i$ is correctly classified if $\xi_i = 0$. If $0 < \xi_i < 1$, then the point is on the correct side of the boundary but lies within the margin. A misclassfication occurs when $\xi_i > 1$.

As in the case of perfectly separable classes, we can multiply both sides of the constraints by $||\boldsymbol{\beta}||$ and choose $||\boldsymbol{\beta}|| = \frac{1}{M}$. This results in the right side of the constraints becoming equal to $1 - \xi_i$ and the optimization problem becomes

$$\begin{aligned} &\min_{\beta_0, \boldsymbol{\beta}} \frac{1}{2}||\boldsymbol{\beta}||^2 + C \sum_{i=1}^{N} \xi_i \\ &\text{Subject to } \xi_i \geq 0,\ y_i(\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0) \geq 1 - \xi_i, \qquad i = 1, ..., N. \end{aligned} \tag{23}$$

The term $C \sum_{i=1}^{N} \xi_i$ bounds the total number of misclassifications. The parameter C controls the trade-off between achieving a wide margin and minimizing

the number of misclassifications. Smaller C creates a wider margin and allows for more misclassifications, while larger C creates a smaller margin and thus fewer misclassifications are allowed. The optimal value for C is determined through cross-validation.

The Lagrange primal function to be minimized with respect to the parameters $\boldsymbol{\beta}, \beta_0$ and ξ_i $(i = 1, ..., N)$ is

$$L_P = \frac{1}{2}||\boldsymbol{\beta}||^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(\boldsymbol{\beta}^T\mathbf{x}_i + \beta_0) - (1 - \xi_i)] - \sum_{i=1}^{N}\mu_i\xi_i. \tag{24}$$

It is once again easier to optimize the dual Lagrange function and it is obtained in the same way. The partial derivates of the primal function equated to 0 are

$$\boldsymbol{\beta} = \sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i, \tag{25}$$

$$0 = \sum_{i=1}^{N}\alpha_i y_i, \tag{26}$$

$$\alpha_i = C - \mu_i, \qquad i = 1, ...N. \tag{27}$$

And the dual Lagrange function is

$$L_D = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j. \tag{28}$$

This function is maximized with respect to the parameters α_i $(i = 1, ..., N)$ under the constraints $(25) - (27)$ and

$$\begin{aligned} &0 \leq \alpha_i \leq C, \\ &\alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] = 0, \\ &\mu_i\xi_i = 0, \\ &y_i(x_i^T\beta + \beta_0) - (1 - \xi_i) \geq 0, \end{aligned} \tag{29}$$

for $i = 1, ..., N$.

2.2.2 Support vector machines and kernels

The support vector classifier described above is only suitable for data that is linearly separable. To handle cases where classes are not linearly separable, we can use basis expansion. Suppose that $h(\cdot)$ represents the transformation into a higher dimensional space, then the dual Lagrange function can be written as

$$L_D = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \langle h(\mathbf{x}_i), h(\mathbf{x}_j)\rangle, \tag{30}$$

where $\langle h(\mathbf{x}_i), h(\mathbf{x}_j)\rangle = h(\mathbf{x}_i)^T h(\mathbf{x}_j)$.

And the estimated classification function becomes

$$f(\mathbf{x}) = \hat{\boldsymbol{\beta}}^T h(\mathbf{x}) + \hat{\beta}_0 \tag{31}$$

$$= \sum_i^N \hat{\alpha}_i y_i \langle h(\mathbf{x}_i), h(\mathbf{x})\rangle + \hat{\beta}_0. \tag{32}$$

We see that both functions only involve inner products of the transformed features. This allows us to use the kernel trick, which is a way to perform computations in the higher-dimensional space without explicitly computing the transformation $h(\cdot)$. This is done by using a kernel function

$$K(\mathbf{x}, \mathbf{x}') = \langle h(\mathbf{x}), h(\mathbf{x}')\rangle \tag{33}$$

which computes inner products in the transformed feature space.

Some commonly used kernels are

$$\begin{aligned} \text{Linear kernel: } & K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}'\rangle, \\ \text{d-th Degree polynomial kernel: } & K(\mathbf{x}, \mathbf{x}') = (1 + \langle \mathbf{x}, \mathbf{x}'\rangle)^d, \\ \text{Radial basis kernel: } & K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma ||\mathbf{x} - \mathbf{x}'||^2). \end{aligned} \tag{34}$$

In this thesis we will use the linear kernel and the radial basis kernel. The linear kernel computes the inner products in the original feature space, while radial basis function kernel computes the inner products in the infinitely dimensional space.

The gamma parameter in the radial basis function (RBF) kernel decides how strongly each training point influences the model. When gamma is low, each point has a widespread effect, spreading the decision boundary. With high gamma, each point has a localized impact, focusing the decision. However, if gamma is too high, it might lead to overfitting.

2.3 Artificial neural networks

The theory that describes neural networks has been taken from [6], [7] and [1].

