ALZHEIMER'S DISEASE PREDICTION BASED ON MULTI-OMICS DATA: A DEEP LEARNING APPROACH

by

ALEJANDRO HERNÁNDEZ DÍAZ
URN: 6524384

A dissertation submitted in partial fulfilment of the
requirements for the award of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

May 2021

Department of Computer Science
University of Surrey
Guildford GU2 7XH

Supervised by: Thomas Thorne

I declare that this dissertation is my own work and that the work of others is acknowledged and indicated by explicit references.

Alejandro Hernández Díaz
May 2021

Abstract

The application of deep learning techniques to Alzheimer's disease diagnosis has mainly been focused on the use of phenotypic data. This is due to the fact that the underlying mechanisms of the condition are still unidentified.

Phenotype-based Alzheimer's diagnoses struggle to provide an early detection of the disease in most cases. The main reason why they have not yet been replaced with other genotype-based techniques is the lack of genetic samples available for research.

However, recent studies have produced large scale genetic datasets that can aid in not only the investigation of the disease but also the creation of new accurate and efficient diagnostic tests.

This dissertation presents a solution to Alzheimer's disease diagnosis/prediction based on two different integrated genetic datasets.

A genetic algorithm based feature selection method was designed to successfully reduce the number of features and deal with the problem of high dimensionality present in the data of this nature.

Furthermore, a deep neural network architecture was implemented in order to accurately perform the classification task.

The results obtained by the architecture when trained on the preprocessed datasets outperform conventional machine learning algorithms, achieving an AUROC of 0.9006.

This demonstrates how applying genetic-trained artificial intelligence techniques to the field could improve the early detection of the disease and shine a light on its investigation.

Acknowledgements

I would like to take this opportunity to express my deep and sincere gratitude to my supervisor, Tom Thorne, for his outstanding guidance during the completion of this project.

Additionally, I would like to thank my family for their unmeasurable support. They have put a lot of work and effort into making sure I become the best version of myself. I would have never been in the place I am if it wasn't for them.

I must also extend my thanks to my English teacher in Bachillerato, Sergio Juan, for encouraging me to study abroad. I will always be in his debt.

Finally, I would like to thank all my friends for helping me through university and making these years the best years of my life.

Contents

1 Introduction **11**

1.1 Motivation . . . 11

1.2 Project Aims and Objectives . . . 12

1.3 Structure of the Report . . . 13

2 Literature Review **14**

2.1 Artificial Intelligence in Genomics . . . 14

2.2 Why Deep Learning? . . . 15

2.2.1 Artificial Neural Networks Explained . . . 16

2.2.1.1 The Artificial Neuron . . . 16

2.2.1.2 The role of the activation function . . . 17

2.2.1.3 The Layered Structure of a Feed-Forward Neural Network . . . 19

2.2.1.4 Assessing the Network: The Loss Function . . . 20

2.2.1.5 The Training Process: Backpropagation and Gradient Descent . 21

2.3 Genetic algorithms . . . 23

2.3.1 The Individual . . . 24

2.3.2 The Fitness Function . . . 26

2.3.3 The Selection Mechanism . . . 27

2.3.4 Genetic Operators: Crossover . . . 28

2.3.5 Genetic Operators: Mutation 29

2.4 Artificial Intelligence and Genomics in Alzheimer's prediction 30

2.4.1 Artificial Intelligence and Genomics in Alzheimer's Prediction: Multi-modal Approaches 30

3 Project Management 34

3.1 Project Management Methodologies 34

3.1.1 Agile Development 35

4 Statement of Ethics: Ethical and Legal Issues 38

4.1 Legal Issues 38

4.2 Ethical Issues 39

5 Methods 40

5.1 Data Acquisition and Preprocessing 40

5.2 Feature Selection 42

5.2.1 Genetic Algorithm for Feature Selection: Design explained 44

5.2.2 Genetic Algorithm for Feature Selection: Evaluating the Population . . . 46

5.2.3 Genetic Algorithm for Feature Selection: Execution 49

5.3 Model Design and Training 50

5.4 Model Testing 53

6 Results 54

6.1 Feature Selection: Results 54

6.2 Model Training and Testing: Results 55

7 Conclusions and future work 61

7.1 Conclusions 61

7.2 Future Work 62

List of Figures

2.1 An Artificial Neural Network based on the human brain. 16

2.2 The structure of an artificial neuron (Perceptron) 17

2.3 Linearly vs non-linearly separable data. 18

2.4 Some of the most popular Activation Functions. 19

2.5 Structure of a Feed-Forward Neural Network. 19

2.6 Visualization of how the error of a model (Z) changes with respect to its parameters' values (Y and X). 21

2.7 Structure of a simple genetic algorithm. 23

2.8 Individual with 2 chromosomes, encoded using 4 bits each. 25

2.9 Individual with 8 chromosomes, encoded using real values. 25

2.10 Fitness proportionate selection mechanism. 28

2.11 One point crossover. The parents (left) form two offspring (right). 29

2.12 Uniform crossover. The parents (left) form two offspring (right). 29

2.13 Individual before (left) and after (right) a mutation of its third gene. 30

2.14 Structure of the network used in the study 32

3.1 The main parts that comprise the Agile methodology. 36

5.1 The encoding of each individual as a feature vector. The individual n (left) and the dataset it represents (right). 44

5.2 Function that decodes the individual's chromosome and turns it into two datasets (Train/Test). 46

5.3 Code that handles the evaluation of the individuals. 47

5.4 Submit file example. 50

6.1 Plots of the best fitnesses across generations in each run. 55

6.2 Loss of the architecture proposed when trained using different datasets. 56

6.3 Performance of the selected architecture when tested. Each independent run is specified by its number of training epochs. 57

6.4 Performance of the selected architecture when tested (bar chart). Each independent run is specified by its number of training epochs. 58

6.5 Performance of the selected classifiers when trained and tested on all 4 datasets. . 59

6.6 Performance of the selected classifiers when trained and tested on all 4 datasets (bar chart). 59

List of Tables

2.1 Description of ADNI data used in the study. 31

2.2 Number of patients by modality and disease stage (CN controls, MCI mild cognitive disorder, and AD Alzheimer's disease). 31

5.1 Genetic profiles used in the creation of the multi-omics dataset. 40

5.2 Dimensions of the 8 subdatasets. 41

5.3 Dimensions of the merged subdatasets. 42

5.4 Dimensions of the final multi-omics datasets. 42

5.5 Characteristics of the genetic algorithm proposed. 45

5.6 Details of the training techniques used. 47

5.7 Structural details of Network 1. 48

5.8 Structural details of Network 2. 48

5.9 Structural details of Network 3. 49

5.10 Structural details of the network architecture proposed for classification/prediction. 51

5.11 The 4 network objects initialised in each run of the train and test process. 53

6.1 Feature selection: summary of the results achieved. 54

Abbreviations

AD	Alzheimer's disease
ANN	Artificial Neural Network
DL	Deep learning
FFNN	Feed-forward Neural Network
HDLSS	High-dimensional low sample size
GA	Genetic algorithm
FS	Feature selection
MSE	Mean squared error
BP	Back-propagation
DEG	Differentially expressed gene
DMP	Differentially methylated position
UC	Uniform crossover
CPU	Central processing unit
AUROC	Area under the receiver operating characteristic curve

Chapter 1

Introduction

1.1 Motivation

According to the NHS, Alzheimer's disease is the most common type of dementia in the UK. As reported by research conducted in 2019, there were over 850.000 UK citizens with dementia from which 50-75% fell into the category of AD (*Alzheimer's Society's View on Demography | Alzheimer's Society* n.d.).

As no clinically reliable genetic risk factor has yet been identified, the majority of AD diagnoses are performed using brain imaging techniques, such as MRI scans, and neuropsychological examinations. These approaches drastically reduce the possibility of an early diagnosis as they rely on the detection of clear manifestations of the disease, at which point, Alzheimer's would have already caused severe brain damage (*Earlier Diagnosis* n.d.).

Recent advances in the field of genomics have led to an increase of genetic data available. Public repositories such as the Gene Expression Omnibus (GEO) archives and distributes this kind of samples, allowing scientist to investigate and analyse the underlying mechanisms of these yet "unexplored" conditions.

The main drawback that comes with the usage of genetic datasets for disease diagnosis is their complexity. Determining the correlation between specific gene interactions and the presence of a condition would not only take great expertise but also time. This is why professionals are focusing their attention on machine learning based approaches in order to tackle this problem in a more efficient way.

Machine learning algorithms, such as deep neural networks, are obtaining state-of-the-art results when applied to data-intensive fields like the one previously addressed. This is mainly due to the ability of said algorithms to learn from the intricate patterns hidden among samples. Furthermore, deep models are able to automatically extract features from raw data in contrast to other machine learning techniques, which need hand-crafted features, making them an extremely powerful tool.

The application of these deep learning techniques to the diagnosis of Alzheimer's using genetic data can not only generate extremely accurate classifiers which will aid on the early detection of the disease, but also help experts to gain a better understanding of such a devastating condition.

1.2 Project Aims and Objectives

The overall aim of this project is to investigate the performance of deep neural networks on the task of predicting Alzheimer's disease with the use of two integrated genetic datasets.

The goals and objectives set for this project are the following:

- Research and understand the topic of deep learning and its applications to the field of genomics.
- Gain experience with the implementation of machine learning approaches.
- Research and understand different feature selection methods.
- Assess the performance of the feature selection approach chosen.
- Research different regularization techniques such as dropout and early stopping.
- Gain experience with the usage PyTorch and other machine learning frameworks.
- Design and Implement a deep neural network architecture for the classification of AD and normal samples.
- Gain a better understanding on different metrics used to asses the performance of deep learning architectures.
- Compare the results achieved by the architecture proposed against other classifiers to prove its efficacy.

1.3 Structure of the Report

- **Section 1: Introduction ->** This section explains the motivation behind this project as well as the aims and objectives of its completion.
- **Section 2: Literature Review ->** This section reviews the technologies and techniques used in this project. Alongside it introduces the existing artificial intelligence methods implemented in the field of genomics and other related work in the field.
- **Section 3: Project Management ->** This section introduces the management guidelines followed during the completion of this project.
- **Section 4: Statement of Ethics: Ethical and Legal Issues ->** This section presents the ethical and legal factors which were considered during the completion of this project.
- **Section 5: Methods ->** This section provides an in depth explanation of the methodology used in this dissertation. The design choices presented are also well documented in order for the reader to achieve a better understanding of the reasoning behind the solution proposed.
- **Section 6: Results ->** This section provides a logical interpretation of the outcome of the experiments conducted in the methods section.
- **Section 7: Conclusions and Future Work ->** This last section summarizes the research conducted in the project. Moreover, it also includes some ideas that could extend the work done.

Chapter 2

Literature Review

This chapter will introduce the existing artificial intelligence methods implemented in the field of genomics. Alongside, will review the technologies, algorithms and techniques used in the project in order to give the reader a basic understanding of their theoretical concepts before explaining how they were implemented. The last subpart of this literature review will be focused on the current applications of the previously mentioned approaches to the prediction of Alzheimer's.

2.1 Artificial Intelligence in Genomics

As biomedical research has grown to be more data-intensive, thanks to the latest technological advances in the field, the amount of work that needs to be done by professionals to analyse, interpret and classify this data has also increased (Álvarez-Machancoses, DeAndrés Galiana, Cernea, Fernández de la Viña & Fernández-Martínez 2020). This growth has likewise impacted the field of genomics as the high dimensionality of the genetic data makes it extremely difficult to work with.

This abrupt increment of the number of data collected in the past years has come with new developments in the technologies used to analyse it but, after applying them to the problem of "Big data" in genomics, two major problems arose:

- A vast majority of the medical data collected falls into the HDLSS (High Dimension Low Sample Size) category, which means that there only exists a limited amount of samples compared to the number of features (eg. genes).

- The data has an inherent level of noise that may falsify predictions made by the algorithms (Álvarez-Machancoses et al. 2020).

The problems stated above are still to be solved however, they are more related to the nature of the samples collected than to the algorithms used to process them.

AI experts started experimenting with new approaches as another problem was discovered: The machine learning (ML) algorithms deployed for clinical use required the data to be highly preprocessed in order to extract features that could then be used for training (Dias & Torkamani 2019). Manually extracting features from datasets of that size and complexity is often disregarded due to the amount of time and effort that would take to complete, making these algorithms not worth implementing.

The researchers' attention started to drift away from these traditional ML methods into more promising approaches such as deep learning architectures, as they have proven to achieve higher accuracies than any competitor when dealing with vast amounts of raw data (*Deep Learning Models in Genomics; Are We There yet? | Elsevier Enhanced Reader* n.d.).

