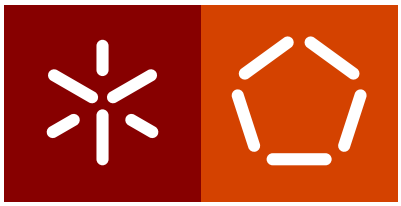
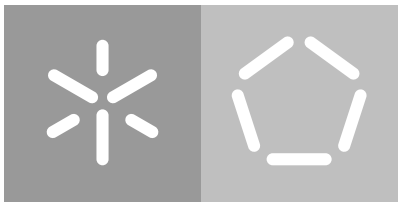


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Luís Miguel da Cunha Peixoto

Developing deep learning computational tools for cancer using omics data

January 2018



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Luís Miguel da Cunha Peixoto

Developing deep learning computational tools for cancer using omics data

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
Miguel Francisco de Almeida Pereira da Rocha
Pedro Gabriel Ferreira

January 2018

ACKNOWLEDGEMENTS

Deep learning is a fascinating field of machine learning, and I would like to thank in first place to Professor Miguel Rocha for giving me the advice and the opportunity to take my first step into the field. I am also thankful for his availability and time whenever I asked. I would also like to thank Pedro Ferreira for his support and advices that were precious to make a better job. Also his way of working and organization inspired me to achieve my goals. Thank you for receiving me on I3S and spend your precious time with me. To Vitor Vieira, thank you for your friendship, for helping me with all the technical stuff, from working with the cluster to the data wrangling of datasets. To Abel Sousa, thank you for being a big friend of mine and also for all the help with understanding the omics data and computational biology issues, and for the I3S reception. I also thank to Pedro Rosas for giving me the motivation even when it seemed impossible, and also for technical support. I also thank the entire University of Minho bioinformatics community and my friends. I remember my parents for supporting me over those intense months while working and studying. Thanks to my colleagues at the pharmacy for understanding all the troubles caused by schedule changes due to the meetings. And a last but intense thank you for Elodie, for understanding me and for giving me the emotional support I needed to pursuit my dreams.

ABSTRACT

There has been an increasing investment in cancer research that generated an enormous amount of biological and clinical data, especially after the advent of the next-generation sequencing technologies. To analyze the large datasets provided by omics data of cancer samples, scientists have successfully been recurring to machine learning algorithms, identifying patterns and developing models by using statistical techniques to make accurate predictions.

Deep learning is a branch of machine learning, best known by its applications in artificial intelligence (computer vision, speech recognition, natural language processing and robotics). In general, deep learning models differ from machine learning "shallow" methods (single hidden layer) because they recur to multiple layers of abstraction. In this way, it is possible to learn high level features and complex relations in the given data.

Given the context specified above, the main target of this work is the development and evaluation of deep learning methods for the analysis of cancer omics datasets, covering both unsupervised methods for feature generation from different types of data, and supervised methods to address cancer diagnostics and prognostic predictions.

We worked with a *Neuroblastoma (NB)* dataset from two different platforms (RNA-Seq and microarrays) and developed both supervised (*Deep Neural Networks (DNN)*, *Multi-Task Deep Neural Network (MT-DNN)*) and unsupervised (*Stacked Denoising Autoencoders (SDA)*) deep architectures, and compared them with shallow traditional algorithms.

Overall we achieved promising results with deep learning on both platforms, meaning that it is possible to retrieve the advantages of deep learning models on cancer omics data. At the same time we faced some difficulties related to the complexity and computational power requirements, as well as the lack of samples to truly benefit from the deep architectures.

There was generated code that can be applied to other datasets, wich is available in a github repository `https://github.com/lmpeixoto/deepl_learning` [49].

RESUMO

Nos últimos anos tem havido um investimento significativo na pesquisa de cancro, o que gerou uma quantidade enorme de dados biológicos e clínicos, especialmente após o aparecimento das tecnologias de sequenciação denominadas de "próxima-geração". Para analisar estes dados, a comunidade científica tem recorrido, e com sucessso, a algoritmos de aprendizado de máquina, identificando padrões e desenvolvendo modelos com recurso a métodos estatísticos. Com estes modelos é possível fazer previsão de resultados. O aprendizado profundo, um ramo do aprendizado de máquina, tem sido mais notório pelas suas aplicações em inteligência artificial (reconhecimento de imagens e voz, processamento de linguagem natural e robótica). De um modo geral, os modelos de aprendizado profundo diferem dos métodos clássicos do aprendizado de máquina por recorrerem a várias camadas de abstração. Desta forma, é possível "aprender" as representações complexas e não lineares, com vários graus de liberdade dos dados analisados. Neste contexto, o objetivo principal deste trabalho é desenvolver e avaliar métodos de aprendizado profundo para analisar dados ómicos do cancro. Pretendem-se desenvolver tanto métodos supervisionados como não-supervisionados e utilizar diferentes tipos de dados, construindo soluções para diagnóstico e prognóstico do cancro. Para isso trabalhámos com uma matriz de dados de Neuroblastoma, proveniente de duas plataformas diferentes (RNA-seq e microarrays), nos quais aplicámos algumas arquiteturas de aprendizado profundo, tanto como métodos supervisionados e não-supervisionados, e com as quais comparámos com algoritmos tradicionais de aprendizado de máquina. No geral conseguimos obter resultados promissores nas duas plataformas, o que significou ser possível beneficiar das vantagens dos modelos do aprendizado profundo nos dados ómicos de cancro. Ao mesmo tempo encontrámos algumas dificuldades, de modo especial relacionadas com a complexidade dos modelos e o poder computacional exigido, bem como o baixo número de amostras disponíveis. Na sequência deste trabalho foi gerado código que pode ser aplicado a outros dados e está disponível num repositório do github `https://github.com/lmpeixoto/deepl_learning` [49].

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

A

ACGH Array Comparative Genomic Hybridization.

AE Autoencoders.

ANN Artificial Neural Networks.

API Application Programming Interface.

AUC Area Under The Curve.

AUROC Area Under The ROC Curve.

B

BN Bayesian Networks.

C

CAD Computer Aided Diagnostic.

CAMDA Annual International Conference on Critical Assessment of Massive Data Analysis.

CGC Cancer Genomics Cloud.

CNA Copy Number Alteration.

CNN Convolutional Neural Networks.

CNV Copy Number Variation.

CPU Central Processing Unit.

CSV Comma Separated Values.

CUDA Compute Unified Device Architecture.

CUDNN NVIDIA CUDA Deep Neural Network library.

D

DA Denoising Autoencoders.

DBM Deep Boltzman Machines.

DBN Deep Belief Networks.

DI Departamento de Informática.

DNA Deoxyribonucleic Acid.

DNN Deep Neural Networks.

DT Decision Trees.

E

ENCODE The Encyclopedia of DNA Elements.

F

FPR False Positive Rate.

G

GB Gigabytes.

GDC Genomic Data Commons.

GEO Gene Expression Omnibus.

GPU Graphics Processing Unit.

H

HMM Hidden Markov Models.

I

ICGC International Cancer Genome Consortium.

K

KNN K-Nearest Neighbors.

L

LDA Linear Discriminant Analysis.

LR Logistic Regression.

M

MA Microarrays.

MAD Mean Absolute Deviation.

MCC Mathews Correlation Coefficient.

MRNA messenger Ribonucleic Acid.

MSE Mean Squared Error.

MT-DNN Multi-Task Deep Neural Network.

N

NB Neuroblastoma.

NCI National Cancer Institute.

NGS Next Generation Sequencing.

NIH National Institutes of Health.

NLP Natural Language Processing.

P

PCA Principal Component Analysis.

R

RBM Restricted Boltzman Machines.

RELU Rectified Linear Unit.

RF Random Forests.

RNA Ribonucleic Acid.

RNN Recurrent Neural Networks.

ROC Receiver Operating Characteristic.

RPKM Reads Per Kilobase Million.

S

SA Stacked Autoencoders.

SDA Stacked Denoising Autoencoders.

SGD Stochastic Gradient Descent.

SVM Support Vector Machines.

T

TANH Hyperbolic Tangent.

TARGET Therapeutically Applicable Research to Generate Effective Treatments.

TCGA The Cancer Genome Atlas.

TPR True Positive Rate.

U

UM Universidade do Minho.

V

VA Variational Autoencoders.

1

INTRODUCTION

1.1 CONTEXT AND MOTIVATION

Over the last decades, cancer has become one of the major concerns of medicine. This disease lead to an enormous amount of deaths and elevated costs in healthcare systems all over the world. In 2016, more than 1.5 million Americans were expected to be diagnosed with cancer, while over half a million were expected to die of the disease. Cancer remains the second most common cause of death, surpassed only by cardiovascular causes [5].

Although these facts are by themselves overwhelming, medical sciences are evolving. Much progress has been made against cancer. Both the treatments and earlier diagnostic are crucial checkpoints to succeed in the cancer battle. Returning to numbers, more than 1.7 million deaths were avoided due to improvements in those areas in last decade [5].

It is easier to treat a patient when the cancer is still well circumscribed. When it is metastasized to other organs, the success is lower and will depend on the effectiveness of treatments which are typically more agressive and less efective. When more aggressive treatments are used, there are also more adverse effects and more costs. Earlier diagnostic is, thus, probably the key success factor to avoid the huge number of deaths by cancer.

There were large scale efforts lead by international consortia in order to better characterize and understand the etiology of cancer. That generated an enormous amount of biological and clinical data, especially after the advent of the *Next Generation Sequencing (NGS)* technologies. *The Cancer Genome Atlas (TCGA)*, and later *Genomic Data Commons (GDC)*, are examples of huge databases that contain genomic measurements from more than 20 different types of human cancer. GDC contains almost 5 petabytes of information [32, 58]. But a question arises, is it better to invest in producing more data, or analyzing the available data?

NGS originates what is called "big data". There are two types of data - the raw data that outputs directly from the sequencing machine (i.e. sequence reads), and the processed data (i.e. gene expression data). Raw data is several folds bigger than processed data. For gene expression data, and even more for genome data, those datasets are normally huge, dense and hard to analyze. They have more features (genes) than samples, which makes

it difficult to train and produce a correct model for data prediction. Data analysis of gene expression has made extensive use of machine learning methods. With pattern recognition, using mathematical and statistical foundations, coupled with a computer science approach, scientists are producing models that can be used in several biological problems.

Predicting if a sample is normal or cancerous, or classifying the tumor by stage are two examples, while another scenario is fitting the right treatment for a given patient profile. This leads us to the concepts of personalized or precision medicine, which fits treatments to specific patient characteristics [50]. Although there are no exact methods to solve these problems, with machine learning one can create very accurate models that can be used to answer very defined biological and clinical problems.

Over the last years, as a result of the increasing amounts of available data (big data), more computational power, and because of the development of tools like Theano or TensorFlow, deep learning techniques have been used widely, especially in what concerns to computer vision, speech recognition and natural language processing. Intelligent solutions are now becoming ubiquitous. They are present in our smartphones, applications like Facebook, Twitter, Google, Android or Apple iOS ecosystems, online and physical shops like Amazon, and much more. Research teams like Google and Baidu have reached solutions that even outperform humans in some tasks (i.e. playing games with reinforcement learning [53]). The fact is that deep learning is offering the human being a possibility to live a better life.

One of the most promising areas of application of these new technologies is biological and biomedical research. By developing more accurate models that will help the process of decision making, we hope to have an improvement in every sense in quality of life [39]. In cancer genomics, although deep learning is still a new area of research, there have already been published studies which demonstrate it can achieve better results than classic machine learning implementations [15, 23]. For instance, the *Variational Autoencoders (VA)* from Way et al. [63] and *Deep Neural Networks (DNN)* from Leung et al. [42] are good examples of that.

1.2 OBJECTIVES

The objectives of this project are to develop both unsupervised and supervised deep learning methods to deal with cancer omics data and validate these methods with selected case studies involving real data. As aforementioned, the datasets we will work on, have more features than samples. With regard to gene expression data we can have a dataset with more than 20.000 genes (features) and only 500 cases. So, it is an important step to reduce the dimensionality of the data and work on a subsequent predictive analysis.

Using unsupervised models, we expect to extract relevant features from the different types of omics data (e.g. genomics, gene expression, epigenomics).

The supervised approach is expected to produce pipelines for cancer diagnosis or prediction using the available data and evaluating the different alternatives based on the defined criteria.

The final objective will be addressing specific case studies for the validation of the methods, including TCGA/GDC real data, building models, and generating and submitting predictions for unknown data.

1.3 ORGANIZATION OF THE TEXT

- MACHINE LEARNING AND DEEP LEARNING: A STATE OF THE ART - the purpose of this chapter is to make an introduction of basic machine learning concepts and an overview of some of the most popular shallow machine learning algorithms. There will be presented the *Artificial Neural Networks (ANN)* principles as well as the backpropagation and *Stochastic Gradient Descent (SGD)* algorithm. Then it will be explained some deep learning specific concepts, a small sample of supervised and unsupervised learning architectures, and the libraries available for their construction.
- CANCER OMICS - here we will explain some of the most used databases of cancer omics and its objectives. We will also approach the applications of deep learning in cancer omics and cite some of the most important works made in this field.
- METHODS - this chapter describes our experiments and the pipelines chosen, as well as the written code and the configurations made.
- RESULTS AND DISCUSSION - all the results and their analysis and discussion are presented here.
- CONCLUSION - this is the final chapter and the objective is to give an overview of the entire work as well as to share some insights and experiences achieved.

2

MACHINE LEARNING AND DEEP LEARNING: A STATE OF THE ART

2.1 MACHINE LEARNING FUNDAMENTALS

Machine learning is a subfield of computer science and mathematics. Arthur Samuel, in 1959, described it as a "field of study that gives computers the ability to learn without being explicitly programmed" [21]. In other words, using a computer and recurring to statistical and mathematical techniques, machine learning explores the construction of algorithms that can learn from input data and make predictions [9].

There are basically three categories of machine learning methods: unsupervised learning, supervised learning and semi-supervised learning. Supervised learning refers to a process of inferring a function from labeled training data. On the other hand, unsupervised learning includes methods that return new features or patterns from unlabeled data only. Semi-supervised learning uses unsupervised methods to improve supervised algorithms [9, 36, 39].

Table 1.: Comparison between supervised and unsupervised learning. Adapted from *Python machine learning*, by Sebastian Raschka, 2015 [52].

Supervised Learning	Unsupervised Learning
Labeled data	No labels
Direct feedback	No feedback
Predict outcome	Find hidden patterns

2.1.1 *Supervised Learning*

Let us consider a machine that receives some sequence of inputs $(x_1, x_2, ...x_n)$, what we call data. In supervised learning, the machine is given a sequence of outputs $(y_1, y_2, ...y_n)$, and the goal is to correctly predict an output given a new input.

Supervised learning refers to constructing a model from labeled data to predict new unlabeled cases. For this type of predictive analysis, the model can be built to address two types of tasks: regression and classification. Regression models make predictions regarding

continuous variables (e.g. prediction of death rate in function of pollution variables), while classification models assign discrete class labels to observations as outcomes of a prediction (e.g. prediction of cancer sub-type based on tumor size and characteristics).

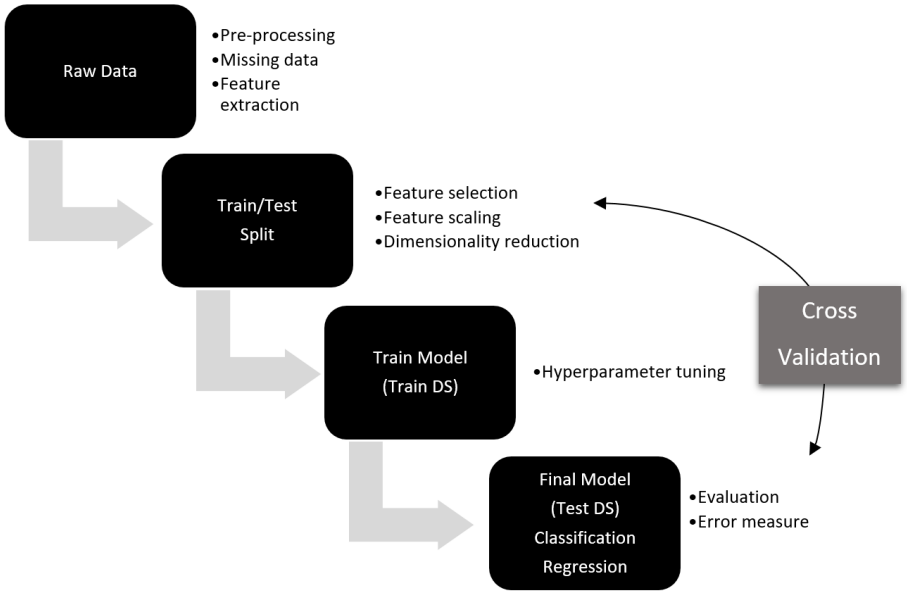


Figure 1.: Supervised learning pipeline. Adapted from *Python machine learning*, by Sebastian Raschka, 2015 [52].

The flowchart of Figure 1 depicts a common pipeline for supervised learning classification or regression. The first step is to collect and process the raw data. If an input is provided with missing data, many machine learning algorithms might fail. In this stage, tasks include feature extraction, handling missing data and creating a dataset with comparable information [52]. The second step will be to split the dataset into training and test subsets. The training dataset is used to train the model, and the test dataset is used to evaluate the performance of the model. This division is essential to avoid overfitting. Testing with instances that were never shown to the model guarantees that evaluation is well done.

Afterwards, we preprocess the data performing tasks as feature normalization, feature selection and dimensionality reduction, depending on the type of data at hand. The normalization rescales the attributes in a certain range, making them comparable, especially when they are measured in different scales. Feature selection will lead to a reduction of dimensionality. The objective is to remove noise and increase computational efficiency by keeping only "useful" features that will give insight into the prediction.

An important task is to select the algorithm hyperparameters that best adjust to our model. To optimize both parameters or feature sets, we need to run our algorithm several

times with different configurations and evaluate the error for each. This is achieved by shuffling the samples, to reduce the sampling bias, performing either a resampling or a cross validation estimation. The next step will be testing our algorithm with the test dataset and then performing an evaluation recurring to error estimation.

Some of the most used supervised machine learning methods for classification are *Logistic Regression (LR)*, *Support Vector Machines (SVM)*, *Decision Trees (DT)*, *Bayesian Networks (BN)* and ANN [36, 52].

Logistic Regression (LR)

LR is a binary classification model, used to classify examples in two different classes. Equation 1 represents the logistic or sigmoid function, used in LR as a probability function, used to calculate the probability of labeling an example as positive when applied to a linear combination of a set of weights (model parameters) multiplied by the input feature vector (Equation 2). The parameters θ are estimated minimizing the error function $J(\theta)$ (Equation 3) using the gradient descent method, which will be addressed in a posterior section [25, 52].

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

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} \tag{2}$$

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))] \tag{3}$$

Decision Trees (DT)

DT are discrete classifiers where nodes represent the input variables, branches represent their possible values, and the leaves represent the outcomes (classes). DT are one of the earliest and most used methods for classification. Based on the architecture of the DT, they are simple to interpret and fast to learn through top-down algorithms as ID3 and its successors. We can traverse the tree to interpret the result and find about its class. The decisions that result from their architecture allow for adequate reasoning which makes them an appealing technique [25, 52].

Support Vector Machines (SVM)

SVM are a more recent machine learning method. Initially, SVM map the input vector into a feature space of higher dimensionality and identify the hyperplane that separates the data points into two classes. The marginal distance between the decision hyperplane and the instances that are closest to boundary is maximized. The resulting classifier achieves

considerable generalization capabilities and can, therefore, be used for the reliable classification of new samples. It is worth noting that probabilistic outputs can also be obtained for SVM. The identified hyperplane can be thought as a decision boundary between the two classes. Obviously, the existence of a decision boundary allows for the detection of any misclassification produced by the method [25, 52].

Bayesian Networks (BN)

BN classifiers produce probability estimations rather than predictions. As their name reveals, they are used to represent knowledge coupled with probabilistic dependencies among the variables of interest via a directed acyclic graph. BN have been applied widely to several classification tasks as well as for knowledge representation and reasoning purposes [38].

2.1.2 *Unsupervised Learning*

In unsupervised learning, sometimes called representation learning [7], the machine simply receives the inputs x but obtains neither target outputs, nor rewards from its environment. The unsupervised learning approach instead of making predictions, tries to "understand" the data searching for patterns. Because it does not require labeled data, the algorithm is not searching for something specific, but rather exploiting the structure of the data. It is the analysis of the results, taking context into account, that will decide whether the patterns extracted are meaningful or not.

The unsupervised learning pipeline (Figure 2) is simpler than the supervised one because there is no training/test and, therefore, the objective here is to validate the extracted knowledge according to our data structure and meaning.

