university of groningen

faculty of science and engineering

A comparison of discriminant analysis, support vector machine, random forest, and neural network supervised learning classification methods with the use of completely synthetic data.

Bachelor's thesis applied mathematics

July 30, 2021

Student: Maike Nützel, s2722925

First supervisor: Dr. W. P. Krijnen

Second assessor: Dr. Ir. R. Luppes

Abstract: This Bachelor's project generates challenging data and compares statistical classification models linear and quadratic discriminant analysis with the supervised learning methods support vector machine, random forest and neural networks. These methods are compared in classification accuracy, precision, recall, and F1 score. In the case of a small number of observations, QDA or RF is the classifier with the highest chance of success. Furthermore, in the case of linearly separable data, LDA will work just as well as QDA, RF, SVM, and NN.

1 Introduction

This Bachelor's project will focus in generating challenging data to in order to compare the traditional statistical classification models linear and quadratic discriminant analysis with the supervised learning methods support vector machine, random forest and neural networks. These methods will be compared in classification accuracy, precision, recall, and F1 score.
The goal of the research is to compare the classification models and find a road map in choosing the right classifier for different types of data. The research question we aim to answer in this project is: on what type of data does the classical classification models linear and discriminant analysis outperform supervised learning classification models random forest, support vector machine, and neural networks?
The report is written in the following structure: chapter 2 focuses on the theoretical framework of the five classification models. Chapter 3 illustrates the method in which the data is generated and the models are evaluated. In chapter 4 the results are outlined and discussed. The final chapter is comprised of the conclusion.

2 Theoretical Framework

2.1 Types of data

Before we dive into the different types of classification models, it is perhaps important to differentiate between two different types of data: linearly separable data and data that is not linearly separable.
Data that is linearly separable means that the data can be separated by a linear hyperplane. A hyperplane is defined as a flat affine subspace of a p-dimensional space of dimension $p-1$, defined as an equation in (2.1).

$$\beta_0 + x^T\beta = 0 \tag{2.1}$$

For example, consider a dataset with only two independent variables x_1 and x_2. Since this data is 2-dimensional, it is linearly separable if it can be separated by a line, see figure 2.1. The dots are the observations, and the ellipses represent the 95% prediction confidence. The hyperplane separating the data is the black line through the figure.

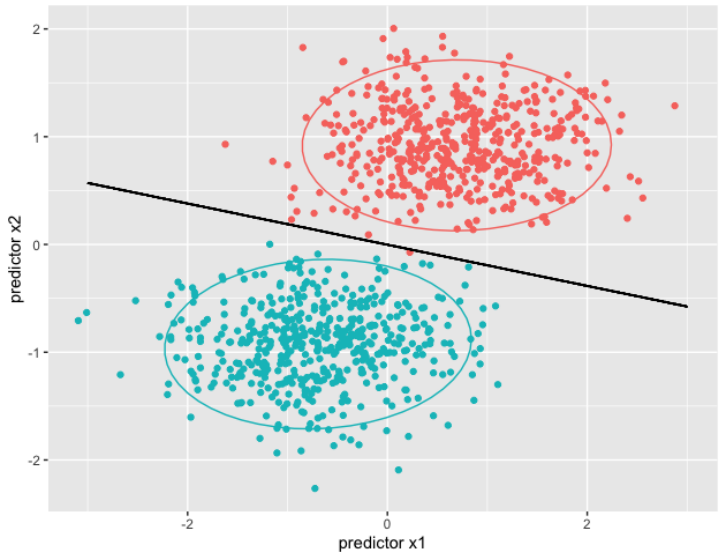

Figure 2.1: Example of linearly separable data

Data that is not linearly separable is data in which a linear hyperplane will not successfully separate the dependent variables. See for example the data in figure 2.2. Here, once again, the ellipses represent the 95% prediction confidence. The line which does successfully separate the data in this figure is not linear. If it were, it would separate the response variables with low accuracy.

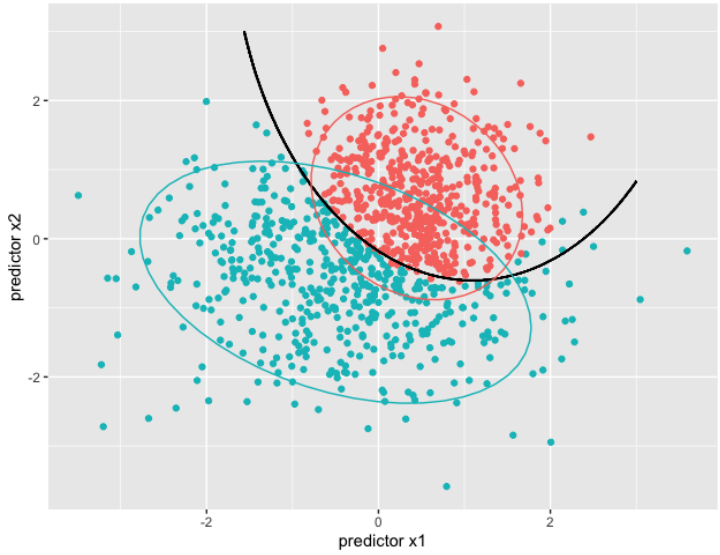

Figure 2.2: Example of non-linearly separable data

So far we have only considered binary response data, for which the dependent variables can only take two values: 'true' or 'false' (1 or 0). For non-binary response, for example response with three categories, the data is separated by different hyperplanes computed between each pair of responses. See for example figure 2.3 for linearly separable data with three possible responses.

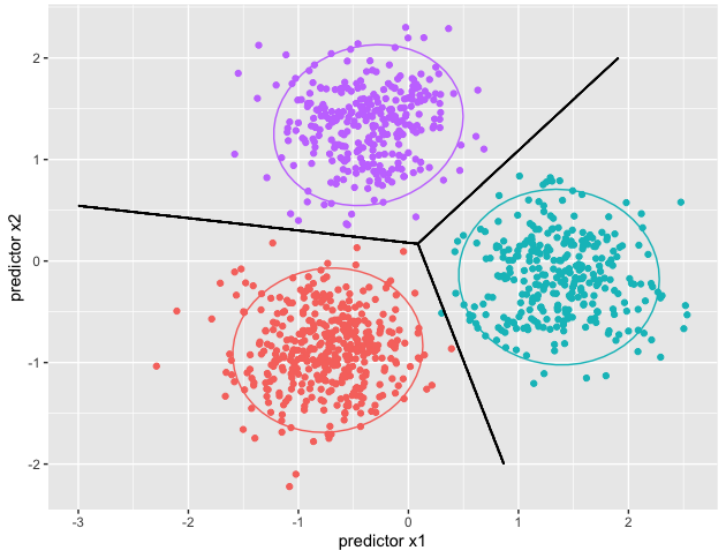

Figure 2.3: Example of linearly separable data with 3 responses

2.2 Discriminant Analysis

Discriminant analysis is a traditional statistical classification method. It is based on a probability theorem by Bayes [2] that flips conditional probabilites. Bayes' thoerem states that:

$$p_k(x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}. \tag{2.2}$$

In this theorem, π_k is the prior probability of class k, hence $\sum_k^K \pi_k = 1$. For example in a binary response model where $\pi_0 = \pi_1$ then $\pi_k = 0.5$ for $k = 0, 1$. Furthermore, p_k represents the posterior probability that an observation $X = x$ belongs to the k-th class, given the predictor value. Hence $p_k = P(Y = k|X = x)$.