2.2 Why Deep Learning?

Deep Learning is a subset of Machine learning. ML represents the set of algorithms or techniques that, if trained on adequate data, would allow computers to mimic human abilities such as prediction or classification, among others (Oppermann 2020). Deep learning, on the other hand, is the subfield of machine learning concerned with algorithms inspired by the structure of the human brain (Brownlee 2019).

There are two main reasons behind the increase of Deep Learning's popularity in genomics:

- DL architectures are perfect for tackling "Big Data" problems due to their impressive scalability. As the number of training samples increases, so does the accuracy of deep models, outperforming any other ML approaches.

- Unlike the algorithms previously used, DL models have the ability to perform automatic feature extraction from raw data, also called feature learning. This characteristic makes a tremendous difference in terms of performance, as the preprocessing steps previously cited in the section above are no longer needed when using this type of models (Brownlee 2019).

In order to achieve these necessary capabilities, deep learning uses a multi-layered structure of algorithms called artificial neural networks (Brownlee 2019)

The next subsection will go through the basics of neural networks, so the reader can gain a deeper insight into how this type of algorithm works before introducing its applications and results in the field of genomics.

2.2.1 Artificial Neural Networks Explained

Artificial neural networks (ANNs) are computational models that try to mimic how the brain processes information on a much smaller scale (*Artificial Neural Network* 2021).

ANNs are comprised of a set of artificial neurons interconnected and structured into layers. Each neuron has a set of inputs, that will then use to compute its level of activation (output).

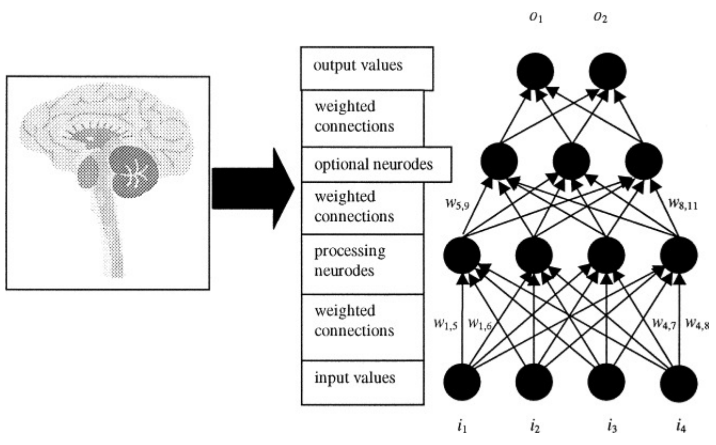


Figure 2.1: An Artificial Neural Network based on the human brain.
Source: (Walczak & Cerpa n.d.).

2.2.1.1 The Artificial Neuron

The smallest units of computation inside an ANN are the artificial neurons (Perceptrons) (ujjwalkarn 2016), these are represented as circles or nodes as it can be seen in figure 2.1. Each neuron receives a set of inputs $\boldsymbol{x}$ that are associated with a unique weight $\boldsymbol{w}$, in order to specify the input's importance compared to the others. The neuron also has one last parameter called bias $\boldsymbol{b}$, which has a weight $\boldsymbol{w} = \boldsymbol{1}$, whose importance will be discussed later in this subsection. Both the weights and biases of each neuron in a network do not have a fixed value, they change during the training process according to the output values produced by the final layer (*Feed Forward Neural Network* 2019), as presented in section 5.3

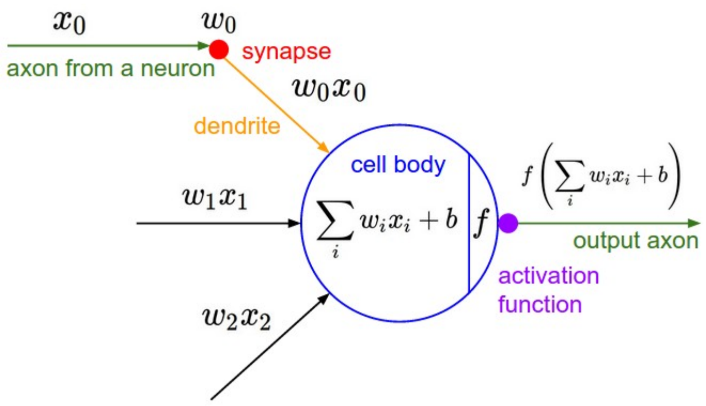


Figure 2.2: The structure of an artificial neuron (Perceptron)
Source: (*CS231n Convolutional Neural Networks for Visual Recognition* n.d.).

Figure 2.2 represents the internal structure and parts of an artificial neuron. In order to process the input signals, the neuron does as follows:

It first performs a weighted sum of the inputs. The bias term is added at the end, as shown in equation (2.1). This term plays the role of the ***y-intercept*** in a linear equation. It is an additional parameter that improves the neuron's flexibility, allowing it to fit best the training data.

$$u = \sum_{i=1}^{N} w_i x_i + b \tag{2.1}$$

When this first computation is complete, instead of sending the result forward in the network, the neuron feeds it into an activation function f, achieving the value that will be sent out, as shown in equation (2.2).

$$O = f(u) \tag{2.2}$$

2.2.1.2 The Role of the Activation Function

An activation function (also known as transfer function) is a function added inside each neuron in order to help the network learn more complex patterns in the data (Jain 2019). In a biological neuron, the cell first receives information from other neurons, then processes these signals and subsequently it decides what to send out to the other cells. This action of "deciding what to output" is carried out by the activation function in an artificial neuron (Jain 2019).

The resemblance to how a biological neuron works is not the only property achieved when implementing this type of functions.

- In an ANN comprised of neurons without activation functions, the output values increase in magnitude as we traverse deeper into the network. The reason behind this is that, according to what was stated at the beginning of the section, each cell would output a weighted sum of its inputs to the next layer of neurons, and so would do the cells in subsequent layers. Some activation functions have a limited range, e.g. the sigmoid function only returns values within the range $(0, 1)$. This property ensures that the outputs' magnitude stays consistent across the whole network, no matter how deep it is.
- The last and most important property of activation functions is their non-lineality. If we recall again what was introduced at the beginning of the subsection, the weighted sum that occurs inside each cell has a degree of 1 hence the neuron would only be able to "understand" linearly separable data (2.3). In a real-life scenario, the data used is often non-linearly separable (2.3) so, the non-linear activation function allows the neurons to recognise the more intricate patterns that separate it. (Jain 2019).

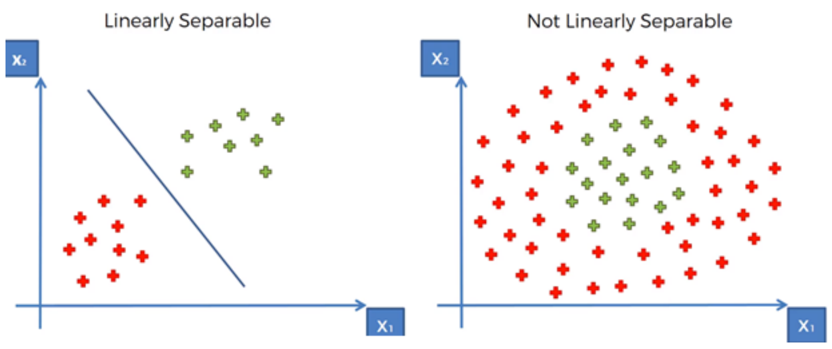


Figure 2.3: Linearly vs non-linearly separable data.
Source: (Girgin 2019).

There are many different activation functions and each one has different properties that may or may not suit the problem being tackled so, deciding which function to use is not always straight-forward. Figure 2.4 displays some of the most popular functions used in ANNs.

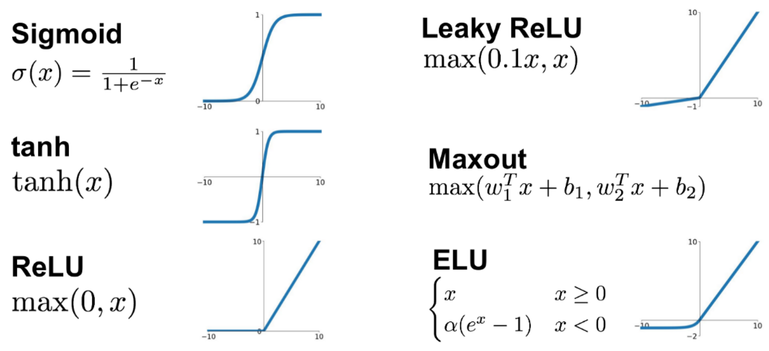


Figure 2.4: Some of the most popular Activation Functions.

Source: (Jadon 2020).

2.2.1.3 The Layered Structure of a Feed-Forward Neural Network

One single neuron is not able to learn the intricate patterns that underlie in nowadays data, that is why more complex models are needed. Artificial Neural Networks stack these neurons in interconnected layers in order to extract more sophisticated features from the samples.

The most basic and commonly used ANN type is the Feed-forward neural network. As shown in figure 2.5, each neuron in a FFNN is located in an interconnected layer.

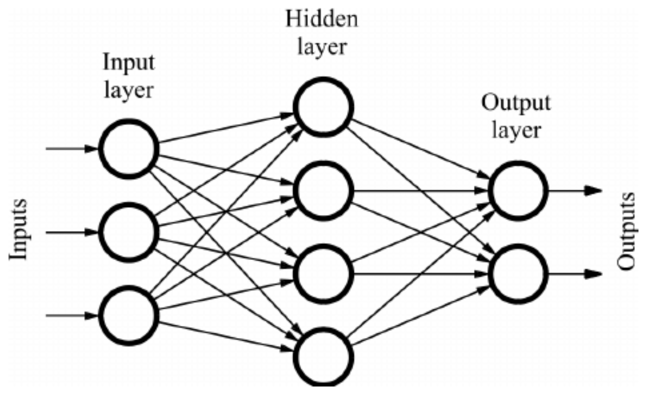


Figure 2.5: Structure of a Feed-Forward Neural Network.

Source: (Quiza & Davim 2011).

There are 3 different types of layers in a Feed-Forward Neural Network:

- **Input Layer:** This is always, as seen in figure 2.5, the first layer of the network. Here, the neurons do not perform any type of operation on the data but only receive it from the exterior in order to send it into the network. Unlike in other layers, the input connections to the neurons of the input layer do not have a weight value associated

- **Hidden Layers:** The hidden layers are the stacks of neurons which are between the input layer and the output layer. In a FFNN there can be as many hidden layer as desired when designing it. The nodes in these layers receive the output from the ones in the previous layer as input, and subsequently send their own output to the cells located in the next one. Each connection coming in and out of the neurons in the hidden layers has a unique weight as discussed in section 2.2.1.1.

- **Output Layer:** The output nodes are collectively referred to as the "Output Layer". They too receive the weighted output of the previous layer as input but, after performing their own computation on the signals, they transfer their output to the outside world (ujjwalkarn 2016).

2.2.1.4 Assessing the Network: The Loss Function

To this point, section 2.2.1 has covered the main components of an ANN and explained how these parts come together to form one. The next step in the sequence is to train the designed model on some data; however, in order to improve the network's performance, a way of assessing it is needed first. Here is where the loss function comes into play.

The error function is a method of evaluating how well does an algorithm model the given data, by analysing how much do the values predicted by the network differ from the desired output. The output of the error function increases as the predicted value gets further away from the desired one (Parmar 2018).

One of the most common error functions is MSE (Mean Squared Error) 2.3:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Y_i^{'})^2 \tag{2.3}$$

To measure the loss using MSE, a testing dataset of n samples is fed into the network in order to achieve a set of predictions $Y^{'}$. The function will then compare each value predicted by the model Y_i' with its corresponding desired output Y_i, adding all the differences together and dividing the result by the number of samples n to get the mean error across the dataset.

2.2.1.5 The Training Process: Backpropagation and Gradient Descent

The goal of the training process of a model is to minimize the error function that is used to assess it, improving the accuracy of its predictions by doing so. In this process, two types of parameters in the network are fine tuned according to how their change affect the loss of the model: *The weights and biases.*

Figure 2.6 illustrates the impact of the parameters' values in the error of a model.