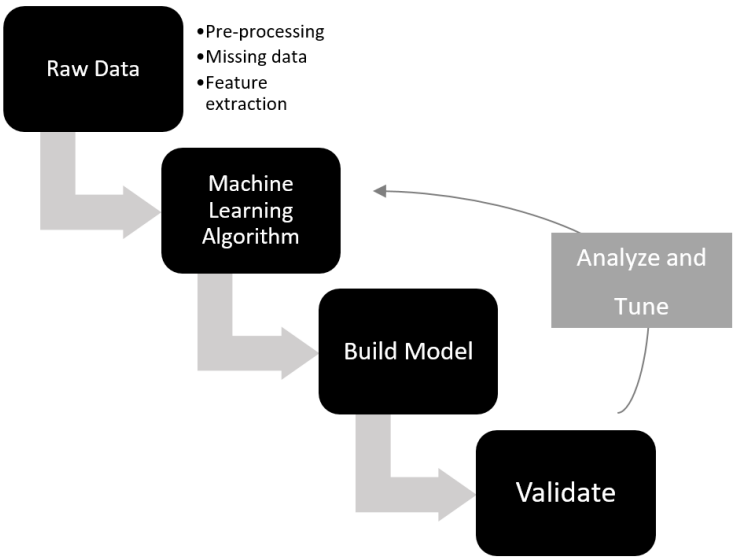


Figure 2.: Unsupervised learning pipeline. Adapted from *Python machine learning*, by Sebastian Raschka, 2015 [52].

Some popular examples of unsupervised learning algorithms are *k-means clustering*, *hierarchical clustering*, *Principal Component Analysis (PCA)*, *Hidden Markov Models (HMM)* or *Restricted Boltzman Machines (RBM)*. There are two main usages of unsupervised learning - clustering and dimensionality reduction [9, 25, 52].

Clustering

Clustering algorithms try to discover hidden structures in the data and group them together in clusters, based on data features. It can be achieved by various algorithms like the Lloyd algorithm, which assumes the *k-means clustering* cost function, which minimizes the distances of the points to the assigned clusters assuming an Euclidean space, being the most widely used clustering algorithm for continuous data.

In *k-means clustering* algorithm, k is the number of clusters we want to find. The cluster centroids μ_j represent the guesses for the positions of the clusters center. In the initialization, we choose k training examples randomly, and set the centroid values to be equal to them. Then, we assign each training example $x^{(i)}$ to the closest centroid μ_j. The next step is to move each centroid to the mean of the points that belong to them and this is repeated iteratively until no further movements are possible.

In *k-means*, the performance depends on the starting configuration - number of centroids.

On the other hand, *hierarchical clustering* does not require to specify the number of clusters as an input. Instead, there is a measure of dissimilarity between (disjoint) groups of

observations. They produce hierarchical representations in which the clusters at each level of the hierarchy are created by merging clusters at the lower level. At the lowest level, each cluster contains a single observation. At the highest level there is only one cluster containing all of the data.

They can be agglomerative (bottom-up) and divisive (top-down). Agglomerative strategies start at the bottom and, at each level, recursively merge a selected pair of clusters into a single cluster. This produces a grouping at the next higher level with one less cluster. The pair chosen for merging consists of the two groups with the smallest dissimilarity. Divisive methods, on the other hand, start at the top and at each level recursively split one of the existing clusters at that level into two new clusters. The split is chosen to produce two new groups with the largest dissimilarity. With both paradigms there are $N - 1$ levels in the hierarchy. Clustering can be very useful for exploratory data analysis and data visualization [25].

PCA and Feature Extraction

The set of features selected by a feature selection method will always be a subset of the original set of features. On the other hand, the set created by a feature extraction method does not have to. For instance, PCA reduces dimensionality by making new latent features from linear combinations of the original ones, and then discarding the less important ones.

The main purposes of a PCA process are the analysis of data to identify and find patterns to reduce the dimensions of the dataset (d) with minimal loss of information. Defining k as the number of dimensions of the new feature subspace, where $k \leq d$, we can summarize the PCA algorithm in five steps:

- Normalize the data (because we will calculate eigenvalues, it is convenient that data is normalized);
- Calculate covariance matrix;
- Choose k eigenvectors that correspond to the k largest eigenvalues;
- Construct the projection matrix W from the selected k eigenvectors;
- Transform the original dataset X via W to obtain a k-dimensional feature subspace Y.

Here, our desired outcome of the PCA is to project a feature space onto a smaller subspace that represents better our data removing co-linearity in the variables, as the new ones are orthogonal. One possible scenario for using PCA would be a classification task where we would want to reduce the dimensions of our feature space. This would reduce the error of parameter estimation and also the computational cost [52].

2.1.3 *Feature Selection*

In machine learning and statistics, both feature selection and feature extraction make what is called dimensionality reduction. Sometimes it is also useful to create new features. Feature selection is the term used for selecting a subset of features for the model construction. There are three major methods to select features - filter methods, wrapper methods and embedded methods. The wrapper methods measure the "usefulness" of features based on the classifier performance. The filter methods are methods that select properties of the features, e.g. statistically measured, used to rank the features, being those measures independent of specific classifiers.

Wrapper methods are essentially optimizing the classifier performance, but they are also computationally more expensive compared to filter methods due to the repeated learning steps and cross-validation. Embedded methods are similar to wrapper methods, because they are also used to optimize the objective function or performance of a learning algorithm or model. The difference to wrapper methods is that an intrinsic model building metric is used during learning [52, 25].

2.1.4 *Error Estimation*

The generalization performance of a learning method relates to its prediction capability on independent test data. The assessment of model's performance is extremely important because it helps to choose the learning method or model, and also gives a measure of the quality of the model [25].

Cross-validation

Probably the simplest and most widely used method for estimating prediction error is cross-validation. This method directly estimates the average generalization error when the model is applied to an independent test sample [25].

In K-Fold cross-validation, all the samples are divided in k groups, called folds, of equal sizes, if possible. For prediction we use $k-1$ folds, and the other fold is used for testing. The overall error is the average of the measures on the n test sets (one for each fold).

Leave One Out is another cross-validation method where each learning set is created by taking all the samples except one, being the test set the sample left out. Thus, for n samples, we have n different training and test sets. This procedure does not waste much data as only one sample is removed from the training set. In K-Fold, if $k = n$, the method is equivalent to Leave One Out [25].

Model Evaluation

Consider T to be the training set, and each example represented as (x, y). In binary (hard) classification, y consists of a "positive" (typically represented as 1) and "negative" (represented as 0) labels. Each classifier c segments a set of examples with known label into four partitions, based on both the true label y_i and the predicted label $c(x_i)$ for each example $(x_i, y_i) \in T$. We will refer to true positives ($y_i = c(x_i) = 1$) as TP, false positives ($y_i = 1$ but $c(x_i) = 0$) as FP, false negatives ($y_i = 0$ but $c(x_i) = 1$) as FN, and true negatives ($y_i = c(x_i) = 0$) as TN. The accuracy of a classifier over the set of examples is defined as: $(TP + TN)/(TP + TN + FP + FN)$. For accuracy we mean $P_{(x,y)\sim D}(c(x) = y)$, where D is an unknown distribution that determines the probability or density to sample a specific example x with a label y [24].

The precision (Pr) and recall (Re) of a classifier are given by Equations 4 and 5.

$$Pr := TP/(TP + FP) \tag{4}$$

$$Re := TP/(TP + FN) \tag{5}$$

The F1-measure combines these two into a single number, which is useful for ranking or comparing methods. Formally, it is the harmonic mean between precision and recall being defined by equation 6 [24].

$$F := 2 \cdot \frac{Pr \cdot Re}{Pr + Re} \tag{6}$$

$$FPR := FP/(FP + TN) \tag{7}$$

The recall is also names as *True Positive Rate (TPR)* or sensitivity (Equation 5). The *False Positive Rate (FPR)*, or $1-$specifity, is given by Equation 7. To combine the TPR and FPR into a single metric we can plot them on a single graph, with the FPR on the abscissa and the TPR values on the ordinate. The resulting curve is called *Receiver Operating Characteristic (ROC)* curve, and the metric we consider is the *Area Under The Curve (AUC)* of this curve, which we call *Area Under The ROC Curve (AUROC)*. Note that a given classifier is represented by a dot on this graph; a graph can be plotted if the classifier has a parameter that can be changed to consider different trade-offs of sensititivity and specificity, which is true for all binary classifiers that allow to estimate positive label probabilities.

When comparing two models based on the AUROC, if the two curves do not cross each other, the ROC curve with the greatest value of the index (Figure 3, red curve) corresponds to the system which presents a better performance, i.e. a greater discriminant power [10].

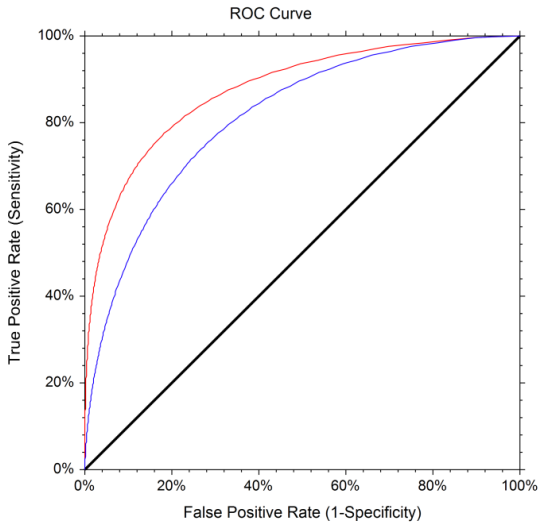


Figure 3.: Roc Curve.

2.1.5 *Bias-Variance Decomposition*

One of the major problems in machine learning is overfitting, especially when we have both complex models and limited data. Overfitting happens when the model is too complex, fitting the training data too well, and hence the model has a poor predictive performance over the test set with unseen examples.

More generally, in machine learning, there is a bias-variance trade-off. If our model has low variance and high bias, the average is far from the correct solution and it is not a good classifier, meaning that the model has poor capacity (or complexity) to learn the problem, or underfitting. To correct this situation we can increase the model complexity (e.g. increasing the number of parameters).

When it has high variance and low bias, some of the predictions might be far from correct value, although the average is near the correct solution. In this case, our model is overfitting and learning noise, and not meaningful patterns. The solutions can be to limit the model complexity (e.g. reducing the number of parameters, or by some kind of regularization, that will be exploited next) or to increase the number of data cases.

The objective is to have a low variance and low bias model. It is a very delicate process and, to detect such situation, we must ensure that the trained model does not vary a lot if the training set changes and the average model is close to the true solution [25]. In practice, it is typically impossible to guarantee that we find the best model, but error estimation of different complexity degrees can provide models with good performance in practice.

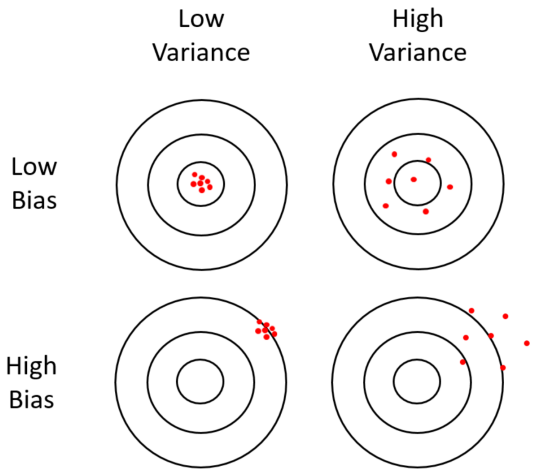


Figure 4.: Bias-Variance Trade-off.

2.1.6 *Regularization*

Regularization is a technique used in an attempt to solve the overfitting problem in statistical models. As previously mentioned, when our model is overfitting, it fits too much to the training data and the model has probably learned the background noise while being fit.

For n features and m training examples, given a set of points $(x_1, y_1), (x_2, y_2)...(x_n, y_n)$, our goal is to fit a function $y = f(x)$ to the given data. Let us call this a hypothesis $h\theta(x)$ (Equation 8), where *theta* are the model parameters.

$$h\theta(x) = \sum_{j=0}^{n} \theta_j x_j \tag{8}$$

For each example i, where x is a vector $[x_1, x_2...x_n]$, we are trying to find values of θ so that the function outputs a value close to given y values. Using *Mean Squared Error (MSE)* to compute our cost function J, we have:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \tag{9}$$

There are two major regularization methods - L1 and L2 [47].

L2 Regularization - Ridge

The L2 regularization, also known as Ridge, penalizes large individual weights. It adds an extra term to the cost function, which performs the sum of the squares of all the parameters (Equation 10). This is multiplied by a factor (regularization parameter) λ, where $\lambda > 0$ [25].

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 + \lambda\sum_{j=1}^{n}\theta_j^2 \tag{10}$$

L1 Regularization - Lasso

L1 regularization comprises adding the sum of the absolute values of the parameters (Equation 11). It is similar to L2 regularization, penalizing large parameters, but the way the parameters shrink during training is different. While in L1 regularization, the parameters shrink by a constant amount toward 0, in L2 regularization, the parameters shrink by an amount which is proportional to θ.

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

With Ridge regularization, the coefficients cannot be zero, so it is computationally more efficient in non-sparse cases due to having analytical solutions. In contrast, the Lasso does both parameter shrinkage and variable selection automatically and can yield sparse models, while Ridge cannot [25].

2.2 Artificial Neural Networks (ANN)

2.2.1 *Perceptrons and neurons*

The perceptron were the first forms of neurons. These are binary classifiers, represented by a function $f(x)$, defined by Equation 12, that maps the binary input x, which can assume the value of 0 or 1, to a binary output (Figure 5). Here, w is a weight vector, and b is the bias, that shifts the decision boundary from origin, and does not depend on the input value. We can think of the bias as a measure of how easy it is to get the perceptron to output 1.

$$f(x) = \begin{cases} 1 & \text{if} \quad (w \cdot x + b) > 0 \\ 0 & \text{else} \end{cases} \quad , \forall x \in \{0,1\} \tag{12}$$

The perceptron classification is based on the dot product between the examples and w (Equation 13), where m is the total number of weights, which will be summed with the bias.

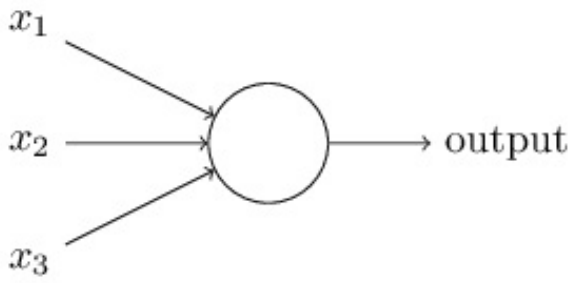


Figure 5.: Perceptron unit.

Perceptrons can be used to make decisions, 0 or 1, based on their constraints. They can also be used to compute elementary logical functions such as AND, OR, and NAND. But, since a perceptron can only perform linear operations through its input space, there are some categorizations that are impossible to do. One of the simplest examples of a non-linearly separable problem is XOR, or "exclusive or" (equivalent to OR and NOT AND). If one perceptron can solve OR and one perceptron can solve NOT AND, then two perceptrons combined can solve XOR.

Adding layers of neurons allows the network to break down the problems into sub-problems, and then combine the results later. So, constructing NAND gates trough neural networks it is possible to build any computation because the NAND gate is universal for computation [47].

$$z = \sum_{i=0}^{m} w_i x_i + b_i \tag{13}$$

In ANN, we have a neuron as elementary unit, which is similar to a perceptron, but neurons have an x input that can assume a real value. The output of a neuron is not 0 or 1, but instead goes through an activation function $\phi(z)$, where z is the net input. The neuron reads the information on vector x and performs a computation on two steps – pre-activation (Equation 14), and output activation (Equation 15).

$$a(x) = b + \sum_{i} w_i x_i = b + w^T x \tag{14}$$

$$h(x) = \phi(a(x)) = \phi(b + \sum_{i} w_i x_i) \tag{15}$$

The activation function ϕ of an ANN is usually the *sigmoid*, *Rectified Linear Unit (ReLU)*, *Hyperbolic Tangent (TANH)* or *softmax* functions. There are differences between these activation functions, especially in what concerns to the squashing of pre-activation values, range and bounds as we can see in Table 2.

Table 2.: Comparison between activation functions for ANN.

Act. Function	Squashing	Range	Bounds
Linear	No squashing	NA	NA
TANH	Between -1 and 1	Positive or negative	Bounded
Sigmoid	Between 0 and 1	Always positive	Bounded
ReLU	Below 0 by 0	Always non-negative	Not upper-bounded

Assuming a sigmoid neuron, we have that $\phi(z)$ is the sigmoid function (Equation 1 and 16).

$$\phi(z) = \frac{1}{1 + e^{-(\sum_i w_i x_i + b)}} \qquad , \forall\ x \in \Re \tag{16}$$

This sigmoid neuron produces an output between 0 or 1 being useful when we are facing a classification problem.

2.2.2 *Feedforward neural networks*

We can couple several neurons to create a layer of neurons. Furthermore, we can stack these into a network of neurons, a feedforward ANN (6). The first layer is the input layer, where we have a bunch of neurons that receive the inputs to the training algorithm. The last layer is called the output layer, where we have a number of neurons equal to the number of output variables (e.g. in classification can be the number of classes), that return a numerical value that can be interpreted in different ways for classification or regression. The layers between the input and output are called hidden layers. In Figure 6, we can see an ANN with 3 layers, one input layer with 3 units x_i, one hidden layer with 4 units and an output layer with 3 units y_i.

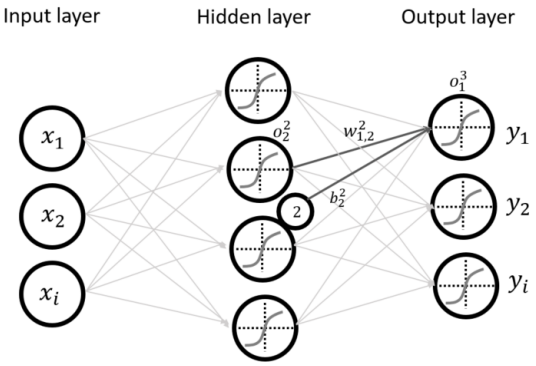


Figure 6.: Artificial Neural Network (ANN).

The Universal Approximation Theorem states that "a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units" [33].

Generalizing, we have a target function $y = f^*(x)$ that we want to learn. Our model provides a function $y = f(x;\theta)$, and our learning algorithm will adapt the parameters θ to make f as similar as possible to f^*.

To check how the output of a feedforward neural network is computed, let us start by defining the notation. With several layers, W^l will be a matrix of weights for every layer l. The symbolic representation w^l_{jk} refers to the weight of layer l, for the connection of the k^{th} neuron of layer $(l-1)$ to the j^{th} neuron in the layer l. We use b^l_j to represent the bias of j^{th} neuron in the layer l (thus, b^l will be a vector with the biases of layer l), and a^l_j represents the output of j^{th}neuron in the layer l (Equation 17), and thus a^l is a vector with the outputs of the neurons in layer l. Thus, the activation function equation of each neuron is given by:

$$a^l_j = \phi(\sum_k w^l_{jk} a^{l-1}_k + b^l_j) \tag{17}$$

A vectorized form (Equation 18) of this equation is provided by:

$$a^l = \phi(W^l a^{l-1} + b^l) \tag{18}$$

To compute the output of an ANN, this expression is applied for all layers of the network, from the input layer to the output layer.

2.2.3 *Training the network: gradient descent*

For most nonlinear problems, there is no way to solve for w and b in closed form (taking the derivative of the cost function and solving via Calculus). We must instead optimize our objective function using a method called gradient descent. In this method, we "travel" along the gradient of J, the cost function, with respect to w and b, until we hit a minimum.

We can define our optimization problem as an empiric risk minimization as expressed in Equation 19, where by θ we mean all the parameters, weights and biases, l is the loss function, x^t is a training example with y^t as the output, $\Omega(\theta)$ is the regularization term, and λ is the hyperparameter control term, that achieves the balance between optimizing the average loss and the regularization term.

$$\min_\theta \quad \frac{1}{T}\sum_t l(f(x^t;\theta), y^t) + \lambda\Omega(\theta) \tag{19}$$

In SGD (Algorithm 1) the parameters θ are updated towards the opposite sign of the calculated gradient ∇ with an α learning rate which dictates the step of update. The opti-

mization can update the weights after each training case (online learning), after a full sweep through the training data (full batch) or after a small sample of training cases (mini-batch). Also, the learning rate α can be fixed or adapted during the optimization process [30].

Algorithm 1 SGD.

1: Initialize the parameters θ
2: **for** N iterations (epochs):
3: **for** each training example $(x^{(t)}, y^{(t)})$
4: $\Delta = -\nabla_\theta l(f(x^{(t)};\theta), y^{(t)} - \lambda\Omega(\theta)$
5: $\theta \leftarrow \theta + \alpha\Delta$

2.2.4 *The Backpropagation algorithm*

The basis of backpropagation is how changing the weights and biases of the network changes the cost function. For that, we must compute the partial derivatives of the cost function in order to the weights and the biases, $\partial J/\partial w^l_{jk}$ and $\partial J/\partial b^l_j$ respectively. Based on the chain rule of derivatives, a small change in x (Δx) gets transformed into a small change in y (Δy) by getting multiplied by its partial derivative $\partial y/\partial x$. It also works when x, y and z are vectors [39]. For regression, we use the MSE function as our measure of fit. For classification we use either MSE or cross-entropy [25].