Discriminant Analysis works by firstly assuming that each class is normally distributed:

$$f_k(x) = \frac{1}{(2\pi)^{p/2}\left|\mathbf{\Sigma}_k\right|^{1/2}} e^{-\frac{1}{2}(x-\mu_k)^T \mathbf{\Sigma}_k^{-1}(x-\mu_k)}. \tag{2.3}$$

This distribution is then plugged into Bayes formula (2.2) for each class. This equation can be simplified and after having taken the natural logarithm of the resulting equation, we arrive at (2.4).

$$\delta_k(x) = -\frac{1}{2}\left(x-\mu_k\right)^T \mathbf{\Sigma}_k^{-1}\left(x-\mu_k\right) - \frac{1}{2}\log\left|\mathbf{\Sigma}_k\right| + \log\pi_k \tag{2.4}$$

The class k for which $\delta_k(x)$ is highest is the class to which the observation will be assigned.
In Linear Discriminant Analysis (LDA) the data is split with a linear decision boundary. The decision boundary is the black line that splits the binary response data in Figure 2.1 and the multi-class response data in Figure 2.3. The decision boundary is the set of points x where the (log) probability of two outcomes is equal. So in predicting a binary output (0 or 1), the decision boundary is therefore $\{x : \delta_1(x) = \delta_0(x)\}$ where the $\delta_k(x)$ represents the equation (2.4). However, in LDA the classes are assumed to have equal variance, hence

$$\mathbf{\Sigma}_k = \mathbf{\Sigma}_l,\ \forall l.$$

This assumption simplifies the equation and, most importantly, cancels out the quadratic terms in (2.4). Hence for LDA the δ_k becomes:

$$\delta_k(x) = x^T\mathbf{\Sigma}^{-1}\mu_k - \frac{1}{2}\mu_k^T\mathbf{\Sigma}^{-1}\mu_k + \log\pi_k \tag{2.5}$$

The decision boundary is therefore where:

$$x^T\mathbf{\Sigma}^{-1}\mu_1 - \frac{1}{2}\mu_1^T\mathbf{\Sigma}^{-1}\mu_1 + \log\pi_1 = x^T\mathbf{\Sigma}^{-1}\mu_0 - \frac{1}{2}\mu_0^T\mathbf{\Sigma}^{-1}\mu_0 + \log\pi_0 \tag{2.6}$$

Note that by this assumption of equal variance, the decision boundary in linear in x

In quadratic discriminant analysis (QDA) the classes are still assumed to be drawn from normal distributions but the equal variance assumption is no longer made. This leaves us with the original equation (2.4) which can be rewritten as:

$$\delta_k(x) = -\frac{1}{2}x^T\mathbf{\Sigma}_k^{-1}x + x^T\mathbf{\Sigma}_k^{-1}\mu_k - \frac{1}{2}\mu_k^T\mathbf{\Sigma}_k^{-1}\mu_k - \frac{1}{2}\log\left|\mathbf{\Sigma}_k\right| + \log\pi_k \tag{2.7}$$

In order to construct the function for LDA (2.5), estimates for the covariance matrices $\mathbf{\Sigma}$, mean μ_k, and prior probability π_k must be made, for the descriminant function for QDA (2.7) the covariance matrix has to be estimated seperately for each class. The parameters are estimated with the following formulae [2]:

$$\hat{\pi}_k = N_k/N \tag{2.8}$$

$$\hat{\mu}_k = \sum_{g_i=k} x_i/N_k \tag{2.9}$$

$$\hat{\mathbf{\Sigma}} = \sum_{k=1}^{K}\sum_{g_i=k}(x_i-\hat{\mu_k})(x_i-\hat{\mu}_k)^T/(N_K) \tag{2.10}$$

Where N_k is the number of observations in class k.

Which discriminant function will outperform the other depends on the data. For data that is not linearly severable, LDA will not perform well, or at all, see Figure 2.4. Its clear that this linear decision boundary is much less accurate than the one in Figure 2.2. In other data sets, the LDA will perform no better than chance alone, see the attempt at a decision boundary in Figure 2.5.