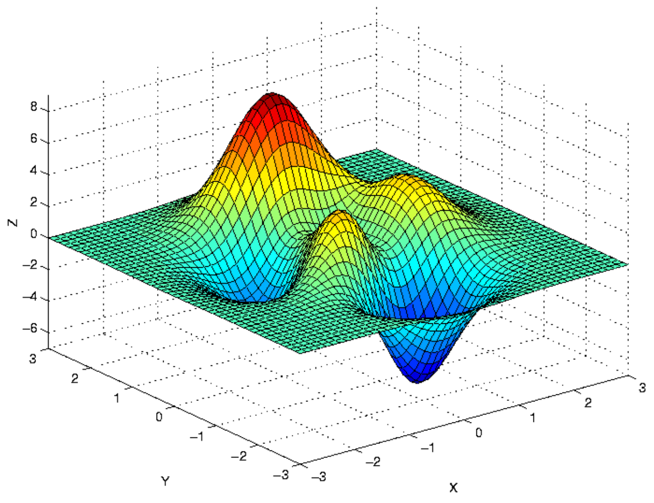


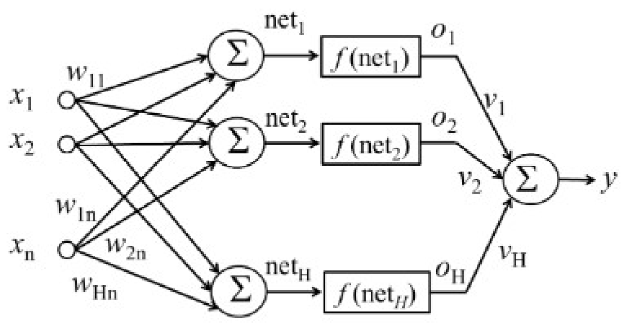
Figure 2.6: Visualization of how the error of a model (Z) changes with respect to its parameters' values (Y and X).

Source: (Deore 2021).

The most widely used algorithm to minimise the error of a model and optimise its parameters is called **Gradient Descent**. As its name suggests, the idea behind this algorithms is to start at a random point of the error function's landscape (figure 2.6) and start traversing it down until the global minimum is reached. This is done by calculating the derivative of the error function with respect to each one of the parameters at each step. Computing the slope of the function with the help of the derivative will tell the algorithm what change does each parameter need i.e. which way to move in the landscape.

These derivatives are calculated using a supervised learning method called **Backpropagation**.

The following example will clarify how both algorithms work together to optimise the parameters of a network:



We are trying to train the model whose architecture is shown above where:

- net_j is the weighted sum of the inputs of neuron j

$$net_j = \sum_{i=1}^{n} w_{ji} x_i \tag{2.4}$$

- O_j is the output of neuron's j after going through the activation function f

$$o_j = f(net_j) \tag{2.5}$$

- y is the output of the network

$$y_i = \sum_{j=1}^{H} v_j o_j \tag{2.6}$$

- The error function used is MSE as shown in equation 2.3

Calculating the derivatives for the weights of the output layer:

The change of the parameter v_j is equal to the derivative of the error function with respect to v_j , then multiplied by a learning rate ξ. The learning rate ξ is a hyperparameter that needs to be specified beforehand and dictates how much should v_j change.

$$\Delta v_j = -\xi \frac{\delta E}{\delta v_j} \tag{2.7}$$

As shown in equation 2.3, the MSE error function is not directly dependant on v_j but on y_i. However, y_i is directly dependant on v_j so, by applying the chain rule the formula changes to what is seen bellow.

$$\Delta v_j = -\xi \frac{\delta E}{\delta y_i} \frac{\delta y_i}{\delta v_j} \tag{2.8}$$

Lastly, the derivatives are calculated and multiplied, resulting in equation 2.9.

$$\Delta v_j = \xi \sum_{i=1}^{n} (y_i - y_i^{'}) o_j \tag{2.9}$$

2.3 Genetic algorithms

This section will review the theory behind genetic algorithms as, despite the fact that they are not as popular in the field of genomics as deep learning is, they play a crucial role in the completion of this project.

A genetic algorithm is a metaheuristic inspired by Darwin's theory of evolution (*Genetic Algorithm* 2021) i.e. a procedure that is designed to find an optimal or sufficiently good solution to an optimization problem by applying concepts that mimic the evolutionary processes in nature such as mutation, crossover and selection. GAs are metaheuristics as they are not problem-dependent, achieving great results in any search space due to their general design.

GAs are population-based, which means that instead of working with one solution in the search space (space that contains all possible solutions to the problem) at a time, they create a population of candidate solutions (Individuals) which is then manipulated throughout generations in order to achieve the best set of individuals possible. The main structure of a GA is shown in figure 2.7

```
Simple genetic algorithm() {
   initialize first population
   evaluate the population
   while(termination criteria not reached) {
      select individuals for next population
      perform crossover and mutation
      evaluate the population
   }
}
```

Figure 2.7: Structure of a simple genetic algorithm.

As seen in the pseudo-code (figure 2.7), a pool of candidate solutions is selected from the search space in a random manner. This first population is then evaluated through a fitness function in order to discern how good each solution is. Once that procedure terminates, a subset of the population is selected to serve as the parents of the next population. These parents go through a process of matting, generating new offspring, which are subsequently mutated. This cycle continues until a certain termination criteria is achieved (Jebari & Madiafi n.d.)

According to the explanation above, this type of algorithm comprises the following components:

- a population of individuals
- a fitness function
- a selection mechanism
- genetic operators (crossover and mutation)

The following subsections will introduce each one of the concepts stated above. Alongside, explanatory examples will be provided to support the theoretical definitions.

2.3.1 The Individual

Nature is a competitive environment. History shows that the individuals which are best suited to compete for resources survive.

The conditions in nature are in constant change so, these individuals are also required to adapt over time in order to acquire the characteristics necessary to endure those conditions (Srinivas & Patnaik 1994). The physical characteristics that each individual possesses directly determine its survival capabilities and these features in turn are determined by the individual's genetic content.

In a genetic algorithm, each individual is a specific solution to the optimization problem being tackled. For example, if a genetic algorithm is used to minimize a function with two parameters, x and y, each member of the population would represent a pair of values, one for x and one for y respectively. Each one of those two values are the individual's features.

These "artificial" individuals posses a number of chromosomes which are the encodings of their features. Each chromosome is in turn composed by genes.

The method used to encode the individual's features into their respective chromosomes is problem dependent and must be chosen carefully. The two main examples of encoding mechanisms are:

- **Binary:** As seen in the example above and in a large number of optimization problems, the individual's features tend to have an integer representation. When using this type of encoding mechanism, each feature is mapped to an integer value defined in a specific range and then it is encoded using a fixed number of binary bits. The same range of integers and number of bits is used to encode each feature. When all chromosomes of an individual are encoded, their binary representation are concatenated to form a binary string like seen in the example bellow.

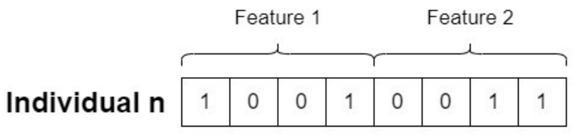


Figure 2.8: Individual with 2 chromosomes, encoded using 4 bits each.

- **Real:** When the individual's features are values complex enough such that binary encoding does not suffice, real encoding is used. No further preprocessing is applied to the features, instead they are directly input to their respective chromosomes and concatenated like seen in the overly simplified example bellow.

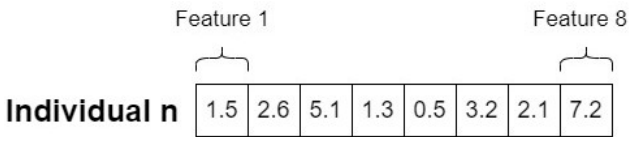


Figure 2.9: Individual with 8 chromosomes, encoded using real values.

Both representations are equally valid and choosing between them depends on the problem that the GA needs to optimize. Each encoding possesses its own set of genetic operators (mutation and crossover) due to their differences in representation. From now on, the following subsections will focus on binary-coded genetic algorithms so only the concepts included in this subset of GAs will be covered.

2.3.2 The Fitness Function

As it was introduced in the beginning of the subsection above, Darwin's theory of evolution describes how each individual in nature has an unique fitness i.e ability to compete for resources and to adapt to the changing environment. Amongst animals, this ability to survive is determined by their respective physical characteristics. One clear example would be a pride of lions. Each lion has a set of quantifiable features such as weight, speed, agility or power, which are directly translated into its level of fitness. The faster and stronger a lion is, the fitter as it will be able to not only catch better prey and defend themselves in case of an attack, but also endure tougher natural conditions.

This notion is incorporated into genetic algorithms via the fitness function. This is the function to be optimized by the GA and it works as a way of evaluating how good a specific solution is according to its features. It is problem-dependent, which means that each every different problem to solve has a different fitness function.

In order to gain a better understanding of how the fitness function works, consider an optimization problem that consists on **maximizing** the function,

$$f(x, y) = 4x + y \tag{2.10}$$

and the set of candidate solutions bellow.

Individual	Feature 1 (x)	Feature 2 (y)
Individual 1	1	3
Individual 2	-2	4
Individual 3	-1	-5

In this specific problem, the genetic algorithm is entitled with the task of finding an individual (pair of x and y values) which maximizes the function 2.10. This implies that, an individual that achieves a high value when being inserted into the function 2.10 (therefore maximizing it), is fitter than one which does not. *(It is important to note that high values are not always a sign of high fitness, as genetic algorithms can also be used to solve* ***minimization*** *problems, where* *the goal is the complete opposite)*

The table bellow shows the fitness of each one of the three candidate solutions.

Individual	Feature 1 (x)	Feature 2 (y)	Fitness
Individual 1	1	3	7
Individual 2	-2	4	-4
Individual 3	-1	-5	-9

After evaluating all 3 specimens, individual 1 shows the highest fitness of all three candidate solutions.

2.3.3 The Selection Mechanism

The fitness of an individual in nature not only determines its capabilities of survival, but also directly influences its reproductive probability. The fact that the stronger and more advanced individuals of a population have greater chances of matting, is one of the key aspects that make evolution possible (Srinivas & Patnaik 1994).

Selection mechanisms in genetic algorithms are a way of ensuring that the better solutions of a population survive, in order to pass on their genetic material (chromosomes) to the following generation.

There are many ways of selecting the parent population for the next generation, and choosing between them also depends on the problem. Some examples are:

- **Fitness proportionate:** The fitness proportionate (also known as roulette wheel) is a probabilistic selection mechanism, where each individual in the population receives a probability of being selected, given by the following formula:

  $$p(I_i) = \frac{f(I_i)}{\sum_{j=1}^{P} f(I_j)} \tag{2.11}$$

  Where $f(I_i)$ is the fitness of the individual i and P is the size of the population (*Fitness Proportionate Selection* 2020).

  After the probabilities are calculated, a "roulette wheel" is rotated n times, equal to the number of individuals to select. This is done as shown in figure 2.10.

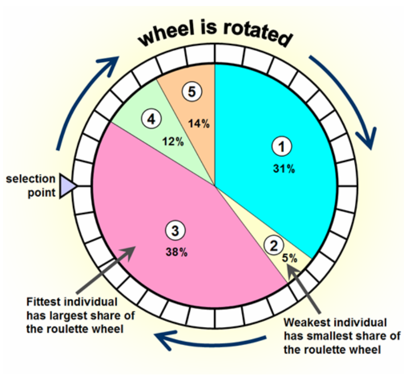

wheel is rotated
selection point
1
31%
2
5%
3
38%
4
12%
5
14%
Fittest individual has largest share of the roulette wheel
Weakest individual has smallest share of the roulette wheel

Figure 2.10: Fitness proportionate selection mechanism.
Source: (*GA Roulette Wheel Selection* n.d.).

- **Tournament selection:** This selection method consists on randomly sampling n batches of k individuals from the population, where n is the number of individuals to select. Each batch works as a small tournament among individuals, where the fittest of them gets selected. After all the "tournaments" conclude, n individuals are selected as the parents for the next population (Jebari & Madiafi n.d.).

2.3.4 Genetic Operators: Crossover

The fact that, at the moment of conception, the offspring inherit a mix of genetic traits from both its parents is one of the main keys to evolution. Individuals do not directly obtain the physical features of their progenitors, but the genes that are responsible for the expression of those characteristics. This is a probabilistic process as in nature, the progenies do not choose which genes they inherit.

This process of genetic material recombination is modelled in GAs via the crossover operation, which is a crucial part of the algorithm. This operation is not applied to the individuals directly but to their encoded chromosomes i.e binary strings as discussed in section 2.3.1, just as animals inherit genes and not physical features.

During the matting process, random pairs of individuals are sampled from the selected parent population. Not all selected pairs are crossed, instead, the algorithm only invokes the operation if a randomly generated number in the range from 1 to 0 is greater than p_c, the crossover probability (Srinivas & Patnaik 1994).

There are many instances of crossover operations, some examples are:

- **One point crossover:** The algorithm randomly chooses a number between 0 and $l - 1$ where l is the length of the binary strings. It then performs a cut on that crossover point and exchanges the portions of string, forming two new offspring as seen in the figure bellow.

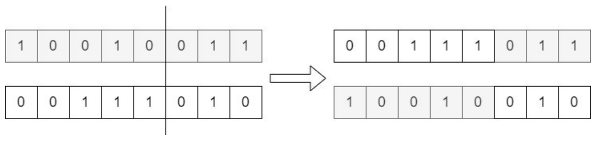


Figure 2.11: One point crossover. The parents (left) form two offspring (right).