We can measure the error of each neuron by Equation 20 using a quadratic cost function, or MSE, where L is the number of layers of the neural network, y_j is the output of each j training example, and a^L_j is the vector of activations where the input is j.

$$J = \frac{1}{2}\sum_j (y_j - a^L_j)^2 \tag{20}$$

The error of each neuron is given by Equation 21:

$$\delta^l_j = \frac{\partial J}{\partial z^l_j} \tag{21}$$

The error in the output layer is represented by Equation 22. $\partial J/\partial a^L_j$ measures how fast the cost is changing as a function of the j^{th} output activation. If the cost function J does not depend much on an output j neuron, then δ^L_j will be small. The second term $\phi\prime(z^L_j)$ measures how fast the activation function ϕ is changing at z^L_j.

$$\delta^L_j = \frac{\partial J}{\partial a^L_j}\phi\prime(z^L) \tag{22}$$

We can also write that in the vectorial form (Equation 23), where $\nabla_a J$ is a vector of the partial derivatives $\partial J/\partial a^L_j$, which expresses the rate of change of J with respect to output

activations. "$\odot$" is the Hadamard product, which means the elementwise product between the two members. Now, we must calculate the partial derivatives of the next layer, bias and weights parameters. Calculating all the gradients, we have the following equations:

- Output layer

$$\delta^L = \nabla_a J \odot \phi\prime(z^L) \tag{23}$$

- Next layer

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \phi\prime(z^l) \tag{24}$$

- Biases

$$\partial J / \partial b_j^l \tag{25}$$

- Weights

$$\partial J / \partial w_{jk}^{l-1} \delta_j^l \tag{26}$$

We can also use the cross entropy (negative log-likelihood) formula, as stated in Equation 27, where n is the total number of items of training data, the sum is over all training inputs, x, and y is the corresponding desired output. This equation is positive and tends toward zero as the neuron gets better at computing the desired output y. It also has the property of avoiding the learning slowdown [47].

$$J = -\frac{1}{n} \sum_x [y \ln a + (1-y) \ln (1-a)] + \lambda \Omega(\theta) \tag{27}$$

2.2.5 *Improvements on training algorithms*

L1 and L2 regularization

L1 regularization comprises adding the sum of the absolute values of the weights, which in ANNs is achieved by Equation 28.

$$J = -\frac{1}{n} \sum_x [y \ln a + (1-y) \ln (1-a)] + \frac{\lambda}{2n} \sum_w |w| \tag{28}$$

The L2 regularization (Equation 29) is also known as weight decay because it is only applied on weights, not on biases. It adds an extra term to the cost function, which performs

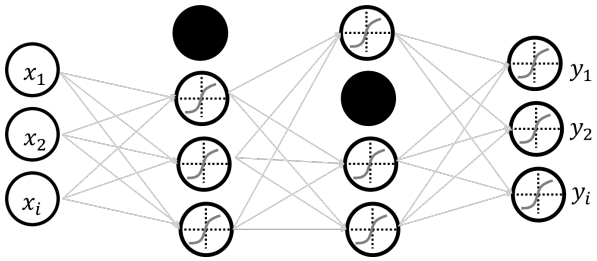


Figure 7.: Dropout regularization.

the sum of the squares of all the weights in the network. This is multiplied by a factor (regularization parameter) of $\lambda/2n$, where $\lambda > 0$ and n is the size of our training set [47].

$$J = -\frac{1}{n}\sum_{x}[y \ln a + (1-y)\ln(1-a)] + \frac{\lambda}{2n}\sum_{w} w^2 \tag{29}$$

Dropout

As we can see in Figure 7, dropout regularization is a method that modifies the neural network structure. We remove hidden units stochastically, where each hidden unit is set to 0 with a p probability. In this way, these hidden units (black) cannot co-adapt to other units, suppressing the connections (weights) between them. As result, hidden units become more generally useful, producing more robust models.

Early stopping

Early stopping is a hyperparameter used with iterative algorithms like SGD, which allows the optimization process to stop before the model starts to overfit the training data. We can choose the number of epochs (N) that can be run before the model begins to overfit. At the end of each N epochs, we evaluate the error on the test set and if we have a better performance we save that network, otherwise we continue. The final model will be the one that achieved better performance.

Parameter Initialization

The initialization of weights and biases can influence in a large extent the learning capacity of the neural network. Biases can all be initialized to zero, but that is not possible for the weights, particularly with a TANH activation function which will produce gradients of zero. The weights cannot be initialized to the same value because in that way, all the hidden units in a layer will behave the same, so we need to break that symmetry. A very powerful solution is to initialize all the weights within a uniform distribution following the Equation 30, where $U[-b, b]$ is the uniform distribution in the interval $[-b, b]$ and H_k is the number

of columns of W, the size of the previous layer. Equation 31 presents a common solution for the TANH activation function units [29].

$$W_{i,j} \sim U[-b, b] \tag{30}$$

$$b = \frac{\sqrt{6}}{\sqrt{H_k + H_{k-1}}} \tag{31}$$

Momentum Method

The momentum method is a technique used to accelerate and improve the convergence rate of gradient-based optimization. While gradients of loss function are being calculated, it accumulates a velocity in the same direction of that gradients, and updates the parameters using that same velocity instead of the gradients. That is represented by Equation 32 and 33, where $\mu \in [0, 1]$ is the momentum coefficient, η is the learning rate, V is the velocity and $\nabla l(\theta)$ is the gradient of the loss function [15].

$$V^{(k+1)} = \mu V^{(k)} - \eta^{(k)} \nabla l(\theta^{(k)} \tag{32}$$

$$\theta^{(k+1)} = \theta^{(k)} + V^{(k+1)} \tag{33}$$

Batch Normalization

Instead of normalizing all the data in preprocessing, we can make it while the data flows through the deep network in each mini-batch. This process avoids a problem named "internal covariate shift", where weights and parameters from hidden layers deviate from zero-mean and variance of one. By doing batch normalization we obtain a faster learning and a higher accuracy [35].

2.2.6 *Model Selection*

There are basically four methods for selecting hyperparameters. By hyperparameters we mean the learning rate, momentum coefficient, number of layers, number of units, activation functions and loss function, and other parameters that we can change to improve the learning and capacity of the neural network.

Table 3.: Hyperparameters of ANN.

Hyperparameter	Description
Number of units	Input and output layers related with the shape of inputs and outputs. Number of hidden units choice is empirically-driven.
Number of layers	More hidden layers means more capacity to detect non-linearities.
Activation functions	Possible to choose different activation functions for the different layers. Output activation function related to the output we want (classification/regression).
Loss function	Measure of error we want to minimize.
Learning rate	Step size in gradient descent.
Early stopping	Number of epochs that can be run before the model begins to overfit.
Number of training iterations (Epochs)	More epochs can lead to overfitting, while less than necessary can lead to underfitting.
L1 and L2 regularizers	Choosing the right regularizer is crucial to avoid overfitting.
Dropout	Better generalization and less overfitting.
Weight Initialization	Weight initialization can have influence on the local minimum found.
Batch size	Number of samples that are going to be propagated through the network on each epoch.
Momentum	Faster convergence when using appropriate momentum.

The first method is manual search, where we select the parameters using the knowledge we have, and observing the results. That process is repeated until we find the best parameters choice. On grid search, we test all the possible combinations of hyperparameters where each takes a set of possible values in their domain, and then select the best combination. Then, we can repeat the process to a larger range of values to fine-tune the hyperparameter.

Random search is another method, where like in grid search we define a range of values for each, and then in a random way, we test combinations of those values. This method showed to perform with the same accuracy of grid search, and sometimes even better [8].

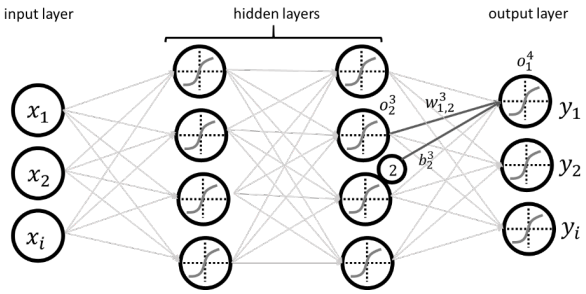


Figure 8.: Example of a multilayered ANN.

Another way of selecting hyperparameters is performing a Bayesian optimization, in which we use the information gained from a previous experiment to select the hyperparameters for the next experiment [55].

2.3 DEEP LEARNING

2.3.1 *Deep neural networks*

Differently from "shallow" methods, deep learning methods have multiple layers of nonlinear modules of representation. Each of them transforms the representation at one level into a more abstract higher level. These transformations permit discovering intricate structures in high-dimensional data with several degrees of freedom. Thus, it is quite promising that deep learning is outperforming the machine learning "shallow methods" in some areas. But, as we know, deep learning does not represent an advantage in every situation, because it is easily prone to overfitting.

The biological motivation in the use of deep architectures instead of one hidden layered ANN is because brains have a deep architecture. Humans organize their ideas hierarchically, through composition of simpler ideas. First we learn simpler concepts, and then we compose them to represent more abstract ones. One example is the visual cortex of humans when processing an image. The image runs through several layers of abstraction in neurons until it reaches the final visual representation.

2.3.2 *Architectures*

According to the task, in deep learning there are several architectures capable of performing better. Here, we will present the most common neural network architectures used in deep learning.

Deep Neural Networks (DNN)

DNN (Figure 8) are neural networks similar to shallow ANN except they have more than one hidden layer. As aforementioned, adding extra hidden layers to a neural network we can gain more degrees of freedom, having the capacity to fit better the training data, even for more complex cases. Here resides the advantage of deep architectures against shallow ones. But that has two major problems - computational power and overfitting. Because today we have more computational power than a few decades ago, scientists can try deep architectures using clusters and distributed systems, and vulgarly using *Graphics Processing Unit (GPU)* to perform the computations. The second problem addressed, overfitting, is solved through regularization techniques and designing a robust model selection. But still sometimes the deep learning architectures fail to outperform shallow ones.

Convolutional Neural Networks (CNN)

Based on experiments made by Hubel [34] in the visual cortex of the cat, surged one of the most used deep learning architecture, the *Convolutional Neural Networks (CNN)*. CNN are models of supervised learning, especially used in computer vision and image recognition. The structure of this network is composed by two types of layers - convolutional and pooling layers (Figure 9). The input is in form of multiple arrays: 1D for signals and sequences, including language; 2D for images or audio spectrograms; and 3D for video or volumetric images.

In a first step, convolution, a filter is passed over a specific area, the receptive field, and a higher order representation is made, recurring to ReLU units. Then, inputs from the convolution layer can be "smoothened" to reduce the sensitivity of the filters to noise and variations, creating an even more abstract representation (pooling). On an array data such as images, local groups of values are often highly correlated, forming distinctive local motifs that are easily detected. The repetition of the processes of convolution and pooling will create the output model of the CNN [39]. For this specific architecture there are some hyperparameters that must be chosen (Table 4).

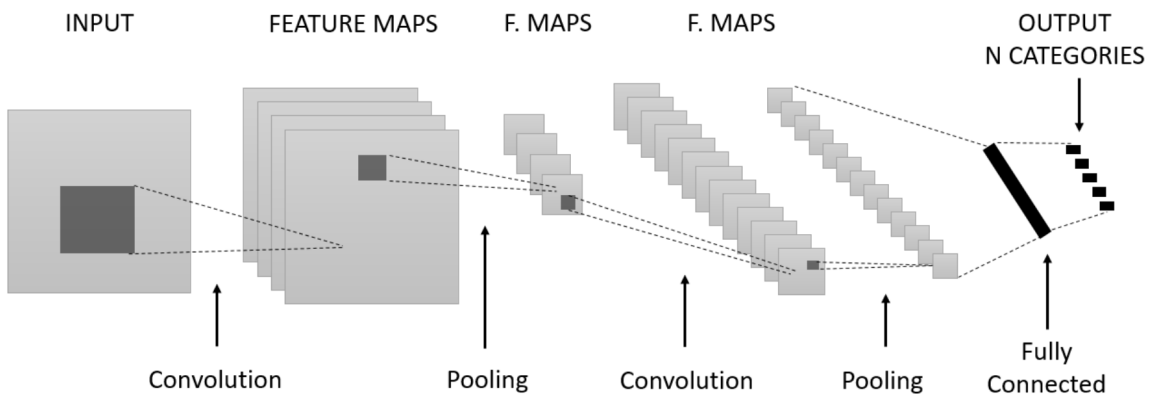


Figure 9.: Convolutional Neural Network.

Table 4.: Hyperparameters of CNN.

Layer	Hyperparameter
Convolution	Number of features
	Size of features (receptive field)
Pooling	Window size
	Window stride
Fully connected	Number of neurons

Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN) are popular models that have shown great promise in *Natural Language Processing (NLP)*. The concept behind RNN is the use of sequential information. RNN are called recurrent because the output depends on previous computations. The training of a RNN is similar to the one of a traditional ANN. We also use backpropagation, but with a difference, the gradient at each output depends on the calculations of the current time step, and also the previous ones [39]. Like CNN and DNN, RNN are used to solve supervised learning problems.

Stacked Autoencoders (SA)

Autoencoders (AE) take the raw input, pass it through a hidden layer and try to reconstruct the same input at the output. To find more robust features, we train the autoencoder

to reconstruct the input from a corrupted version of it (*Denoising Autoencoders (DA)*). The stochastic corruption of data randomly sets some of the inputs (as many as half of them) to zero. Basically, the DA works like a single layer neural network where instead of predicting labels we predict the input. Then, the loss function calculated can be the cross-entropy (binary inputs), or the squared euclidean distance (real inputs), measuring the difference between the raw input and the predicted input at the output layer. The training is composed by two stages - an unsupervised pre-training stage that is useful to build features from data, and a fine-tuning stage where we want to predict the error on a supervised task.

Stacked Denoising Autoencoders (SDA) are similar to DA, but with more than one hidden layer. The pre-training is done one layer at a time. Each layer is trained as a DA by minimizing the error in reconstructing its input (which is the output code of the previous layer). When the first layers are trained, we can train the $k + 1^{th}$ layer because we can now compute the code from the layer below. Summarizing, with SDA we have a first step with an unsupervised training and a second step that corresponds to a supervised training using the features constructed on the first step [60].

Restricted Boltzman Machines (RBM), Deep Boltzman Machines (DBM) and Deep Belief Networks (DBN)

RBM can find patterns in our data by reconstructing the input. An RBM is a two-layer neural network, and so considered as a shallow method. The first layer is known as the visible layer and the second is called the hidden layer. Each node in the first layer is connected to every node in the hidden layer. An RBM is considered "restricted" because there is only one hidden layer.

In the forward pass, an RBM takes the inputs and translates them into a set of numbers that encode the inputs. In the backward pass, it decodes back to form the reconstructed inputs. Three steps are repeated over the training process - the first one is similar to feed-forward ANN, next, in a backward pass, each activation is combined with an individual weight and an overall bias, and the result is passed to the visible layer for reconstruction; in the third step, on the visible layer, the reconstruction is compared against the original input to determine the quality of the result. RBM use a measure called Kullback–Leibler Divergence (relative entropy) for the last step.

RBM are energy-based models which associate a scalar energy to each configuration of the variables of interest. This model can be learnt by performing (stochastic) gradient descent on the empirical negative log-likelihood of the training data. When we have more than one hidden layer we call them *Deep Boltzman Machines (DBM)* or Non-restricted Boltzman Machines. Multiple hidden layers can be learned by treating the hidden layer output of one RBM.

Deep Belief Networks (DBN) are a composite model which are different from DBM, because they have undirected connections between its top two layers and downward directed connections between all its down layers.

All the methods mentioned above are unsupervised, which means that the data does not need to be labeled and we can extract new features [30].

2.3.3 *Deep Learning Frameworks and Tools*

Deep learning frameworks are software solutions that help the construction of neural network deep models. They present some differences in the ease of use and the way models are trained and constructed. One of the main advantages in the use of these frameworks is the automatic differentiation. That way, the framework computes automatically the gradients of neural networks.

Theano

Theano [3] is a framework developed by University of Montreal and is build in Python and compiled to C++. It is a compiler for mathematical expressions that uses a declarative programing paradigm, which means that the programmer does not need to specify the order in which instructions need to be executed. The neural network is declared as a symbolic computational graph, and then is compiled and executed. Theano supports GPU usage which translates into a better performance then with *Central Processing Unit (CPU)*. That is done using CUDA, a NVIDIA parallel computing platform and *Application Programming Interface (API)* for general-purpose computing on GPU units [6].

TensorFlow

TensorFlow [2] is developed by Google and written in C++, offering an interface to Python. It is similar to Theano with respect to the declarative paradigm, the automatic differentiation and the abstraction used (symbolic computational graph). Because TensorFlow is C++ native, it has a native support for parallelization across different platforms (CPUs and GPUs). Another advantage is TensorBoard that allows the visualization of networks to monitoring the training progress (i.e. learning curves or parameter updates) [6, 51].

Keras

Keras [17] is a Python library that works as a wrapper for both Theano and TensorFlow. With it one can make higher-level implementations of neural networks turning it to be easier and faster. The main data structure in Keras is a *model*, which represents a single

layer that can easily be stacked to other layers. The modularity concept of Keras is one of its major advantages. It is possible though to build complex models using Keras API.

3

CANCER OMICS

The Human genome is three billion base pairs long and is split into 23 pairs of chromosomes. Humans have two sets of chromosomes, one from each parent, that differ in their content by around 0.01 percent. An individual's genome, therefore, contains around six billion base pairs. The Human Genome Project (HGP), was the first project that mapped all of the genes of the human genome. The project's goal was to produce a full human genome sequence for $3 billion between 1990 and 2005. Nowadays, the "$1,000 genome" promises to make the *Deoxyribonucleic Acid (DNA)* sequencing so affordable that individuals might think to have a full personal genome sequence.

This revolution is deeply connected with the surge of NGS techniques (also called High Throughput), that turned the sequencing task less expensive. Cheaper sequencing technologies made genomics data more relevant by increasing the number of researchers able to study genomes, and the number of genomes they can use to understand variations among individuals in both sickness and health [18].

The current nomenclature of omics sciences includes genomics for DNA variations, transcriptomics for *messenger Ribonucleic Acid (mRNA)*, proteomics for peptides and proteins, and metabolomics for intermediate products of metabolism, but also many other data types, as epigenomics (e.g. DNA methylation). These data results from techniques as NGS technologies (DNA and *Ribonucleic Acid (RNA)* sequencing), DNA microarrays, Mass Spectrometry or Nuclear Magnetic Resonance, just to name a few.

3.1 CANCER OMICS DATABASES

The generation of low-cost data is leading us to the "big data" era in biomedicine. The advent of NGS techniques, particularly when applied to cancer research, brought a huge amount of information that needs to be stored, and we are talking in the order of petabytes of data. The availability of this kind of data provides a unique opportunity, but also raises some new challenges. To cite some, there is the organization of such an huge amount of data that can come from different platforms and research teams. There is also the privacy of patients information that must be protected. Another challenge is to make these data

available to researchers on an efficient way. There are several repositories of cancer omics data like the GDC. More recently, we have been surrounded by some portals that combine both databases and some statistical inference in real time (e.g. cBioPortal) [32, 31].

It is very important that those resources are available, and sometimes a question arises - is it better to keep investing on new experiments, or in analyzing the existent data? The answer is not so obvious, but we can conclude that there must be a huge amount of data to be processed, and lots of discoveries to be made in the future from what we have until now. The largest omics datasets available until the date are TCGA (now part of GDC) and *The Encyclopedia of DNA Elements (ENCODE)*. Over the next sections we will refer to some of the most popular cancer omics databases.

3.1.1 *The Cancer Genome Atlas (TCGA)*

TCGA was a United States government project, started in 2005, and supervised by the National Cancer Institute's Center for Cancer Genomics and the National Human Genome Research Institute. Its main goal was to catalogue genetic mutations responsible for cancer, using NGS and bioinformatics. They started studying glioblastoma multiforme, lung, and ovarian cancer, but in the end, they characterized 33 cancer types. The TCGA project was very important to the development of cancer knowledge, and many of cancer studies presented in this document used TCGA data. Now TCGA data is hosted in GDC [58]. The TCGA data is organized through tiers.

3.1.2 *NCI Genomic Data Commons (GDC)*

GDC is a project developed by the *National Cancer Institute (NCI)*, the Univeristy of Chicago, the Ontario Institute for Cancer Research, and Leidos Biomedical Research. It contains information from NCI-funded projects such as TCGA and *Therapeutically Applicable Research to Generate Effective Treatments (TARGET)*. Besides those two, GDC integrates data from: International Cancer Genome Consortium, NCI clinical trials, and user-submited studies.

With this project, one of the goals was to harmonize the data, using pre-defined analytic pipelines to ensure that researchers can use these data, regardless of data provenience. GDC also has an interface to upload new cancer genomic data from researchers worldwide. There are also concerns about security and access to patients data. There is an API for developers, to easily access data from their applications.

There are several goals of the GDC team, which are to develop a new taxonomy of cancer based on molecular pathogenesis, to identify low-frequency cancer drivers, to define

genomic determinants of treatment response, and to compose clinical trial cohorts sharing targetable genetic lesions.

The different types of data GDC stores are: mutation calls, structural variants, copy number, and digital gene expression. For that they implement state-of-the-art methods [32]. Currently, GDC Data Portal stores 39 projects, 14,531 cases, and 4.98 petabytes of data.

