Universidade do Minho
Escola de Engenharia

Óscar Marques Soares

Developing deep learning methods to predict cancer and its outcome from transcriptomics data

Outubro de 2019

Universidade do Minho
Escola de Engenharia
Departamento de Informática

Óscar Marques Soares

Developing deep learning methods to predict cancer and its outcome from transcriptomics data

Dissertação de Mestrado
Mestrado em Bioinformática

Trabalho realizado sob orientação de

Professor Miguel Francisco de Almeida Pereira da Rocha

Outubro de 2019

ACKNOWLEDGEMENTS

I would like to thank Professor Miguel Rocha. Thanks for always helping me when I needed and thanks for giving me all the tools I needed to understand the mysterious and exciting world of deep learning. I would also like to thank for integrating me in the BisBII group while I was only a student. The social and academic experiences not only helped me improve my knowledge but also meeting and interacting with new people.

I would like to thank Vítor Vieira for always being available to help me in any task or difficulty I had. Thank you for being not only a great professional but also a great friend always ready to support me when I most needed. Thanks to my bioinformatics colleagues that were always available to help me and had confidence in my skills in order to ask for my help when they needed. Thanks to my friends for always being present when I needed and for being such an important part of my life.

A really special thanks to U.DREAM Braga for believing in me more than I do myself. UD helped me to know myself, to integrate better in the city of Braga. Furthermore, UD taught me the way for me to achieve my goals and improve myself while changing the world, even if it is by just a little.

Last, but not least, an heartfelt thanks to my parents and my family. I know the sacrifices they did for me to be able to study and that's why I always did my best. I know you don't fully understand why did I choose the degrees I did, but you never ever questioned the reason behind my choices or asked me to change to more known degrees. I hope you're as proud of me as I'm grateful and proud of being your son.

STATEMENT OF INTEGRITY

RESUMO

O cancro é uma das principais causas de morte em países desenvolvidos. Não é uma única doença, mas um grupo de diferentes tipos de doenças com sintomas, tratamentos e prognósticos específicos. O diagnóstico precoce e a determinação do prognóstico são essenciais para selecionar o melhor tratamento para cada caso.

"Deep learning" é um ramo da área da aprendizagem máquina que se tornou popular nos últimos anos. Métodos de "deep learning" têm sido empregados num conjunto de áreas alargado nas quais se incluem veículos autónomos, processamento de linguagem natural, visão por computador, saúde, entre outras.

O objetivo principal desta dissertação é o de desenvolver métodos de "deep learning" para prever cancro e o seu prognóstico a partir de dados de transcriptómica. A revisão da literatura, a exploração de conjuntos de dados, o desenvolvimento de "pipelines" e a validação dos métodos usando casos de estudo são alguns das tarefas necessárias para cumprir os objectivos do trabalho.

Os métodos desenvolvidos constituem uma "pipeline" para criação de modelos a partir de dados de expressão genética. A plataforma é capaz de ler dados de expressão genética, fazer pré-processamento, treino, otimização e avaliação de modelos de aprendizagem máquina tradicionais e de "deep learning".

A plataforma foi demonstrada usando o dataset do *Molecular Taxonomy of Breast Cancer International Consortium (METABRIC)* que contém amostras de pacientes com cancro da mama, como caso de estudo. Os dados de expressão genética de microarrays foram usados para gerar modelos de aprendizagem máquina tradicionais, modelos de "deep learning" e modelos multi-tarefa. Os modelos foram usados para prever a expressão do *receptor de estrogénio (ER)*, do *fator de crescimento epidérmico humano 2 (HER-2)* e do *recetor da progesterona (PR)*, bem como para prever o prognóstico de pacientes usando o *Índice de Prognóstico de Nottingham (NPI)*. Um segundo conjunto de dados permitiu uma validação adicional, considerando dados de RNAseq.

De forma geral, os resultados foram promissores com as tarefas de classificação a obterem bons resultados enquanto os modelos de regressão tiverem um menor desempenho. Enquanto os melhores resultados foram obtidos com modelos de aprendizagem máquina tradicionais, os modelos de "deep learning" estiveram perto e poderiam obter melhores reultados se os dados tivessem um maior número de amostras.

Palavras-chave: Cancro; *Deep learning*; Apredinzagem máquina; Transcriptómica

ABSTRACT

Cancer is one of the major causes of death in developed countries. It is not a single disease, but a group of different types of diseases with specific symptoms, treatments and prognosis. Early diagnosis and prognostic assessment are essential to select the best treatment for each case.

Deep learning is a branch of machine learning that became popular in recent years. Deep learning methods have been employed in a broad range of areas including self-driving cars, natural language processing, computer vision, health, among others.

The main goal of the thesis is to develop deep learning methods to predict cancer and its outcome from transcriptomics data. Reviewing literature, exploring datasets, developing pipelines and validating the methods using a case study are some of the tasks needed to achieve the goals of the thesis.

The developed methods are implemented as a pipeline for creating models from gene expression data. The framework is capable of reading and pre-processing these data, and training, optimizing and evaluating traditional machine learning and deep learning models.

The framework was showcased by using the METABRIC dataset as a case study, which contains samples from breast cancer patients. The gene expression microarray data from the dataset was used to generate traditional, deep learning and multi-task models. The models were used to predict the expression of *Estrogen Receptor (ER)*, the subtype of breast cancer regarding ER, *Human Epidermal Growth Factor (HER-2)* and *Progesterone Receptor (PR)* and the prognosis of breast cancer patients with *Nottingham Prognostic Index (NPI)*, respectively. Another dataset allowed the use of single-cell RNAseq data and confirmed the main trends of the results.

Overall, the results were promising with classification tasks obtaining good results while regression models had a poorer performance. While the best results were obtained with traditional machine learning models, deep learning models were near and could provide better results if the dataset contained a larger number of samples.

Keywords: Cancer; Deep learning; Machine learning; Transcriptomics

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

A

ADAM	Adaptive Moment Estimation.
AE	Autoencoder.
AI	Artificial Intelligence.
ANN	Artificial Neural Networks.
ANOVA	Analysis of Variance.
API	Application Programming Interface.
AUC	Area Under The ROC Curve.

C

CNA	Copy Number Alteration.
CNN	Convolutional Neural Network.
CNV	Copy Number Variation.
COSMIC	Catalogue Of Somatic Mutation In Cancer.
CPU	Central Processing Unit.
CSV	Comma Separated Values.
CUDA	Compute Unified Device Architecture.
CUDNN	NVIDIA CUDA Deep Neural Network.

D

DA	Denoising Autoencoders.
DEPMAP	Cancer Dependency Map.
DI	Departamento de Informática.
DNN	Deep Neural Network.

DT Decision Tree.

E

EN Elastic Net.

ER Estrogen Receptor.

G

GAN Generative Adversarial Networks.

GDC The Genomic Data Commons.

GDSC Genomics of Drug Sensitivity in Cancer.

GEO Gene Expression Omnibus.

GPU Graphics Processing Unit.

GRU Gated Recurrent Unit.

H

HDF5 Hierarchical Data Format.

HER-2 Human Epidermal Growth Factor.

HGP Human Genome Project.

HPA Human Protein Atlas.

I

ICGC The International Cancer Genome Consortium.

J

JSON JavaScript Object Notation.

K

KNN K-Nearest Neighbor.

L

LR Logistic Regression.

LSTM	Long Short-Term Memory.

M

MAE	Mean Absolute Error.
MCC	Matthews Correlation Coefficient.
METABRIC	Molecular Taxonomy of Breast Cancer International Consortium.
ML	Machine Learning.
MSE	Mean Square Error.
MTL	Multi-task Learning.

N

NGS	Next Generation Sequencing.
NPI	Nottingham Prognostic Index.

P

PC	Principal Component.
PCA	Principal Component Analysis.
PCR	Polymerase Chain Reaction.
PR	Progesterone Receptor.

Q

QOL	Quality of Life.

R

R^2	R squared.
RELU	Rectified Linear Unit.
RF	Random Forest.
RGB	Red Green Blue.
RMSE	Root Mean Square Error.

RNN Recurrent Neural Network.

ROC Receiver Operating Characteristics.

S

SAE Stacked Autoencorders.

SGD Stochastic Gradient Descen.

SNE Stochastic Neighbor Embedding.

SVM Support Vector Machines.

T

T-SNE t-Distributed Stochastic Neighbor Embedding.

TANH Hyperbolic Tangent.

TCGA The Cancer Genome Atlas.

U

UM Universidade do Minho.

V

VAE Variational Autoencoders.

W

WES Whole Exome Sequencing.

WGS Whole Genome Sequencing.

1 INTRODUCTION

1.1 CONTEXT AND MOTIVATION

Over the last years, cancer has been one of the major causes of death worldwide. Only in the United States of America in 2018, 1.735.350 new cases were estimated to occur and 609,640 people were estimated to die [6].

Being cancer such a major threat for public health, crucial efforts have been made to prevent, diagnose and treat the disease. Currently, cancer is the main focus of a considerable part of biomedical research worldwide [6].

The survival probability of a cancer patient depends on a wide variety of factors. Cancer specific type and subtype, stage of cancer, genetic profile and more factors are fundamental for the treatment to be effective. A metastasized cancer is generally associated with a poor prognosis and a more aggressive treatment. The effectiveness of the treatment is low, while the costs and side effects are high. In order to reduce the number of deaths associated with cancer, an early and detailed diagnosis is essential.

Cancer research was not always of the same magnitude. The impact of the advent of the *Next Generation Sequencing (NGS)* in biomedical research was huge, including oncology. The use of NGS technologies allowed a large generation of data obtained from cancer samples. Databases as *The Genomic Data Commons (GDC)* contain large amounts of data generated from genome sequencing and other technologies. With GDC and similar databases, the importance of analyzing the existing data can be considered of the same or higher importance than generating new data [7].

The information generated from NGS techniques is portrayed as "Big Data". The term is used to express the large amounts of data that are generated. When considering "Big Data" from NGS, two types of data can be specified: raw data and processed data. Raw data includes all the data directly outputted from the sequencing machine while the processed data consists in the data resulting from the usage of processing methods on raw data, one example being gene expression data. Processed data is smaller than raw data. The problem associated with gene expression datasets is the difficulty in processing and analyzing such large amounts of data. As gene expression datasets generally have significantly more features than samples, the difficulty in training a predictive model is increased.

To create predictive cancer models, machine learning algorithms have been used on gene expression datasets as well as other types of data. Using machine learning algorithms on cancer related data can be useful for a wide variety of tasks. Diagnosing, predicting prognosis, classifying specific types of cancer and predicting the effectiveness of a drug in a specific patient are some of the tasks that machine learning can assist. Tools of this kind are used for achieving what is considered to be precision medicine. In precision medicine, genomic or other types of data are used to create a personalized treatment [8]. We can not consider the models absolutely precise, but they can definitely assist physicians and other health science workers in making the best possible decisions when treating cancer patients [9][10][11].

Deep learning is a branch of machine learning based on artificial neural networks. Comparing to typical artificial neural networks, deep learning models possess multiple hidden layers which are capable of extracting progressively higher level features and patterns from raw data. The concept of deep learning is not new, so a typical question is "If deep learning is not new, how come I have never heard of it?". The answer consists in three points: available data, computational power and algorithms/tools. In recent years, the increase of available data, higher computational power associated with optimized algorithms and tools like *Keras* and *TensorFlow*, allowed to fully exploit deep learning. Deep learning algorithms have been applied to a wide variety of scenarios. Speech recognition, visual recognition and natural language processing are some of the most common applications of deep learning. Facebook, Amazon, Google, Microsoft, Android, Apple and Twitter are some of the companies that use deep learning algorithms in applications of our daily routine. Self-driving cars, product recommendation and language translation are other practical usages of deep learning algorithms. The fact is that deep learning has the potential to significantly change our lives [12].

Biology and biomedical sciences have also been affected by the rise in popularity of the machine and deep learning. The models created with deep learning algorithms can significantly benefit the cancer research community. Despite deep learning being "new", in cancer omics, some papers have been published showing improvements over the common "shallow" learning algorithms. Various deep learning architectures and techniques have been used in the area, including transfer learning and multi-tasking models [13][14][15].

1.2 OBJECTIVES

The main aim of the thesis is the development of deep learning models for the analysis of cancer transcriptomics datasets. The models will be generated using supervised methods to predict diagnostic or prognostic outcomes for specific patients. More specifically, the work will address the following scientific/technological objectives:

- Review the relevant literature for deep learning methods and their applications in related scenarios;
- Studying the available data sources of cancer omics to identify suitable datasets, with an emphasis on the data available on GDC;
- Developing deep learning supervised pipelines for cancer diagnosis and prognostic predictions, using the available data and evaluating the different alternatives based on the defined criteria;
- Addressing specific case studies for the validation of the methods, including different types of cancer, building models, and generating and submitting predictions for unknown data;
- Writing the thesis and, possibly, scientific publications with the main results of this work.

1.3 ORGANIZATION OF THE TEXT

Machine Learning and Deep Learning: The objective of this chapter is to give a basic introduction of the machine learning fundamentals including the most popular types of classical machine learning. Artificial neural networks will be explained in further detailed because they are the basis of the concept of deep learning. Afterwards, the basic concepts of deep learning including architectures and used libraries.

Omics data and machine learning methods in cancer: In this chapter the most popular omics cancer databases will be explained. Applications of bioinformatics in cancer will be reviewed as precision medicine among others. The final of the chapter is expected to show machine learning applications on cancer omics data.

Development: The development chapter contains the explanation of the choices made in the framework software implementation. The structure and the pipeline behind the framework usage are explained as well as specific methods and tools.

First Case Study Showcase of the framework by using it to predict various clinical endpoints in a cancer patients using a gene expression dataset (microarrays). Presentation and discussion of the results regarding the cancer dataset clinical endpoint prediction. Discussion around the benefits and the disadvantages of the framework.

Second Case Study Showcase of the framework by using it to predict if cells are malignant or non-malignant from tumor isolated from melanoma patients gene expression dataset (single-cell RNA-seq). Presentation and discussion of the results regarding malignancy prediction of tumor cells. Discussion around the benefits and the disadvantages of the framework.

Conclusion: Conclusions regarding the result of the developed framework and future works.

2

MACHINE LEARNING AND DEEP LEARNING

2.1 MACHINE LEARNING FUNDAMENTALS

Classical *Artificial Intelligence (AI)* surged in the 1950s with the expectation of being a revolutionary technology. People thought that AI would allow to automatically translate, control and obtain specialized advice from machines. Significant advances occurred in the beginning but soon, many researchers started doubting such techniques. Some researchers changed theirs focus from classical AI to different type of systems. The newly developed systems focused on statistics and rapidly were named as algorithms belonging to the *Machine Learning (ML)* field [16].

Many applications of ML are presented to us in our daily lives. Recommendation systems in websites, search engines, language translation, image recognition and generation are some of the most common applications of ML [16].

Three main types of ML algorithms can be specified regarding their learning process as presented in Table 1: supervised learning, unsupervised learning and semi-supervised learning. In supervised methods, labeled training data are used to induce a model. Unsupervised methods are used to discover new patterns or features from unlabeled data exclusively. Semi-supervised methods usually use a small amount of labeled data and a large amount of unlabeled data [2][16] [17].

Table 1.: Summary of main differences between supervised and unsupervised learning

Type of Learning	**Type of Data**	**Feedback**	**Outcome**
Supervised Learning	Labeled Data	Direct Feedback	Outcome prediction
Unsupervised Learning	Unlabeled Data	Absent	Feature and pattern discovery

2.1.1 *Supervised Learning*

Models trained with supervised learning are able to predict a set of output values given a set of input values. Two types of variables can be predicted in supervised learning: continuous and discrete variables. Models trained with supervised learning can be either classification models (classifiers) or regression models. Classification models are used to predict discrete variables (e.g. cancerous tissue or normal tissue), given a set of input features from the sample. Regression models can predict a continuous variable (e.g. concentration of alcohol in the blood) given a set of input features of a sample. The general pipeline starts by collecting raw data as shown in Figure 1. The pre-processing of raw data includes treating missing values and extracting features. Missing values cannot exist in the training data, therefore handling missing values by generating values from averaging features or removing samples is necessary. Furthermore, extracting columns of non-significant features is also an important aspect to achieve better results [2][9].

After applying the pre-processing methods, training data is split into training and test datasets, with the first being typically larger than the second. While the training dataset is used to train the model, the test dataset is used to evaluate the performance of the model. Feature normalization is applied rescaling different features to comparable values. Feature selection can be applied not only to use the features that are relevant to the problem but also to improve computer efficiency by reducing the number of features analyzed [2][9].

The processed training data is used to train the model. Each time the algorithm iterates, the values of the parameters are updated and used. In the end of the training process, error is estimated. To generate an error estimation a loss function is needed. The function calculates the error between the predicted values and the real values. The loss is essential for training algorithms as backpropagation. In these algorithms, the loss function calculated error is used to update the coefficients of the model. The term ”loss” is applied to a single sample while ”cost function” is used when multiple samples are predicted. Afterwards, the test dataset is used to evaluate the performance of the model. Figure 1 represents a simple view of the general pipeline used in supervised learning problems [2][9].

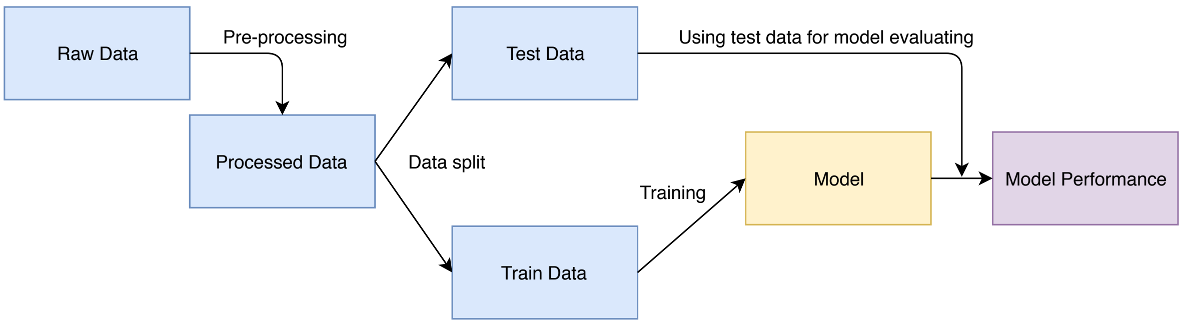


Figure 1.: Overview of a supervised learning pipeline

Error Estimation

The performance of a model can be assessed by using independent test data. Different error estimation methods can be applied to evaluate the performance of the model including hold-out, cross-validation and leave-one-out.

Holdout The holdout method is one of the simplest partition techniques in ML. This method consists in splitting the training data in two separate sets: training set and test set. Generally in holdout, 2/3 of the dataset are used for trainning while the remaining 1/3 will consist in an independent test set. Holdout is a good option when large amounts of data are available. If the available data is small, the error estimation is highly variable depending on the partitions created [18].

Cross-validation K-fold cross-validation consists in splitting the data into k partitions, also called folds. One of the folds is chosen to be used for test and the remaining k-*1* are used for training. Thereafter, the previously stated process is repeated k times for the different partitions. In the end of the cross-validation, the overall error is calculated averaging the error from each iteration. Cross-validation is the most popular error estimation method used in ML [18].

Leave-one-out Leave-one-out is a specific case of cross-validation where k is equal to the number of available samples in the data N. In leave-one-out, N error estimations are calculated, with only one sample being used as test data in each iteration. The method is not generally applied because of the associated high computational cost. Still, leave-one-out can be a useful when small amounts of data are available [18].

Metrics

The performance evaluation of a model can be done using different error metrics. Different types of metrics can be used depending if the problem is a classification or a regression one. Confusion matrices are applied in binary classification problems. For classification problems, the concept of confusion matrix (Table 2) is essential. While *tn* and *tp* represent the correctly predicted negative and positive instances respectively, *fn* and *fp* correspond to incorrectly predicted negative and positive instances, respectively.

Table 2.: Confusion Matrix for binary classification problems adapted from Hossin et al. [4].

	Real Positive	**Real Negative**
Predicted Positive	True positive (*tp*)	False positive (*fp*)
Predicted Negative	False negative (*fn*)	True negative (*tn*)

A wide range of metrics can be formulated from the values observed in a confusion matrix. The most common metrics can be observed in Table 3. The last four metrics are extensions of binary classification metrics for multi-class classification problems. Accuracy, precision and recall are popular metrics for evaluating performance. In Table 2, accuracy is formulated for a binary classification problem, but this metric can be also applied to multiclass problems. The formulation remains similar as overall accuracy is calculated by dividing the sum of correct predictions by the total number of predictions. Each metric has advantages and disadvantages when comparing to the others. F-measure and MCC have been reported to be better in evaluating performance of models when comparing with other metrics because they compute a balance between positive and negative measures, a characteristic not present in other metrics [4][5].

Table 3.: Metrics used for binary and multi-class classification problems based in Hossin et al.[4] and Bourghorbel et al. [5].

Metrics	Formula	Evaluation Focus
Accuracy (acc)	$\frac{tp+tn}{tp+fp+tn+fn}$	Ratio of correct predictions over the total number of instances
Error Rate (err)	$\frac{fp+fn}{tp+fp+tn+fn}$	Ratio of incorrect predictions over the total number of instances
Precision (p)	$\frac{tp}{tp+fp}$	Positive patterns correctly predicted from total positively predicted instances
Recall(r)	$\frac{tp}{tp+tn}$	Positive patterns correctly predicted from the total of correct predicted instances.
F-Measure	$\frac{2*p*r}{p+r}$	Harmonic mean between recall and precision values
MCC	$\frac{tp*tn-fp*fn}{\sqrt{(tp+fp)(tp+fn)(tn+fp)(tn+fn)}}$	Balanced measure that can be used in classes with considerably different sizes
Averaged Accuracy	$\frac{\sum_{i=1}^{l}\frac{tp_i+tn_i}{tp_i+tn_i+fn_i+fp_i}}{l}$	Average effectiveness of all classes
Averaged Error Rate	$\frac{\sum_{i=1}^{l}\frac{fp_i+fn_i}{tp_i+tn_i+fn_i+fp_i}}{l}$	Average error rate of all classes
Averaged Precision	$\frac{\sum_{i=1}^{l}\frac{tp_i}{tp_i+fp_i}}{l}$	Average of per-class precision
Averaged Recall	$\frac{\sum_{i=1}^{l}\frac{tp_i}{tp_i+fn_i}}{l}$	Average of per-class recall
Averaged F-Measure	$\frac{2*p_M*r_M}{p_M+r_M}$	Average of per-class F-measure

Area Under The ROC Curve (AUC) is one of the most used metrics. As understood by the name, the value reflects the area under the *Receiver Operating Characteristics (ROC)* curve. A ROC curve is a plot that visually describes the performance of a binary classification problem. The x axis represents the false positive rate, while the y axis represents the true positive rate. The two previous values are calculated by dividing *tp* by condition positive and by diving *fn* by condition negative, respectively. Unlike the aforementioned metrics, the AUC represents the overall performance of a classifier model. For a binary classification problem, the metric can be determined using Equation 1.

$$AUC = \frac{S_p - n_p(n_n + 1)/2}{n_p n_n} \tag{1}$$

where is calculated with $S_p = \sum r_i$ where the r_i corresponds to the *ith* positive in the ranked list. The number of negative and positive instances are represented by n_n and n_p, respectively. AUC not only can be used in binary classification problems but also in multi-class problems. Although the AUC value shows better results when comparing to other metrics as accuracy, the computational cost associated with AUC is high. The metric excels in discrimination and evaluation tasks, but when large amounts of data are processed, the time complexity of the problem is not negligible [4] [19][20].

The continuous output of regression problems require other type of metrics. *Mean Square Error (MSE)*, *Root Mean Square Error (RMSE)* and *Mean Absolute Error (MAE)* are the most popular regression metrics. RMSE reflects the standard deviation between real and predicted values and a easier way to interpret MSE. MAE is the average of the absolute error between real values and predicted values. While the MAE score is linear with all differences having the same weight, in RMSE the same does not occur [21][22][23]. Equations 2 and 3 represent the formulas for RMSE and MAE, respectively. MSE equation is equal to RMSE formula squared.

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{j=1}^{n}\left(y_j - \hat{y}_i\right)^2} \tag{2}$$

$$MAE = \frac{1}{n}\Sigma_{j=1}^{n}|y_j - \hat{y}_i| \tag{3}$$

where n represents the number of total predictions, y_j and $\hat{y}_j$ represent the real output value and the predicted output values respectively.