- **Uniform crossover:** In uniform crossover, the algorithm iterates through the progenitors' chromosomes, gene by gene. It then forms offspring by choosing, for each gene, the bit from either parent with equal probability (*Crossover (Genetic Algorithm)* 2021). This is illustrated in the figure bellow.

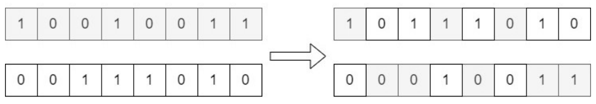


Figure 2.12: Uniform crossover. The parents (left) form two offspring (right).

2.3.5 Genetic Operators: Mutation

A mutation is a change in the DNA sequence, either due to errors during the DNA copying process in cell division or many environmental factors such as exposure to chemicals or UV light. Mutation plays a crucial role in the evolutive process as some of them occur in the reproductive cells and can be passed on to offspring (*Mutation* n.d.).

In GAs, mutation is treated as a secondary operation behind crossover. It works as a way of not only introducing some variety to the population but also restoring lost genetic material (Srinivas & Patnaik 1994) .

The mutation method used in binary-coded genetic algorithms is called **Bit flip**. As it can be deduced by its name, if an individual is to be mutated, the algorithm will traverse through its binary string and, for each gene it will generate a random number between 0 and 1. If this number is greater than p_m (the mutation probability), that specific bit (gene) will be flipped from 0 to 1 or vice-versa. The figure bellow shows an example of this mechanism.

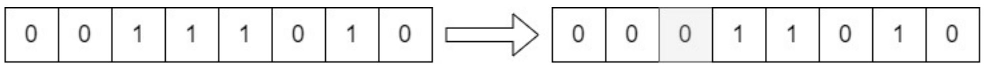


Figure 2.13: Individual before (left) and after (right) a mutation of its third gene.

As it was stated above, mutation allows the possibility of restoring lost genetic material. Consider an example where all the individuals of the population have converged to have a 0 in the 3rd position of their binary strings (chromosomes), but the optimal solution contains a 1 in that specific position. The crossover operator previously discussed in section 2.3.4 would not be able to bring that 1 back as none of the parent strings would have it but a mutation of that bit would cause it to flip, restoring that previously lost gene.

2.4 Artificial Intelligence and Genomics in Alzheimer's prediction

This section will cover the current applications of the previously introduced concepts to the field of Alzheimer's disease prediction.

2.4.1 Artificial Intelligence and Genomics in Alzheimer's Prediction: Multimodal Approaches

When applying deep learning techniques to Alzheimer's classification and prediction, the most common approach is to utilize phenotypic data such as MRI (Magnetic resonance imaging) images or clinical symptoms. The main reason behind this is that the underlying processes that can cause this disease are not yet completely understood. Alzheimer's is known to have a genetic component but scientists are still a long way from learning what exact genetic mutations can determine its appearance.

However, researchers have been working to integrate different types of data modalities to the prediction of AD.

One key example is the paper published in 2021 (Venugopalan, Tong, Hassanzadeh & Wang

2021), where scientist implemented multimodal deep learning models for the detection of Alzheimer's achieving state-of-the-art results.

The models were trained and tested on data from the Alzheimer's Disease Neuroimaging Initiative (ADNI) database. This dataset contains imaging, clinical and genetic data for over 2220 patients achieved across 4 different studies. The authors focused on the data of 3 of those 4 studies as the last one was still ongoing. The details of the data used are summarized in the tables bellow.

Type of data	**Example Features**
Clinical Data	Demographics, neurological exams, cognitive assessments, bio-markers, medication...
Imaging	Cross-sectional MRI data
Genetic	Whole genome sequencing data

Table 2.1: Description of ADNI data used in the study.

Type of data	**CN**	**MCI**	**AD**
Clinical Data	598	699	707
Imaging	132	104	266
Genetic	245	338	226

Table 2.2: Number of patients by modality and disease stage (CN controls, MCI mild cognitive disorder, and AD Alzheimer's disease).

This data was used in the following manner:

- The 3D neuroimaging data was fed into a 3D convolutional neural network in order to obtain intermediate imaging features.
- The Clinical data data was fed into a sparse denoising AutoEncoder in order to obtain intermediate EHR features.
- The genetic data data was fed into another sparse denoising AutoEncoder in order to obtain intermediate SNP features.
- These three types of intermediate features were passed into a fully connected layer for classification.

This process and the network structure can be seen in the figure below.

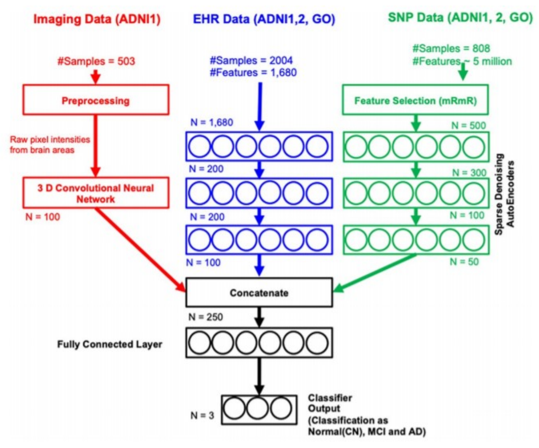


Figure 2.14: Structure of the network used in the study .

Source: (Venugopalan et al. 2021).

After using random forests as an alternative to the classification layer, the scientist confirmed that deep multi modal models outperform not only shallow models like KNN or SVMs, but also deep implementations that only make use of one data modality. The internal cross validation accuracy achieved by this architecture was 80%, way ahead of shallower models.

This study also showed the great importance of genetic data in the prediction of AD as it was one of the most important factors that determined their outstanding achievements.

Chapter 3

Project Management

3.1 Project Management Methodologies

Project management methodologies are sets of guiding principles and processes applied to managing a project. They dictate the different phases in which the work packages are distributed during the completion of a certain project.

There are many project management methodologies, each of them with their own pros and cons, and choosing the right one can be a challenging task.

One of the most used examples of methodology is the so called Waterfall. Said methodology was first considered as the one to be applied during the completion of this dissertation. This top-down approach breaks down the project in the following steps/stages:

- **Requirements Analysis:** In this first stage of the project, the requirements, objectives and goals are set. Having all the requirements specified at the beginning allows for the project's progress to be more easily controlled. In this phase, a feasibility analysis is also conducted in order to ensure that the plan is viable and can be developed.
- **System Design:** After the requirements are clearly defined, the next step in this methodology is to design a system that fulfils those objectives. In this phase, many possible solutions are considered.
- **Implementation:** As its own name suggests, during this stage all the theoretical concepts chosen for the solution of the problem presented are applied.

- **Verification:** During the verification phase of the project, the completed product is tested in order to ensure that it complies with the specification and requirements determined in the first stage.
- **Maintenance:** This is the last step in the waterfall methodology. It mainly consist on either adding more functionality to the product or solving errors that may appear during the verification phase.

After a more extensive research on different project management methodologies, the following problem arose:

- The different phases in which the waterfall model distributes the workload of the project follow a strict linear order. This may not be the best way of structuring a research type project as, for example, the system design is in constant change due to the experimental nature of this dissertation.

In order to solve this problem, a more flexible methodology was elected.

3.1.1 Agile Development

After considering different project management methodologies, the agile model was chosen to be the one implemented during the completion of this dissertation.

The Agile methodology is one of the most simple but effective ways to manage a research project. It focuses on the importance and inevitability of change which makes it the perfect choice when a more flexible model is desired. The main structure of said methodology is shown in the figure bellow.

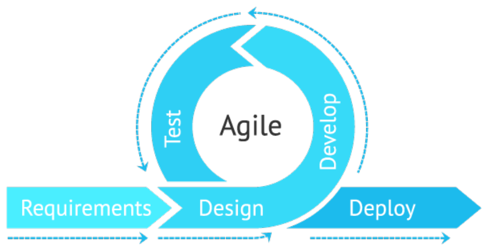


Figure 3.1: The main parts that comprise the Agile methodology. Source: (*Agile Advantages For Software Development | DevCom* 2020-02-26GMT+000010:54:21+00:00).

As it can be seen in the figure above, the Agile model shares similar phases to the ones described by the Waterfall methodology. The main difference is that the selected approach allows for the requirements and design of the final product to be revisited according to the results obtained by it at any point of the project life-cycle.

The Agile methodology was adapted and applied to this project in the following manner:

- The first step was to conduct an extensive research on the problem presented. The main goal of this phase was to not only gain a better understanding of the theoretical concepts that would be then applied in a later stage but also investigate the already published solutions to obtain ideas. The best part of using an Agile methodology was that a smaller-scale research could also continue to be conducted when needed in order to extend the work being done.

- After the main ideas and requirements were clear, the next step was to design, implement and test each component of the solution proposed. This was done in a cyclic manner. Each component would be first designed and tested. The results obtained during the testing phase were studied and design changes were made accordingly before being tested again. For example, the first design of the FS method chosen was changed many times as the results of the experiments dictated. During this phase, the communication between me and my supervisor was key as he would oversee and provide feedback on each of the changes.

- When each component of the solution was finished, its implementation process, results and subsequent changes were properly documented in this report.

The previously described cycle allowed for a better experimentation as its flexible structure made the implementation and documentation of changes a very easy task.

Chapter 4

Statement of Ethics: Ethical and Legal Issues

During the completion of this project, both legal and ethical factors have been considered in order to comply with the UK legislation. This chapter will justify how this dissertation fits into the code of ethical and legal practice of the university.

4.1 Legal Issues

All dissertations produced in the University of Surrey are expected to comply with the current UK legislation. In many disciplines, but even more in computer science, some projects may rise some legal concerns in terms of data protection and use and computer misuse. These risk factors were taken into account during the research conducted as part of this project. Some examples are:

The project presented makes use of two publicly available datasets. They can be found in the following GitHub repository (https://github.com/ChihyunPark/DNN_for_ADprediction). These same datasets are an adaptation of three genetic databanks also publicly available in the Gene Expression Omnibus (GEO) database. These databanks contain genetic profiles extracted from brain samples of deceased humans in the Harvard Brain tissue resource centre. They were obtained legally under US law and, as previously stated, made public. The public status of the data gives me, the conductor of this research, the ability to use it lawfully under the Data Protection Act (1998).

No human participants were required during the completion of this work.

The solution presented in this dissertation does not require for the collection of any other data. Neither does it need to store any personal information.

This project does not constitute an offence to the Computer Misuse Act as no unauthorised access to computer material with either intent to impair its operation or commit further offences has been carried out.

4.2 Ethical Issues

This project also complies with the university code of ethics. During the research phase of this dissertation, all the ethical issues that tend to appear in projects of this type were taken into account. Some examples may be:

As no human participation has been required during the completion of this work, there is no need for informed consent when using any of the data used. This is because, as stated in the legal section, this data is publicly available and has been obtained in a legitimate manner. This said data also takes care of keeping the confidentiality of the data as all patients studied to obtain it are kept undisclosed.

This work is also a contribution to society and human well-being as it looks to apply machine learning concepts to the field of medicine. This dissertation promotes the use of said technologies as a way to investigate yet "unknown" diseases in order to push humanity forward.

Finally, this dissertation does not mean to do any harm. The solution proposed is not intended to be used clinically or by any individual party. This project was carried out only for research purposes and none of the architectures produced are medically approved in order to be trusted as Alzheimer's predictors.

To make sure that this project complies with the ethical guidelines of the university, the SAGE form was completed. Its outcome determined that no further action was required.

Chapter 5

Methods

This chapter will provide a comprehensive overview of the system proposed. Furthermore, it will also include detailed explanations of each step followed in its design and implementation alongside the reasoning behind them. The presented approach generally comprises of the following parts: Data acquisition and preprocessing, feature selection, model design and training and model testing.

5.1 Data Acquisition and Preprocessing

One of the key factors that drastically affect the performance of any deep learning algorithm is the data used to train it. In the proposed approach, two different genetic databanks are integrated to produce the multi-omics dataset used during the training and testing process of the model: gene expression and DNA methylation profiles from the prefrontal cortex of AD-affected and normal deceased brains (Park, Ha & Park 2020).

Dataset	Number of normal samples	Number of AD samples	Number of features
Gene expression	257	439	200
DNA methylation	68	74	500

Table 5.1: Genetic profiles used in the creation of the multi-omics dataset.

A major characteristic of the used data is that the gene expression and DNA methylation profiles were measured from different brain samples. However, both datasets were generated from the tissue extracted from the same region of the brain of late-onset AD affected and normal patients (Park et al. 2020). Therefore, if the samples in the gene expression dataset were to be measured to extract their DNA methylation profiles, similar readings to the ones in the DNA methylation databank would be achieved.