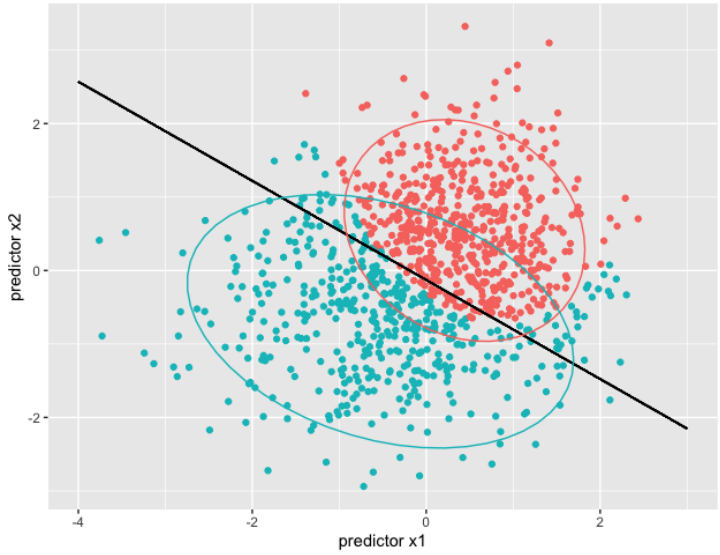

Figure 2.4: Example on LDA on non-linearly seperable data

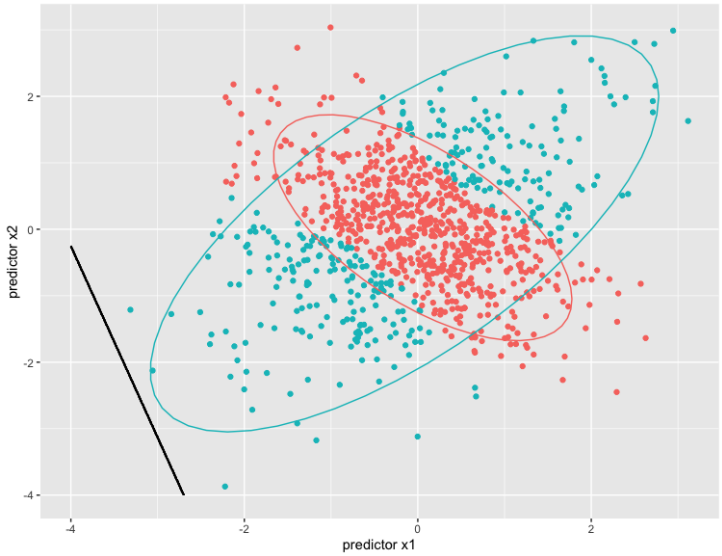

Figure 2.5: Another example of LDA on non-linearly separable data

For data that is linearly separable, there exists a trade-off in complexity between variance and bias .
Firstly, the amount of parameters in both methods. In order to construct the discriminant function for LDA, $p+1$ parameters are needed to construct where p is the number of predictors. This is since it takes p predictors to estimate the mean, 1 to estimate the prior probability. The covariance matrix does not need additional parameters since it is defined in terms of the already predicted means. Hence for a dataset with K classes, LDA estimates $(K-1)(P+1)$ parameters. Due to having to estimate separate covariance matrices, QDA on the other hand also takes p predictors to estimate the mean and 1 to estimate the prior probability but also needs $p(p+1)/2$ to estimate the covariance matrices. Hence together this is $1+p(p+2)/2$ parameters. For a dataset with K classes, QDA estimates $(1+p(p+2)/2)(K-1)$ parameters. Thus QDA has a much higher complexity than LDA and increases in complexity quadratically in p while LDA increases linearly in p.
Secondly, the bias. In QDA the high amount of parameters can lead to high variance. However, on the other hand, in a dataset where the classes have very different variances, LDA will classify with a very high bias due to the equal variance matrix assumption.
The performance of QDA and LDA is therefore dependent on the dataset with LDA generally performing

better than QDA on datasets where the training set is small.

2.3 Random Forest

2.3.1 Classification Trees

To understand Random Forest (RF), we must first gain a foundation in classification trees. The classification tree is a tree that consists of a root node, branches, internal nodes, and terminal nodes or leaves. Each node represents a decision which splits the data based on a condition. In the classification tree algorithm, the first step is to build the tree with recursive binary splitting. This means that for every independent variable a cost function is computed that represents how predictive the variable is for the outcome. The most predictive variable (hence the variable with the lowest cost) will become the root node. The algorithm is recursive so the same algorithm will be performed on the child nodes until a stop condition is reached. There are different possibilities for the cost function, the RF algorithm from the `randomForest` package [5], uses the Gini index (G).

$$G = \sum_{k=1}^{K} p_{mk}(1 - p_{mk}) \tag{2.11}$$

Where p_{mk} is the proportion of test observations in m^{th} input variable that are from the k^{th} class.

2.3.2 Random Forest

In Random forrest (RF) a certain amount of classification trees are made from bootstrapped samples (of $63,2\%$) from the training set. This sample is called 'the in-bag set'. The remaining $36,8\%$ is known as 'the out-of-bag data'. Each tree may only train using a random sample $\sqrt{p}$ of predictors for each split, where p is the amount of predictors. The out-of-bag sample is then used to test the trees. Out of the all the trees produces, the classification choice is made by majority vote.

The advantage of RF is that classification trees are produced without considering strongest predictor. This is a good counter-measure against overfitting.

2.4 Support Vector Machine

2.4.1 Binary response classification

Support Vector Machine (SVM) is built as a binary classifier, this can be extended to multi-class classification. We will primarily explain SVM for two response variables and then touch on how SVM classifies for more than two classes in 2.4.2.
SVM is an extension of the Support Vector Classifier (SVC) which is in turn an extension of the Maximum Margin Classifier (MMC). If the two classes are separable by a linear boundary, MMC works by finding a maximal margin hyperplane. It is defined by equation (2.1).
Hence the classification is decided by the sign on the function $f(x)$

$$f(x) = \beta_0 + x^T\beta \tag{2.12}$$

Thus given training observations $x_1, \ldots, x_n \in \mathbb{R}^p$ and response variables $y_1, \cdots, y_n \in \{-1, 1\}$, MMC works maximizing the margin M, hence finding a solution to:

$$\max_{\beta, \beta_0, \|\beta\|_2 = 1} M \tag{2.13}$$

$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq M \;\forall\; i \;=\; i, \ldots, n \tag{2.14}$$

Where $\|\cdot\|_2$ represents the Euclidean norm.
In the case that a separating hyperplane does not exist, there will exist no solution to equations (2.13) - (2.14). SVC bypasses this issue by finding a hyperplane that separates the classes with a margin that allows misclassification if it will ensure better overall classification for new data points. SVC also works in the case that a hyperplane does exist but the plane is too sensitive to new data (hence will perform well

on the training set but not on the test test) for MMC. SVC works by maximizing the margin M just as MMC but a conditional is added. SVC finds a solution to:

$$\max_{\beta,\beta_0,\epsilon,\|\beta\|_2=1,} M \tag{2.15}$$

$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq M(1-\epsilon_i) \tag{2.16}$$

$$\text{and } \epsilon_1 \geq 0,\ \sum_{i=1}^{n} \epsilon_1 \leq C \tag{2.17}$$

Here, The ϵ_i is a is a slack variable. It is a location parameter for the ith observation relative to the margin. $\epsilon_i = 0$ indicates that the i^{th} observation is right side of the margin, $\epsilon_i > 0$, indicates that it is on the wrong side. C is known as the turning parameter, and since it bounds the sum of the slack variables ϵ_i, it determines the amount violations that the margin will tolerate.
In the case that the data is not linearly separable, the extension of SVC, is to be used: Support Vector Machine (SVM). SVM produces a function $g(x)$, the sign which makes the classification decision:

$$g(x) = \beta_0 + h(x)^T\beta \tag{2.18}$$

$h(x_i) = (h_1(x_i), h_2(x_i), \ldots, h_n(x_i))$ where h_i are the basis functions. Which changes the minimization problem to :

$$\max_{\beta,\beta_0,\epsilon,\|\beta\|_2=1,} M \tag{2.19}$$

$$\text{subject to } y_i(\beta_0 + h(x_i)^T\beta) \geq M(1-\epsilon_i) \tag{2.20}$$

$$\text{and } \epsilon_1 \geq 0,\ \sum_{i=1}^{n} \epsilon_1 \leq C \tag{2.21}$$

In order to explain how SVM works, we must first compute the solution of (2.19) - (2.21). First, the $\|\beta\|_2 = 1$ constraint can be combined with equation (2.16):

$$\frac{1}{\|\beta\|} y_i(\beta_0 + h(x_i)^T\beta) \geq M(1-\epsilon_i)$$

Then we can arbitrarily set $\|\beta\| = \frac{1}{M}$ to simplify the equation. Now we can substitute (2.15) - (2.17) to:

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N} \epsilon_i \tag{2.22}$$

$$\text{subject to } y_i(\beta_0 + h(x_i)^T\beta) \geq 1-\epsilon_i \tag{2.23}$$

$$\epsilon_i \geq 0 \tag{2.24}$$

In order to solve this minimization problem, we must minimize the Lagrange function with respect to $\beta_0, \beta, \epsilon_i$[2].

$$L = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N} \epsilon_i - \sum_{i=1}^{N} \alpha_1[y_i(\beta_0 + h(x_i)^T\beta) - (1-\epsilon_i)] - \sum_{i=1}^{N} \mu_i\epsilon_i \tag{2.25}$$

In this equation the positive constants $\mu_i, \alpha_i, \epsilon_i$ are the positivity constraints. In order to minimize the equation, we take the derivative of the Lagrange function and set it to zero:

$$\frac{1}{\beta} L = -\sum_{i=1}^{N} \alpha_i y_i h(x_i) + \beta = 0, \tag{2.26}$$

$$\frac{1}{\beta_0} L = -\sum_{i=1}^{N} \alpha_i y_i, \tag{2.27}$$

$$\frac{1}{\epsilon_i} L = C - \sum_{i=1}^{N} \alpha_i = 0. \tag{2.28}$$

$$\tag{2.29}$$

Since from (2.26) we know that $\beta = \sum_{i=1}^{N} \alpha_i y_i h(x_i)$ The function (2.18) can therefore be written as:

$$g(x) = h(x)^T \beta + \beta_0 \tag{2.30}$$

$$= h(x)^T (\sum_{i=1}^{N} \alpha_i y_i h(x_i)) + \beta_0 \tag{2.31}$$

$$= \sum_{i=1}^{N} \alpha_i y_i h(x)^T h(x_i) + \beta_0 \tag{2.32}$$

$$= \sum_{i=1}^{N} \alpha_i y_i \langle h(x), h(x_i) \rangle + \beta_0 \tag{2.33}$$

In order to avoid computing $h(x)$, we can use a generalisation of the inner product, the kernel.

$$K(x, x_i) = \langle h(x), h(x_i) \rangle$$

The solution function then becomes:

$$g(x) = \beta_0 + \sum_{i=1}^{N} \alpha_i y_i K(x, x_i) \tag{2.34}$$

Different kernels can be used, for non-linear data the radial kernel is the best choice, and the one used in this thesis.

$$K(x, x_i) = \exp(-\gamma \sum_{j=1}^{p} (x_j - x_{ij})^2) \tag{2.35}$$

What this means intuitively is that the further away a test observation is from a training observation, the smaller $K(x, x_i)$ will be hence the smaller influence the observation will have on the decision function (2.34) [3]

2.4.2 Multi-class classification

Since SVM is built around binary classification, extension to multi-class is not directly possible. The two possible workaround methods are the One-Versus-One or the One-Verus-Many. The SVM classifier from `e1071` package, the one used in this thesis, applies the One-Versus-One method [4] in which a test observation is classified by using SVM on every possible pair of classes and classifying based on the most frequently occurring decision.

2.5 Neural Networks

Neural network is a complex classification model that tries to copy the layout of the human brain in order to make decisions. The most basic version of neural network (NN) architecture is the single-layer network, called the perceptron. The perceptron consists of an input layer and an output layer. The input layer consists of nodes that each transmit an observation $x = (x_1, x_2, \ldots, x_p)$ with a weight $w = (w_1, w_2, \ldots, w_p)$ to an output node. This output node classifies the observation in one of two classes $y \in \{-1, 1\}$.It is a single-layer network because it has only one computational layer, the output layer. Hence the observation is estimated with the following formula:

$$\hat{y} = \Phi(\sum_{j=1}^{p} w_i x_i) \tag{2.36}$$

The Φ is known as an activation function. The activation function translates the computation to an output. A simple example of an activation function is : $\Phi(u) = \text{sign}(v)$. The one that the `neuralnet`[6] package's default is the sigmoid function.

$$\Phi(u) = \frac{1}{1 + e^{-v}} \tag{2.37}$$

The neural network algorithm works by using small inputs in order to update the weights based on the error value:

$$w_{\text{new}} = W_{\text{old}} + \alpha E(x) x \tag{2.38}$$

Where α is a constant representing the learning rate and $E(x)$ is the error of x.

$$E(x) = y - \hat{y} \tag{2.39}$$

The perceptron is a linear model, the data is separated by the hyperplane $\sum_{j=1}^{p} w_i x_i = 0$.It will therefore not perform well on non-linear data. Multi-layer neural networks resolve this issue. Multi-layer means that, unlike with the perceptron, we do not simply only have the output layer as a computing layer but there are additional layers in between the input and output layer which also make computations. These are called the hidden layers because the computations are not visible to the user. In this thesis we will be using `nnet` from `neuralnet`. The model is a single hidden layer neural network which is trained using backpropagation [7].
The backpropagation algorithm is an algorithm with two phases, the forward phase and the backward phase. In the forward phase the training data is fed trough the neural network and all its hidden layers with the current (or initially: random) weights. Then, at the final output, the derivative of the loss function is computed. A loss function representative of the prediction error. Then, in the backward phase, the gradient of the loss function is cothemputed in order to compute new, better, weights. This is repeated until an acceptable loss function is reached.

3 Method

3.1 Generated Data Sets

Synthetic data is produced in order to compare the different classification methods. The data sets all consist of 5 dependent variables, of which 2 have zero predictive power. The data sets differ in the following items:

- Linear- or non-linearly seperably data (L/NL): tells the reader if the data is seperable by a hyperplane or not.
- Amount of response variables (RV): In datasets 1 - 7 the response is binary and in datasets 8-14 the number of response variables is 3.
- Prior probability (PP): for most data sets it is assumed that the response classes are approximately equal in size. In data set 5 and 10, one response is much less likely than the other(s).
- Number of observations (N): The models all have 1000 observations with the exception of dataset 4 and 11 where the classification models are tested on a low number of observations.
- Co-variance matrices (COV): the co-variance matrices between the three predictors x_1, x_2, x_3. Note that if $i = j$:

  $$COV(x_i, x_i) = \sigma^2_{x_i} = var(x_i)$$

  and if $i \neq j$:

  $$COV(x_i, x_j) = corr(x_i, x_j) \cdot \sigma_{x_i} \sigma_{x_j}$$

  Where *corr* is the correlation. Datasets 1,2 and 3 are identical with the exception of the correlation between the predictors. In dataset 1 there is no correlation, in dataset 2 there is a correlation of -0.5 between two predictors in both response classes and in dataset 3 there is a correlation of $-0,5$ between two predictors in only one response class. The same is done for the datasets 8,9,10, and the datasets 6,7.
- Mean (μ) The mean of the three predictive classes. Note that, for example, the three classes in the first data set have the same mean and the classes in dataset 6 are completely separated.