Adjusted R Squared (adjusted R^2) is another metric commonly applied to regression problems. Adjusted R^2 is used to understand how independent variables influence the variability of dependent variables [24]. Equations 4 and 5 are the equations for R^2 and adjusted R^2.

$$R^2 = 1 - \frac{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\frac{1}{n}\sum_{i=1}^{n}(y_i - \overline{y}_i)^2} \tag{4}$$

$$R^2_{adj} = 1 - \left[\frac{(1-R^2)(n-1)}{n-k-1}\right] \tag{5}$$

while n,y_i and $\hat{y}_i$ represent the same as in RMSE and MAE formulas, k is the number of explanatory variables present in the model.

Overfitting

As ML models are trained by using limited sets of data, it's common that the model becomes exceptional in predicting values or classes from the samples in the training. The problem is that the model can be so overly fitted to the training data, that it becomes obsolete when is used with new samples. Overfitting is the term used to represent this problem. Some methods are used in order to decrease overfitting. Regularization methods as L1 and L2 regularizations are popular algorithms to assess overfitting problems. Cross-validation is a popular technique used to detect overfitting as multiple models are trained with different subsets of samples and by averaging the performance of all the models.

2.1.2 *Unsupervised Learning*

The goal of *unsupervised learning* differs from the goal of *supervised learning*. Unsupervised learning intent is not directly to classify or predict a value given a set of N observations. The objective is to detect patterns in the given data, without using labeled data. As large amounts of data do not contain viable labeled data to be used, unsupervised learning is a good alternative to supervised learning. The detected patterns can be useful regarding the data and the context of the data. Furthermore, recently *semi-supervised learning* models have been implemented for a wide range of problems. The value of semi-supervised models consists in the capability of training supervised models without containing labeled data for all the input observations [25].

As shown in Figure 2, the pipeline used for unsupervised learning is less complex when compared to the aforementioned supervised learning pipeline. The pre-processing step is similar, but the data does not need to be split in train, validation and test partitions to build the model. Afterwards, by analyzing the built model, it is possible to improve the model. The process of tuning consists in changing the values of the hyper-parameters to improve the performance of the model.

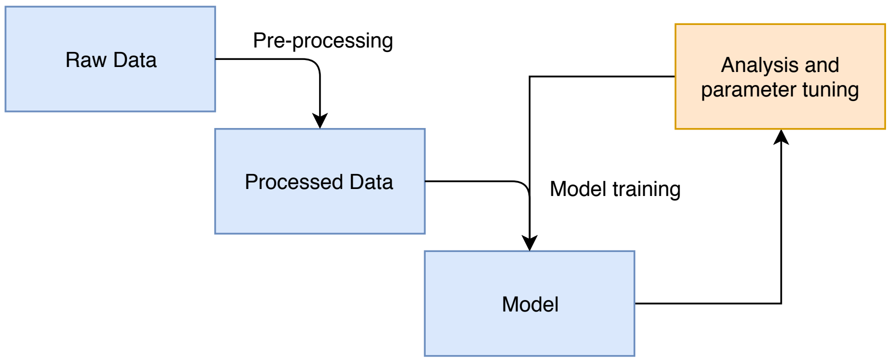


Figure 2.: Overview of an unsupervised learning pipeline

The most common types of unsupervised learning models are *k-means*, *hierarchical clustering* and *Principal Component Analysis (PCA)*. *t-Distributed Stochastic Neighbor Embedding (t-SNE)* is a recent approach of an unsupervised model focused on data dimensionality reduction.

2.2 SUPERVISED ML MODELS

2.2.1 *Linear and Logistic Regression*

Linear Regression

Linear regression is one of the first algorithms to be used in ML applications. For the last 30 years, linear regression has been used and remains an important application in statistics and ML in general. Linear models receive a vector of input values $x^T = (x_1, x_2, ..., x_p)$ and the model predicts the output y, a numerical [25]. Equation 6 corresponds to the linear regression formula.

$$\hat{y} = \theta_0 + \sum_{j=1}^{p} x_j \theta_j \tag{6}$$

where $\hat{y}$ is the output of the linear regression model for a given x^T vector, p is the number of values in the input vector, θ is a vector with *parameters* values and θ_0 is the intercept, commonly referred as *bias* in the ML community [25]. If variable 1 is included in X and the bias is included in the vector of parameters coefficients θ, the linear regression can be calculated as an inner product in a matrix/vector form as shown in Equation 7.

$$\hat{y} = x^T \theta \tag{7}$$

Using the second variation of the linear regression formula, better computational efficiency can be obtained.

In order to train a linear regression model, a cost function is needed, the most common being MSE. For parameter estimation,the goal is to minimize the value of the cost function, updating the coefficients in each iteration [25].

Logistic Regression

As *Logistic Regression (LR)* is used for binary classification problems, the dependent variable is discrete. The term "logistic" refers to the sigmoid or logistic equation applied to linear regression. The sigmoid equation is represented as Equation 8.

$$\phi(z) = \frac{1}{1+e^{-z}} \tag{8}$$

If the value of z is equal to 0, the sigmoid function probability is 0.5. If $z >> 0$, sigmoid function approaches 1 and if $z << 0$, the function approaches 0. By considering the z value equal to a multiplication of a vector of weights (coefficients of parameters) by a given input vector, the LR is formulated as showed in Equation 9 [25][26].

$$h_\theta(s) = \frac{1}{1+e^{-\theta^T x}} \tag{9}$$

The weights θ are estimated by minimizing the cost function $J(\theta)$. Equation 10 represents the formula for the cost function for LR [25] [26].

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right] \tag{10}$$

Regularization

Regularization methods introduce components to the cost function that represent model complexity in the models to reduce overfitting and increase the robustness of the model. Two major model regularization methods are generally used: L1 or Lasso regularization and L2 or Ridge regularization.

L1 or Lasso regularization can be used for variable selection or shrinkage with linear and logistic regression models. Lasso regularization consists in the addition of the absolute values of the parameters multiplied by a λ hyperparameter. With the increase of λ the model tends to under-fit, while a low value of λ tends to have a small effect in decreasing the overfitting of the model. The parameter shrinkage is proportional to the parameters value θ. The cost function with L1 regularization is shown in Equation 11 [25].

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 + \lambda\sum_{j=1}^{n}|\theta_j| \tag{11}$$

where h_θ is the function for a linear or logistic regression that outputs the sum of the multiplication of parameters with input values. In Equation 11, L1 regularization is added to the MSE cost function of a linear regression [25].

L2 or Ridge regularization consists in an additional term added to the cost function similarly to R1 regularization. While L1 regularization uses the absolute values, R2 regularization uses the square of the parameters values. The cost function with Ridge regularization is represented in Equation 12 [25].

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 + \lambda\sum_{j=1}^{n}\theta_j^2 \tag{12}$$

While Ridge is not a robust solution, it has always one single stable solution, Lasso is robust but has multiple unstable solutions. Lasso can shrink and select variables automatically while Ridge is useful to work with values different from zero because it is efficient in calculating the analytic solutions [25].

*Elastic Net (EN)*s are a combination of Lasso and Ridge regularizations. ENs combine the feature elimination of L1 regularization with the coefficient reduction of L2 regularization. The cost function for an EN is show in Equation 13 [25][27].

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 + \lambda\sum_{j=1}^{n}\theta_j^2 + \lambda\sum_{j=1}^{n}|\theta_j| \tag{13}$$

An additional parameter α is used in some implementations to balance the weight of each type of regularization technique has in the cost function as seen in the *Scikit-learn* package [28].

2.2.2 *K-Nearest Neighbor*

K-Nearest Neighbor (KNN) is a non-parametric algorithm used for classification and regression. Being non-parametric allows the algorithm to be used in cases where there is no knowledge regarding the dataset and can be used when the data does not have a normal distribution [29]. KNN is a called a lazy learner because the training phase is almost instantaneous. Comparing to a LR where during training phase the weights are updated, the KNN instance only memorizes the training data. The computation mainly occurs when a new sample is being classified. The KNN algorithm calculates the distance among the sample to be predicted and the "k" nearest neighbors in the search space. The "closest" neighbors will serve as reference to classify the new sample. The distance between samples can be calculated with different distance metrics, but the default metric for KNN is the Euclidean distance. The Euclidean distance between x and y samples is shown in Equation 14 [30].

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(a_i(x) - a_i(y))^2} \tag{14}$$

where "n" is the number of attributes in the samples and "a" is the value of an attribute in x and y instances. The value of K is an important factor that affects KNN models performance. Having a small K can result in disparity of prediction values while a large value of K can result in a model with a large bias. When used for binary classification problems, it is a good practice to select odd numbers to avoid ties between votes. The previous practice can be extended to multi-class classification problems by not using multipliers of the value of possible classes [31].

2.2.3 *Support Vector Machine*

Support Vector Machines (SVM) are ML models based in four basic concepts: maximum-margin hyperplane, kernel function, hyperplane separation and soft margin. The **hyperplane** concept consists in a straight plane in a high-dimensional space, capable of separating the samples in the different categories. The term hyperplane is used because the straight line can be used in one dimension (a point), in two dimensions (a straight line), in three dimensions (a plane) and can mathematically be applied to higher dimensions.

Maximum-margin hyperplane consists in choosing the best hyperplane to separate the samples of different classes. The samples to be classified can be separated by various high-dimensional hyperplanes. What differentiates SVMs from other classifiers is the criterion used in hyperplane selection. The chosen hyperplane is the one that separates two classes with maximal distance between any data points from samples of distinct classes.

Soft margin is an SVM feature used to ignore part of the samples when computing the loss function. Errors in datasets are common and can improve the difficulty of finding the appropriate high-dimensional hyperplane. If a single "error" has similar features to the other class, then the chosen hyperplane would not separate the datasets easily. Soft margin allows for the chosen hyperplane to contain outliers in the margin without affecting the model training process.

Kernel functions are applied to the data for simplifying the selection of the hyperplane. Frequently, datapoints cannot be easily separated in a low dimensional space. Kernel functions project the data to higher dimensional spaces, allowing the data to be linearly separated. The choice of the appropriate kernel function is fundamental to the quality of the SVM. The associated problem with the application of kernel models is the ”curse of dimensionality”. With the increase in the number of variables, the number of possible solutions and the probability of overfitting also increases. Furthermore, kernels can be applied to non-vector inputs, increasing the types of data processed by SVMs. The combination of different types of data can occur because of the mathematical formalism that kernels provide [25][32].

2.2.4 *Decision Trees and Regression Trees*

The application of *Decision Tree (DT)*s in ML was one of the first algorithms used to create predictive models. DTs can be applied to classification while regression trees can be applied to regression problems for discrete and continuous values predictions, respectively. In a trees, each node corresponds to an input variable and each branch to a possible value. Trees are efficient in the training process and easy to analyze. ID3 is one of the most popular DTs, but further algorithms have originated from ID3. The class of a sample can be predicted by crossing a DT from the top to the bottom evaluating the values for each node(feature). Generally, the features with highest variation are the first to appear from a top-bottom perspective. DTs have been used in ensembles as random forests and gradient boosting algorithms because of the high efficiency associated to training [25][33].

2.2.5 *Ensembles*

In the current state of the art in ML, generally more than one model is trained for each problem. The usage of multiple models in the same problem is called an ensemble. With multiple models being used to generate outputs, some methods are necessary for choosing the right prediction. Generally using ensembles decreases the risk of model overfitting and improves the results.

Voting methods are used to predict the class output of the ensemble. Two types of methods are currently used for class prediction models: majority voting or weighted voting. Majority voting consists in creating a pool with the predicted class from each model and the chosen ensemble output corresponds to the class with most votes. Weighted voting process is similar but each vote has a different ”importance” in the voting process depending on the model’s performance. A model with better predictive quality has more weight in the voting process when compared to models with worse results. For regression problems, generally two methods are used: averaging and weighted averaging. The simple averaging method consists in calculating the mean of the outputs generated by the models. The weighted averaging method is similar to the weighed voting since different weights are applied to each model output value [25] [34].

Bootstrap Aggregation (Bagging)

Bootsrap aggregation, also known as bagging, is an algorithm used to create ensembles. Bagging can be used for different types of ML algorithms including trees, neural networks among others. Given a D dataset, m models are created with different randomly selected samples. It is important to emphasize that samples selected in an iteration, can be selected again in the following. The number of samples used in each model training is the same. Generally, 60% of the samples are used to train each model. Afterwards, the outputs of the m models are averaged or voted in classification problems and the final output of the ensemble is calculated in regression problems [25][35].

Boosting

Boosting can be considered a meta-algorithm to reduce bias and variance. Boosting ML algorithms convert weak learners into strong learners. In the beginning, all data points have the same weight and consecutively the same probability of being chosen to train the model. After each weak model, the values of the weights are re-calculated. Samples classified incorrectly have a higher weight while correctly classified samples have lower weights. Calculating new weights after each model, improves the results of the ensemble because the next model is trained using more misclassified data. The described process causes the next model to learn data the previous model could not learn. The final output of a boosting method is the result of the voting or averaging of all generated models. The most popular boosting algorithm is AdaBoost, but other boosting methods exist as xgboost, LPBoost and gradient boosting [25][36][37][38][39].

Random Forest

Random Forest (RF) consists in an ensemble of DTs. RF can be used for classification and regression problems as trees can process both types of ML problems. RF has a similarity to bagging algorithm. Each generated tree uses only a subset of data in the training process, similar to bagging. Furthermore, instead of selecting the features with highest variance for criteria for spliting in the nodes, the features used in each tree are selected randomly. Using only a subset of data in the training process and the random selection of features lowers the correlation among individual trees. The robustness of RF results from the trees being different among them to be able to captured features with low correlation. Further features have been added to RF base algorithm. *Extremely randomized trees* or ExtraTrees add more randomization to RF base algorithm. Each model generated by ExtraTrees uses the full dataset for training and the *cut-point* for each feature(node) is randomized instead of being calculated the optimal value [25] [40].

2.2.6 *Feature Selection*

Feature selection is an essential step for the aforementioned models and supervised learning in general. Selecting the best subset of features to be used for model training is the goal of feature selection. Features can be selected from already existing feature or new features can be obtained by algorithms as PCA. Feature selection methods can be divided among three major categories: filters, wrappers and embedded methods. **Filters** select features without being influenced by model type. Filters are based in correlations and other statistical metrics (e.g. entropy or information gain). The evaluated metrics are not specific for each model, because they consist on measuring general properties and ranking feature ”usefulness” by the resulting statistics.

Unlike filter methods, **wrapper methods** evaluate the interactions among variables. Wrapper methods have a high computational time and effort when compared to filters because multiple models must be trained with different subsets of features. The risk of overfitting by using wrapper methods is high if the number of samples is reduced.

Embedded methods are similar to wrapper methods. The difference remains in the fact that embedded uses a metric intrinsic to model training to train the model simultaneously to feature selection. One example of an embedded method are elastic nets. They use L1 and L2 regularization techniques during the training while features are ranked by their value multiplied by the coefficients. The lower ranked features are removed from the training process [25][41][42] .

2.3 UNSUPERVISED ML MODELS

2.3.1 *Hierarchical Clustering*

The concept of hierarchical clustering is based in dividing data in clusters from top to bottom or reverse. The algorithm is called hierarchical because depending on the level that is examined, the members and number of clusters are different. In hierarchical clustering the similarity is determined by calculating the distance among observations. The clustering of observations iterates, grouping observations or groups of observations with the lowest distances. Clusters in the bottom level contain a single observation while the above levels contain more than one instance, with the top cluster containing all observations. Observations can be grouped using two different methods: divisive and agglomerative.

The divisive method starts the clustering process with a single cluster with all the observations. Each iteration of the algorithm splits a cluster into two separate clusters. The goal is to generate two different clusters with the highest distance/lowest similarity between two groups of observations. The agglomeration method consists in a bottom to top approach. In each iteration of the method, the two observations or groups of observations with largest similarity/lowest distance are merged together into a single cluster. Each level has always one less cluster when compared to the level below. An example of an hierarchical clustering is illustrated in Figure 3. Different distance metrics can be used including Euclidean and Manhattan distances. Average-linkage, single-linkage and complete-linkage are the three main methods used to calculate the distance between clusters. While in single-linkage the distance between two clusters is given by the the distance between the smallest distance between two points in each cluster, in complete-linkage, the distance is given by the longest distance between two points in each cluster. Average-linkage considers the distance as the average of the distance of each point with all the points in the other cluster. [25][43].

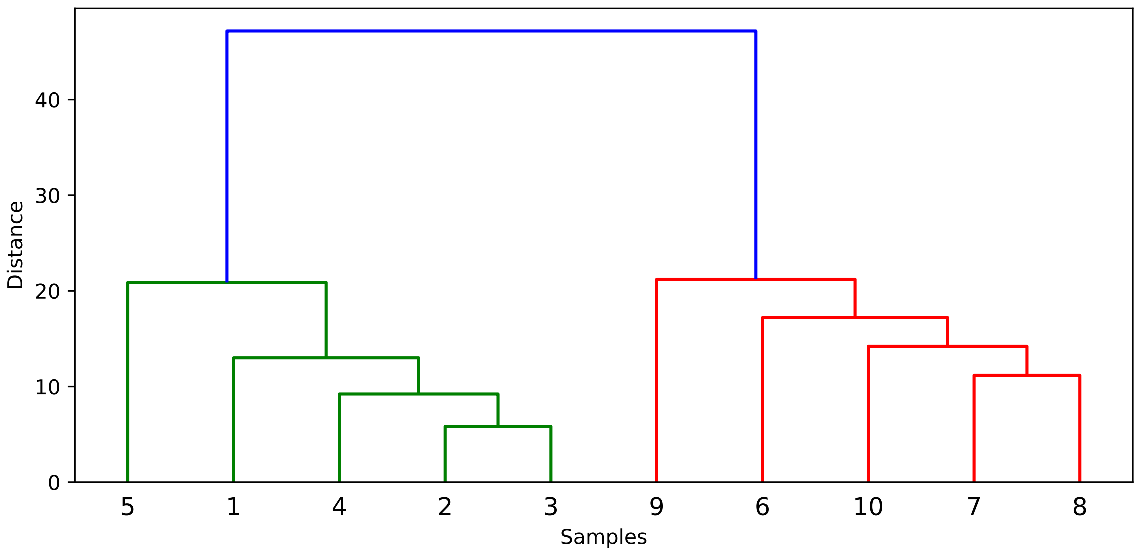


Figure 3.: Schematic representation of an hierarchical clustering.

2.3.2 *K-means*

K-means is a class of clustering problems seeking to group observations with similar patterns in clusters. In k-means, k is the input number of clusters provided as a parameter. The performance of the algorithm is highly influenced by the value of k because each observation will be assigned to one of the k clusters. At algorithm start, a k centroids are set equal to some observations. The goal of these centroids is to be the center of each cluster. In each iteration, each observation is assigned to the closest centroid and posteriorly, the values of the centroids are recalculated to the mean value of the assigned observations. This recalculation can be seen as an optimization problem as the minimum value for the objective function is searched in each iteration. The objective function is calculated by the sum of the squared distances between each point and the centroid of the correspondent cluster. The distance used to measure the distance among points and centroids is Euclidean distance. The model runs until a given number of iterations is reached or no further improvements occur. [25][44].

2.3.3 *Principal Component Analysis (PCA)*

PCA is a popular technique used to reduce the dimensionality of datasets with numerical variables. PCA creates new variables from the original p variables. The q latent variables are generated by calculating linear combinations.

The q variables explain the maximum variance in the original data. In this process, the capability of identifying patterns is retained while variables with low variance are not used. Resolving a maximization problem, the vectors can correspond to eigenvectors. Each eigenvector has the value of variance for the corresponding *Principal Component (PC)*. The proportion resulting from the ratio of the sum of the first q eigenvalues divided by the sum of the variances of p original variables is equal to the proportion of the total variance of the original data explained by the same first q PCs [25][45].

2.3.4 *t-Distributed Stochastic Neighbor Embedding (t-SNE)*

t-SNE is a recent powerful technique, that not only captures local patterns from data with a large number of dimensions, but is also able to identify clusters in a global level. *Stochastic Neighbor Embedding (SNE)*, the first version of t-SNE, starts by calculating the conditional probabilities from the Euclidean distances between instances. Each conditional probability represents a similarity between the two datapoints, representing the probability of the first datapoint to choose the second datapoint as neighbor, if neighbors were chosen depending on their probability in the Gaussian curve centered in the first datapoint. The conditional probability of similar datapoints is high, while different datapoints have probability values approximated to zero. The cost function used in SNE and t-SNE minimizes the sum of KullBack-Leibler divergences over the total instances using gradient descent which will be explained in a later section. The quality of the local structure of the data is assessed by the cost function.

t-SNE differs from the SNE in two points. First, the cost function is a symmetric version of the originally used in SNE that uses more efficient and simple gradients. Second, instead of using the Gaussian curve for calculating the similarity among two instances, Student-t distribution is used. The disadvantages of t-SNE consist in not being able to use the algorithm for other use than data visualization. The Student-T distribution cannot be used for dimensions larger than three because of the Student-T heavy tails may not capture the local data structures. The results of t-SNE can get worse if the algorithm is used in largely intrinsic dimensional datasets. An example of a t-SNE visualization is shown in Figure 4 where each different symbol [1].

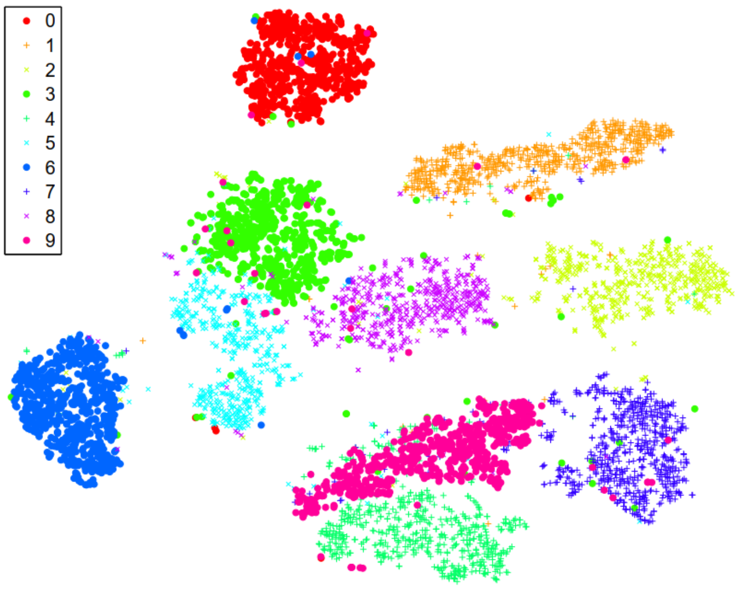


Figure 4.: t-SNE visualization of MNIST dataset. Source: Maaten et al. [1].

2.4 ARTIFICIAL NEURAL NETWORKS (ANN)

2.4.1 *Perceptrons and neurons*

A perceptron is the ancestor model of a neuron. A group of perceptrons or neurons are needed to build an *Artificial Neural Networks (ANN)*. A perceptron can have multiple connections as inputs and outputs as seen in Figure 5.

Each connection has an associated weight or coefficient that modifies the input values. The output z is the result of the sum of the input values from each previous perceptron multiplying by a w coefficient as seen in Equation 15.

$$z = \sum_{i=0}^{m} w_i x_i \tag{15}$$

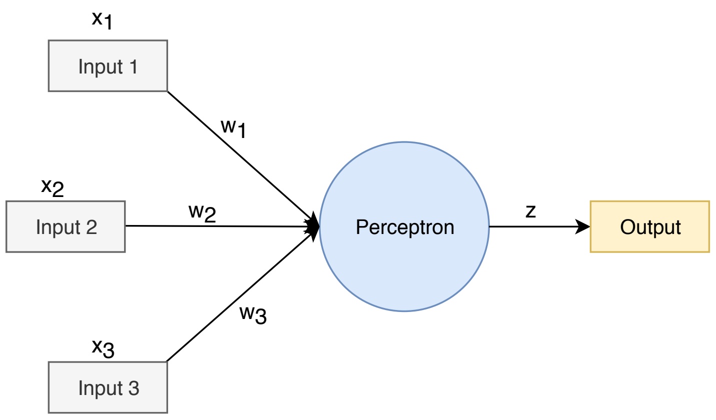


Figure 5.: Schematic representation of a perceptron.

where m corresponds to the number of weights and by association, the number of input connections. The ouput of the first perceptrons is binary (0 or 1). If the value of the network is positive, the perceptron outputs the value 1, otherwise 0. Perceptrons can be used in logic functions as OR, AND and NAND due to the binary ouput. Combinations of perceptrons with simple logic functions as OR and NAND can be used to solve an XOR (OR and NAND) [46].

In ANNs, neurons are the building blocks rather than perceptrons. The network input of a neuron includes a bias value as shown in Equation 16.

$$z = \sum_{i=0}^{m} w_i x_i + b_i \tag{16}$$

The previous equation can be formulated in a vector form as shown in Equation 17.