Having the previous information in mind, it is justifiable to integrate both datasets by obtaining combinations of AD and normal samples respectively. The process of creating the multi-omics datasets works as follows:

- The gene expression and DNA methylation datasets are first separated into two subdatasets each, one containing the samples corresponding to AD positive individuals and vice versa.
- Before merging these new datasets, they need to be divided again into training and test samples, ending up with 8 sub-datasets in total. This is done because when combining the samples, a high number of duplicate values arise. This process ensures that, even though the training and test datasets will have their own duplicated values, they will not share them so the models can be tested on completely unseen data. The eight subdatasets are as shown bellow.

Dataset	Number of samples
Gene expression AD positive (Train)	307
Gene expression AD positive (Test)	132
Gene expression AD negative (Train)	192
Gene expression AD negative (Test)	65
DNA methylation AD positive (Train)	51
DNA methylation AD positive (Test)	23
DNA methylation AD negative (Train)	47
DNA methylation AD negative (Test)	21

Table 5.2: Dimensions of the 8 subdatasets.

- Now that the samples are properly divided, the next step is merging them e.g the gene expression AD positive (Train) subdataset is merged with the DNA methylation AD positive (Train) one by achieving all possible combinations of their samples. The achieved datasets can be seen in the table bellow

Dataset	Number of samples
GE + DNAm AD positive (Train)	15657
GE + DNAm AD positive (Test)	3036
GE + DNAm AD negative (Train)	9024
GE + DNAm AD negative (Test)	1365

Table 5.3: Dimensions of the merged subdatasets.

- The last step is to concatenate the train and test subdatasets (positive and negative) to achieve the final tables that will be used in the project. These dataset were also shuffled in order to ensure there was not any bias due to existing patterns in the data. This improves the generalisation and prediction capabilities of the model as it will be forced to learn the existing underlying relationship between features.

Dataset	Total N of samples	N of positive samples	N of normal samples	N of features
GE + DNAm (Train)	24681	15657	9024	700
GE + DNAm (Test)	4401	3036	1365	700

Table 5.4: Dimensions of the final multi-omics datasets.

5.2 Feature Selection

Feature selection is the process of selecting the subset of attributes in a dataset that are the most relevant to the problem being tackled (*Feature Selection* 2021). This type of procedures are typically needed when working with complex data, such as the one being used in this project.

As it was already introduced in the literature review, the data present in the field of genomics falls into the high dimensional low sample size (HDLSS) category, which makes it extremely complicated to work with from a machine learning perspective. Models trained on this type of data tend to have lower accuracies and therefore cannot be relied on and implemented on a clinical level. One critical advantage that comes with the implementation of such methods to datasets from the medical field is the identification of features that are important for biological processes of interest (Bommert, Sun, Bischl, Rahnenführer & Lang 2020). When working with diseases such as AD, where their causes are still not completely clear, this procedures may help gain a better insight on what factors should scientists focus their attention on.

The application of these techniques comes with a lot of technical advantages as well. Some of these are:

- **Model simplification:** When training on a dataset with a big amount of variables, smaller models struggle to generalize the relationship among them. Reducing the number of features allows for the model to be simpler thus easier to understand and interpret.
- **Short training times:** A key advantage of having a simpler model and a smaller dataset is the huge reduction in training time it implies.
- **Overfitting avoidance:** Vast amounts of columns in our dataset will mainly make a smaller model fit perfectly the training data, but will not allow it to generalize, making it useless. If the model is not able to generalize, it will not know what to do when new samples appear.

There are many dimensionality reduction methods available to deal with this type of problem, so an extensive research was conducted prior to implementation. Some of the techniques which were first taken into account are **correlation coefficient**, **Chi-square test** and **analysis of variance**. These three techniques are part of the so-called **Filter methods** category. They select features based on scores acquired via statistical tests, that rate their correlation with the output variable (in this case AD or not-AD). Using filter methods comes with a lot of advantages such as their general design (they can be implemented independently of the type of model used later on) or their computational efficiency.

Despite the great advantages of using the previously mentioned methods, when further research was carried out, a problem arose. As stated in the paragraph above, these methods apply statistical tests to the features in order to figure out their importance but, they do not account for the biological meaning of the data used in this project (Park et al. 2020). After discarding these statistical procedures, a more unorthodox approach was elected.

In this project, a binary-coded genetic algorithm was designated to perform the feature selection. This approach operates in the following manner:

- Each individual represents a feature vector i.e a binary string of length l equal to the number of features where, if there is a 1 in position n of the string, it means that the n^{th} feature of the dataset is selected, and vice versa.

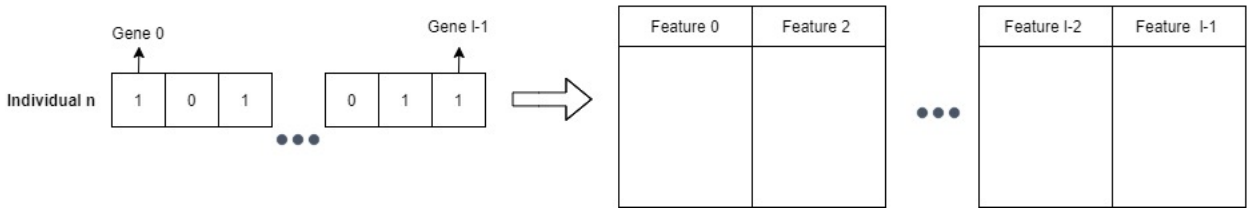


Figure 5.1: The encoding of each individual as a feature vector. The individual n (left) and the dataset it represents (right).

- The fitness of each individual is calculated by first training a raw implementation of the final model on the subset of features selected from the training dataset. After that process is finished, the trained model is tested against the same subset of features taken from the test dataset. The fitness of that individual is then equal to the inverse of the loss achieved by that network. The lower the loss, the fitter the individual.
- The role of the genetic algorithm is to find an optimal combination of features such that minimizes the error.

One of the main reasons behind the choice of using a genetic algorithm as the feature selection method for this project is their domain independence. There are many FS procedures specifically designed to be implemented on genetic data such as **differentially expressed gene (DEG)** and **differentially methylated position (DMP)** based approaches but, due to my insufficient knowledge of the field I decided to opt for a more general technique (Park et al. 2020). GAs on the other hand, have shown to produce impressive results in initially unknown search spaces, where domain knowledge is difficult to provide (Vafaie & De Jong 1992), making them the best option available.

The following subsection will cover in detail the design of this proposed approach, alongside the reasoning behind it.

5.2.1 Genetic Algorithm for Feature Selection: Design explained

Genetic algorithms are highly customizable, making them a great choice for research purposes. This high degree of freedom comes with its own drawbacks, such as the fact that not all designs work in all types of problems.

After an extensive research on hyperparameter choices, the characteristics of the genetic algorithm used are the ones shown in table 5.5.

Genetic Algorithm proposed	
Population size	100
Number of generations	500
Crossover operator	Two point crossover
Crossover probability (p_c)	0.7
Mutation operator	Flip-bit
Mutation probability (p_m)	0.1
Probability of a bit being flipped	0.001
Selection mechanism	Binary tournament selection

Table 5.5: Characteristics of the genetic algorithm proposed.

As presented before, these characteristics made sure that the genetic algorithm fitted the problem in the most accurate way. The reasons behind these choices are the following:

- The population size of 100 individuals ensures a great diversity of solutions in the first generation. For a problem of this dimension, where there are many combinations available, a population of this size grants great exploratory properties, which are necessary for optimal convergence.

- For a problem this wide, a big number of generations will certainly give the algorithm room to explore and exploit the search space. One key factor to have in mind is the amount of computational power that this approach requires as evaluating each individual means training and testing a model. After some runs, 500 generations provided great results without making the process computationally infeasible.

- At the beginning of the research phase, uniform crossover was considered as one possible matting technique for this algorithm. UC's disruptive nature is often a drawback when working with big populations. Two point crossover, on the other hand, still allows the algorithm to explore the search space, but does so in a much less disruptive manner.

- GAs rely on crossover as their main operation to traverse the search space, that is why its probability to happen has to be high enough. In state of the art implementations of GAs, the crossover probability tends to be around 0.6 but, due to the high amount of possible solutions available in this case, the decision to raise it to 0.7 makes it easier for the algorithm to find optimal individuals.

- On the other hand, a high mutation rate would turn the heuristic search into a pseudo-random one, as it would disrupt good individuals by swapping their attributes. This is the main reason why a small probability is more than enough.

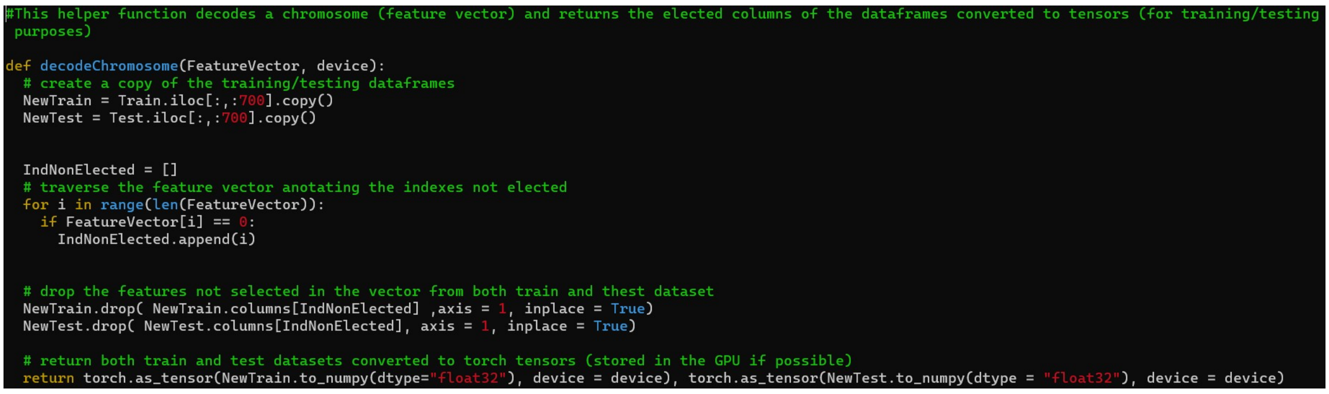
After the structure and characteristics of the genetic algorithm were clearly defined, the last step left was to specify a way of evaluating each individual. This was briefly introduced in section 5.2. A more in-depth explanation is presented in the next subsection.

5.2.2 Genetic Algorithm for Feature Selection: Evaluating the Population

One of the most important parts of a genetic algorithm is defining a way of evaluating each individual. As it was explained in section 2.3 of the literature review, GAs are designed to model a survival-of-the-fittest environment in order to allow the population to evolve. This means that not all solutions are equally valid and need to be sorted in terms of their suitability to solve the problem being tackled.

The solution proposed presents an evaluation process that works as following:

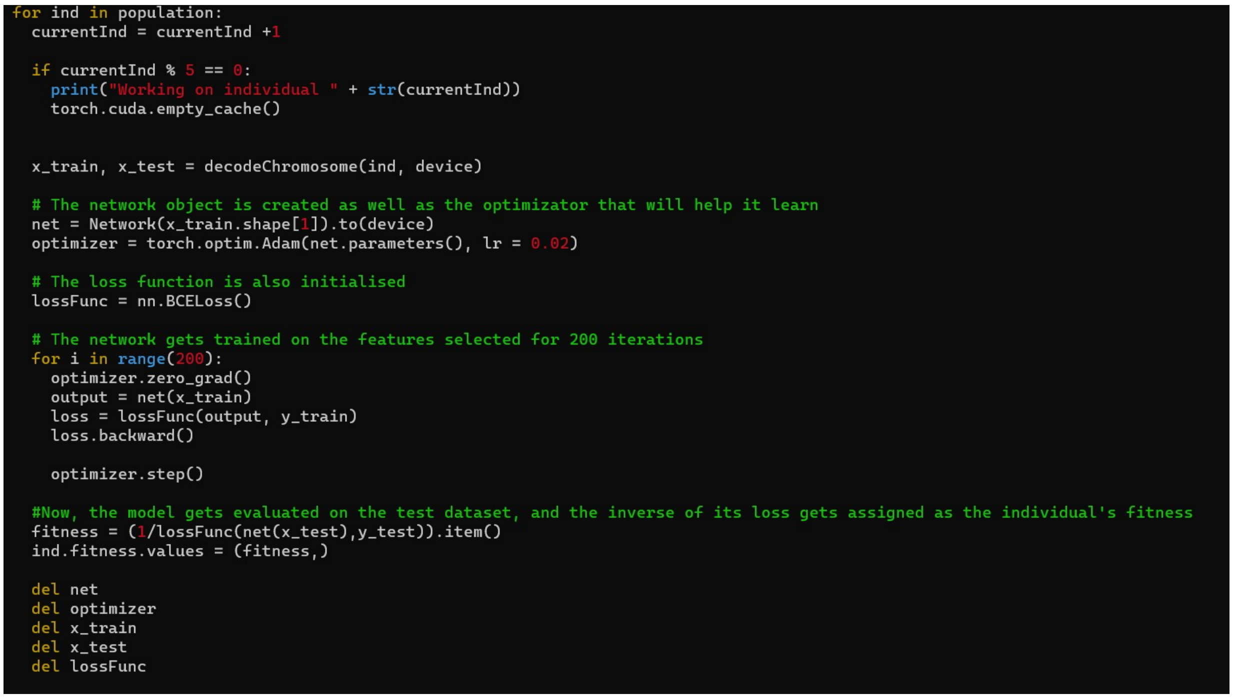
- The feature vector represented by an specific individual gets converted into a train and test dataset. This is done by extracting two subdatasets from the ones created at the end of section 5.1, by only selecting the features specified by the individual.



```
#This helper function decodes a chromosome (feature vector) and returns the elected columns of the dataframes converted to tensors (for training/testing
 purposes)

def decodeChromosome(FeatureVector, device):
  # create a copy of the training/testing dataframes
  NewTrain = Train.iloc[:,:700].copy()
  NewTest = Test.iloc[:,:700].copy()


  IndNonElected = []
  # traverse the feature vector anotating the indexes not elected
  for i in range(len(FeatureVector)):
    if FeatureVector[i] == 0:
      IndNonElected.append(i)


  # drop the features not selected in the vector from both train and thest dataset
  NewTrain.drop( NewTrain.columns[IndNonElected] ,axis = 1, inplace = True)
  NewTest.drop( NewTest.columns[IndNonElected], axis = 1, inplace = True)

  # return both train and test datasets converted to torch tensors (stored in the GPU if possible)
  return torch.as_tensor(NewTrain.to_numpy(dtype="float32"), device = device), torch.as_tensor(NewTest.to_numpy(dtype = "float32"), device = device)
```

Figure 5.2: Function that decodes the individual's chromosome and turns it into two datasets (Train/Test).