The datasets were synthetically created in R. The data sets are made by firstly making a p uniformly distributed multivariate datasets around a certain mean and variance. Here p is the number of response variables, hence either 2 or 3. these are combines into one big 3 dimensional dataset. Two more independent variables are added to the dataset with no predictive power. With the formula for the LDA or QDA decision boundary the dependent variables for each synthetic observation is computed and added to the dataset. Noise is then added to all the independent variables by adding a random sample from the $N(0, 0.2)$ distribution to each observation:

$$\boldsymbol{x} = \boldsymbol{x} + \mathcal{N}(0, .2) \tag{3.1}$$

Finally, the dataset is standardised and shuffled. Also see appendices A for the R program for a binary response dataset and B for a multi-class reponse dataset.

D	L/NL	RV	PP	N	COV	μ
1	NL	2	0.5, 0.5	10^3	$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}2&0&0\\0&2&0\\0&0&2\end{bmatrix}$,	$\begin{bmatrix}1\\1\\1\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}$
2	NL	2	0.5, 0.5	10^3	$\begin{bmatrix}1&-0.5&0\\-0.5&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}2&-1&0\\-1&2&0\\0&0&2\end{bmatrix}$,	$\begin{bmatrix}1\\1\\1\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}$
3	NL	2	0.5, 0.5	10^3	$\begin{bmatrix}1&-0.5&0\\-0.5&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}2&1&0\\1&2&0\\0&0&2\end{bmatrix}$,	$\begin{bmatrix}1\\1\\1\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}$
4	NL	2	0.5, 0.5	50	$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}2&0&0\\0&2&0\\0&0&2\end{bmatrix}$,	$\begin{bmatrix}1\\1\\1\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}$
5	NL	2	0.7, 0.3	10^3	$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}2&0&0\\0&2&0\\0&0&2\end{bmatrix}$,	$\begin{bmatrix}1\\1\\1\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}$
6	L	2	0.5, 0.5	10^3	$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$,	$\begin{bmatrix}1\\5\\10\end{bmatrix}, \begin{bmatrix}3\\7\\12\end{bmatrix}$
7	L	2	0.5, 0.5	10^3	$\begin{bmatrix}1&-0.5&0\\-0.5&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}1&-0.5&0\\-0.5&1&0\\0&0&1\end{bmatrix}$,	$\begin{bmatrix}1\\5\\10\end{bmatrix}, \begin{bmatrix}3\\7\\12\end{bmatrix}$
8	NL	3	0.33, 0.33 0.34	10^3	$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}2&0&0\\0&2&0\\0&0&2\end{bmatrix}, \begin{bmatrix}3&0&0\\0&3&0\\0&0&3\end{bmatrix}$	$\begin{bmatrix}1\\1\\1\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}$
9	NL	3	0.33, 0.33 0.34	10^3	$\begin{bmatrix}1&-0.5&0\\-0.5&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}2&-1&0\\-1&2&0\\0&0&2\end{bmatrix}, \begin{bmatrix}3&-1.5&0\\-1.5&3&0\\0&0&3\end{bmatrix}$	$\begin{bmatrix}1\\1\\1\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix} \begin{bmatrix}1\\1\\1\end{bmatrix}$
10	NL	3	0.33, 0.33 0.34	10^3	$\begin{bmatrix}1&-0.5&0\\-0.5&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}2&0&0\\0&2&0\\0&0&2\end{bmatrix}, \begin{bmatrix}3&1.5&0\\1.5&3&0\\0&0&3\end{bmatrix}$	$\begin{bmatrix}1\\1\\1\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix} \begin{bmatrix}1\\1\\1\end{bmatrix}$
11	NL	3	0.33, 0.33 0.34	100	$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}2&0&0\\0&2&0\\0&0&2\end{bmatrix}, \begin{bmatrix}3&0&0\\0&3&0\\0&0&3\end{bmatrix}$	$\begin{bmatrix}1\\1\\1\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}$
12	NL	3	0.2, 0.2 0.6	10^3	$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}2&0&0\\0&2&0\\0&0&2\end{bmatrix}, \begin{bmatrix}3&0&0\\0&3&0\\0&0&3\end{bmatrix}$	$\begin{bmatrix}1\\1\\1\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}$
13	L	3	0.33, 0.33 0.34	10^3	$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$	$\begin{bmatrix}1\\5\\10\end{bmatrix}, \begin{bmatrix}3\\7\\12\end{bmatrix} \begin{bmatrix}5\\9\\14\end{bmatrix}$
14	L	3	0.33, 0.33 0.34	10^3	$\begin{bmatrix}1&-0.5&0\\-0.5&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}1&-0.5&0\\-0.5&1&0\\0&0&1\end{bmatrix}, \begin{bmatrix}1&-0.5&0\\-0.5&1&0\\0&0&1\end{bmatrix}$	$\begin{bmatrix}1\\5\\10\end{bmatrix}, \begin{bmatrix}3\\10\\20\end{bmatrix} \begin{bmatrix}5\\9\\14\end{bmatrix}$

Table 3.1: The Datasets

3.2 Classifier Evaluation

Each data set is randomly split in 20% testing set and 80% training set. Repeated 5-fold cross-validation is then used to train classifiers to prevent overfitting. In 5-fold cross-validation, the observations are split up into 5 groups, or folds. The first fold then becomes the test set and the other 4 folds together form the training set. The model is trained using the training set and then tested on the test set. This process is repeated three times with all k folds having served as a test set. Finally we obtain 14 confusion matrices per classification model. The confusion matrix displays the amount of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). For every classification matrix, we get multiple values for these computations considering which response class is chosen as the 'true' value. In order to make conclusions about the classification model in general we consider the overall accuracy (Ac) and the macro-averaged precision (Pr) (also known as positive predictive value) and recall (Re) (also known as sensitivity or true-positive rate). Macro-averaged simply means the mean of the two or three values for recall and precision. The accuracy considers amount of correct classifications. This can, however, give very inaccurate results, for example in the case of the dataset where the classifier is to classify into either 'man' or 'woman' and a dataset in which 90% is men. If the classifier then randomly classifies all the observations as men, it would get a 90% accuracy rate but would in fact be performing very poorly. This is why we also consider precision, recall, and F1. Recall measures the amount of true positives out of all the results, hence measures how well a test can identify true positives. Precision tests how much of the positives are true positives. A classifier with a high accuracy but a low precision score would therefore not be useful in the application where a false positive is not desirable.The most concise method of comparing classification models is the F1 score, the harmonic mean of the precision and recall score. It is a general measure of accuracy between 0 and 1. 1.0 therefore indicates perfect recall and precision. We will be using the traditional $F1$ score where the precision and recall are assumed to be of equal importance. Below are the formulas used to calculate these scores and the Bayesian conditional probability interpretation of the score. Here 1 represents the case where the response variable is 1, K is either 2 or 3 , depending on the

number of responses, and π_i is the prior probability.

