UNIVERSITÀ DEGLI STUDI DI PISA

DIPARTIMENTO DI INFORMATICA

CORSO DI LAUREA SPECIALISTICA IN INFORMATICA

TESI DI LAUREA

Predicting mortality in low birth weight infants: a machine learning perspective

Autore:
Marco PODDA

Relatori:
Prof. Alessio MICHELI
Dott. Davide BACCIU

Controrelatore:
Prof.ssa Anna BERNASCONI

ANNO ACCADEMICO 2015-2016

Acknowledgements

First and foremost, I would like to thank Prof. Micheli and Doct. Bacciu, that followed me in the development of this thesis. I was initially intrigued, and I then decided to dedicate my life to work and study in the Machine Learning field mainly because of their commitment and expertise. Many thanks also to Doct. Gagliardi an Doct. Placidi, whose help was the key to the success of this work.

Secondly, I thank my family for their endless support and the motivation they never ceased to provide. You were always there for me, and I'm pretty sure I couldn't do it without you.

Last but not least, I thank all my friends, the ones I made in Pisa and the ones I carry with me since I was a child. We shared many great moments during this years, I hope we'll share many others together.

Contents

A mia zia Giannina, e a mia madrina Giannina.

Chapter 1

Introduction

These last years, machine learning has emerged as one of the most prolific research fields in computer science, in conjunction with the availability of large quantities of data and the increasing interconnection of people and things. With such huge amounts of information, the most common tools of statistics to describe and predict from data quickly became outdated, and their results sometimes misleading or incomplete: that is because most of these tools make the assumption that the relations that underlie the data are linear. Machine learning models, on the other hand, are thought and created with the intent to exploit the complex, potentially non-linear, interactions in the data, that oftentimes lead to a better representation of the problem and consequently to an improvement of the results. Among the many fields where machine learning has proven to be successful, we mention image, speech and sound recognition.

Another field where machine learning proved to be extremely useful is data description and understanding, especially when large quantities of information are in play. Algorithms like clustering and dimensionality reduction are nowadays essential to make sense of data, because their results can be easily translated in additional domain knowledge by field experts.

One of the fields where machine learning is not yet used up to its full potential is medical research, in particular in the subfields of diagnosis and outcome prediction. Especially in the medical community, linear models are still preferred when it comes to predict from data. This scenario is changing with the latest progresses in medical research, that have been accompanied by a literal flood of data: patient records, exam outcomes, genomic information are now digitally collected on a daily basis and have reached a size that cannot be managed anymore if not with carefully crafted techniques. The choice to use models such as Logistic Regression for prediction was understandable up until some years ago, since they have strong theoretical background from statistics and offer an interpretation of the predicting process that remains unmatched by many machine learning models. In addition, complex models required an amount of computing power that we did not possess at the time. Now that the technology gap has been filled and data is everywhere, however, we believe that machine learning has to be considered the primary tool

(used alone or together with more traditional methods of analysis) to obtain more information, and to obtain the correct ones.

1.1 Motivation

The problem of predicting mortality in low birth-weight infants is well studied in medicine, although not to guide therapeutic decisions. Instead, mortality probabilities are used in comparison studies aimed to identify Neonatal Intensive Care Units (NICUs) where the observed mortality rate differs significantly from the predicted rate. A major contribution to this studies comes from networks of hospitals around the world, which began to collect data from NICUs to improve the quality of health care of infants. The data used in this study comes from one of such networks, the Vermont Oxford Network (VON), and specifically from its Italian member centers. This thesis was conceived in partnership with a committee of clinicians from a VON member center: Dr. Luigi Gagliardi, and Dr. Giulia Placidi, from the Neonatology ward of Ospedale Versilia, Lido di Camaiore (LU), which provided us with the data, the expertise that allowed us to gain a basic understanding of the subject and the intelligence to resolve any technical doubt on the specific task.

In this study, we intend to investigate the problem of predicting mortality in a cohort of low birth-weight infants from Italy. To do so, we will use different machine learning algorithms and techniques and assess their possible qualitative and quantitative contribution in improving the predictive capabilities of infant mortality models. Our approach includes:

- the training and evaluation of a pool of of state-of-the-art machine learning models with different complexities, comparing their results against a well-established linear probabilistic classifier constructed by the VON, called VON-Risk Adjusted model (VON-RA), that has been used for many years by the medical community as a reference;
- the assessment of the impact in performance that a promising set of feature, identified by the neonatologists, have with respect to the benchmark model, as well as a survey on how models perform on the task in relation to their complexity;
- a better characterization of the problem in general, by detecting which features are the most influential in the final outcome and whether the data has inherent structure that might be exploited in the future.

The training and evaluation of the pool of models will be conducted with experiments designed to rigorously follow machine learning best practices. At the highest level, the trials include choosing a pool of candidate models to be tested, estimating the performance of the whole class of models to which each candidate belongs,

build a working model from each candidate with the highest expected performance on unseen data, and evaluate its behavior in a real-case scenario. The results are obtained, validated and discussed using performance measures and statistical tools that are widely known and accepted in the medical community, in such a way that they are understandable by both clinicians and computer scientists.

The experiments spawned a series of challenges inherent of the main objective, that are typical in machine learning contexts. Some of them are related to the clinical domain from which the data comes from. Typically, clinical data is incomplete and badly formatted: this problem is explainable by considering that the data is aggregated from multiple sources. Although rigid guidelines to collect the data are in place in multi-center organizations, the differences between hospitals and, more often, the human error, produce a data base that is heterogeneous and cannot be processed as-is by algorithms. This problem is initially characterized through a statistical analysis, in order to detect inconsistencies in the data along with peculiarities that we want to exploit. After that, we transformed the data, with the two-fold intent to make it suitable to be used by the algorithms, and more easily analyzable with classical statistics tools. We focus in particular on the handling of missing data, in the attempt to retain features that make our analysis original, but at the same time preserving a sufficient number of observations to ensure good quality of the predictions.

One other challenge in this study comes from the nature of the task itself. Death in low birth-weight infants is a rare outcome, mainly thanks to the advances of neonatology research and new clinical practices, which shrunk the impact of illnesses and complications that might lead to death before and after birth to a minimum. Even if the main performance metric used in this work to evaluate the models is not affected by this reassuring disproportion between infants that die and infants that survive, we will take it into consideration and reason about its implications.

A last challenge is strictly related to machine learning, and comes from the peculiarities of the different models that were examined in our studies. Learning algorithms are generally conceived following paradigms that aim to solve the learning problem from a specific perspective. When it comes to practice, this heterogeneity is translated into a variety of algorithm-dependent constraints which have to be met in order to unleash the true predictive power of the model. Our experiments are thus designed to be general, in order to provide a unified method of comparison between all the different algorithms, but at the same time flexible enough to account for such practical differences.

The third scenario we explore in this study is related to interpretation. As we talked earlier, the size of datasets nowadays is orders of magnitude bigger than it was before. More than that, data grows in two directions: the number of observations is increasing, as well as the quantity of information that a single observation holds.

Large quantities of data are difficult to treat exclusively with descriptive statistics, since they contain so much variability that common measurements do not express their true structure. For this reason, we utilized specific machine learning models such as the Self-Organizing Maps to provide field experts with visual representations of the data, which can then be used to facilitate their interpretation and dig up hidden knowledge. Another aspect that we analyzed is the single contribution that each feature in the data has with respect to the final prediction. To do so, we utilized pure statistical tools such as box plots and data exploration techniques in general, along with an analysis that employed a common machine learning model (Decision Trees) to derive a ranking of features based on their influence in the determination of the final outcome.

1.2 Thesis outline

Chapter **2** provides a brief excursus in the field of machine learning: its main concepts and the analytical description of the algorithms and tools we will use throughout this work. Chapter **3** will provide some medical background on the problem and describe the data we were given; in addition, we will explain how the information contained in the dataset was processed to be used for training machine learning models. Finally, we are going to describe in detail how the experiments were set-up. In Chapter **4**, we will present and discuss the results of our experiments, while in Chapter **5** we will draw our conclusions and point towards future work that might follow from this thesis.

Chapter 2

Machine Learning

Machine learning is a discipline that provides data-driven methodologies and algorithms to tackle problems where an analytical solution is unknown, or impractical to obtain. In a typical machine learning use case, there is an unknown function that one tries to "learn" from data; this function is later used to solve some domain-specific problem, or *task*.

Learning tasks are usually categorized into *predictive* and *descriptive*. Predictive tasks, as the name implies, deal with the prediction of outcomes based on historical data, while descriptive tasks relate to the characterization of a problem and the discovery of hidden structure in the data. Each task is associated with a specific form of learning, where with the term learning we intend improving of some performance at the task through the use of data [37]. Predictive tasks are related to *supervised learning*, while descriptive tasks are related to *unsupervised learning*.

2.1 Binary classification

In supervised learning, data consists of a set of labeled vectors and we wish to learn the relationship between these vectors and the labels. Based on the domain of the labels y, we distinguish *regression problems*, where $y \in \mathbb{R}$ and *classification problems*, where the set of labels is a discrete and finite domain (in the latter case, the labels are also called *classes*). Our study is focused on *binary classification*, which is a particular instance of classification where the set of labels is a dichotomy (i.e. it contains exactly two classes, which are often referred to as *positive* and *negative* class).

2.1.1 Problem definition

Notation

We will now briefly describe a basic framework to define binary classification. A more in-depth definition can be found for example in [52].

We define our labeled data as a set of N[1] pairs $D = \{\mathbf{x}_i, y_i\}_{i=1}^{N}$, where $\mathbf{x} \in \mathbb{R}^d$ are d-valued real vectors (called *feature vectors*, and their components *features*), and $y \in Y \stackrel{\text{def}}{=} \{negative, positive\}$. We assume that the data comes from a fixed but unknown joint probability distribution $F(\mathbf{x}, y)$. D is usually called *training data*.

From now on, we are going to adopt the convention that the generic index i indicates the i-th row of the training data when it is subscripted, and the i-th feature of a fixed vector when it is superscripted and parenthesized (for example, the notation $\mathbf{x}_j^{(i)}$ indicates the i-th feature of j-th vector in the training data). Whenever possible, we will use vector notation to avoid scripting the index at all. As an additional mathematical convenience, we will use 0 to indicate the negative class and 1 to indicate the positive class, thus $y \in Y \stackrel{\text{def}}{=} \{0, 1\}$.

Formulation

We can express the binary classification problem as the problem of inferring the conditional distribution $F(y|\mathbf{x})$ using D, restricting ourselves to the case where F is a function $f : \mathbb{R}^d \to \{0, 1\}$ such that $f(\mathbf{x}) = y$. Since the exact inference of f is generally impossible because $F(\mathbf{x}, y)$ is unknown, we resort to function approximation: assuming the existence of a space of functions $\mathcal{H}$ of the form $h : \mathbb{R}^d \to \{0, 1\}$, we search in this space for the best approximation of f. $\mathcal{H}$ is called *hypothesis space* and its members *hypotheses*.

To quantify the error that a hypothesis makes over the all the possible realizations of the training data, we introduce the *risk* of a hypothesis as the expected loss under $F(\mathbf{x}, y)$:

$$R(h) = \int L(h(\mathbf{x}), y)\, dF(\mathbf{x}, y),$$

where $L : \{0, 1\} \times \{0, 1\} \to \mathbb{R}$ is a *loss function* that for a given data point $\mathbf{x}$, measures how much the prediction $h(\mathbf{x})$ differs from the true output y.

The binary classification problem can thus be posed as the search of an optimal $h^* \in \mathcal{H}$ that minimizes the expected loss:

$$h^* = \arg\min_{h \in \mathcal{H}} R(h).$$

2.1.2 Learning algorithms and generalization

To solve a binary classification problem, different *learning algorithms* can be used. Informally, a learning algorithm $\mathcal{A}(L, D)$ is a function that receives as input a particular loss function and the training data, and finds h^* by minimizing the loss function using the data.

[1] Unless otherwise specified, for the rest of this chapter we are going to assume that our data has size N.

To be able to do so, learning algorithms must have some way to represent the particular hypothesis space they explore. Usually, this consists in one or more real-valued vectors θ, called *parameters* or *weights*: different hypotheses correspond to different values of the parameters. To incorporate this notion, we will rewrite the generic hypothesis $h \in \mathcal{H}$ as h_θ. Such parametrized representation of the hypothesis space is called *model*.

Given a particular model, the binary classification problem can be formulated equivalently as the search for the best parameters θ^* that minimize the expected loss:

$$\theta^* = \arg\min_\theta R(h_\theta), \quad h_\theta \in \mathcal{H}.$$

As stated before, $R(h_\theta)$ is not directly computable since the joint distribution $F(\mathbf{x}, y)$ is unknown. In practice, the learning algorithm minimizes a related and computable quantity, the so-called *empirical risk* (or *training error*) R_D, defined as the average loss over all the training samples:

$$R_D(h_\theta) = \frac{1}{N} \sum_{i=1}^{N} L(h_\theta(\mathbf{x}_i), y_i).$$

Hence, the minimization objective becomes:

$$\theta^* = \arg\min_\theta R_D(h_\theta), \quad h_\theta \in \mathcal{H}.$$

This principle is called *empirical risk minimization* (ERM).

The ultimate goal for a learning algorithm is to achieve *generalization*: that is, the final hypothesis must be capable to predict correctly not only the training data, but especially new, previously unseen examples. Being sure that this happens is not obvious, since true generalization could be verified only on the whole data population, but all we have at our disposal is a limited number of examples. Ideally, we would want the empirical risk (which is the in-sample error of the hypothesis) to "track" the true risk (the out-of-sample error, on the whole population), resulting in their absolute difference being minimal.

One useful result coming from the field of *statistical learning theory* tells us that the following holds:

$$R(h_\theta) \leq R_D(h_\theta) + \varepsilon(N, M, \delta).$$

That is, we can construct an upper bound for the true risk (with confidence $1 - \delta$), using the empirical risk of the hypothesis and an additional term ε that depends on the number of samples N and the *complexity* (or *capacity*) of the model M, which, for some classes of models and for our purposes, we can consider roughly equivalent to the number of parameters of the model[2].

[2] M actually refers to the *VC-dimension* of the model, whose definition is outside the scope of this thesis. We strongly recommend reading [56] to understand this concepts in detail.

The implications of this result are two-fold: on one hand, it provides justification for using ERM instead of the true risk in the optimization problem; on the other hand, it gives us insurance that we can indeed construct useful models that are able to generalize (with an upper bound on the risk) even with a limited amount of data.

2.1.3 Overfitting and regularization

During the so-called *training* phase, where the hypothesis space is explored to search for the optimal h, we oftentimes observe that hypotheses learn perfectly how to classify the training data, but are not able to generalize their predictions to unseen examples. This behavior is closely related to the complexity of the model. As the complexity of a model increases, a hypothesis is able to discriminate better and better the training examples, up to a point where it learns aspects of the data that are present in that particular training set (maybe because of noise or pure chance), but not in the larger population from which the training set was sampled. This problem is known as *overfitting*.

Typically, machine learning algorithms provide "handles" that allow to control the *bias-variance trade-off* of the model they implement to prevent overfitting [16]. These handles are known as *regularization techniques*, and they basically work by reducing the variance (the dependence from a particular choice of training set) of the model.

2.2 Models for binary classification

We will now describe models and learning algorithms that were used in this study, exposing;

- how the predictions are computed (the so-called of *prediction rule* of the algorithm);
- how the optimal values for the weights are calculated;
- how regularization is achieved.

For in-depth details, we will refer to the corresponding literature where needed.

2.2.1 Logistic Regression

Logistic Regression [20] is the mainstay model for classification used in a broad variety of fields, especially biostatistics. It originated in the statistics field, but it fits the supervised learning paradigm as it assumes the same setting we discussed.

The Logistic Regression model is defined as follows:

$$\mathcal{H} = \{h_{\theta_0,\theta} \mid h_{\theta_0,\theta}(\mathbf{x}) = sigm(\theta_0 + \theta^T \mathbf{x})\},$$

where

$$sigm(x) = \frac{1}{1 + e^{-x}}$$

is the *sigmoid function*, whose output domain is the real interval $(0, 1)$, and the additional weight θ_0 is called *bias*.

The great advantage of using Logistic Regression comes from its simplicity, the scalability for very large datasets, and the interpretation it provides in terms of how a unit change in one feature $\mathbf{x}^{(i)}$ changes the *log-odds* of its associated parameters $\theta^{(i)}$ [27]. In contrast, its main limitation is the shape of the *decision boundary* (the hyper-surface that a model uses to separate the positive class from the negative class in feature space) that this model induces. For Logistic Regression, such decision boundary is linear; depending on the degree of non-linearity of the true decision boundary, it can fail to capture the true underlying relation between the feature vectors and the targets, resulting in a poor separation between the two classes. When this happens, Logistic Regression is abandoned in favor of models that are able to represent more complex decision boundaries.

Prediction rule

Logistic Regression models directly estimate the conditional distribution $F(y|\mathbf{x})$. Thus their output is a probability; this relaxation can be constrained back to the usual $Y \stackrel{\text{def}}{=} \{0, 1\}$ discrete domain by choosing as the final prediction for a feature vector $\mathbf{x}$ the class that maximizes the conditional probability.

The Logistic Regression model is derived starting from the assumption that the labels are distributed according to a Bernoulli distribution parametrized by p, the conditional probability of one of the two classes. In the following, we will assume $p = \mathbb{P}(1|\mathbf{x})$ and consequently $1 - p = \mathbb{P}(0|\mathbf{x})$.

The linear combination $\theta_0 + \theta\mathbf{x}$ identifies a hyperplane in $\mathbb{R}^d$ (shifted from the origin by a quantity θ_0) that separates the negative from the positive class. In fact, given a positive example $\mathbf{x}_{pos}$, we have $\theta_0 + \theta\mathbf{x}_{pos} \geq 0$; conversely, if $\mathbf{x}_{neg}$ is a negative example, $\theta_0 + \theta\mathbf{x}_{neg} < 0$.

We wish to map this relation to a probabilistic domain. Such mapping is accomplished through the *logit* function, which is the logarithm of the odds-ratio of p, defined as $logit(p) = \log\left(\frac{p}{1-p}\right)$. To see how this is possible, note that $logit(p) \geq 0$ whenever $p \geq 1-p$ (that is when the odds-ratio is in favor of class 1), and $logit(p) < 0$ otherwise.

The Logistic Regression equation can be therefore written as:

$$\log\left(\frac{p}{1-p}\right) = \theta_0 + \theta^T \mathbf{x},$$

which we can solve for the distribution parameter p, obtaining the following prediction rule:

$$p = \frac{1}{1 + e^{-\theta_0 + \theta^T \mathbf{x}}} \stackrel{\text{def}}{=} sigm(\theta_0 + \theta^T \mathbf{x})$$

Indeed, the Logistic function is the inverse of the logit: $logit^{-1}(x) = sigm(x)$. Once p is estimated, we can calculate the conditional probability of the negative class with $1 - p$.

Training Algorithm

In order to provide the best hypothesis, Logistic Regression minimizes its associated loss function, which is the log-likelihood of the data given the parameters, defined as:

$$L(h_\theta(\mathbf{x}_i), y_i) = \mathcal{L}(\mathbb{P}(\mathbf{x}_i), y_i|\theta)) \stackrel{\text{def}}{=} -\sum_{i=1}^{N} y_i \; log(\alpha_i) + (1 - y_i) \; (1 - log(\alpha_i))$$

where $\alpha_i = sigm(\theta_0 + \theta^T \mathbf{x}_i)$. Thus, the optimal values for the parameters can be obtained through Maximum Likelihood Estimation (MLE). One common method to perform MLE is *gradient descent*, which iteratively updates the weights using the negative gradient of the log-likelihood (that is indeed differentiable), thus moving the weight vector in the direction of steepest descent towards the minimum of the loss function. For a more thorough explanation of gradient descent, see for example [14].

Regularization

Regularization in Logistic Regression is implemented by adding a penalty term C to the loss function, in order to penalize hypothesis on the basis of the number of parameters they use (according to the famous *Occam's razor principle*). Among the schemes that were developed (many of them are illustrated in [20]), the most common is the *L2* penalty term. Logistic Regression with an additional L2 penalty term is often referred to in literature as *ridge regression*. The penalty term is defined as:

$$C(h_\theta) = \lambda \left\|\theta\right\|_2^2 = \lambda \sum_{i=1}^{d} {\theta^{(i)}}^2.$$

By using this regularization scheme, the values of the weights are shrunk by a factor λ (which controls the strength of the imposed regularization). Since this penalty

eventually constrains some parameters to have small values, their contribution in the Logistic Regression equation becomes less and less important: this results in the model effectively making use of a minor number of parameters, which serves to decrease the hypothesis variance.

2.2.2 k-Nearest Neighbor

The k-Nearest Neighbor (k-NN) learning algorithm [20] is an instance of so-called *memory-based* models, which do not perform actual learning, but compare new examples to the available data set, which remains stored to be used for prediction, as opposed to the other types of learning algorithms that do not need training samples anymore once the optimal values of the weights are computed. Despite its apparent naivety, k-Nearest Neighbors is seldom utilized in classification problems, for its simplicity and the fact that the hypothesis complexity increases with the size of the dataset.

Prediction Rule

The prediction rule for the k-NN learning algorithm is straightforward, and does not require a previous training phase: all the computation is deferred to the prediction phase. Once a new pattern is presented, the algorithm simply finds k examples in the training set that are more similar to the example, according to some distance measure (typically, euclidean distance is chosen but there are also other metrics in use depending on the specific task). Then, a class is assigned to the new observation by majority vote among its neighbors. In cases where k is an even number, ties are broken at random. Note that, to assign probabilities instead of a class, it is sufficient to compute the ratio between examples in the neighbor belonging to one class and k.

Regularization

The regularization parameter in the k-NN learning algorithm is the size of the neighborhood. The larger the value of k, the more populated the neighbor will be, resulting in a smoother decision boundary that, if set correctly, allows good generalization. On the other hand, a neighborhood size of 1 generally causes overfitting, especially for large datasets, but has some nice theoretical properties: it has been demonstrated [12] that the 1-NN error rate is at most twice as the Bayes error rate, which is the optimal error rate that one can theoretically obtain in a classification problem, as the size of training set approaches infinity. So, 1-NN classification is usually helpful to get a sense of what the optimal performance on a certain task could be.