There are four levels of data according to the processing degree:

- level 1 - Raw data
- level 2 - Normalized data
- level 3 - Aggregated data
- level 4 - Regions of Interest data
- level 0 - No level

3.2 CANCER BIOINFORMATICS APPLICATIONS

3.2.1 *Precision Medicine*

According to the *National Institutes of Health (NIH)*, precision medicine is "an emerging approach for disease treatment and prevention that takes into account individual variability in genes, environment, and lifestyle for each person" [59]. Although precision medicine can be applied to many diseases, oncology sits at its vanguard (precision oncology). One obvious reason is the fact that cancer is a genomic disease: most cancers harbor a cocktail of mutated oncogenes and tumor suppressors that work in concert to specify the molecular pathways that lead to their genesis, maintenance, and progression.

The main steps in precision oncology are: first characterize the genomes of tumors; second, filter the genomic data through a knowledge base of existing and emerging anticancer drugs; and third, present an annotated list to the treating oncologist that can be incorporated into clinical decision making [27]. Although we already know what needs to be done, it is not so straightforward to apply these concepts to real cases. There are several challenges related to the quality and quantity of genomic information, the cost of various platforms, and finally the analytic challenges and methods chosen to perform data analysis.

Bioinformatics, of course, plays a determinant role, and there are already attempts to apply the machine and deep learning technologies in *Computer Aided Diagnostic (CAD)* software tools [16]. We will cover below a number of applications which can contribute to this effort, focusing on the application of machine/deep learning over cancer omics data.

3.2.2 *DREAM Challenge*

The DREAM Challenges is a platform for computational biologists and data scientists that introduces the concept of crowd-sourcing to solve real problems in biomedicine. That way it is possible to have teams all over the world participating and contributing to science in a competitive way. It is developed by Sage Bionetworks, which has made important contributions to the TCGA project. The expertise and institutional support are provided by Sage Bionetworks, along with the infrastructure to host challenges via the Synapse open platform. In the DREAM project, there have been various DREAM challenges related to cancer (Table 5). Recently, there have been two related challenges - ICGC-TCGA DREAM Somatic Mutation Calling RNA Challenge (SMC-RNA) and Multiple Myeloma DREAM Challenge.

Table 5.: DREAM challenges related to cancer.

Challenge	Name	Year
DREAM 6	FlowCAP2 Molecular Classification of Acute Myeloid Leukaemia Challenge	2011
DREAM 7	Sage Bionetworks-DREAM Breast Cancer Prognosis Challenge	2012
DREAM 7	NCI-DREAM Drug Sensitivity Prediction Challenge	2012
DREAM 8	HPN-DREAM Breast Cancer Network Inference Challenge The Broad-DREAM Gene Essentiality Prediction Challenge	2013
DREAM 8.5	ICGC-TCGA-DREAM Somatic Mutation Calling Tumor Heterogeneity Challenge	2013
DREAM 9	Acute Myeloid Leukemia (AML) Outcome Prediction	2014
DREAM 9.5	Prostate Cancer DREAM Challenge	2015
DREAM 10	Multiple Myeloma DREAM Challenge	2017

Along the DREAM project, there were some important discoveries such as PAM50 and Oncotype DX, that reveal the extensive work done for improving treatment based on the incorporation of different features [38].

3.2.3 *cBioPortal*

The cBio Cancer Genomics Portal (http://cbioportal.org), is developed at the Memorial Sloan-Kettering Cancer Center. The main goal of cBioPortal is to provide a web resource for exploring, visualizing, and analyzing multidimensional cancer genomics data. It contains 148 cancer genomics studies, including the aforementioned TCGA, GDC and *International Cancer Genome Consortium (ICGC)* datasets. With cBioPortal it is possible for researchers to easily interact with omics data with a graphical interface and design fast data analysis without any programming barriers. cBioPortal also has an API for developers [11, 26].

3.2.4 *Cancer Genomics Cloud (CGC)*

The CGC is a web portal (http://www.cancergenomicscloud.org/), developed by Seven Bridges. Within *Cancer Genomics Cloud (CGC)* we can access TCGA data (both public and private cohorts) and perform analysis within the standard bioinformatics pipelines using a graphical interface and an API. Somehow it resembles the cBioPortal, but the advantage of CGC is that we can upload our own custom analysis tools and share them with researchers around the world.

3.3 MACHINE LEARNING APPLICATIONS IN CANCER

Over the last years, within the context of "cancer big data", and using gene expression data, several studies were published recurring to machine learning "shallow" methods. There are both supervised and unsupervised approaches. The major classes of supervised cancer biological problems are: prediction of cancer susceptibility, recurrence and survival [38, 31, 19]. For unsupervised learning we can refer the cancer staging, sub-typing, and feature extraction from gene expression datasets.

3.3.1 *Classical Machine Learning*

Using classical machine learning algorithms we have several studies. On Table 6 we summarize some representative papers that give a good perspective over what has been done. There are both unsupervised and supervised learning approaches, dealing with several types of cancer. We can observe the importance of the stratification of cancer subtype and risk prognosis. The objectives concentrate on the development of more accurate tools for clinical decision and patient treatment. The most used algorithms are SVM, ANN, and glsbn. We can mention Tan et. al [56] work with DA. Here, with the help of an unsupervised learning neural network architecture (DA), this team constructed features that shown

to be more accurate to predict several types of endpoints. It is very interesting because we can exploit the advantages of neural networks with gene expression data.

Table 6.: Published papers of machine learning and cancer over gene expression data.

Type of study	Article title	ML method	Ref.
Susceptibility prediction	Predicting cancer susceptibility from single-nucleotide polymorphism data: a case study in multiple myeloma	SVM	[61]
	Predictive models for breast cancer susceptibility from multiple single nucleotide polymorphisms	SVM	[45]
Recurrence prediction	Predicting cancer susceptibility from single-nucleotide polymorphism data: a case study in multiple myeloma	BN	[22]
	Integrative gene network construction to analyze cancer recurrence using semi-supervised learning	Graph-based semi-supervised learning	[48]
Survival prediction	Risk classification of cancer survival using ANN with gene expression data from multiple laboratories	ANN	[14]
	Oral cancer prognosis based on clinicopathologic and genomic markers using a hybrid of feature selection and machine learning methods	SVM	[12]
	A Gene Signature for Breast Cancer Prognosis Using Support Vector Machine	SVM	[64]
	Predicting the prognosis of breast cancer by integrating clinical and microarray data with Bayesian networks	BN	[28]
Sub-typing	Unsupervised analysis of transcriptomic profiles reveals six glioma subtypes	*k-means*, Nonnegative Matrix Factorization	[43]
Feature Extraction	Extraction From Genome-Wide Assays of Breast Cancer	DA	[56]

3.3.2 *Deep Learning*

Deep learning is a recent field of research, and so there are still few applications in Bioinformatics. The majority of papers published are relative to image recognition. With CNN, they recur to medical images from radiology [37] or histology [62]. Another interesting approach is the integration of electronic records of patients [46]. The deep learning pipeline used on previous studies is usually the previously explained in Figures 1 and 2, with the model selection and regularization techniques characteristic from deep learning [6].

There are several publications using deep learning in the biological area, over omics data. In Table 7, we show some of the most relevant studies in different areas, such as gene expression, predicting splicing and transcription factors binding regions, epigenomics, microRNA binding sites, signaling and metagenomics. These studies are not directly related to cancer, but with the cellular and molecular biology processes, which may also be very useful in cancer research.

If we restrict the domain to cancer-only studies, the number of papers decreases drastically. For cancer itself, Fakoor et al. [23] used unsupervised learning stacked autoencoders to perform the task of cancer detection and cancer type classification and achieved better results with deep learning. Liang et al. [44] also achieved better results with deep learning. They used DBN to perform clustering by integrating genomics information with clinical data, which they compared with *k-means clustering*. Way et al. [63] used VA to construct features over genomics cancer data. It is a very promising study for unsupervised learning application of deep learning. The VA is similar to a SDA, the only difference is that they have a sampling layer which samples form a distribution (usually Gaussian) and then feeds the generated samples to the decoder part.

Table 7.: Published papers of deep learning in omics data.

Biological Problem	Article title	Arch.	Ref.
Gene expression	Gene expression inference with deep learning	DNN	[15]
	ADAGE signature analysis: differential expression analysis with data-defined gene sets	SDA	[57]
	Evaluating deep variational autoencoders trained on pan-cancer gene expression	VA	[63]
Splicing	Deep learning of the tissue-regulated splicing code	DNN	[42]
	Boosted Categorical Restricted Boltzmann Machine for Computational Prediction of Splice Junctions	DBN	[41]
Transcription factors and RNA-binding proteins	A deep learning framework for modeling structural features of RNA-binding protein targets	DBN	[65]
Epigenomics	Predicting effects of noncoding variants with deep learning–based sequence model	CNN	[67]
	DeepChrome: deep-learning for predicting gene expression from histone modifications	CNN	[54]
Micro-RNA binding	deepTarget: End-to-end Learning Framework for microRNA Target Prediction using Deep Recurrent Neural Networks	RNN	[40]
Signaling	Trans-species learning of cellular signaling systems with bimodal deep belief networks	DBN	[13]
Metagenomics	Multi-Layer and Recursive Neural Networks for Metagenomic Classification	DNN, RNN	[20]
Others	Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning	N/A	[4]

4

METHODS

In this chapter, we will present the project pipeline and the details of the developed machine learning models, as well as an explanation about the used the data. For software development, the programming languages used were Python, for the majority of jobs, and R for accessing microarrays data. In Python, the most important libraries used were Numpy, SciPy, Pandas, Scikit-learn, and Keras with Theano backend. Within R we used Bioconductor, and the Limma package for accessing Agilent two-color microarrays. The gene expression data origin is the Neuroblastoma Data Integration Challenge, part of the 16th *Annual International Conference on Critical Assessment of Massive Data Analysis (CAMDA)*. Because of computational power demanded, mainly by the deep learning models, we used the Search 6 cluster from *Departamento de Informática (DI)*, *Universidade do Minho (UM)*. Deep learning models were run using GPU NVIDIA Tesla K20 (node 652-1 and 652-2). For *Multi-Task Deep Neural Network (MT-DNN)*, unsupervised learning, and CPU vs. GPU experiments we used an Amazon EC2 p2.xlarge instance with a NVIDIA Tesla K80 GPU.

4.1 DEEP LEARNING SOFTWARE

The code for the entire pipeline for this work is available in a github repository `https://github.com/lmpeixoto/deepl_learning` [49].

It is composed by the following modules:

- ShallowModel - supervised shallow machine learning models pipeline.
- DNNModel - supervised DNN models pipeline.
- DNNMTModel - supervised multi-task DNN models pipeline.
- SDAEModel - unsupervised SDA models pipeline.

ShallowModel creates the supervised shallow machine learning experiments, DNNModel and DNNMTModel create supervised deep learning experiments with DNN and MT-DNN models. SDAEModel creates deep learning unsupervised experiments. These will be described in the next sections.

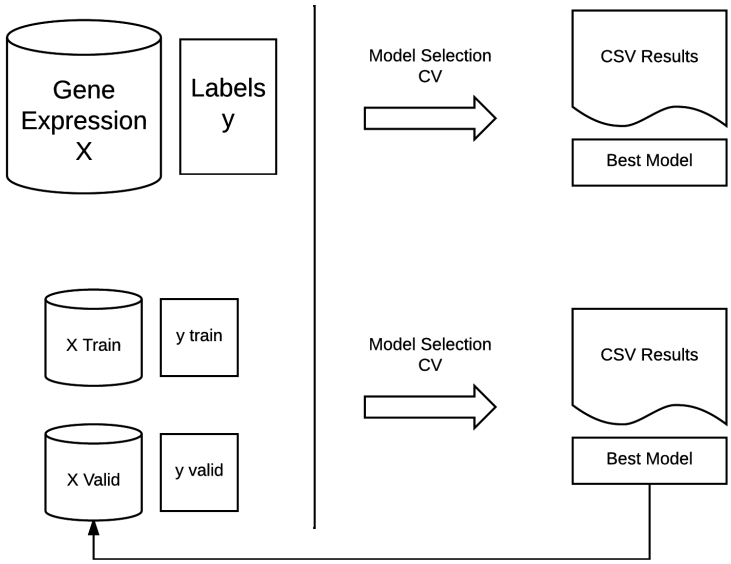


Figure 10.: Global supervised learning pipeline.

Supervised Learning - ShallowModel

The task of using supervised shallow machine learning algorithms is carried out with the python class named ShallowModel. This class was constructed to set a pipeline based on Figure 10 with:

1. Model selection - Grid search for the hyperparameters tuning of shallow classifiers through the several possible output variables in the dataset (e.g. clinical endpoints). That step creates a table with the grid search results and selects the best model.
2. Best model fit - the best model is fit with the selected data using the best set of hyperparameters.
3. Model evaluation - the best model is evaluated according to the pipeline. The results are saved to a report text file.

The machine learning classifiers tested were : LR, SVM, *Random Forests (RF)*, *Linear Discriminant Analysis (LDA)* and *K-Nearest Neighbors (KNN)*.

The ShallowModel class constructor takes as arguments an X and y matrices or the train/test splitted matrices (for using a separated test set), and the number of folds for cross validation (cv). Accordingly to the input arguments, the class decides whereas there is as splitted or non-splitted experiment (cross validation vs. hold out validation set). We consider a splitted experiment whenever we save a part of our data and keep it until we want to evaluate our model, similar to the one used in [66].

Inside ShallowModel, the model selection is performed by a grid search with cross validation carried out using Sci-kit Learn library methods. We obtain as result a table with the score results of each classifier and the parameters, with information of the fitting time. The best model is evaluated and the scores are calculated in different metrics (ROC AUC, F1 score, *Mathews Correlation Coefficient (MCC)*, accuracy, precision, recall, log loss) and saved in a plain text file that we call report. It is possible to run a model selection through all endpoints, or a selection of endpoints.

Table 8.: Shallow models parameter grid.

Model	Parameter Grid
SVM	C: [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 1.5, 2, 5, 10, 20, 50, 100, 150, 200, 500, 750, 1000] kernel: linear, rbf gamma: [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 1.5, 2, 5, 10, 20, 50, 100, 150, 200, 500, 750, 1000]
RF	n_estimators: [10, 50, 100, 200, 500, 1000, 2000, 5000, 10000]
KNN	n_neighbors: [1, 3, 5]
LDA	n_components: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
LR	C: [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 1.5, 2, 5, 10, 20, 50, 100, 150, 200, 500, 750, 1000] penalty: l1, l2

Supervised Learning - Deep Learning

For deep learning, we constructed a class named DNNModel. In this class, we designed a Keras sequential model of a DNN. For hyperparameter tuning we performed random search model selection keeping track of metrics along epochs. The random selection picks each parameter from a batch with ranges of values to test. The model takes the following parameters: dropout, output activation function, optimization algorithm, learning rate, units in input layer, units in hidden layers, number of epochs, batch size, early stopping, patience.

After model selection is completed, the code generates a file with results, similar to the previous class (ShallowModel). The best model combination is selected and evaluated. The fit results are saved - both a plot representing loss and accuracy metrics along epochs, and a report with metrics values. Inside deep learning classes, the resulting plot represents each fold from the cross validation fit.

Another model created was a MT-DNN. The MT-DNN includes several clinical endpoints in the y matrix (n x m, where n are the samples and m the clinical endpoints) forcing the neural network weights to adapt to predict all the endpoints at the same time. The Python class is DNNMTModel.

Unsupervised Learning

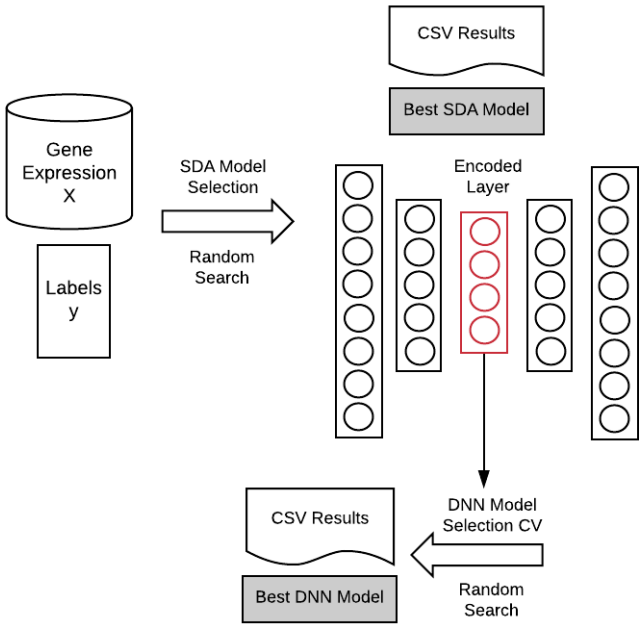


Figure 11.: SDA pipeline.

The *Stacked Autoencoders (SA)* is the model we chose to perform an unsupervised learning approach to gene expression data as depicted in Figure 11. The Python class of our code is SDAEModel. We can divide this module in two parts - SA and DNN. On a first moment, we have a SA that will encode the gene expression data. This way the SA creates a compressed representation of data. It is a kind of feature construction.

The second step occurs when we use that encoded representation to perform supervised learning classification. The autoencoder has a model selection method with random search

for hyperparameter tuning. We intend to find the optimum parameters that can reconstruct the enconded layer with minumum loss. After we find the best encoded representation to compress our data, we submit that layer to a regular DNN or MT-DNN. When it is complete, the code generates a file with the results.

4.2 CONFIGURATION OF THEANO AND KERAS LIBRARIES

To select Theano as default backend for Keras, we created a file (keras.json) where we set the "KERAS_BACKEND" flag to theano. Within Theano, we configured the library to use the GPU. For that, we created a file (.theanorc) specifying the parameters like *Compute Unified Device Architecture (CUDA)* path, memory alocation for *NVIDIA CUDA Deep Neural Network library (cuDNN)*, and other settings. Because of memory constraints, as the GPU used (NVIDIA Tesla K20) has limited 6 *Gigabytes (GB)* memory, we restricted the number of features of deep learning models to 5000, the batch size was also limited to a maximum of 200, and we did not use more than 4 hidden layers.

4.3 NEUROBLASTOMA DATASET

Neuroblastoma is a type of cancer that affects mostly children. It forms in certain types of nerve tissue. Most frequently, it starts from one of the adrenal glands, but can also develop in the neck, chest, abdomen or spine. The assessment of patient risk is based on both clinical and molecular parameters like tumor stage, patient age at diagnosis, and genomic amplification status of MYCN proto-oncogene. There have been previous models based on gene expression to predict patient outcome, but the prediction of high-risk patient remains challenging. The success of treatment depends on the risk group within the patient belongs. Because of that it is very important to develop a model that can predict accurately high-risk patients [66].

The Neuroblastoma dataset contains RNA-seq and Agilent two-color microarray gene expression profiles of 498 children patients, as well as matched *Array Comparative Genomic Hybridization (aCGH)* data for 145 of these patients for *Copy Number Variation (CNV)* and *Copy Number Alteration (CNA)* analysis. There are also clinical data for each patient. The main objective of this challenge was to integrate different types of data and produce better predictive models about survival time and other clinical endpoints [1]. For our study, we used the RNA-seq and Agilent microarrays data (accession numbers GSE49711 and GSE49710 from *Gene Expression Omnibus (GEO)* database, respectively).

We based our experiments in Zhang et al. [66]. In their experiments, they generated 360 gene expression machine learning models to predict six different endpoints:

- SEX - the patient gender;

- CLASS LABEL - label that indicates membership of a subgroup with extreme disease outcome: event-free survivors without chemotherapy for at least 1000 days post diagnosis (favorable), or patients died from disease despite chemotherapy (unfavorable).
- EFS ALL - the occurrence of events, that is, progression, relapse, or death;
- OS ALL - the occurrence of death from disease;
- EFS HR - the occurrence of events in the subset of high-risk patients (patients with stage 4 disease, more than 18 months at diagnosis and patients of any age and stage with MYCN-amplified tumors);
- OS HR - the occurrence of death from disease in the subset of high-risk patients.

The various endpoints have different predictive difficulties. SEX and CLASS LABEL are of low difficulty, EFS ALL and OS ALL of intermediate difficulty, EFS HR and OS HR are of high difficulty [66]. On the Table 9 we can have an overview of the *Neuroblastoma (NB)* dataset, the distribution between the cohorts and the different clinical endpoints.

Table 9.: Characterization of NB dataset. Adapted from Zhang et al. [66].

Cohort	Endpoint (bin 1/0)	1	0
All patients (498)	SEX (F/M)	211	287
	EFS ALL (Event Y/N)	183	315
	OS ALL (Death Y/N)	105	393
Class labeled patients (272)	CLASS LABEL (Unfavorable/Favorable)	91	181
High-risk patients (176)	EFS HR (Event Y/N)	120	56
	OS HR (Death Y/N)	92	131

On a first phase of our project we decided to reproduce the results of applying machine learning shallow predictors to the six different clinical endpoints and compare with the results published in Zhang et al. [66]. For that purpose, we used the original train/validation

split and then repeated the experiment with 10-fold cross validation, that is the configuration we will use for deep learning models. Then, we did the same with deep neural networks, to check if there is an improvement. Because on Zhang et al. [66] they used MCC as main metric to compare, we decided to replicate this setup and use MCC (Equation 34) as the main metric.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{34}$$

4.3.1 *RNA-seq data*

For RNA-seq data, we used the CAMDA dataset that was already preprocessed (log 1 + *Reads Per Kilobase Million (RPKM)* transformed). The RNA-seq dataset is composed by 498 samples and 60778 features, representing genes.