2.3.1 Description of the model

Artificial neural networks are a group of machine learning models inspired by the structure and function of biological neural networks. They consist of interconnected nodes or neurons, with weights assigned to the connections between them. These neurons are organized into layers, with the first layer as the input layer and the last layer as the output layer. The layers in-between are known as hidden layers since their outputs are not directly observable. In this project we focused on fully-connected feed-forward neural networks. In other words networks in which all neurons in one layer are connected to all neurons in the next layer, with information flowing only in one direction (from the input layer through one or more hidden layers and to the output layer). An example of this type of network can be seen in Figure 2.

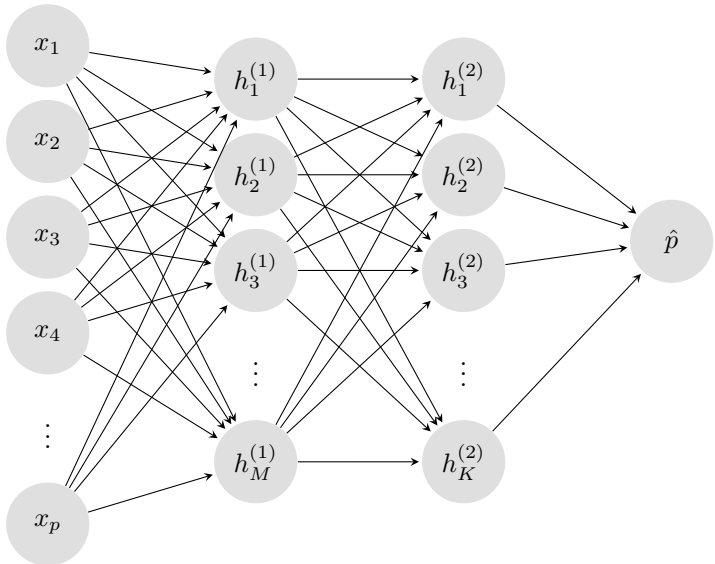


Figure 3: *A fully-connected feedforward neural network with 2 hidden layers*

The architecture of a two-hidden-layer fully-connected feed-forward neural network in more detail can be described as follows:

The input layer is defined by a set of input variables $\mathbf{x} = (x_1, x_2, ..., x_p)^T$ with the number of neurons matching the number of input features. The purpose of the input layer is to pass the input data to the first hidden layer of the network.

In the first hidden layer, each neuron processes input data by computing a linear combination of the variables and applying a non-linear activation function. Mathematically, this is expressed as

$$h_m^{(1)} = f^{(1)}\left(\sum_{j=1}^{p} w_{mj}^{(1)} x_j + b_m^{(1)}\right), \qquad m = 1, ..., M, \tag{35}$$

where $w_{mj}^{(1)}$ represents the weight connecting the j-th input node to the m-th node in the first hidden layer, $b_m^{(1)}$ represents the bias term in the m-th node, $f^{(1)}()$ represents the activation function, $h_m^{(1)}$ is the output of the m-th neuron, and M is the number of units in the first hidden layer.

This process is then repeated in the second hidden layer. Each neuron receives the output data from the first hidden layer, computes a weighted sum of the variables, and applies a non-linear activation function. This can be described using the following equation

$$h_k^{(2)} = f^{(2)}\left(\sum_{m=1}^{M} w_{km}^{(2)} h_m^{(1)} + b_k^{(2)}\right), \qquad k = 1, ..., K. \tag{36}$$

Here $w_{km}^{(2)}$ represents the weight connecting the m-th node in the first hidden layer to the k-th node in the second hidden layer, $b_k^{(2)}$ represents the bias term in the k-th node, $f^{(2)}()$ represents the activation function, $h_k^{(2)}$ is the output of the k-th node, and K is the number of neurons in the second hidden layer.

In this project we used the same activation function in both hidden layers. Activation functions play an important role in neural networks because they introduce non-linearity to the model, allowing it to capture complex relationships between the input and output data. Without a non-linear activation function, the neural network would collapse into a linear model. One of the most commonly used activation functions is the Rectified Linear Unit (ReLU) function, which was used in this project. It is defined as

$$g(z) = \max\{0, z\}. \tag{37}$$

This function returns z for all positive values of z and 0 for all negative values of z. The reason it is commonly used as an activation function is because it is very similar to a linear function which makes it computationally efficient and easily differentiable, but at the same time it introduces non-linearity to the model. The derivative is 0 for all negative values and 1 for all positive values. The derivative is undefined at $z = 0$, but in practice, one often selects one of the one-sided derivatives.

Finally, the output layer takes as input the output data from the second hidden layer and generates the predicted value. In binary classification, the output layer consists of a single neuron that produces the probability that Y = 1. It is defined as

$$\hat{p} = f^{(3)}\left(\sum_{k=1}^{K} w_k^{(3)} h_k^{(2)} + b^{(3)}\right). \tag{38}$$

$w_k^{(3)}$ represents the weight connecting the k-th node in the second hidden layer to the output node, $b^{(3)}$ is the bias term, $f^{(3)}$ is the activation function, and $\hat{p}$ is the predicted outcome.

The activation function $f^{(3)}()$ is the sigmoid function. The reason for this is the same as for logistic regression: it transforms an unbounded function into a probability.