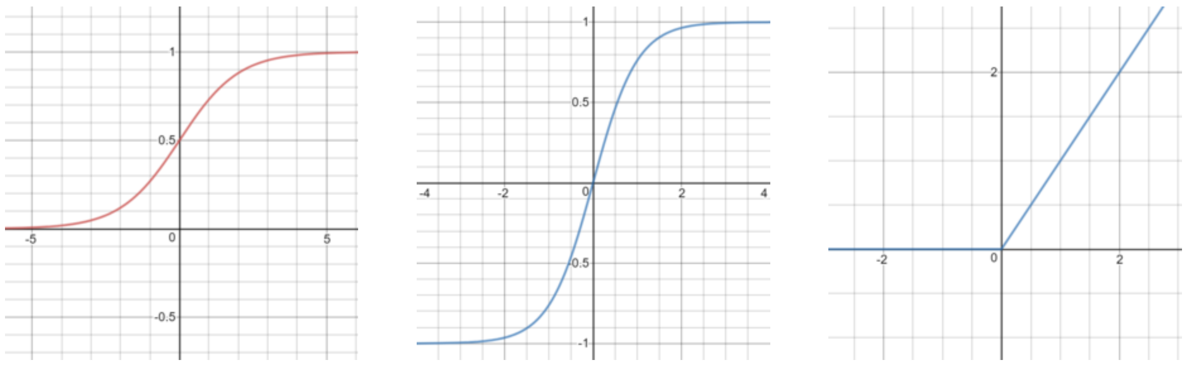
$$z = w^T x + b \tag{17}$$

Another of the major differences between the modern neuron and the first perceptron is the activation function $\phi(z)$. The outputted value of a neuron is not necessarily binary, rather it is produced by $\phi(z)$. Firstly, the neuron calculates the net value z. Afterwards, the z value is run through the activation function producing the neuron output value [46].

The most popular activation functions are *softmax*, *sigmoid*, *heaviside step* (binary), *Hyperbolic Tangent (TANH)* and *Rectified Linear Unit (ReLU)*. Functions are used in different types of problems and possess different ranges. Table 4 contains the ranges and formulas for the most popular neuron activation functions. In deep learning, ReLU is the most used activation function in hidden layers. Sigmoid is popular in a wide range of application from classical machine learning to recent deep learning architectures. In Figure 6 it is shown how sigmoid, TANH and ReLU functions are visualized in plots [46][47].

Table 4.: Most commonly used activation function with respective formulas and ranges

Activation Function	Formula	Range
Softmax	$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$	(0,1)
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	(0,1)
Heavy step	$f(x) = \begin{cases} 0 & for \quad x < 0 \\ 1 & for \quad x \geq 0 \end{cases}$	{0,1}
TANH	$f(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	(-1,1)
ReLU	$f(x) = \begin{cases} 0 & for \quad x < 0 \\ x & for \quad x \geq 0 \end{cases}$	[0,∞)



(a) Sigmoid (b) TANH (c) ReLU

Figure 6.: Graphs representing some of the most popular activation functions

2.4.2 *Feedforward neural networks*

Feedforward neural networks are built by groups of neurons. The classic topology is composed by one input layer, one or more hidden layers and one output layer. The first layer contains one neuron for each input feature. In the last layer or output layer, the number of neurons is equal to the number of possible labels in a classification problem while generally one single neuron is present in models for regression problems. Each layer is fully connected with the posterior layer as shown in Figure 7 [25][47].

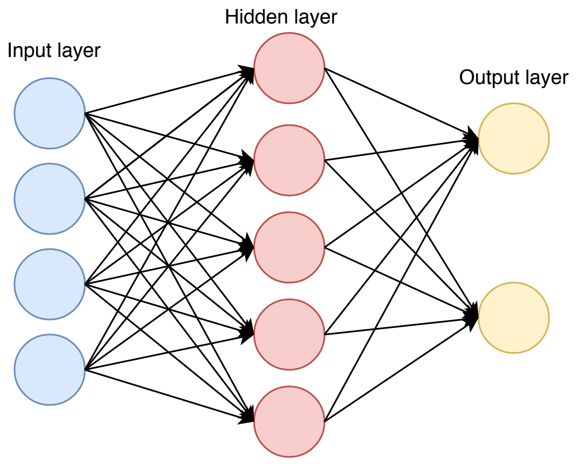


Figure 7.: Schematic illustration of a feedforward neural network

In ANNs, the model is trained by updating the w parameters. The updated values of w are calculated by comparing the error between predicted and real data. The training process will be further explained in a later section. Equation 18 shows how to calculate the output value of a neuron in a specific layer.

$$a_i^l = \phi(\sum_j w_{ij}^l a_j^{l-1} + b_i^l) \tag{18}$$

where a_i^l is the output of the i^{th} neuron from the l^{th} layer. The a_i^l results from applying the network input value of a neuron to $\phi(z)$ activation function. w_{ij}^l represents the value of the weight of the connection from the j^{th} neuron to the i^{th} neuron while a_j^{l-1} represents the output value of the j^{th} neuron in the previous layer. b_i^l is the the value of bias added to the i^{th} neuron from the l^{th} layer. The previous equation can be formulated in the following vectorial form as shown in Equation 19 [25][47].

$$a^l = \phi(w^l a^{l-1} + b^l) \tag{19}$$

In order to obtain an ANN output, one of the previous equations must be applied to every neuron in each layer in a feedforward manner, from the input layer up to the output one.

2.4.3 *Gradient Descent*

The process of model training fundamentally consists in giving slight adjustments to the w weights. In each iteration, coefficients adopt new values after calculating the error between the predicted and real values. One of the most popular approaches is gradient descent. The gradient descent method takes advantage of *differentiable* operations and computes the *gradient* of the cost function. Basically, gradient descent uses the previous assumptions to compute the coefficients in order to minimize the loss function. Applying to an ANN, the goal of gradient descent is to find the combination of w weights that minimizes the value of the error [2]. At each iteration, weights are updated in the opposite direction of the gradient. The iterative method of a *Stochastic Gradient Descen (SGD)* applied to ML follows the steps present in Algorithm 1. While gradient descent uses all the samples to update a parameter, SGD uses one sample or subset of samples (batch) to update a parameter in each iteration [2].

Algorithm 1 Stochastic Gradient Descent (SGD)

```
Initialize the weights $w$
Define a step learning rate
for batch of training samples x and corresponding targets y do
    y_predicted = model(x,y)
    loss_val = loss_function(y,y_predicted)
    gradient = grad_func(cost,w)
    $w$ -= step * gradient
end for
```

The choice of step is essential for determining the minimum of the cost function. If a large value is selected for the step factor, the algorithm can diverge or take random locations without finding the optimal coefficient values. A small step value causes the algorithm to take more iterations, increasing computational time, and the risk of finding a local minimum instead of a global minimum increases considerably. Figure 8 shows how the algorithm can be stuck in a local minimum if the value of the step is small.

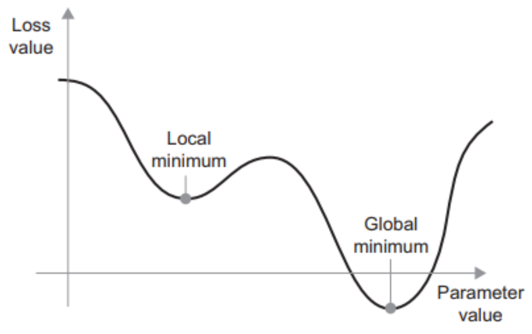


Figure 8.: Example of a the curve of a loss as function of a parameter. Source: Chollet et al. [2].

The algorithm previously shown can be called a *mini-batch* SGD. Other variants of the SGD algorithm can occur, depending on the number of samples used in each iteration. The *true* SGD uses one single sample and label in each iteration. The algorithm is efficient but has high randomness. The *batch* SGD uses the entire batch in the training process updating the weights more accurately but becomes more expensive. The balance in the number of samples used in a SGD is the key to obtain good results without sacrificing accuracy. AdaGrad, RMSProp and other, also known as *optimizers*, use momentum to increase the speed of convergence and decreasing the probability of getting stuck in a local minimum [2].

2.4.4 *Backpropagation Algorithm*

The backpropagation algorithm is used to calculate the gradient of a cost function regarding an ANN using the chain rule of derivatives. A derivative of the network with respect for the input of a given layer can be obtained from working backwards from the derivative of the output. The algorithm can be applied to each layer starting from the output layer of the ANN, to the input layer. After computing the gradients for each layer, the calculus of the derivatives regarding the coefficients of each layer is simple. Starting in the output, the error derivative of an output unit is computed by differentiating the cost function. In the case of MSE, the equation of the error derivative is shown in Equation 20 [48].

$$\frac{\partial E}{\partial y_l} = y_l - t_l \tag{20}$$

In the Equation 20, l refers to the output layer, t is the target values and y are the predicted values. Equation 21 describes how a small change in the value of activation modifies the error.

$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l} \tag{21}$$

The degree of error variation caused by a small change in the activation value is calculated with the Equation 20 equation multiplying by the partial derivative of y_l regarding changes in the value of activation z_l. The next step is to calculate the error derivative of the previous layer k regarding the variation of output y_k. This includes the sum of the error derivatives multiplied by the corresponding weights w_{kl} as shown in Equation 22.

$$\frac{\partial E}{\partial y_k} = \sum_{l\ \varepsilon\ out} w_{kl}\frac{\partial E}{\partial z_l} \tag{22}$$

As shown in the previous equations, the chain rule of derivatives can be applied to ANNs. The chain rule of derivatives allows a small change in x ($\triangle x$) to be transformed into a small change in y by multiplying $\triangle x$ by $\partial y/\partial x$. Further equations can be linked to obtain changes of variables by multiplying partial derivatives by the value of small changes in the variables [48].

2.5 DEEP LEARNING

The previously assessed models can be classified as different forms of ”shallow” learning. The ”shallow” term is used to distinguish the traditional models of ML from deep learning algorithms. The ”deep” in deep learning is based in the depth of the model. While a typical ANN has one hidden layer, a DNN can have multiple hidden layers. Each layer consists in multiple units capable of non-linear processing data transformation and feature extraction. The concept of deep learning is not new, but recently the conditions changed, providing the means needed to take advantage of deep learning. The increase in computational power (usage of *Graphics Processing Unit (GPU)*, new and improved algorithms and the rise o ”Big Data” fulfilled the needs [2][48].

The concept of deep learning is based in the human learning process. First, simple concepts and patterns are learned, then more difficult and abstract concepts. Each hidden layer works similarly to the biological process, learning simple patterns in the first layers and abstract patterns in later layers. Deep learning models have been presenting good results in the areas of speech recognition, image recognition, natural language processing, bioinformatics and others. One table regarding diverse applications of deep learning in bioinformatics will be presented in a later section [2][48].

2.5.1 *Deep Neural Networks (DNN)*

A DNN, also known as ”dense neural networks”, has small changes when comparing to an ANN. The major difference is the number of hidden layers used in DNNs when comparing to ANNs. DNNs are fully connected. As shown in Figure 9, the units in a given layer have connections with previous and next layers. The generic DNNs can provide better results comparing to the ANNs, but the improvements in results have their drawbacks. The training of the DNNs is similar to ANNs, while it needs more computational power and quantity of data than conventional ANNs. Other major problem with using DNNs is the risk of overfitting. As the increased number of hidden layers better fits the data, overfitting is a common issue. Overfitting can be solved by applying regularization methods as L1, L2 among others [2][48].

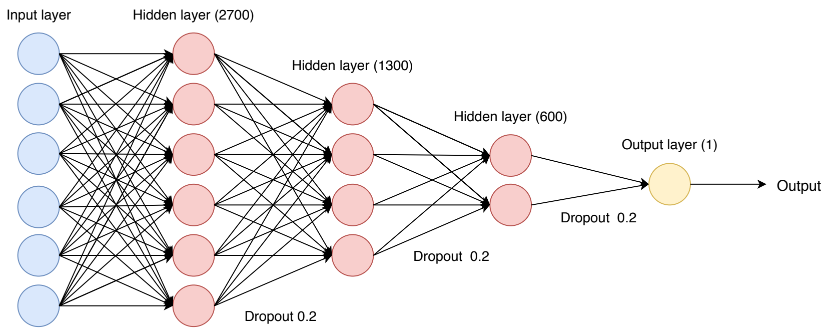


Figure 9.: Representation of binary classification DNN with three hidden layers.

2.5.2 *Convolutional Neural Networks (CNN)*

CNN are mainly used for computer vision problems. The input of a CNN consists in tensors with height, width and number of channels. The number of channels depends on the colors of the images. Grey-scale images only have one channel while "normal" images contain three channels due to *Red Green Blue (RGB)* color model. A CNN contains stacks of alternating convolutional and pooling layers. The number of dimensions of the images tend to shrink while they go through the model. After the alternating stack of convolutional and pooling layers, a layer is used to transform the tensors in vectors of only one dimension. In the end, one or more dense layers (common layers) are used to determine the output of the model.

The main difference from CNNs to other deep learning architectures is the convolution operation. While dense layers detect global patterns, convolutional layers detect local patterns. The convolutional layers can detect the local patterns in any location of a given image and can be considered to follow a spatial hierarchy. The first convolutional layer detects small patterns while the next layer will learn larger patterns based in the captured patterns from the previous layers. The spatial hierarchy allows CNNs to learn abstract and complex patterns that other models cannot learn. In a convolutional layer, patches are generated by sliding a window trough an input feature map. Regarding the selection of patches, the dimensions and the stride parameters, can change significantly the method how patches are captured. A dot product is computed with each patch and a kernel, creating transformed patches. All the transformed patches are grouped together creating an output feature map. The convolutional process can be observed in Figure 10. The maxpooling layers are used to downsample the feature-maps reducing the number of coefficients of feature maps increasing the efficiency. While convolutional operation applies a linear transformation to each patch, maxpooling operation consists in an hardcoded max tensor operation applied only outputting the maximum value. Another goal of maxpooling layers is to induce spatial hierarchies because each convolutional layer will receive a proportionally larger window when compared to the previous layer. Figure 11 represents how a CNN works [2][48].

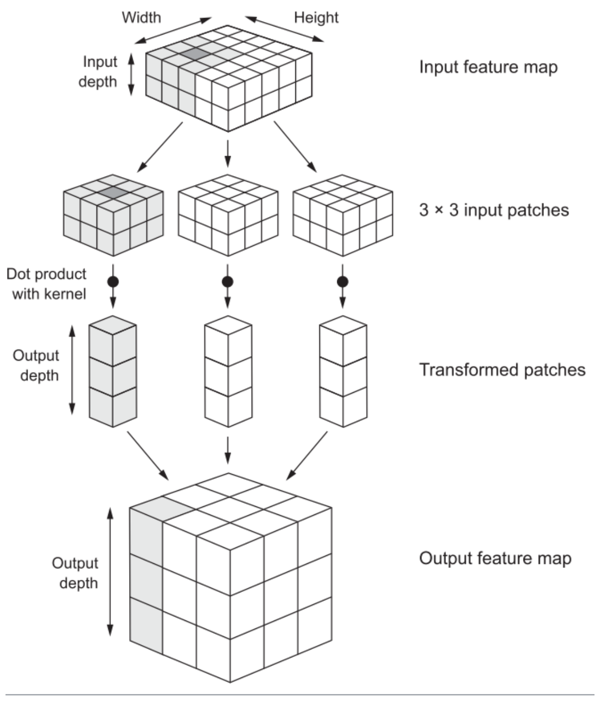


Figure 10.: Representation of convolution operation. Source: Chollet et al. [2].

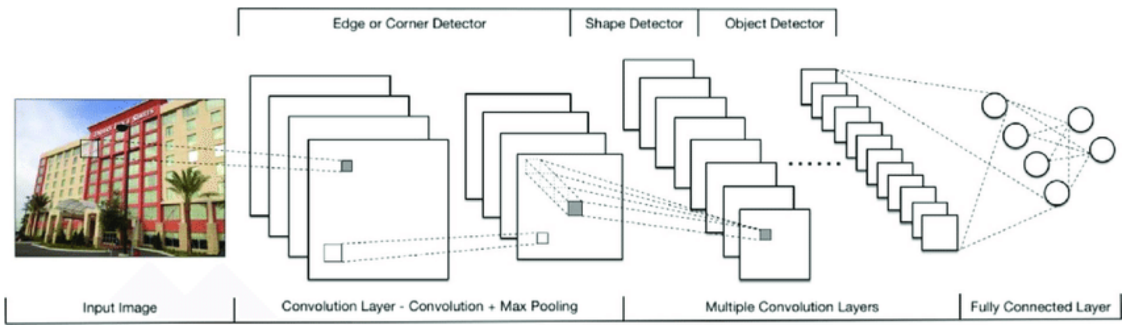


Figure 11.: Representation of a CNN. Source: Ma et al. [3].

2.5.3 *Recurrent Neural Networks (RNN)*

Dense and convolutional layers used in the previous architectures do not have memory. The fact of having no memory increases difficulty for training models from sentences and other inputs where order matters. Biologically, the analysis of a word in a certain sentence is different depending on the previous words in the sentence. RNNs differ from DNNs and

CNNs because RNNs have a memory. RNNs process a sentence by iterating through the sequence and maintaining a *state* with information relative to previous words [2][48].

As the algorithms cannot have raw text as an input, the data must be split in *tokens*. After the *tokenization process*, each token must be associated with a numerical vector so that the algorithm is fed with a sequence of vectors. *Word embedding* is a popular method to develop the association between words and vectors. While one-hot encoding generates high-dimensional, binary and sparse vectors, word embedding generates low-dimensional dense vectors. As a word embedding is learned from data, generally more information can be captured in considerably fewer dimensions. An important characteristic of word embeddings are the geometric relationships between vectors. While each word could have a randomly selected vector as one-hot encoding, in word embedding selection it is not random. In the learning process of the embedding, similar words are assigned with similar vectors. Semantic relationships can be learned, where relationship vectors can be applied to word vectors to obtain new word vectors. One example is applying a relationship vector ″plural″ to the word vectors ″cat″ and ″dog″ to obtain the word vector for ″cats″ and ″dogs″. Each word embedding learned is specially useful when large amounts of data are available. While this is true, it is possible to use already existing world embeddings, also called *pretrained word embeddings*. These embedding are useful when low amounts of data exist, making it impossible to train good quality RNNs [2].

In practice, a RNNs is a network with a loop that uses the previous output in the next iteration, but the state is not preserved between different input sequences or samples as seen in Figure 12. As a state corresponds to an output influenced by the previous outputs, each state has relative information to all the previous segments of the sequences without needing to maintain all the states [2][48].

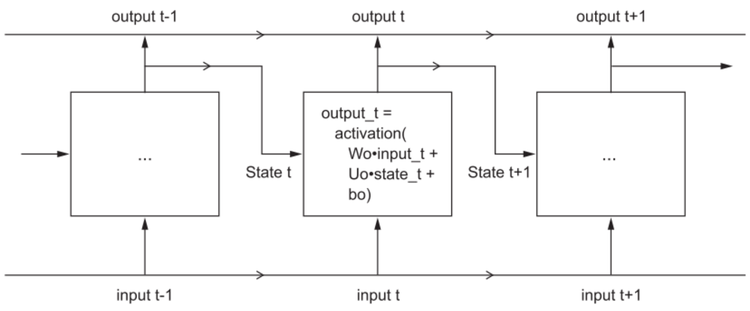


Figure 12.: Representation of a simple RNN. Source: Chollet et al. [2].

The most popular types of layers used in RNNs are the LSTM and *Gated Recurrent Unit (GRU)*. LSTMs are capable of using old information in forward time steps. Contrary to simple recurrent layers that consider the output of the immediately previous time-step, in LSTMs, information relative to any of the previous states can influence the current time-step as can be seen in Figure 13. GRUs layer use the same principle as LSTMs, although

GRU can be considered streamlined to a certain point. GRUs are more computationally efficient but sacrifice some representational power more significant in longer sequences [2].

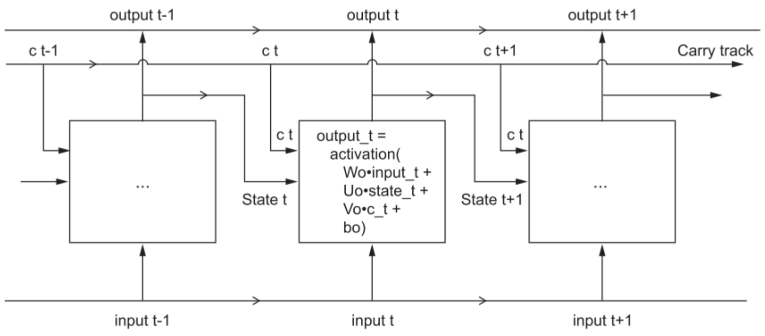


Figure 13.: Representation of a LSTM. Source: Chollet et al. [2].

2.5.4 *Alternative Architectures*

The previous architectures are the most popular among the deep learning community. However, some recent architectures with unique concepts have been developed in recent years. Creating photo realistic images, generating texts and similar achievements could not be possible using the aforementioned networks. Two of the recently popular architectures are Autoencoders and *Generative Adversarial Networks (GAN)*.

Autoencoders (AE)

Autoencoder (AE) can be used in two different roles: dimensionality reduction and to create generative models. The key to AE is the "bottleneck" layer. As seen in Figure 14, in a specific hidden layer, also known as bottleneck, the number of units is generally reduced when compared to all the other layers. The role of the bottleneck layer is to capture the most relevant features in a reduced number of dimensions while being able to reconstruct the original sample from the bottleneck layer. Some variations of the generic AE exist, including *Denoising Autoencoders (DA)*, *Stacked Autoencorders (SAE)* and *Variational Autoencoders (VAE)* [2][17][49].

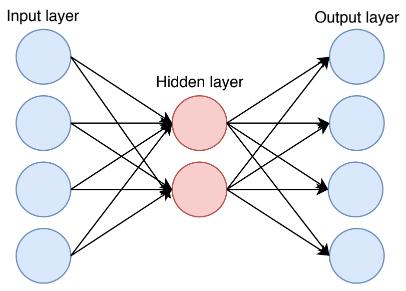


Figure 14.: Illustration of a simple autoencoder

Similarly to the base AE, DAs try to capture features of the input. The difference resides in the ”denoising” concept. Before being used for training the model, noise is introduced into the input sample. The DA must be able to capture the essential features of the input while being able to reconstruct the original sample. The level of noise added to the data is variable, but ideally a large percentage of noise must be added if the dataset is small. DAs generally create more robust models when compared to standard AEs [49][50].

SAE can use most of AEs stacked. Considering an SAE with two hidden layers, each of the layers must be pre-trained. First, the raw input is used to train the first hidden layer. Afterwards, the primary features captured by the first hidden layer are used to train the second hidden layer to learn the secondary features. Having the two hidden layers ready, the classifier can be trained using the secondary features obtained from the second hidden layer. Finally, the model is assembled with input layer, two separate AEs and classifier. Similar to CNNs, the first hidden layer tends to capture simpler patterns, while the later layers learn more complex or abstract features resulting from the composition of patterns from the previous layer [2][17][49].

A classical image AE creates a latent vector space capable of reconstructing the original image. Instead of creating a latent vector space, VAE transform images in parameters (mean and variance) of a statistical distribution. The input image is assumed to be generated from a specific statistical distribution. The mean and variance are the parameters used to randomly sample one data point to be decoded to the original input. The randomness associated with VAEs provides robustness to the model because each point in latent space can be decoded to an output. Great results have been obtained using VAEs to generate realistic photo realistic images as reported by Pu et al.[2][17][51].

Generative Adversarial Networks (GAN)

GAN were presented by Ian Goodfellow in 2014. GANs are capable of creating realistic images by pressuring forged images to be as similar as possible to the original. GANs are composed by two parts: a generator network and a discriminator network. The generator network creates new images from random input vectors while the discriminator networks predicts if an input image is real or forged by the generator network. Contrary to the majority of deep learning architectures, GANs are not optimized to reach a minimum, but to find a balance between the performance of the generator and discriminator networks [2]. The common steps to train a GAN are the following:

1. Create vectors from random points in the latent space;
2. Generate new images using the random vectors as input;
3. Cross real and generated images;
4. Train the discriminator with all the images with the respective labels;
5. Create new vectors from random points in the latent space;
6. Train the model with frozen discriminator, allowing to train the generator to be able to fool the discriminator.

Generators never use the training set, because the information received for training is received by the discriminator. Training a GAN is a difficult process because fine tuning and heuristic tricks are needed to obtain good results. Although GANs have potential to generate high realistic images, as the latent space is not continuous as VAEs, they may not be applied to certain applications.

2.5.5 *Training Algorithms*

Over the past decade, different methods and algorithms have been improved to obtain better results and be more efficient in either "shallow" or deep learning. Algorithms as RMSProp and *Adaptive Moment Estimation (Adam)* are improvements when compared to the original optimizers as gradient descent algorithm. Batch normalization during the training process and pre-training are also popular methods in the current state of the art.