2.2.3 Decision Trees

Decision trees [7] are a popular choice among machine learning practitioners, not really for their predictive power (which has been outperformed by state-of-the-art models over the years), but mainly because of their simplicity of understanding and use, and how they allow an easy visualization of the prediction process as a set of binary rules that are easily interpretable.

The hypothesis space explored by a Decision Tree algorithm is the set of trees obtainable by splitting the features on the basis of some threshold, for every possible value of such threshold. If all the features are boolean (True/False values), then it is a finite space of 2^{2^d} trees, where d is the dimension of the feature vector. Even if the hypothesis space can sometimes be finite, its enumeration is computationally infeasible even for a small d: as such, the exploration is usually carried out using greedy heuristics.

Prediction rule

Decision tree algorithms recursively split the training data according to the values of one of their features at a time. Each node of the tree represents a test performed on a particular feature: training examples are assigned to the node children based on the outcome of the test. Usually truthfulness is tested for boolean values, while continuous values are tested on whether or not their value exceeds some threshold.

The leaves of the tree implement a particular *decision function*, that returns the label of an example based on two possible scenarios:

- if the leaf contains only examples of one class, that class is predicted;
- if the leaf contains examples from both classes, the most frequent class is predicted.

Class predictions are simply shortest length paths from the root to the leaves of the tree. A test example is sent to one children per level by testing its features; once arrived at a leaf, it is labeled by the corresponding decision function.

Leaves naturally offer an estimation of the conditional distribution $F(y|\mathbf{x})$, based on the relative class frequencies with respect to the total number of examples assigned.

Training Algorithm

The most sensible part in the Decision Tree learning process is how to select the best possible split. We remind that the selection is local, since the algorithm is greedy; it utilizes only the information about the current node to make the choice. Thus,

even if the split is indeed optimal for a given node, it is not necessary the best one overall.

In general, one would wish that the split generates "pure" children, that is nodes containing only examples belonging to one class: this way, the classification would be unambiguous. As such, comparison criteria between splits are often called *impurity measures*. Thus, the loss function minimized by Decision Tree models at each node is its impurity. One usual impurity measure for Decision Trees is *entropy*, that is defined (for the binary case) as:

$$\text{ENTROPY}(S) = (-p_p \log_2(p_p) + (-p_n \log_2(p_n))$$

where S is the set of examples assigned to a children after the split, and ps are the empirical frequencies of respectively positive ($_p$) and negative ($_n$) examples in S. Note that $\text{ENTROPY}(S) = 0$ if only one class is present in the node, and $\text{ENTROPY}(S) = 0.5$ whenever the number of positive and negative examples is equal: thus, among the possible splits, the one with the lowest entropy will be selected at each node. Another popular split criterion uses the *Gini index* instead of entropy, which for a binary classification problem is defined as:

$$\text{GINI}(S) = S_0 S_1,$$

where S_0 constitutes the fraction of examples in S that belong to class 0, and S_1 is the fraction of examples in S that belong to class 1. Note again that $\text{GINI}(S) = 0.25$ whenever the classes are perfectly balanced, and conversely $\text{GINI}(S) = 0$ whenever S contains only examples belonging to one class.

Regularization

A common way to achieve regularization in Decision Trees is to make a node become a leaf earlier than expected (usually, a node is considered a leaf if it contains a single sample): a common strategy is *pruning*, i. e. discard nodes that violate a certain constraint, to prevent the tree from adapting to the noise in the training data. Among pruning criteria, the most used are:

- stop the generation of children nodes whenever the number of samples in the node is less than a fixed threshold value (this is known as *top-down pruning*);
- generate the whole tree, and then eliminate all the nodes whose depth from the root is above a certain threshold (known as *bottom-up pruning*).

2.2.4 Ensemble methods

The term *ensembling* in machine learning refers to combining multiple models together to improve generalization, such that the performance of the ensemble is better than the one yielded by each model alone. The intuition behind ensembling is that a pool of simple models can be better than a unique complex model (which has strong variance and might end up overfitting the data).

The prediction phase is usually handled by obtaining a prediction for an example by each component model, eventually assigning its label by majority or weighted voting. More in general, the key points of ensemble learning are:

- control the bias of each component. This is accomplished by using both trivial or heavily regularized base components;
- promote diversity among the components. To understand why this is important, consider three hypothesis, $\{h_1, h_2, h_3\}$. If their output on some feature vector $\mathbf{x}$ is identical, when $h_1(\mathbf{x})$ is wrong, $h_2(\mathbf{x})$ and $h_3(\mathbf{x})$ will also be wrong. However, if their output is uncorrelated, it might be the case that $h_1(\mathbf{x})$ is wrong, but $h_2(\mathbf{x})$ and $h_3(\mathbf{x})$ are right. Thus, majority voting would correctly classify that vector.

Machine learning literature lists an extensive number of different ensembling strategies. In particular, we will focus on:

- *bagging*, which stands for *bootstrap aggregation*, an approach that aims to decrease the variance of the component models by decoupling them from the original dataset;
- *boosting*, which trains a set of so-called *weak learners*, that perform poorly on the data, and then aims to increase the overall performance by combining them in some way. Usually this is achieved by making subsequent models in the combination focus on examples that the previous learner misclassified.

In this work, we used ensembles of Decision Trees - that is, ensemble models where the basic component is a Decision Tree. Next, we will look at two popular implementations: Random Forests and Gradient Boosting Machines.

Random Forests

Random Forests [6] are an implementation of the bagging principle. Given the dataset D, at each step n examples are sampled with replacement to form a new dataset $\hat{D}$, of the same size as D. A Decision Tree is fitted to the new dataset and the whole process is repeated for every tree in the forest. By giving each component a fresh dataset at each iteration, the variance of the overall predictor is decreased without increasing its bias. Final predictions for a new example are then cast by

outputting the most frequently predicted class by the trees in the forest (majority voting).

In addition to this "pure" bagging approach, Random Forests add an additional randomization step: each tree in the forest performs classification using only a random subset of the features, i.e., if $\mathbf{x} \in \mathbb{R}^d$, then $0 < d' \leq d$ features are selected. This determines that on average, features that are strong predictors are excluded from the training set half of times, and that leads to a decrease of the correlation between the outputs of different trees in the forest.

Gradient boosting machines

Gradient boosting machines [17] combine a series of weak learners to obtain a stronger predictor. The way they work is by initially training a very simple model (the *base learner*) for the data. For example, this might be a model that naively predicts just one class out of the two. Then, another model h_J is trained to minimize the *residual error function* $J(y, h(\mathbf{x})) = y - h(\mathbf{x})$ of the base learner, where $h(\mathbf{x})$ is the output of the model. J is also called *binary deviance*. Note that this formulation leaves the choice of $h(\mathbf{x})$ unspecified, though in practice common choices are Logistic Regression or Decision Trees.

Gradient boosting is basically a technique to create a series of subsequent models, such that latter models learn how to correct the errors that the former ones made. These models are then combined in a linear fashion

$$h_{i+1} = h_i + \alpha_i h_{J_i},$$

where h_{i+1} is the model at iteration $i+1$, h_{J_i} is the model trained with the residual errors of h_i, and α_i is the *step size* that specifies how much the residual error model accounts in the combination. This procedure can be thought of as a gradient ascent optimization of h using h_{J_i} as the error gradient. After m steps, a final model h_m is obtained and predictions for a new example can then be cast by feeding it to all the m models, weigh their prediction with their corresponding α and then taking the average.

$$h(\mathbf{x}) = \frac{1}{m} \sum_{i=1}^{m} \alpha_i \, h_i(\mathbf{x}).$$

Regularization

Regularization in ensemble methods is for the most part achieved implicitly by construction, since the ensemble components have low-variance (are heavily regularized). To this extent, Random Forests often offer the possibility to control the regularization of each tree in the forest with pruning. Gradient boosting machines

use the parameter α to control the influence of single component in the final model. Both methods allow to further tune the complexity of the final model by changing the number of components they utilize.

2.2.5 Support Vector Machines

Support Vector Machines (SVMs) [5] are a learning algorithm that implements the concept of *maximum margin classifier*. It originated in the context of statistical learning theory [56]. As of today, it is one of the most widely used learning algorithms together with Neural Networks, because it provides good generalization and, with a simple "trick", allows its use even in case of non-linear decision boundaries. Although SVMs were a learning algorithm originally not thought to be used as a probabilistic classifier, a technique has been developed to constrain its predictions to a probabilistic output [42].

Training algorithm

The idea behind SVMs is that points that are distant from the decision boundary should have their class predicted with more "confidence" than points that lie nearby, because, in the latter case, a small change in the decision boundary would cause a change in prediction. Thus, finding the margin with the largest distance from points of the two classes would result in more confident predictions. For notational convenience, in this section we will assume $Y \stackrel{\text{def}}{=} \{-1, 1\}$, and denote the bias with the letter b. Furthermore, we will assume that the data is linearly separable. If that is the case, then we can choose two parallel hyperplanes such that the distance between the two closest points belonging to different classes is the largest possible. We call the bounding region inside the two hyperplanes *margin*, the hyperplane that lies halfway between the two "supporting" hyperplanes *optimal margin*, and the data points with which the two hyperplanes are constructed *support vectors*. The distance between these two hyperplanes is geometrically equal to $\frac{2}{\|\theta\|}$. We can thus formulate the optimization objective for SVMs as follows:

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\|\theta\| \\ &\text{subject to} && y_i(\theta^T \mathbf{x}_i - b) \geq 1 \quad \forall\{\mathbf{x}_i, y_i\},\, i = 1, ..., n, \end{aligned}$$

where we reversed $\frac{2}{\|\theta\|}$ to change the type of problem from maximization to minimization and added the additional constraint to prevent points belonging to one class to lie inside the margin (indeed, the constraint is a mathematical convenience to express the following conditions: $\theta^T \mathbf{x}_i - b \geq 1$ if $y_i = 1$ and $\theta^T \mathbf{x}_i - b \leq 1$ if $y_i = -1$).

This formulation can be solved with linear programming libraries to obtain the values of θ and b that characterize the margin (most libraries actually do not solve this problem directly, but its dual). Once the values of the weights and the bias are obtained, new examples can be classified with:

$$h(\mathbf{x}) = sign(\theta^T \mathbf{x} - b).$$

Hinge loss and regularization

The formulation of the optimization problem of SVMs is valid only if the data is linearly separable, otherwise the two constraints are not satisfiable. When this is the case, it is said that the SVM implements a *hard-margin*. To extend SVMs to handle the case where data is not linearly separable, the optimization objective needs to be slightly modified with the introduction of the so-called *hinge loss* function, defined as:

$$hinge(\mathbf{x}_i, y_i) = \max(0,\, 1 - y_i(\theta^T \mathbf{x}_i - b)), \quad i = 1, ..., n,$$

which assumes the value of 0 if the margin constraint is satisfied, otherwise has a value proportional to the distance of the point from the margin. This results in a new formulation of the objective function, which is implemented with a vector of so-called *slack variables* ξ[3]:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\theta\| + C\sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y_i(\theta^T \mathbf{x}_i - b) \geq 1 - \xi_i \quad \forall\{\mathbf{x}_i, y_i\},\, i = 1, ..., n, \end{aligned}$$

This is also known as the *soft-margin* SVM. This formulation allows some points to lie outside the margin, at the cost of a penalty $C\xi_i$ per point in the minimization. The parameter C reflects how regularization is added to a SVM. It is a constant which trades off the size of the margin against how many errors the classifier is allowed to commit. Large values of C decrease the size of the margin, increasing the quality of the fitting at the risk of overfitting, while smaller values allow some points to be misclassified, decreasing overfitting and increasing generalization.

Kernel trick

Another limitation with these formulations of SVM, both hard and soft margin, is that the decision boundary they implement is strictly linear. To overcome this issue, SVMs provide an easy and efficient way to map the original feature space to some

[3]Slack variables are basically a clever way to implement the hinge loss, since it is not a differentiable function, and does not allow to solve the dual problem with the use of Lagrangian multipliers.

higher-dimensional feature space where the training set is separable. This procedure is known as *kernel trick*, and it is what allows SVMs to retain their predictive power even in complex classification problems.

Informally, a *kernel function* K is a function that corresponds to a dot product in some unspecified feature space, that is:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j),$$

where ϕ is a function that maps points from feature-space to points in the above mentioned extended feature space. Since going through the whole explanation of kernels is outside the scope of this study (for an extended survey, see [51]), we will limit ourselves to say that kernel functions can be precomputed using the training data and used in the dual formulation of the SVM optimization problem to achieve the search for a maximum margin hyperplane in a new feature space. One common choice for kernel functions is the so-called *Radial Basis Function* (or RBF kernel), defined as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{\frac{-(\mathbf{x}_i - \mathbf{x}_j)^2}{2\sigma^2}}.$$

2.2.6 Artificial Neural Networks

Artificial Neural Networks [22] are a machine learning model loosely inspired by the way our brain learns. The key idea is that neurons can be thought of as computational units that acquire signal from other neurons through connections and propagate it if the signal strength exceeds a certain threshold (a pioneering study in this sense is [46]). Over the years, Neural Networks have emerged as the algorithm of choice for many classification problems, mostly due to their high predictive power and ability to represent complex, non-linear decision boundaries.

Prediction rule

Based on the biological inspiration we described above, a single neuron is a unit that computes a certain function of an input vector. That is, takes as input a linear combination of the input features, weighted by their *connection* to the unit, and produces an hypothesis $g(x)$, were g is a non-linear mapping of the inputs, called *activation function*. For example, g can be the sigmoid function; in this case, a single neuron basically performs Logistic Regression. Other common activation functions are:

- the *hyperbolic tangent function*

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

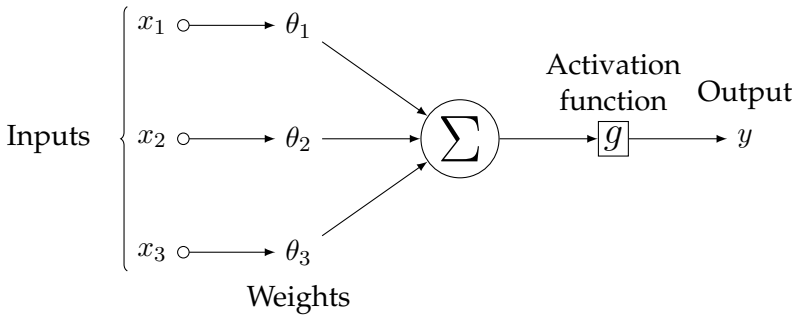


Figure 2.1: The structure of a Neural Network unit.

which has basically the same properties as the sigmoid function but is defined in the real interval $(-1, 1)$;

- the *rectified linear unit* functions, which is a family of functions that compute exact or approximated forms of $max(0, x)$.

Fig. **2.1** shows a graphical representation of a neuron.

An example of the structure of a Neural Network is shown in Fig. **2.2**. The idea is to attach the input vector to a *layer* of neuron units, and use their output as the input for a subsequent layer. This process can be iterated arbitrarily many times until an *output layer* is reached. The output layer is responsible for providing the prediction for the given input. The layers created between the input layer and the output layer are usually called *hidden*. The number of layers and the number of neurons that each of them contains constitute the *architecture* of the network.

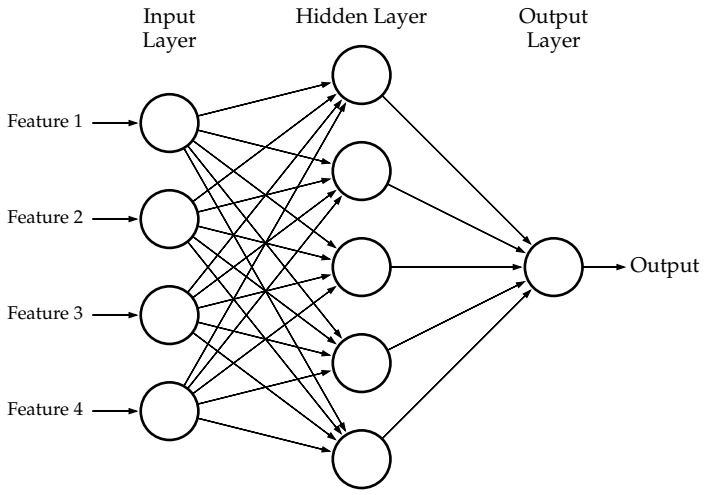


Figure 2.2: A Neural Network consisting of a 4-units input layer, 5-units hidden layer and a single-unit output layer.

This formulation expresses a very complex, non-linear hypothesis space. To give an example, the space spanned by a Neural Network for inputs of size d, one hidden layer and a single-neuron output layer is:

$$h_\theta(\mathbf{x}) = g_k\left(\sum_{j=1}^{h}\theta_{kj}\, g_h\left(\sum_{i=1}^{d}\theta_{ji}\mathbf{x}_i\right)\right),$$

where the subscripts are used to index a particular weight between two units. Although theoretically each activation function can be different from unit to unit, in practice one tends to assign a given activation function for all the units in a layer.

The equation above provides the corresponding prediction rule for Neural Networks. When the network is trained and its final weights θ are computed, new inputs $\mathbf{x}$ are simply fed to the network: the prediction is then the value calculated at the output layer. The prediction phase is known as *forward propagation*, in contrast with *backpropagation* that is the procedure that learns the proper values of the weights. For binary classification, how the output is calculated depends on the architecture and the activation function of the output layer. Specifically, two common choices for binary classification with probabilistic outputs are:

- a single-neuron output layer which computes the sigmoid function; the output can then be treated as a probability or cast into an outcome as seen for the Logistic Regression algorithm;
- a two-neuron output layer where each neuron computes a piece of the *softmax function*

  $$g_k(x)_i = \frac{e^{x_i}}{\sum_{j\in\{0,1\}} e^{x_j}}, \quad i \in 0, 1.$$

  The softmax function basically transforms its input into a posterior probability.

Since these two approaches are basically different ways to compute the same conditional probability $\mathbb{P}(y|\mathbf{x})$, both produce comparable results. In this work, the single-neuron architecture for the output layer was chosen.

Training Algorithm

Learning the weights of a Neural Network is not trivial, because of the presence of the hidden layer. Usually computing the values of the weights involves calculating the gradient of the loss function[4], which is then utilized to perform the minimization. For the hidden layers, the concept of loss function is less clear, since they really are non-linear representations of the layer below them and nothing more. Thus, a

[4]For Neural Networks with sigmoid activation functions, this is exactly the log-likelihood of the data given the parameters, same as the Logistic Regression.

way is needed to calculate how the output of a hidden layer influences the final output of the network.

This problem was solved by the famous backpropagation algorithm, that was developed in the early eighties [50]. It basically consists of a four-step procedure:

- initialize the weights with random values;
- perform forward propagation to calculate predictions for the training data;
- use those predictions to calculate the gradient of the loss function at the output layer;
- propagate backwards the derivative of the activation function to calculate the loss gradient in the layer below.

The gradient is then used to update the weights according to the rule:

$$\theta(t+1) = \theta(t) + \Delta\,\theta(t),$$

where t indicates a time step or iteration, and Δ is an expression that includes the gradient of the loss function.

Backpropagation is implemented in different flavors, all with their pros and cons. Among the most common implementations, we list:

- *stochastic* backpropagation, where the update is staged to occur every time a single example from the dataset is presented to the network;
- *mini-batch* backpropagation, where the update is staged to occur once the gradients of m examples in the dataset are calculated (where $1 < m \ll N$, and N as usually denotes the training set size);
- *full-batch* (sometimes just *batch*) backpropagation, where the update is staged to occur once the gradients for the entire training set are calculated.

A full forward propagation pass on the training set is called *epoch*: backpropagation updates the weights for a certain number of epochs until the loss function is minimized or cannot be decreased further.

Mathematically, the gradient of the loss function for a given layer is calculated through multiple applications of the chain rule for function composition. We will not step into the details, but provide the final formula for the gradient[5], which can be summarized as

$$\frac{\partial L}{\partial \theta_{ji}} = \delta_i o_j,$$

[5] In this and the following formulas, j indicates the layer above a given layer i, while the notation l_i indicates the number of neurons in the generic layer indexed by i.

where:

$$\delta_i = \begin{cases} (o_i - y_i)\ g_i\prime, & \text{if } i \text{ is an output neuron;} \\ \sum_{j=1}^{l_j} \delta_j \theta_{ji}\ g_i\prime, & \text{if } i \text{ is an inner neuron.} \end{cases}$$

This formula makes clear that for backpropagation to succeed, the activation function must be differentiable (although in practice several tricks have been proposed to allow the use of non-differentiable activation functions, like the rectifiers) and that all the error derivatives of the layer above must be calculated beforehand (for the output layer, this corresponds to the derivative of the loss function). Since we now know how to calculate the gradient of the loss function, we can expand the $\Delta\theta$ part of the update formula as:

$$\Delta\theta_{ji} = -\eta \frac{\partial L}{\partial \theta_{ji}} = \begin{cases} -\alpha\ o_j (o_i - y_i)\ g_i\prime, & \text{if } i \text{ is an output neuron;} \\ -\alpha\ o_j \sum_{j=1}^{l_j} \delta_j \theta_{ji}\ g_i\prime, & \text{if } i \text{ is an inner neuron.} \end{cases}$$

The parameter η is known as *learning rate*, and determines the relative size of the update. We will discuss later how one can choose "good" values for the learning rate and the other parameters (such as the architecture) that a Neural Networks implicitly defines.

Regularization

One way to achieve regularization in Neural Networks uses the L2 penalty we described for Logistic Regression, which in the context of Neural Networks is often referred to as *weight decay*, to emphasize its role in shrinking the value of the weights.

However, over the last years, a new technique as emerged to regularize Neural Networks, known as *dropout*. The working principle of dropout is simple: when forward propagation is performed in the training phase, the output of each neuron is allowed to be zeroed with probability p. Hence, that neuron would be excluded from that particular training iteration. Since the number of effectively active neurons for a given training pass becomes random, dropout has been shown to construct the *average model* (a form of what Gradient Boosting Machines do) between a large number of different Neural Networks. This means that the variance of the whole network can be easily controlled to prevent overfitting by changing the value of p. The dropout technique is best described in [54].