More generally, when predicting a categorical variable with G categories, the neural network's output layer consists of G neurons. The activation function that is used is the softmax function

$$\text{Softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{G} e^{z_j}}. \tag{39}$$

2.3.2 Backpropagation and gradient descent

Training a neural network means minimizing the loss function $R(\boldsymbol{\theta})$ with respect to the parameters $\boldsymbol{\theta}$. In the case of binary classification, the most commonly used loss function is the binary cross-entropy loss function, which measures the difference between the predicted probability distribution and the true probability distribution of the data. For a dataset with N datapoints $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$, the binary cross-entropy loss is defined as

$$R(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} -\Big\{ y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i) \Big\}. \tag{40}$$

Here, y_i represents the outcome of the variable Y_i, and $\hat{p}_i$ is the predicted probability that $Y_i = 1$. The vector of parameters $\boldsymbol{\theta}$ represents all the weights and biases present in the model. In a two-hidden-layer network described above there are in total Mp weights and M biases in the first layer, KM weights and K biases in the second layer, and K weights and one bias in the output layer.

The most commonly used method to minimize the loss function is the *gradient descent method*, which is a first-order optimization method that aims to find the minimum of a function by iteratively adjusting the parameters in the direction of steepest decrease of the gradient.

The update rule for the gradient descent method is

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - \alpha \frac{\partial R(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}, \tag{41}$$

where $\frac{\partial R(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ is the gradient of the loss function evaluated at $\boldsymbol{\theta}^t$. The learning rate α is a small positive value that determines the stepsize in each iteration.

The loss function can be written as a sum of individual losses

$$R(\boldsymbol{\theta}) = \sum_{i=1}^{N} R_i(\boldsymbol{\theta}). \tag{42}$$

This fact can be used to derive different variants of the gradient descent method. If we use the entire dataset to compute the gradient, the method is called *batch gradient descent*. In this version of the method, the parameters are updated after each epoch, i.e after one complete cycle through all datapoints in the training process. Batch gradient descent works well when the dataset is small and there are few parameters. However, for large datasets with many parameters, the algorithm becomes very slow and, in some cases, it becomes computationally too expensive to compute the gradient.

Other variants of gradient descent that work better for large datasets are *stochastic gradient descent* and *mini-batch gradient descent*. In stochastic gradient descent, the gradient is calculated on a single randomly selected datapoint from the dataset, and the parameters are updated based on the gradient from that observation. In mini-batch gradient descent, the gradient is computed on a small batch of randomly selected datapoints from the dataset.

To compute the gradient of the loss function we use the *backpropagation algorithm*. Because of the compositional form of the loss function this is done using the chain rule of calculus, which allows us to break down the computation of the gradient into simpler parts. The algorithm begins in the output layer and computes the partial derivatives of the loss function with respect to the parameters in that layer. These partial derivatives are then used to compute the derivatives of the loss function with respect to the parameters in the previous layer, and this process is repeated until the input layer is reached.

2.3.3 Common issues during the training process

Common challenges during the training process of neural networks include overfitting and the handling of the non-convexity of the loss function.

Neural networks are very susceptible to overfitting because they have a lot of parameters, so it is important to apply some sort of regularization during training. One common technique is to add a penalty term to the loss function

$$R(\boldsymbol{\theta}, \lambda) = R(\boldsymbol{\theta}) + \lambda J(\boldsymbol{\theta}), \tag{43}$$

where $J(\boldsymbol{\theta})$ represents the regularization term.

Common penalty terms include L1 norm and L2 norm. They have been described for logistic regression in section 2.1.3.

Another way to regularize a model is through *early stopping*, which involves stopping the training process before the model reaches the global minimum. This is usually done by dividing the training data into a training set used for updating the model parameters and a validation set used for monitoring performance and deciding when to stop training.

However, due to the non-convex nature of the loss function, there is no guarantee that the optimization process will find the global minimum. There is always a possibility it may get stuck on a local minimum or a saddle point. Therefore, it is important to choose appropriate initial values and learning rate. If the learning rate is too large, the algorithm might oscillate or even diverge. If it is too small, the algorithm may converge too slowly. The initial values for the weights and biases in a neural network are often set to small random values.

2.4 Theoretical comparison of the models

Both support vector classifiers and logistic regression are linear models and thus are limited to linear decision boundaries.

The optimization problem for the soft margin classifier in (23) can be rewritten as

$$\min_{\beta_0,\boldsymbol{\beta}} \sum_{i=1}^{N} \left[1 - y_i(\boldsymbol{\beta}^T \mathbf{x}_i + \beta_0)\right]^+ + \frac{\lambda}{2}||\boldsymbol{\beta}||^2, \tag{44}$$

where $\sum_{i=1}^{N}[z_i]^+ = \sum_{i=1}^{N} \max\{0, z_i\}$ is called the hinge loss.

So it can be written as the hinge loss + L2 penalty.

The hinge loss and the negative log-likelihood function (which is the negative of the log-likelihood function) have been shown to have similar shapes [14]. This indicates that logistic regression and support vector classifiers are expected to have similar performance.