Batches

Gradient descent is slow due to computing the gradient of the cost function for the parameters of the entire dataset. As aforementioned, SGD is more efficient because one parameter is updated for each training sample. Although SGD is faster and better at finding local minimum when compared to the original gradient descent, convergence at the exact minimum is more difficult as the algorithm will keep overshooting. Other advantage of SGD is the possibility of updating our model *online* by removing or adding new entries to the dataset. The mini-batch gradient descent is a good balance between the previous algorithms. Instead of updating one parameter for each training example, one parameter is updated for each mini-batch n training examples. Depending on the quantity of data and the size of the mini-batches, the algorithm can have a good trade-off between efficiency and quality of the results. During the training process, batches can be normalized providing better accuracy and efficiency [2][52].

Pre-training

One of the problems with training ML algorithms is the weight initialization. The starting parameter values are random and can affect the training and performance of the model. One way to avoid the problem is by pre-training the model. The pre-training can be done with a different dataset related to the main dataset. The process can be unsupervised because the objective is not to create a full operational model, but obtaining related initial weights. SAE and Restricted Boltzmann machines are common methods used for pre-training deep learning networks [52][53].

Momentum

The concept of momentum is applied in gradient descent in order to achieve better convergence speeds and prevent getting stuck in local minima. The improvement is applied by using a fraction γ of the update vector from the previous time step to compute the current update vector as seen in the equations 23 and 24.

$$v_t = \gamma v_{t-1} + \eta \bigtriangledown_\theta J(\theta) \tag{23}$$

$$\theta = \theta - v_t \tag{24}$$

where η is the learning rate, $\bigtriangledown_\theta$ is the gradient and $J(\theta)$ is the cost function. The most recent optimizers use the momentum concept in their formulations.

RMSProp

RMSProp is an algorithm with some similarities with gradient descent with momentum. Contrary to gradient descent algorithm, RMSProp restricts vertical oscillations. Therefore, larger learning rates can be used to converge faster. RMSProp adapts each learning rate for each parameter. The computation is similar to gradient descent with momentum with some differences as seen in Equations 25 and 26 [53].

$$v_t = \gamma v_{t-1} + (1 - \gamma) * g_t^2 \tag{25}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} * g_t \tag{26}$$

where g represents the gradient at time t and ϵ corresponds to a small value to ensure the denominator is not zero. The value of γ is set to 0.9 by default and the value of ϵ is generally $1e^{-10}$ [53].

Adaptive Moment Estimation (Adam)

Adam combines the approaches from gradient descent with momentum and RMSProp. For each parameter, Adam not only stores the exponential averages of the gradients m_t but also stores the squares of the exponential averages of the gradients v_t. The m_t and v_t estimates for the first moment (mean) and second moment (uncentered variance) of the gradients, respectively. Equations 27, 28 and 29 are the basis for computing Adam algorithm [53].

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) * g_t \tag{27}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) * g_t^2 \tag{28}$$

$$w_{t+1} = w_t - \eta \frac{m_t}{\sqrt{v_t + \epsilon}} * g_t \tag{29}$$

Generally, the values of β_1 are set to 0.9, β_2 set to 0.999 and ϵ to 10^{-8} [53].

2.5.6 *Overfitting and regularization*

One of the major problems of deep learning is overfitting. As deep learning networks are able to capture a large number of patterns from the data, the occurrence of overfitting is common. Different techniques are used to solve the problem. The most popular include weight regularization, early stopping, dropout and data augmentation for CNNs. Weight regularization methods as L1 and L2 were already assessed in the linear and logistic regression section [2].

Early Stopping

Early stopping can be used in iterative methods based on gradient descent, including SGD, RMSProp and Adam. The goal of early stopping is to stop the algorithm when the model stops improving. Gradient descent based algorithms run through epochs. While a model trained with a small number of epochs has a high risk of underfitting, using a large number of epochs increases the risk of overfitting. Early stopping evaluates the error in a validation set not used for training at the end of each epoch and saves the model if the performance is better when compared to the previous epoch. When the performance of a model stops increasing for a specified number of epochs, the algorithm stops. The used model consists in the model with best results before early stopping the algorithm. As further epochs not only take more time of the algorithm but also causes overfitting, early stopping is a popular technique [2][54].

Dropout

Dropout is one of the most popular techniques used in neural networks for preventing overfitting and improving the robustness of a model. Dropout when applied to a layer, sets the output to zero of a random set of neurons as seen in Figure 15. The *dropout rate*, usually set between 0.2 and 0.5, is the fraction of neurons that are randomly set to zero. While during test no unit is set to zero, the output values of the layer are scaled down by the dropout rate to balance the difference in the number of active units between training and test. The dropout method introduces noise in the output values, making possible to separate relevant patterns from coincidence patterns [2].

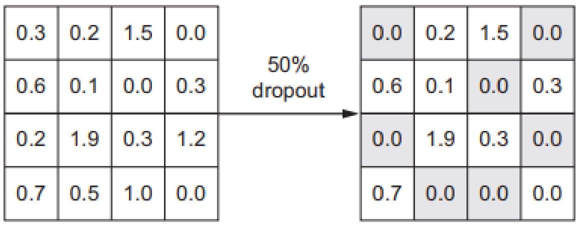


Figure 15.: Example of a dropout with a 0.5 dropout rate. Source: Chollet et al. [2].

2.5.7 *Hyperparameter optimization*

The number of layers, number of units, type of activation function and dropout rates are some of the hyperparameters of a deep learning network. The concept of hyperparameters was created for distinguishing from the training parameters also known as weights. There is no correct method for selecting the best set of hyperparameters. If the best result possible is the goal, arbitrary human choices are not sufficient. Most built models are sub-optimal as hyperparameters were chosen based on intuition and experience. To achieve the model with best performance, hyperparameter optimization should be done automatically [2]. The most popular hyperparameter optimization techniques are grid search and random search. Grid search tests every combination of hyperparameter values, training a model of each possible combination. While grid search is an expensive method because it tests every combination, random search selects n random combinations of hyperparameters, being n a parameter corresponding to the number of iterations of the algorithm. Random search is a stochastic technique, as it may not discover the best hyperparameter combination, but it is more efficient when compared to grid search. The steps of a random search are the following:

1. Random selection of hyperparameters.
2. Build a model using the selected hyperparameters.
3. Train the model and evaluate the performance using validation data.
4. Select a new set of hyperparameters .
5. Repeat.
6. Measure the performance with test data for the best hyperparameters.

The key to obtain the best model efficiently, is using the past validation performances to choose the next group of hyperparamenters. Updating hyperparameters is difficult considering two factors. First, computing the feedback is expensive because it requires fully training a model. Second, the space of hyperparameters generally is discrete. Thus, less efficient gradient-free methods must be used because the hyperparameter space is not differentiable. The field of hyperparameter optimization is recent, but there are some available algorithms as Hyperopt already available on *Keras* library. If building the best model possible is the goal, hyperparameter optimization is essential [2].

2.5.8 *Multi-task Learning (MTL)*

The large quantity of data needed to train a deep learning model is a recurrent problem. A recent technique used to bypass the problem is using related tasks and training the tasks simultaneously, called *Multi-task Learning (MTL)*. MTL is a recent area in ML. The goal of MTL is to improve performance of individual learners by training multiple learners for different tasks, simultaneously. To use MTL in different tasks, they must be related otherwise the technique will decrease the performance of the individual learners. In MLT, two essential points exist. First, the relatedness among tasks. How tasks are somehow related is fundamental for projecting the design for MTL models. The second point is the nature of the tasks. As different types of tasks can be used in MTL models, different settings must be applied.

MTL can be classified in different types depending on the nature of the tasks to be learned, including: unsupervised learning, supervised learning, semi-supervised learning, active learning, reinforced learning and online learning. Supervised, unsupervised and semi-supervised follow the principles addressed in aforementioned sections. MTL active learning uses unlabeled data to help improve the learning of labeled data. Unlabeled entries are selected and they are queried for the potential labels. In MTL reinforcement learning, the learning of the tasks is focused on maximizing the cumulative performance of the tasks. MTL online learning consists in the managing of sequential data for each single task. MTL is an area of ML closely related to transfer learning, that will be further explained in the next section [55].

2.5.9 *Transfer learning*

The goal of transfer learning is to improve classical ML by transfering knowledge learned in separate, but related tasks, to improve the performance of a specific task. Transfer learning can improve learning by three different factors. First, the performance of the model using only the transferred knowledge without further training, is better when compared to the model with random data. Second, the time spent in learning is reduced when the learning process starts from related values instead of random values. Third, the final performance obtained with transfer learning is better comparing to the same training process without transferring knowledge. Multiple methods of transfer exist in transfer learning, including: inductive transfer, bayesian transfer, hierarchical transfer and transfer with missing data or labels [56].

When a transference technique decreases the performance of the model, then it is considered as negative transference. One of the goals of research in transfer learning is to develop techniques to emphasize positive transfer among similar tasks while diminishing transfer learning among unrelated tasks. When knowledge is transferred, a mapping of the features must be done in order to correspond features among tasks. Transfer learning differs from MTL in the type of training. Although the techniques use the knowledge from other tasks in order to improve a task, transfer learning used knowledge from one task while MTL benefits by simultaneously training multiple related tasks [56]. The use of word embeddings in different problems than the original is an example of transfer learning [57]. Another application of transfer learning is in CNNs. It is possible to use a pre-trained CNN model without repeating the training process. The use of pre-trained models can be used as a method for achieving better results or avoiding the training process by simply replacing the last layer in a pre-trained CNN model [58].

2.5.10 *Deep Learning Frameworks and tools*

As deep learning has become more popular in recent years, a variety of frameworks and tools have been developed. From frameworks to automate differentiation to libraries that provide simple methods to create and train deep learning models. The tools to be used in the thesis are *Keras*, *TensorFlow* and *NVIDIA CUDA Deep Neural Network (cuDNN)*.

Keras

Keras is a deep learning framework for Python that provides simple and intuitive tools to design and train different deep learning models. As *Keras* has a MIT permissive license, it can be used educationally or commercially. *Keras* has over 200.000 users ranging from casual users, to researchers and engineers. Large companies as Google, CERN, Netflix and hundreds more use *Keras* in some of their problems. A considerable number of start-ups focused on deep learning use *Keras* as their major framework. In Kaggle, the popular ML online contest, *Keras* has the spotlight as the last winners in deep learning used the library. *Keras* runs over *Theano* or *TensorFlow*, backends capable of tensor manipulation and differentiation. *Keras* allows simple and quick prototyping while running efficiently on *Central Processing Unit (CPU)* or GPU [2][59].

TensorFlow

TensorFlow is a library focused on numerical computation, created as an open source project by Google. The library is based in computation graphs, allowing multiple operations to be applied to a data flow. *TensorFlow* is written with a Python *Application Programming Interface (API)* over a C/C++ engine, providing a really efficient processing. *TensorFlow* is available in a wide variety of platforms including Linux, Mac OS X and recently on Windows. ML models focused on *TensorFlow* can be deployed in a various platforms from Android systems to super computers with powerful GPUs. *TensorFlow* supports the *Compute Unified Device Architecture (CUDA)* platform developed by NVIDIA. One important factor is the data parallelism. As the non-linear operations used in a deep learning model can be computed simultaneously, *TensorFlow* achieves a fast processing speed by using the multiple cores available on GPUs. *TensorFlow* includes a visualization tool named TensorBoard where various parameters and metrics can be visually observed in order to debug and optimize the model. The documentation of *TensorFlow* is extensive, complete and user-friendly [2][59].

NVIDIA CUDA Deep Neural Network library (cuDNN)

cuDNN is a library developed by NVIDA that provides primitive computation of deep learning networks on NVIDIA GPUs. The computation of low-level primitives allows a simple integration of other high-level frameworks without requiring for the users to use any particular framework or data format. cuDNN features high performance implementations of typical operations as convolutions, activations, poolings among others. Furthermore, cuDNN includes auxiliary tensor transformations methods for easier manipulation of four dimensional tensors. *TensorFlow* and similar backend libraries for deep learning, rely on cuDNN run efficiently by using NVDIA GPUs [2][59][60].

3

OMICS DATA AND MACHINE LEARNING METHODS IN CANCER

3.1 INTRODUCTION

More than three billion base pairs exist in the human genome. Around twenty thousand genes are divided in 23 pairs of chromosomes with each chromosome in a pair differing slightly from the other. The *Human Genome Project (HGP)* was a global initiative with the goal of sequencing the entire human genome. The cost of the project is estimated to be around 3 billion US dollars. The projected was lead by the US, but the International Human Genome Project Consortia included research centers all over the world. Although the project was estimated to take 15 years starting in 1990, the final version of the human genome was announced two years before scheduled, in 2003. The project provided and promoted new discoveries in the biomedical research, precision medicine and drug discovery fields. Currently, laboratories provide genome sequencing services under 1000 $ allowing people to know their own genome [61][62].

Oncology is one of areas that took more advantage from the progress made with HGP. Cancer is the disease of the century. Contrary to most diseases, cancer has hundreds of different types and subtypes. Each cancer has specific symptoms, behavior and treatments. A generic cure for cancer does not exist, while some types of cancer can be treated effectively. Mutations in some genes are already considered as biomarkers of cancer while the presence of specific alleles are considered as tumor associated genes because they increase the probability of malignant tumor development. As each patient responds differently to the same treatment, omics data become relevant. Knowing the genotype and gene expression of a patient it is possible to choose the most effective treatment for the patient. This is the goal of precision medicine [63].

Precision medicine can be roughly defined as selecting the most effective treatment for a patient with the lowest secondary effects possible. Precision medicine can be applied to all the pathology areas, but oncology benefits the most. As large numbers of subtypes of cancer exist, multiple genes can be mutated in different groups. Knowing the specific mutated genes in a cancer can be the difference between administrating a non-effective drug with severe secondary effects and administrating an effective drug without adverse effects. Omics data are key to success of precision medicine. Genomics, transcriptomics, proteomics, metabolomics among others, provide the data needed to create databases to individually select the treatment for each patient. Genomics and transcriptomics are the two major omics researched in context of cancer. Having knowledge of the genotype and gene expression of a patient, the most effective treatment can be provided [64].

Omics data not only support the choice of the best drug, but also promote drug discovery. In the following years after the HGP, incredible efforts were made in drug discovery. By studying the genome and gene expression, identifying genes and proteins responsible for a disease is possible. Knowing the target protein, the process of designing or discovering a drug is easier when compared to random testing of compounds [17][65][66].

3.2 Cancer Data

3.2.1 *Omics Data*

After the conclusion of the HGP, the advent of *Next Generation Sequencing (NGS)*. NGS provided cheaper techniques to sequence our genes. With the reduction of sequencing costs, the amount of data regarding the areas of genetics increased substantially. Research centers and companies started to invest more efficiently in the quality and quantity of sqeuenced genomes. The study of multiple genomes became essential for the study of individual variability and a large number of diseases including cancer [67].

Omics data as aforementioned, can be divided a wide range of areas. Genomics, study of genome by DNA analysis, transcriptomics, study of gene expression by mRNA, proteomics, study of peptides and proteins, and metabolomics, study of compounds and metabolites produced by metabolism. The previous omics are more popular but more recent omics fields exist as epigenomics, study of epigenetic modifications and secretomics, study of secreted compounds by the cell. A large number of techniques are used to obtain omics data. In the context of the project, DNA sequencing techniques as *Whole Genome Sequencing (WGS)*, and mRNA quantification techniques as micro-arrays and RNA-Seq are some of the techniques used in genomics and transcriptomics, respectively [68][69].

The research of mutations is essential to understand the source of diseases and how to treat them. Multiple techniques are currently used for detecting mutations. DNA sequencing techniques as WGS and *Whole Exome Sequencing (WES)* coupled with bioinformatics tools are some of the current methods for detecting mutations [67][70]. *Polymerase Chain Reaction (PCR)* and eletrophoresis can be used to detect known mutations in specific genes [71][72]. Single-cell sequencing can not only be used for detecting mutations in DNA but also for sequencing and quantifying RNA similarly to RNA-seq technologies [73].

Transcriptomics has a big role in the cancer research field. The cause is the difference observed in gene expression between normal tissue and cancerous tissue. By analyzing gene expression, not only can it be possible to classify samples, but also to discover which mutations are responsible for the development of a tumor. Microarrays and RNA-Seq are the main techniques used for measuring gene expression by quantifying the absolute or relative levels of mRNA [74].

Microarrays were one of the first gene expression measuring techniques. The quantification occurs by the hybridization between sample cDNA and the DNA strands fixed in the probes. Microarrays are robust, easy to use, cheap and quantify the relative abundance of mRNAs [75]. RNA-Seq is a relatively recent technique that not only measures mRNA abundance but also sequence the transcriptome of the sample. Currently, RNA-Seq is popular because quantifies the absolute abundance of all the mRNAs, while being a relatively cheap technique. Contrary to microarrays, RNA-Seq, does not require prior knowledge of mRNA sequence [76].

3.2.2 *Omics Databases*

As obtaining data became affordable with NGS, large amounts of data were and are being generated. Databases are essential because they serve as repositories for data and allow sharing them with the community. As the data stored in repositories is in the order of petabytes, analyzing the existing data instead of spending resources obtaining new is a reasonable thought. New data is always useful, but as large amounts of data are available in multiple databases, processing existing datasets can lead to new discoveries.

Various omics databases are available to the community. From general databases for genomics and transcriptomics as GenBank, to specific cancer databases as GDC [77] .Table 5 contains information relative to multiple omics databases and their respective type of stored data.

Table 5.: Examples of popular databases regarding cancer or omics in general

Database	Type of Data	Characteristics
GenBank	Genomics	Genetic sequences database of DNA data available to the public.
Gene Expression Omnibus (GEO)	Transcriptomics	International repository that stores and distributes high-throughput functional genomics data.
ArrayExpress	Transcriptomics	Repository of archive functional genomics data from microarray and sequencing platforms. Contains data imported from GEO.
Uniprot	Proteomics	Database containing protein sequence and annotation data from multiple European research centers.
Human Protein Atlas (HPA)	Proteomics	Program aiming to map all human proteins in cells, tissues and organs using various omics technologies.
GDC	Multiple Omics	Unified platform for omics data regarding cancer. Contains data from multiple programs including *The Cancer Genome Atlas (TCGA)*.
The International Cancer Genome Consortium (ICGC)	Multiple Omics	Database from with genomics, epigenomics and transcriptomics data regarding over 20.000 cancer genomes.
ChEMBL	Compounds	Manually curated database of bio-active chemical molecules with proprieties similar to drugs.
PubChem	Compounds	Open database for chemical compounds with data regarding identifiers, properties, safety among others.
Catalogue Of Somatic Mutation In Cancer (COSMIC)	Multiple Omics	Database with manually curated data and Genome-Wide Screen data regarding somatic mutations in cancer.
Cancer Dependency Map (DepMap)	Multiple Omics	The objective of DepMap is identifying genetic and pharmacologic dependencies regarding cancer in order to discover genetic targets and predict drug sensitivity.
Genomics of Drug Sensitivity in Cancer (GDSC)	Genomics	The goal of GDSC is to identify biomarkers that can be used to identify subsets of patients with better responde to specific anti-cancer therapeutics.

3.3 MACHINE LEARNING APPLICATION IN CANCER

With the beginning of "Big Data", all fields of research were influenced, including oncology. First, various articles regarding "shallow" and deep learning applications have been published. Both unsupervised and supervised methods have been used, in either classical machine learning or deep learning. The main problems assessed in the literature consist in susceptibility prediction, diagnosis, recurrence, survivability, cancer subtyping and drug sensitivity effectiveness prediction [78][79][80][81].

3.3.1 *"Shallow" Learning*

Table 6 shows published studies using different approaches of "shallow" learning to solve various cancer problems. The objective of the table is to show how a wide range of problems regarding cancer have been assessed using different types of supervised and unsupervised learning approaches. The most popular models are SVMs and ANNs, with an increasing popularity of ensembles as RFs and Boosting methods in the past recent years.

Table 6.: Articles "shallow" learning applications in various problems in the field of oncology.

Problem Type	**Title**	**Method**	**Ref.**
Subtyping	Molecular classification of cancer types from microarray data using the combination of genetic algorithms and support vector machines	SVM	[82]
	Identification of human triple-negative breast cancer subtypes and preclinical models for selection of targeted therapies	K-means and consesus clustering	[83]
Susceptibility	Breast cancer risk estimation with artificial neural networks revisited	ANN	[84]
	Screening test data analysis for liver disease prediction model using growth curve	ANN	[85]
Survivability	Survival model in oral squamous cell carcinoma based on clinicopathological parameters, molecular markers and support vector machines	SVM	[86]
	Predicting breast cancer survivability: a comparison of three data mining methods	DT	[87]
Recurrence	Development of Novel Breast Cancer Recurrence Prediction Model Using Support Vector Machine	SVM	[88]
	Using Three Machine Learning Techniques for Predicting Breast Cancer Recurrence	SVM	[89]
Drug Sensitivity	The Cancer Cell Line Encyclopedia enables predictive modelling of anticancer drug sensitivity	Bayesian network	[90]
	Machine Learning Prediction of Cancer Cell Sensitivity to Drugs Based on Genomic and Chemical Properties	ANN and RF	[91]

3.3.2 *Deep Learning*

As the popularity of deep learning only increased in the past decade, the number of applications is still reduced. The main problems assessed by the community consist in medical imaging, biological sequences and protein structures [17][92]. In medical imaging, the most popular architecture is the CNN. Organ segmentation[93], histology analysis [94], time-series analysis [95] and abnormality detection [96] are some of the problems regarding medical imaging. In the field of protein structures, the focus is to predict protein structures [97][98]. Deep learning plays a major role in current papers in the area of genomic sequencing and gene expression analysis. Some of the problems assessed are gene expression inference [99], splicing patterns prediction [100], predicting miRNA targets [101] and predicting effects of gene variants [102]. In Table 7 contains publications referring to applications of deep learning in general bioinformatics and cancer-specific problems.

Table 7.: Studies with distinct types of deep learning applications in bioinformatics.

Problem Type	**Title**	**Architecture**	**Ref.**
Organ segmentation	Brain Tumor Segmentation Using Convolutional Neural Networks in MRI Images	CNN	[93]
Histology	Multiple clustered instance learning for histopathology cancer image classification, segmentation and clustering	CNN	[94]
Time-series analysis	Recurrent neural network based prediction of epileptic seizures in intra- and extracranial EEG	RNN	[95]
Abnormality detection	Large scale deep learning for computer aided detection of mammographic lesions	CNN	[96]
Protein structure prediction	Malphite: A convolutional neural network and ensemble learning based protein secondary structure predictor	CNN	[97]
Gene expression inference	Gene expression inference with deep learning	DNN	[99]
Splicing patterns prediction	Deep learning of the tissue-regulated splicing code	DNN	[100]
miRNA targets prediction	deepMiRGene: Deep Neural Network based Precursor microRNA Prediction	RNN	[101]
Effects of gene variants prediction	DANN: a deep learning approach for annotating the pathogenicity of genetic variants	RNN	[102]
Tumor drug response	Predicting drug response of tumors from integrated genomic profiles by deep neural networks	DNN,AE	[65]
Cancer prediction	A deep learning-based multi-model ensemble method for cancer prediction	DNN	[103]
Prognosis prediction	A multimodal deep neural network for human breast cancer prognosis prediction by integrating multi-dimensional data	Multi-modal DNN	[15]
Pattern extraction	Unsupervised feature construction and knowledge extraction from genome-wide assays of breast cancer with denoising autoencoders.	DA	[50]

4 DEVELOPMENT

In the current chapter, we will explain the project pipeline and details regarding the developed methods. The global goal of the framework is to allow users to generate supervised models ("shallow" or deep) to determine diagnosis or prognosis of cancers. The programming language used to develop was Python 3. The choice of the language rests in the available libraries for implementing ML related software. The developed methods are available in the github repository `https://github.com/omsoares/DeepLearning_Framework`.

The developed framework should be easy to use, flexible and able to assess different types of models with its different modules. The framework should be divided in separate modules capable of working together in a pipeline or working individually for specific tasks. The framework should allow users to load gene expression and patient/sample clinical data files. It should be capable of treating the data, applying different normalization, using feature selection techniques and splitting the data in train/test datasets. The pre-processed datasets should be used to generate supervised ML models, including traditional and deep learning models. One of the modules should be implemented to generate and train multi-tasking deep learning models capable of predicting more than one clinical data label simultaneously. Each type of ML focused modules should also be capable of performing hyperparameter optimization to obtain better results.

4.1 SOFTWARE

The framework's architecture is divided into 4 main segments as seen in Figure 16:

- tPreprocessing: pre-processing data;
- `Shallow`: supervised "shallow" or traditional learning pipelines;
- `DNN`: supervised DNN pipelines;
- `Multi-task DNN`: supervised multi-task DNN pipeline.