2.3 Performance metrics for binary classification

After a model is trained, one needs to assess how good are its predictions. This is in general a different problem than learning, since the loss that is minimized during the learning phase might not be the metric one is interested in to measure the goodness of the classifier. To this purpose, several *performance metrics* are used in machine learning. For the moment, we can loosely define a performance metric as a function $P(h, D)$ that takes a particular hypothesis and a dataset and assigns a real-valued "grade" to that hypothesis on that dataset. There is no best evaluation performance overall: each one serves its own purpose and it is good at explaining a certain aspect of the model behavior. We remind that since these metrics are calculated from a sample of a much larger population, we are performing *estimations* of the true performance. We will see the implications of this in the next sections. All the performance metrics discussed in this section and many other are analyzed with greater detail in [53].

2.3.1 Accuracy and the class imbalance problem

A classic measure of performance for binary classification problems is the *accuracy*, which is the proportion of examples that were correctly classified by the model among the total number of examples in a generic dataset D of size N:

$$\text{ACCURACY}(h, D) = \frac{correct}{total} = \frac{TP(D) + TN(D)}{TP(D) + TN(D) + FP(D) + FN(D)}.$$

where:

- $TP(D) = \sum_{i=1}^{N} tp(\mathbf{x}_i, y_i),\ \{\mathbf{x}_i, y_i\} \in D$, with tp being defined as

$$tp(x, y) = \begin{cases} 1 & \text{if } h(x) = 1 \text{ and } y = 1 \\ 0 & \text{otherwise,} \end{cases}$$

  which stands for True Positives, positive instances that were correctly classified as positive by the model;

- $TN(D) = \sum_{i=1}^{N} tn(\mathbf{x}_i, y_i),\ \{\mathbf{x}_i, y_i\} \in D$, with tn being defined as

$$tn(x, y) = \begin{cases} 1 & \text{if } h(x) = 0 \text{ and } y = 0 \\ 0 & \text{otherwise,} \end{cases}$$

  which stands for True Negatives, negative instances that were correctly classified as negative by the model;

- $FP(D) = \sum_{i=1}^{N} fp(\mathbf{x}_i, y_i),\ \{\mathbf{x}_i, y_i\} \in D$, with fp being defined as

$$fp(x, y) = \begin{cases} 1 & \text{if } h(x) = 1 \text{ and } y = 0 \\ 0 & \text{otherwise,} \end{cases}$$

  which stands for False Positives, instances that the model classified as positive but were in fact negative;

- $FN(D) = \sum_{i=1}^{N} fn(\mathbf{x}_i, y_i),\ \{\mathbf{x}_i, y_i\} \in D$, with fn being defined as

$$fn(x, y) = \begin{cases} 1 & \text{if } h(x) = 0 \text{ and } y = 1 \\ 0 & \text{otherwise,} \end{cases}$$

  which stands for False Negatives, instances that the model classified as negative but were in fact positive.

In cases where the proportion of examples is biased towards one of the two classes, accuracy by itself does not tell much about the overall performance of a model. As an example, consider the problem of predicting whether a patient has a rare disease or not given its medical condition: it is clear that the large majority of the examples would be labeled as negative and only a small fraction would fall in the positive class. Say, for example, that only 1% of training cases are labeled as positive. Such situations are easily detectable by computing the *null accuracy* of a classifier, that is the performance of a "dumb" model that predicts only the most frequent class. If the null accuracy is largely different from 50%, accuracy becomes not the best performance metric one would evaluate.

This issue is known as the *class imbalance problem*. Some common solutions to class imbalance are:

- choose a different performance evaluation than accuracy, or at least accompany accuracy with other measures that take into account class imbalance;
- rebalance the proportion of the classes in the dataset with random sampling with replacement from the dataset. We distinguish *oversampling*, that increases the proportion of the minority class, from *undersampling*, where the proportion of the majority class is decreased;
- use *cost-sensitive* techniques. These approaches are based on the intuition that not all the misclassified examples have the same cost. For example, think of a setting where a model is trying to predict the malfunctioning of a device given features regarding its production process. It is clear that the cost of making a false negative (that is, devices that are faulty but are considered functioning) is greater than the cost of a false positive (where a devices is flagged as faulty but is in fact functioning).

For a discussion about cost-sensitive learning, we refer to [31].

2.3.2 Imbalance-aware metrics

Since a great number of binary classification problems in imbalanced, there is a wide variety of performance metrics that are independent from the particular proportion between the two classes. To illustrate them, one useful tool is the *confusion matrix*, a contingency table that compares the predictions of the model against the ground truth (the true labels)[6]. Table **2.1** shows one example of confusion matrix.

		Actual		
		Positive	Negative	Total
Predicted	Positive	TP	FP	$TP + FP$
	Negative	FN	TN	$FN + TN$
	Total	$TP + FN$	$FP + TN$	M

Table 2.1: An example of confusion matrix.

From the confusion matrix, a series of useful metrics can be derived:

- *precision* measures the proportion of true positives in the total of positives predicted by the classifier. It can be derived from the first row of the confusion matrix:
$$\text{PRECISION}(h, D) = \frac{TP}{TP + FP};$$
- *recall*, also known as *true positive rate* or *sensitivity*, measures the proportion of positive examples that were correctly classified with respect to the actual number of positive examples in the data. It can be derived from the first column of the confusion matrix:
$$\text{RECALL}(h, D) = \text{TPR}(h, D) = \frac{TP}{TP + FN};$$
- *specificity*, also known as *true negative rate*, measures the proportion of negative examples that were correctly classified with respect to the actual number of negative examples in the data. It can be derived from the second column of the confusion matrix:
$$\text{SPECIFICITY}(h, D) = \frac{TN}{TN + FP}.$$

Precision and recall are usually combined into a single measure that is robust to class imbalance, known as the *F1 measure*, which basically consists in the harmonic

[6] From now on, for notation simplicity, we will drop the dependence from D on the functions TP, TN, FP, FN, considering it implicitly assumed.

mean of the two metrics:

$$\mathrm{F1}(h, D) = \frac{2 \times \mathrm{PRECISION}(h, D) \times \mathrm{RECALL}(h, D)}{\mathrm{PRECISION}(h, D) + \mathrm{RECALL}(h, D)}.$$

2.3.3 Metrics for probabilistic classifiers

In cases where the output of a classifier is a probability, the evaluation metric of choice is usually the *Receiver Operating Characteristics curve*, shortened ROC. A ROC curve depicts how the true positive rate $\mathrm{TPR}(h, D)$ and the *false positive rate* $\mathrm{FPR}(h, D) = 1 - \mathrm{TPR}(h, D)$ vary as one moves a threshold parameter α. It is drawn as a curve plot where the x-axis represents the false positive rate, and the y-axis represents the true positive rate, both with values between 0 and 1. A point of the curve has thus coordinates $(\mathrm{FPR}(h, D, \alpha), \mathrm{TPR}(h, D, \alpha)$, for a fixed value of α, with point $(0; 0)$ being associated with $\alpha = 1$ and point $(1; 1)$ being associated with $\alpha = 0$. The point $(0; 1)$ is the optimum of the curve, since the false positive rate is minimized and the true positive rate is maximized. An example of ROC curve is shown in Fig. **2.3**.

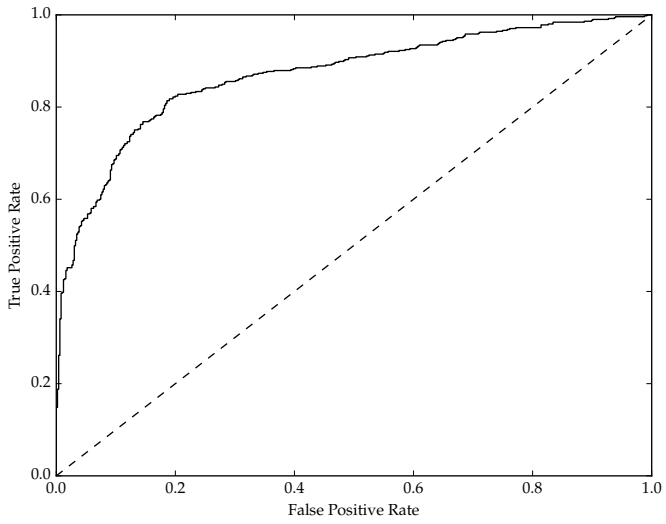


Figure 2.3: An example of ROC curve. The dashed line is the ROC curve of a random classifier.

To illustrate how the curve behaves, it is useful to consider its three extremes:

1. for a perfect classifier, which makes no errors, the curve goes from $(0; 0)$ to $(1; 0)$ to $(1; 1)$;
2. for a totally random classifier, which whatever the threshold always produces the same number of false positives and true positives, the curve created is the line segment bounded by the points $(0; 0)$ and $(1; 1)$;

3. no reasonable classifier is assumed to produce a ROC curve whose points are located under the $(0;0)$, $(1;1)$ segment.

ROC curves are usually expressed numerically by computing the *Area Under the ROC curve*, or *AUC-ROC* - that is, the integral of the area under the ROC curve. Thus, an AUC-ROC value of 0.5 describes a totally random classifier (since it is equal to the area under the segment described in case 2), while an AUC-ROC of 1 describes a perfect classifier. In general, AUC-ROC scores higher than 0.8 define a classifiers with good predictive ability.

Besides being a metric not affected by class disproportions, the AUC-ROC also offers a nice probabilistic interpretation. In fact, the AUC-ROC score of a classifier is equal to $\mathbb{P}(score(\mathbf{x}_{pos}) > score(\mathbf{x}_{neg})$, the probability that it will rank a randomly chosen positive example higher than a randomly chosen negative example. This is particularly useful in medicine for example, where $score$ could be the result of a clinical test. For an extensive survey of ROC curves, see for example [15].

2.4 Model evaluation

One thing that can go wrong when evaluating the performance of a classifier is that the estimation can be overly optimistic or *biased* (very different from the true out-of-sample performance). That is the case, for example, if we measure the performance of a classifier on the same data used to fit the training set: if overfitting occurs, the model would probably obtain a good in-sample performance, while performing bad out-of-sample. In this section, we will describe two different methods that are used to ensure that the evaluation of a model is as unbiased as possible.

2.4.1 The hold-out validation estimator

A first approach towards unbiasedness is to *hold-out* some part of the data from training. Such reserved data is called *validation set*, to emphasize the fact that is used for evaluation purposes. Since the validation set does not enter the training process, we are guaranteed that the evaluation takes place on a fresh dataset, giving a less biased estimation of the out-of-sample performance of the model. In addition, overfitting can be easily detected by observing if the performance in validation decreases as the training performance increases. However, hold-out validation is still a biased estimator, because we could obtain a good performance on a particular statistically favorable validation set. This suggests that to decrease the bias, we have to repeat the hold-out process multiple times on different datasets.

2.4.2 The cross-validation estimator

The *cross-validation* approach aims to mitigate the estimation bias of the hold-out technique. It is based on the use of *resampling* to create multiple different versions of the dataset, and then combine their result to obtain the final estimation.

The most used variant of cross-validation is *K-fold cross-validation*, where the dataset is split into K disjoint partitions of almost equal size, termed *folds*; for each $k = 1, ..., K$ we train the model on D/D_k and evaluate on D_k, thus obtaining K different performance scores using K different training sets (although with a bit of overlap) and K different validation sets. The final estimation of the out-of-sample performance is then computed as the mean of the K different performance scores:

$$CV_K(h_\theta) = \frac{1}{K} \sum_{i=k}^{K} P(h_\theta^{\hat{k}}, D_k),$$

where the classifier $h_\theta^{\hat{k}}$ is the output of the learning algorithm $\mathcal{A}(L, D/D_k)$ and P is the general formulation of a performance score, as defined in Section **2.3**. Since it employs different datasets at each iteration, K-fold cross-validation concretely decreases the bias of the hold-out approach, giving a more realistic estimation.

Cross-validation is especially useful when data is scarce, since its approach allows to use large percentages of the data for model training, and, at the same time, the whole data for model evaluation (although in different stages). In common machine learning practice, it is advised on empirical basis [21] to use 5-fold or 10-fold CV, since they provide a good trade-off between quality of the estimation and computational cost. Many other cross-validation strategies are described in detail in [2].

2.5 Model Selection

Constructing a model does not usually involve just the calculation of the optimal values of the weights. There are also a number of algorithm-dependent parameters to set properly: examples are the number of trees composing a Random Forests, or the learning rate and the number of neurons for a Neural Network. These additional parameters are referred to as *hyper-parameters*, to distinguish them from the usual free parameters of a model. Discovering good hyper-parameters is critical, since their values can either boost or dump the performances of a classifier. Thus, a proper machine learning system must include, besides the usual learning phase, an additional procedure to tune the hyper-parameters in order to maximize the performance. At a higher level,this problem is tackled using two nested loops:

- the inner loop is used to minimize the loss function of one model. To avoid bias in the estimation, usually its generalization error is estimated with the use of cross-validation;
- the outer loop is used to optimize a performance criterion (which is usually different from the loss and measures the predictive ability of the model, such as accuracy or ROC-AUC) by exploring different values of its hyper-parameters.

This process produces a set of scores for different models instantiated with different configurations of their hyper-parameters, from which the classifier with the best score can be eventually selected. For this reason, this technique is referred to with the term *model selection*.

In general, this *hyper-parameter optimization* is not trivial, because the space of hyper-parameters is a mathematically difficult object to explore. Indeed, it can include continuous (i. e. regularization coefficients or learning rates), discrete (i. e. the number of trees in a Random Forest) or even mutually dependent (i. e. we can associate different activation functions to different layers in a Neural Network) components. As a result of this complexity, the optimization function associated to this space is not differentiable, hence it cannot be solved with common techniques such as gradient descent.

Even though many clever algorithms have been developed recently to address this issue (see [4]), the main approaches in practice are oriented to exploring the hyper-parameters space through brute-forcing (*grid-search*) or random-walk (*randomized search*), mainly because they can be trivially parallelized and can exploit the power of multi-core machines. The latter approach, described in [3], is the one that was used in this work.

2.5.1 Double cross-validation

Machine learning practitioners sometimes misuse cross-validation for model selection, obtaining estimations that do not reflect the real out-of-sample performance of a model. Two common mistakes that are committed are:

- select supposedly relevant features on the model that optimized the model selection criteria, instead of performing the selection inside each cross-validation fold (discussed in [20]);
- selecting the final model based only on the model selection performance, without evaluating its out-of-sample performance separately.

The latter mistake could result in situations where the model selection criterion is such heavily optimized that the out-of-sample performance is affected negatively. This problem is discussed in [8] and referred to as "overfitting the model selection

procedure". To prevent such mistakes, the right approach is to set aside a portion of the data for evaluation purposes only, perform the model selection on the rest of the data, and finally evaluate the model on the reserved test set.

A more sophisticated approach is to employ cross-validation in both the model selection and evaluation steps. This is useful in a scenario where one is trying to compare the performances of different learning algorithms. For a fixed learning algorithm, in fact, it allows to get a sense on what the out-of-sample generalization performance of the set of models represented by different choices of the hyper-parameters might be. This procedure, known as *nested* or *double cross-validation*, can be described as follows:

- the data is split in K folds;
- a number of hyper-parameter configurations is chosen (either with grid search of random search);
- all these configurations are evaluated in a model selection step that uses the data of $K - 1$ folds and a J-fold cross-validation internally;
- the resulting best model is evaluated on the remaining fold;
- the process is repeated K times.

Since the K model selections are likely to result in models with different hyper-parameters (because each model selection is performed using different portions of the dataset), one can get a sense of which range of performances we should expect from a particular learning algorithm. Furthermore, overfitting in model selection can be detected by observing if the selected models estimations are over-optimistic in comparison to their respective evaluation scores. Fig. **2.4** shows a graphical representation of a typical double cross-validation algorithm workflow.

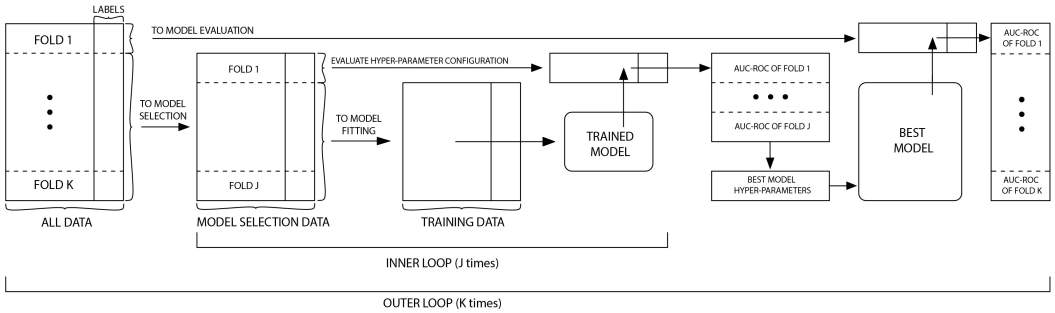


Figure 2.4: Workflow of a K-fold outer loop, J-fold inner loop double cross-validation algorithm to optimize the AUC-ROC performance criterion.

2.6 Model comparison

Our study concerns the test of multiple machine learning models. A natural question that arises in this sense is: how can we assess that the potential differences in performance between the various models we are going to construct are meaningful and not just caused by statistical variability in the samples? This is a problem that in machine learning is addressed by using *statistical hypothesis testing* [33]. The process is straightforward: to start off, one assumes that the difference between the two measured performances is not statistically relevant (i. e. its expected value is 0). We call this initial hypothesis the *null hypothesis*, indicated with H_0. Then, experiments are performed (in our case, a classifier is trained) and the outcomes observed (its performance is tested). If the results fall in a so-called *rejection region* of outcomes which represent evidence against the null hypothesis, it can be rejected in favor of the *alternative hypothesis*, indicated with H_1, which in our case is the fact that the difference between the two performances is significant (i. e. its expected value is not equal to 0). Note that if the results do not fall in the rejection region, we are not proving the null hypothesis: we can only claim that we do not have enough evidence against it. The probability that observations fall in the rejection region is the *significance level* of the test, and it is usually fixed in statistics to be 5% (although sometimes 1% or 10% significance level tests are performed). This is just a scratch in the surface of hypothesis testing, which serves for the purposes of our work (a thorough discussion would require a thesis by itself). More information, especially regarding the use of hypothesis testing in the context of classification, can be found for example in [29].

2.7 Unsupervised Learning

With the term unsupervised learning, we describe a broad category of problems that are tackled by processing unlabeled data. Among the most known we can list:

- *clustering*, where we look for natural groupings, or "clusters", in the data;
- *density estimation*, where we assume that data comes from a certain probability density function and we try to estimate its parameters;
- *dimensionality reduction*, where we project the data in a smaller dimension, in a way that the information it contains is preserved up to a desired amount;
- *anomalies* and *outliers detection*, where we look in the data for uncommon examples that do not conform to regular patterns.

In this work, we are going to direct our focus towards the clustering task.

We can use probability to describe unsupervised learning as we did before for supervised learning: in this case, since data is unlabeled, it has now the form $D = \{\mathbf{x}_i\}_{i=1}^N$. Once again, we can assume that there is an unknown probability distribution $F(\mathbf{x})$ from which our data set comes from, and we are required to estimate an unknown function f that helps characterize F (in density estimation, f is the distribution itself).

Note however that unsupervised learning is not as crisply defined as the supervised counterpart. This depends on the different notions of what a "structure" can be, which sometimes involve some degree of subjectivity (like in clustering for example, where the notion of "right cluster" may vary from person to person, or density estimation, where the whole task is biased by an a-priori assumption of what the final distribution will be).

The power of unsupervised methods is in their ability to "compress" the data: for example, with clustering we could represent a whole cloud of points by just the mean point of the cluster. This allows a much more simple interpretation, which might come in handy when one is trying to "make sense" of a great number of data points, or even trying to display graphically what the data "looks like".

2.7.1 Clustering

Clustering [35] is an unsupervised learning task in which a partition of the data is sought, such that observations that end up in a certain partition are similar between themselves, and dissimilar to observations in other partitions. From a machine learning point of view, it is convenient to treat clustering as a problem of *vector quantization*, because it is posed in terms of minimizing an error function like supervised problems.

Vector quantization tries to obtain an optimal encoding of a (possibly infinite) set of vectors $\mathbf{X} \subseteq \mathbb{R}^d$ using a finite set of K *prototypes* or *codebooks* $\mathbf{m}_i \in \mathbb{R}^d$, $i = 1, 2, ..., K$. The criterion that guides the choice of the prototypes is the minimization of the so-called *expected distortion error*, defined as

$$E(\mathbf{x}, \mathbf{m}_c) = \int \|\mathbf{x} - \mathbf{m}_c\|^2 \, p(\mathbf{x}) \, d\mathbf{x},$$

where c is the index of the "winning" prototype, the one in the set of prototypes that has the minimum Euclidean distance[7] from the considered example $\mathbf{x}_i$:

$$c = \arg\min_i \|\mathbf{x} - \mathbf{m}_i\|^2,$$

and $p(\mathbf{x})$ is the density function of $\mathbf{x}$.

[7] In this section, as well as the SOM part, we assume Euclidean distance to measure similarity. In practice, different measures of similarity can be used.

The discrete version of the error distortion function is:

$$E(\mathbf{x}, \mathbf{m}_c) = \sum_{i=1}^{N} \sum_{j=1}^{K} \|\mathbf{x} - \mathbf{m}_c\|^2 \, \delta_{winner}(i, j),$$

where δ_{winner} is defined as:

$$\delta_{winner}(i, j) = \begin{cases} 1 & \text{if } j \text{ is the index of the winning prototype for } \mathbf{x}_i, \\ 0 & \text{otherwise.} \end{cases}$$

From this general formulation, a wide variety of clustering algorithms can be derived; among them the famous *K-Means* clustering algorithm [34] and Self-Organizing Maps, the latter of which we will discuss later.

2.7.2 Self-Organizing Maps

Self-Organizing Maps (SOMs) [30] can be loosely described as a Neural Network based approach to solve the vector quantization problem for clustering. They are widely used in science due to their power and the outstanding visual capabilities. In particular, the main advantage with respect to classical methods such as the K-Means algorithm is that they are able to map high-dimensional input spaces to a low-dimensional grid (generally two dimensional), enabling the user to visualize the results of the clustering, which in turn allows easy interpretation and reasoning about the data.