The negative log-likelihood function is also equivalent to the binary cross entropy used in neural networks. Thus, artificial neural networks and logistic regression use the same function to optimize the parameters. Both of these methods also use the sigmoid function to convert a linear function into a probability. The main difference lies in the fact that logistic regression applies the sigmoid function to a linear combination of the input variables, while a neural network model applies the sigmoid function to the linear combination of the outputs of the last hidden layer. (Note that it can also be used as an activation function in hidden layers). Because the activation functions in the hidden layers are not linear, the neural network is thus a lot more flexible and can capture more complex relationships in the data.

But logistic regression can use basis expansion and support vector classifiers can be extended to support vector machines to become more flexible.

2.5 Model evaluation

The theory has been taken from [5] and [4].

2.5.1 Confusion matrix

In binary classification, a confusion matrix is a 2×2 matrix that summarizes the performance of a classification method. The positive class is represented by Y = 1, and the negative class is represented by Y = 0 (-1 for SVMs). The matrix contains four values: true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). These describe how well the model's predicted classes align with the true classes of the observations. The sums of column values correspond to the actual number of observations in each class, and the sums of row values correspond to the predicted number of observations in each class.

		Real classes	
		1	0
Predicted classes	1	True Positives (TP)	False positives (FP)
	0	False Negatives (FN)	True Negatives (TN)

Table 1: *A confusion matrix*

From the confusion matrix, we can compute several metrics to evaluate the performance of the model:

- *Accuracy* is the proportion of all observations that are correctly classified by the model

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}. \tag{45}$$

- *True positive rate*, which is also called *recall* or *sensitivity* is the proportion of observations that belong to class 1 that are correctly classified by the model

$$TP\ rate = \frac{TP}{TP + FN}. \tag{46}$$

- *False positive rate* is the proportion of observations that belong to class 0 that are uncorrectly classified as belonging to class 1 by the model

$$FP\ rate = \frac{FP}{FP + TN}. \tag{47}$$

- *Precision* is the proportion of observations that are predicted to belong to class 1 that actually belong to class 1.

$$Precision = \frac{TP}{TP + FP} \tag{48}$$

- *Specificity* is the proportion of observations that belong to class 0 that are correctly classified by the model.

$$Specificity = \frac{TN}{FP + TN}. \tag{49}$$

- F_1 Score is the harmonic mean of recall and precision.

$$F_1 \text{ Score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{50}$$

2.5.2 ROC and AUC

A Receiver Operating Characteristic (ROC) curve is a graphical representation of a binary classification model's performance at different classification thresholds. The true positive rate (sensitivity) is plotted on the y-axis, while the false positive rate (1 - specificity) is plotted on the x-axis. The diagonal line, where the true positive rate equals the false positive rate, represents the performance expected from random guessing. AUC is the area under the ROC graph. AUC ranges from 0 to 1, and the higher the AUC the better the performance of a classification method.

Figure 4 shows an example of a ROC-graph.

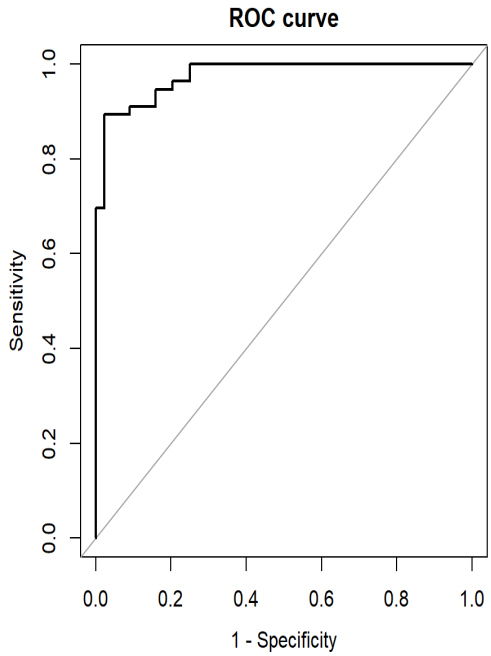


Figure 4: *A ROC curve created in R.*

2.5.3 Cross validation

Cross validation is a method that is used to evaluate the model performance. In the simplest form, data gets divided into two groups, a training and a test set. The training set is used to estimate the parameters, i.e to train the model and the test set is used to evaluate the model performance. In more advanced kind of cross validation, which is called k-fold cross validation, data gets divided into k groups and one of the subsets is chosen for testing and the rest of them are used for training. This is repeated k times, so that each subset is used for testing exactly once. The performance measures are computed for each test set and the final result is obtained by averaging those.

3 Data and model fitting

3.1 Data

To compare the three methods described above, we tested their predictive power using both simulated and real-world data.

3.1.1 Setup of the simulation study

In this simulation study, we generated datasets with different characteristics and compared the predictive power of the methods. Each dataset was split into a training set (80%) and a testing set (20%). To get more accurate results, each simulation experiment was repeated 30 times.

In Experiment 1 and 2 and in the first part of Experiment 3, the response variable was simulated using the logistic function. In other words, we first computed some function of the independent variables and then applied the sigmoid function

$$\mathrm{P}(Y = 1|\mathbf{x}) = \frac{1}{1 + e^{-(f(\mathbf{x}))}}. \tag{51}$$

After that, we used the `rbinom` function in R to get the outcome of a Bernoulli random variable with that probability.