4.3.2 *Microarray data*

For Microarray Agilent 4x44 data, we used Bioconductor in R to load the GEO GSE49710 dataset. Then, using Limma package we performed background correction (normexp) and normalized with the quantile method. We removed the control probes and log2 transformed the expression measurements. In the end, we exported the expression matrix to be processed by Python Pandas package. The microarray dataset is composed by 498 samples and 30200 features, representing genes.

4.3.3 *Clinical data*

We extracted the gene expression matrix (X). After that, we integrated the information of the clinical data file with patient samples. For the several clinical endpoints, we also extracted a labels (y) matrix.

NBHighThroughput and NBMicroarrays are NB dataset specific modules to extract the expression matrices (X) and the output labels (y).

The NBHighTroughput class is composed of several properties (Table A.1.1) like the name of the dataset and clinical data files, number of samples, number of features, as well as the X and y matrices extracted. For reading the RNA-seq data file and clinical data there are some methods (Table A.1.2), aggregated in load_data. For each clinical endpoint, it generates different X and y matrices, according to clinical data information, keeping only complete cases. There is also a method for feature selection (filter method) based on *Mean Absolute Deviation (MAD)* (top_genes) and other based on variance (variance_filter).

The NBMicroarrays module overrides the NBHighthroughput, changing only the way it reads the expression matrix.

4.3.4 *Experiments Performed*

Supervised Learning

We used the information from supplementary files from [66] to reconstruct the boxplots that depict the MCC score of their models over the different clinical endpoints. For shallow models, we tested both the original split and 10-fold cross validation pipelines with RNA-Seq and Microarray datasets. For DNN, we replicated the same experiments. For MT-DNN, we tested with RNA-seq dataset.

Unsupervised Learning

For unsupervised learning, we built a SA. Then, we extracted the encoded layer and ran a DNN in the endpoint Sex on a supervised learning way, from RNA-Seq dataset. The architecture of those networks are the ones in Table 10.

Table 10.: SA and DNN of unsupervised experiment.

Architecture	Parameters	Type
Stacked Autoencoder	batch size: 60 early stopping: False learning rate: 0.001 nb epoch: 1000 optimization: SGD units in hidden layers: [2500, 500 ,2500] units in input layer: 5000	Unsupervised
Deep Neural Network	batch size: 70 dropout: 0.6 early stopping: False learning rate: 0.001 nb epoch: 500 optimization: SGD output activation: sigmoid units in hidden layers: [250, 100, 50] units in input layer: 500	Supervised

GPU vs CPU fit time comparison

To compare the fit time difference between CPU and GPU we ran a DNN with the RNA-Seq dataset, using the Endpoint Sex to perform a supervised learning experiment. The DNN configuration is detailed on Table 11.

Table 11.: DNN parameters for GPU vs CPU performance comparison.

batch size: 80
dropout: 0.6
early stopping: False
learning rate: 0.001
nb epoch: 10
optimization: SGD
output activation: sigmoid
units in hidden layers: [1000, 10]
units in input layer: 5000

5

RESULTS AND DISCUSSION

5.1 NEUROBLASTOMA DATASET

5.1.1 *Supervised Learning*

RNA-Seq

Using supplementary files from Zhang et al. [66], we were able to reconstruct the results of the original publication, shown in Figure 12. Here, we can clearly distinguish the separation between the low difficulty endpoints (Sex and Class Label) and high difficulty ones (EFS HR and OS HR). To validate our methodology, we reproduced the same experiments, using the same train/validation split as used in the original work [66]. On the original experiment, the teams submitted several models for each endpoint (10 x 5-Fold Cross Validation) and they draw a boxplot representing all those models. For shallow models we obtained results, as can be checked by looking at Figure 13.

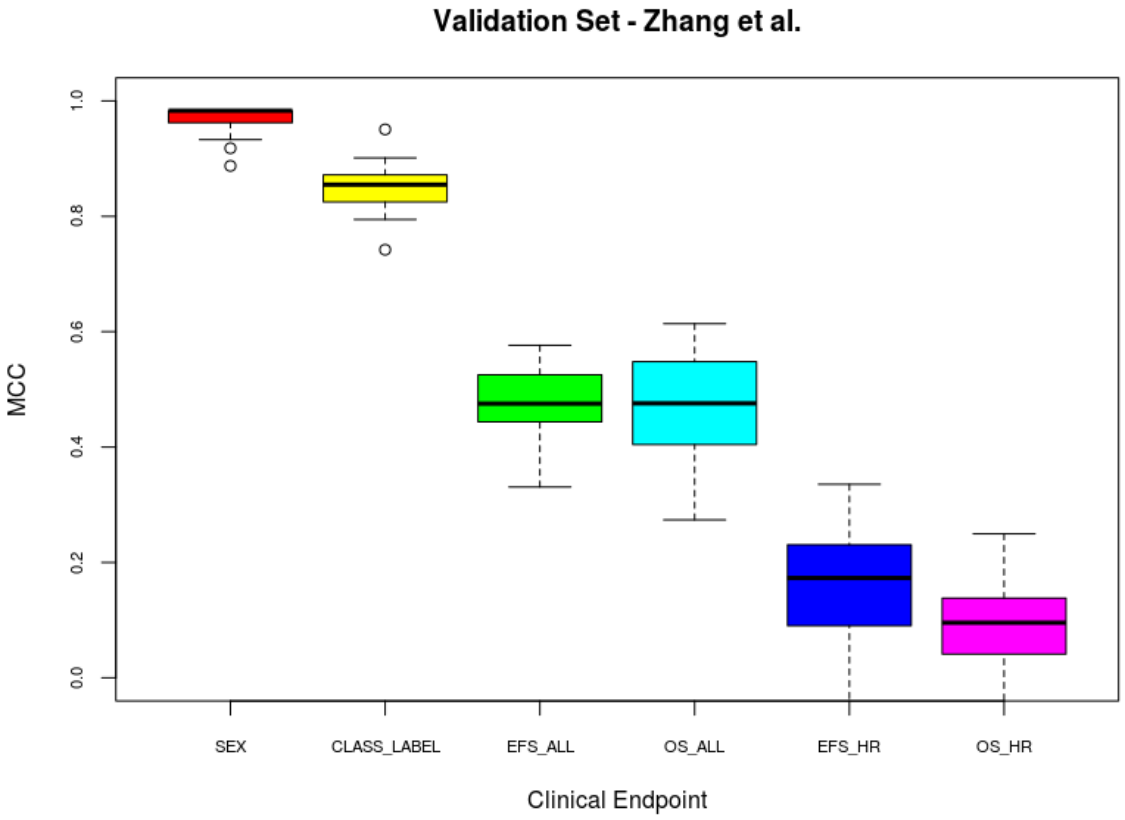


Figure 12.: Zhang et al results - shallow models with original split (RNA-seq).

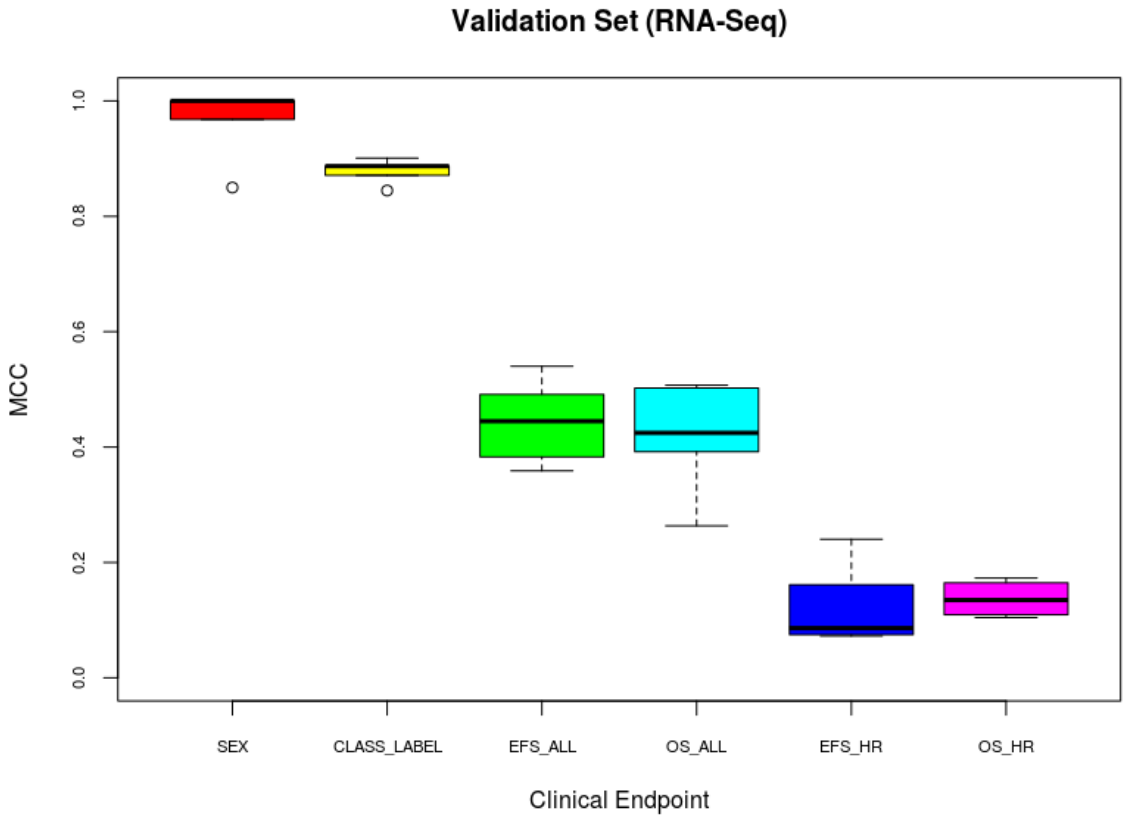


Figure 13.: Results for shallow models with original split (RNA-Seq) obtained in this work.

Since in our experiments we used a different pipeline, recurring to 10-fold cross validation, we checked if using our pipepline we would obtain similar results. In fact we did, as stated in Figure 14.

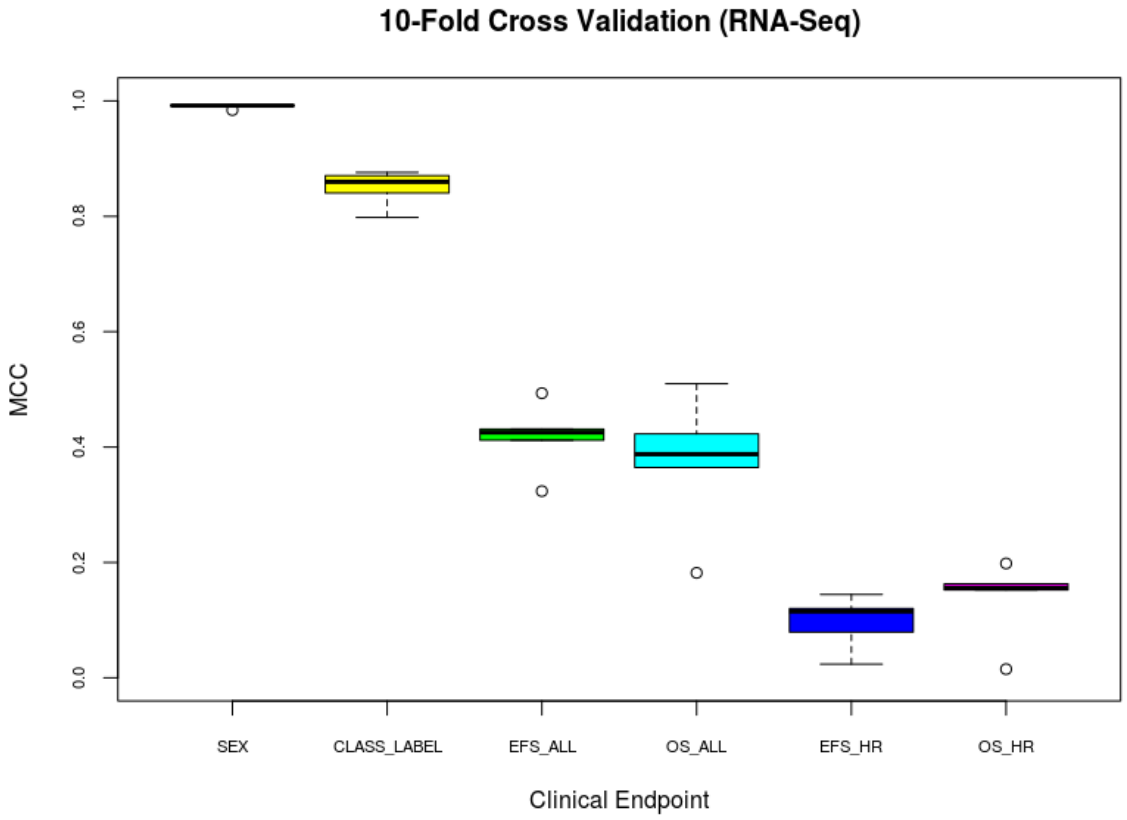


Figure 14.: Results for shallow models obtained in this work with 10-fold cross validation (RNA-Seq).

On Table 12 we have the best models for shallow and deep learning experiments and the respective scores. The best result for each endpoint is stressed in bold. For results in detail, on the Tables B.1.1 and B.1.2, we can see the configurations (type of model, number of features and parameters) and results for shallow models with the RNA-Seq dataset. On Tables B.1.3 and B.1.4 we did the same for DNN models.

Table 12.: NB best models (RNA-Seq).

Endpoint	Best Model	MCC
SEX	**LR**	**0.992 ± 0.0161**
	DNN	0.9841 ± 0.0195
CLASS LABEL	SVM	0.8762 ± 0.0957
	DNN	**0.8951 ± 0.1060**
EFS ALL	**LR**	**0.4933 ± 0.1491**
	DNN	0.4694 ± 0.1493
OS ALL	LR	0.5097 ± 0.1786
	DNN	**0.5284 ± 0.1707**
EFS HR	LDA	0.1445 ± 0.1940
	DNN	**0.1985 ± 0.2166**
OS HR	LR	0.1982 ± 0.3265
	DNN	**0.2211 ± 0.3197**

To better compare the two approaches we have a graph plotting the mean and standard deviation (error) of shallow models versus deep learning models (graph on Figure 15).

As we can observe from the graph, deep learning models outperformed shallow ones (in terms of mean) on Class Label, OS All, EFS HR and OS HR clinical endpoints. Looking for the errors we can observe that on those DNN outperforming endpoints, the error was slightly lower than from the shallow ones. But because the error was generally very high we cannot conclude which one was better. The same is true for the cases of the Sex and EFS ALL outputs, where the advantage is gained by the shallow learning methods, also quite slight and not significant.

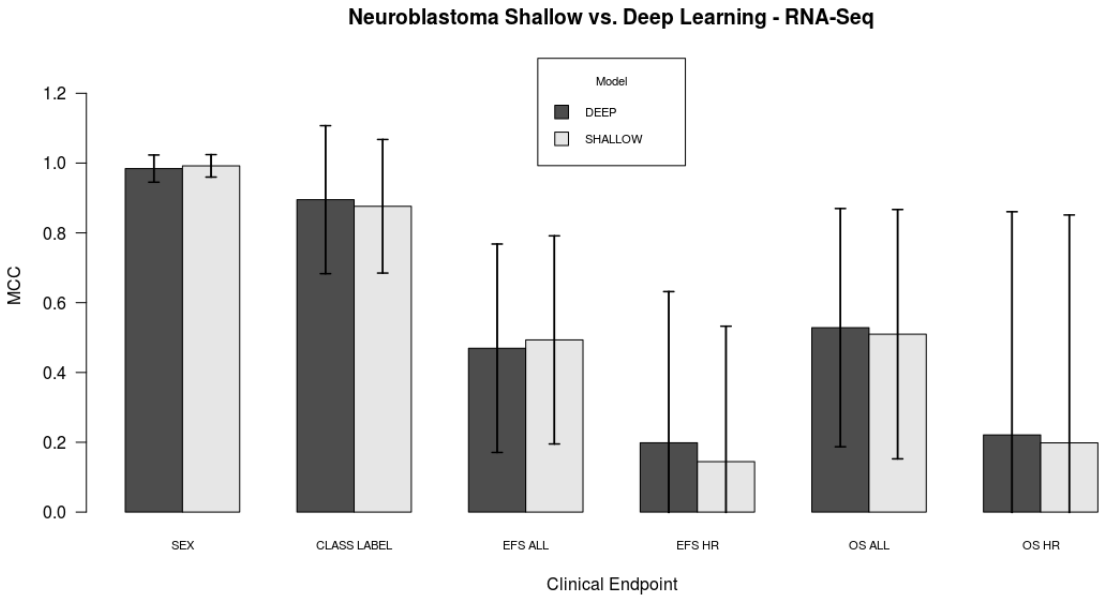


Figure 15.: Comparison between shallow and deep learning models (RNA-Seq).

Another experiment we did was the construction of a MT-DNN. The network configuration that performed best was the presented in Table 14. The results were not consistent with previous experiments (shallow and DNN) as we can observe on Table 13. For more detailed results we can consult Tables B.3.1 and B.3.2.

Table 13.: NB MT-DNN models results (RNA-Seq).

Endpoint	MCC
Sex	0.0604 $\pm$ 0.2120
Class Label	0.0754 $\pm$ 0.2016
EFS All	0.0919 $\pm$ 0.3717
OS All	0.1010 $\pm$ 0.3178
EFS HR	0.1445 $\pm$ 0.1940
OS HR	0.1893 $\pm$ 0.2874

A probable explanation is the reduced number of samples (note that this approach can only be applied to complete cases, where all endpoints exist) that were only 176.

Table 14.: MT-DNN parameters.

batch size: 100
dropout: 0
early stopping: True
patience: 80
learning rate: 0.01
nb epoch: 1000
optimization: Adam
output activation: sigmoid
units in hidden layers: [2500, 1000, 500, 100]
units in input layer: 5000

Microarrays

For the microarrays dataset, we performed the same experiments as RNA-Seq. As previously, we can compare our experiments with the results from [66]. We can see that reproducing the original split we obtained the results on Figure 16.

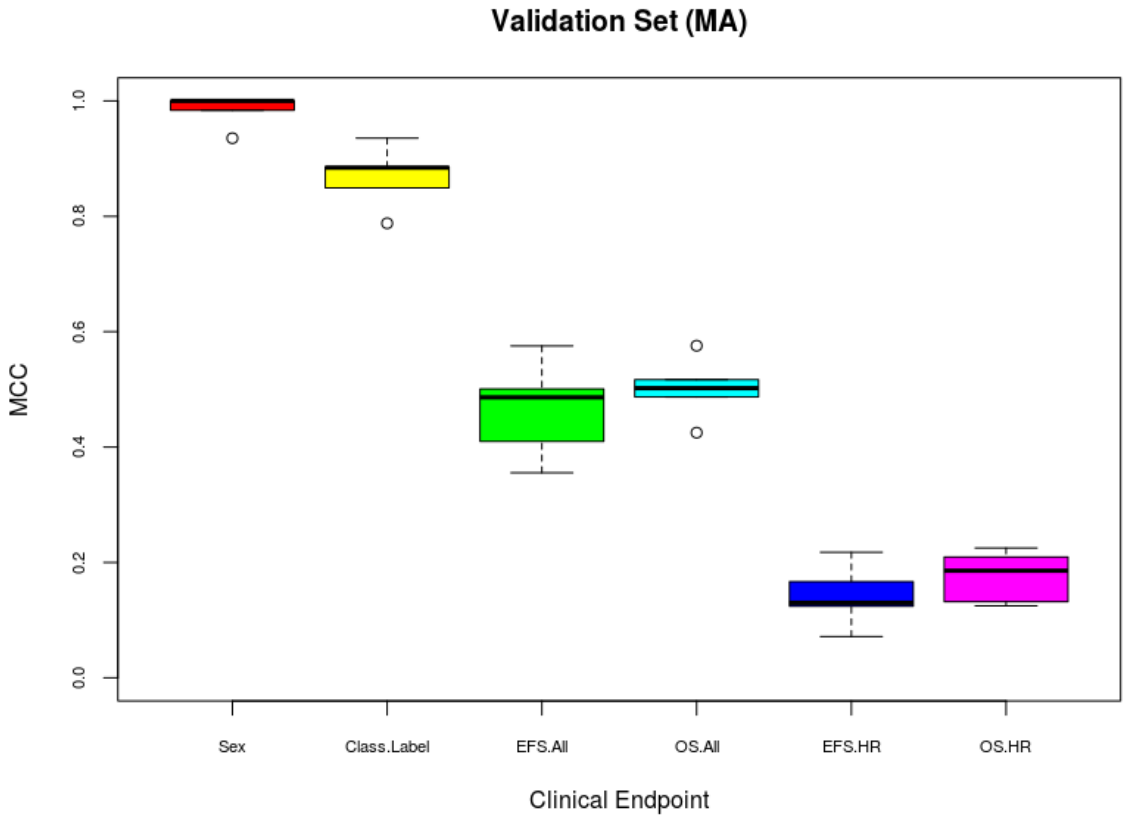


Figure 16.: Results for the shallow models with the original split (*Microarrays (MA)* dataset).