Training Algorithm

The basic SOM algorithm begins by defining a two dimensional grid of $m = m_1 \times m_2$ prototype neurons. Each input pattern is connected with weights θ to every neuron in the grid, and the weights are initialized with random values. The objective of the SOM training is to provide a topology preserving mapping from the input space to the map units. Since the neurons are usually organized in a two-dimensional lattice, what it is obtained after the training is a mapping from the input space to a plane. The training phase of a SOM implements a form of *competitive learning*, which means that the neurons compete to "own" the input examples. The inputs are assigned to their Best Matching Unit (BMU), i. e. the closest unit according to its Euclidean distance. This phase is identical to the learning vector quantization algorithm.

After the competitive training, a *cooperative* phase starts, where the weights of the map receive an update weighed by their topological relation with the BMU: units receive a rate of update that is inversely proportional to their distance from the

BMU in the map. This process is repeated until a convergence criterion is met (usually whenever the decrease in quantization error becomes null or not significant). The cooperative phase literally moves units in the neighborhood of the BMU towards it: this principle is called *Hebbian learning* [23]. The corresponding update rule at iteration $t+1$ is defined as:

$$\theta(t+1) = \theta(t) + \eta(t)\,\kappa_{i,i^*}(t)\,\|\mathbf{x} - \theta(t)\|^2,$$

where:

- $\eta(t)$ is the learning rate, staged to decrease after every successive iteration t;
- i is the index of a generic neuron;
- i^* is the index of the BMU for the input $\mathbf{x}$;
- $\kappa_{i,i^*}(t)$ is the *neighborhood function*, a function that decreases monotonically with the distance from the BMU.

A popular choice for κ is a Gaussian function centered in the winning neuron:

$$\kappa_{i,i^*}(t) = \frac{\|\mathbf{r}_i - \mathbf{r}_{i^*}\|}{2\sigma^2}$$

where $\mathbf{r}_i$ and $\mathbf{r}_{i^*}$ indicate the topological distance of units i and i^* in the map. Another common choice is a stepwise function that assigns 1 to the winning neuron, a constant value $B < 1$ to its adjacent neighbors in the map, and 0 otherwise, known as the *zero-one* neighborhood function.

Visualization in SOMs

As we said before, the true power of the SOM is in its visualization capabilites. The common basis of each one of this visualizations is to represent the neurons in a hexagonal grid where every hexagon is a unit. After that, units are colored according to various properties, such as distances, values of the features of their prototypes, density, and many more. Some common SOM visualizations are:

- *hit maps* (Fig. **2.5**), which are a visualization of the grid where each hexagon is resized based on the number of input patterns that were assigned to it. With this visualization, high density regions are revealed, which might indicate clusters if bordered by regions of low density;
- *component planes* (Fig. **2.6**), where the grid is colored according to the value of a particular prototype feature (for example, large values receive a more light color and small values receive a darker one). Again, this visualization can help find zones where inputs with similar values for a given feature lie;

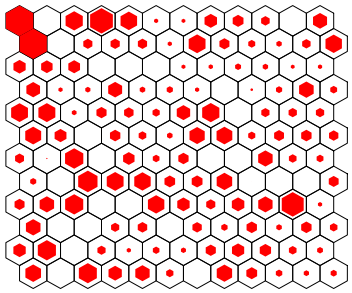


Figure 2.5: An example of SOM hit map.

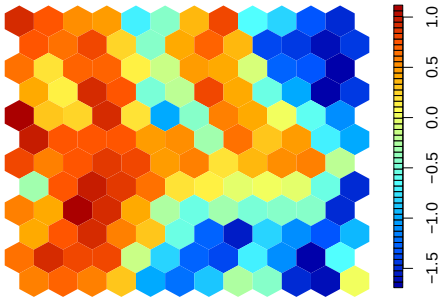


Figure 2.6: An example of a SOM component plane.

- *color coding* (Fig. **2.7**) of the map, obtained by assigning similar colors to units that have similar values of their prototypes.

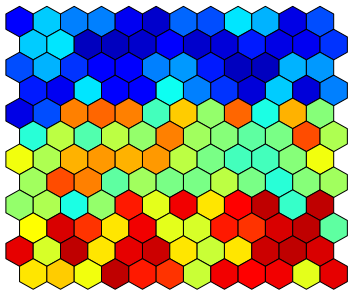


Figure 2.7: An example of SOM color coding.

There are many other visualizations the SOM can offer. For a complete review, as well as a technical explanation of how the ones shown in this thesis are implemented, see for example [57].

Chapter 3

Background and Methodology

This chapter is structured as follows: in the first part (consisting of the first three sections), we provide some background information, to pinpoint the research area where this study was conceived and provide a general description of the reference model that we compared our results against. The following sections serve to describe the data we were given, and illustrate the methodology that we utilized to set-up and carry on our experiments, listing in detail the approach we utilized to work on the problem.

3.1 Background

3.1.1 The Vermont Oxford Network

Our entire analysis is based on data collected from the Vermont Oxford Network, a nonprofit voluntary collaboration of health care professionals established in 1988. As of today, the Network is comprised of nearly 1000 NICUs around the world. The purpose of the VON is to improve the effectiveness and efficiency of medical care for newborn infants and their families through a coordinated program of research, education, and quality-improvement projects [24].

To help participants fulfill its mission in the particular context of neonatal care, the VON provides two main databases regarding low and very low birth-weight infants and infants meeting other eligibility requirements. In particular:

- the Very Low Birth-Weight (VLBW) Database, which contains data from infants whose birth weight spans from 401 to 1500 grams, or whose gestational age spans from 22 weeks 0 days to 29 weeks 6 days;
- the Expanded Database, which comprises the VLBW Database, plus data from infants who are admitted to a NICU at a member center or who died at any location in the center within 28 days of birth without first having gone home.

Members of VON can submit patient data by conforming to the instructions provided in a Manual of Operations [38], which is updated on a yearly basis and specifies eligibility criteria, data definitions and infant forms.

These databases are used extensively for randomized clinical trials, outcomes research[1], and health care quality improvement in general. Examples of studies based on these datasets can be found in [59] and [25].

3.1.2 Risk-adjustment for severity of illness

Our task of concern descends from the general problem of estimating the risk of death in hospital patients. Although medical research is still far from obtaining risk estimates that are sufficiently precise to guide therapeutic decisions, mortality predictions are used widely and with success to perform comparisons of outcomes across hospitals, for example to assess the quality of health care they provide. However, such comparisons need to take into account the severity of illness of patients treated in institutions. To understand why this is important, consider two hospitals, A and B, which have equal mortality rates, but hospital A treats relatively healthy patients, while B treats relatively sick patients. Without a proper measurement of how sick a patient is, a comparison study would wrongly infer that A and B provide approximately the same quality of health care, even though hospital B is probably better than hospital A. The process of statistically accounting for differences in illness criticality that influence health care outcomes is called *risk adjustment*.

In neonatology, the adjustment of risk for severity of illness is of particular interest especially in populations of very low birth-weight infants - i. e. neonates whose birth-weight is below 1500 grams, which constitute the clinical group where most deaths occur and where most advancements in clinical practices were made [43]. Indeed, severity of illness in the first hours of life is considered to have a prominent impact in the survival of low and very low birth-weight infants, along with more clinically obvious factors such as gestational age and birth-weight [18].

In this sense, several scoring schemes based on probabilistic outputs of logistic regression models have been developed by researchers, with the intent of adjusting mortality rates to account for the different illness severity levels of infants treated in NICUs. Most of them employ among their predictors birth-weight and Apgar

[1] Outcomes research seeks to understand the relationship between clinical services and practices and their end results, in order to provide scientific evidence which can help health carers make better decisions.

scores[2]. Much recently, more sophisticated scores were created, which use measurements of physiological variables in the first 12 or 24 hours of life of the infants. A wide range of such scoring methods is reviewed in [40].

Given that clinical practices in the treatment of low and very low birth-weight infants are constantly evolving in hope to reach better survival rates, it is crucial that state-of-the-art predictive models are constantly developed to adapt to such changes. At the same time, better performances of such models increase the unbiasedness of the comparisons between NICUs, resulting in turn in better decisions taken to improve the quality of health care in hospitals.

3.1.3 The VON-RA model

Among VON centers, inter-institutional adjusted mortality rates are estimated using a multivariate logistic regression classifier known as the *VON-Risk Adjusted* model. Such estimations are then communicated in the form of periodic reports that the VON provides to its member centers, with the intent of highlighting areas of concern in hospitals, where improvement in health care is possible or needed. The VON-RA model is retrained every year to adjust for changes in clinical practices and health care in general, and its validity is subjected to periodical revisions [60].

To guarantee that predictions are as fair as possible and not biased by differences in patients illnesses or clinical practices, the predictors used by the VON-RA are based on factors that occur exclusively before or immediately after birth, and are not influenced by any treatment received in the NICU. These features record:

- gestational age in weeks (both raw and squared);
- sex of the infant;
- inborn or outborn status of the infant[3];
- Apgar score measured one minute after birth;
- vaginal or cesarean delivery;
- if the infant was part of a multiple gestation;
- if the infant was diagnosed with congenital anomalies;
- if the infant was small for gestational age[4].

[2]The Apgar score is a measure of health of the newborn developed by Virginia Apgar in 1952. It is determined by evaluating the baby's health status on five simple criteria on a scale from zero to two, then summing up the five values thus obtained. The resulting Apgar score ranges from zero to 10. These criteria are: skin color, pulse rate, reflexes, activity and respiratory effort.

[3]Indicates whether the child was delivered in the center, or delivered in another center and then brought to the NICU (for example by ambulance transfer).

[4]Indicates whether the infant's weight was below the 10th percentile for the gestational age

3.2 Data

The dataset that we utilized in our study is a subset of the VON's Extended Dataset, consisting of low birth-weight infants who were registered in Italian centers from 1997 to 2014. It comprises 35282 records and 179 features, which include demographics of the newborn and his/her mother, measurements of the health status of the newborn, medical history of the mother, as well as clinical procedures that the infant might or might not have underwent after birth (for example in presence of post-delivery complications).

From this wide spectrum of predictors, our neonatologists support team identified, on the basis of their medical experience, a restricted subset of features that are considered influential in the mortality of neonates immediately after birth. The features record some aspects of the medical history of the mother before and during pregnancy, and a series of demographic information and measurements assessed directly on the newborn, before or after birth. After this screening, a total of 20 features were preserved:

- `bwgt`: birth-weight of the infant;
- `gaweeks`: weeks of gestational age;
- `gadays`: days of gestational age. This sums up to `gaweeks` to obtain the total number of days of gestational age;
- `bheadcir`: head circumference of the infant at birth;
- `deldie`: whether the patient was born in the center, but was not admitted in the NICU and died in delivery room within 12 hours after birth;
- `locate`: inborn or outborn status of the infant;
- `pcare`: whether the mother received prenatal obstetrical care prior to the admission during which birth occurred;
- `aster`: whether antenatal corticosteroids were administered to the mother during pregnancy at any time prior to delivery;
- `amagsulf`: whether magnesium sulfate was administered to the mother during pregnancy at any time prior to delivery;
- `chorio`: whether a diagnosis of chorioamnionitis[5] was recorded in the maternal or infant medical record;
- `mhypertens`: whether maternal hypertension, chronic or pregnancy-induced, was recorded in the maternal or infant medical record;
- `vagdel`: mode of delivery (either vaginal or cesarean);

[5]Chorioamnionitis is an inflammation of the fetal membranes (amnion and chorion) due to a bacterial infection.

- `sex`: sex of the infant (male or female);
- `mult`: whether the birth was part of a multiple gestation (two or more live fetuses were documented at any time during the pregnancy which resulted in the birth of the infant);
- `nbirths`: if the birth was part of a multiple gestation, records the number of infants actually delivered;
- `ap1`: Apgar score measured one minute after delivery;
- `ap5`: Apgar score measured five minutes after delivery;
- `atempm`: whether the infant's body core temperature was measured within one hour after admission;
- `atemp`: if the temperature was measured, records its value;
- `sga10`: whether the infant was small for gestational age.

Notice that in this selection, all the features used to construct the VON-RA model are included or easily computable from the available data (such as the gestational age in weeks squared). The only exception is the feature indicating whether the newborn was diagnosed with congenital anomalies, which is not present in the selection. To avoid bias in the results, the data was filtered beforehand to exclude infants with such major birth defects.

The target of interest for the purposes of this work is mortality prior to discharge home during the first 28 days after birth. The observed mortality for each patient is reported in the dataset in a column named `died`. Finally, column `pred` reports the mortality estimation for each patient, as calculated by the VON-RA model. Table **3.1** provides a quick summary of the data, reporting for each feature its unit of measurement/accepted values, type, number of missing values and their percentage ratio with respect to the total number of observations.

3.3 Objectives

We can divide this study in two separate lines of work. The first part is purely supervised, and focuses on:

- understanding if the new set of features selected by the neonatologists support team has a positive contribution in the assessment of LBWI mortality, with respect to using a set of predictors that resembles the ones used by the VON-RA model;
- comparing the VON-RA against a pool of state-of-the-art machine learning models and see in which cases we could reach an improvement, trying to

Name	Units/Values	Value Type	Missing	Missing %
bwgt	grams	Float	5	0.01%
gaweeks	weeks	Integer	7	0.02%
gadays	days	Integer	70	0.20%
bheadcir	cm	Float	4060	11.51%
deldie	yes/no	Boolean	0	0%
locate	yes/no	Boolean	0	0%
pcare	yes/no	Boolean	427	1.21%
aster	yes/no	Boolean	690	1.96%
amagsulf	yes/no	Boolean	24285	68.83%
chorio	yes/no	Boolean	9147	25.93%
mhypertens	yes/no	Boolean	7704	21.84%
vagdel	vaginal/cesarean	Boolean	17	0.05%
sex	male/female	Boolean	7	0.02%
mult	yes/no	Boolean	3	0.01%
nbirths	range	Integer	24379	69.10%
ap1	range	Integer	181	0.51%
ap5	range	Integer	751	2.13%
atempm	yes/no	Boolean	4140	11.73%
atemp	degrees	Float	11625	32.95%
sga10	yes/no	Boolean	15	0.04%
pred	probability	Float	36	0.10%
died	yes/no	Boolean	418	1.18%

Table 3.1: Description of the dataset.

understand how the complexity of each model and the particular feature set of choice affect the final performances.

The first objective lead us to create two different datasets, one based on features used by the VON-RA model, another by manipulating and transforming the set of features that were suggested to us by our neonatologists support team to obtain a new selection of predictors that our understanding of the problem and statistical analysis considered of relevance. To fulfill the second point, we built a framework that comprises optional feature scaling to optimize predictive power, double cross-validation to estimate the performance of the class of learning algorithms to which the candidate belongs, model selection with cross-validation to produce a final model, model evaluation through out-of-sample testing, and statistical comparison of the final results to assess significance of the differences in performance. Everything was devised in accordance with machine learning theoretical and practical principles, in a way that is as agnostic as possible with respect to the particular model and set of features of choice, to allow us to reason about which factors are likely to be relevant in the prediction of LBWI mortality without the constraints imposed by a fixed learning algorithm or dataset. Our belief at this stage of the study was that joint effort of the new set of predictors and the use of machine learning models that are able to represent complex hypothesis spaces could result in improved performances on the task. We present the results of the experiments in Section **4.3**.

The second branch of this study focuses on interpretation and on characterizing the task of predicting LBWI mortality, improving the knowledge of the problem. It is composed of a mix of both unsupervised and supervised techniques. Unsupervised learning models were employed to understand if the VON data has an inherent structure that can be discovered by exploratory analysis: in particular, we used Self-Organizing Maps and analyzed the results, both statistically and using the visualization techniques a SOM is capable of. Obviously, no prior information on the observed mortality was used in the training (as an unsupervised technique implies). Our intent was to understand if and how the SOM would aggregate units that represent clusters of higher mortality, and whether unexpected factors had a contribution in making an infant more or less likely to die. The results are shown in Section **4.2**.

As a companion to the unsupervised analysis, we decided to use Decision Trees, which cater for an easy interpretation of the decision process, to have an insight on the choices that a supervised model makes when performing predictions on LBWI mortality. The advantages of using Decision Trees are two-fold: on one hand, they allow to see the prediction process as a series of sequential conditional choices that result in an outcome, resembling the process of how a diagnosis is made. On the other hand, the order in which these choices are made provides a rank among the features, which is useful to understand the discriminative power that each feature provides. The results are shown in Section **4.1**.

This second part also marks the starting point for future studies, where the additional knowledge of the underlying data acquired with unsupervised techniques and the means of interpretation provided by the Decision Trees learning analysis might be exploited to construct better models for the problem.

3.4 Data preprocessing

In this section, we report the procedure we utilized to extract the two datasets of interest from the larger one that was provided to us by our neonatologists support team.

3.4.1 The LBWI-V dataset

In light of the need to compare the impact of the new set of features against the VON-RA predictors, as explained in Section **3.3**, we created an initial dataset selecting the same set of features as the ones of the VON-RA model. The only data preprocessing steps that were required in this phase were:

- the creation of the `gasq` feature by squaring the values of `gaweeks`;

- the selection of the features that are present in the VON-RA set of predictors;
- the exclusion of rows that contained missing values or outliers.

The resulting dataset comprised 33808 observations and 8 features (plus the target columns `died` and `pred`, which was kept as reference to perform the comparisons against the VON-RA). We named this dataset **LBWI-V(ON-RA)**, to make clear its relationship with the set of features of the VON-RA model. The statistics of the LBWI-V dataset are summarized in Table **3.2**.

Name	Mean	St. Dev.	Min.	1st Perc.	Median	3rd Perc.	Max.
gaweeks	28.99	2.91	23.00	27.00	29.00	31.00	40.00
gasq	848.73	169.13	529.00	729.00	841.00	961.00	1600.00
locate	0.09	0.29	0.00	0.00	0.00	0.00	1.00
vagdel	0.19	0.39	0.00	0.00	0.00	0.00	1.00
sex	0.49	0.50	0.00	0.00	0.00	1.00	1.00
mult	0.32	0.46	0.00	0.00	0.00	1.00	1.00
ap1	5.83	2.30	0.00	4.00	6.00	8.00	10.00
sga10	0.28	0.44	0.00	0.00	0.00	1.00	1.00

Table 3.2: Statistics of the LBWI-V dataset.

To help understand the distribution associated to each feature, we visualize in Fig. **3.1** the box plots of the predictors, where the values of `gaweeks`, `gasq` and `ap1` have been transformed to have mean 0 and unit standard deviation for readability purposes.

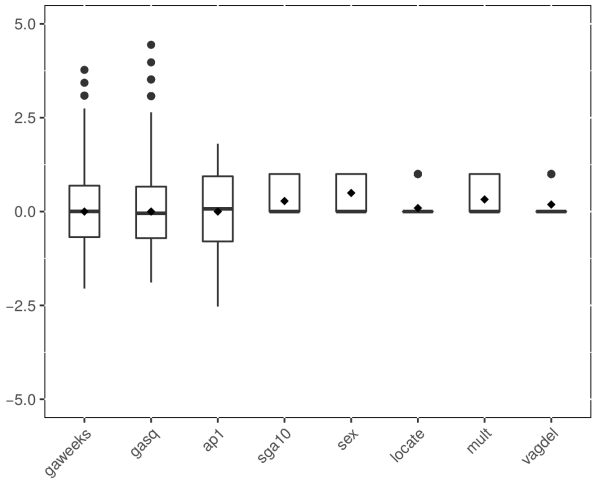


Figure 3.1: Box plots of the LBWI-V dataset. The boxes represent the interquartile range of the population distribution, with the median indicated by a black bar; the bars outside the box represent the remaining quartiles; outliers are represented as circles at the edges of the bars; the mean of each feature is indicated with a diamond shape.

3.4.2 The LBWI-G dataset

The second dataset, which was derived from the features selected by the neonatologists, is conceptually no different from the VON-RA. Its features represent information collected on the newborn immediately after birth, not taking into account possible treatments occurred later in the hospital stay; the main difference is the presence of information regarding the clinical history of the neonate's mother, which are believed to play a role in the infant's outcome (as an example, [19] states that the administration of antenatal corticosteroids is indicated for women at risk of premature delivery). Unlike the LBWI-V dataset, where the set of features was predefined, this dataset was derived by dropping or transforming the original 20 features, reducing their number from 20 to 14. In particular:

- `gadays` and `gaweeks` were aggregated into the sole `gadays` feature to express the whole gestational age in days. Consequently, `gaweeks` was dropped;
- the information contained in `mult` and `nbirths` was aggregated into the sole `nbirths` feature. Consequently, `mult` was dropped;
- `sga10` was dropped, because its information was believed to be redundant with respect to `bwgt`;
- `deldie` was dropped, because the fraction of patients who did not died in the delivery room was too small to provide useful discriminative power;
- `atemp` and `atempm` were dropped, because their inclusion was believed to bias the predictions, since temperature is measured almost always only on newborns that are already in critical conditions;

Name	Mean	St. Dev.	Min.	1st Perc.	Median	3rd Perc.	Max.
bwgt	1106.46	297.94	295.00	880.00	1150.00	1365.00	2450.00
gadays	206.14	20.16	161.00	192.00	207.00	221.00	280.00
bheadcir	26.32	2.65	18.00	24.50	27.00	28.00	43.00
locate	0.08	0.27	0.00	0.00	0.00	0.00	1.00
pcare	0.91	0.28	0.00	1.00	1.00	1.00	1.00
aster	0.83	0.37	0.00	1.00	1.00	1.00	1.00
amagsulf	0.08	0.27	0.00	0.00	0.00	0.00	1.00
chorio	0.13	0.33	0.00	0.00	0.00	0.00	1.00
mhypertens	0.25	0.43	0.00	0.00	0.00	1.00	1.00
vagdel	0.18	0.39	0.00	0.00	0.00	0.00	1.00
sex	0.49	0.50	0.00	0.00	0.00	1.00	1.00
nbirths	0.72	1.04	0.00	0.00	0.00	2.00	4.00
ap1	5.90	2.22	0.00	5.00	6.00	8.00	10.00
ap5	7.77	1.60	0.00	7.00	8.00	9.00	10.00

Table 3.3: Statistics of the LBWI-G dataset.