Experiment 1

In the first experiment, the aim was to investigate the predictive performance of the methods while varying the number of observations (N) and predictors (p). We simulated datasets with 10, 30, 50, and 100 continuous predictors and with 500 and 2000 observations. The predictor variables were simulated from a standard multivariate normal distribution.

The response variable was generated using a linear function

$$f(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}^T \mathbf{x}. \tag{52}$$

Each element of the vector of coefficients $\boldsymbol{\beta}$ was drawn independently from a uniform random variable on the interval (1.5, 2.5), and the intercept β_0 was manually chosen to achieve a balanced distribution of classes. These coefficients were simulated once and used in all 30 simulations.

Experiment 2

In the second experiment, the aim was to investigate the predictive performance of the methods while varying the number of observations and the number of categorical predictors. We simulated datasets with 7, 10, and 15 predictors (5 of which were continuous) and with 500 and 2000 observations. The continuous variables were simulated from a standard multivariate normal distribution. The categorical variables were generated by first simulating standard uniform random variables and then dividing them into 4 bins, resulting in categorical predictors with 4 categories. After that, these variables got transformed into dummy variables.

The response variable was generated using this function

$$f(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}^T \mathbf{x} + \sum_{j=1}^{m} \sum_{k=1}^{3} \beta_{jk} d_{jk}. \tag{53}$$

Here, m is the number of categorical predictors, and d_{jk} represents the k-th dummy variable in the j-th categorical predictor.

All coefficients were once again drawn independently from a uniform random variable on the interval $(1.5, 2.5)$, and the intercept β_0 was chosen subsequently to get a balanced distribution of classes. These coefficients were used in all 30 simulations.

Experiment 3

In the third experiment, the goal was to investigate how the classifiers would perform on data that cannot be separated by a single linear decision boundary. We generated two datasets with 500 observations and 2 predictor variables. We decided to only have two predictors because it allowed us to visualise the decision boundary.

Part 1

In the first dataset, the predictor variables were simulated from a standard bivariate normal distribution.

The response variable was modeled using this function

$$f(\mathbf{x}) = \beta_0 + \beta_1 x_1^2 + \beta_2 x_2. \tag{54}$$

The coefficients were chosen to be $\beta_1 = 4, \beta_2 = -4$ and $\beta_0 = -3.17$.

Part 2

For the second dataset we simulated data with four distinct clusters. Each cluster was generated from a bivariate normal random variable with a unique mean vector and the identity covariance matrix. The first two clusters, representing the positive class ($y = 1$), were chosen to have mean vectors $(1.5, 1.5)$ and $(-1.5, -1.5)$, while the remaining two clusters, representing the negative class ($y = 0$), were chosen to have mean vectors $(1.5, -1.5)$ and $(-1.5, 1.5)$. Both variables got standardized and the R function `sample` was used to shuffle the data.

3.1.2 Real-world data

For the real data, we used two datasets obtained from the UCI machine learning repository. To evaluate the performances of the classification methods, we decided to use 5-fold cross-validation. We used the R package `caret` to create folds. Before fitting the models, the continuous variables got normalized and the binary variables got transformed into dummy variables.

Breast Cancer Wisconsin (Diagnostic) dataset

The first dataset that we used contains 569 observations and 31 variables. 30 of these variables are continuous and computed from images of a breast mass, describing various characteristics of the cells present in the images. The last variable is a binary response variable indicating whether these cells are malignant (cancerous) or benign (non-cancerous). Out of the 569 observations, 357 are classified as benign and 212 as malignant. Some of the variables are higly correlated.

Early stage diabetes risk prediction dataset

The second dataset that we used consists of 520 observations and 17 variables. The response variable indicates whether a patient is diabetic or not. Among the 16 predictors, 15 are binary and include factors such as the patient's gender and (yes-no) responses to questions regarding symptoms or experiences that are associated with diabetes such as if the patient has experienced sudden weight loss or visual blurring. There is one continuous variable and it represents patient's age. Out of the 520 observations, 320 are classified as diabetic (positive), and 200 are classified as non-diabetic (negative).

3.2 Model fitting

Logistic regression

The logistic regression models were fitted using the `glmnet` package in R. We used the $L2$ regularized version of logistic regression, and the value for λ was chosen through 5-fold cross-validation. (We decided to use L2 regularized version to avoid overfitting and because the breast cancer dataset has correlated predictors)

Support vector machines

The SVM models were fitted using the `e1071` package in R. The hyperparameters, namely C and γ, were selected through 5-fold cross-validation. In experiments 1 and 2, we used the linear kernel and in experiment 3, the radial basis kernel was used. For real-world datasets, we fitted models with both kernels.

Artificial neural networks

The neural network models were fitted using the `keras` package in R. We used the mini-batch gradient descent method with a batch size or 50 to train the models, and the learning rate was set to 0.03. To prevent overfitting, we employed early stopping. To determine the optimal architecture of the neural network, we used cross-validation. We divided the training set into three sets: a training set, a validation set and a test set. A total of 240 models were generated, with the number of hidden units in the first layer ranging from 1 to 15 and in the second layer from 0 to 15. For each model, accuracy was computed, and the model with the highest accuracy was selected. All models were trained for a maximum of 300 epochs. When the optimal number of neurons was obtained, we divided the entire training data into a training set (90%) and a validation set (10%) and the training process stopped if the validation loss started to increase.