Then, with 10-fold cross-validation we obtained the results shown in Figure 17. Comparing with DNN models, we obtained the results shown in Figure 18.

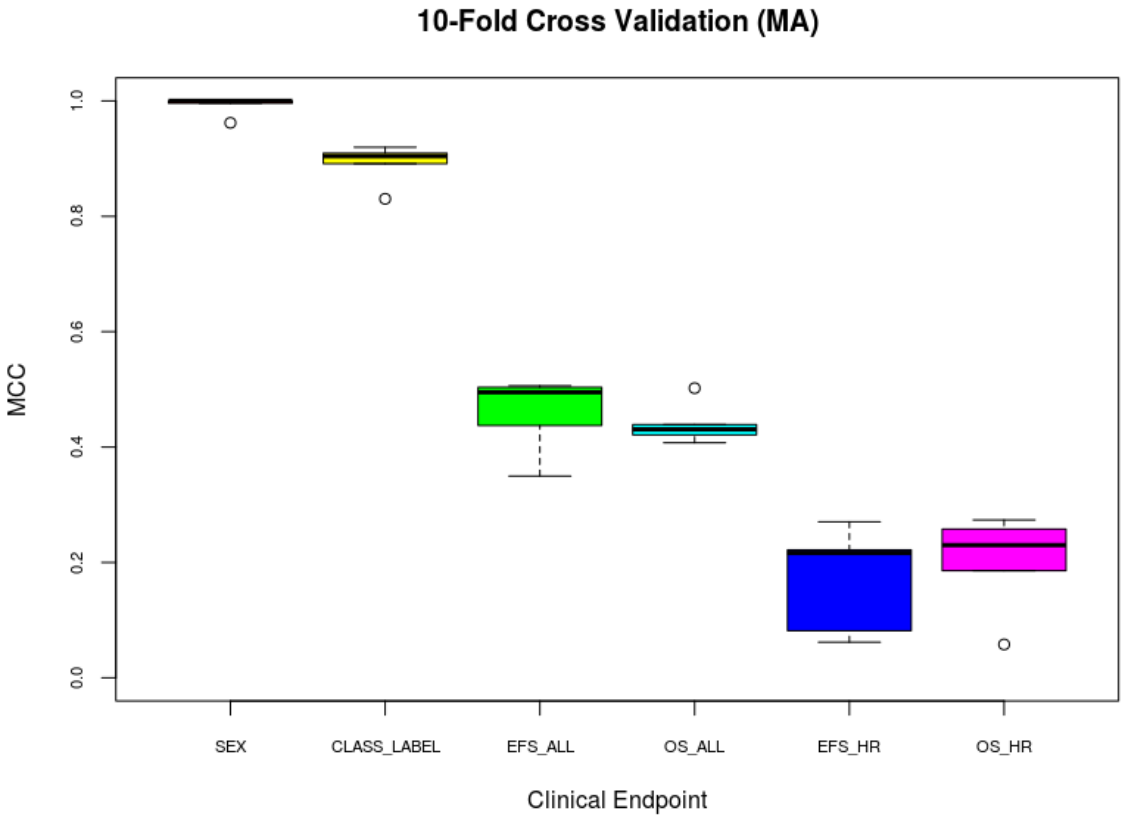


Figure 17.: Results for the shallow models with 10-fold cross validation (MA dataset).

Table 15.: NB best models (MA).

Endpoint	Best Model	MCC
Sex	**LR**	**1.0 ± 0.0**
	DNN	0.9796 ± 0.0204
Class Label	**SVM**	**0.9197 ± 0.0885**
	DNN	0.9075 ± 0.0957
EFS All	SVM	0.4330 ± 0.1154
	DNN	**0.4871 ± 0.1257**
OS All	LR	0.5021 ± 0.1259
	DNN	**0.5303 ± 0.1672**
EFS HR	**SVM**	**0.2704 ± 0.2521**
	DNN	0.2034 ± 0.1811
OS HR	LR	0.2738 ± 0.1644
	DNN	**0.3093 ± 0.1881**

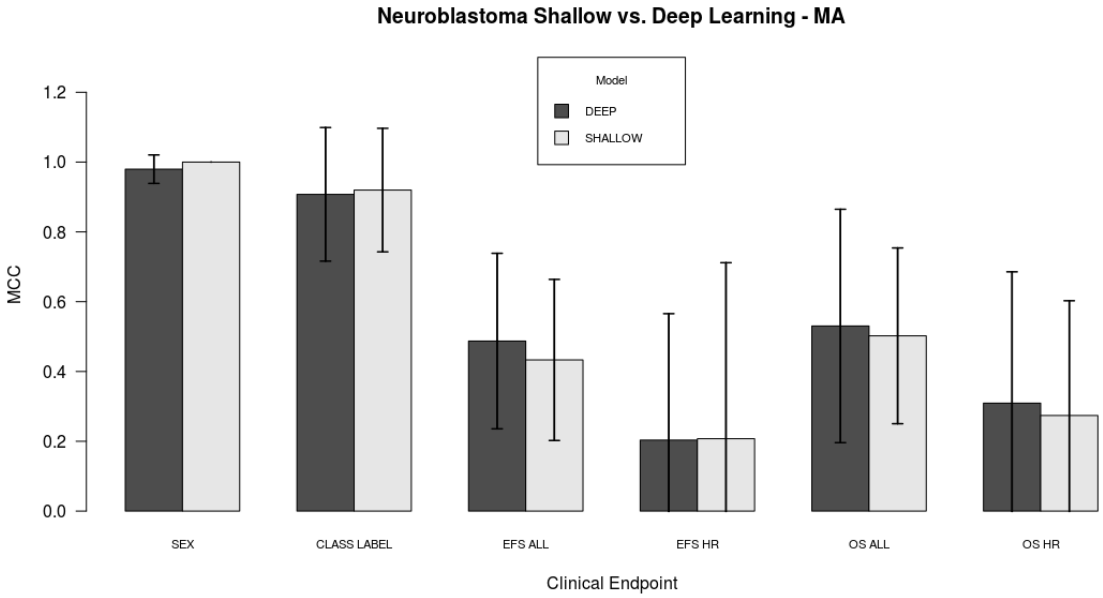


Figure 18.: Comparison between shallow and deep learning models (MA dataset).

As we can see, the DNN outperformed (in terms of mean) shallow ones on EFS ALL, OS ALL and OS HR. The error from DNN is higher than the shallow ones, except for EFS HR. But one more time the high error makes it impossible to compare both methods.

5.1.2 *Unsupervised Learning*

The unsupervised learning experiments generated the results that we can observe on Figure **??** and Table 16. The results were clearly worst than without the stacked autoencoder compression when comparing to shallow and DNN.

Because of the lack of computational power, we restricted the entry layer to 5000 features and limited the number of epochs to 500. If we observe the graphs on Figure 19 we can see that if we increased the number of epochs, probably the score would be better. Also, the number of features would be of considerable importance because we could capture more information as we increase the number of features.

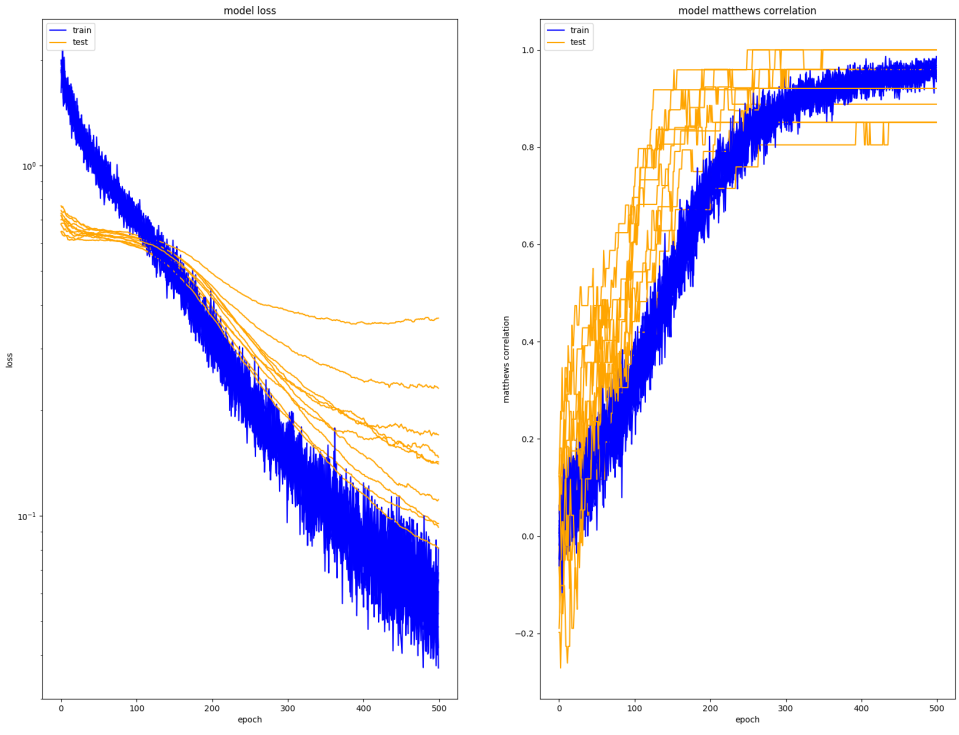


Figure 19.: Loss and MCC values for DNN coupled with a stacked autoencoder on endpoint Sex (RNA-Seq).

Table 16.: NB Stacked Autoencoder results (RNA-Seq).

Endpoint	Scores
Sex	roc auc: 0.9646 ± 0.0263 accuracy: 0.9600 ± 0.0309 f1 score: 0.9559 ± 0.0332 mcc: 0.9241 ± 0.0571 precision: 0.9222 ± 0.0651 recall: 0.9952 ± 0.0143 log loss: 0.1382 ± 1.0668

5.2 NEURAL NETWORKS FINE TUNING OF HYPERPARAMETERS

For model selection, we ran 4 times the random search, obtaining 4 models, and keeping the best of those. The first approach used early stopping, which means that the gradient descent stopped when it reached the maximum score and minimum loss, after advancing a few iterations more to see if there was an improvement (parameter patience, set to 30 epochs). If there was no improvement, the last best configuration was the selected one. In the end, we intend to reach an optimum configuration knowing the exact number of epochs of each neural network.

In this section, we will present the manual process that was used to fine tune a neural network. Note that there is no ground truth nor a guide to select the best configuration, meaning that sometimes this process can be tedious and time-consuming. For that, we will present the case of selection for Class Label endpoint from RNA-Seq dataset. On Table 17, we can observe the best model configuration obtained which we will start to make small changes.

Table 17.: Class Label case study.

Parameters	MCC
batch size: 150 dropout: 0.8 early stopping: True patience: 30 learning rate: 0.01 nb epoch: 1000 optimization: RMSprop output activation: sigmoid units in hidden layers: [2500, 500] units in input layer: 5000	0.8801 ± 0.0917

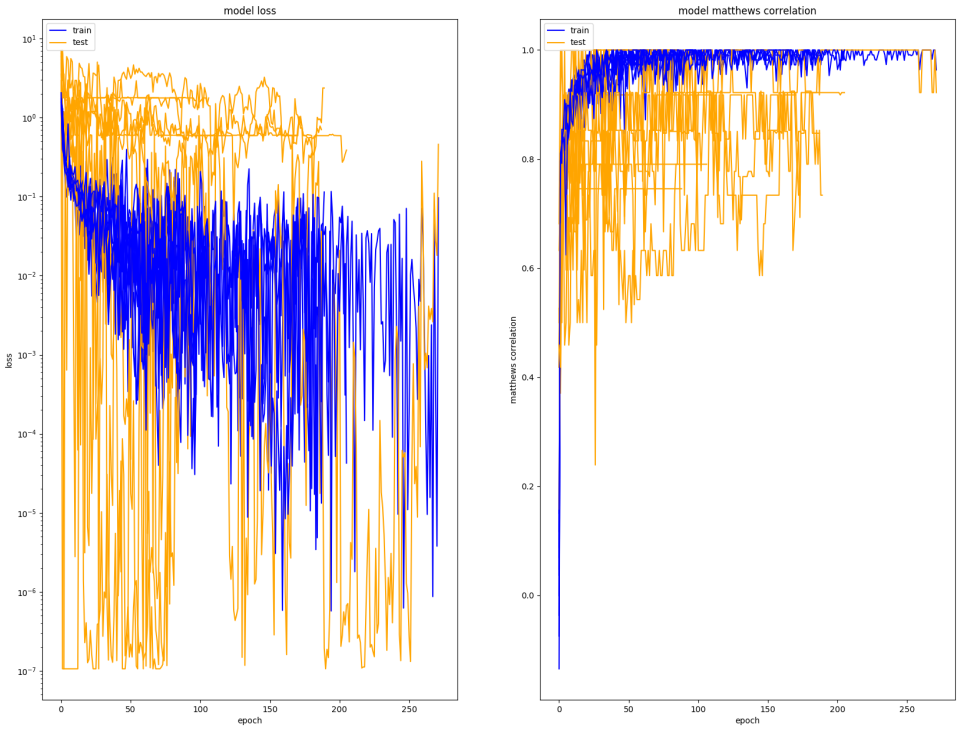


Figure 20.: Plot of loss (left) and MCC (right) through epochs for DNN on NB MA dataset.

Observing the graph from Figure 20 we can find it a little confusing to understand. This graph represents the loss and MCC over epochs. The orange line represents the test set,

while the blue line represents the training set. But because we have a 10-Fold cross validation pipeline, each fold has a represented line, and because we applied early stopping, each fold can have a different number of epochs. But trying to consolidate the number of epochs, we can admit that the optimum number is somewhere between 60 and 120, where MCC values from certain folds start to drop.

The first step was to test different epoch number, but now without early stopping. The results can be observed on graph from Figure 21. It is clear that the winner is 100 epochs. We will fix this epoch number and continue to tune some more parameters.

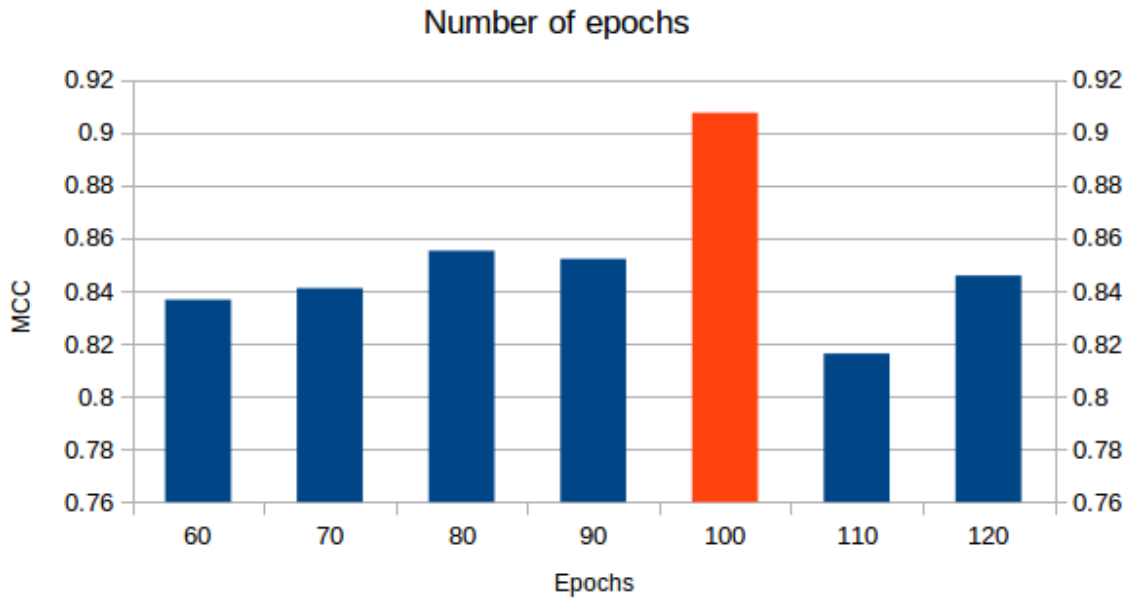


Figure 21.: Variation of MCC with number of epochs.

With respect to the dropout regularization parameter, we tested again several configurations, as we can observe on the graph from Figure 22.

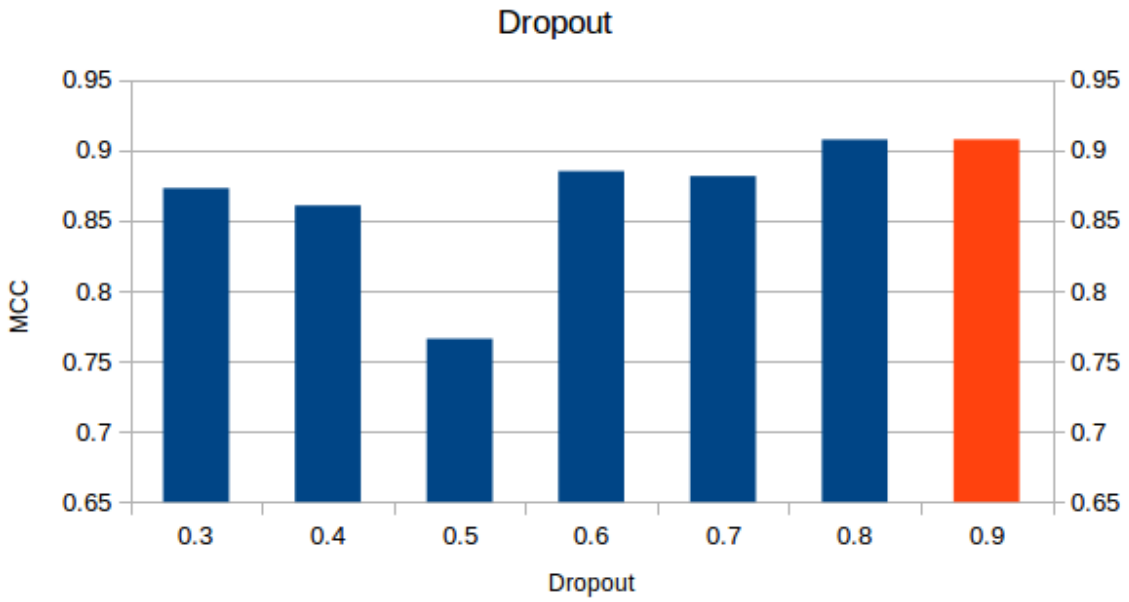


Figure 22.: Variation of MCC with dropout regularization parameter.

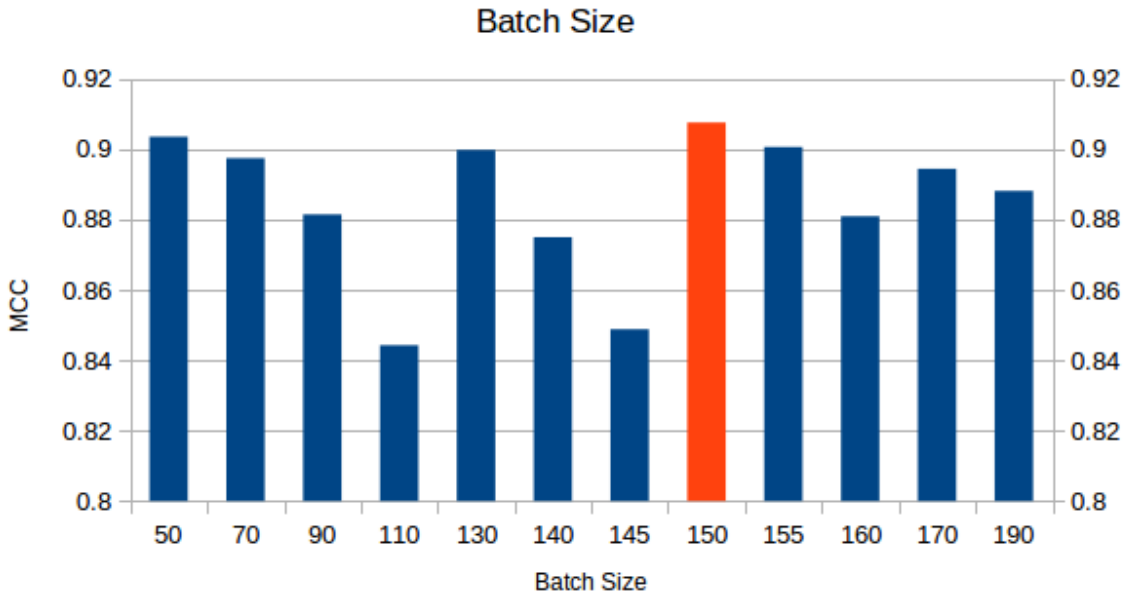


Figure 23.: Variation of MCC with batch size parameter.