- A network object is created. All individuals are evaluated using the same network structure to avoid bias but, as each solution contains a different number of features, the number of neurons of the input layer is adapted. This means that the only difference in the structure of the network used to evaluate the individuals is the number of inputs.
- The network created is trained during 200 epochs in order to adjust its weights. The details of the optimizer and loss function used are in table **??**.
- When the training concludes, the network is tested against the test subdataset acquired in step 1.
- The inverse of the loss on the test subdataset is calculated and then used as the fitness of the individual.



```python
for ind in population:
  currentInd = currentInd +1

  if currentInd % 5 == 0:
    print("Working on individual " + str(currentInd))
    torch.cuda.empty_cache()


  x_train, x_test = decodeChromosome(ind, device)

  # The network object is created as well as the optimizator that will help it learn
  net = Network(x_train.shape[1]).to(device)
  optimizer = torch.optim.Adam(net.parameters(), lr = 0.02)

  # The loss function is also initialised
  lossFunc = nn.BCELoss()

  # The network gets trained on the features selected for 200 iterations
  for i in range(200):
    optimizer.zero_grad()
    output = net(x_train)
    loss = lossFunc(output, y_train)
    loss.backward()

    optimizer.step()

  #Now, the model gets evaluated on the test dataset, and the inverse of its loss gets assigned as the individual's fitness
  fitness = (1/lossFunc(net(x_test),y_test)).item()
  ind.fitness.values = (fitness,)

  del net
  del optimizer
  del x_train
  del x_test
  del lossFunc
```

Figure 5.3: Code that handles the evaluation of the individuals.

Loss function	Binary cross entropy (BCE)
Optimizer	Adam

Table 5.6: Details of the training techniques used.

This evaluation process comes with another great advantage: as it uses simple network implementations that will serve as a starting point when creating the final classifier, this FS algorithm can also be used to test the average performance of different structural combinations. In order to do this, three different runs of the algorithm were executed, each of them using a network structure with different number of layers and neurons. The architectures chosen are shown in tables 5.7, 5.8 and 5.9.

Layer (Type)	Number of neurons	Activation (Function)
Input_layer	x = number of features in individual	None
Hidden_1(Dense)	100	Yes (ReLu)
Hidden_2(Dense)	80	Yes (ReLu)
Hidden_3(Dense)	60	Yes (ReLu)
Hidden_4(Dense)	50	Yes (ReLu)
Hidden_5(Dense)	30	Yes (ReLu)
Output_layer(Dense)	1	Yes (Sigmoid)

Table 5.7: Structural details of Network 1.

Layer (Type)	Number of neurons	Activation (Function)
Input_layer	x = number of features in individual	None
Hidden_1(Dense)	100	Yes (ReLu)
Hidden_2(Dense)	80	Yes (ReLu)
Hidden_3(Dense)	60	Yes (ReLu)
Hidden_4(Dense)	50	Yes (ReLu)
Hidden_5(Dense)	30	Yes (ReLu)
Hidden_6(Dense)	25	Yes (ReLu)
Output_layer(Dense)	1	Yes (Sigmoid)

Table 5.8: Structural details of Network 2.

Layer (Type)	Number of neurons	Activation (Function)
Input_layer	x = number of features in individual	None
Hidden_1(Dense)	100	Yes (ReLu)
Hidden_2(Dense)	80	Yes (ReLu)
Hidden_3(Dense)	60	Yes (ReLu)
Hidden_4(Dense)	50	Yes (ReLu)
Hidden_5(Dense)	30	Yes (ReLu)
Hidden_6(Dense)	25	Yes (ReLu)
Hidden_7(Dense)	20	Yes (ReLu)
Output_layer(Dense)	1	Yes (Sigmoid)

Table 5.9: Structural details of Network 3.

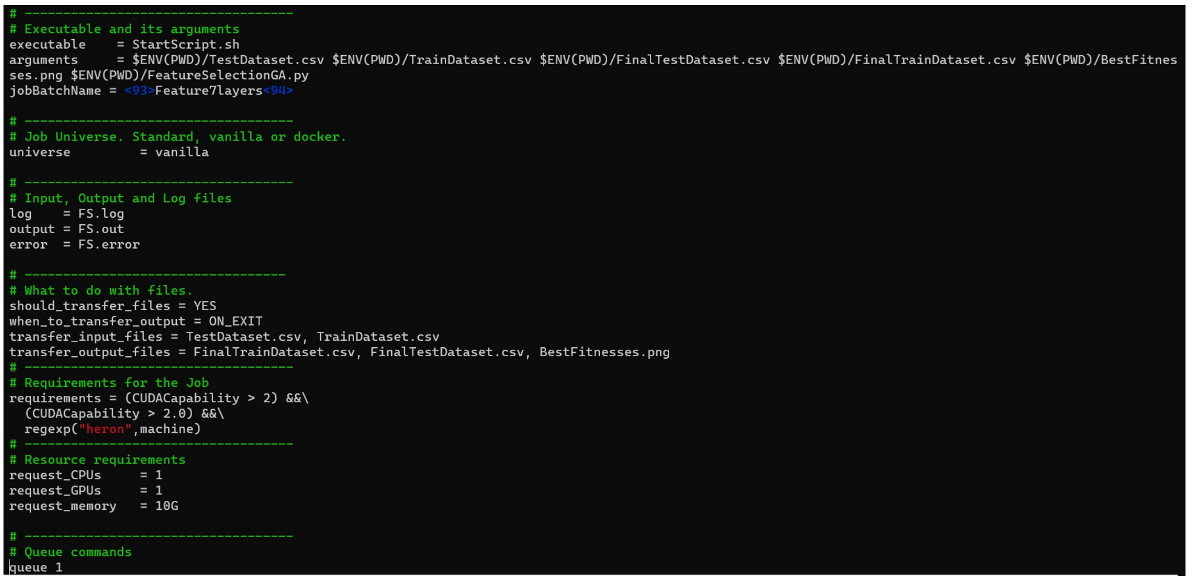
5.2.3 Genetic Algorithm for Feature Selection: Execution

The main drawback that comes with using a genetic algorithm for feature selection is the computational complexity of its execution. Each generation requires the training of 100 networks for 200 epochs each, which would take an absurd amount of time to complete in an average CPU.

In order to reduce the running time of the algorithm, the code was adapted to support training in GPU (only when possible). This was achieved using PyTorch as the deep learning library of choice for building, training and testing the models as GPU training is one of its built in functionalities.

The code would first check for the existence of a CUDA supported GPU. If it was found, all the datasets and models would be created and stored inside it. Otherwise, all the data and operations would be stored and performed in a CPU, so the code can run in any computer with the required dependencies installed. Figure 5.3 shows how the loop that handles the training process creates the network object as well as the subdatasets inside a 'device' which represents the processor previously selected.

The GPUs used to perform the training of the models were the Nvidia Quadro P4000 available in the university's Orca pool. The administration of files and job execution was carried out using HTCondor, a specialized workload management system for compute-intensive jobs. HTCondor uses ʹsubmit_files", where all aspects of the job to be run, such as number of GPUs or amount of memory required are specified. One example of submit file can be seen in the figure bellow.



```
# ------------------------------------
# Executable and its arguments
executable    = StartScript.sh
arguments     = $ENV(PWD)/TestDataset.csv $ENV(PWD)/TrainDataset.csv $ENV(PWD)/FinalTestDataset.csv $ENV(PWD)/FinalTrainDataset.csv $ENV(PWD)/BestFitnes
ses.png $ENV(PWD)/FeatureSelectionGA.py
jobBatchName = <93>Feature7layers<94>

# ------------------------------------
# Job Universe. Standard, vanilla or docker.
universe         = vanilla

# ------------------------------------
# Input, Output and Log files
log    = FS.log
output = FS.out
error  = FS.error

# -----------------------------------
# What to do with files.
should_transfer_files = YES
when_to_transfer_output = ON_EXIT
transfer_input_files = TestDataset.csv, TrainDataset.csv
transfer_output_files = FinalTrainDataset.csv, FinalTestDataset.csv, BestFitnesses.png
# ------------------------------------
# Requirements for the Job
requirements = (CUDACapability > 2) &&\
  (CUDACapability > 2.0) &&\
  regexp("heron",machine)
# ------------------------------------
# Resource requirements
request_CPUs     = 1
request_GPUs     = 1
request_memory   = 10G

# ------------------------------------
# Queue commands
queue 1
```

Figure 5.4: Submit file example.

Each run of the program produced the following output:

- A list of length l equal to the number of generations, containing the best fitness achieved in each population.
- The feature vector corresponding to the individual with the highest fitness.
- Two .csv files containing the train and test subdatasets respectively, only including the features present in the feature vector mentioned above.

The outcome of this feature selection approach is clearly explained in section 6.1 of this report.

5.3 Model Design and Training

Once the feature selection process was defined, the next step was to design and implement a deep neural network for the processing and classification of the samples.

The proposed network architecture is summarized in the table bellow:

Layer (Type)	Number of neurons	Activation (Function)
Input_layer	x = number of features in individual	None
Hidden_1(Dense)	100	Yes (ReLu)
Hidden_2(Dense)	80	Yes (ReLu)
Hidden_3(Dense)	60	Yes (ReLu)
Hidden_4(Dense)	50	Yes (ReLu)
Dropout_1		
Hidden_5(Dense)	30	Yes (ReLu)
Dropout_2		
Hidden_6(Dense)	25	Yes (ReLu)
Hidden_7(Dense)	15	Yes (ReLu)
Output_layer(Dense)	1	Yes (Sigmoid)

Table 5.10: Structural details of the network architecture proposed for classification/prediction.

The network proposed is a modification of one of the models used during the FS process, whose structure is presented in table 5.9. The main reason behind this choice is that, as it can be seen in the results section of this report, the 8 layer architecture already showed very promising results when trained with this type of data for a relative small amount of epochs.

Two dropout layers were added to ensure a better generalization and reduction of overfitting. The dropout rate chosen for both of the layers was 0.5. The standard dropout rate tends to be between 0.5 and 0.8 but, due to the size of the network proposed, the value of 0.5 gave the best regularization results without decreasing the learning power of the model by a excessive amount.

As it was stated in the previous section, three runs of the feature selection process were executed, producing 3 different training and test datasets. In order to determine which one produced better results, 4 network objects were created and trained on either one of three previously mentioned datasets or on the raw multi-omics dataset 5.4 respectively.

In order to compute the error of the models during training, the binary cross entropy loss function was used. The formula of this loss function is shown bellow.

$$BCE = -\frac{1}{N}\sum_{i=1}^{N} y_i * \log(p(y_i)) + (1 - y_i) * \log(1 - p(y_i)) \tag{5.1}$$

As the activation function of the last layer of the architecture proposed is a sigmoid function, the output of a model for a given input i will be a value between 0 and 1 i.e the probability of sample i having AD. Having that in mind, the BCE loss function takes that probability $p(y_i)$

and calculates the error according to how much it differs from the desired output.

One great advantage that comes with using BCE as the error function instead of the previously presented MSE (equation 2.3) is that BCE maximizes the probability of a correct output by heavily penalizing confidently wrong predicted values. This means that, the closer the output is to the wrong class (to 0 if the desired output is 1 and vice versa), the higher the error.