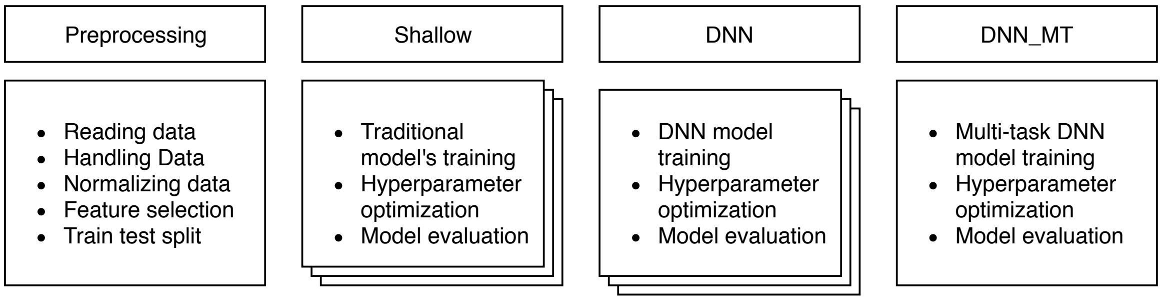


Figure 16.: Simple representation of the developed modules.

The `Preprocessing` module allows reading and pre-processing the data before using it for generating ML models. The module is capable of normalizing and transforming data as well as performing feature selection and selecting the label or labels to be predicted from the clinical data.

The modules of the `Shallow` category are used to develop supervised shallow ML models while `DNN` modules are used to develop supervised deep learning models. In shallow modules, multiple types of classifiers or regressors are trained to obtain the best performance for each type of model, using hyperparameter optimization. In `DNN` modules, different types of hyperparameters are tested. `DNN_MT` differs from the previous modules by being able to generate multi-task multi-output deep learning models capable of predicting more than one label in a single model. The general pipeline for the developed methods is presented in Figure 17.

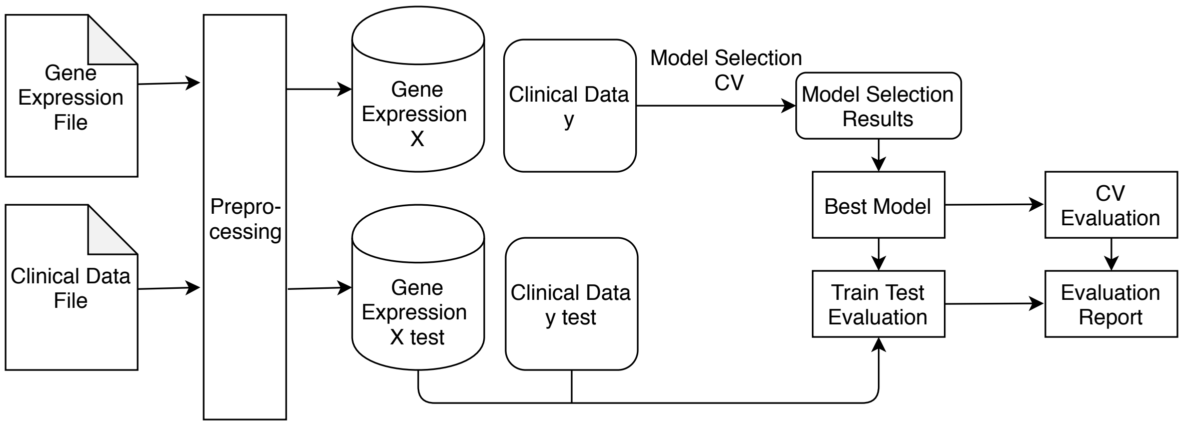


Figure 17.: Simple representation of the developed supervised learning pipeline.

4.1.1 *Preprocessing modules*

The Python module `Preprocessing` is responsible for preprocessing data to be later used in ML experiments. The class is capable of:

- Transposing and selecting the intended labels;
- Handling missing data;
- Normalizing data;
- Applying feature selection methods;
- Generating and saving train and test datasets.

The `Preprocessing` class takes an expression dataset and a clinical data dataset as shown in Figure 18. The class has a method for loading the datasets that has as arguments: the integer index of the column where expression values start in the expression dataset, the name of the column with the names/IDs of the genes, a list with the names of the columns to be used as labels and the name of the column that contains the patient IDs. The files are read using *Pandas CSV reader*, but text files can also be used as input. During the loading method, missing data and rows where the label value in the clinical data dataset is empty are dropped. The two generated *Pandas* dataframes have the same row indexation corresponding to patient or sample IDs. While one the *Pandas* dataframes contain patient's gene expression data, the other contain metadata from the patient. A *Pandas* dataframe is a two dimensional data structure similar to a table. The gene expression dataframe has an expression value for a specific gene in each column with each row being from a different patient. The other dataframe contains metadata regarding the patient in each column with each row corresponding to a single patient. Figure 18 contains the pipeline that can be performed using the `Preprocessing` module from the starting datasets to the ML ready datasets with data handling, normalization, feature selection and dataset partitioning.

The dataframes can contain all entries or be divided in train and test dataframes according to the user. During the preprocessing, normalization and feature selection techniques can be applied. Normalization transforms data from different features into the same range of values and feature selection reduces the amount of features used in model training as previously mentioned in the supervised ML section 2.1.1. The normalization techniques available are *StandardScaler* and *MinMaxScaler* methods from the *Scikit-learn* package. The *StandardScaler* method transforms the values of each feature into values with a mean of 0 and standard deviation of 1. The transformation is performed by subtracting each value by the mean of the values and dividing by the standard deviation. The *MinMax* method transforms the values into a range between 0 and 1 or -1 and 1 if negative values exit [28].

As feature selection is an essential step in pre-processing as mentioned in section 2.2.6, some feature selection techniques available in the framework are the mean absolute deviation filter, variance threshold and *SelectKBest* methods from the *Scikit-learn* package. The mean absolute deviation filter selects a given number of features with higher values of mean absolute deviation. Mean absolute deviation is calculated by the sum of absolute deviation of each value to the mean, dividing by the number of values. The variance threshold is a filter method which selects the features with variances above a specified threshold. The *SelectKBest* method is a filter that selects the K features with higher scores. The metric used to calculate the scores varies, but in the case of the framework, the method computes the F value of an *Analysis of Variance (ANOVA)* test [28]. The `Preprocessing` class is a complementary module that can be used to apply a simple data pre-processing prior to ML usage. Being complementary, the user is not required to use the class before applying to the ML modules provided that the data is in the same format.

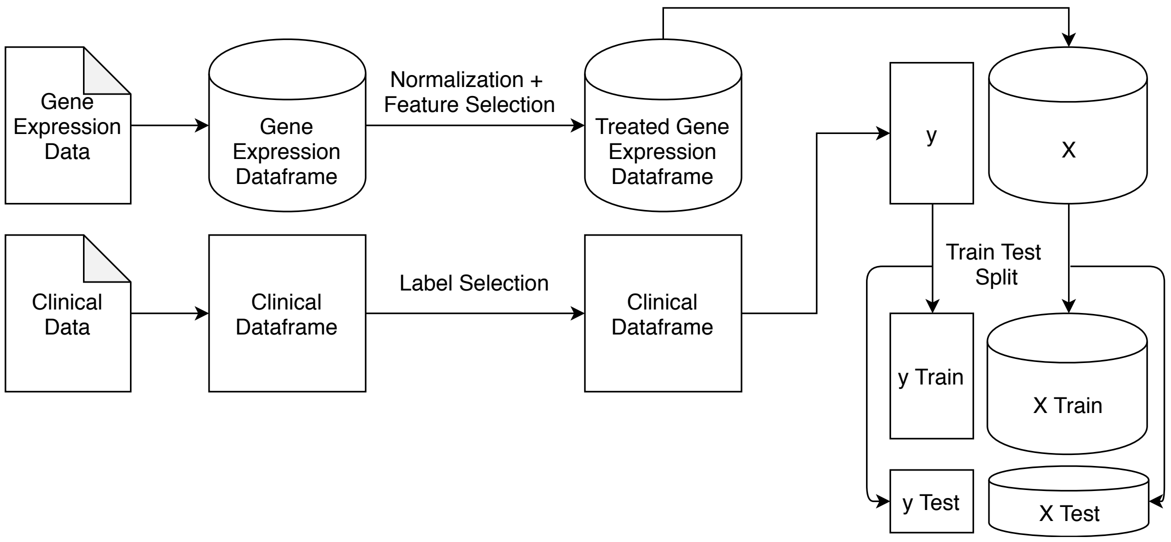


Figure 18.: Representation of Preprocessing class workflow.

4.1.2 *Shallow machine learning modules*

Generating "shallow" or traditional ML models can be useful depending on the available data. The Python classes used to generate multiple traditional models are `Shallow_bin`, `Shallow_multi` and `Shallow_reg` that can be used to create shallow learning models for binary classification, multi-class classification and regression experiments, respectively. All the different model types used in each of the `Shallow` classes are from the *Scikit-learn* package. Using the models from *Scikit-learn* allows for easy use of ML models and related methods. As the structure of the models is similar, the implementation of new types of ML models in the framework is easy. As shown in Figure 19, the workflow of the `Shallow` classes can be summarized in the following points:

- Model selection: Grid search is used for finding the best hyperparameters for different types of "shallow" learning models by training a model with each possible combination of hyperparameters.

- Best model: For each type of "shallow" learning classifier or regressor, the model with the best results is selected.
- Model evaluation: The best models are evaluated and the results are saved in a report file.

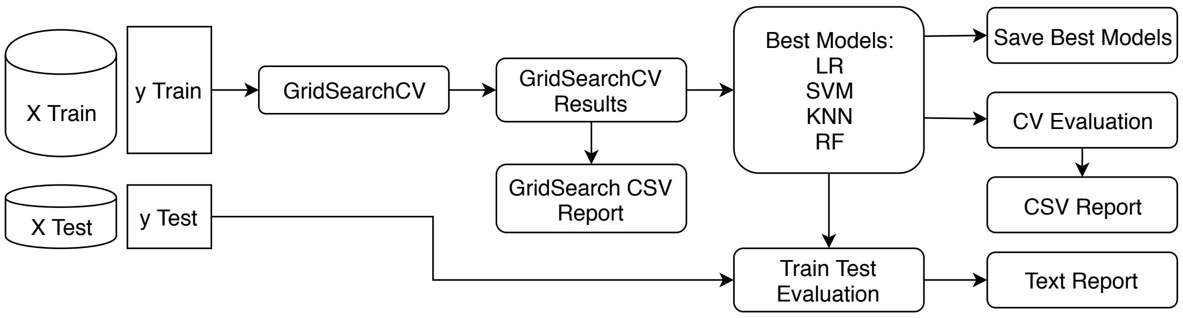


Figure 19.: Representation of Shallow classes workflow.

The types of ML algorithms available are: KNN, RF, LR and SVM for the classification classes. In Shallow_reg, KNN, SVM and RF are available. The description of the available ML models was presented in section 2.2.

`Shallow` classes constructors take X and y dataframes or the same data divided in train/test datasets and the number of folds to be used in cross-validation. As two or four matrices can be provided to the class constructor, two types of evaluation can be applied. Cross-validation is used for model evaluation if X and y matrices are provided while hold out evaluation is executed if the input contains train and test matrices. If the experiment is considered to be split in train and test, the test matrices are only used at the evaluation of the model with best results after selecting model selection.

The model selection is performed by using *GridSearchCV* and different ML classifiers and regressors available in the *Scikit-learn* library. Table 8 and Table 9 contain the hyper-parameters used in the *GridSearchCV* for traditional classification and regression models, respectively. In *GridSearchCV*, multiple models are trained with every parameter combination possible as mentioned in section 2.5.7. The model with best the performance is selected based on the cross validation results. MCC, accuracy and R^2 are the metrics used to evaluate model performance during the model selection for binary classification, multi class classification and regression problems, respectively. After model selection, a *Comma Separated Values (CSV)* report file is generated with the results of the model selection,which includes results from multiple metrics.

The best performing classification models are evaluated using different metrics (precision, accuracy, recall, log_loss, AUC, F1 score and MCC). For the multi-class classification experiments, log_loss and AUC are not calculated. Furthermore, in regression experiments, the evaluated metrics are R^2, MAE and MSE as the metrics used for classification cannot be applied to continuous variables. More detailed information regarding metrics is available in section 2.1.1. The scores of the metrics and the parameters of the model with best results for each type of classifier or regressor are saved in a report file. The format of the report file differs depending on the used evaluation method, CSV and text files for cross-validation and text file for hold out evaluation. The models with best performance can be saved in the pickle (binary) format to be used in further experiments.

Table 8.: Hyperparameters used in *GridSearchCV* for traditional classification models.

Model	**Hyperparameters**
KNN	n_neighbors
RF	n_estimators
LR	C penalty
SVM	C kernel gamma

Table 9.: Hyperparameters used in *GridSearchCV* for traditional regression models.

Model	**Hyperparameters**
SVM	C kernel gamma
RF	n_estimators
EN	fit_intercept positive selection
KNN	n_neighbours

4.1.3 *DNN*

The modules mentioned previously are used to generate traditional ML models but are not able to generate deep learning models. With the objective of generating deep learning models three `DNN` classes are available: `DNN_bin`, `DNN_multi` and `DNN_reg`. While the first two can be used for binary and multi-class classification, the last is used for regression. The class constructor takes the same X and y matrices and number of folds in cross-validation used in the `Shallow` classes with the addition of a batch of hyperparameters. The DNN architectures in the `DNN` modules were implemented using the *Sequential* model from *Keras* with *TensorFlow* as backend. The hyperparameter tuning process is performed using random selections of hyperparameters from a batch with lists of possible hyperparameter values. The number of iterations of the tuning process and the hyperparameters are provided by the user. The input hyperparameters must contain: the hyperparameters shown in Table 10.

Table 10.: Hyperparameters used in the random hyperparameter optimization for DNNs.

Hyperparameters
dropout
optimization
learning_rate
batch_size
nb_epoch
units_in_hidden_layers
units_in_input_layer
early_stopping
batch_normalization
patience

The general architecture of the generated DNN models contains one input layer, one or more hidden layers (Dense) and an output layer. One hidden layer is added for each element of the list with the number of units for each hidden layer. For example, if units_in_hidden_layers = [5000, 2500, 1000], three hidden layers will be added with 5000, 2500 and 1000 units, respectively. After each hidden layer, two layers are added: a batch normalization layer (optional) and a dropout layer. The generic architecture of the DNN is shown in Figure 20. The number of units in the output layer is selected according to the type of problem. While a single unit is used in the output layer of binary classification and regression problems, the number of units in the multi-class classification problems corresponds to the number of possible classes of the label to be predicted. The activation function of the hidden layers is ReLU and the activation function of the output layers are sigmoid, softmax and linear for binary classification, multi-class classification and regression, respectively. The loss function varies depending on the type of problem: binary cross-entropy for binary classification, sparse categorical cross-entropy for multi-class classification and mean squared error for regression.

The results of the hyperparameter tuning are saved in a CSV file. The model with best performance is fit and evaluated. The MCC, accuracy, R^2 and loss are saved in a plot, as well as a report with the evaluated metrics. The user can define the name of the directory where the reports are stored and the file name of the reports. The metrics used for evaluating the set of hyperparameters with best results are the same used in the Shallow classes for each type of problem. The model can be saved to be used in future experiments. The architecture

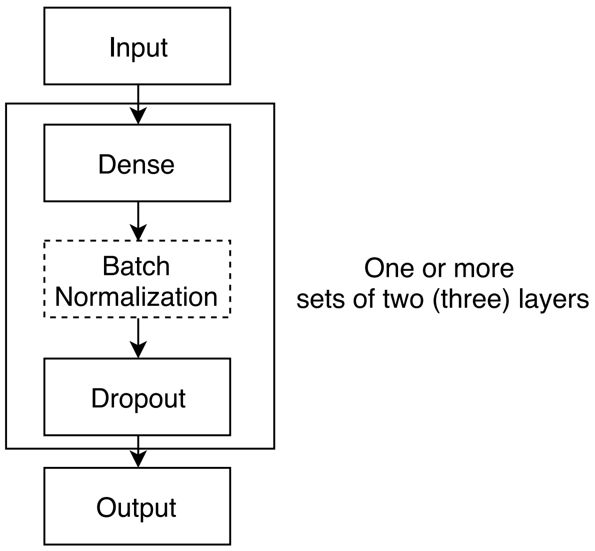


Figure 20.: Schematic representation of the general DNN architecture. The layers inside the rectangle are repeated for each hidden layer added.

of the model is saved in a *JavaScript Object Notation (JSON)* file while the network weights are saved in a separate *Hierarchical Data Format (HDF5)* file. The model is stored in two different files because it is more versatile. The JSON file with the architecture can be altered manually if the user wants. The HDF5 file with the model weights can be loaded in different architectures for fine-tuning or transfer learning.

4.1.4 *Multi-task DNN*

The Python class `DNN_MT` performs MTL with more than one output. The constructor takes the same arguments of the `DNN` classes, with the addition of a list of labels to be predicted and a list with types of problems associated with each label (binary, multi-class or regression). The `DNN_MT` class uses the *Keras Functional* API, while the `DNN` classes use *Keras Sequential* model. The choice to use the *Functional* API was made considering the fact that the *Keras Sequential* model cannot be used to create multi input or multi output DNN architectures. One output layer is inserted for each label with the appropriate metric, output activation function and number of units. Loss weights can be inserted to change the contribution of each output to the loss value during model fitting. Figure 21 is a simple representation of the type of architecture generated by the `DNN_MT` module.

The model selection is similar to both `Shallow` and `DNN` classes but the criterion for selecting the best performing model is different from the previous modules. Having more than one output, we cannot select the best model based in a single metric value as at least two outputs are predicted for each inserted input. The performance of the models is mea-

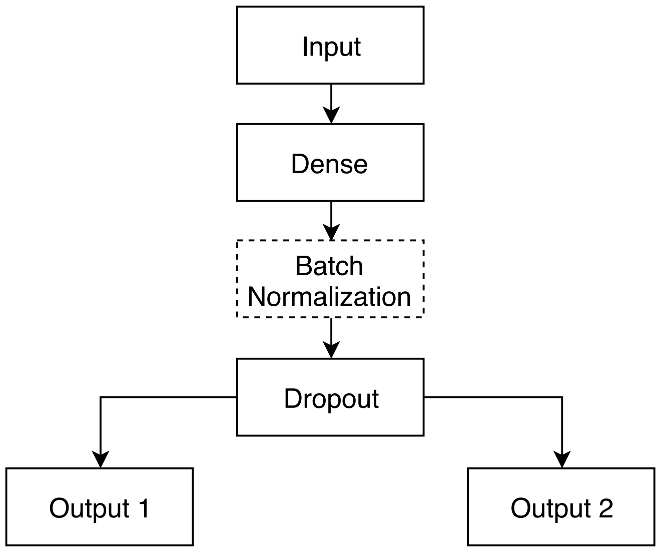


Figure 21.: Schematic representation of the general DNN_MT architecture. Output 1 and Output 2 represents separate output layers for two different labels.

sured by calculating the average of the metrics values for each output. The metrics used for evaluating the performance during the model selection for each output individually were accuracy for binary and multi-class classification problems and R^2 for regression problems.

The model with the highest average of metric scores is selected for further evaluation as it possesses the best overall performance of the tested models. The model can be evaluated either by cross-validation with a CSV and text report, or holdout evaluation if the data is partitioned in train/test data producing a text report. In the evaluation of the selected model, the calculated metrics are the same used in the single label predicting DNN modules.

5

FIRST CASE STUDY

5.1 METABRIC: DESCRIPTION OF THE DATASET

Breast cancer is the second most common cancer worldwide the fifth cancer with overall higher death count and the leading cause of cancer death in women. One of the main problems of breast cancer is the heterogeneity of the disease. HER-2, ER, PR are relevant receptors to diagnosis and treatment of different subtypes of breast cancer. HER-2 corresponds to a receptor responsible for epidermal growth, while ER and PR are estrogen and progesterone receptors, respectively. Depending on the presence of receptors in tumour cells, the appropriate treatment must be assessed. Triple-negative breast cancer is more challenging because the three types of receptors are not present and tumour cells do not respond to most of the hormone therapies, resulting in a aggressive type of cancer with a harsher treatment with chemotherapy, radiotherapy and surgery combination [104]. The NPI score is a tool used in the prognostic of primary breast cancer. The NPI uses the number of lymph nodes affected, tumor grade and tumor size to predict the percentage of survival in the ten following years. Depending on the interval of values where it rests, the survival chance ranges from 96% to 44% for cancer specific survival and ranges from 88% to 42% for death caused by other causes [105].

The dataset used as case study was obtained from the cBioportal [106]. METABRIC is a project between Canada and UK with the goal of obtaining subcategories of breast cancer based on molecular signatures obtained from gene expression, *Copy Number Alteration (CNA)* and long-term clinical data. The collected data is composed from over 2,000 primary fresh frozen breast cancer samples [107][108]. The METABRIC data available on cBioportal comprises gene expression, CNA, mutation data and clinical files.

To demonstrate the developed methods, predictive models taking as input the gene expression data for three different targets were generated using the METABRIC dataset. The targets are ER, THREEGENE and NPI. ER is a binary variable for the presence or absence of estrogen receptors. THREEGENE is a multi-class variable where each class takes into account the presence or absence of ER, HER-2 and PR. NPI is a continuous variable that is used in breast cancer prognosis as already mentioned. Multiple shallow models and deep learning models were trained for each of the targets. Furthermore, a deep learning multi-tasking model was trained to predict ER, HER-2 and PR simultaneously. The multi-tasking model for the three receptors could be useful if it has better performance when compared to the simple DNN for predicting THREEGENE.

5.1.1 *Expression Dataset*

The gene expression data was measured using the Illumina HT-12 v3 microarray platform. The results were pre-processed and normalized. The raw data was obtained using a script in R to process each chip. The raw data was pre-processed obtaining logarithmic intensities, number of entries and standard errors. The resulting intensity matrices were concatenated into a single matrix and pre-processed. The pre-processing included exclusion of arrays with failed hybridization and arrays that failed in the multivariate outlier testing procedure available in the Bioconductor package. ER-positive and ER-negative samples were separated, normalized and averaged separately to obtain the target distribution. The probes belonging to the target distribution were quantile normalized while the other probes were interpolated using weighted normalized intensities of the probes within the target distribution with highest similarity before normalization [107]. The previous processing of the dataset was not part of the thesis work as the dataset available was already processed.

5.1.2 *Clinical dataset*

The dataset with clinical data is the result of merging two separate datasets removing the duplicate columns. The original datasets contain clinical data from samples and patients. The tables 11 and 12 contain all the available features in the metadata for samples and patients, respectively. The number of patients and samples is the same as each sample is provided for a single patient. The dataset contains data for 2508 samples/patients but a significant portion has incomplete data. Numerical and categorical data are available in the dataset. Age (discrete numerical variable), tumor size (continuous numerical variable), ER status (binary categorical variable) and tumor grade (multi-class categorical variable) are some examples of the data in the clinical dataset. Virtually any column can be used as target for predictive models.

Table 11.: Clinical data available in the samples file.

Clinical data	Description
PATIENT_ID	Patient identifier
SAMPLE_ID	Sample identifier
CANCER_TYPE	Cancer type
CANCER_TYPE_DETAILED	Specific type of cancer
ER_STATUS	Expression of ER
HER2_STATUS	Amplification of HER-2
GRADE	Grade of the cancer
ONCOTREE_CODE	OncoTree code of the cancer
PR_STATUS	Expression of PR
SAMPLE_TYPE	Type of sample
TUMOR_SIZE	Size of the tumor in millimiters
TUMOR_STAGE	Tumor stage

Table 12.: Clinical data available in the patients file.

Clinical Data	Description
PATIENT_ID	Patient Identifier
LYMPH_NODES_EXAMINED_POSITIVE	Number of lymphnodes positive
NPI	Nottingham prognostic index
CELLULARITY	Tumor cell content
CHEMOTHERAPY	Administration of chemotherapy
ER_IHC	Immunohistochemistry test for ER
HER2_SNP6	HER-2 status measured by SNP-6
HORMONE_THERAPY	Administration of hormone therapy
INFERRED_MENOPAUSAL_STATE	Inferred menopausal status based on age
INTCLUST	IntClust cancer subtype
AGE_AT_DIAGNOSIS	Patient age when diagnosed
OS_MONTHS	Number of months survived
OS_STATUS	Alive or deceased
CLAUDIN_SUBTYPE	Claudin subtype of cancer
THREEGENE	3 gene classifier subtype
VITAL_STATUS	Cause of death in case of dead
LATERALITY	Tumor location
RADIO_THERAPY	Administration of radio therapy
HISTOLOGICAL_SUBTYPE	Subtype based on tumor histology
BREAST_SURGERY	Type of surgery

5.2 EXPERIMENTAL SETUP

5.2.1 *Pre-processing*

The pre-processing of the two datasets was similar among different experiments. The name of the column with patient IDs, column or columns to be used as labels and the name of the column with gene IDs were provided in each case to select the appropriate data. Both datasets have the same indexation and rows where the labels missing were excluded from the expression and clinical datasets. For ER and NPI, the number of remaining rows was 1904 while the number of remaining rows for the THREEGENE label was 1700. As all labels must be available to generate predictive models, in the multi-tasking model with ER and

THREEGENE, the datasets contained 1700 rows. After selecting the intended columns and rows, the labels with categorical values were encoded into integers.