4 Results

Once the models were trained, we evaluated their performance on the test data using the accuracy and AUC as performance metrics. For real data, as mentioned earlier, we employed 5-fold cross-validation to test the model performance.

4.1 Simulated data

Experiment 1

Model	N	p	Accuracy	AUC
LR	500	10	0.9197 (0.023)	0.9778 (0.009)
SVC			0.9167 (0.024)	0.9770 (0.011)
ANN			0.9096 (0.024)	0.9711 (0.012)
LR	500	30	0.927 (0.016)	0.9834 (0.009)
SVC			0.93 (0.023)	0.9844 (0.008)
ANN			0.9193 (0.024)	0.9779 (0.01)
LR	500	50	0.9177 (0.027)	0.9768 (0.017)
SVC			0.9183 (0.034)	0.9790 (0.012)
ANN			0.8853 (0.054)	0.9529 (0.035)
LR	500	100	0.875 (0.030)	0.9586 (0.017)
SVC			0.877 (0.034)	0.9568 (0.021)
ANN			0.8033 (0.046)	0.8880 (0.041)

(a) Results for $N = 500$.

Model	N	p	Accuracy	AUC
LR	2000	10	0.9183 (0.015)	0.9779 (0.006)
SVC			0.9174 (0.016)	0.9776 (0.006)
ANN			0.9148 (0.016)	0.9752 (0.007)
LR	2000	30	0.9432 (0.011)	0.9897 (0.003)
SVC			0.9412 (0.009)	0.9889 (0.003)
ANN			0.938 (0.014)	0.9864 (0.004)
LR	2000	50	0.9523 (0.012)	0.9932 (0.003)
SVC			0.9528 (0.012)	0.9928 (0.003)
ANN			0.9486 (0.01)	0.9905 (0.004)
LR	2000	100	0.9458 (0.010)	0.9889 (0.005)
SVC			0.9484 (0.012)	0.9918 (0.003)
ANN			0.9439 (0.014)	0.9909 (0.004)

(b) Results for $N = 2000$.

Table 2: *Experiment 1: The mean value and standard deviation for accuracy and AUC for different N and p.*

Experiment 2

Model	N	p	Accuracy	AUC
LR	500	7	0.8823 (0.03)	0.9574 (0.017)
SVC			0.882 (0.03)	0.9554 (0.018)
ANN			0.8703 (0.033)	0.9456 (0.019)
LR	500	10	0.879 (0.03)	0.9574 (0.016)
SVC			0.8793 (0.028)	0.9556 (0.016)
ANN			0.8403 (0.0411)	0.9263 (0.032)
LR	500	15	0.8773 (0.03)	0.9477 (0.022)
SVC			0.864 (0.034)	0.9436 (0.022)
ANN			0.813 (0.0436)	0.8743 (0.0357)

(a) Results for $N = 500$.

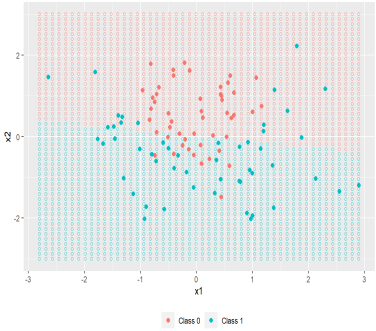
Model	N	p	Accuracy	AUC
LR	2000	7	0.8869 (0.015)	0.9575 (0.008)
SVC			0.8868 (0.016)	0.9573 (0.008)
ANN			0.8815 (0.0154)	0.9545 (0.008)
LR	2000	10	0.8898 (0.016)	0.9621 (0.009)
SVC			0.889 (0.016)	0.9618 (0.009)
ANN			0.8844 (0.0173)	0.9588 (0.0104)
LR	2000	15	0.8952 (0.014)	0.9669 (0.006)
SVC			0.8957 (0.016)	0.9659 (0.007)
ANN			0.8911 (0.018)	0.9616 (0.008)

(b) Results for $N = 2000$.

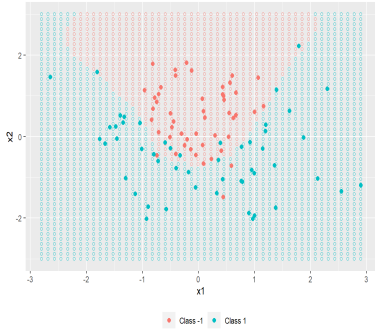
Table 3: *Experiment 2: The mean value and standard deviation for accuracy and AUC for different N and p.*