The networks were trained using the ADAM optimizer (Kingma & Ba 2017). This optimizer combines the advantages of two other extensions of Stochastic gradient descent: AdaGrad and RMSProp. This optimizer is one of the most popular algorithms in the field of deep learning as it achieves outstanding results while being computationally efficient.

In order to determine the optimal amount of training epochs, an experiment was conducted. The models were trained for 300, 400, 430, 450 and 500 epochs and their results were compared. This experiment could have been substituted by using early stopping but due to the absence of a validation dataset it could not be implemented. The metrics used to measure the performance of the models in each one of the 5 runs are shown in section 5.4.

In light of the above, the code that handles the training of the network works as follows:

- The 4 datasets which will be used to train the network objects are imported and preprocessed.
- 4 network objects are created. They all share the same structure (table 5.10) and they only differ on their number of input neurons, as each of them will be trained on a different number of input features.
- The same type of optimizer, learning rate and loss function is used for every network to ensure unbiased results.
- The 4 models are trained on their respective dataset for the same amount of epochs. As 5 different runs of this training process are executed, the number of epochs depend on the run.

5.4 Model Testing

In each one of the already mentioned training runs, the performance of the network objects was compared using their respective test datasets. This means that the model trained using the train dataset given by the first run of the FS was tested using the test dataset also given by that same run and vice versa. The last network object was trained and tested on the full multi-omics dataset in order to prove the efficacy of the feature selection approach proposed.

Model	Trained on	Tested on
Model 1	Train dataset (First FS run)	Test dataset (First FS run)
Model 2	Train dataset (Second FS run)	Test dataset (Second FS run)
Model 3	Train dataset (Third FS run)	Test dataset (Third FS run)
Model 4	Train dataset (Multi-Omics dataset)	Test dataset (Multi-Omics dataset)

Table 5.11: The 4 network objects initialised in each run of the train and test process.

The metrics utilized to measure the performance of the models are the following:

- Accuracy
- Area under the receiver operating characteristic curve (AUROC)
- Precision (when predicting AD)
- Precision (when predicting non-AD)

From the metrics presented above, AUROC is the one which will most accurately describe the models' efficacy as it tends to be preferred when the data in the test dataset is unbalanced i.e the number of samples belonging to each class is not equal. Accuracy, on the other hand, tends to be less informative in these cases as it can be overly optimistic e.g if 68% of the samples contained in the test dataset used were labelled as AD, the classifier could obtain 68% accuracy just by always predicting 1.

The outcome of the testing process is also described and interpreted in the next section of the report, specifically, subsection 6.2.

Chapter 6

Results

This section of the report will introduce and interpret the results obtained during the feature selection process as well as the train and testing of the architecture proposed. Furthermore, the performance of the networks introduced in section 5.3 will not only be compared against each other but also against two other classifiers to reaffirm the choice of deep neural networks when tackling this type of problem.

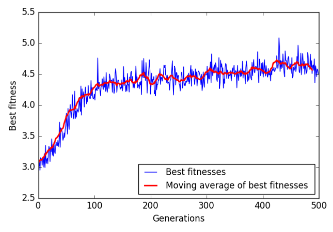
6.1 Feature Selection: Results

As it was stated in section 5.2.3, three different runs of the genetic algorithm were executed, each of them using a different network architecture when evaluating the individuals.

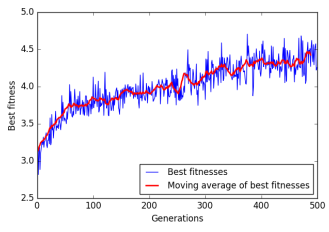
A summarized version of the results obtained in each run can bee seen in table 6.1 and figure 6.1

Run number	Network architecture used	Best fitness achieved	Mean of the best fitnesses	n^o of features selected
Run 1	Network 1	5.08	4.32	349
Run 2	Network 2	4.70	4.002	342
Run 3	Network 3	6.01	4.96	334

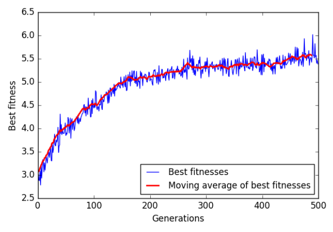
Table 6.1: Feature selection: summary of the results achieved.



(a) Results achieved (Run 1)



(b) Results achieved (Run 2)



(c) Results achieved (Run 3)

Figure 6.1: Plots of the best fitnesses across generations in each run.

After processing the output it can be seen that the genetic algorithm proposed obtains the results expected. All three runs of the feature selection process show a clear evolution of their individuals. Furthermore, the third run achieves the highest results among them. This reaffirms how deeper models tend to outperform shallower ones in problems with this amount of features. When comparing subfigure a and c it can be seen that after 100 generations, the individuals of the run which used a shallower model for evaluation (subfigure a) start evolving much slower than the ones in the run which used the deepest model (subfigure c) due to the previously addressed characteristic.

The line representing the best fitnesses in figure 6.1 can appear chaotic compared to the smoother results that tend to be achieved by GAs in other problems. The main reason behind that pattern is the evaluation method chosen: the training process included in the evaluation of each individual does not always achieve the best results due to the random initialization of weights among other reasons.

In order to obtain a better view of the results, the moving average of the best fitnesses was calculated, using a window of 12. This processing method makes the curve smoother and allows for the plot to be more easily interpreted.

6.2 Model Training and Testing: Results

Section 5.3 summarizes the process of training the architecture proposed using the datasets obtained from the feature selection algorithm as well as the raw multi-omics dataset (table 5.4). As it was stated there, 4 network objects were trained and tested in 5 independent runs in order to not only compare their results but also find the optimal amount of epochs that minimized their error. When measuring the performance of the networks in each of the runs, the metrics presented in 5.4 were used.

After each execution of the training script, the program plots the training loss over the epochs of all 4 network objects (5.11). One example of these plots can be seen in the figure bellow.

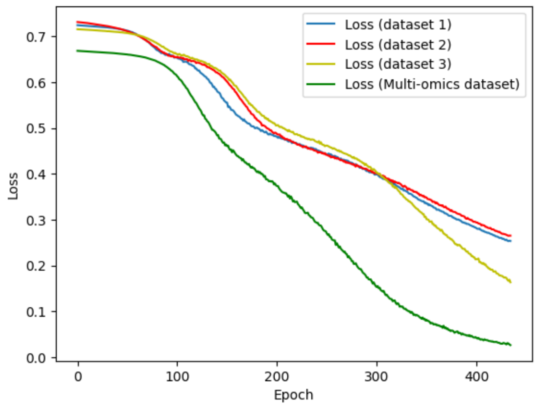
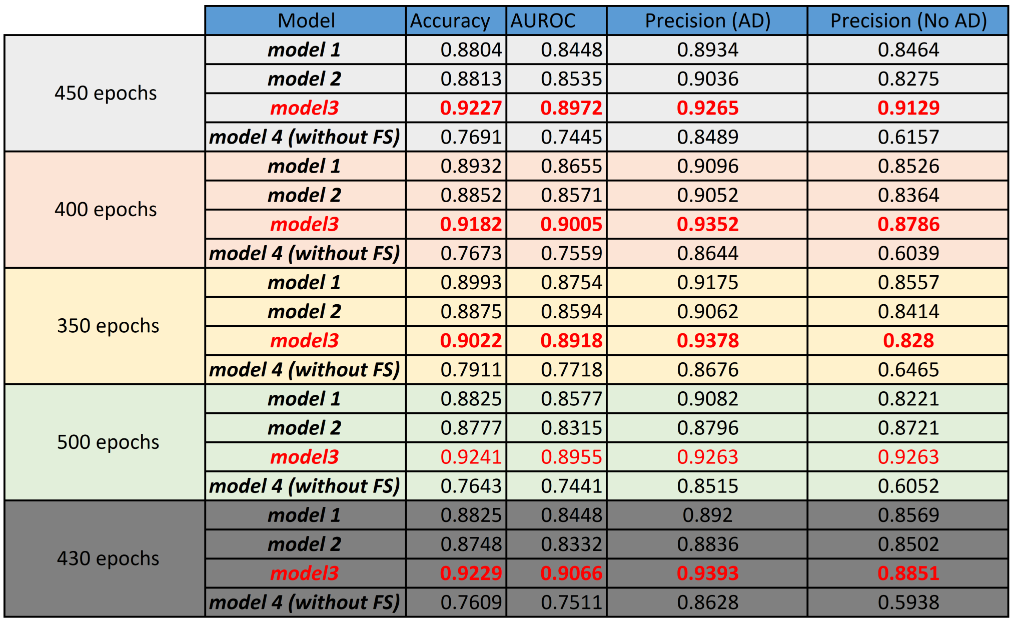


Figure 6.2: Loss of the architecture proposed when trained using different datasets.

Figure 6.2 shows how the training loss of the architecture when trained using the raw multi-omics dataset is considerably lower to the one achieved when using any of the other datasets preprocessed with the proposed FS approach. This behaviour is better understood when compared to the performance of said architecture when being tested.



	Model	Accuracy	AUROC	Precision (AD)	Precision (No AD)
450 epochs	***model 1***	0.8804	0.8448	0.8934	0.8464
	model 2	0.8813	0.8535	0.9036	0.8275
	model3	**0.9227**	**0.8972**	**0.9265**	**0.9129**
	model 4 (without FS)	0.7691	0.7445	0.8489	0.6157
400 epochs	***model 1***	0.8932	0.8655	0.9096	0.8526
	model 2	0.8852	0.8571	0.9052	0.8364
	model3	**0.9182**	**0.9005**	**0.9352**	**0.8786**
	model 4 (without FS)	0.7673	0.7559	0.8644	0.6039
350 epochs	***model 1***	0.8993	0.8754	0.9175	0.8557
	model 2	0.8875	0.8594	0.9062	0.8414
	model3	**0.9022**	**0.8918**	**0.9378**	**0.828**
	model 4 (without FS)	0.7911	0.7718	0.8676	0.6465
500 epochs	***model 1***	0.8825	0.8577	0.9082	0.8221
	model 2	0.8777	0.8315	0.8796	0.8721
	model3	0.9241	0.8955	0.9263	0.9263
	model 4 (without FS)	0.7643	0.7441	0.8515	0.6052
430 epochs	***model 1***	0.8825	0.8448	0.892	0.8569
	model 2	0.8748	0.8332	0.8836	0.8502
	model3	**0.9229**	**0.9066**	**0.9393**	**0.8851**
	model 4 (without FS)	0.7609	0.7511	0.8628	0.5938

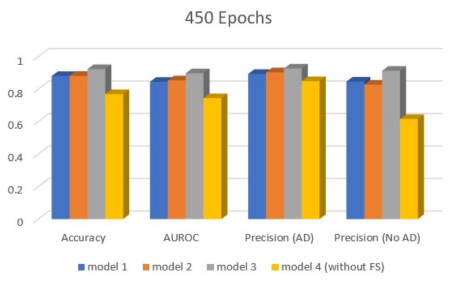
Figure 6.3: Performance of the selected architecture when tested. Each independent run is specified by its number of training epochs.

Before addressing in depth the results achieved, it is worth noting how, even though model 4 (5.11) achieved the lowest training loss among all networks (figure 6.2), it then poorly performed when tested on unseen data when compared to the others. This is a clear sign of overfitting: the model learns how to perfectly fit the training data but fails to generalize so when new unseen samples are feed into it, they tend to be misclassified.

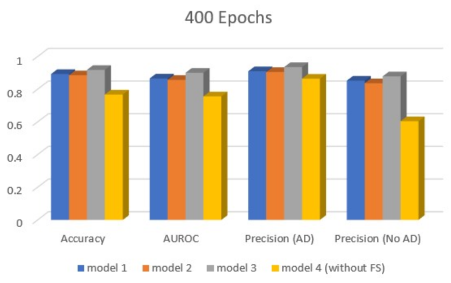
On the other hand, when the architecture is trained using the already preprocessed datasets (Models 1, 2 and 3) its training loss is higher, only because it is generalising much better. This leads to an increase in performance when dealing with unseen data.

Diving into a more in depth interpretation of the results, lets start by comparing the performance of the architecture proposed when trained on each one of the three different preprocessed datasets. As shown in the previous subsection of this report, the third run of the feature selection process achieved the highest fitness of all three and this is directly reflected on figure 6.3. Independently of the number of training epochs, model 3 obtains the highest AUROC values among the three (0.89, 0.9005, 0.891, 0.895 and 0.906). This is due to the greater abstraction capabilities of deeper models, which allowed the third FS run to select a more optimal subset of features than the other two.