As the gene expression data was already normalized, no additional normalization method was used. Feature selection was applied to select the best 5000 features. The choice to apply the feature selection to the traditional ML models was to remove features that would not improve the model predictive performance. The feature selection of 5000 features was also applied for the deep learning models because the full dataset contained 17530 features. When the whole dataset was used in the deep learning experiments, a memory error would appear. Although some methods were implemented to solve the problem, it could not be fixed. The 5000 features with highest value of mean absolute deviation were selected to the final dataframe. Mean absolute deviation was used as criterion for feature selection because the higher the difference of values within a feature, the higher the probability of being useful to model training. The pre-processed gene expression and clinical data datasets gave origin to X and y train and test matrices.

5.2.2 *Shallow learning*

To predict each label individually, two types of ML experiments were executed. One method used the `Shallow` modules to create multiple shallow learning models applying grid search for each type of model. The number of folds used in the *GridSearchCV* was five. The types of models and hyperparameters tested are presented in Table 13 and Table 14 for classification and regression problems, respectively. The model with best results found in *GridSearchCV* was then evaluated using the test dataset and the results were saved in both text reports.

Table 13.: Hyperparameters used in *GridSearchCV* for traditional classification models in METABRIC case study.

Model	**Hyperparameters**
KNN	n_neighbors: [1, 3, 5, 7]
RF	n_estimators: [10, 50, 100, 200, 500]
LR	C: [0.001, 0.01, 0.1, 1, 10, 100, 1000] penalty: [l1, l2]
SVM	C: [0.001, 0.01, 0.1, 1, 10, 100, 1000] kernel = [linear, rfb] gamma: [0.001, 0.01, 0.1, 1, 10, 100, 1000]

Table 14.: Hyperparameters used in *GridSearchCV* for traditional regression models in METABRIC case study.

Model	Hyperparameters
SVM	C: [0.001, 0.01, 0.1, 1, 10, 100, 1000]
	kernel: [linear, rbf]
	gamma: [0.001, 0.01, 0.1, 1, 10, 100, 1000]
RF	n_estimators: [10, 50, 100, 200, 500]
EN	fit_intercept: [True, False]
	positive: [True, False]
	gamma: [cyclic,random]
KNN	n_neighbours: [1, 3, 5, 7]

5.2.3 *Deep Learning*

In addition to shallow models, an experiment using DNNs was executed for predicting each label. Using the `DNN` modules, multiple models were generated for each label using a random selection of hyperparameters from a set. The set of hyperparameters provided as input is presented in the Table 15. As the `DNN` modules do not perform grid search, the number of models trained in the experiment do not cover all the possible combinations of hyperparameters. The selected number of iterations was fifty as the tuning process should provide good results. Although a higher number of iterations could provide better results, the computational cost could be too expensive for the available resources. At each iteration, a five-fold cross-validation was performed and the results stored until the end of tuning process. The models with best results were evaluated using evaluated using the test dataset and the results stores in text reports.

5.2.4 *Multi-tasking*

A single experiment was performed using the `DNN_MT` module. The goal of the experiment was to predict ER, HER-2 and PR in a single ML model. In addition to the batch of hyperparameters, X and y matrices, a list with the labels ER, HER-2, PR and a list representing binary and multi-class were provided as input. The number of iterations, the number of folds in cross-validation and the hyperparameters were the same used in the single label predicting DNNs. The model with best average results was further evaluated with the test dataset and reports were generated with multiple metrics values for each label.

Table 15.: Set of values for the hyperparameters used in DNN experiments.

Hyperparameters	**Values**
dropout	[0.2, 0.3, 0.4, 0.5]
optimization	['Adadelta', 'Adam', 'RMSprop', 'SGD']
learning_rate	[0.015, 0.010, 0.005, 0.001, 0.0001]
batch_size	[16, 32, 64, 128, 256]
nb_epoch	[100, 150, 200]
units_in_hidden_layers	[[2048, 1024, 512], [1024, 128], [2048, 1024, 512, 128], [2048, 128, 16], [2048, 128], [2048, 512]]
units_in_input_layer	[5000]
early_stopping	[True, False]
batch_normalization	[True, False]
patience	[80]

5.2.5 *Computational resources*

As high computational power is required for the of training multiple ML models with different combinations in the hyperparameters optimization process, the Search 6 cluster from *Departamento de Informática (DI), Universidade do Minho (UM)* was used. As the traditional ML models do not benefit from GPU usage for fast parallel computation but can benefit from multi-threaded processing, the tradional ML models were trained using multiple cores from the nodes 771-1, 771-2, 781-1 and 781-2 with Broadwell Intel CPU microarchitectures from Search 6 cluster. The deep learning models were trained using a machine with a NVIDIA GeForce RTX 2080 Ti GPU. The use of the GPU greatly reduces the amount of time needed to train the deep learning models by using its multiple cores for parallel computing.

5.3 CASE STUDY RESULTS

5.3.1 *Overall results*

For each clinical data endpoint to be predicted, multiple "shallow" and deep learning experiments were executed. The Table 16 and the Figure 22 show the metric results for the prediction of each label for "shallow" and deep learning models. The metric used to select the best model for ER and THREEGENE was MCC, while the metric for NPI was R^2. The best performing model for each endpoint is highlighted in bold. The values of the calculated metrics and sets of selected hyperparameters during the model selection are available in the appendix for both "shallow" or deep learning models.

Table 16.: Results of "shallow" and deep learning experiments. ER and THREEGENEe metric is MCC while NPI metric is R^2.

Endpoint	Model	Metric
ER	KNN	0.8059
	LR	0.9823
	RF	0.9168
	SVM	0.9234
	DNN	0.8790
THREEGENE	KNN	0.5879
	LR	0.8769
	RF	0.8956
	SVM	0.8791
	DNN	0.8423
NPI	KNN	0.0799
	EN	0.0116
	RF	0.2288
	SVM	0.1478
	DNN	0.1604

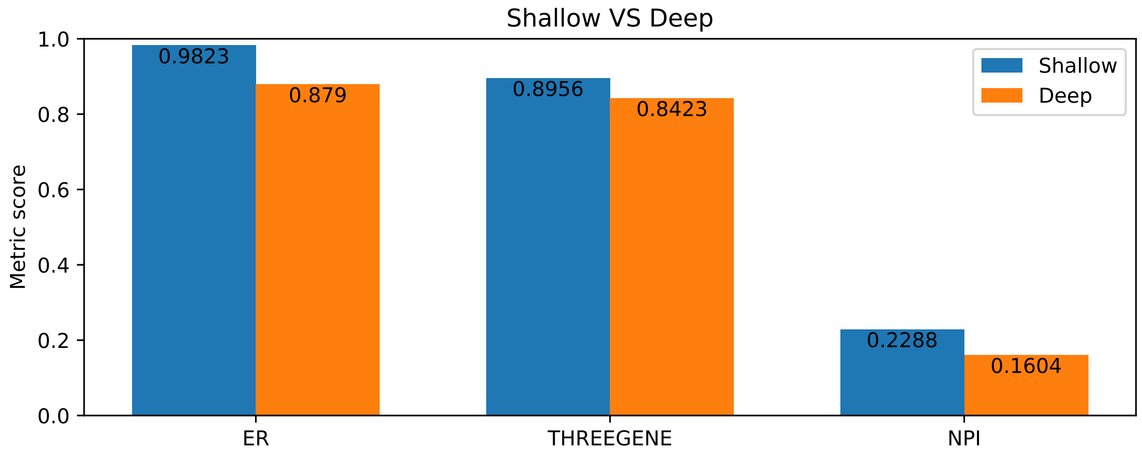


Figure 22.: Barplot with comparison between "shallow" and deep learning models for three endpoints. ER and THREEGENE metric is MCC while NPI is R^2.

5.3.2 *ER prediction results*

The MCC values calculated from the evaluation performed with the test dataset for each of the generated models are shown in Figure 23. As seen in the previous barplot, the RF and SVM obtained a similar MCC score. The DNN model presented a value of 0.879 slightly lower than the RF model. The model with worst performance was KNN with a MCC of 0.8059, considerably lower than other models. LR was the best performing model with 0.9823 of MCC value.

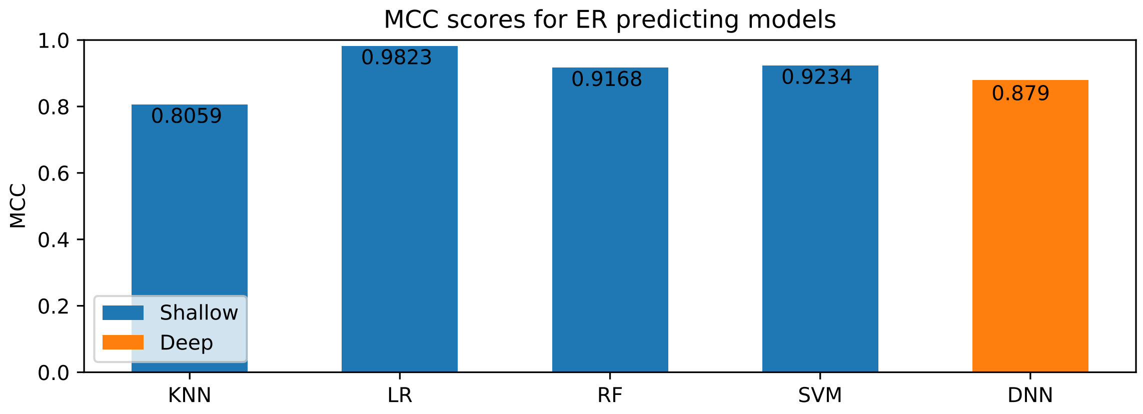


Figure 23.: Barplot with comparison between "shallow" and deep learning models for ER prediction.

5.3.3 *THREEGENE prediction results*

The results of the models used to predict the breast cancer subtype classification based on ER, PR and HER-2 prediction (THREEGENE in the dataset) are shown in Figure 24. Similar to ER predicting experiments, the MCC was used to compare the overall performance among different types of predictive models. As seen in Figure 24, the majority of the generated models have a good performance. The only exception was the KNN model with a MCC of 0.5879, considerably smaller score comparing with others. The remaining models had a similar performance with the DNN model having the second lowest score with 0.8769 and the RF model having the highest score with an MCC of 0.8956. The LR model was the second best performing model with a score of 0.8711, slightly lower when compared to the SVM model.

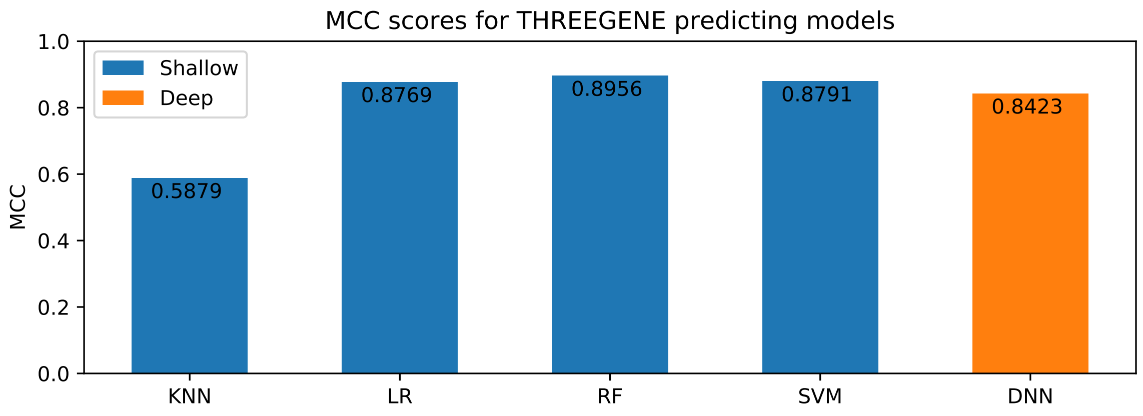


Figure 24.: Barplot with comparison between "shallow" and deep learning models for THREEGENE prediction.

5.3.4 *NPI prediction results*

The results of the evaluation using the test dataset of the NPI predicting models are presented in Figure 25. Contrary to previous experiments, the NPI predicting models seem to have a worse performance, while we do not have any comparative result. While the larger R^2 score was of 0.2288 for the RF model, the model with worst performance, the EN model, obtained a R^2 score of 0.0116. The performance of NPI models was not only worse but more heterogeneous among the types of models when compared to ER and THREEGENE models. The DNN model obtained an MCC of 0.1604, the second highest among the different types of models.

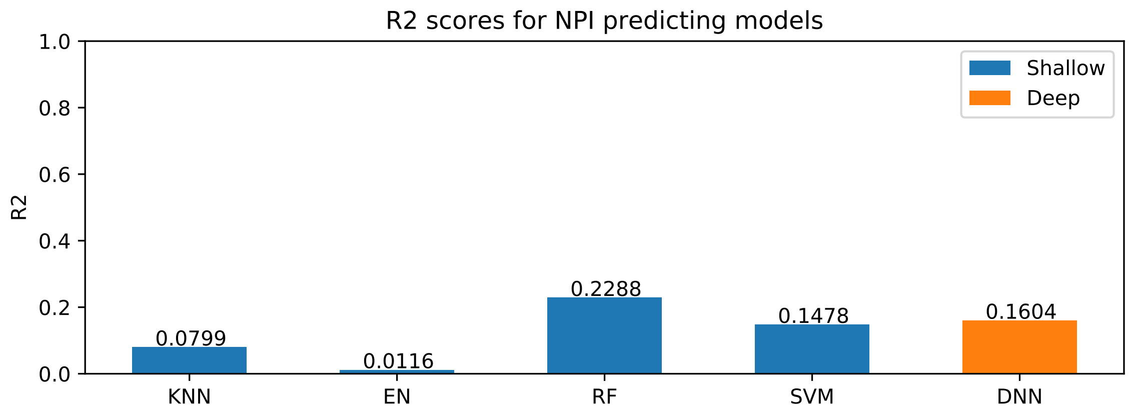


Figure 25.: Barplot with comparison between "shallow" and deep learning models for NPI prediction.

5.3.5 *Multi-tasking DNN results*

The results of the multi-tasking DNN for predicting ER, HER-2 and PR endpoints simultaneously are presented in Figure 26. Although THREEGENE and the three endpoints predicted in the multi-tasking DNN are used for the same clinical purpose the results cannot be directly compared.

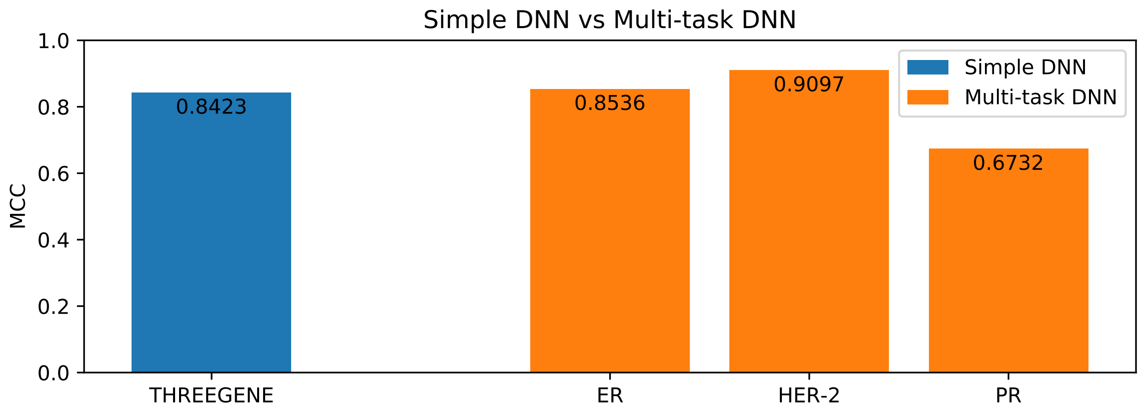


Figure 26.: Barplot with comparison between simple DNN for THREEGENE and multi-tasking DNN for predicting ER, HER-2 and PR.

5.4 DISCUSSION

The obtained overall results were satisfactory with the exception of the NPI endpoint, while we have no comparative performances available. As presented in Table 16 and in Figure 22, the models trained to predict classification tasks seem to perform better when compared to regression models. As presented in bold in Table 16 and presented in the orange bars in Figure 22, the deep learning approach obtained lower scores when compared to the best traditional ML models.

The performance of the framework providing better results for classification when compared to regression problems can be due to a group of factors. It is possible that the gene expression data in this dataset is not suited to regression problems. The NPI can be difficult to predict due to the type of data it represents. The number of affected nodes, size of the primary tumor and stage of the tumor are the different variables used to calculate the NPI. While the stage of tumor is a categorical variable, the number of affected nodes and size of the primary tumor are numerical variables. Predicting the previous numerical variables from gene expression data is a difficult task on itself, but adding a classification task only increases the complexity of predicting the NPI in the case study. Although the results for the NPI endpoint prediction were poor, the capability of the framework to handle regression problems should not be considered to be weak. Other case-studies with different continuous variables should be used to demonstrate the framework potential.

Overall, when comparing the results of traditional ML models and deep learning models, the best traditional models outperformed the DNNs. Still, while the best performing model for each case was a traditional machine learning model, the DNNs were better that part of the traditional models in all cases. In all tasks, the DNNs were better than the KNN models. In the classification tasks, the DNNs had similar performances when compared to the traditional ML models. In the task of predicting NPI, the DNN was the second model with highest R^2. The two main factors responsible for these results are: the amount of available data and the choice of hyperparameters. As mentioned in section 5.2.1 of the case-study chapter 5, the number of samples (entries) was 1700 for THREEGENE and multi-tasking DNN, and 1904 for ER and NPI models. The small amount of available data is not enough for the DNN to achieve their best performance.

Applying feature selection can also be a factor that explains the results of the DNNs. Although traditional ML models benefit from feature selection, the same cannot be said for DNNs. As feature extraction occurs automatically in the training process of a DNN, applying feature selection previously can prevent the model of having access to useful data. The decision of applying feature selection to the dataset was made due to memory errors that appeared when the whole dataset with over seventeen thousand features was used. Various methods were used to clear the memory between iterations, but the problem could not be fully solved.

The hyperparameters tested for the DNN architecture were randomly selected through 50 iterations from a given set of hyperparameters in each experiment. Two problems can occur with the previous approach: the random selection and the input set of hyperparameters. Performing a random selection is faster when compared to a extensive grid search, but it does not check all possible hyperparameter configurations. It is possible that the configuration with best results was not selected during the random search. The problem with the input set of hyperparameters is that it can be provided without the most appropriate hyperparameters for the problem. If provided with inappropriate hyperparameters, the model is expected to underperform. One possible method to improve the results would be to increase the number of iterations, testing a larger number of possible hyperparameter configurations or performing the hyperparameter optimization with a grid search process.

The models predicting ER, THREEGENE and NPI presented different performances. Overall, all models generated for predicting ER presented a good performance. The LR model was the best for ER prediction. The KNN was the worst performing model in every experiment with the exception of the NPI prediction task where the EN performed even worse. The reason behind the poor scores could be that the remaining default hyperparameters of KNN were not suited for the problem. The classifiers excluding KNN obtained a good performance with MCC scores around 0.90 for the ER prediction task. The THREEGENE prediction task presented good results although not as good as the ER predicting models. The best performing model (RF) had a MCC of 0.8956 and accuracy and precision scores over 0.9. As the classification task needed to predict four different classes, the results can be considered to be good. The models to predict NPI underperformed when compared to the classification models for ER and THREEGENE. The performance among regression models was heterogeneous and not satisfactory. The model with best score was RF with an R^2 of 0.2288 and the worst was EN with a R^2 of 0.0116. Overall, the ER and THREEGENE classifiers shown good results while NPI regressor presented poor results.

As the simple DNN for predicting NPI had poor performance, the multi-tasking experiment was only focused on classification tasks. As seen in 26, the results were not similar for all the endpoints. The results presented in the figure cannot be directly compared. The simple DNN for predicting the THREEGENE endpoint contains the classification regarding specific combinations of the genes and there is no available data regarding the performance of the model for classifying each individual gene. The most appropriate analysis would be to compare the results of individual DNN models for each individual gene and compare with the results of the multi-tasking DNN for prediction of the three genes. Comparing the results of the simple DNN for predicting ER with the results of the multi-tasking DNN, the simple model obtained a MCC score of 0.8790 while the multi-tasking model obtained an MCC score of 0.8536. While multi-tasking model obtained a slightly lower result, the same model was capable of predicting two other endpoints in a single training process. The usage of multi-tasking in other cases can be useful for obtaining a model with reasonable or even higher performance with a lower training time when compared to the training process of multiple simple DNNs.

6 SECOND CASE STUDY

6.1 DESCRIPTION OF MELANOMA DATASET

Tumors cannot be considered to be an homogeneous group of malignant cells. A tumor is a full complex ecosystem of different cell types including: malignant, immune and stromal cells. The ecosystem is variable with different amounts of each cell type in the tumor. Each different tumor composition and the interactions among different cells can affect the development and behaviour of each specific tumor. As identifying the tumor cell composition and the interactions among cells in a tumor microenviroment remains a challenge, understanding the full development of a specific tumor is still a difficult task to accomplish [109].

The heterogeneous nature of a tumor is both an opportunity and a challenge in cancer therapy. Having a variable cell composition in each tumor increases the difficulty in using a general therapy for all tumors. However, the diversity in tumor cell composition allows for the usage of targeted and immune therapies [109].

Skin cancer is one of the most common cancers. Among the different types of skin cancer, melanoma is the most rare. Although the melanoma accounts for about 1% of the total number of skin cancer cases, melanoma is responsible for the majority of skin cancer deaths. The American Cancer Society estimates that in 2019 the number of new cases of melanoma will be around 96,480 cases and that 7,230 are expected to die of melanoma [110].

The melanoma is a malignant neoplasm of the melanocytes cells that are are located in the basal layer of the epidermis. Melanocytes are known because of the melanin production, which is responsible for skin pigmentation and ultra-violet radiation protection. The mortality of melanoma is around 11% for primary melanoma patients, while this value is significantly higher in the cases of metastatic melanomas. The large mortality rate is due to low efficacy of the generic cancer therapies that are currently used as treatment [111].

One example of a specific treatment for metastatic melanomas is the RAF/MEK inhibition in melanomas with the $BRAF^{V600E}$ mutation. Although the previous inhibitors improve the survival rate, almost all tumors develop a resistance to these drugs. Unfortunately, patients with lack of the *BRAF*^{V600E} mutations cannot be treated with the previously mentioned inhibitors [111].

The knowledge regarding melanoma tumor composition, microenvironment interactions and response to drugs is essential to increase the survival rate of metastatic melanoma. The ideal methods would be to analyse each tumor to identify the cell composition, interactions among cells and the resistance of the tumor ecosystem to specific drugs [111].

The dataset used in this second case study was obtained from the GEO database [112]. The dataset with the acession GSE72056 is the result of single-cell RNA-seq of patients with melanoma. Two separate files were used from the study: one dataset with the the processed single-cell RNA-seq results and one dataset with metadata regarding the study. The developed framework was used to create classification ML models to predict if each cell was malignant or non-malignant. The dataset was chosen because it shows that the framework is capable of handling single-cell RNA-seq data, a technique increasing in popularity in cell and molecular biology, and it shows that the framework is capable of easily adjust to different datasets obtained from different sources.

6.1.1 *Expression dataset*

The cells used to obtain the single-cell RNA-seq results were obtained after disaggregating the cells from the tumor in a suspension. The viable immune and non-immune cells were recovered to be sequenced using flow cytometry. The cDNA was obtained for each individual cell with a posterior library construction and parallel sequencing. The sequencing protocol previously mentioned was the *Smart-seq2* and the platform used for the single-cell RNA-seq was the *Illumina NextSeq 500*. The dataset contains a total of 4,645 rows regarding the same number of cells obtained from 19 patients and a number of columns of 23,686 with each column regarding a different transcript. The previous protocol was already executed and the dataset with the processed data was the only dataset available.

6.1.2 *Metadata dataset*

The majority of data available in the metadata dataset was not useful for the experiment. However, the dataset contained data regarding the malignancy of each cell inferred by *Copy Number Variation (CNV)* analysis (malignant, non-malignant or unresolved). As the metadata dataset contained the same sample source identifier available in the expression dataset, the cell malignancy data from the metadata dataset can be used to associate to the single-cell RNA-seq data from the expression dataset.