$$Accuracy = \sum_{i=0}^{K} P(y \text{ is classified as } i | y = i)\pi_i = \frac{TP + TN}{TP + TN + FN + FP} \tag{3.2}$$

$$Recall = P(y \text{ is classified as } 1 | y = 1) = \frac{TP}{TP + TN} \tag{3.3}$$

$$Precision = P(y = 1 | y \text{ is classified as } 1) = \frac{TP}{TP + FP} \tag{3.4}$$

$$F1 = \frac{2}{recall^{-1} + precision^{-1}} \tag{3.5}$$

The R programs for the classification models can be found in appendices C - G

4 Results and Discussion

4.1 Accuracy, precision, and recall

The accuracy, precision, and recall of the datasets per classifier are summarised in tables 4.2,4.3 and 4.4. The missing values in the precision column is caused by the fact when the amount of true and negatives positives is zero. In this case, the classifier is definitely inaccurate for that dataset since zero amount of positives never occurs.
As is expected, the LDA only produces accurate results for datasets 6,7, 13 and 14 since these were the datasets where the datapoints were separable by a linear hyperplane. In these sets LDA either outpeforms the other classifiers or performs with simuliar accuracy, precision and recall. However, unexpectibly, LDA did not outpeform QDA in the case of a small set (and thereby training set) as was expected.
The importance of the precision and recall as evaluation of the classifier is highlighted in dataset 5. If one were to evaluate a classifier purely on accuracy, it would seem that LDA performs fine on this dataset but in reality the confusion matrix tells a different story. All the instances were classified to class 0 but since

	0	1
0	181	0
1	19	0

Table 4.1: Confusion matric of dataset 5

this test set only had a small amount of observations belonging to the class 1, the accuracy is very high ($\frac{181}{200} = 0.905$) when in reality the classifier is very inaccurate.
QDA peforms well on almost all datasets. It outperforms all the other classifiers in the case of dataset 4, the dataset of with a binary response and a low number of observations. In the little difference in performance between data sets 1, 2, 3, between 6 and 7, and 8, 9 and 10, we can conclude that performance of the model is not significantly influenced by correlation between predictors. QDA slightly outperforms all other classifiers in the case of dataset 2, the dataset with correlation between two predictors in both response classes, this is not the case in dataset 9 where which is the same setup of correlation but in the case of three response variables.
Random forest performs well overall but cannot classify better than chance in dataset 4. RF outperforms SVM in the cases of dataset 5, the one with the binary response where the response classes are of different sizes, and dataset 11, the case where the number of observations were low. This is however not the case for datasets 4 and 12, the other classes with low amount of observations or unbalanced response classes. SVM slightly outperforms RF on almost every dataset but miss-classifies many observations in the case of dataset 5, a dataset with binary response where the response classes are of different sizes, and dataset 11, the case where the number of observations were low.
NN performs well overall with most accuracy above 90%. It cannot classify better than chance alone in the case of dataset 11 where the amount of observations are low.
There is no clear difference between performance of the datasets between correlated and non-correlated datasets.

D	LDA			QDA		
	Ac	Pr	Re	Ac	Pr	Re
1	0.59		0.50	0.89	0.88	0.88
2	0.63	0.81	0.53	0.94	0.93	0.93
3	0.61	0.81	0.51	0.90	0.89	0.90
4	0.60	0.56	0.54	0.80	0.79	0.79
5	0.91	tg	0.50	0.93	0.86	0.74
6	0.98	0.98	0.98	0.98	0.99	0.98
7	0.99	0.99	1.00	0.99	0.99	1.00
8	0.49		0.35	0.81	0.78	0.76
9	0.48		0.33	0.80	0.78	0.75
10	0.55		0.34	0.80	0.76	0.76
11	0.65		0.33	0.70	0.63	0.63
12	0.52		0.30	0.88	0.94	0.64
13	0.97	0.98	0.98	0.97	0.97	0.97
14	0.98	0.98	0.98	0.98	0.99	0.98

Table 4.2: Accuracy, Precision, Recall of LDA and QDA

D	RF			SVM		
	Ac	Pr	Re	Ac	Pr	Re
1	0.93	0.93	0.92	0.91	0.90	0.89
2	0.90	0.90	0.89	0.91	0.89	0.90
3	0.90	0.90	0.89	0.92	0.92	0.91
4	0.60	0.62	0.60	0.70	0.81	0.70
5	0.92	0.85	0.81	0.85		0.50
6	0.98	0.98	0.99	0.98	0.98	0.99
7	0.97	0.97	0.97	0.99	0.99	1.00
8	0.81	0.78	0.77	0.84	0.79	0.78
9	0.84	0.79	0.80	0.84	0.83	0.81
10	0.82	0.80	0.77	0.80	0.76	0.74
11	0.85	0.75	0.70	0.65	0.47	0.53
12	0.87	0.91	0.77	0.90	0.94	0.73
13	0.96	0.97	0.97	0.99	0.99	0.99
14	0.97	0.97	0.97	0.98	0.98	0.99

Table 4.3: Accuracy, Precision, Recall of RF and SVM

D	NN		
	Ac	Pr	Re
1	0.93	0.92	0.93
2	0.91	0.91	0.91
3	0.93	0.92	0.92
4	0.70	0.83	0.62
5	0.94	0.93	0.83
6	0.97	0.97	0.98
7	0.99	0.99	1.00
8	0.82	0.78	0.80
9	0.84	0.80	0.79
10	0.83	0.81	0.80
11	0.45	0.42	0.46
12	0.90	0.80	0.77
13	0.98	0.98	0.98
14	0.98	0.98	0.98

Table 4.4: Accuracy, Precision, Recall of NN

4.2 Comparison with F1 score

Comparing the F1-scores, it can be remarked that QDA outperforms RF, SVM and NN in the case of low number of observations for a binary response in dataset 4. Furthermore it can be observed in the case of linearly separable data (dataset 6,7,13, and 14), that LDA has an accuracy equal to the other models. In the case of dataset 11, which is the low amount of observations for a multi-class response, only the QDA and RF perform adequately.

D	LDA	QDA	RF	SVM	NN
1		0.88	0.93	0.90	0.92
2	0.43	0.93	0.90	0.90	0.91
3	0.40	0.89	0.89	0.91	0.92
4	0.52	0.79	0.58	0.67	0.60
5		0.79	0.83		0.87
6	0.98	0.98	0.98	0.98	0.97
7	0.99	0.99	0.97	0.99	0.99
8		0.76	0.78	0.78	0.79
9		0.76	0.79	0.82	0.79
10		0.75	0.78	0.75	0.81
11		0.62	0.70		0.42
12		0.66	0.82	0.78	0.78
13	0.97	0.97	0.97	0.99	0.98
14	0.98	0.98	0.97	0.98	0.98

Table 4.5: F1 score of classification models

5 Conclusion

This goal of this research was to describe a road map to choosing the right classification model for different types of synthetic data. Based on 14 datasets that differ in size, correlation, class balance, number of response variables, and mean, it is clear that no significant classifier stood out as the best for most of these variables.