All the other features were left as-is. After having excluded rows that contained missing values, the dataset consisted in 9631 rows and 14 features, losing something close to 70% of the original size. This reason to such important loss of data is the indication, expressed by the neonatologists support team, to retain features

`aster`, `chorio`, `mhypertens` and `amagsulf` in the modeling (as per specific request of the clinicians), which have a conspicuous number of missing values, mainly because their inclusion in the VON Manual of Operations was established only lately. This is however a problem that will be mitigated in the years to come, since they are nowadays steadily collected by VON centers.

For the rest of this work, we will name this dataset the **LBWI-G(agliardi)** dataset, after the head of the clinicians team that helped us in our study. Table **3.3** shows a summary of the statistics of the dataset after the described preprocessing, while Fig. **3.2** provides a graphical representation of the feature distributions with the use of box plots.

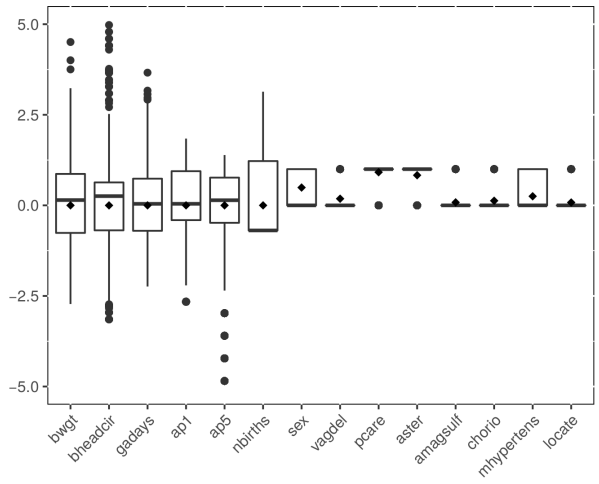


Figure 3.2: Box plots of the LBWI-G dataset. See Fig. **3.1** for a better understanding

.

3.5 Exploratory analysis of the two datasets

Comparing the two dataset from a statistical point of view is difficult, because they share only a small portion of features (although some are strongly related), and because one outnumbers the other in terms of observations. However, we can derive some interesting insights by inspecting the distributions of the data. Some notions of interest are:

- `gaweeks` in the LBWI-V and `gadays` in the LBWI-G dataset share a very similar distribution, although one is scaled by a factor of 7 (the number of days in a week). Indeed, the quotient of the mean of `gaweeks` divided by 7 is 29, which is close to the mean of `gaweeks` (28.99). The same is true for the standard deviation: dividing by 7 the one of `gadays`, we obtain 2.88, which is again close to the standard deviation of `gadays` (2.90);

- all the binary features that are shared between the two datasets (`locate`, `vagdel` and `sex`) have resembling means and standard deviation, as can be easily verified by inspecting Table **3.2**) and Table **3.3**;
- although the `ap1` feature appears to have a slightly different distribution in the two datasets, it shares a very similar normal curve approximation[6], as shown in Fig. **3.3**.

The datasets also present some dissimilarities. If we count observations whose birth-weight is below the tenth percentile (which is the condition for being defined small for gestational age) in the LBWI-G dataset, we observe that the ratio of their number with the total number of observations is 0.098, which is almost three times smaller than the mean of the `sga10` feature in the LBWI-V dataset (0.28). Another inconsistency, although less impacting, is present with respect to the `mult` feature in the LBWI-V dataset, whose mean in 0.32. If we select observations in the LBWI-G dataset whose `nbirths` value is 0, their mean is 0.28, which is a little smaller than what should have been expected in a perfect-matching setting.

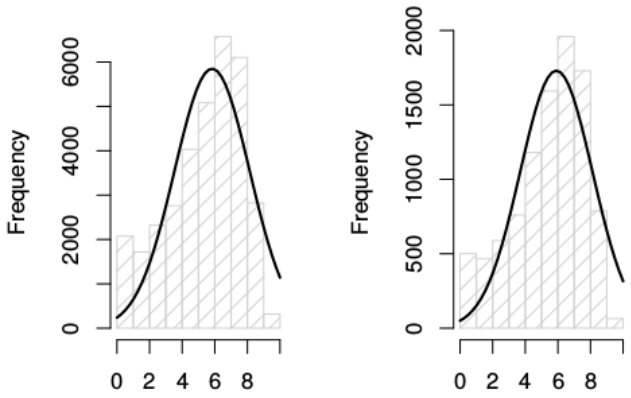


Figure 3.3: Histograms of the `ap1` feature (in light gray) on the LBWI-V dataset (left) and LBWI-G dataset (right). The overlaying black line represents the normal curve approximation of the feature distribution.

Table **3.4** shows the comparison between the observed outcomes for the datasets and the predicted outcomes as produced by the VON-RA model. From this table, we can make some useful considerations. Firstly, as expected, outcomes are heavily imbalanced in all the datasets: only 14% of the infants in the original dataset effectively died after birth. The rate drops only slightly (13% for the LBWI-V data and 12% for LBWI-G data) in the two datasets we constructed. Since our main performance measure of reference is AUC-ROC, which is not affected by class imbalance, we did not consider the disproportion of the classes to be a problem. However, we

[6]We made the implicit assumption that the distribution of Apgar scores is normal, which is plausible at least from a visual point of view, since much of the data is concentrated around the mean and its tails are thin.

wish to remind that future studies which use different metrics to evaluate classifiers on these data need to take appropriate counter-measures to mitigate the imbalance.

Secondly, the predictions of the VON-RA model seem to underestimate the true mortality rate in the dataset: a statistical deviance between the observed and predicted mortality was again to be expected since the sample sizes between our dataset and the larger one used to build the VON-RA model (which comprises infants admitted in about 1000 NICUs from all around the world) are very different. However, both the LBWI-V and the LBWI-G datasets have about the same predicted mortality rate of 9%, despite the latter using only less than a third of the data. This suggests us that, although any hypothesis test to compare LBWI-V and LBWI-G would surely reject the null hypothesis because of the large difference between the two sample sizes, the two datasets might actually have more similar characteristics than they look like from a statistical point of view.

Dataset	No. records	No. features	Obs. outcome	Std. Dev.	Pred. outcome	Std. Dev.
Original	35282	20	0.14	0.35	0.10	0.17
LBWI-V	33082	8	0.13	0.34	0.09	0.15
LBWI-G	9631	14	0.12	0.32	0.09	0.15

Table 3.4: Comparison between observed and predicted mortalities, in the original data collection as well as in the two derived datasets.

3.6 Decision Tree learning experiments setup

As a first experiment, we trained a simple decision tree and used it to understand more about the data. Since decision trees offer great interpretability of the classification outcome under the form of sequential rules which can be easily visualized, we used them in order to:

- understand which features are the most influential in the decision process;
- investigate nodes of the tree where the learner is uncertain;
- analyze the statistical properties of the misclassified patients, to see in which cases the learner is prone to commit a classification error.

The objective of this analysis was to provide a preliminary insight into the dataset with respect to the aspects described above. In this sense, we did not expect a simple Decision Tree to achieve the best predictive performance nor we targeted at accuracy optimization in this stage. Since the objective of this particular experiment is problem understanding and characterization, we need not follow a strict machine learning protocol for learning, hence we did not use any performance estimator like cross-validation or even training and testing folds. We simply resorted to perform

some statistical analysis on a moderately complex tree built on the whole LBWI-G dataset, to verify if any interesting pattern emerged with respect to the matters listed above.

3.7 Unsupervised experiments setup

Most of the work for the unsupervised part of our experiments involved the selection of a suitable map size for the SOM. Since the unsupervised nature of the task does not provide clear guidelines on how to select sensible parameters of the SOM, most of this part consisted in trial-and-error on different map sizes until results that were considered useful were observed. We do not describe in detail the trial-and-error phase; for the purposes of our work, we only report that the final map that was selected after this phase was 48×48 in size, totaling 2304 neurons. However, there were other hyper-parameters that had to be selected for the map, such as the type of neighborhood function. All these additional parameters were selected with a grid search strategy, trying a number of different combinations and choosing the configuration with the lowest quantization error. These parameters were:

- `init`: the algorithm used to initialize the weights of the map. We chose among random initialization and a procedure where the values of the weights are selected from the subspace spanned by the first two principal components derived from the data;
- `annealing`: the annealing scheme used to shrink the learning rate after each epoch, either based on a linear or quadratic function;
- `kernel`: the type of neighborhood function.

Table **3.5** illustrates the type of hyper-parameters and their range of values. In addition to this configuration, we tested 50 equally-spaced values for the starting radius of the neighborhood function, ranging from 2/3 of the diameter of the map down to 2 (meaning adjacent neuron interactions only). With this optimization set-up in place, we fixed an a-priori squared shape for the map, trained 32 maps with different side lengths and supplied their respective visualizations to the clinicians, letting them select the most promising map based on its visual properties.

Hyper-parameter	Type	Range/Values
`init`	List	`random, pca`
`annealing`	List	`power, linear`
`kernel`	List	`zeroone, gaussian`

Table 3.5: Hyper-parameters of the SOM.

3.8 Supervised experiments setup

The supervised experiments whose results we expose in **??** were thought as a sequence of definite steps, which at a high level consisted in:

- perform an initial selection of promising models to be compared with VON-RA;
- set-up a reliable process to select the hyper-parameters of the learning algorithms;
- obtain a final model to be tested on new data;
- test the significance of the results.

Many of the algorithms that were used in this work make use of generated random numbers. To ensure that all of our work is reproducible, we set a unique random seed for every algorithm tested, meaning that the way random numbers are generated is always the same, independently of the learning algorithm or the particular step of the analysis.

The learning algorithms of choice for this study were Logistic Regression, k-Nearest Neighbors, Support Vector Machines, Random Forests, Gradient Boosting Trees, which is a particular instance of Gradient Boosting Machines that uses Decision Trees, and Neural Networks. Other learning algorithms were tried in an initial screening phase, but were then discarded because they did not provide satisfactory results. In populating the pool, we particularly cared that the included algorithms expressed different learning paradigms as well as different capabilities in terms of the complexity of the hypothesis space they represent, to ensure an exhaustive as possible comparison with VON-RA. A more in-depth description of the various steps that compose our framework is illustrated in the next sub-sections.

3.8.1 Feature scaling

Scaling features (i. e. transforming them to have fixed mean and standard deviation, or to lie in a specified interval) before the data is used to train machine learning models is of primary importance. To think why *feature scaling* is useful, remember how the value of the input features play a role in the update of the weights during gradient descent. By having the input on different scales, we are allowing weights that correspond to larger input values to receive a greater update than weights that link smaller input values. As a results, some weights might update faster than others and the algorithm might be slower or unable to converge. Tree-based algorithms such as Random Forests and Gradient Boosting Machines do not usually suffer from this issue, since the scale of the features does not affect the way the split

of the nodes is performed. Such differences need to be taken into account by the framework.

Therefore, we designed feature scaling as an optional step of the model selection phase: the scaling is performed only *after* the data is split by the cross-validation procedure into training and validation, and we processed the data in the validation fold with the parameters obtained in the training phase. This ensures that there is no "leak" of information from the test to the training data, that might lead to bias the cross-validation estimation. The feature scaling strategies that were taken into account in the pipeline are:

- transform the features to have 0-mean (known as *demean transformation*);
- transform the features to have 0-mean and unit variance (known as *standardization*);
- scaling the features in the range $(0, 1)$ by subtracting their minimum value and dividing by their range (known as *min-max scaling*);
- no scaling.

Feature scaling is a step intended only for continuous (or continuous-like) predictors. Since binary features take only values 0 or 1, they already lie in a suitable interval for training and hence need not to be scaled.

3.8.2 Double cross-validation

As we explained in Section **2.5.1**, double cross-validation is a tool to estimate which out-of-sample performances we should expect from a fixed learning algorithm. For our experiments, we tested each of the models in the pool using a 3-fold cross-validation outer loop and a 5-fold validation inner loop. The number of hyperparamter configurations that are tested in each repetition of the inner loop were set to 100. Fine grainer solutions (such as 5-fold outer-loop/10-fold inner loop, which in theory provide even more precise ranges) were discarded mainly for computational reasons. However, we believe that 3/5 double cross-validation is suitable to our purposes: in the end, 1500 different models for each learning algorithm were evaluated on 15 different realization of the dataset, which we believe is a sufficient amount of computation to estimate of the behavior of the various learning algorithms at the task. At the same time, we needed to compare the various estimates with the VON-RA performances. To this extent, we had at our disposal the `pred` column in the dataset. To obtain comparable performances, we hold it aside, calculating partial results on subsets based on the same data splitting that was performed on the dataset at each moment of the procedure. As a result, the final performances of the VON-RA are obtained on samples of equal size, containing the exact same observations which were used to calculate performances of the

particular learning algorithm that was tested. The actual procedure is explained with the help of pseudo-code in Appendix **A**.

3.8.3 Final model selection

After having obtained reasonable estimations of the behavior of each learning algorithm with double cross-validation, the next phase of our experiments was to select a final model for each learning algorithms using all the data at our disposal, to perform out-of-sample tests.

Learning algorithm	Hyper-parameter	Type	Range/Values
All	`scaler`	List	`None, minmax, demean, std`
Logistic Regression	`C`	Float	$0.01 - 2000$
	`penalty`	List	`l1, l2`
	`warm_start`	List	`True, False`
k-Nearest Neighbor	`n_neighbors`	Integer	$3 - 300$
	`p`	Integer	$1 - 2$
	`weights`	List	`uniform, distance`
Support Vector Machine	`C`	Float	$0.001 - 1000$
	`gamma`	Float	$0.001 - 10$
Random Forest	`bootstrap`	List	`True, False`
	`criterion`	List	`gini, entropy`
	`max_depth`	Integer	$3 - 15$
	`max_features`	List	`None, log2, sqrt`
	`min_samples_split`	Integer	$10 - 100$
	`n_estimators`	Integer	$10 - 100$
Gradient Boosting Trees	`colsample_bytree`	List	$0.1 - 0.9$
	`learning_rate`	Float	$0.01 - 0.3$
	`max_depth`	Integer	$3 - 15$
	`n_estimators`	Integer	$10 - 500$
	`gamma`	Float	$0.000001 - 0.001$
	`subsample`	Float	$0.1 - 0.9$
Neural Network	`activation`	List	`relu, tanh`
	`batch_size`	Integer	$64, 128, 256, 512$
	`dropout`	Float	$0.1 - 0.8$
	`epochs`	Integer	$20 - 150$
	`hidden_units`	Integer	$14 - 300$
	`init`	List	`uniform, normal, lecun`
	`learning_rate`	Float	$0.0001 - 0.1$
	`momentum`	Float	$0.7 - 0.95$
	`optimizer`	List	`sgd, adam, rmsprop`

Table 3.6: Summary of the hyper-parameter distributions that were randomly sampled to create candidates during model selection.

This desire is motivated by the fact that all the models trained and evaluated during the double cross-validation phase used only smaller subsamples of the data, which for the LBWI-G dataset were (with slight variations due to the non-exact splittings):

- 1330 examples for training the models given a fixed hyper-parameters configuration inside the inner loop;
- 6554 examples to train the best model that resulted from the model selection that was performed by the inner loop;

- 3277 examples to evaluate the above-mentioned best model.

Therefore, we performed a subsequent model selection using 10-fold cross-validation, in order to obtain valid estimates of the generalization capabilities of each of the learning algorithms in the pool. The double cross-validation phase also gave us suitable ranges for the hyper-parameter configurations to be explored during this last model selection phase, in order to find their best values. Table **3.6** provides a summary of the hyper-parameter distributions of each learning algorithm, with the type of values they accept and their sampling ranges. The explanations of each hyper-parameter can be found in Appendix **B**, while the final hyper-parameter values of the best models found during this phase are reported, for each learning algorithm, in Appendix **C**.

3.8.4 Out-of-sample testing

The subsequent step in our experiments was to evaluate the final models on a fresh dataset of infants from the Italian cohort, born in year 2015, which was not part of the initial data but was obtained separately. The key point to emphasize is that this test dataset concerns a completely new year with respect to the data used to train the models: on the basis of this fact, we set our expectations on the performances we would be able to obtain. Specifically, assuming that the modeling part was carried out correctly, at least two mutually exclusive scenarios could have happened:

- new clinical practices (or advancements in neonatal care in general) could have changed the nature of the problem, perhaps adding new relationships among the predictors and making others no more useful to correctly predict infant mortality;
- the nature of the problem could have remained somehow "stationary", meaning that infant mortality in 2015 was still well-predictable using past data.

In the former case, we expected our models to perform poorly, simply because they would fail to capture new relationships that were not present in their training data. In the latter case, any possible improvement accomplished, even if narrow, would acquire additional value because it was obtained against a consolidated model such as the VON-RA, which is adjusted year after year to adapt to advancements in neonatal care.

The test dataset was preprocessed in the same way as the original dataset: from an initial size of 3245 examples and 20 features, two test datasets were created, one using the LBWI-V preprocessing, and the other using the LBWI-G preprocessing. Statistics for both the datasets can be viewed in Table **3.7** and Table **3.8** respectively.

Name	Mean	St. Dev.	Min.	1st Perc.	Median	3rd Perc.	Max.
gaweeks	28.94	2.89	23.00	27.00	29.00	31.00	39.00
gasq	845.90	167.48	529.00	729.00	841.00	961.00	1521.00
locate	0.07	0.25	0.00	0.00	0.00	0.00	1.00
vagdel	0.19	0.39	0.00	0.00	0.00	0.00	1.00
sex	0.49	0.50	0.00	0.00	0.00	1.00	1.00
mult	0.36	0.48	0.00	0.00	0.00	1.00	1.00
ap1	5.83	2.20	0.00	4.00	6.00	7.00	10.00
sga10	0.20	0.40	0.00	0.00	0.00	1.00	1.00

Table 3.7: Statistics of the LBWI-V test dataset.

To complete our description of the test data, we provide in Table **3.9** the comparison of the observed and predicted outcomes for both the LBWI-V and LBWI-G datasets, as well as the original one from where they were extracted.

Name	Mean	St. Dev.	Min.	1st Perc.	Median	3rd Perc.	Max.
bwgt	1112.72	288.88	285.00	900.00	1155.00	1358.00	2409.00
gadays	206.19	19.90	161.00	193.00	207.00	221.00	276.00
bheadcir	26.38	2.58	19.00	25.00	27.00	28.00	39.50
locate	0.06	0.23	0.00	0.00	0.00	0.00	1.00
pcare	0.94	0.22	0.00	1.00	1.00	1.00	1.00
aster	0.88	0.33	0.00	1.00	1.00	1.00	1.00
amagsulf	0.12	0.33	0.00	0.00	0.00	0.00	1.00
chorio	0.14	0.35	0.00	0.00	0.00	0.00	1.00
mhypertens	0.25	0.43	0.00	0.00	0.00	1.00	1.00
vagdel	0.18	0.38	0.00	0.00	0.00	0.00	1.00
sex	0.49	0.50	0.00	0.00	0.00	1.00	1.00
mult	0.36	0.47	0.00	0.00	0.00	1.00	1.00
ap1	5.91	2.16	0.00	5.00	6.00	7.00	10.00
ap5	7.82	1.53	0.00	7.00	8.00	9.00	10.00

Table 3.8: Statistics of the LBWI-G test dataset.

3.8.5 Assessing the significance of the results

Since in this study the performance metric of reference is AUC-ROC, we needed a statistical hypothesis test to compare the differences in the areas under the ROC curve that our models produce, to understand if the models we trained have significantly different performances with respect to the VON-RA model or among themselves. One key observation in this sense is that, since the performances are evaluated on the same samples (the out-of-sample testing dataset we described previously), we need to take into account the correlation between the out-of-sample

Dataset	No. records	No. features	Obs. outcome	Std. Dev.	Pred. outcome	Std. Dev.
Original	3245	20	0.12	0.33	0.09	0.16
LBWI-V	3178	8	0.12	0.32	0.09	0.15
LBWI-G	2792	14	0.10	0.30	0.08	0.14

Table 3.9: Comparison between observed and predicted mortalities in the test datasets.

performances produced by the learning models. Given the premises, we resorted to employ a common and well-known, especially in the medical field, statistical hypothesis test for correlated ROC curves, by DeLong et al. ([13]). We refer to the paper for a full explanation of how the test is performed; to our purposes, it is sufficient to state that we rejected the null hypothesis (the true difference of the computed AUCs is 0) at the 5% significance level.

3.9 Software and tools

For the experiments conducted in this study, several frameworks and libraries were used. Because of the variety of such experiments, we employed different tools whenever a certain capability was desired, whether it was plotting facilities, better computational complexity or ease of use. The programming languages of choice to accomplish the tasks were the Python programming language [48] and the R programming language [44].

The bulk of the supervised experiments was realized using the `scikit-learn` library [41]. It is a well-known Python library for machine learning that comprises almost every tool a machine learning practitioner is likely to use: it provides implementations for a wide range of learning algorithms, evaluation metrics, estimators and data preprocessors. It is based on reliable Python libraries for scientific computing such as `numPy` and `sciPy` [39], which in turn provide wrappers for the underlying `C` and `Fortran` implementations of linear algebra routines. These low-level libraries are devised to allow faster calculations of vector-matrix and matrix-matrix multiplications, which are at the core of most learning algorithms. In addition, a wide variety of models is optimized to run on multi-core machines, exploiting the power of parallel computing: this feature was useful to decrease the total running time of the algorithms, especially during cross-validation. Moreover, it supports the famous data-manipulation library `pandas` [36], which was used to manage all the transformations applied on the raw data.

We employed two main alternatives to the `scikit-learn` framework during the supervised analysis, mainly because they were computationally faster than the corresponding `scikit-learn` implementations.
All the experimented Neural Networks were trained using the reliable `Keras` library [10], which is a wrapper for two framework libraries that are built to train Neural Networks using GPUs: `Theano` [55] and `TensorFlow` [1]. In our experiments, the `Theano` wrappers were used. For Gradient Boosting Trees, we used `XGBoost` [9], a library that provides a multi-core optimized implementation of Gradient Boosting Machines in a whole variety of languages, including Python. Both libraries, however, provide `scikit-learn` wrappers, hence they were easily integrated into the main framework.

The unsupervised part was mainly done using R. For the training of the SOM, we used the `yasomi` package [47], which provided the tools that were needed, in particular the ability to plot the SOM units as hexagons (which was desired to visualize the map in a more appealing way), as well as a number of useful visualization capabilities such as hit maps, U-Matrices and color coding. The Decision Tree was trained using R's `tree` package [45], which is a very well-known and reliable library that includes functions to visualize the tree and exploring its nodes.