6.2 EXPERIMENTAL SETUP

6.2.1 *Pre-processing*

The first step of pre-processing was performed by selecting the column with the malignancy data in the metadata dataset. After selecting the intended column as a *Pandas* Series, the rows with invalid values were eliminated in both datasets. The rows where the malignancy data was empty (1,396 samples) or unresolved (85 samples) were excluded from the datasets. The *Pandas* Series labels were encoded into integers for posterior usage in ML classification models. The expression dataset was normalized using a *Scikit-learn* StandardScaler method and feature selection was applied selecting the 5,000 features with largest values of mean absolute deviation. The feature selection of 5,000 features was not only applied to traditional ML models, but was also applied for the deep learning models because the full dataset contained 23,686 features. This was needed, since when the whole dataset was used in the deep learning experiments, a memory error occurs due to current hardware limitations. The final datasets were split in X and y (inputs and outputs) train and test matrices using 75% of the data for training and the remaining 25% for testing. The final train matrices contained 1,677 entries and the test matrices contained 560 entries.

6.2.2 *Shallow learning*

To predict the label regarding the malignancy of each cell, multiple traditional ML models were trained using the `Shallow_bin` class. For each type of model, *GridSearchCV* from *Scikit-learn* was used with the number of folds set to 5. The hyperparameters used in the models are the same used in the METABRIC case study as are presented in Table 13. The model with best results found in *GridSearchCV* for each type of traditional ML model was then evaluated using the test dataset and the results were saved in text reports.

6.2.3 *Deep Learning*

In addition to the different traditional ML models, DNNs were trained to predict the malignancy of the cells using the `DNN_bin` class. The set of hyperparameters to be used in the hyperparameter optimization is the same used in METABRIC case study and are presented in Table 15. The number of tested hyperparameter combinations was set to 50. At each iteration, a five-fold cross-validation was performed. The model with best performance was then tested with the test dataset and the results were stored in a text file report.

6.2.4 *Computational resources*

The resources used in the melanoma case study were the same used in the METABRIC case study. The traditional ML models were trained in the Search 6 cluster from DI, UM. The traditional ML classifiers were trained in the cluster because the computation of these models cannot take advantage of a GPU. The DNNs were trained using a server equipped with a *NVIDIA GeForce RTX 2080 Ti* GPU.

6.3 CASE STUDY RESULTS

The results regarding traditional and deep learning models are presented in Figure 27. As seen in the previous figure, all the models had high values of MCC. The different models had a similar performance with the exception of the KNN model that still obtained a high value of a MCC of 0.9472. The model with highest performance regarding the MCC value was the LR model. The DNN model was the second worst model regarding the MCC value, but when compared to the LR MCC, the obtained metric value was just slightly lower. The full results with multiple metrics for each model are presented in the appendix in C.1.

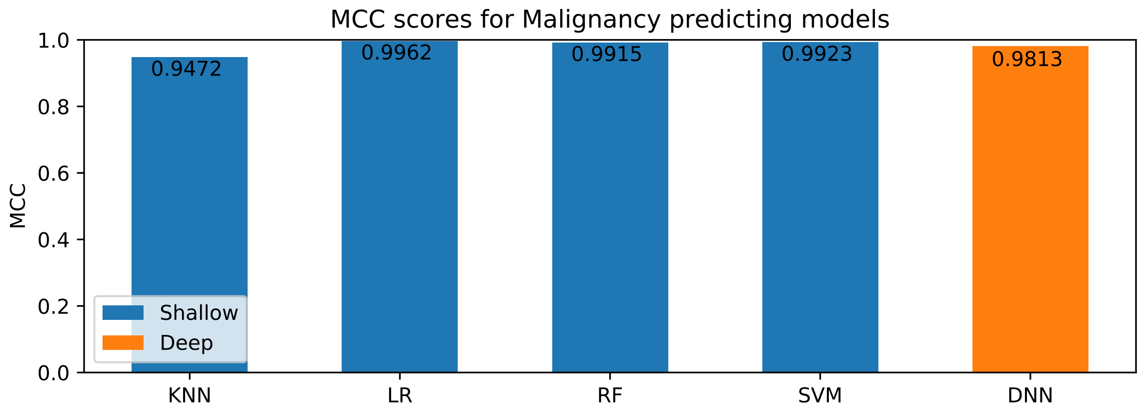


Figure 27.: Barplot with comparison between the different types of models for cell malignancy prediction. The blue bars

6.4 DISCUSSION

The results of the melanoma case study were concordant with the results of the METABRIC case study. The different types of models had a strong performance in the task of predicting the cell malignancy, confirming the capability of the framework to handle binary classification models. Although the KNN model had a strong performance with an MCC of 0.9472, the model had the lower value of MCC among all the tested models similar to what occured with the METABRIC dataset.

The DNN had a lower performance compared to the remaining models similar to what occurred in the METABRIC case study. The cause for the lower performance can be due mainly to the factors previously mentioned in the METABRIC case-study. Applying feature selection, the reduced number of entries and the usage of random search of hyperparameters for hyperparameter optimization were some of the factors that can explain the lower performance of the DNN models.

Although the task for predicting the malignancy of a cell of low difficulty in the used dataset, the generated models could be useful to assist in classification of cells as malignant or non-malignant. As mentioned in the pre-processing section (6.2.1), 1,396 samples had no data regarding the malignancy of the cells and 85 samples had unresolved results. Even if we consider the samples with the malignancy column empty as entries that were not completed when the sample was registered, the 85 unresolved cells malignancy could be predicted using the developed ML models. Using the inference of the cell malignancy using the CNV data with the addition of ML models could improve the labelling process of tumor cells for single-cell RNA-seq experiments regarding cancer patients.

One of the goals of choosing the melanoma dataset was showing that the framework is not only capable of handling microarray data, but also other gene expression techniques as RNA-seq. In the specific case of the selected dataset, single-cell RNA-seq is an interesting alternative as the method has been increasing in popularity in recent years. The good results, even for an easy prediction task, show that the framework can by applied to different types of gene expression datasets provided including recent techniques.

7 CONCLUSION

The developed framework consists in an easy-to-use approach to execute deep learning experiments using gene expression data for individuals with entry-level knowledge of programming, traditional machine learning and deep learning. It provides simple methods for pre-processing datasets including normalization and feature selection techniques. Multiple traditional ML models are available for usage with integrated hyperparameters otimization. The framework allows the user to generate simple DNNs from an input set of hyperparameters to predict a single endpoint and is capable of generating multi-tasking DNNs to predict multiple endpoints in the same model.

Overall, the developed framework provided good results. Although the classification tasks generated good results in the case studies, the regression task obtained a poorer performance. The results obtained from RF and LR were better when compared to simple and multi-tasking DNNs. The classification tasks were capable of predicting the expression of receptors in samples obtained from breast cancer tumor samples. The regression modules were not able of predicting the NPI used to assess the prognosis in breast cancer patients. For the METABRIC case-study, traditional ML models had a better performance and took a shorter amount of time to train when compared to the DNNs. The performance of the simple or multi-tasking DNNs could be improved if the the used dataset was larger and the computational resources allowed the use of all the features of the dataset. For the melanoma case study, the different types of models had a strong performance in the binary classification task of predicting the cell malignancy using the data from experiment metadata and single-cell RNA-seq.

In future work, implementing a more complete `Preprocessing` module with additional normalization, scaling and feature selection techniques could be important to obtain better results. Designing methods that allow users to custom design their own DNN architectures to improve the framework's flexibility and usage by advanced users is another important goal. Redesigning the architecture or including more features than gene expression data can be used to increase the overall performance of the DNNs. Refactoring the framework to be more easy-to-use, streamlined and efficient could also provide a better *Quality of Life (QoL)* and attract more users.

BIBLIOGRAPHY

[1] L. van der Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, no. 9, pp. 2579–2605, 2008.

[2] F. Chollet, *Deep Learning with Python*. Shelter Island,NY: Manning Publications Co., 2017.

[3] Y. Ma, Z. Xiang, Q. Du, and W. Fan, "Effects of user-provided photos on hotel review helpfulness: An analytical approach with deep leaning," *International Journal of Hospitality Management*, vol. 71, pp. 120–131, apr 2018.

[4] M. Hossin and M. Sulaiman, "A Review on Evaluation Metrics for Data Classification Evaluations," *International Journal of Data Mining & Knowledge Management Process (IJDKP)*, vol. 5, no. 2, pp. 1–11, 2015.

[5] S. Boughorbel, F. Jarray, and M. El-Anbari, "Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric," *PLOS ONE*, vol. 12, p. e0177678, jun 2017.

[6] R. L. Siegel, K. D. Miller, and A. Jemal, "Cancer Statistics, 2018.," *CA: a cancer journal for clinicians*, vol. 67, no. 1, pp. 7–30, 2018.

[7] F. Sanchez-Vega, M. Mina, J. Armenia, W. K. Chatila, and A. Luna, "Oncogenic Signaling Pathways in The Cancer Genome Atlas," *Cell*, vol. 173, no. 2, pp. 321–337.e10, 2018.

[8] B. A. Perkins, C. T. Caskey, P. Brar, E. Dec, D. S. Karow, A. M. Kahn, Y.-C. C. Hou, N. Shah, D. Boeldt, E. Coughlin, G. Hands, V. Lavrenko, J. Yu, A. Procko, J. Appis, A. M. Dale, L. Guo, T. J. Jönsson, B. M. Wittmann, I. Bartha, S. Ramakrishnan, A. Bernal, J. B. Brewer, S. Brewerton, W. H. Biggs, Y. Turpaz, and J. C. Venter, "Precision medicine screening using whole-genome sequencing and advanced imaging to identify disease risk in adults.," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 115, pp. 3686–3691, apr 2018.

[9] E. Lin and H.-Y. Lane, "Machine learning and systems genomics approaches for multi-omics data," *Biomarker Research*, vol. 5, no. 1, p. 2, 2017.

[10] R. Fakoor, F. Ladhak, A. Nazi, and M. Huber, "Using deep learning to enhance cancer diagnosis and classification," *Proceeding of the 30th international conference on machine learning Atlanta, Georgia,USA*, vol. 28, 2013.

[11] G. P. Way and F. Sanchez-Vega, "Machine Learning Detects Pan-cancer Ras Pathway Activation in The Cancer Genome Atlas," *Cell Reports*, vol. 23, no. 1, pp. 172–180.e3, 2018.

[12] A. Mosavi, "Deep Learning : a Review," *Advances in Intelligent Systems and Computing*, no. July, 2017.

[13] R. K. Sevakula, V. Singh, N. K. Verma, C. Kumar, and Y. Cui, "Transfer Learning for Molecular Cancer classification using Deep Neural Networks," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pp. 1–1, 2018.

[14] T. Turki, Z. Wei, and J. T. Wang, "Transfer Learning Approaches to Improve Drug Sensitivity Prediction in Multiple Myeloma Patients," *IEEE Access*, vol. 5, pp. 7381–7393, 2017.

[15] D. Sun, M. Wang, and A. Li, "A multimodal deep neural network for human breast cancer prognosis prediction by integrating multi-dimensional data," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, pp. 1–1, 2018.

[16] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization Methods for Large-Scale Machine Learning," *Society for Industrial and Applied Mathematics*, vol. 60, no. 2, pp. 223–311, 2018.

[17] M. Wainberg, D. Merico, A. Delong, and B. J. Frey, "Deep learning in biomedicine," *Nature Biotechnology*, vol. 36, no. 9, pp. 829–838, 2018.

[18] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*, vol. 2, no. 0, pp. 1137–1143, 1995.

[19] M. J. Pencina, R. B. D'Agostino Sr, R. B. D'Agostino Jr, and R. S. Vasan, "Evaluating the added predictive ability of a new marker: From area under the ROC curve to reclassification and beyond," *STATISTICS IN MEDICINE*, vol. 27, no. 1, pp. 157–172, 2008.

[20] J. Huang and C. X. Ling, "Using AUC and accuracy in evaluating learning algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.

[21] C. Willmott and K. Matsuura, "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance," *Climate Research*, vol. 30, pp. 79–82, 2005.

[22] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, pp. 679–688, oct 2006.

[23] R. Todeschini, D. Ballabio, and F. Grisoni, "Beware of Unreliable Q2! A Comparative Study of Regression Metrics for Predictivity Assessment of QSAR Models," *Journal of Chemical Information and Modeling*, vol. 56, no. 10, pp. 1905–1913, 2016.

[24] J. Miles, "R Squared, Adjusted R Squared," in *Wiley StatsRef: Statistics Reference Online*, Chichester, UK: John Wiley & Sons, Ltd, sep 2014.

[25] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Stanford, California: Springer, 2nd editio ed., 2017.

[26] M. A. Mansournia, A. Geroldinger, S. Greenland, and G. Heinze, "Separation in Logistic Regression: Causes, Consequences, and Control," *American Journal of Epidemiology*, vol. 187, pp. 864–870, apr 2018.

[27] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, vol. 67, no. 2, pp. 301–320, 2005.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[29] M. E. Syed, "Attribute weighting in K-nearest neighbor classification," no. November, p. 50, 2014.

[30] K. Hechenbichler and K. Schliep, "Weighted k-Nearest-Neighbor Techniques and Ordinal Classification," *discussion paper*, vol. 399, no. January 2004, p. 17, 2004.

[31] L. Jiang, H. Zhang, and Z. Cai, "Dynamic K-Nearest-Neighbor Naive Bayes with Attribute Weighted," *Fuzzy Systems and Knowledge Discovery*, vol. 4223, pp. 365–368, 2006.

[32] W. S. Noble, "What is a support vector machine?," *Nature Biotechnology*, vol. 24, pp. 1565–1567, dec 2006.

[33] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda, "Decision Trees for Mining Data Streams Based on the Gaussian Approximation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, pp. 108–119, jan 2014.

[34] T. G. Dietterich, "Ensemble Methods in Machine Learning," pp. 1–15, Springer, Berlin, Heidelberg, 2000.

[35] T. Hothorn and B. Lausen, "Double-bagging: combining classifiers by bootstrap aggregation," *Pattern Recognition*, vol. 36, pp. 1303–1309, jun 2003.

[36] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," 2001.

[37] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors)," *The Annals of Statistics*, vol. 28, pp. 337–407, apr 2000.

[38] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, pp. 367–378, feb 2002.

[39] G. Rätsch, T. Onoda, and K.-R. Müller, "Soft Margins for AdaBoost," *Machine Learning*, vol. 42, no. 3, pp. 287–320, 2001.

[40] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[41] I. Guyon, "An Introduction to Variable and Feauture Selection," *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[42] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature Selection: A Data Perspective," *ACM Computing Surveys*, vol. 50, pp. 1–45, dec 2018.

[43] C. M. Farrelly, S. J. Schwartz, A. Lisa Amodeo, D. J. Feaster, D. L. Steinley, A. Meca, and S. Picariello, "The analysis of bridging constructs with hierarchical clustering methods: An application to identity," *Journal of Research in Personality*, vol. 70, pp. 93–106, oct 2017.

[44] S. Gupta, R. Kumar, K. Lu, B. Moseley, and S. Vassilvitskii, "Local search methods for k-means with outliers," *Proceedings of the VLDB Endowment*, vol. 10, pp. 757–768, mar 2017.

[45] I. Jolliffe, "Principal Component Analysis," in *International Encyclopedia of Statistical Science*, pp. 1094–1096, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[46] D. Graupe, *PRINCIPLES OF ARTIFICIAL NEURAL NETWORKS*, vol. 53. Toh Tuck Link: World Scientific Publishing Co. Pte. Ltd., 2nd editio ed., 2007.

[47] D. Riesel, "A brief Introduction on Neural Networks," *Springer-Verlag, Berlin*, p. Second edition, 2007.

[48] Y. LeCun, Y. Bengio, G. Hinton, L. Y., B. Y., and H. G., "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[49] P. Baldi, "Autoencoders , Unsupervised Learning , and Deep Architectures," pp. 37–50, 2012.

[50] J. Tan, M. Ung, C. Cheng, and C. S. Grenne, "Unsupervised feature construction and knowledge extraction from genome-wide assays of breast cancer with denoising autoencoders.," *Biocomputing 2015*, pp. 132–143, 2014.

[51] Y. Pu, W. Wang, R. Henao, L. Chen, Z. Gan, C. Li, and L. Carin, "Adversarial Symmetric Variational Autoencoder," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 4330–4339, Curran Associates, Inc., 2017.

[52] S. Ruder, "An overview of gradient descent optimization," pp. 1–14, 2016.

[53] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The Difficulty of Training Deep Architectures and the Effect of Unsupervised Pre-Training," *Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS), JMLR Workshop and Conference Procedings*, vol. 5, pp. 153–160, 2009.

[54] Y. Yao, L. Rosasco, and A. Caponnetto, "On Early Stopping in Gradient Descent Learning," *Constructive Approximation*, vol. 26, pp. 289–315, aug 2007.

[55] Y. Zhang and Q. Yang, "An overview of multi-task learning," *National Science Review*, vol. 5, no. 1, pp. 30–43, 2018.

[56] L. Torrey and J. Shavlik, "Transfer Learning," in *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, ch. Transfer L, Information Science Reference, 2010.

[57] B. Zoph, D. Yuret, J. May, and K. Knight, "Transfer Learning for Low-Resource Neural Machine Translation," *CoRR*, pp. 1568–1575, 2016.

[58] S. Hoo-Chang, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning and Daniel Mollura are with Center for Infectious Disease Imaging HHS Public Access," *IEEE Trans. Med. Imag.*, vol. 35, no. 5, pp. 1285–1298, 2016.

[59] A. Parvat, J. Chavan, S. Kadam, S. Dev, and V. Pathak, "A survey of deep-learning frameworks," *Proceedings of the International Conference on Inventive Systems and Control, ICISC 2017*, pp. 1–7, 2017.

[60] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cuDNN : Efficient Primitives for Deep Learning," pp. 1–9.

[61] B. A. Fine, "The Evolution of Nondirectiveness in Genetic Counseling and Implications of the Human Genome Project," in *Genetic Counseling*, pp. 101–118, Routledge, jul 2017.

[62] F. S. Collins, M. Morgan, and A. Patrinos, "The Human Genome Project: lessons from large-scale biology.," *Science (New York, N.Y.)*, vol. 300, pp. 286–90, apr 2003.

[63] R. J. DeBerardinis, J. J. Lum, G. Hatzivassiliou, and C. B. Thompson, "The Biology of Cancer: Metabolic Reprogramming Fuels Cell Growth and Proliferation," *Cell Metabolism*, vol. 7, pp. 11–20, jan 2008.

[64] J. Larry Jameson and D. L. Longo, "Precision MedicinePersonalized, Problematic, and Promising," *Obstetrical & Gynecological Survey*, vol. 70, pp. 612–614, oct 2015.

[65] Y.-C. Chiu, H.-I. H. Chen, T. Zhang, S. Zhang, A. Gorthi, L.-J. Wang, Y. Huang, and Y. Chen, "Predicting drug response of tumors from integrated genomic profiles by deep neural networks," may 2018.

[66] E. W. Sliwoski, Gregory. Kothiwale, Sandeepkumar. Meiler, Jens. Lowe Jr, "Computational Methods in Drug Discovery," *Pharmacological Reviews*, vol. 66, no. 1, pp. 334–395, 2014.

[67] S. C. Schuster, "Next-generation sequencing transforms today's biology," *Nature Methods*, vol. 5, pp. 16–18, jan 2008.

[68] K. J. Karczewski and M. P. Snyder, "Integrative omics for health and disease," *Nature Reviews Genetics*, vol. 19, no. 5, pp. 299–310, 2018.

[69] K. Chaudhary, O. B. Poirion, L. Lu, and L. X. Garmire, "Deep learningbased multi-omics integration robustly predicts survival in liver cancer," *Clinical Cancer Research*, vol. 24, no. 6, pp. 1248–1259, 2018.

[70] D. C. Koboldt, Q. Zhang, D. E. Larson, D. Shen, M. D. McLellan, L. Lin, C. A. Miller, E. R. Mardis, L. Ding, and R. K. Wilson, "VarScan 2: somatic mutation and copy number alteration discovery in cancer by exome sequencing.," *Genome research*, vol. 22, pp. 568–76, mar 2012.

[71] K. Hayashi, "PCR-SSCP: A method for detection of mutations," *Genetic Analysis: Biomolecular Engineering*, vol. 9, pp. 73–79, jun 1992.

[72] R. Fodde and M. Losekoot, "Mutation detection by denaturing gradient gel electrophoresis (DGGE)," *Human Mutation*, vol. 3, pp. 83–94, jan 1994.

[73] C. Gawad, W. Koh, and S. R. Quake, "Single-cell genome sequencing: current state of the science," *Nature Reviews Genetics*, vol. 17, pp. 175–188, mar 2016.

[74] R. Kappelhoff, X. S. Puente, C. H. Wilson, A. Seth, C. López-Otín, and C. M. Overall, "Overview of transcriptomic analysis of all human proteases, non-proteolytic homologs and inhibitors: Organ, tissue and ovarian cancer cell line expression profiling of the human protease degradome by the CLIP-CHIP DNA microarray," *Biochimica et Biophysica Acta (BBA) - Molecular Cell Research*, vol. 1864, pp. 2210–2219, nov 2017.

[75] S. Michiels, S. Koscielny, and C. Hill, "Prediction of cancer outcome with microarrays: a multiple random validation strategy," *The Lancet*, vol. 365, pp. 488–492, feb 2005.

[76] Z. Wang, M. Gerstein, and M. Snyder, "RNA-Seq: a revolutionary tool for transcriptomics.," *Nature Reviews Genetics*, vol. 10, pp. 57–73, 2009.

[77] M. A. Jensen, V. Ferretti, R. L. Grossman, and L. M. Staudt, "The NCI Genomic Data Commons as an engine for precision medicine," *Blood*, vol. 130, no. 4, pp. 453–459, 2017.

[78] Y. Chang, H. Park, H.-J. Yang, S. Lee, K.-Y. Lee, T. S. Kim, J. Jung, and J.-M. Shin, "Cancer Drug Response Profile scan (CDRscan): A Deep Learning Model That Predicts Drug Effectiveness from Cancer Genomic Signature," *Scientific Reports*, vol. 8, p. 8857, dec 2018.

[79] Y. Yuan, Y. Shi, C. Li, J. Kim, W. Cai, Z. Han, and D. D. Feng, "DeepGene: an advanced cancer type classifier based on deep learning and somatic point mutations," *BMC Bioinformatics*, vol. 17, p. 476, dec 2016.

[80] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, pp. 115–118, 2017.

[81] J. A. Cruz and D. S. Wishart, "Applications of Machine Learning in Cancer Prediction and Prognosis," *Cancer Informatics*, vol. 2, p. 117693510600200, jan 2006.

[82] S. Peng, Q. Xu, X. B. Ling, X. Peng, W. Du, and L. Chen, "Molecular classification of cancer types from microarray data using the combination of genetic algorithms and support vector machines," *FEBS Letters*, vol. 555, no. 2, pp. 358–362, 2003.

[83] B. D. Lehmann, Y. Shyr, J. A. Pietenpol, B. D. Lehmann, J. A. Bauer, X. Chen, M. E. Sanders, A. B. Chakravarthy, Y. Shyr, and J. A. Pietenpol, "Identification of human triple-negative breast cancer subtypes and preclinical models for selection of targeted therapies," *Journal of Clinical Investigation*, vol. 121, no. 7, pp. 2750–2767, 2011.

[84] T. Ayer, O. Alagoz, J. Chhatwal, J. W. Shavlik, C. E. Kahn, and E. S. Burnside, "Breast cancer risk estimation with artificial neural networks revisited," *Cancer*, vol. 116, pp. 3310–3321, apr 2010.

[85] Y. S. Kim, S. Y. Sohn, and C. N. Yoon, "Screening test data analysis for liver disease prediction model using growth curve," *Biomedicine & Pharmacotherapy*, vol. 57, pp. 482–488, dec 2003.

[86] P. Rosado, P. Lequerica-Fernández, L. Villallaín, I. Peña, F. Sanchez-Lasheras, and J. C. de Vicente, "Survival model in oral squamous cell carcinoma based on clinicopathological parameters, molecular markers and support vector machines," *Expert Systems with Applications*, vol. 40, pp. 4770–4776, sep 2013.

[87] D. Delen, G. Walker, and A. Kadam, "Predicting breast cancer survivability: a comparison of three data mining methods," *Artificial Intelligence in Medicine*, vol. 34, pp. 113–127, jun 2005.

[88] W. Kim, K. S. Kim, J. E. Lee, D.-Y. Noh, S.-W. Kim, Y. S. Jung, M. Y. Park, and R. W. Park, "Development of Novel Breast Cancer Recurrence Prediction Model Using Support Vector Machine," *Journal of Breast Cancer*, vol. 15, p. 230, jun 2012.