Experiment 3

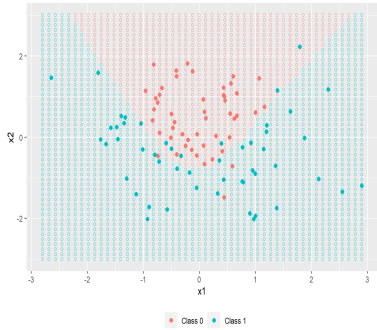
Part 1



(a) Logistic Regression



(b) Support vector machine



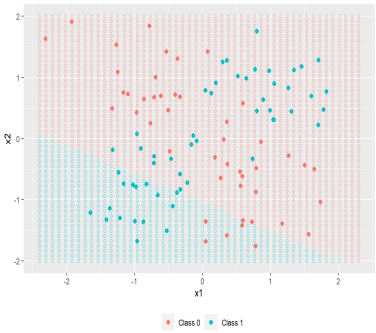
(c) Neural Network

Figure 5: *Experiment 3 part 1: Predicted decision boundaries for the first simulated test dataset.*

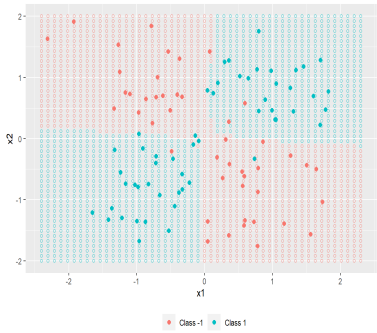
Model	Accuracy	AUC
LR	0.74 (0.038)	0.8169 (0.037)
SVM	0.884 (0.03)	0.9579 (0.015)
ANN	0.906 (0.029)	0.9674 (0.014)

Table 4: *Experiment 3 part 1: The mean value and standard deviation for accuracy and AUC*

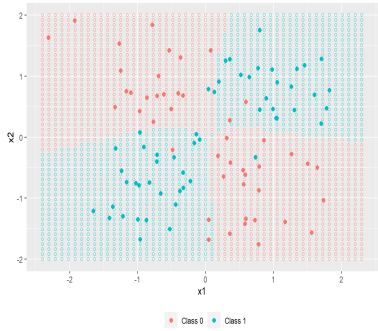
Part 2



(a) Logistic Regression



(b) Support vector machine



(c) Neural Network

Figure 6: *Experiment 3 part 2: Predicted decision boundaries for the first simulated test dataset.*

Model	**Accuracy**	**AUC**
LR	0.467 (0.08)	0.4567 (0.04)
SVM	0.8877 (0.04)	0.9414 (0.03)
ANN	0.888 (0.04)	0.9551 (0.023)

Table 5: *Experiment 3 part 2: The mean value and standard deviation for accuracy and AUC.*

4.2 Real-world data

Method	Accuracy	AUC
LR	0.9738	0.9938
SVC	0.9772	0.9949
SVM	0.9298	0.9859
ANN	0.9685	0.9940

Table 6: *The average accuracy and AUC obtained with 5-fold cross-validation for the breast cancer dataset.*

Method	Accuracy	AUC
LR	0.9231	0.9757
SVC	0.9307	0.9666
SVM	0.9750	0.9940
ANN	0.9500	0.9809

Table 7: *The average accuracy and AUC obtained with 5-fold cross-validation for the diabetes dataset.*

5 Discussion

5.1 Predictive power

Experiment 1 and 2

In Table 2, the results from the first experiment are presented.

In the case $N = 500$ the average accuracy and AUC scores are very similar for the LR and the SVC models for all p, while the average scores for the ANN models are slightly lower when $p = 10, 30$ and the difference increases as the number of predictors increases. We can see that the average scores seem to increase slightly as we go from 10 to 30 predictors for all methods, but after that, they decrease. The most rapid decrease happens for the ANN models. These results are quite expected. The increase in the number of predictors leads to a higher number of parameters in the model. This, in turn, makes it more computationally challenging to train the model and thus more data is needed. Because ANN have more parameters than the other models, they are more affected by the increase in dimensionality.

In the case $N = 2000$, the average scores for all models are quite similar, with marginal differences between the classifiers. However we observe quite unexpected results. The average scores for accuracy and AUC seem to increase for all models as p increases (although it appears to reverse when $p = 100$). One possible explanation that we could find for this lies in the way we simulated the data. During the simulation process, the first step in computing the response variable was to calculate a linear combination of the predictors. Since these predictors were simulated from standard normal random variables, the sum was also normally distributed with a variance equal to the sum of individual variances (since the predictors were independent). Thus adding more variables increased the variance and in turn led

to more separated classes.[10] In other words, the improved performance is not due to the models becoming better at separating the data, but because the data is more separated. This could probably have been avoided if we let the magnitude of the vector of coefficients be equal to 1.

In Table 3, the results for the second experiment are summarized.

In the case $N = 500$, the average scores for the LR and SVC models are very similar. The average scores for the ANN model is similar to the LR and SVC models when $p = 7$, however, as p increases, the predictive performance of the ANN models decreases more rapidly.

In the case $N = 2000$, all models have similar performance, and the difference between classifiers stays more or less the same as p increases. We can see that the average values for accuracy and AUC increase as p increases in the case of mixed variables too, but not as much. Likely due to the fact that the variance of the sum of the predictors doesn't increase as much.