For all the remaining plots we made in this work we used interchangeably Python (through the `matplotlib` library [28]) and R (through the `ggplot2` package [58]) graphical libraries.

The supervised experiments were almost exclusively performed on a machine made available by our Computer Science Department, a PowerEdge C4130 Dell server equipped with 2 Xeon E5-2670v3 CPUs, for a total of 48 virtual cores (24 physical), 128 GB of RAM and 4 NVidia M40 GPUs. The unsupervised part of the study, as well as all the plots, were all realized in a local machine.

Chapter 4

Results

In This chapter, we present the results of our experiments. It is divided in three parts: Section **4.1** describes the results of the use of a Decision Tree model in the context of data understanding, feature ranking and problem characterization in general. Section **4.2** illustrates the results of the training of a SOM on the LBWI-G dataset, with the intent of discovering potential structure that underlies the data using its powerful visualization capabilities. Finally, Section **4.3** focuses on the supervised learning experiments: we expose and discuss the performances of the models that were trained and evaluated using the machine learning framework we described in Section **3.8**.

4.1 Results of the Decision-Tree learning analysis

Before constructing the tree, we plotted in Fig. **4.1** a boxplot of the features grouped by outcome. This served as a source of comparison for the various sub-population that constitute the leaves of the tree. Some of the features were normalized to have mean 0 and standard deviation 1 for readability purposes. Looking at the picture, a clear distinction emerges between infants that died and infants that survived, especially regarding birth-weight, head circumference, gestational age and Apgar scores. We also see that mothers of children at risk have higher incidence of chorioamnionitis, lower administration of antenatal steroids, lower incidence of maternal hypertension, and higher incidence of delivery by cesarean section.

Fig. **4.2** shows the resulting Decision Tree. The library that we used to build the tree provides the user with a parameter to tune the complexity of the tree with pruning, allowing to regularize the model to achieve better generalization. Without pruning, the tree consisted in 1520 nodes and 527 leaves, which was clearly not suited for interpretation.

Furthermore, the lower levels of the tree add close to no discriminative power, at the point that one split has no real reason to be taken before another. In order to make the tree both readable and meaningful, we set the complexity parameter to cease to split a node (therefore creating a leaf) whenever its entropy is less than

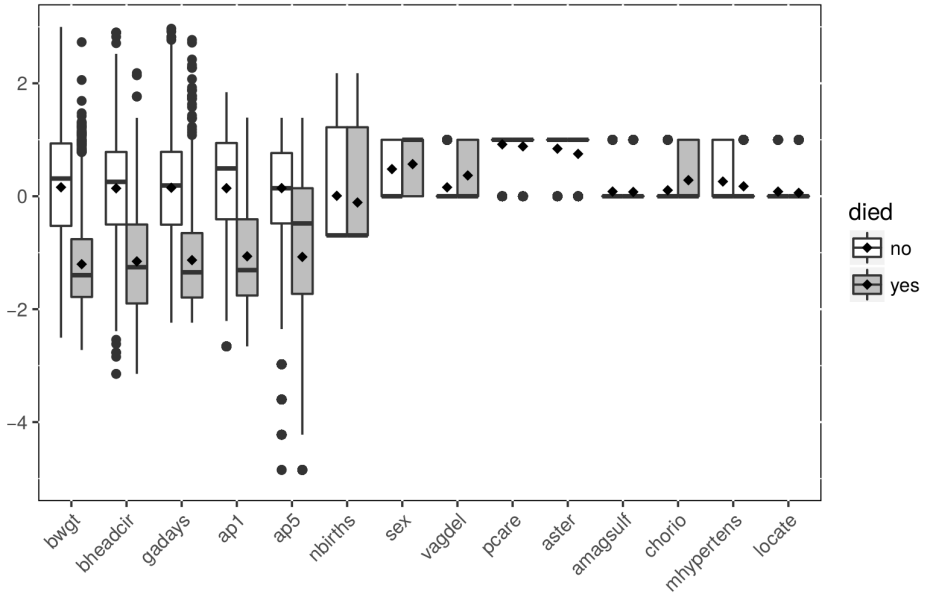


Figure 4.1: Box plots of the LBWI-G dataset grouped by outcome. Boxes represent the interquartile range of the population distribution, external bars represent the remaining quartiles. The median is indicated by a black bar. Outliers are represented as circles. The mean is indicated with a diamond shape.

1% of the one at the root node. This conforms with what anticipated in Chapter **3**, where we specified that the goal of this analysis is not to perform learning with the objective of optimizing predictive performance. As an aside, we report that the

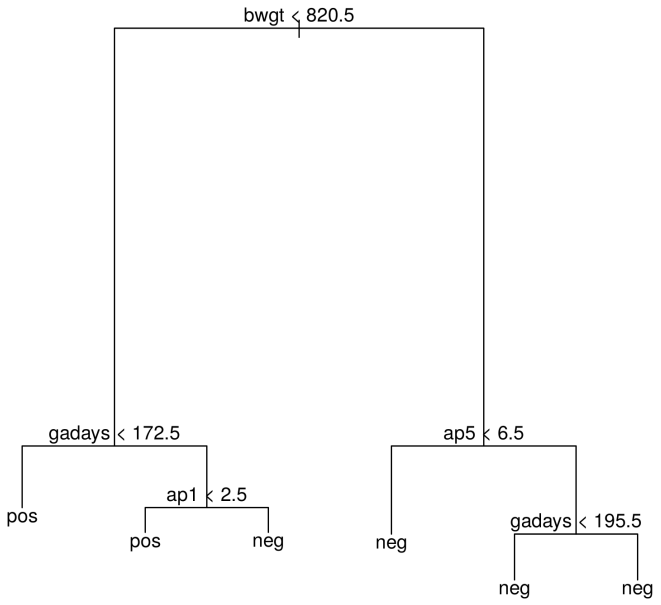


Figure 4.2: Plot of the tree obtained by the Decision Tree algorithm. The length of the vertical bars represents the decrease in loss: the longer the bar, the greater the decrease.

Decision Tree AUC-ROC was 0.845 in an out-of-sample test with the 2015 dataset, which is a fair and decent score for a standard learning algorithm using such a simplified structure. The pruned tree has 5 nodes and 6 leaves, and it is remarkable that one can obtain such a good AUC-ROC score with so few splits. We also see that the right sub-tree can be substituted with a single leaf labeled as negative: however, we decided not to prune the tree further since it seems to be not overly complex and have good generalization. The features that provide the largest decrease in the loss function are `gadays`, `ap1` and `ap5`.

Table **4.1** provides a summary of the leaves composition, starting from left to right: for each label, it is reported its reference number, the split to which it is associated, the number of observations contained in the leaf, the class label that was assigned by majority voting, and the probability associated to the positive class. The table shows that observations that are assigned to certain leaves are more likely to be misclassified. In particular, leaf 2 has almost a 48% chance to commit a false positive (since it is labeled with the positive class with 52% of the patterns) on new observations, leaf 3 has a 23% chance to commit a false negative, and so on. Leaf 6, on the other hand, correctly classifies as negative 99% of its 5931 assigned observations, meaning that a great number of patients (about 60% of the total dataset size) can be easily recognized by the algorithm as very likely to survive.

No.	Split	No. obs.	Class	Pos. prob.
1	`gadays` < 172.5	576	positive	0.65104
2	`ap1` < 2.5	291	positive	0.52234
3	`ap1` > 2.5	1120	negative	0.23214
4	`ap5` < 6.5	770	negative	0.20649
5	`gadays` < 195.5	945	negative	0.10476
6	`gadays` > 195.5	5931	negative	0.01551

Table 4.1: Description of the leaves of the tree.

Fig. **4.3** shows the box plot related to observations that ended up in the second leaf of the tree. If we focus on the infants that survived, we see that they have very similar characteristics with respect to infants that later died, in particular very low Apgar scores. This explains the large uncertainty of the decision process which resulted in a very poor classification inside the leaf. The observations in the leaf are also associated with below average birth-weight, gestational age and head circumference.

The opposite situation happens in leaf 3, whose box plots are displayed in Fig. **4.4**. This leaf was labeled as negative by majority voting. Note that the distributions of birth-weight, gestational age and head circumference are very much alike the ones in leaf 2. The largest differences are the Apgar scores: in leaf 3, both groups (dying and not dying) of infants are very similar, this time almost on average, even if observations in the dying group score slightly lower. All this analysis suggests that the Apgar scores are the most influential features in determining the outcome of the

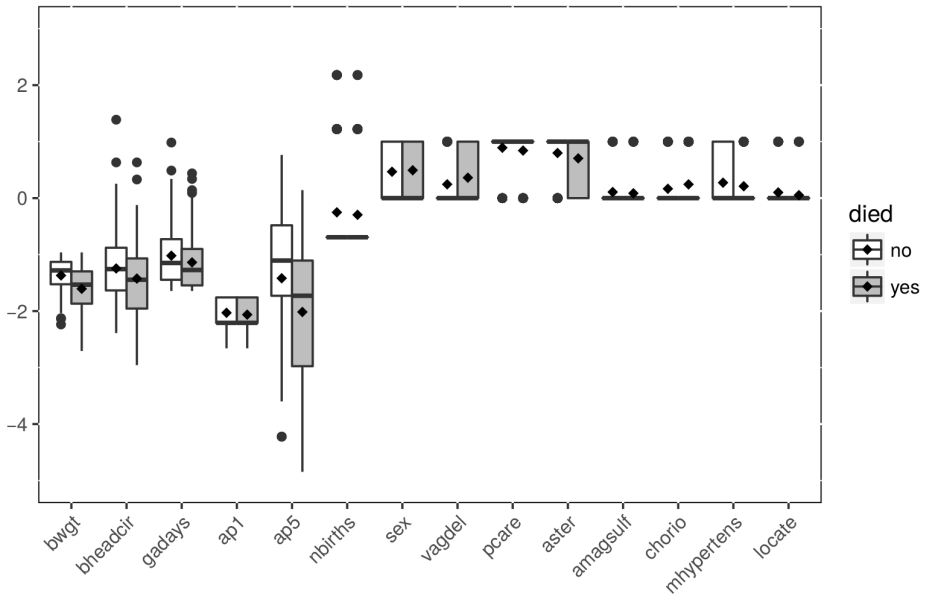


Figure 4.3: Box plots of leaf 2 of the Decision Tree, grouped by outcome.

infant, because newborns with similar Apgar scores but different outcomes have a higher chance to be misclassified. This is a result to be expected given the nature of the Apgar score, which is based on clinical parameters but is influenced by the effort of the clinician, which for integrates the evaluation with information coming from his/her working experience and professional training, as well as antenatal knowledge about the infant and the mother.

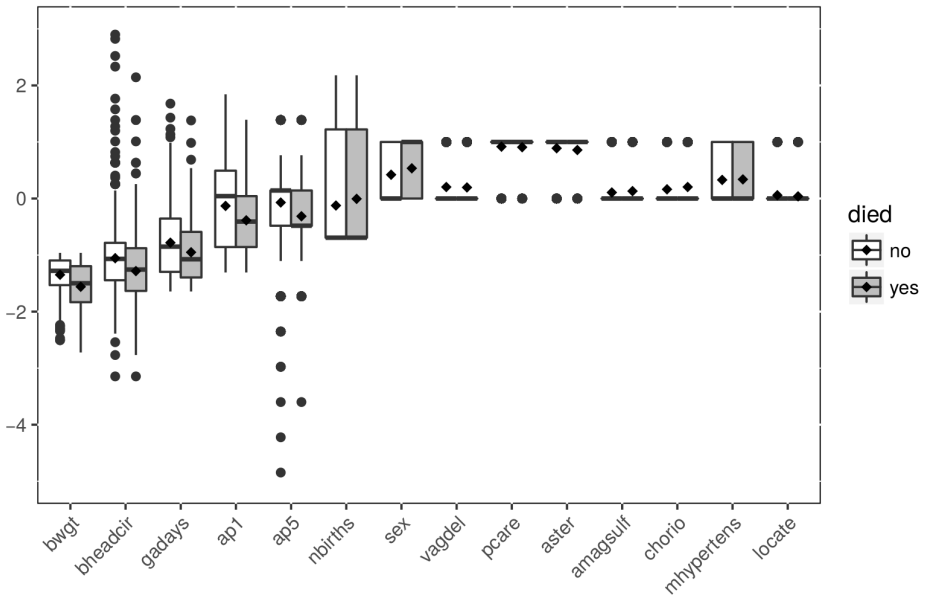


Figure 4.4: Box plots of leaf 3 of the Decision Tree, grouped by outcome.

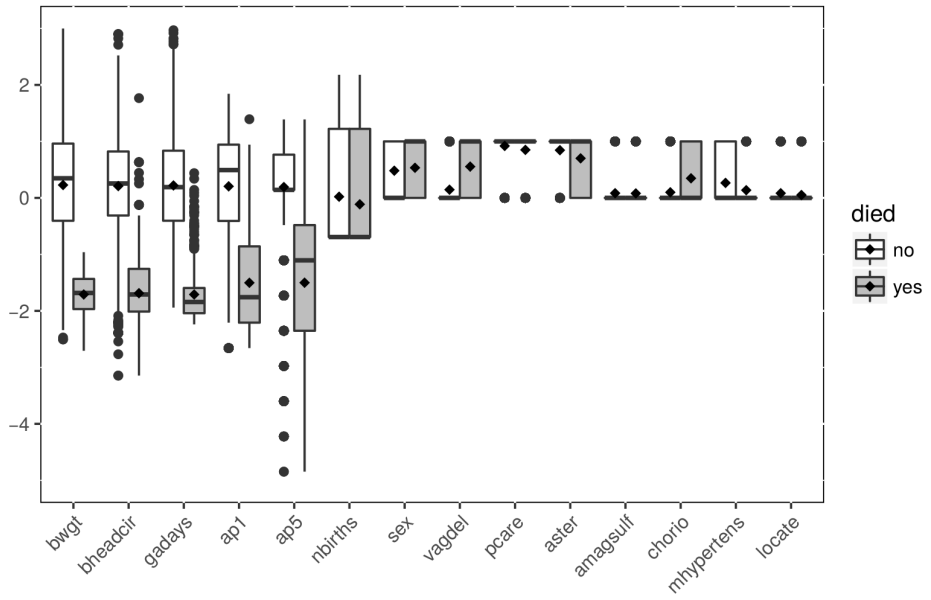


Figure 4.5: Box plots of observations correctly classified by the Decision Tree algorithm.

Finally, Fig. **4.5** shows the statistics of all the observations that were correctly classified by the algorithm, while Fig. **4.6** shows the statistics of misclassified observations. As clearly visible, most statistics are almost reversed if compared with patients that were correctly classified, which is again to be expected since the algorithm is wrong on infants that had a safe delivery and then died later due to complication that were not seen before birth, or by infants who were delivered in critical conditions but recovered their health later.

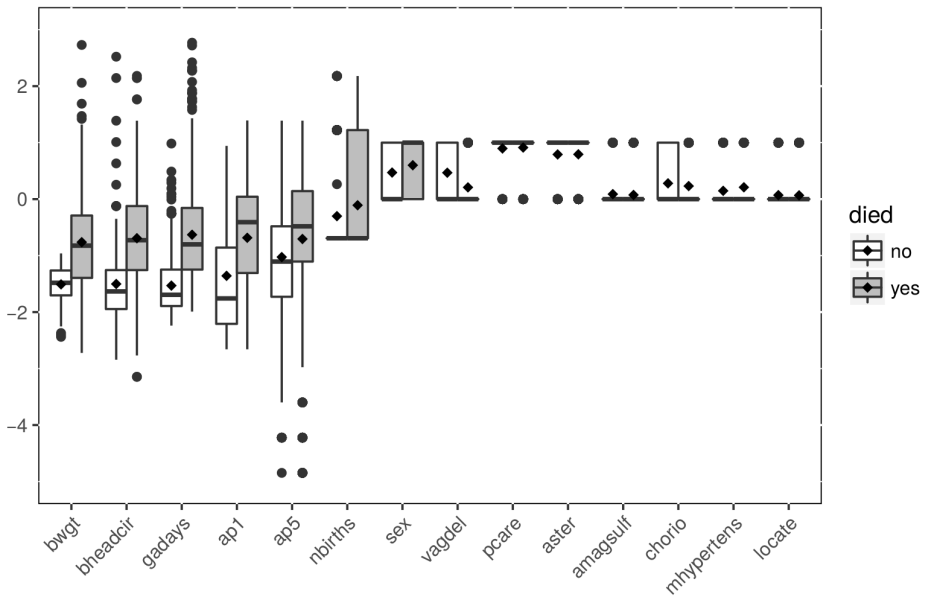


Figure 4.6: Box plots of observations misclassified by the Decision Tree algorithm.

4.1.1 Feature ranking

Decision Trees are useful also because they allow to compute a ranking among features. To calculate this, we used the complete tree that was built in the previous section, without pruning. The calculation was performed by inspecting every node of the tree and summing up, for each feature, the decrease in impurity that the related split caused. Since at each node only one feature is split, the sum over all nodes gives an overview on what are the most influential nodes in the tree construction, and also what is their overall contribution in the discrimination of examples. Table **4.2**, shows the importance that was calculated, with the features ordered by decreasing importance.

Feature	Mean Decrease in Impurity
bwgt	761.1006
gadays	737.5351
bheadcir	622.0572
ap1	525.0391
ap5	503.5190
sex	39.9395
vagdel	35.0369
aster	33.6687
nbirths	26.2478
chorio	22.1862
pcare	17.2850
locate	16.5322
mhypertens	14.8870
amagsulf	4.5313

Table 4.2: Feature importances calculated by the Decision Tree algorithm.

The results show that there are at least two groups of features based on the ranking: the first five have a strong impact in the predictive process, and are useful in discriminating the majority of the observations. While the remaining features seem to have less influence, they might be crucial to discriminate between examples in the lower levels of the tree. As we talked earlier, Decision Trees are not a very powerful machine learning algorithm: later in this chapter, we will see what is the feature importance as calculated by more sophisticated algorithms such as Random Forests, which also allow ranking of the features, but averaged on a much larger set of classifiers.

4.2 Results of the unsupervised analysis

4.2.1 SOM prototypes

The composition of the SOM prototypes is reported in Table **4.3**. The data underwent a postprocessing phase after the training, where binary features were rounded

to 0 or 1 to be meaningful. The same preprocessing was applied to `nbirths`, while the rest of the features were left unrounded, since having decimals values instead of whole decimal did not cause issues.

If we confront the results with Table **3.3** we can immediately see that the prototypes reflect the actual distribution of the data with sufficient precision. The only mentionable exception is `nbirths`, where the SOM failed to assign extreme values (number of births > 2) to its prototypes, due to the low frequency with which they appear on the original data. The final quantization error of the SOM was 0.4178.

Name	Mean	St. Dev.	Min.	1st Perc.	Median	3rd Perc.	Max.
bwgt	1099.23	293.92	362.00	864.38	1136.66	1366.85	2205.00
gadays	205.97	18.85	163.85	191.74	205.68	219.63	268.43
bheadcir	26.28	2.62	18.90	24.46	26.66	28.16	39.25
locate	0.06	0.24	0.00	0.00	0.00	0.00	1.00
pcare	0.91	0.29	0.00	1.00	1.00	1.00	1.00
aster	0.82	0.38	0.00	1.00	1.00	1.00	1.00
amagsulf	0.06	0.25	0.00	0.00	0.00	0.00	1.00
nbirths	0.35	0.48	0.00	0.00	0.00	1.00	2.00
chorio	0.12	0.32	0.00	0.00	0.00	0.00	1.00
mhypertens	0.24	0.43	0.00	0.00	0.00	0.00	1.00
vagdel	0.17	0.38	0.00	0.00	0.00	0.00	1.00
sex	0.49	0.50	0.00	0.00	0.00	1.00	1.00
ap1	5.84	2.20	0.00	4.50	6.25	7.60	10.00
ap5	7.75	1.61	0.00	7.00	8.00	9.00	10.00

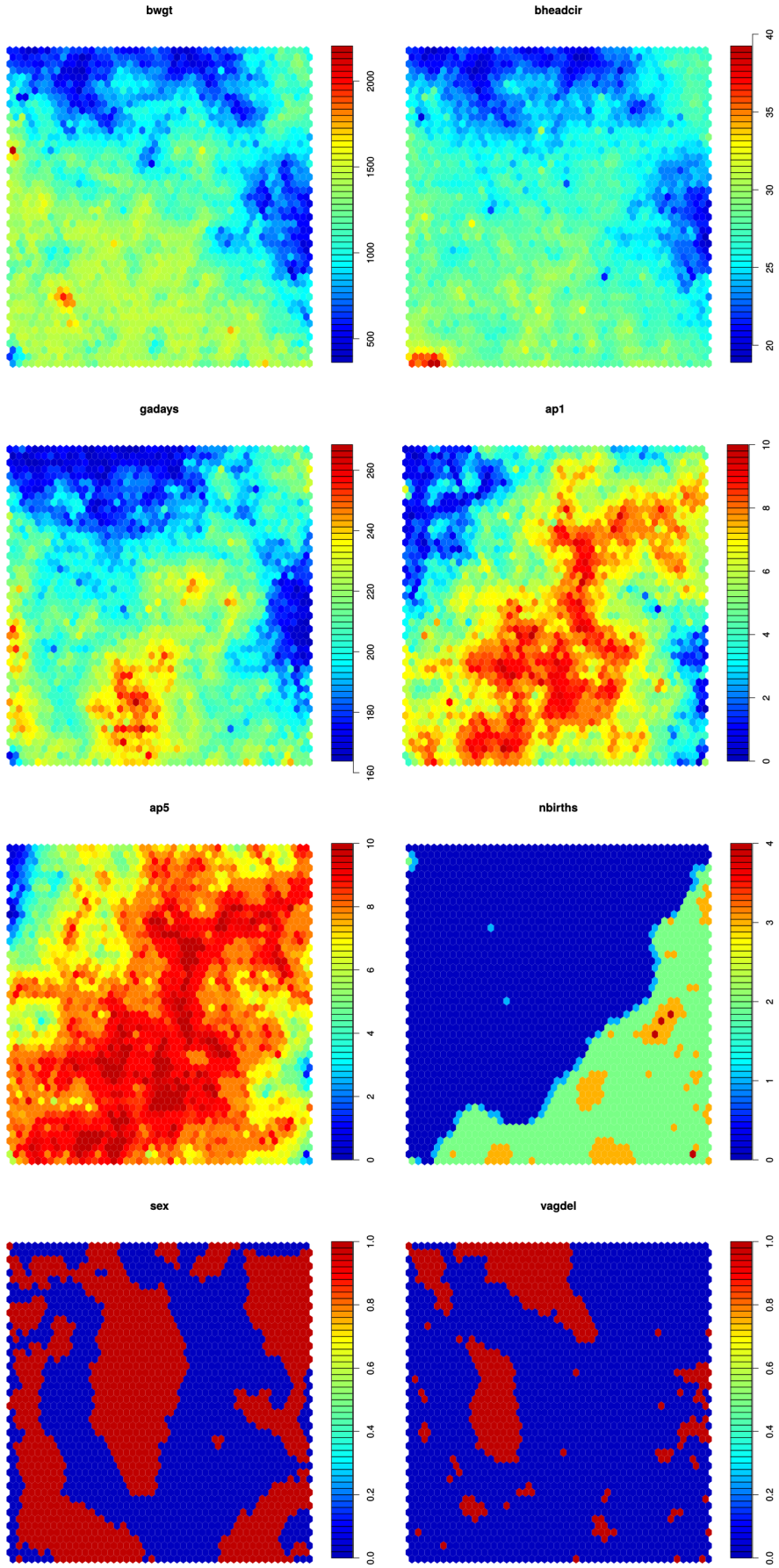
Table 4.3: Statistics of the SOM prototypes.