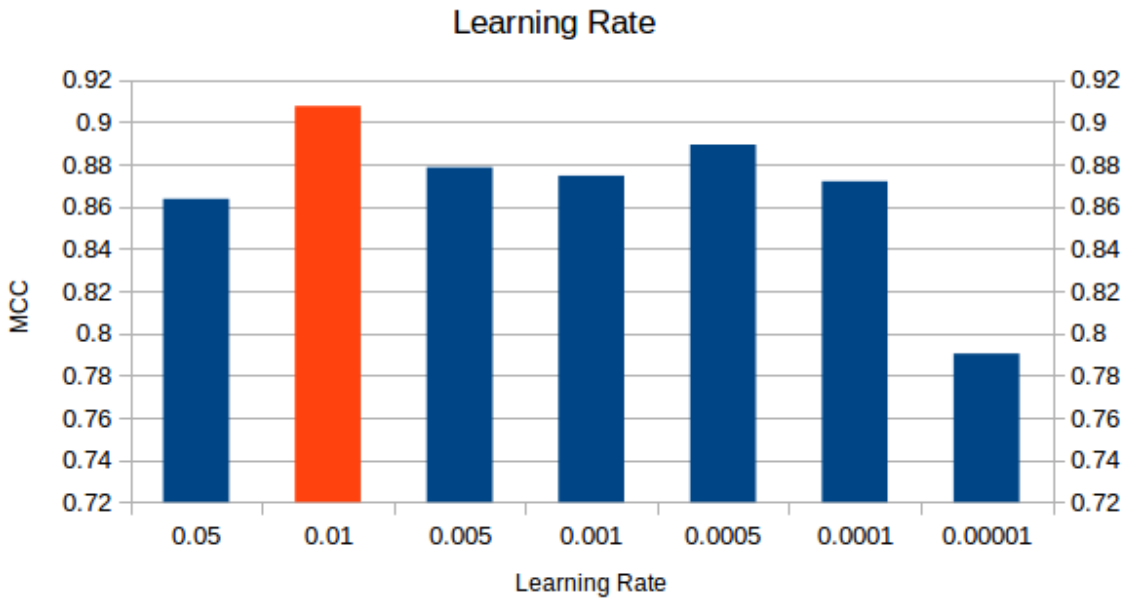


Figure 24.: Variation of MCC with learning rate parameter.

We reached the best configuration with the value of 0.9. Making changes to batch sizes resulted in the best value of 150 (graph from Figure 23). And, finally, the best learning rate value achieved was 0.01 (graph from Figure 24).

Overall, we improved our first model from 0.8801 $\pm$ 0.0917 to 0.9075 $\pm$ 0.0957 (Table 18).

Table 18.: NB DNN models results (MA) for CLASS LABEL endpoint.

Parameters	MCC
batch size: 150 dropout: 0.9 early stopping: False learning rate: 0.01 nb epoch: 100 optimization: RMSprop output activation: sigmoid units in hidden layers: [2500, 500] units in input layer: 5000	0.9075 $\pm$ 0.0957

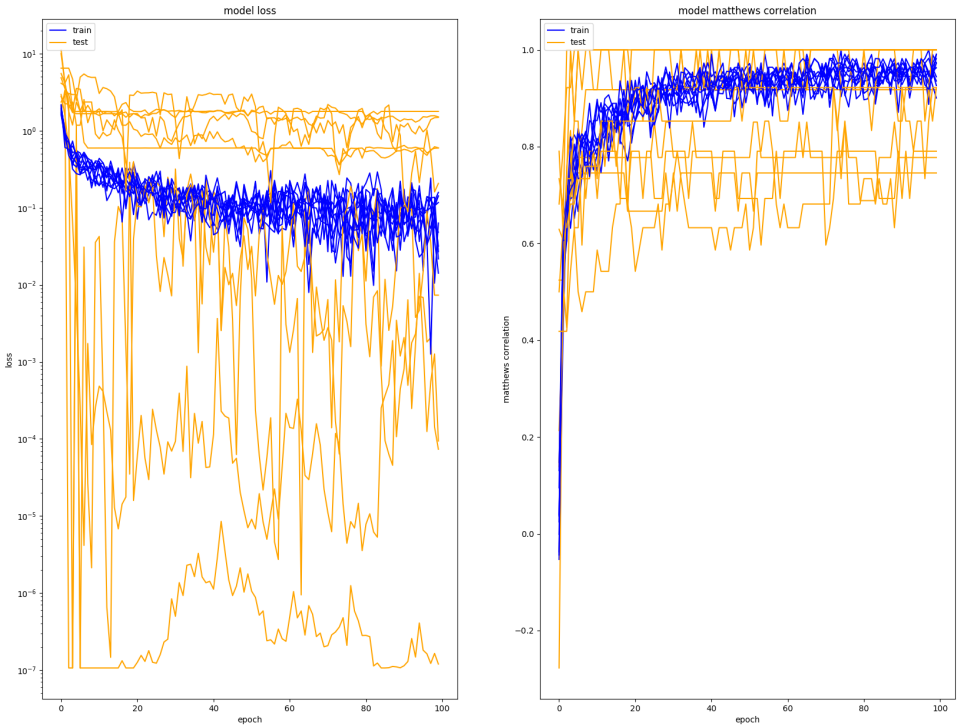


Figure 25.: Plot of loss and MCC against epochs for DNN on NB MA dataset.

The final model obtained with the best parameters selected originated a plot represented by the graph on Figure 25. Here, we can observe clearly the loss and MCC over the epochs because every fold ran the exact same amount of epochs (100). Another observation we can make is that not every fold achieves the same score, as expected. The error measurement was made, and we already compared it with the one from shallow models. But, the discrepancy that we observed from the plot lines from different folds makes us consider that our pipeline of 10-fold cross validation may not be the best one, suggesting that some overfitting could be occurring. If computational power would not be a problem, maybe a nested cross validation like the one used on the original paper [66] of 10 x 5-Fold cross validation would be preferable.

As we exploited before, there are more parameters that we can use to construct a DNN model. We can mention L1 and L2 regularization, weight initialization, momentum, we could choose not to use batch normalization or choose which layers to use, as well as dropout.

5.3 Fitting time of shallow vs. deep models comparison

We fixed an endpoint (EFS All) in the RNA-seq dataset and compared the fitting time of different models, as we can observe in the graph from Figure 26. The DNN model took more than 25 times more time on most cases.

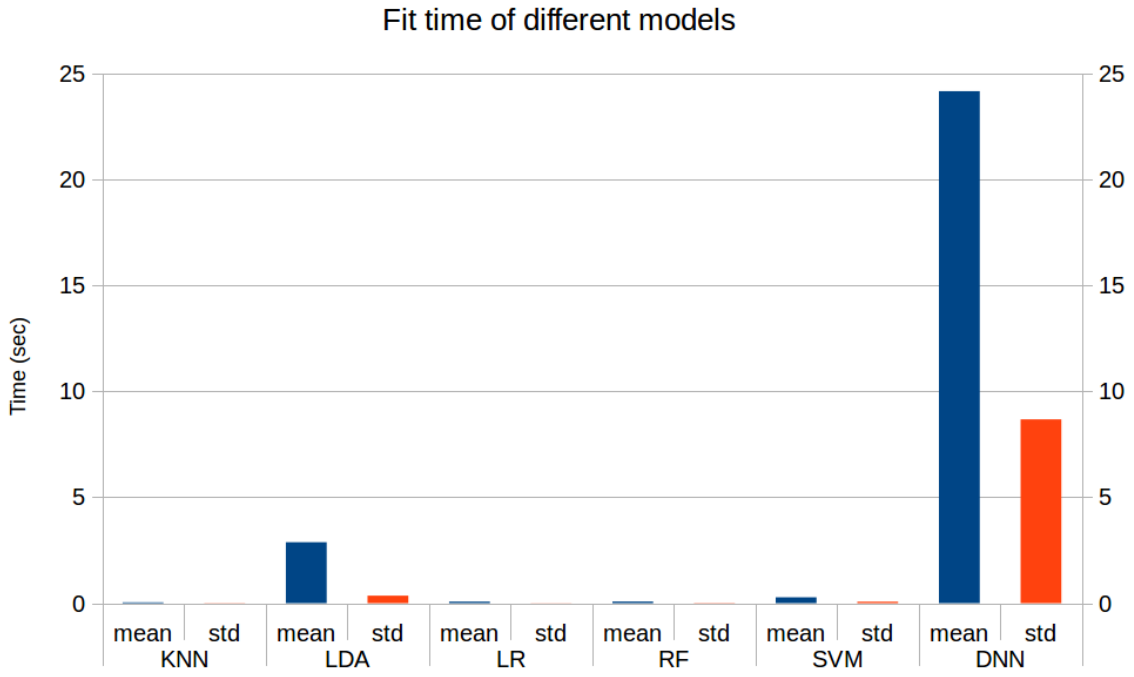


Figure 26.: Variation of fit time between shallow and deep models.

5.4 CPU vs. GPU fit time

As we can observe on the boxplot from Figure 27, a GPU is faster than a CPU. While a GPU fits a DNN model in a median time of 8.64 seconds, a CPU took a median of 90.48 seconds to perform the same task. We can say that, in this task, GPUs are over 10 times faster than CPUs.

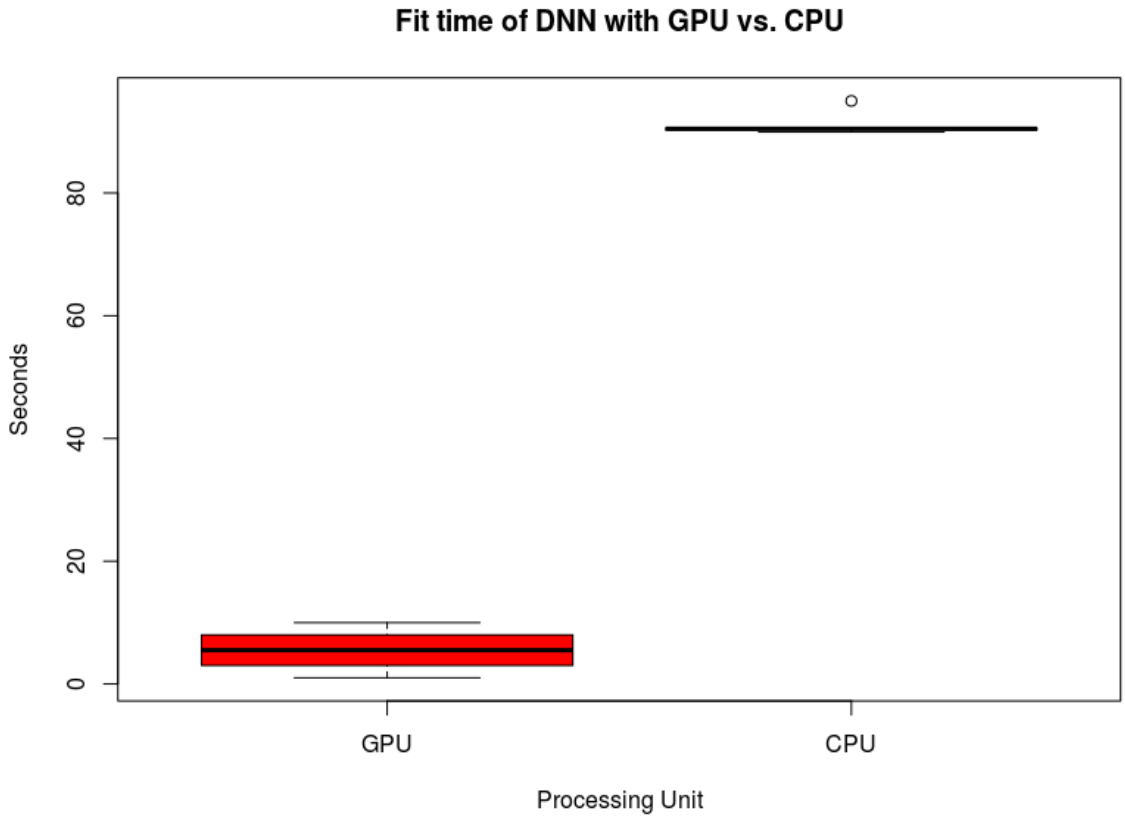


Figure 27.: Variation of fit time of DNN with GPU and CPU.

6

CONCLUSION

The application of deep learning methods to analyze gene expression data revealed to be somehow complex and computationally expensive. Training deep learning models took several folds more time than shallow models. On supervised learning, looking for the results we obtained, there seems to be a possibility to achieve better results when compared to shallow models like SVM, LR, RF, KNN and LDA. But it takes a huge amount of time as well as computational power to achieve that, which can be a problem.

Also, the DNN and MT-DNN models have shown some difficulties because of the nature of datasets. The number of samples is low. It was reflected especially on MT-DNN which dealt with only complete cases, that were below 200. The last two problems lead to a third, that was to develop a more robust pipeline where we could perform nested cross validation. Also, when using deep learning models we intend to capture the high dimensionality of the data. However, that was not entirely possible because we limited the number of features to 5000, where we had datasets with more than 30,000 and 60,000 features.

For unsupervised learning, the SA model seemed to show potential to reduce the dimensionality of data and capture the intricate patterns of gene expression datasets. But, one more time, it would be interesting to conduct experiments with more computational power, an adequate pipeline and try to increase the number of features. Another improvements could be to use a different feature selection method and try a different approach to the model selection step like using artificial intelligence algorithms to tune the parameters.

BIBLIOGRAPHY

[1] Camda 2017 conference, 2017. URL `http://www.camda.info`. Accessed: 2017-08-20.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[3] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

[4] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of dna-and rna-binding proteins by deep learning. *Nature biotechnology*, 33(8):831–838, 2015.

[5] American Cancer Society. Cancer Facts & Figures 2016. Technical report, American Cancer Society, Atlanta, 2016.

[6] Christof Angermueller, Tanel Pärnamaa, Leopold Parts, and Oliver Stegle. Deep learning for computational biology. *Molecular systems biology*, 12(7):878, 2016.

[7] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[8] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[9] Christopher M Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

[10] AC Braga, L Costa, and P Oliveira. An alternative method for global and partial comparison of two diagnostic systems based on roc curves. *Journal of Statistical Computation and Simulation*, 83(2):307–325, 2013.

[11] Ethan Cerami, Jianjiong Gao, Ugur Dogrusoz, Benjamin E Gross, Selcuk Onur Sumer, Bülent Arman Aksoy, Anders Jacobsen, Caitlin J Byrne, Michael L Heuer, Erik Larsson, et al. The cbio cancer genomics portal: an open platform for exploring multidimensional cancer genomics data, 2012.

[12] Siow-Wee Chang, Sameem Abdul-Kareem, Amir Feisal Merican, and Rosnah Binti Zain. Oral cancer prognosis based on clinicopathologic and genomic markers using a hybrid of feature selection and machine learning methods. *BMC bioinformatics*, 14(1): 170, 2013.

[13] Lujia Chen, Chunhui Cai, Vicky Chen, and Xinghua Lu. Trans-species learning of cellular signaling systems with bimodal deep belief networks. *Bioinformatics*, 31(18): 3008–3015, 2015.

[14] Yen-Chen Chen, Wan-Chi Ke, and Hung-Wen Chiu. Risk classification of cancer survival using ann with gene expression data from multiple laboratories. *Computers in biology and medicine*, 48:1–7, 2014.

[15] Yifei Chen, Yi Li, Rajiv Narayan, Aravind Subramanian, and Xiaohui Xie. Gene expression inference with deep learning. *Bioinformatics*, 32(12):1832–1839, 2016.

[16] Jie-Zhi Cheng, Dong Ni, Yi-Hong Chou, Jing Qin, Chui-Mei Tiu, Yeun-Chung Chang, Chiun-Sheng Huang, Dinggang Shen, and Chung-Ming Chen. Computer-aided diagnosis with deep learning architecture: applications to breast lesions in us images and pulmonary nodules in ct scans. *Scientific reports*, 6, 2016.

[17] François Chollet. keras. `https://github.com/fchollet/keras`, 2015. Accessed: 2017-02-05.

[18] George M Church. Genomes for all. *Scientific American*, 294(1):46–54, 2006.

[19] Joseph A Cruz and David S Wishart. Applications of machine learning in cancer prediction and prognosis. *Cancer informatics*, 2, 2006.

[20] Gregory Ditzler, Robi Polikar, and Gail Rosen. Multi-layer and recursive neural networks for metagenomic classification. *IEEE transactions on nanobioscience*, 14(6):608–616, 2015.

[21] Issam El Naqa, Ruijiang Li, Martin J Murphy, et al. Machine learning in radiation oncology. *Theory Appl*, pages 57–70, 2015.

[22] Konstantinos P Exarchos, Yorgos Goletsis, and Dimitrios I Fotiadis. Multiparametric decision support system for the prediction of oral cancer reoccurrence. *IEEE Transactions on Information Technology in Biomedicine*, 16(6):1127–1134, 2012.

[23] Rasool Fakoor, Faisal Ladhak, Azade Nazi, and Manfred Huber. Using deep learning to enhance cancer diagnosis and classification. In *Proceedings of the International Conference on Machine Learning*, 2013.

[24] George Forman and Martin Scholz. Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. *ACM SIGKDD Explorations Newsletter*, 12 (1):49–57, 2010.

[25] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics Springer, Berlin, 2001.

[26] Jianjiong Gao, Bülent Arman Aksoy, Ugur Dogrusoz, Gideon Dresdner, Benjamin Gross, S Onur Sumer, Yichao Sun, Anders Jacobsen, Rileen Sinha, Erik Larsson, et al. Integrative analysis of complex cancer genomics and clinical profiles using the cbioportal. *Science signaling*, 6(269):pl1, 2013.

[27] Levi A Garraway, Jaap Verweij, and Karla V Ballman. Precision oncology: an overview. *Journal of Clinical Oncology*, 31(15):1803–1805, 2013.

[28] Olivier Gevaert, Frank De Smet, Dirk Timmerman, Yves Moreau, and Bart De Moor. Predicting the prognosis of breast cancer by integrating clinical and microarray data with bayesian networks. *Bioinformatics*, 22(14):e184–e190, 2006.

[29] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[30] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. Accessed: 2017-02-05, 2016. URL `http://www.deeplearningbook.org`.

[31] Casey S Greene, Jie Tan, Matthew Ung, Jason H Moore, and Chao Cheng. Big data bioinformatics. *Journal of cellular physiology*, 229(12):1896–1900, 2014.

[32] Robert L Grossman, Allison P Heath, Vincent Ferretti, Harold E Varmus, Douglas R Lowy, Warren A Kibbe, and Louis M Staudt. Toward a shared vision for cancer genomic data. *New England Journal of Medicine*, 375(12):1109–1112, 2016.

[33] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[34] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[35] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[36] Anil K Jain, Robert P. W. Duin, and Jianchang Mao. Statistical pattern recognition: A review. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1):4–37, 2000.

[37] Zhicheng Jiao, Xinbo Gao, Ying Wang, and Jie Li. A deep feature based framework for breast masses classification. *Neurocomputing*, 197:221–231, 2016.

[38] Konstantina Kourou, Themis P Exarchos, Konstantinos P Exarchos, Michalis V Karamouzis, and Dimitrios I Fotiadis. Machine learning applications in cancer prognosis and prediction. *Computational and structural biotechnology journal*, 13:8–17, 2015.

[39] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

[40] Byunghan Lee, Junghwan Baek, Seunghyun Park, and Sungroh Yoon. deeptarget: end-to-end learning framework for microrna target prediction using deep recurrent neural networks. *arXiv preprint arXiv:1603.09123*, 2016.

[41] Taehoon Lee and Sungroh Yoon. Boosted categorical restricted boltzmann machine for computational prediction of splice junctions. In *ICML*, pages 2483–2492, 2015.

[42] Michael KK Leung, Hui Yuan Xiong, Leo J Lee, and Brendan J Frey. Deep learning of the tissue-regulated splicing code. *Bioinformatics*, 30(12):i121–i129, 2014.

[43] Aiguo Li, Jennifer Walling, Susie Ahn, Yuri Kotliarov, Qin Su, Martha Quezado, J Carl Oberholtzer, John Park, Jean C Zenklusen, and Howard A Fine. Unsupervised analysis of transcriptomic profiles reveals six glioma subtypes. *Cancer research*, 69(5):2091–2099, 2009.

[44] Muxuan Liang, Zhizhong Li, Ting Chen, and Jianyang Zeng. Integrative data analysis of multi-platform cancer data with a multimodal deep learning approach. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 12(4):928–937, 2015.

[45] Jennifer Listgarten, Sambasivarao Damaraju, Brett Poulin, Lillian Cook, Jennifer Dufour, Adrian Driga, John Mackey, David Wishart, Russ Greiner, and Brent Zanke. Predictive models for breast cancer susceptibility from multiple single nucleotide polymorphisms. *Clinical cancer research*, 10(8):2725–2737, 2004.

[46] Riccardo Miotto, Li Li, Brian A Kidd, and Joel T Dudley. Deep patient: An unsupervised representation to predict the future of patients from the electronic health records. *Scientific reports*, 6, 2016.

[47] Michael A. Nielsen. Neural Networks and Deep Learning, 2015. URL `http://neuralnetworksanddeeplearning.com/`. Accessed: 2017-02-05.

[48] Kanghee Park, Amna Ali, Dokyoon Kim, Yeolwoo An, Minkoo Kim, and Hyunjung Shin. Robust predictive model for evaluating breast cancer survivability. *Engineering Applications of Artificial Intelligence*, 26(9):2194–2205, 2013.

[49] L.M.C. Peixoto. Deep learning. `https://github.com/lmpeixoto/deepl_learning`, 2016.

[50] Vinay Prasad. Perspective: The precision-oncology illusion. *Nature*, 537(7619):S63–S63, 2016.