Experiment 3

In Table 4 and Figure 5 the results for the first part of experiment 3 are presented. We see that the model with the best predictive performance is the ANN model, followed by the SVM model with the RBF kernel. As expected, the LR model is not capable of capturing the non-linearity of the data, as illustrated in Figure 5. However, it still performs reasonably well in terms of the performance metrics. Even though the LR model cannot handle non-linear patterns, its straight-line decision boundary still does a pretty good job of separating the data in this case. Both the ANN and the SVM model are capable of producing the correct decision boundary. In Figure 5, we can see that both methods generate very similar decision boundaries.

In Table 5 and Figure 6 the results for the second part of experiment 3 are presented. We see that the ANN model is once again the best model in terms of predictive power, but the difference between the ANN and SVM with the RBF kernel is very small. In Figure 8, we see that both models are capable of generating correct decision boundaries. The LR model, on the other hand, performs very poorly. The accuracy and AUC scores indicate that it performs worse than if the model would just predict the same class every time.

Real data

In Table 6, the results for the breast cancer dataset are presented. The method with the highest accuracy and AUC is the SVC model, followed by the LR model. The ANN model has slightly lower performance, but the difference between classifiers is quite small. The SVM model with the RBF kernel has overall good performance but lower than the other methods. Both SVC and LR produce linear (hyperplane) decision boundaries, leading us to conclude that the classes in this dataset can be

effectively separated with a hyperplane. Thus, using the SVM with RBF kernel probably leads to overfitting.

In Table 7 the results for the diabetes dataset are presented. The model with the highest accuracy and AUC is the SVM model with the RBF kernel, followed by the ANN model. The LR and SVC models both have very good performance but lower than the orther methods. This makes us conclude that there is some non-linearity in this data, but that a hyperplane is able to separate the classes pretty well.

5.2 Improvements

In this study, we only looked at the predictive power of the methods, but it would also be interesting to look at the interpretabilty of the models. In real-life applications, it is often important to try and understand the relationship between the response variable and the explanatory variables, especially when trying to understand what factors play a role in the development of a serious disease like cancer or diabetes.

Another improvement that we could do would be to use a different simulation method to avoid the problem that we encountered in this simulation study.

5.3 Conslusions

In this study, we compared three methods for binary classification: logistic regression, support vector machines, and artificial neural networks. The results indicate that LR and SVC models exhibit very similar performance when the data can be separated by a hyperplane. On small and high-dimensional datasets, ANN models seem to have worse performance. However, for larger datasets, their performance becomes comparable to the other methods.

For data that cannot be separated by a linear boundary (hyperplane), there is some indication that LR and SVC models still perform similarly, but they exhibit worse performance compared to the ANN and SVM models. ANN models tend to perform slightly better on lower-dimensional data, while SVM with an RBF kernel performs better on higher-dimensional data. Overall, their performance is quite comparable.

6 References

[1] Charu C. Aggarwal. *Neural Networks and Deep Learning.* 2018.

[2] P Amini et al. "Evaluating the High-Risk Groups for Suicide: A Comparison of Logistic Regression, Support Vector Machine, Decision Tree, and Artificial Neural Network". In: *Iranian Journal of Public Health* 45.9 (2016), pp. 1179–1187.

[3] A Arora et al. "Comparison of logistic regression, support vector machines, and deep learning classifiers for predicting memory encoding success using human intracranial EEG recordings". In: *Journal of Neural Engineering* 15.6 (Dec. 2018), p. 066028. DOI: 10.1088/1741-2552/aae131.

[4] Geoff Dougherty. *Pattern Recognition and Classification.* 2013.

[5] Tom Fawcett. *An introduction to ROC analysis.* 2005.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* http://www.deeplearningbook.org. MIT Press, 2016.

[7] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* 2nd ed. 2009.

[8] RW Issitt et al. "Classification Performance of Neural Networks Versus Logistic Regression Models: Evidence From Healthcare Practice". In: *Cureus* 14.2 (Feb. 2022), e22443. DOI: 10.7759/cureus.22443.

[9] Frank E. Harrell Jr. *Regression Modeling Strategies.* 2nd ed. 2015.

[10] Amanda Möller. *Predictive Power of Logistic Regression versus Random Forest: A simulation study.* 2019. URL: https://www.math.su.se/publikationer/uppsatsarkiv/.

[11] Joachim Sester et al. "A comparative study of support vector machine and neural networks for file type identification using n-gram analysis". In: *Forensic Science International: Digital Investigation* 36 (2021). DFRWS 2021 EU - Selected Papers and Extended Abstracts of the Eighth Annual DFRWS Europe Conference, p. 301121. ISSN: 2666-2817. DOI: https://doi.org/10.1016/j.fsidi.2021.301121. URL: https://www.sciencedirect.com/science/article/pii/S2666281721000184.

[12] SH Teshnizi and SM Ayatollahi. "A Comparison of Logistic Regression Model and Artificial Neural Networks in Predicting Student's Aca-

demic Failure". In: *Acta Informatica Medica* 23.5 (Oct. 2015), pp. 296–300. DOI: 10.5455/aim.2015.23.296-300.

[13] Vladimir Vapnik. *The nature of statistical learning theory.* 1995.

[14] Ji Zhu and Trevor Hastie. "Kernel Logistic Regression and the Import Vector Machine". In: *Journal of Computational and Graphical Statistics* 14.1 (2005), pp. 185–205. DOI: 10.1198/106186005X25619.