4.2.2 SOM visualizations

The first useful visualization obtained from the SOM were the component planes of each feature (see Section **2.7.2**), displayed in Fig. **4.7**. Looking at the first plane, one can easily see how the SOM grouped prototypes with low birth-weight into two clusters, one going from the upper left corner to the middle of the upper side, and another in the middle of the right side. The same pattern is repeated for `gadays`, `bheadcir`, `ap1` and `ap5`, indicating some sort of correlation. This was somehow expected, since for example a newborn with a longer gestational age is supposed to be more developed and thus have a higher birth-weight and larger head circumference. However, the fact that the SOM created two separate clusters out of a single correlation pattern is interesting.

Other facts that can be derived from the analysis of the component planes are:

- cesarean delivery and a diagnosis of chorioamionitis seem to be correlated with a lower gestational age;
- a diagnosis of maternal hypertension seems to be correlated with a moderately high birth-weight/gestational age;


bwgt
500
1000
1500
2000
bheadcir
20
25
30
35
40
gadays
160
180
200
220
240
260
ap1
0
2
4
6
8
10
ap5
0
2
4
6
8
10
nbirths
0
1
2
3
4
sex
0.0
0.2
0.4
0.6
0.8
1.0
vagdel
0.0
0.2
0.4
0.6
0.8
1.0

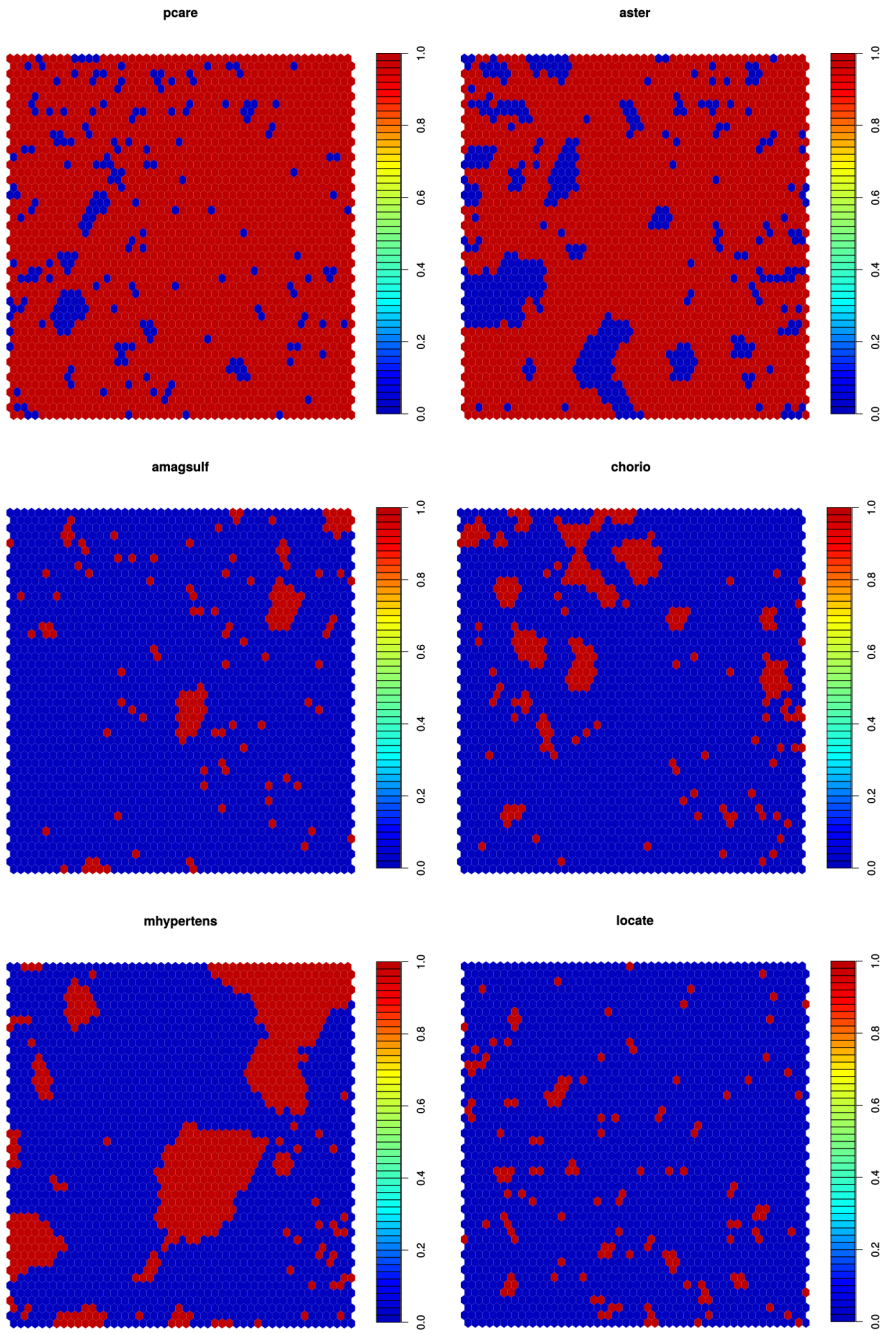


Figure 4.7: Component planes of the SOM. Each plot represents a single feature in the LBWI-G dataset.

- the SOM successfully grouped the number of births and the sex of the newborn into compact clusters (if the SOM training was poorly done, we would expect them scattered around the map in a random way);
- there is a small percentage of newborns that have a large head circumference but low birth-weight and gestational age (see the lower left corner of the `bheadcir` plane and compare with `bwgt` and `gadays` planes). This seems

somehow counter-intuitive and could constitute a cluster of newborns whose mortality prediction is difficult to be "guessed right" by a learning algorithm, since their strongest predictors present unconventional values.

The second useful visualization was to plot the label information against the constructed map to see if newborns that are likely to die are grouped contiguously on the map. To do so, we firstly mapped each record in the dataset to its closest prototype, and then counted how many records had a positive label among the total of records mapped for each neuron.

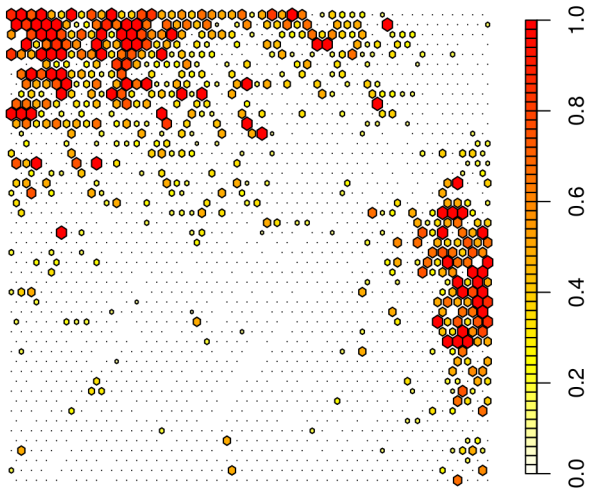


Figure 4.8: Plot of the label information onto the SOM.

In other terms, with this visualization we plotted the probability of a newborn dying given his/her assignment to a particular neuron. We once again want to stretch the point that all this information is derived by using the SOM without any supervised help. The map is displayed in Fig. **4.8**: the size of each cell in the map expresses the probability of dying. As is clearly visible, the map clustered newborns who are likely to die in two distinct clusters, which appear in the same position as the low birth-weight, low gestational age areas that were identified in their respective component planes.

To understand why the SOM created these two different clusters, we performed a statistical analysis to characterize each cluster, in hope to obtain two different patient "profiles" that are likely to develop a higher mortality risk.

The criteria for being included in one of the two clusters was, besides topographic vicinity, the fact that at least 30% of the newborns assigned to the cell actually died. The two clusters accounted for 1231 records combined, constituting the 12.78% of the total records in the dataset. The first cluster (the one starting from the left corner of the map) contained 832 records, the second (the one in the middle of the right side) 399. Tables Table **4.4** and Table **4.5** contain the collected statistics for the two clusters.

Name	Mean	St. Dev.	Min.	1st Perc.	Median	3rd Perc.	Max.
bwgt	690.10	232.05	295.00	550.00	650.00	760.00	2450.00
gadays	177.90	12.53	161.00	168.00	175.00	186.00	262.00
bheadcir	22.39	1.98	18.00	21.00	22.00	23.50	32.10
locate	0.06	0.24	0.00	0.00	0.00	0.00	1.00
pcare	0.87	0.34	0.00	1.00	1.00	1.00	1.00
aster	0.74	0.44	0.00	0.00	1.00	1.00	1.00
amagsulf	0.11	0.32	0.00	0.00	0.00	0.00	1.00
nbirths	0.03	0.23	0.00	0.00	0.00	0.00	2.00
chorio	0.32	0.47	0.00	0.00	0.00	1.00	1.00
mhypertens	0.21	0.41	0.00	0.00	0.00	0.00	1.00
vagdel	0.50	0.50	0.00	0.00	1.00	1.00	1.00
sex	0.58	0.49	0.00	0.00	1.00	1.00	1.00
ap1	2.87	1.79	0.00	1.00	3.00	4.00	8.00
ap5	5.45	2.20	0.00	4.00	6.00	7.00	10.00
died	0.61	0.49	0.00	0.00	1.00	1.00	1.00

Table 4.4: Statistics of cluster 1 (on the left in Fig. **4.8**).

The analysis of the statistics show that the two clusters have similar values with respect to birth-weight, gestational age, head circumference and mortality, which indicates that the distribution of died newborns in the two cluster is homogeneous.

However, the second cluster has a vast majority of babies coming from multiple gestations, while the first contains prevalently lone babies. We also see that infants from cluster 1 have lower Apgar average scores, are more subjected to cesarean delivery (50% against 30%), have double the incidence of maternal hypertension (20% against 10%) than infants in cluster 2. We also note that females are most present than males in both clusters.

If we compare the population belonging to both clusters against the whole dataset, we also see that infants ending up in the clusters have a `chorio` mean tree times higher than the dataset mean, and a `vagdel` mean about two, two and a half times

Name	Mean	St. Dev.	Min.	1st Perc.	Median	3rd Perc.	Max.
bwgt	686.70	157.20	330.00	580.00	661.00	790.00	1270.00
gadays	178.70	11.40	161.00	170.00	177.00	185.00	214.00
bheadcir	22.68	1.79	18.00	21.50	22.90	24.00	28.00
locate	0.04	0.20	0.00	0.00	0.00	0.00	1.00
pcare	0.92	0.27	0.00	1.00	1.00	1.00	1.00
aster	0.83	0.38	0.00	1.00	1.00	1.00	1.00
amagsulf	0.07	0.25	0.00	0.00	0.00	0.00	1.00
nbirths	2.19	0.42	1.00	2.00	2.00	2.00	4.00
chorio	0.30	0.46	0.00	0.00	0.00	1.00	1.00
mhypertens	0.10	0.30	0.00	0.00	0.00	0.00	1.00
vagdel	0.30	0.46	0.00	0.00	0.00	1.00	1.00
sex	0.57	0.50	0.00	0.00	1.00	1.00	1.00
ap1	3.79	2.05	0.00	2.00	4.00	5.00	9.00
ap5	6.51	1.66	1.00	6.00	7.00	8.00	10.00
died	0.62	0.49	0.00	0.00	1.00	1.00	1.00

Table 4.5: Statistics of cluster 2 (on the right in Fig. **4.8**).

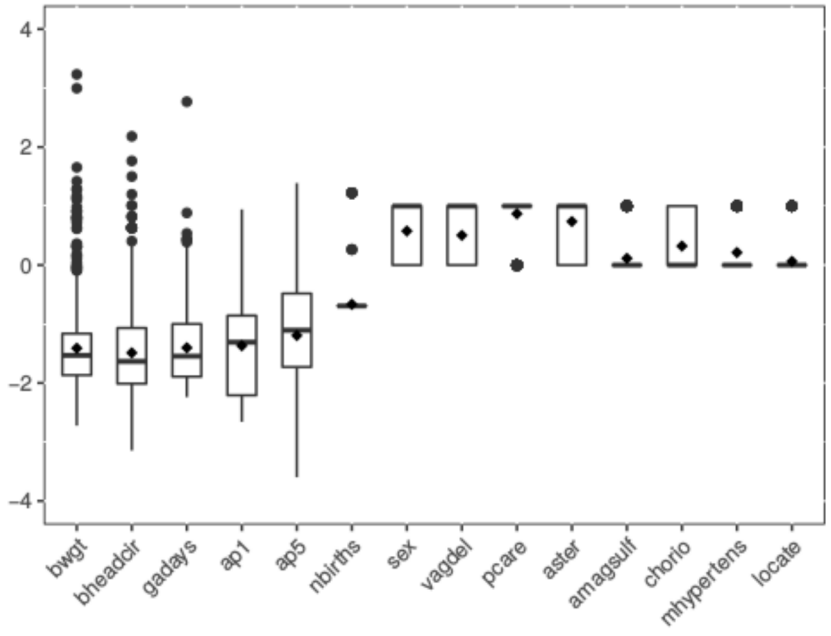


Figure 4.9: Box plots of cluster 1.

higher than its correspondent dataset mean. The distribution of the features was plotted with the help of box plots in Fig. **4.9** and Fig. **4.10**.

Another visualization technique that proved to be useful is the color coding of the map. We assigned a color (ranging from green to red) to the prototypes in the map, in a way that similar prototypes (according to the euclidean distance) get similar colors. The result of such coloring is shown in Fig. **4.11**.

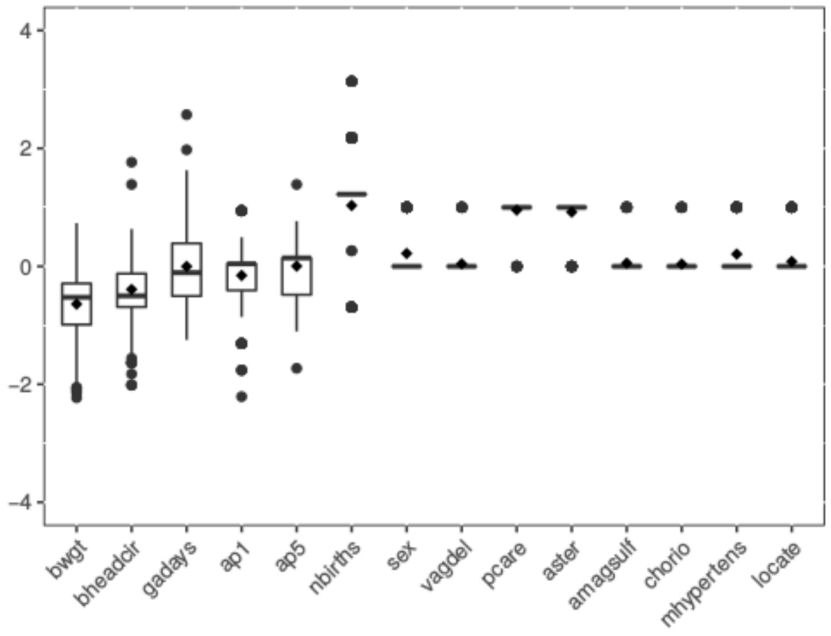


Figure 4.10: Box plots of cluster 2.

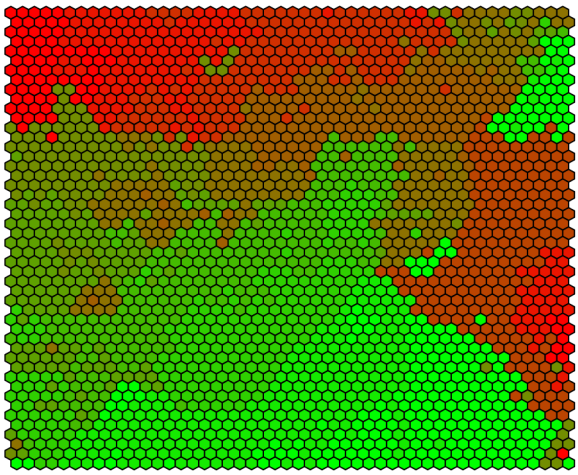


Figure 4.11: Color code of the SOM.

We see that, without any a-priori information about the labels, the coloring successfully identifies the two zones where the mortality is concentrated as similar. This map provides other useful keys of interpretation: for example, if we cross the results of this map with the one with the label information, we can say that newborns with low mortality chance correspond to the green-ish zones, while newborns with high mortality chance correspond to red-ish zones. The orange zones might be interesting for further investigation, because they mark the border between the two extremes, and are likely to represent newborns that are difficult to predict by a learning algorithm.

4.3 Results of the supervised analysis

In this section, we present the results of the training and testing of the pool of supervised models. For each dataset of reference, we will show the results of the 3×5 double cross-validation results, the results of the 10-fold cross-validation of the final model selection, the out-of-sample performance obtained on the test datasets, the estimation of mortality that the models provided, the statistical hypothesis testing of the differences in AUC-ROC we obtain. We will accompany this exposition with a discussion, suitable to highlight what was found in each of the dataset and in comparison between the two.

4.3.1 LBWI-V dataset

Double cross-validation results

The results of the double cross-validation procedure obtained on the LBWI-V dataset are reported in Table **4.6**. For each learning algorithm, the mean AUC-ROC of the three best models obtained through double cross-validation is shown.

VON-RA	LR	KNN	SVM	RF	GB	NN
0.8755 ±0.004	0.8795 ±0.004	0.8789 ±0.002	0.8808 ±0.003	0.8814 ±0.002	**0.8830** ±0.002	0.8822 ±0.008

Table 4.6: Double cross-validation AUC-ROC scores (LBWI-V dataset). The highest performance is displayed in bold.

As the results show clearly, we expect all the models of the pool to perform better than the VON-RA model in subsequent phases. We also expect the performance on the final model selection for each learning algorithm to lie approximately in the range of model performances yielded by the double cross-validation procedure, safe for some statistical variability induced by the largest samples employed in subsequent experiments. This is better understandable by looking at Fig. **4.12**. With respect to the VON-RA performance, the smallest increase in AUC-ROC comes from k-Nearest Neighbor (KNN in the table) with 0.0034, followed by Logistic Regression (LR) with 0.0040, Support Vector Machines (SVM) with 0.0052, Random

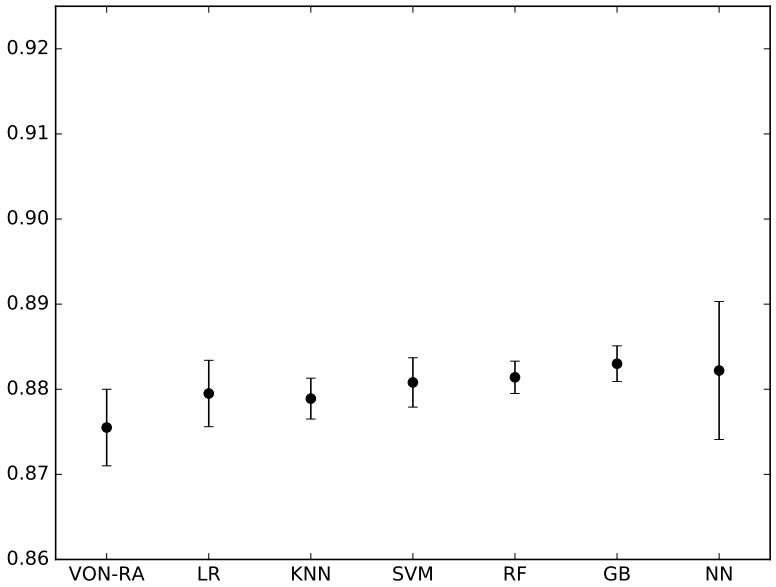


Figure 4.12: Plot of the double cross-validation AUC-ROC scores (LBWI-V dataset). The bar indicates the range of performances obtained by the three best models selected in the inner loop.

Forests (RF) with 0.0059, Neural Networks (NN) with 0.0067. Gradient Boosting (GB) models achieved the best score with 0.0074.

Cross-validation results

The next experiment consisted in selecting a final model with 10-fold cross-validation, choosing among 100 configurations of hyper-parameters. The means and standard deviations of the AUC-ROC obtained by the selected models across the folds are reported in Table **4.7** and shown graphically in Fig. **4.13**.

VON-RA	LR	KNN	SVM	RF	GB	NN
0.8754 ±0.015	0.8793 ±0.014	0.8800 ±0.015	0.8825 ±0.014	0.8820 ±0.014	**0.8837** ±0.015	0.8835 ±0.012

Table 4.7: Cross-validation AUC-ROC scores (LBWI-V dataset).