It cannot be concluded on which type of data sets discriminant analysis outperforms random forest, support vector machine, and neural network. In can be said that in the case of a low amount of observations, that QDA is a classifier with the higher chance of success than most other classifiers. It is only RF that performd simularly. It was also apparent that in the cas of linearly seperable data, LDA will work just as well as QDA, RF, SVM, and NN.

In this thesis, due to the small number of datasets, choices had to be made in which components in the dataset to vary. In further research, the classification models could be tested on many more datasets which all vary in more components: mean, variance, number of response variables, et cetera. This would give the reader a more complete and concise picture of which dataset works on which classifier.

References

[1] C. C. Aggarwal. *Neural networks and deep learning: a textbook*. Springer, 2019.

[2] T. Hastie, J. Friedman, and R. Tisbshirani. *The Elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2017.

[3] G. James, D. Witten, T. Hatie, and R. Tibshirani. *An Introduction to Statistical Learning: with applications in r*. Springer, 2021.

[4] D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2021. R package version 1.7-6.

[5] F. original by Leo Breiman, R. p. b. A. L. Adele Cutler, and M. Wiener. *Breiman and Cutler's Random Forests for Classification and Regression*, 2018. R package version 4.6-14.

[6] M. N. W. Stefan Fritsch, Frauke Guenther. *Training of Neural Networks*, 2019. R package version 1.44.2.

[7] W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S.* Springer, New York, fourth edition, 2002. ISBN 0-387-95457-0.

A Appendix: Dataset 1

```
set.seed(101)
n <- 1000 # number of observations
mu<- cbind(c(1,1,1),c(1,1,1)); # mean
var1 <- rbind(c(1,0,0),c(0,1,0),c(0,0,1)) #covariance matrix
var2 <- rbind(c(2,0,0),c(0,2,0),c(0,0,2)) #covariance matrix
pp <- c(0.5,0.5)
x.y1 <- as.matrix(rmvnorm(n*pp[1], mu[,1],var1))
x.y2 <- as.matrix(rmvnorm(n*pp[2], mu[,2],var2))

mvndat <- as.matrix(rbind(x.y1, x.y2))

y <- NULL
for(i in 1:n) {
  delta1 <- -0.5*t(mvndat[i,]- mu[,1])%*%inv(var1)%*%(mvndat[i,] - mu[,1]) - (0.5*log(det(var1)) + lo
  delta2 <- -0.5*t(mvndat[i,]- mu[,2])%*%inv(var2)%*%(mvndat[i,] - mu[,2]) - (0.5*log(det(var2)) + lo
  y[i] <- as.integer(delta1 >= delta2)
}
x4 <- rnorm(n, 0, 3)
x5 <- rnorm(n, 0, 3)
dataset <- cbind.data.frame(mvndat, x4, x5)

#added  noise for the indepent variables
for (i in 1:5){
  dataset[,i] = dataset[,i] + rnorm(n,0,.2)
}
# bind and standardize data
dataset <- scale(dataset)
dataset <- cbind.data.frame(dataset,y)
colnames(dataset) <- c('x1', 'x2', 'x3', 'x4', 'x5', 'y')
dataset.shuffle <- dataset[sample(nrow(dataset)),] #shuffle the data

write.csv(dataset.shuffle, file = "/Users/maike/Desktop/Scriptie/Datasetsnew/dataset1.csv",row.names
```

B Appendix : Dataset 7

```
set.seed(108)

n <- 1000 # number of observations
mu<- cbind(c(1,1,1),c(1,1,1),c(1,1,1)); # mean
var1 <- rbind(c(1,0,0),c(0,1,0),c(0,0,1)) #covariance matrix
var2 <- rbind(c(2,0,0),c(0,2,0),c(0,0,2)) #covariance matrix
var3 <- rbind(c(3,0,0),c(0,3,0),c(0,0,3)) #covariance matrix
pp <- c(0.33,0.33,0.34)
x.y1 <- as.matrix(rmvnorm(n*pp[1], mu[,1],var1))
x.y2 <- as.matrix(rmvnorm(n*pp[2], mu[,2],var2))
x.y3 <- as.matrix(rmvnorm(n*pp[3], mu[,3],var3))

mvndat <- as.matrix(rbind(x.y1, x.y2, x.y3))

y <- NULL
for(i in 1:n) {
  delta1 <- -0.5*t(mvndat[i,]- mu[,1])%*%inv(var1)%*%(mvndat[i,] - mu[,1]) - (0.5*log(det(var1)) + lo
  delta2 <- -0.5*t(mvndat[i,]- mu[,2])%*%inv(var2)%*%(mvndat[i,] - mu[,2]) - (0.5*log(det(var2)) + lo
  delta3 <- -0.5*t(mvndat[i,]- mu[,3])%*%inv(var3)%*%(mvndat[i,] - mu[,3]) - (0.5*log(det(var3)) + lo
  if((delta1 >= delta2) && (delta1 >= delta3)){
    y[i] <- 1
  }
  if((delta2 >= delta3) && (delta2 >= delta1)){
    y[i] <- 2
  }
  if((delta3 >= delta2) && (delta3 >= delta1)){
    y[i] <- 3
  }
}

x4 <- rnorm(n, 5, 4)
x5 <- rnorm(n, 2, 3)
dataset <- cbind.data.frame(mvndat, x4, x5)
#add noise
for (i in 1:5){
  dataset[,i] = dataset[,i] + rnorm(n,0,.2)
}
# bind and standardize data
dataset <- scale(dataset)
dataset <- cbind.data.frame(dataset,y)
colnames(dataset) <- c('x1', 'x2', 'x3', 'x4', 'x5', 'y')
dataset.shuffle <- dataset[sample(nrow(dataset)),] #shuffle the data

#partimat(as.factor(y) ~ x1 + x2 , data = dataset, method="qda")
write.csv(dataset.shuffle, file = "/Users/maike/Desktop/Scriptie/Datasets_new/dataset8.csv",row.names
```