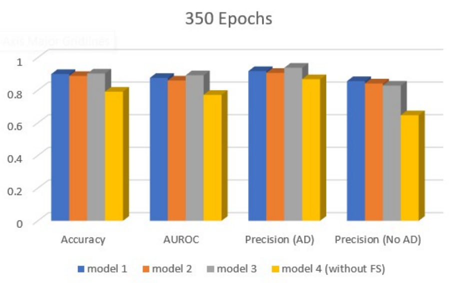
When comparing model 3 to model 4 in any of the runs, their performance difference is even



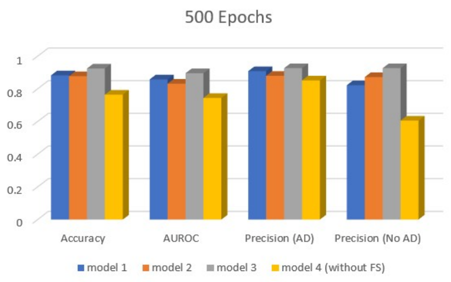
(a) 450 epochs



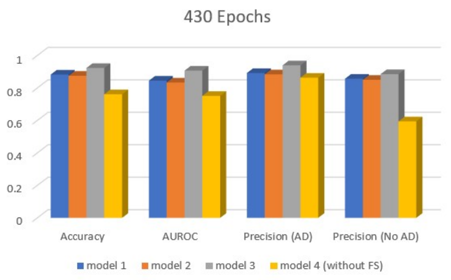
(b) 400 epochs



(c) 350 epochs



(d) 500 epochs



(e) 430 epochs

Figure 6.4: Performance of the selected architecture when tested (bar chart). Each independent run is specified by its number of training epochs.

more noticeable. Model 3 achieves AUROC values at least 0.12 higher than model 4. Moreover, even though the number of normal samples is 42% lower than the number of AD samples in all training datasets, model 3 managed to correctly predict no-AD with at least 18% more precision than model 4 when being tested. This reaffirms the efficacy of the feature selection approach presented.

Section 5.3 also introduces the experiment conducted to determine the optimal amount of training epochs. Figure 6.3 shows how model 3 when trained for 430 epochs obtains the best overall performance among all the classifiers. When focusing the attention on that run, it can be seen how Model 3 achieves an AUROC of 0.9066, which is considered outstanding performance according to (Mandrekar 2010).

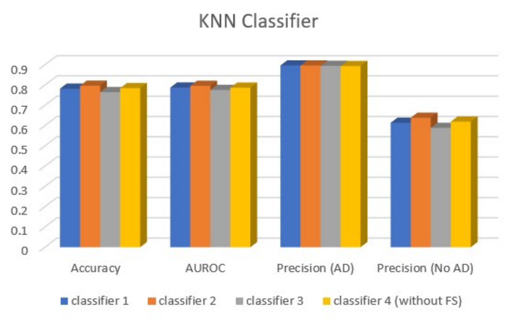
In order to contrast these results, two more classifiers were fitted and tested on all 4 datasets. These classifiers are:

- K Nearest Neighbors classifier.
- Logistic regression.

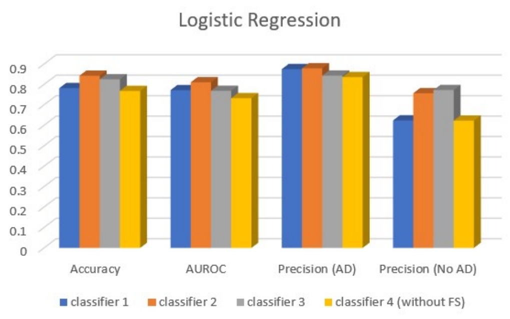
The results obtained by the previously stated classifiers are summarized in the figures bellow.

	Model	Accuracy	AUROC	Precision (AD)	Precision (No AD)
Logistic Regression	***model 1***	0.7816	0.7719	0.8749	0.6236
	model 2	**0.8425**	**0.8102**	**0.8787**	**0.7568**
	model 3	0.8252	0.7685	0.8427	0.7723
	model 4 (without FS)	0.768	0.7324	0.8357	0.6228
KNN classifier	***model 1***	0.782	0.7874	0.8965	0.6139
	model 2	**0.7966**	**0.7959**	**0.896**	**0.6383**
	model 3	0.7648	0.7757	0.8946	0.5884
	model 4 (without FS)	0.7845	0.7868	0.8933	0.6193

Figure 6.5: Performance of the selected classifiers when trained and tested on all 4 datasets.



(a) KNN classifier



(b) Logistic Regression

Figure 6.6: Performance of the selected classifiers when trained and tested on all 4 datasets (bar chart).

Figure 6.5 shows how even simpler classifiers benefit from the feature selection approach presented. Both algorithms achieved the best results when fitted on already preprocessed datasets.

When comparing the results obtained by the best model proposed (model 3 trained for 430 epochs) to the ones achieved by the classifiers it can be seen how the deep learning architecture outperforms other classification methods.

Even though both classifiers obtain good results on the task proposed, they struggle when it comes to correctly classifying normal samples. The architecture proposed clearly shows better learning capabilities as even though the number of normal training samples was much lower than the amount of positive ones, it managed to classify them 12 and 24% more precisely than the logistic regression and KNN classifier respectively.

Chapter 7

Conclusions and future work

This chapter will summarize the research conducted in the project as well as the knowledge obtained during its completion. Moreover, it will also introduce some ideas that could enhance and extend the work done.

7.1 Conclusions

This dissertation has presented a deep learning approach to Alzheimer's disease prediction with the usage of genetic data.

First, some basic machine learning concepts such as activations, backpropagation, optimizers and loss functions, were investigated in depth in order to then be applied to the problem in hand. At the same time, already published work on the matter was also reviewed to extract useful information and ideas that could help in the completion of this dissertation. The outcome of this research phase is reflected in the literature review section of this report.

Subsequently, two different collection of genetic samples were obtained and integrated together to form the multi-omics dataset which would be used in this project.

A binary coded genetic algorithm based feature selection method was designed and implemented in order to extract the most important features from the multi-omics dataset. Three runs of said algorithm were executed, all functioning exactly as expected. Each execution produced new processed versions of the train and test multi-omics datasets given as input. The outcome of these runs shows how the one that used the deepest architecture during evaluation obtained the highest fitness.

The last part of the project was to design a deep neural network that could perform the classification task in an accurate and efficient way. This report presents an 8 layer architecture which was trained and tested on all three datasets obtained in the different feature selection runs in order to compare its performance. This model was also trained and tested on the raw multi-omics dataset to prove the efficacy of the feature selection approach designed.

The model designed obtained an AUROC of 0.9066 (the best result achieved overall) when trained and tested on the third preprocessed dataset, which is considered outstanding performance. The highest AUROC value obtained by the architecture when trained and tested on the raw dataset was only 0.7718. This clearly justifies the use of the FS approach presented.

In order to contrast even more the results obtained and justify the choice of a deep model for this solution, two more classifiers (KNN and LR) were fitted and tested on all 4 datasets. The highest AUROC achieved by the KNN and LR was 0.7059 and 0.8787 respectively.

In light of the above, this dissertation has successfully demonstrated how multi-omics data can be integrated and used to accurately predict Alzheimer's disease with the help of deep learning models and other machine learning techniques.

7.2 Future Work

Despite the fact that the outcome of this project satisfied the objectives set at the beginning of the year, there are some aspects that could be implemented in order to extend the work presented:

The technique used to integrate the genetic datasets used in the project could be replaced by the use of ensemble learning. This consists on the creation of three different classifiers, two of them trained on one of the two genetic datasets used each, and the third one trained on the predictions of the first two. This could improve the classification capabilities of the presented model but would come with drawbacks such as an increase in training time.

The dimensionality reduction of the multi-omics dataset could be performed by an autoencoder. The autoencoder architecture could be first trained to reconstruct the train and test samples. While doing that, the model would also be learning how to extract a good and reduced representation of the features. After a low enough value of reconstruction loss is achieved, a classifier could be trained on the reduced representation of the features that the encoder part of the autoencoder would produce.

When separating the samples into different datasets, a validation set could be created. This would help when it comes to overfitting avoidance as it could be used to perform early stopping.

As it was stated in the methods section, the datasets used contain genetic information of post-mortem brain tissue samples. The nature of said samples makes it difficult for the classifier to be implemented on a clinical level. The work done in this project could be applied to more easy to obtain genetic data such as blood gene expression samples.

Bibliography

Agile Advantages For Software Development | DevCom (2020-02-26GMT+000010:54:21+00:00), https://devcom.com/tech-blog/agile-advantages-for-business/.

Álvarez-Machancoses, Ó., DeAndrés Galiana, E. J., Cernea, A., Fernández de la Viña, J. & Fernández-Martínez, J. L. (2020), 'On the Role of Artificial Intelligence in Genomics to Enhance Precision Medicine', *Pharmacogenomics and Personalized Medicine* **13**, 105–119.

Alzheimer's Society's View on Demography | Alzheimer's Society (n.d.), https://www.alzheimers.org.uk/about-us/policy-and-influencing/what-we-think/demography.

Artificial Neural Network (2021), *Wikipedia* .

Bommert, A., Sun, X., Bischl, B., Rahnenführer, J. & Lang, M. (2020), 'Benchmark for filter methods for feature selection in high-dimensional classification data', *Computational Statistics & Data Analysis* **143**, 106839.

Brownlee, J. (2019), 'What is Deep Learning?'.

Crossover (Genetic Algorithm) (2021), *Wikipedia* .

CS231n Convolutional Neural Networks for Visual Recognition (n.d.), https://cs231n.github.io/neural-networks-1/.

Deep Learning Models in Genomics; Are We There yet? | Elsevier Enhanced Reader (n.d.), https://reader.elsevier.com/reader/sd/pii/S2001037020303068?token=712BF16A81268F4358535AB7790C

Deore, M. (2021), 'Deep Learning : In gradient descent style! — Part 1', https://mdeore.medium.com/deep-learning-in-gradient-decent-style-part-1-ed747e7cc2a3.

Dias, R. & Torkamani, A. (2019), 'Artificial intelligence in clinical and genomic diagnostics', *Genome Medicine* **11**(1), 70.

Earlier Diagnosis (n.d.), https://alz.org/alzheimers-dementia/research_progress/earlier-diagnosis.

Feature Selection (2021), *Wikipedia* .

Feed Forward Neural Network (2019), https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network.

Fitness Proportionate Selection (2020), *Wikipedia* .

GA Roulette Wheel Selection (n.d.), http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php.

Genetic Algorithm (2021), *Wikipedia* .

Girgin, S. (2019), 'Day-12: Kernel SVM (Non-Linear SVM)', https://medium.com/pursuitnotes/day-12-kernel-svm-non-linear-svm-5fdefe77836c.

Jadon, S. (2020), 'Introduction to Different Activation Functions for Deep Learning', https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092.

Jain, V. (2019), 'Everything you need to know about "Activation Functions" in Deep learning models', https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253.

Jebari, K. & Madiafi, M. (n.d.), 'Selection Methods for Genetic Algorithms', p. 13.

Kingma, D. P. & Ba, J. (2017), 'Adam: A Method for Stochastic Optimization', *arXiv:1412.6980 [cs]* . Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Mandrekar, J. N. (2010), 'Receiver Operating Characteristic Curve in Diagnostic Test Assessment', *Journal of Thoracic Oncology* **5**(9), 1315–1316.

Mutation (n.d.), https://www.genome.gov/genetics-glossary/Mutation.

Oppermann, A. (2020), 'What is Deep Learning and How does it work?', https://towardsdatascience.com/what-is-deep-learning-and-how-does-it-work-2ce44bb692ac.

Park, C., Ha, J. & Park, S. (2020), ‘Prediction of Alzheimer’s disease based on deep neural network by integrating gene expression and DNA methylation dataset’, *Expert Systems with Applications* **140**, 112873.

Parmar, R. (2018), ‘Common Loss functions in machine learning’, https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23.

Quiza, R. & Davim, J. P. (2011), Computational Methods and Optimization, *in* J. P. Davim, ed., ‘Machining of Hard Materials’, Springer London, London, pp. 177–208.

Srinivas, M. & Patnaik, L. (1994), ‘Genetic algorithms: A survey’, *Computer* **27**(6), 17–26.

ujjwalkarn (2016), ‘A Quick Introduction to Neural Networks’.

Vafaie, H. & De Jong, K. (1992), Genetic algorithms as a tool for feature selection in machine learning, *in* ‘Proceedings Fourth International Conference on Tools with Artificial Intelligence TAI ’92’, IEEE Comput. Soc. Press, Arlington, VA, USA, pp. 200–203.

Venugopalan, J., Tong, L., Hassanzadeh, H. R. & Wang, M. D. (2021), ‘Multimodal deep learning models for early detection of Alzheimer’s disease stage’, *Scientific Reports* **11**(1), 3254.

Walczak, S. & Cerpa, N. (n.d.), ‘Artificial Neural Networks’, p. 28.