[89] L. Ahmad and A. Eshlaghy, "Using Three Machine Learning Techniques for Predicting Breast Cancer Recurrence," *Journal of Health & Medical Informatics*, vol. 04, no. 02, p. 2018, 2013.

[90] J. Barretina, G. Caponigro, N. Stransky, K. Venkatesan, A. A. Margolin, S. Kim, C. J. Wilson, J. Lehár, G. V. Kryukov, D. Sonkin, A. Reddy, M. Liu, L. Murray, M. F. Berger, J. E. Monahan, P. Morais, J. Meltzer, A. Korejwa, J. Jané-Valbuena, F. A. Mapa, J. Thibault, E. Bric-Furlong, P. Raman, A. Shipway, I. H. Engels, J. Cheng, G. K. Yu, J. Yu, P. Aspesi, M. de Silva, K. Jagtap, M. D. Jones, L. Wang, C. Hatton, E. Palescandolo, S. Gupta, S. Mahan, C. Sougnez, R. C. Onofrio, T. Liefeld, L. MacConaill, W. Winckler, M. Reich, N. Li, J. P. Mesirov, S. B. Gabriel, G. Getz, K. Ardlie, V. Chan, V. E. Myer, B. L. Weber, J. Porter, M. Warmuth, P. Finan, J. L. Harris, M. Meyerson, T. R. Golub, M. P. Morrissey, W. R. Sellers, R. Schlegel, and L. A. Garraway, "The Cancer Cell Line Encyclopedia enables predictive modelling of anticancer drug sensitivity," *Nature*, vol. 483, pp. 603–607, mar 2012.

[91] M. P. Menden, F. Iorio, M. Garnett, U. McDermott, C. H. Benes, P. J. Ballester, and J. Saez-Rodriguez, "Machine Learning Prediction of Cancer Cell Sensitivity to Drugs Based on Genomic and Chemical Properties," *PLoS ONE*, vol. 8, p. e61318, apr 2013.

[92] C. Cao, F. Liu, H. Tan, D. Song, W. Shu, W. Li, Y. Zhou, X. Bo, and Z. Xie, "Deep Learning and Its Applications in Biomedicine," *Genomics, Proteomics and Bioinformatics*, vol. 16, no. 1, pp. 17–32, 2018.

[93] S. Pereira, A. Pinto, V. Alves, and C. A. Silva, "Brain Tumor Segmentation Using Convolutional Neural Networks in MRI Images," *IEEE Transactions on Medical Imaging*, vol. 35, pp. 1240–1251, may 2016.

[94] Yan Xu, Jun-Yan Zhu, E. Chang, and Zhuowen Tu, "Multiple clustered instance learning for histopathology cancer image classification, segmentation and clustering," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 964–971, IEEE, jun 2012.

[95] A. Petrosian, D. Prokhorov, R. Homan, R. Dasheiff, and D. Wunsch, "Recurrent neural network based prediction of epileptic seizures in intra- and extracranial EEG," *Neurocomputing*, vol. 30, pp. 201–218, jan 2000.

[96] T. Kooi, G. Litjens, B. van Ginneken, A. Gubern-Mérida, C. I. Sánchez, R. Mann, A. den Heeten, and N. Karssemeijer, "Large scale deep learning for computer aided detection of mammographic lesions," *Medical Image Analysis*, vol. 35, pp. 303–312, jan 2017.

[97] Yang Li and T. Shibuya, "Malphite: A convolutional neural network and ensemble learning based protein secondary structure predictor," in *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pp. 1260–1266, IEEE, nov 2015.

[98] S. Zhang, J. Zhou, H. Hu, H. Gong, L. Chen, C. Cheng, and J. Zeng, "A deep learning framework for modeling structural features of RNA-binding protein targets," *Nucleic Acids Research*, vol. 44, pp. e32–e32, feb 2016.

[99] Y. Chen, Y. Li, R. Narayan, A. Subramanian, and X. Xie, "Gene expression inference with deep learning," *Bioinformatics*, vol. 32, pp. 1832–1839, jun 2016.

[100] M. K. K. Leung, H. Y. Xiong, L. J. Lee, and B. J. Frey, "Deep learning of the tissue-regulated splicing code," *Bioinformatics*, vol. 30, pp. i121–i129, jun 2014.

[101] S. Park, S. Min, H. Choi, and S. Yoon, "deepMiRGene: Deep Neural Network based Precursor microRNA Prediction," apr 2016.

[102] D. Quang, Y. Chen, and X. Xie, "DANN: a deep learning approach for annotating the pathogenicity of genetic variants," *Bioinformatics*, vol. 31, pp. 761–763, mar 2015.

[103] Y. Xiao, J. Wu, Z. Lin, and X. Zhao, "A deep learning-based multi-model ensemble method for cancer prediction," *Computer Methods and Programs in Biomedicine*, vol. 153, pp. 1–9, 2018.

[104] L. Zhang, Q. Yu, X. C. Wu, M. C. Hsieh, M. Loch, V. W. Chen, E. Fontham, and T. Ferguson, "Impact of chemotherapy relative dose intensity on cause-specific and overall survival for stage IIII breast cancer: ER+/PR+, HER2- vs. triple-negative," *Breast Cancer Research and Treatment*, vol. 169, no. 1, pp. 175–187, 2018.

[105] Y. Fong, J. Evans, D. Brook, J. Kenkre, P. Jarvis, and K. Gower-Thomas, "The Nottingham Prognostic Index: five-and ten-year data for all-cause survival within a screened population," *Annals of The Royal College of Surgeons of England*, vol. 97, no. 2, pp. 137–139, 2015.

[106] J. Gao, B. A. Aksoy, U. Dogrusoz, G. Dresdner, B. Gross, S. O. Sumer, Y. Sun, A. Jacobsen, R. Sinha, E. Larsson, E. Cerami, C. Sander, and N. Schultz, "Integrative Analysis of Complex Cancer Genomics and Clinical Profiles Using the cBioPortal," *Science Signaling*, vol. 6, pp. pl1 LP – pl1, apr 2013.

[107] C. Curtis, S. P. Shah, S.-F. Chin, G. Turashvili, O. M. Rueda, M. J. Dunning, D. Speed, A. G. Lynch, S. Samarajiwa, Y. Yuan, S. Gräf, G. Ha, G. Haffari, A. Bashashati, R. Russell, S. McKinney, A. Langerød, A. Green, E. Provenzano, G. Wishart, S. Pinder, P. Watson, F. Markowetz, L. Murphy, I. Ellis, A. Purushotham, A.-L. Børresen-Dale, J. D. Brenton, S. Tavaré, C. Caldas, and S. Aparicio, "The genomic and transcriptomic architecture of 2,000 breast tumours reveals novel subgroups," *Nature*, vol. 486, no. 7403, pp. 346–352, 2012.

[108] B. Pereira, S. F. Chin, O. M. Rueda, H. K. M. Vollan, E. Provenzano, H. A. Bardwell, M. Pugh, L. Jones, R. Russell, S. J. Sammut, D. W. Tsui, B. Liu, S. J. Dawson, J. Abraham, H. Northen, J. F. Peden, A. Mukherjee, G. Turashvili, A. R. Green, S. McKinney, A. Oloumi, S. Shah, N. Rosenfeld, L. Murphy, D. R. Bentley, I. O. Ellis, A. Purushotham, S. E. Pinder, A. L. BØrresen-Dale, H. M. Earl, P. D. Pharoah, M. T. Ross, S. Aparicio, and C. Caldas, "The somatic mutation profiles of 2,433 breast cancers refines their genomic and transcriptomic landscapes," *Nature Communications*, vol. 7, no. May, 2016.

[109] I. Tirosh, B. Izar, S. M. Prakadan, M. H. W. Ii, D. Treacy, J. J. Trombetta, A. Rotem, C. Rodman, C. Lian, G. Murphy, M. Fallahi-sichani, K. Dutton-regester, J.-r. Lin, S. W. Kazer, A. Gaillard, and K. E. Kolb, "Dissecting the multicellular ecosystem of

metastatic melanoma by single-cell RNA-seq," *Science*, vol. 352, no. 6282, pp. 189–196, 2016.

[110] The American Cancer Society, "Melanoma Skin Cancer Statistics," 2019.

[111] Y. Liu and M. S. Sheikh, "Melanoma: Molecular pathogenesis and therapeutic management," *Molecular and Cellular Pharmacology*, vol. 6, no. 3, pp. 31–44, 2014.

[112] T. Barrett, S. E. Wilhite, P. Ledoux, C. Evangelista, I. F. Kim, M. Tomashevsky, K. A. Marshall, K. H. Phillippy, P. M. Sherman, M. Holko, A. Yefanov, H. Lee, N. Zhang, C. L. Robertson, N. Serova, S. Davis, and A. Soboleva, "NCBI GEO: Archive for functional genomics data sets - Update," *Nucleic Acids Research*, vol. 41, no. D1, pp. 991–995, 2013.

A

CODE EXPLANATION

A.1 PREPROCESSING

Table A.1.1.: Preprocessing properties

Property	**Description**
expr_file	Expression file path
clinic_data_file	Clinical data file path
exprs	Gene expression dataframe
clinic_data	Clinical data dataframe
n_features	Number of features
n_samples	Number of samples
features	List of features
mt	Boolean True for multi-tasking experiments

Table A.1.2.: Preprocessing methods

Method	Description
read_exprs_data	Reads expression file genes in columns and samples in rows
read_clinical_data	Reads clinical data file and generates a dataframe with one or more labels
set_feature_number	Sets the variable with the number of columns
set_list_features	Sets a list with the features
set_sample_number	Sets the variable with the number of rows
set_index	Indexes both dataframes with the same indexation
nom_to_num	Categorizes string values from one or more labels to numeric values
load_data	Reads both files, sets index, feature list, number of features and samples
get_feature_number	Returns the number of features
get_list_features	Returns the list of features
get_sample_number	Returns the number of features
variance_filter	Selects features with variance higher than a given threshold
mse_filter	Selects the n features with higher mean absolute deviation values
filter_genes	Selects the n best features using SelectKBest with f_classif filter
normalize_zero_one	Applies MinMaxScaler to expression data
normalize_data	Applies StandardScaler to expression data
save_matrices_train_test	Saves train and test matrices
split_dataset	Applies normalization, feature selection and returns train and test matrices

A.2 SHALLOW

Table A.2.1.: Shallow properties

Property	Description
X	X matrix
y	y matrix
X_train, X_test, y_train, y_test	X and y train and test matrices
splitted	Boolean True if X and y are split in train and test
feature_number	Number of features
list_models	List of available types of models
model	Scikit-learn object generated after GridSearchCV
model_name	Name of model type
scoring	Scorer used to evaluate performance in GridSearchCV
cv	Number of folds to be used in cross validation

Table A.2.2.: Shallow methods

Method	Description
print_parameter_values	Print hyperparameter values
predict_values	Returns a prediction for a given input
evaluate_model	Evaluates model performance using different metrics
calculate_scores_cv	Calculates and returns cross validation results
format_scores_cv	Formats cross validation scores
save_best_model	Saves the best performing model
load_model	Loads a stored model
write_cv_results	Writes cross validation scores in a csv file
write_report	Writes evaluation results in a text file
write_report_cv	Writes the results of cross validation applied to model evaluation
model_selection_(model type)	Performs GridSearchCV for a type of model
multi_model_selection	Performs model selection and evaluation

A.3 DNN

Table A.3.1.: DNN properties

Property	Description
X	X matrix
y	y matrix
X_train, X_test, y_train, y_test	X and y train and test matrices
splitted	Boolean True if X and y are split in train and test
feature_number	Number of features
parameters	Dictionary with model hyperparameters
filename	Name for files generated in the experiment
verbose	If 1, prints the data during the DNN training
parameters_batch	Batch of hyperparameters used in model selection
model_selection_history	Stores scores and history of DNN training during model selection
cv	Number of folds to be used in cross validation
history	History of the training of a DNN

Table A.3.2.: DNN methods

Method	**Description**
print_parameter_values	Prints hyperparameter values
create_DNN_model	Creates a *Keras* sequential model
cv_fit	Performs cross validation
fit_model	Fits DNN model
print_fit_results	Prints fitting results
predict_values	Returns a prediction for a given input
evaluate_model	Evaluates model performance using different metrics
format_scores_cv	Formats cross validation scores
model_selection	Performs a random search hyperparameter optimization
find_best_model	Returns best performing model
select_best_model	Loads the parameter configuration of best the performing model
batch_parameter_shuffler	Selects a random set of hyperparameters
set_filename	Sets the filename
plot_model_performance	Generates a graph with training metrics
write_model_selection_results	Writes a csv file with results of the model selection
write_report	Writes a csv file with model evaluation results
model_fit_results	Fits and evaluates a model with the given hyperparameters
save_best_model	Saves the best performing model
load_model	Loads a stored model
multi_model_selection	Performs model selection and evaluation

A.4 DNN_MT

Table A.4.1.: DNN_MT properties

Property	Description
X	X matrix
y	y matrix
X_train, X_test, y_train, y_test	X and y train and test matrices
splitted	Boolean True if X and y are split in train and test
feature_number	Number of features
parameters	Dictionary with model hyperparameters
filename	Name for files generated in the experiment
verbose	If 1, prints the data during DNN training
parameters_batch	Batch of hyperparameters used in model selection
model_selection_history	Stores scores and history of DNN training during model selection
cv	Number of fold to be used in cross validation
history	History of the training of a DNN
types	List of types of ML problems of each label
loss_weights	List with weight loss for each label
labels	List with labels
labels_number	Number of labels

Table A.4.2.: DNN_MT methods

Method	**Description**
insert_labels	Inserts a new list of labels
insert_loss_weights	Inserts a list of loss weights
create_DNN_model	Creates a model with *Keras* functional API
print_parameter_values	Prints hyperparameter values
fit_model	Fits multi-task DNN model
predict_values	Returns a prediction for a given input
evaluate_model	Evaluates model performance using different metrics
batch_parameter_shuffler	Returns a random set of hyperparameters
set_new_parameters	Sets a set of hyperparameters to the model
y_splitter	Splits the different labels
hold_out_fit	Fits a DNN model and evaluates using test data
fold_generator	Generates and returns folds for cross validation
cv_fit	Performs cross validation
model_selection	Performs a random search hyperparameter optimization
find_best_model	Return best performing model
select_best_model	Loads the parameter configuration of the best performing model
best_model_selection	Performs model selection and evaluation
save_best_model	Saves the best performing model
load_model	Loads a stored model

B

DETAILS OF RESULTS - FIRST CASE STUDY

B.1 ER MODEL RESULTS

Table B.1.1.: Results of the ER predicting KNN model.

Hyperparameters	Metrics
algorithm: auto	AUC: 0.8715
leaf_size: 30	Accuracy: 0.9328
metric: cityblock	F1 score: 0.9574
metric_params: None	MCC: 0.8059
n_neighbors: 3	Precision: 0.9302
p: 2	Recall: 0.9863
weights: uniform	Log loss: 2.3220

Table B.1.2.: Results of the ER predicting LR model.

Hyperparameters	Metrics
C: 0.1 class_weight: None dual: False fit_intercept: True intercept_scaling: 1 max_iter: 100 multi_class: warn penalty: l1 random_state: None solver: warn tol: 0.0001 warm_start: False	AUC: 0.9896 Accuracy: 0.9937 F1 score: 0.9959 MCC: 0.9823 Precision: 0.9945 Recall: 0.9972 Log loss: 0.2177

Table B.1.3.: Results of the ER predicting RF model.

Hyperparameters	Metrics
bootstrap: True	
class_weight: None	
criterion: gini	
max_depth: None	
max_features: auto	AUC: 0.9432
max_leaf_nodes: None	Accuracy: 0.9706
min_impurity_decrease: 0.0	F1 score: 0.9811
min_impurity_split: None	MCC: 0.9168
min_samples_leaf: 1	Precision: 0.9680
min_samples_split: 2	Recall: 0.9945
min_weight_fraction_leaf: 0.0	Log loss: 1.0159
n_estimators: 200	
oob_score: False	
random_state: None	
warm_start: False	

Table B.1.4.: Results of the ER predicting SVM model.

Hyperparameters	Metrics
C: 0.01	
cache_size: 200	
class_weight: None	
coef0: 0.0	AUC: 0.9602
decision_function_shape: ovr	Accuracy: 0.9727
degree: 3	F1 score: 0.9822
gamma: auto_deprecated	MCC: 0.9234
kernel: linear	Precision: 0.9809
max_iter: -1	Recall: 0.9836
probability: False	Log loss: 0.9433
random_state: None	
shrinking: True	
tol: 0.001	

Table B.1.5.: Results of the ER predicting DNN model.

Hyperparameters	Metrics
batch_size: 256 dropout: 0.4 early_stopping: True learning_rate: 0.01 nb_epoch: 100 optimization: Adadelta batch_normalization: True patience: 80 units_in_hidden_layers: [1024, 128] units_in_input_layer: 5000	AUC: 0.9462 Accuracy: 0.9559 F1 score: 0.9710 MCC: 0.8789 Precision: 0.9778 Recall: 0.9644 Log loss: 1.5238

B.2 THREEGENE MODEL RESULTS

Table B.2.1.: Results of the THREEGENE predicting KNN model.

Hyperparameters	Metrics
algorithm: auto leaf_size: 30 metric: cityblock metric_params: None n_neighbors: 3 p: 2 weights: uniform	Accuracy: 0.6965 F1 score: 0.6766 MCC: 0.5879 Precision: 0.7308 Recall: 0.6965

Table B.2.2.: Results of the THREEGENE predicting LR model.

Hyperparameters	Metrics
C: 0.1 class_weight: None dual: False fit_intercept: True intercept_scaling: 1 max_iter: 100 multi_class: warn penalty: l1 tol: 0.0001 warm_start: False	Accuracy: 0.9129 F1 score: 0.9130 MCC: 0.8769 Precision: 0.9156 Recall: 0.9129

Table B.2.3.: Results of the THREEGENE predicting RF model.

Hyperparameters	Metrics
bootstrap: True	
class_weight: None	
criterion: gini	
max_depth: None	
max_features: auto	
max_leaf_nodes: None	
min_impurity_decrease: 0.0	Accuracy: 0.9271
min_impurity_split: None	F1 score: 0.9271
min_samples_leaf: 1	MCC: 0.8956
min_samples_split: 2	Precision: 0.9290
min_weight_fraction_leaf: 0.0	Recall: 0.9271
n_estimators: 500	
n_jobs: None	
oob_score: False	
random_state: None	
verbose: 0	
warm_start: False	

Table B.2.4.: Results of the THREEGENE predicting SVM model.

Hyperparameters	Metrics
C: 0.01	
cache_size: 200	
class_weight: None	
coef0: 0.0	
decision_function_shape: ovr	Accuracy: 0.9153
degree: 3	F1 score: 0.9155
gamma: auto_deprecated	MCC: 0.8791
kernel: linear	Precision: 0.9160
max_iter: -1	Recall: 0.9153
probability: False	
random_state: None	
shrinking: True	
tol: 0.001	

Table B.2.5.: Results of the THREEGENE predicting DNN model.

Hyperparameters	Metrics
batch_size: 64 dropout: 0.3 early_stopping: False learning_rate: 0.005 nb_epoch: 150 batch_normalization: False optimization: SGD patience: 80 units_in_hidden_layers: [2048, 1024, 512, 128] units_in_input_layer: 5000	Accuracy: 0.8894 F1 score: 0.8888 MCC: 0.8423 Precision: 0.8894 Recall: 0.8894

B.3 NPI MODEL RESULTS

Table B.3.1.: Results of the NPI predicting KNN model.

Hyperparameters	Metrics
algorithm: auto leaf_size: 30 metric: cityblock metric_params: None n_neighbors: 5 p: 2 weights: uniform	R^2: 0.0799 MAE: 0.8914 MSE: 1.2396

Table B.3.2.: Results of the NPI predicting EN model.

Hyperparameters	Metrics
fit_intercept: True	R^2: 0.0116
positive: True	MAE: 0.8770
selection: random	MSE: 1.3316

Table B.3.3.: Results of the NPI predicting RF model.

Hyperparameters	Metrics
bootstrap: True criterion: mse max_depth: None max_features: auto max_leaf_nodes: None min_impurity_decrease: 0.0 min_impurity_split: None min_samples_leaf: 1 min_samples_split: 2 n_estimators: 500	R^2: 0.2288 MAE: 0.8253 MSE: 1.0390

Table B.3.4.: Results of the NPI predicting SVM model.

Hyperparameters	Metrics
C: 0.001 cache_size: 200 coefo: 0.0 degree: 3 epsilon: 0.1 gamma: auto_deprecated kernel: linear shrinking: True tol: 0.001	R^2: 0.1477 MAE: 0.8545 MSE: 1.1482

Table B.3.5.: Results of the NPI predicting DNN model.

Hyperparameters	Metrics
batch_size: 64 batch_normalization: True dropout: 0.5 early_stopping: False learning_rate: 0.01 nb_epoch: 100 optimization: SGD patience: 80 units_in_hidden_layers: [2048, 128, 16] units_in_input_layer: 5000	R^2: 0.1604 MSE: 1.0848 MAE: 0.8169

B.4 Multi tasking DNN model results

Table B.4.6.: Results of the ER, HER-2 and PR multi-tasking DNN.

Hyperparameters	ER Results	HER-2 Results	PR Results
batch_size: 32 batch_normalization: True dropout: 0.5 early_stopping: True learning_rate: 0.005 nb_epoch: 200 optimization: SGD patience: 80 hidden_layers: [2048, 512] units_in_input_layer: 5000	AUC: 0.9475 Accuracy: 0.9281 F1_score: 0.9657 MCC: 0.8536 Precision: 0.9670 Recall: 0.9644 Log loss: 1.8140	AUC: 0.9790 Accuracy: 0.9549 F1_score: 0.9219 MCC: 0.9097 Precision: 0.9219 Recall: 0.9219 Log loss: 0.7256	AUC: 0.8361 Accuracy: 0.8343 F1_score: 0.8482 MCC: 0.6732 Precision: 0.8165 Recall: 0.8826 Log loss: 5.6598

C

DETAIL OF RESULTS - SECOND CASE STUDY

C.1 MELANOMA MODEL RESULTS

Table C.1.1.: Results of the cell malignancy predicting KNN model.

Hyperparameters	Metrics
algorithm: auto	AUC: 0.9678
leaf_size: 30	Accuracy: 0.9750
metric: cityblock	F1 score: 0.9803
metric_params: None	MCC: 0.9472
n_neighbors: 5	Precision: 0.9640
p: 2	Recall: 0.9971
weights: uniform	Log loss: 0.8635

Table C.1.2.: Results of the cell malignancy predicting LR model.

Hyperparameters	Metrics
C: 100 class_weight: None dual: False fit_intercept: True intercept_scaling: 1 max_iter: 100 multi_class: warn penalty: l1 random_state: None solver: warn tol: 0.0001 warm_start: False	AUC: 0.9986 Accuracy: 0.9982 F1 score: 0.9986 MCC: 0.9962 Precision: 1.0 Recall: 0.9971 Log loss: 0.0617

Table C.1.3.: Results of the cell malignancy predicting RF model.

Hyperparameters	Metrics
bootstrap: True	
class_weight: None	
criterion: gini	
max_depth: None	
max_features: auto	AUC: 0.9934
max_leaf_nodes: None	Accuracy: 0.9935
min_impurity_decrease: 0.0	F1 score: 0.9945
min_impurity_split: None	MCC: 0.9915
min_samples_leaf: 1	Precision: 0.9923
min_samples_split: 2	Recall: 0.9923
min_weight_fraction_leaf: 0.0	Log loss: 0.4323
n_estimators: 100	
oob_score: False	
random_state: None	
warm_start: False	

Table C.1.4.: Results of the cell malignancy predicting SVM model.

Hyperparameters	Metrics
C: 0.001	
cache_size: 200	
class_weight: None	
coef0: 0.0	AUC: 0.9962
decision_function_shape: ovr	Accuracy: 0.9964
degree: 3	F1 score: 0.9971
gamma: auto_deprecated	MCC: 0.9924
kernel: linear	Precision: 0.9971
max_iter: -1	Recall: 0.9971
probability: False	Log loss: 0.1234
random_state: None	
shrinking: True	
tol: 0.001	

Table C.1.5.: Results of the cell malignancy predicting DNN model.

Hyperparameters	Metrics
batch_size: 256 dropout: 0.5 early_stopping: False learning_rate: 0.015 nb_epoch: 150 optimization: Adam batch_normalization: True patience: 80 units_in_hidden_layers: [2048, 1024,512] units_in_input_layer: 5000	AUC: 0.9928 Accuracy: 0.9911 F1 score: 0.9928 MCC: 0.9813 Precision: 1.0 Recall: 0.9857 Log loss: 0.3084