C Appendix : Linear discriminant analysis R code

```
library(tidyverse)
library(caret)
library(MASS)
set.seed(201)
lda.results <- list()
lda.accuracy <- list()
lda.conf.matrix <- list()
lda.F1<- list()
for (d in 1 : 14) {
  location <- sprintf("/Users/maike/Desktop/Scriptie/Datasets_new/dataset%s.csv", d)
  dataset <- read.csv(location)
  training.samples.dataset <- createDataPartition(y = dataset$y, p = 0.8,list =FALSE)
  train.dataset <- dataset[training.samples.dataset,]
  test.dataset <- dataset[-training.samples.dataset,]
  train.control <- trainControl(method = "repeatedcv",
                                number = 5, repeats = 3)
  start.time <- Sys.time()
  lda.model <- train(as.factor(y) ~., data = train.dataset,
                     method = "lda", trControl = train.control)
  end.time <- Sys.time()
  time.taken <- end.time - start.time
  print(time.taken)
  predict.lda <- predict(lda.model, test.dataset)
  lda.conf.matrix[[d]] <- confusionMatrix(table(test.dataset$y, predict.lda))
  lda.results[[d]] <-  ml_test(predict.lda, as.factor(test.dataset$y), output.as.table = TRUE)
  lda.accuracy[[d]] <-  ml_test(predict.lda, as.factor(test.dataset$y), output.as.table = FALSE)$accu
  lda.F1[[d]] <-  mean(ml_test(predict.lda, as.factor(test.dataset$y), output.as.table = FALSE)$F1)
}
```

D Appendix : Quadtratic discriminant analysis R code

```
library(tidyverse)
library(caret)
library(MASS)
set.seed(202)
# QUADREATIC DISCRIMINANT ANALYSIS
qda.results <- list()
qda.accuracy <- list()
qda.F1 <- list()
for (d in 1 : 14) {
  location <- sprintf("/Users/maike/Desktop/Scriptie/Datasets_new/dataset%s.csv", d)
  dataset <- read.csv(location)
  training.samples.dataset <- createDataPartition(y = dataset$y, p = 0.8,list =FALSE)
  train.dataset <- dataset[training.samples.dataset,]
  test.dataset <- dataset[-training.samples.dataset,]
  train.control <- trainControl(method = "repeatedcv",
                                number = 5, repeats = 3)

  qda.model <- train(as.factor(y) ~., data = train.dataset,
                     method = "qda", trControl = train.control)
  predict.qda <- predict(qda.model, test.dataset)
  qda.conf.matrix <- confusionMatrix(table(test.dataset$y, predict.qda))
  qda.results[[d]] <-  ml_test(predict.qda, as.factor(test.dataset$y), output.as.table = TRUE)
  qda.accuracy[[d]] <-  ml_test(predict.qda, as.factor(test.dataset$y), output.as.table = FALSE)$accu
  qda.conf.matrix[[d]] <- confusionMatrix(table(test.dataset$y, predict.qda))
  qda.F1[[d]] <-  mean(ml_test(predict.qda, as.factor(test.dataset$y), output.as.table = FALSE)$F1)
}
```

E Appendix : Random forest R code

```
library(tidyverse)
library(caret)
library(MASS)
library(randomForest)
library("mltest")
set.seed(203)
rf.results <- list()
rf.accuracy <- list()
rf.F1 <- list()
for (d in 1 : 14) {
  location <- sprintf("/Users/maike/Desktop/Scriptie/Datasets_new/dataset%s.csv", d)
  dataset <- read.csv(location)
  training.samples.dataset <- createDataPartition(y = dataset$y, p = 0.8,list =FALSE)
  train.dataset <- dataset[training.samples.dataset,]
  test.dataset <- dataset[-training.samples.dataset,]
  train.control <- trainControl(method = "cv",
                                number = 5)
  start.time <- Sys.time()
  rf.model <- train(as.factor(y) ~ x1+x2+x3, data = train.dataset,
                    method = "rf", trControl = train.control)
  end.time <- Sys.time()
  time.taken <- end.time - start.time
  print(time.taken)
  predict.rf <- predict(rf.model, test.dataset)
  rf.conf.matrix <- confusionMatrix(table(test.dataset$y, predict.rf))
  rf.results[[d]] <- ml_test(predict.rf, as.factor(test.dataset$y), output.as.table = TRUE)
  rf.accuracy[[d]] <-  ml_test(predict.rf, as.factor(test.dataset$y), output.as.table = FALSE)$accura
  rf.F1[[d]] <-  mean(ml_test(predict.rf, as.factor(test.dataset$y), output.as.table = FALSE)$F1)
}
```

F Appendix : Support vector machine R code

```
library(tidyverse)
library(caret)
library(MASS)
set.seed(204)
svm.results <- list()
svm.accuracy <-list()
svm.F1 <- list()
for (d in 1 : 14) {
  location <- sprintf("/Users/maike/Desktop/Scriptie/Datasets_new/dataset%s.csv", d)
  dataset <- read.csv(location)
  training.samples.dataset <- createDataPartition(y = dataset$y, p = 0.8,list =FALSE)
  train.dataset <- dataset[training.samples.dataset,]
  test.dataset <- dataset[-training.samples.dataset,]
  train.control <- trainControl(method = "repeatedcv",
                                number = 10, repeats = 3)
  start.time <- Sys.time()
  svm.model <- train(as.factor(y) ~., data = train.dataset,
                     method = "svmRadial", trControl = train.control)
  end.time <- Sys.time()
  time.taken <- end.time - start.time
  print(time.taken)
  predict.svm <- predict(svm.model, test.dataset)
  svm.conf.matrix <- confusionMatrix(table(predict.svm,test.dataset$y ))
  svm.results[[d]] <-  ml_test(predict.svm, as.factor(test.dataset$y), output.as.table = TRUE)
  svm.accuracy[[d]] <-  ml_test(predict.svm, as.factor(test.dataset$y), output.as.table = FALSE)$accu
  svm.F1[[d]] <-  mean(ml_test(predict.svm, as.factor(test.dataset$y), output.as.table = FALSE)$F1)
}
```

G Appendix : Neural networks R code

```
library(ggplot2)
require(neuralnet)
require(nnet)
library(tidyverse)
library(caret)
library(MASS)
set.seed(205)
nn.results <- list()
nn.accuracy <- list()
nn.F1 <- list()
for (d in 1 : 14) {
  location <- sprintf("/Users/maike/Desktop/Scriptie/Datasets_new/dataset%s.csv", d)
  dataset <- read.csv(location)
  training.samples.dataset <- createDataPartition(y = dataset$y, p = 0.8,list =FALSE)
  train.dataset <- dataset[training.samples.dataset,]
  test.dataset <- dataset[-training.samples.dataset,]
  train.control <- trainControl(method = "repeatedcv",
                                number = 5, repeats = 3)

  nn.model <- train(as.factor(y) ~., data = train.dataset,
                    method = "nnet",  trControl = train.control, )
  predict.nn <- predict(nn.model, test.dataset)
  nn.results [[d]] <-  ml_test(predict.nn, as.factor(test.dataset$y), output.as.table = TRUE)
  nn.accuracy[[d]] <-  ml_test(predict.nn, as.factor(test.dataset$y), output.as.table = FALSE)$accura
  nn.F1[[d]] <-  mean(ml_test(predict.nn, as.factor(test.dataset$y), output.as.table = FALSE)$F1)
}
```