[51] Ladislav Rampasek and Anna Goldenberg. Tensorflow: Biology's gateway to deep learning? *Cell systems*, 2(1):12–14, 2016.

[52] Sebastian Raschka. *Python machine learning*. Packt Publishing Ltd, 2015.

[53] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[54] Ritambhara Singh, Jack Lanchantin, Gabriel Robins, and Yanjun Qi. Deepchrome: deep-learning for predicting gene expression from histone modifications. *Bioinformatics*, 32(17):i639–i648, 2016.

[55] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[56] Jie Tan, Matthew Ung, Chao Cheng, and Casey S Greene. Unsupervised feature construction and knowledge extraction from genome-wide assays of breast cancer with denoising autoencoders. In *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, volume 20, page 132. NIH Public Access, 2015.

[57] Jie Tan, Matthew Huyck, Dongbo Hu, René A Zelaya, Deborah A Hogan, and Casey S Greene. Adage signature analysis: differential expression analysis with data-defined gene sets. *BMC bioinformatics*, 18(1):512, 2017.

[58] Katarzyna Tomczak, Patrycja Czerwinska, Maciej Wiznerowicz, et al. The cancer genome atlas (tcga): an immeasurable source of knowledge. *Contemp Oncol (Pozn)*, 19(1A):A68–A77, 2015.

[59] Department of Health U.S. National Library of Medicine, National Institutes of Health and Human Services. What is precision medicine?, 2017. URL `https://ghr.nlm.nih.gov/primer/precisionmedicine/definition`. Accessed: 2017-02-05.

[60] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec): 3371–3408, 2010.

[61] Michael Waddell, David Page, and John Shaughnessy Jr. Predicting cancer susceptibility from single-nucleotide polymorphism data: a case study in multiple myeloma. In *Proceedings of the 5th international workshop on Bioinformatics*, pages 21–28. ACM, 2005.

[62] Dayong Wang, Aditya Khosla, Rishab Gargeya, Humayun Irshad, and Andrew H Beck. Deep learning for identifying metastatic breast cancer. *arXiv preprint arXiv:1606.05718*, 2016.

[63] Gregory P Way and Casey S Greene. Extracting a biologically relevant latent space from cancer transcriptomes with variational autoencoders. *bioRxiv*, page 174474, 2017.

[64] Xiaoyi Xu, Ya Zhang, Liang Zou, Minghui Wang, and Ao Li. A gene signature for breast cancer prognosis using support vector machine. In *Biomedical Engineering and Informatics (BMEI), 2012 5th International Conference on*, pages 928–931. IEEE, 2012.

[65] Sai Zhang, Jingtian Zhou, Hailin Hu, Haipeng Gong, Ligong Chen, Chao Cheng, and Jianyang Zeng. A deep learning framework for modeling structural features of rna-binding protein targets. *Nucleic acids research*, 44(4):e32–e32, 2016.

[66] Wenqian Zhang, Ying Yu, Falk Hertwig, Jean Thierry-Mieg, Wenwei Zhang, Danielle Thierry-Mieg, Jian Wang, Cesare Furlanello, Viswanath Devanarayan, Jie Cheng, et al. Comparison of rna-seq and microarray-based models for clinical endpoint prediction. *Genome biology*, 16(1):133, 2015.

[67] Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature methods*, 12(10):931–934, 2015.

A

CODE EXPLANATION

A.1 NBHIGHTHROUGHPUT

Table A.1.1.: NBHighThroughput properties.

Property	Description
ht_file	Hightroughput data path
cd_file	Clinical data path
exprs	Hightroughput gene expression matrix
clinical_data	Clinical data table
n_features	Number of features
n_samples	Number of samples
features	List of features (gene/transcript/probe names)
X	X matrix
y	y matrix
endpoint_name	Name of clinical endpoint currently selected

Table A.1.2.: NBHighThroughput methods.

Method	Description
read_exprs_data	Loads hightroughput file to a Pandas dataframe object
read_clinical_data	Loads clinical data file to a Pandas dataframe object
set_feature_number	Reads the number of columns in gene expression matrix and saves the number of features
set_list_features	Saves a list with the features (genes/transcripts/probes)
set_sample_number	Reads the number of rows in gene expression matrix and saves the number of samples
load_data	Runs all previously described methods to properly load expression file and clinical data
get_feature_number	Returns the number of features
get_list_features	Returns the list of features
get_sample_number	Returns the number of samples
mad_filter	Filters features by mean absolute deviation - feature selection
normalize_data	Scale expression data using StandardScaler normalization (Scikit learn)
endpoint_ENDPOINT	Returns X and y matrices according to the selected clinical endpoint
multi_task	Returns X and y matrices including all the endpoints to construct a MT-DNN

A.2 SHALLOWMODEL

Table A.2.1.: ShallowModel properties.

Property	Description
X	X matrix
y	y matrix
X_train, X_test, y_train, y_test	train/test split resulting matrices
model	Sci-kit learn model object returned by model selection function
model_name	Name of generated files related to that model
cv	Number of folds for k-fold cross validation
splitted	Boolean True if it is a train and test splitted experiment
feature_number	Number of features (genes/probes) of X matrix
scores	Storage of model selection scores
endpoint_name	Name of endpoint currently in execution
results	Storage of best models results
list_endpoints	List of endpoints to be tested
list_models	List of models to be tested

Table A.2.2.: ShallowModel methods.

Method	Description
print_parameter_values	Prints the selected model parameters
calculate_score	Calculates the selected model score with hold out test set
save_best_model	Saves a file with the best model
load_model	Loads a model from a file
model_selection_(MODEL)	Performs grid search model selection for each MODEL within a pipeline where the data is first normalized.
write_cv_results	Creates a *Comma Separated Values (csv)* file with the results from model selection in separated folders by endpoint and by model

A.3 DNNMODEL

Table A.3.1.: DNNModel properties.

Property	Description
X	X matrix
y	y matrix
X_train, X_test, y_train, y_test	train/test split resulting matrices
model	Keras model object returned by model selection function
valid_size	Factor of train/test split
feature_number	Number of features (genes/probes) of X matrix
parameters	Dictionary with neural network parameters - dropout, output activation function, optimization algorithm, learning rate, units in input layer, units in hidden layers, number of epochs, batch size, early stopping, patience.
cv	Number of folds for k-fold cross validation
splitted	Boolean True if it is a train and test splitted experiment
filename	Name of generated files related to that model
verbose	When 1 prints all the information that Keras generates during the training of DNN
parameters_batch	Dictionary with neural network parameters list to be tested in model selection function
model_selection_history	Storage of model selection scores and history of DNN training

Table A.3.2.: DNNModel methods.

Method	Description
print_parameter_values	Prints the selected model parameters
create_DNN_model	Using sequential model from Keras creates the DNN model based on parameters
fit_model	Fits the DNN model, calculating time of execution
print_fit_results	prints the fit results - loss and MCC
evaluate_model	Calculates the selected model score
model_selection	Performs random search model selection keeping track of fit history for each model tested.
find_best_model	After the model selection runs it picks the best model from model_selection_history
select_best_model	Loads the parameter configuration of the best model saved
batch_parameter_shuffler	Shuffles the parameters to feed the model selection function
set_filename	Sets the name of files to be generated
plot_model_performance	Creates a Matplotlib graph with history of loss and MCC of a model fit and writes a file with it in separated folders by endpoint
write_model_selection_results	Writes a csv file with the results of model selection in separated folders by endpoint
write_report	Writes a file with the results of a model fit in separated folders by endpoint

B

DETAILED RESULTS

B.1 RNA-SEQ RESULTS

Table B.1.1.: NB shallow models results (RNA-Seq) part 1.

Endpoint	Best Model	No. Features	Parameters	Scores
Sex	LR	100	C: 0.05 penalty: l1	roc auc: 0.9959 ± 0.0083 accuracy: 0.9960 ± 0.0080 f1 score: 0.9952 ± 0.0095 mcc: 0.9920 ± 0.0161 precision: 0.9955 ± 0.0136 recall: 0.9952 ± 0.0142 log loss: 0.1381 ± 0.2763
Class Label	SVM	5000	C: 0.0005 kernel: linear	roc auc: 0.9283 ± 0.0560 accuracy: 0.9447 ± 0.0415 f1 score: 0.9114 ± 0.0699 mcc: 0.8762 ± 0.0957 precision: 0.9535 ± 0.0579 recall: 0.8789 ± 0.1050 log loss: 1.9100 ± 1.4344
EFS All	LR	5000	C: 0.0005 penalty: l2	roc auc: 0.7403 ± 0.0763 accuracy: 0.7299 ± 0.0977 f1 score: 0.6824 ± 0.0849 mcc: 0.4933 ± 0.1491 precision: 0.6467 ± 0.1458 recall: 0.7787 ± 0.1645 log loss: 9.3274 ± 3.3731

Table B.1.2.: NB shallow models results (RNA-Seq) part 2.

Endpoint	Best Model	No. Features	Parameters	Scores
OS All	LR	1000	C: 0.0001 penalty: l2	roc auc: 0.7896 ± 0.1046 accuracy: 0.7043 ± 0.1840 f1 score: 0.6079 ± 0.1406 mcc: 0.5097 ± 0.1787 precision: 0.4740 ± 0.1621 recall: 0.9345 ± 0.0592 log loss: 10.2135 ± 6.3554
EFS HR	LDA	2000	n components: 1	roc auc: 0.5442 ± 0.0588 accuracy: 0.6941 ± 0.0521 f1 score: 0.8100 ± 0.0348 mcc: 0.1445 ± 0.1940 precision: 0.7034 ± 0.0413 recall: 0.9583 ± 0.0559 log loss: 10.5650 ± 1.7983
OS HR	LR	1000	C: 500 penalty: l1	roc auc: 0.5958 ± 0.1601 accuracy: 0.5957 ± 0.1594 f1 score: 0.5950 ± 0.1621 mcc: 0.1982 ± 0.3265 precision: 0.6256 ± 0.1605 recall: 0.5833 ± 0.1929 log loss: 13.9654 ± 5.5045

Table B.1.3.: NB DNN models results (RNA-Seq) part 1.

Endpoint	Parameters	Scores
Sex	batch size: 75 dropout: 0.6 early stopping: False learning rate: 0.015 nb epoch: 800 optimization: SGD output activation: sigmoid units in hidden layers: [2500, 500] units in input layer: 5000	roc auc: 0.9924 ± 0.0094 accuracy: 0.9920 ± 0.0098 f1 score: 0.9907 ± 0.0114 mcc: 0.9841 ± 0.0195 precision: 0.9866 ± 0.0205 recall: 0.9952 ± 0.0143 log loss: 0.2750 ± 0.3368
Class Label	batch size: 150 dropout: 0.6 early stopping: False learning rate: 0.005 nb epoch: 80 optimization: Adam output activation: sigmoid units in hidden layers: [1000, 100] units in input layer: 5000	roc auc: 0.9450 ± 0.0570 accuracy: 0.9521 ± 0.0471 f1 score: 0.9269 ± 0.0752 mcc: 0.8951 ± 0.1060 precision: 0.9375 ± 0.0803 recall: 0.9233 ± 0.0997 log loss: 1.6542 +/- 1.6255
EFS All	batch size: 80 dropout: 0.6 early stopping: False learning rate: 0.001 nb epoch: 800 optimization: SGD output activation: sigmoid units in hidden layers: [1000, 100] units in input layer: 5000	roc auc: 0.7163 ± 0.0783 accuracy: 0.7341 +/- 0.0820 f1 score: 0.6306 ± 0.1128 mcc: 0.4694 ± 0.1493 precision: 0.7096 ± 0.1762 recall: 0.6480 ± 0.2135 log loss: 9.1823 ± 2.8329

Table B.1.4.: NB DNN models results (RNA-Seq) part 2.

Endpoint	Parameters	Scores
OS All	batch size: 75 dropout: 0.2 early stopping: True learning rate: 0.015 nb epoch: 1000 optimization: SGD output activation: sigmoid units in hidden layers: [2500, 100] units in input layer: 5000	roc auc: 0.7798 ± 0.0813 accuracy: 0.7814 ± 0.1285 f1 score: 0.6236 ± 0.1333 mcc: 0.5284 ± 0.1707 precision: 0.5707 ± 0.1928 recall: 0.7745 ± 0.1562 log loss: 7.5502 ± 4.4389
EFS HR	batch size: 80 dropout: 0.6 early stopping: False learning rate: 0.015 nb epoch: 30 optimization: SGD output activation: sigmoid units in hidden layers: [2500, 100, 10] units in input layer: 5000	roc auc: 0.5925 ± 0.0999 accuracy: 0.6431 ± 0.0943 f1 score: 0.7305 ± 0.0797 mcc: 0.1985 ± 0.2166 precision: 0.7500 ± 0.0843 recall: 0.7250 ± 0.1294 log loss: 12.3257 ± 3.2561
OS HR	batch size: 80 dropout: 0.6 early stopping: False learning rate: 0.015 nb epoch: 25 optimization: SGD output activation: sigmoid units in hidden layers: [2500, 100] units in input layer: 5000	roc auc: 0.6047 ± 0.1546 accuracy: 0.6012 ± 0.1576 f1 score: 0.5733 ± 0.2042 mcc: 0.2211 ± 0.3197 precision: 0.6683 ± 0.1933 recall: 0.5622 ± 0.2356 log loss: 13.7735 ± 5.4442

B.2 MA results

Table B.2.1.: NB shallow models results (MA) part 1.

Endpoint	Best Model	No. Features	Parameters	Scores
Sex	LR	100	C: 0.05 penalty: l1	roc auc: 1.0 ± 0.0 accuracy: 1.0 ± 0.0 f1 score: 1.0 ± 0.0 mcc: 1.0 ± 0.0 precision: 1.0 ± 0.0 recall: 1.0 ± 0.0 log loss: 9.9920e-16 ± 1.9722
Class Label	SVM	5000	C: 0.0001 kernel: linear	roc auc: 0.9533 ± 0.0527 accuracy: 0.9632 ± 0.0406 f1 score: 0.9419 ± 0.0661 mcc: 0.9197 ± 0.0885 precision: 0.9700 ± 0.0640 recall: 0.9233 ± 0.997 log loss: 1.2704 ± 1.4016
EFS All	SVM	5000	C: 0.0001 kernel: linear	roc auc: 0.6982 ± 0.668 accuracy: 0.7269 ± 0.0585 f1 score: 0.5943 ± 0.1189 mcc: 0.4330 ± 0.1154 precision: 0.6946 ± 0.1431 recall: 0.5889 ± 0.2226 log loss: 9.4338 ± 2.020

Table B.2.2.: NB shallow models results (MA) part 2.

Endpoint	Best Model	No. Features	Parameters	Scores
OS All	LR	100	C: 0.001 penalty: l2	roc auc: 0.7761 ± 0.0673 accuracy: 0.7556 ± 0.1235 f1 score: 0.5997 ± 0.1090 mcc: 0.5021 ± 0.1259 precision: 0.5175 ± 0.1518 recall: 0.8100 ± 0.1650 log loss: 8.4403 ± 4.2665
EFS HR	SVM	2000	C: 0.001 kernel: linear	roc auc: 0.6217 ± 0.1180 accuracy: 0.6889 ± 0.0970 f1 score: 0.7791 ± 0.0725 mcc: 0.2704 ± 0.2521 precision: 0.7628 ± 0.1064 recall: 0.8167 ± 0.1225 log loss: 10.7455 ± 3.3502
OS HR	LR	1000	C: 10 penalty: l1	roc auc: 0.6240 ± 0.0746 accuracy: 0.6244 ± 0.0768 f1 score: 0.6207 ± 0.1322 mcc: 0.2738 ± 0.1644 precision: 0.6629 ± 0.0842 recall: 0.6467 ± 0.2334 log loss: 12.9745 ± 2.6536

Table B.2.3.: NB DNN models results (MA) part 1.

Endpoint	Parameters	Scores
Sex	batch size: 80 dropout: 0.9 early stopping: False learning rate: 0.001 nb epoch: 500 optimization: RMSprop output activation: sigmoid units in hidden layers: [1000, 100] units in input layer: 5000	roc auc: 0.9888 ± 0.0114 accuracy: 0.9899 ± 0.0101 f1 score: 0.9879 ± 0.0121 mcc: 0.9796 ± 0.0204 precision: 0.9955 ± 0.0136 recall: 0.9810 ± 0.0233 log loss: 0.3496 ± 0.3497
Class Label	batch size: 150 dropout: 0.9 early stopping: False learning rate: 0.01 nb epoch: 100 optimization: RMSprop output activation: sigmoid units in hidden layers: [2500, 500] units in input layer: 5000	roc auc: 0.9517 ± 0.0548 accuracy: 0.9563 ± 0.0452 f1 score: 0.9340 ± 0.0694 mcc: 0.9075 ± 0.0957 precision: 0.9425 ± 0.0806 recall: 0.9367 ± 0.1056 log loss: 1.5086 +/- 1.5612
EFS All	batch size: 60 dropout: 0.7 early stopping: False learning rate: 0.01 nb epoch: 80 optimization: SGD output activation: sigmoid units in hidden layers: [2500, 100, 100] units in input layer: 5000	roc auc: 0.7232 ± 0.0695 accuracy: 0.7590 +/- 0.0555 f1 score: 0.6281 ± 0.1118 mcc: 0.4871 ± 0.1257 precision: 0.7451 ± 0.1337 recall: 0.5874 ± 0.1949 log loss: 8.3240 ± 1.9176

Table B.2.4.: NB DNN models results (MA) part 2.

Endpoint	Parameters	Scores
OS All	batch size: 80 dropout: 0 early stopping: False learning rate: 0.015 nb epoch: 90 optimization: SGD output activation: sigmoid units in hidden layers: [2500, 500] units in input layer: 5000	roc auc: 0.7944 ± 0.0741 accuracy: 0.7673 ± 0.1288 f1 score: 0.6301 ± 0.1333 mcc: 0.5303 ± 0.1672 precision: 0.5328 ± 0.1834 recall: 0.8391 ± 0.0751 log loss: 8.0368 ± 4.4473
EFS HR	batch size: 60 dropout: 0.6 early stopping: False learning rate: 0.005 nb epoch: 80 optimization: SGD output activation: sigmoid units in hidden layers: [2500, 1000, 500] units in input layer: 5000	roc auc: 0.5983 ± 0.0914 accuracy: 0.6314 ± 0.0853 f1 score: 0.7123 ± 0.0955 mcc: 0.2034 ± 0.1811 precision: 0.7654 ± 0.1087 recall: 0.7000 ± 0.1546 log loss: 12.7321 ± 2.9454
OS HR	batch size: 75 dropout: 0.6 early stopping: False learning rate: 0.005 nb epoch: 100 optimization: Adam output activation: sigmoid units in hidden layers: [1000, 100] units in input layer: 5000	roc auc: 0.6488 ± 0.0906 accuracy: 0.6488 ± 0.0853 f1 score: 0.6590 ± 0.0700 mcc: 0.3093 ± 0.1881 precision: 0.7016 ± 0.1367 recall: 0.6489 ± 0.1145 log loss: 12.1303 ± 2.9455

B.3 MT-DNN results

Table B.3.1.: NB MT-DNN models results (RNA-Seq) part 1.

Endpoint	Scores
Sex	roc auc: 0.5215 ± 0.0649 accuracy: 0.6876 ± 0.1123 f1 score: 0.7992 ± 0.0909 mcc: 0.0604 ± 0.2120 precision: 0.6908 ± 0.1264 recall: 0.9652 ± 0.0451 log loss: 10.7910 ± 3.8791
Class Label	roc auc: 0.5253 ± 0.0623 accuracy: 0.6931 ± 0.1125 f1 score: 0.8031 ± 0.0916 mcc: 0.0754 ± 0.2016 precision: 0.6925 ± 0.1267 recall: 0.9730 ± 0.0438 log loss: 10.5989 ± 3.8860
EFS All	roc auc: 0.5381 ± 0.1707 accuracy: 0.5402 ± 0.1880 f1 score: 0.5196 ± 0.2695 mcc: 0.0919 ± 0.3717 precision: 0.5209 ± 0.2978 recall: 0.5821 ± 0.2767 log loss: 15.8813 ± 6.4935

Table B.3.2.: NB MT-DNN models results (RNA-Seq) part 2.

Endpoint	Scores
OS All	roc auc: 0.5467 ± 0.1373 accuracy: 0.5575 ± 0.1627 f1 score: 0.5314 ± 0.2617 mcc: 0.1010 ± 0.3178 precision: 0.5190 ± 0.2837 recall: 0.6016 ± 0.2692 log loss: 15.2830 ± 5.6204
EFS HR	roc auc: 0.5442 ± 0.0588 accuracy: 0.6941 ± 0.0521 f1 score: 0.8100 ± 0.0348 mcc: 0.1445 ± 0.1940 precision: 0.7034 ± 0.0413 recall: 0.9583 ± 0.0559 log loss: 10.5650 ± 2.0349
OS HR	roc auc: 0.5811 ± 0.1651 accuracy: 0.5784 ± 0.2141 f1 score: 0.3101 ± 0.2474 mcc: 0.1893 ± 0.2874 precision: 0.4983 ± 0.4000 recall: 0.2781 ± 0.2980 log loss: 14.5610 ± 7.3946