We can see that the cross-validation AUC-ROC scores confirm the trend expressed by the double-cross-validation phase. Again, all the models performed better than VON-RA. The best increase was registered by Neural Networks, which outperformed VON-RA by 0.0083, followed by Gradient Boosting (0.0076), Random Forests (0.0066), Support Vector Machines (0.0062), k-Nearest Neighbors (0.0039) and Logistic Regression (0.0038).

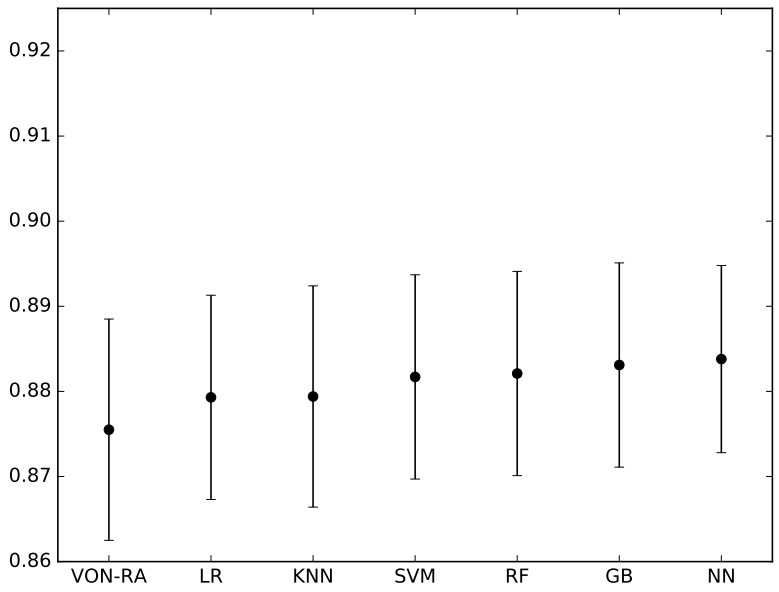


Figure 4.13: Plot of the cross-validation AUC-ROC scores (LBWI-V dataset). The bar indicates the standard deviation of the mean score.

Out-of-sample results

The last experiment on the LBWI-V dataset was to train the final model with the best hyper-parameters found in the previous cross-validation phase, using all the training data at our disposal, and test it with the 2015 dataset to see the behavior of the obtained models out-of-sample.

VON-RA	LR	KNN	SVM	RF	GB	NN
0.8821	0.8869	0.8871	0.8895	0.8896	**0.8906**	**0.8906**

Table 4.8: Out-of-sample AUC-ROC results (LBWI-V test dataset).

The results are reported numerically in Table **4.8** and graphically in Fig. **4.14**. From the analysis of the out-of-sample performances, it is noticeable that the same increasing trend appears (with approximately the same magnitude) if we compare the AUC-ROC of our pool of models to the one obtained by the VON-RA. Logistic Regression accounts for an increase of 0.0048, k-Nearest Neighbors for 0.0049, Support Vector Machines for 0.0074, Random Forests for 0.0075; Neural Networks and Gradient Boosting Machines scored the best results, with an increase in performance of 0.0085.

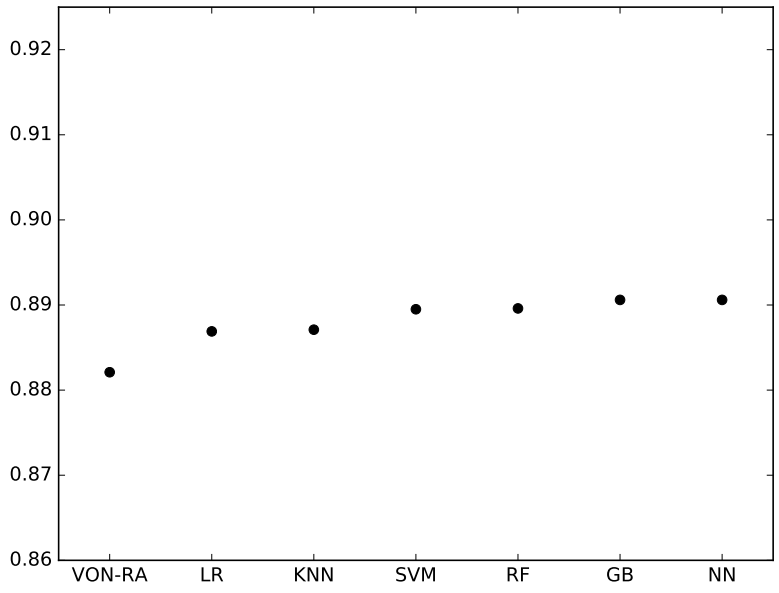


Figure 4.14: Plot of the out-of-sample AUC-ROC results (LBWI-V test dataset).

Mortality estimation

We report in Table **4.9** the mortality as estimated by each learning algorithms, in comparison with the true value that can be obtained from the data. We observe that all the models, including VON-RA, overestimate the true mortality. We register a better adherence to the true mortality rate of every model in the pool with respect to the VON-RA (whose difference in the estimation accounts for 0.0029): the model that shows better adherence to the true mortality is Logistic Regression (0.0065 difference), followed by Random Forests (0.0066), Neural Network (0.0067), Gradient Boosting Machines and Support Vector Machines (0.0072), and finally k-Nearest Neighbor (0.0101).

Observed	VON-RA	LR	KNN	SVM	RF	GB	NN
0.1167	0.0877	0.1103	0.1066	0.1095	0.1101	0.1096	0.1101

Table 4.9: Out-of-sample observed and estimated mortality (LBWI-V test dataset).

4.3.2 Discussion

The results clearly show an improvement in AUC-ROC of all the models in our pool with respect to the VON-RA performance, in all experiments. In Fig. **4.15**, we can see how the double cross-validation correctly estimated the subsequent cross-validation score obtained by the models, even if in the latter phase the models used more data for training (which could cause a deviation in the results). We consider this datum evidence of the fact that the model selection phase was correctly setup and we did not overfit during the model selection phase.

Furthermore, Fig. **4.16** shows the excellent adherence of the final cross-validation phase with respect to the out-of-sample performance: we see that in every case, the out-of-sample performance falls inside of the interval of variability estimated by the cross-validation procedure. With this particular test set, we report that the out-of-sample performance was above the mean cross-validation score; this is true even for the VON-RA model. This might indicate a particularly "favorable" test set, meaning that it is probably constituted of patients that are more easily classifiable by the learning algorithms with respect to the training dataset. Such optimistic performance was however correctly foresaw by the cross-validation phase, and has to be considered in the context of a normal statistical variability that might occur in the data.

To understand which of the differences in AUC-ROC between the models are significant, we run the DeLong's test for correlated ROC curves at the 0.05 level of significance for each pair of models, and reported its results in Table **4.10**.

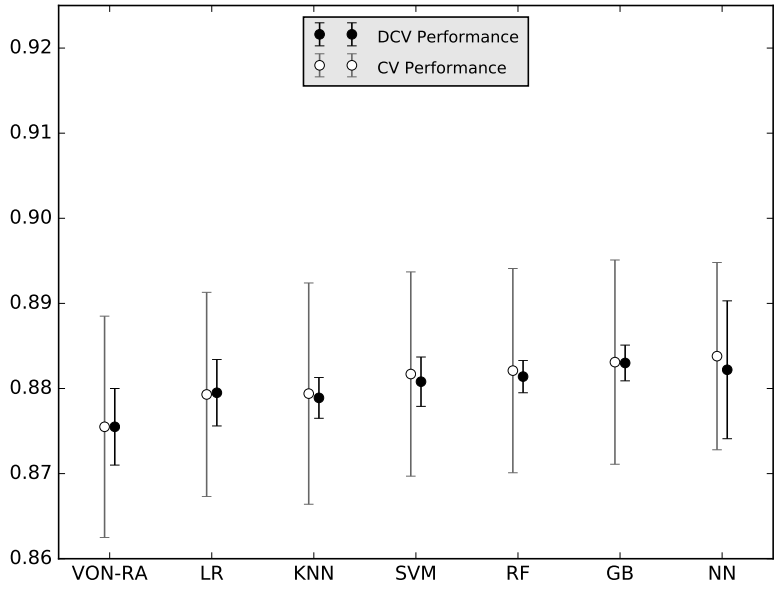


Figure 4.15: Plot of the double cross-validation AUC-ROC range against the cross-validation score (LBWI-V dataset). The grey bar indicates the standard deviation of the cross-validation score, while the black bar expresses the mean performance obtained by the three best models picked during the double cross-validation phase.

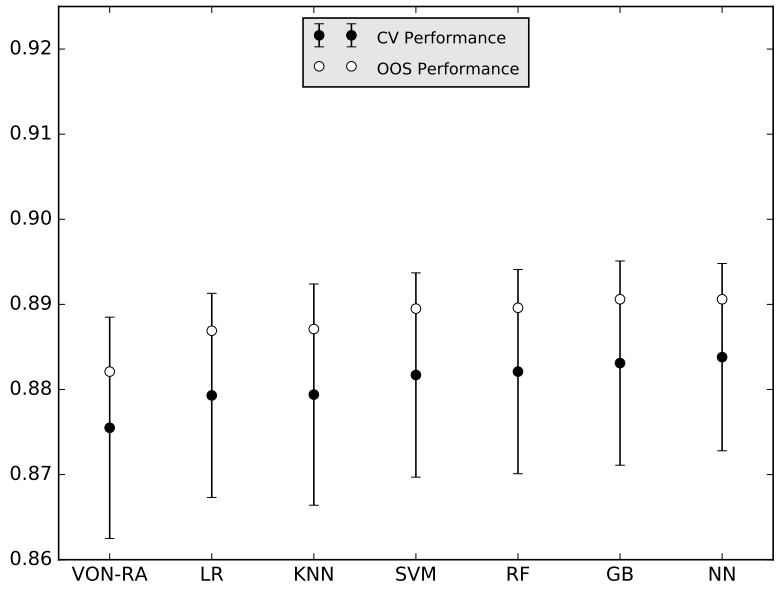


Figure 4.16: Plot of the cross-validation AUC-ROC estimation against the out-of-sample performance (LBWI-V dataset).

The table shows that we reached significance comparing the differences of our models and the VON-RA, with the only exception of k-Nearest Neighbors. Furthermore, the Support Vector Machines learning algorithm reached significance of the results even in comparison with the Logistic Regression model. The statistical hypothesis testing provides further statistical evidence that we indeed produced models that can perform significantly better than the VON-RA using its same set of features.

	LR	KNN	SVM	RF	GB	NN
VON-RA	0.0205	0.1640	0.0114	0.0261	0.0193	0.0073
LR		0.9634	0.1217	0.2610	0.0798	0.0996
KNN			0.3405	0.1825	0.1225	0.1275
SVM				0.9371	0.4621	0.5318
RF					0.4333	0.5532
GB						0.9560

Table 4.10: p-Values of DeLong's AUC-ROC test of significance between correlated ROC curves (LBWI-V test dataset). Cells highlighted in gray represent comparisons where the difference in AUC-ROC was significant.

Table **4.11** reports the improvement in out-of-sample AUC-ROC obtained by each of our trained models. We can see that models that are able to represent more complex hypothesis spaces (such as SVMs, ensemble methods and Neural Networks) achieve an AUC-ROC score that is on average approximately 65% better than the one obtained by models which use more simple hypothesis spaces (such as Logistic Regression). k-Nearest Neighbors is a model that somehow stays in between linear and complex models, and this fact is reflected by its improvement in performance with respect to VON-RA, which is slightly higher than Logistic Regression but not up to the rest of the machine learning algorithms considered.

LR	KNN	SVM	RF	GB	NN
0.54%	0.56%	0.83%	0.85%	0.96%	0.96%

Table 4.11: Improvement (in percentage) in out-of-sample AUC-ROC with respect to the VON-RA (LBWI-V test dataset).

As far as mortality predictions are concerned, we note that every one of the models in the pool produced a more closer estimation of the true mortality in the out-of-sample data with respect to VON-RA. Such improvement is shown in percentage in Table **4.12**.

LR	KNN	SVM	RF	GB	NN
25.68%	21.53%	24.84%	25.50%	24.88%	25.49%

Table 4.12: Improvement (in percentage) of the out-of-sample mortality estimation with respect to the VON-RA (LBWI-V test dataset).

As an aside, we provide in Table **4.13** the calculated F1 scores that the final models obtained on the out-of-sample data. Note that, since the model selection was intended to maximize the AUC-ROC criteria and not the F1, these scores are not the best possible and can be improved with a dedicated model selection. The increases of performance obtained were 0.0070 by k-Nearest Neighbor, 0.0107 by Support Vector Machines, 0.0113 by Logistic Regression, 0.0120 by Gradient Boosting Machines, 0.0123 by Neural Networks (0.0123) and finally 0.0133 by Random Forests. While these results are to be considered partial because no proper optimization was performed, we can once again see approximately the same increasing trend, with all the models that tend to perform better than the VON-RA.

VON-RA	LR	KNN	SVM	RF	GB	NN
0.8839	0.8953	0.8910	0.8946	**0.8972**	0.8959	0.8962

Table 4.13: Out-of-sample F1 scores (LBWI-V test dataset).

4.3.3 LBWI-G dataset

Double cross-validation results

The results of the double cross-validation procedure on the LBWI-G dataset are reported in Table **4.14**, and represented graphically in Fig. **4.17**, which shows a clear increasing trend. Logistic Regression outperformed the VON-RA model by 0.0072, followed by k-Nearest Neighbors (0.0117 increase), Gradient Boosting Machines (0.0148 increase), Support Vector Machines (0.0151 increase), Neural Networks (0.0160 increase), and finally Random Forests (0.0162 increase). The first noticeable difference with these results and the ones obtained with the double cross-validation in the LBWI-V dataset is that both the VON-RA and the models in the pool obtained lower AUC-ROC scores. However, the difference between the VON-RA and all the other models persists and appears amplified.

VON-RA	LR	KNN	SVM	RF	GB	NN
0.8885 $\pm$0.007	0.8957 $\pm$0.002	0.9002 $\pm$0.006	0.9036 $\pm$0.002	**0.9047** $\pm$0.008	0.9033 $\pm$0.001	0.9045 $\pm$0.007

Table 4.14: Double cross-validation AUC-ROC scores (LBWI-G dataset).

Cross-validation results

We report in Table **4.15** the results of the model selection with 10-fold cross-validation. Fig. **4.18** shows that the final models obtained AUC-ROC scores that strongly resemble what was foresaw by the double cross-validation phase.

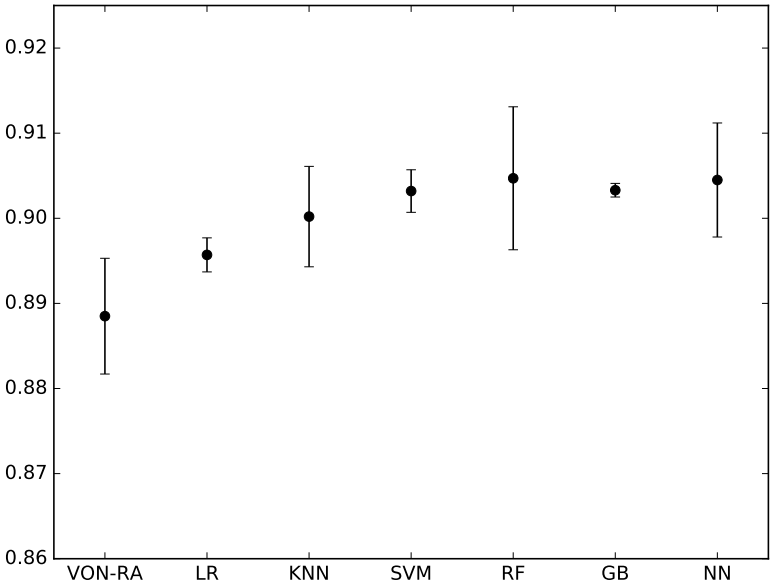


Figure 4.17: Plot of the double cross-validation AUC-ROC scores (LBWI-G dataset).

We observe a similar increasing trend as the one observed in the double cross-validation phase, with Logistic Regression improving the AUC-ROC by 0.0079 with

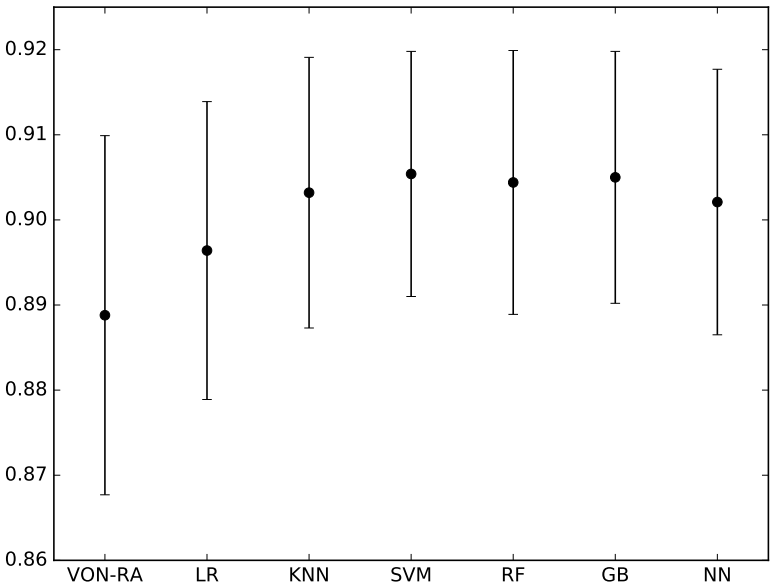


Figure 4.18: Plot of the cross-validation AUC-ROC scores (LBWI-G dataset).

respect to VON-RA, followed by Neural Networks (0.0133 increase), k-Nearest Neighbors (0.0145 increase), Random Forests (0.157 increase), Gradient Boosting Machines (0.0162) and finally Support Vector Machines (0.0166).

VON-RA	LR	KNN	SVM	RF	GB	NN
0.8888 ±0.021	0.8964 ±0.017	0.9032 ±0.016	**0.9054** ±0.014	0.9044 ±0.015	0.9050 ±0.015	0.9021 ±0.016

Table 4.15: Cross-validation AUC-ROC scores (LBWI-G dataset).

Out-of-sample results

As with the LBWI-V dataset, we trained the models selected in the previous phase using all the training data to construct the final models to be tested with the out-of-sample dataset.

VON-RA	LR	KNN	SVM	RF	GB	NN
0.8735	0.8822	0.8901	0.8927	0.8926	0.8931	**0.8939**

Table 4.16: Out-of-sample AUC-ROC results (LBWI-G test dataset).

Fig. **4.19** shows the results graphically. Logistic Regression outperformed the VON-RA model by 0.0077, k-Nearest Neighbor (0.0166 increase), Random Forests (0.191 increase), Support Vector Machines (0.193 increase), Gradient Boosting (0.196 increase) and Neural Networks (0.0204 increase).

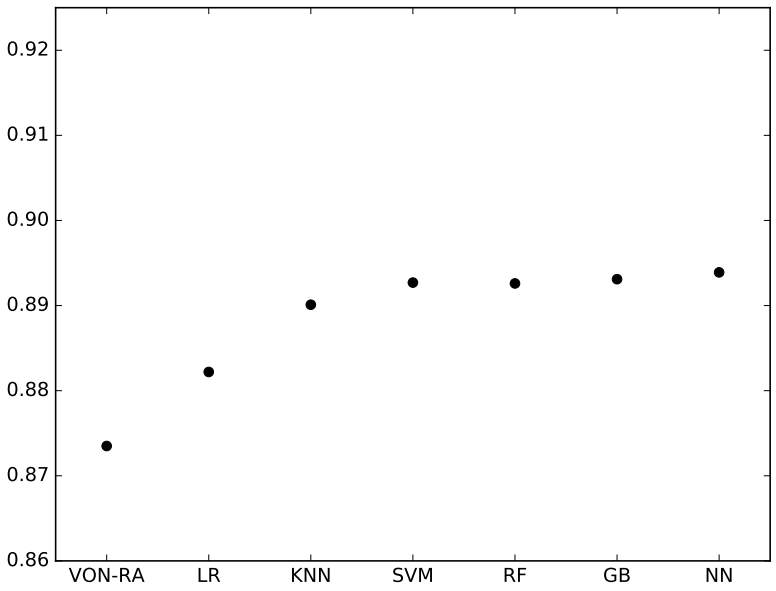


Figure 4.19: Plot of the out-of-sample AUC-ROC results (LBWI-G test dataset).

Confronting the out-of-sample results shown in Table **4.16** with the ones obtained on the LBWI-V (Table **4.8**, we notice that the out-of-sample AUC-ROC of the VON-RA (-0.0086) and the Logistic Regression (-0.0044) is less than what was achieved previously, suggesting that this set of features might not be suited for linear models. On the contrary, all the remaining models achieved consistently better scores than what was registered with the LBWI-V dataset: Gradient Boosting Machines increases its performance by 0.0025 with respect to the LBWI-V, k-Nearest Neighbors and Random Forests by 0.0030, Support Vector Machines by 0.0032, Neural Network by 0.0033.

Mortality estimation

Table **4.17** reports the estimated mortality of each tested model. Very similarly to what observed on the LBWI-V dataset, both the VON-RA and our models seem to underestimate the true mortality. However, like in the LBWI-G setting, the estimates of models coming from the pool of models remain more precise.

Observed	VON-RA	LR	KNN	SVM	RF	GB	NN
0.1014	0.0805	0.0967	0.0915	0.0972	0.0971	0.0964	0.0962

Table 4.17: Out-of-sample observed and estimated mortality (LBWI-G test dataset).

Once again, our models provide in general better estimates than the VON-RA (which registers a 0.209 difference from the true value): the model that best approximated the mortality was the Support Vector Machine (0.0042 difference), followed by Random Forests (0.0043 difference), Logistic Regression (0.0047 difference), Gradient Boosting Machines (0.0054 difference), Neural Networks (0.0056 difference) and Nearest Neighbor (0.0099 difference). The gap between the VON-RA and the observed mortality is 0.0209.

4.3.4 Discussion

Fig. **4.20** shows again that the double cross-validation step provided once again a correct estimation of the behavior of the models in the subsequent model selection phase. As was the case with the LBWI-V dataset, the cross-validation scores are estimated very tightly by the three models outputted by the double cross-validation phase, despite the use of a smaller amount of data.

Fig. **4.21** shows how the out-of-sample performance relates to the cross validation scores obtained in the final model selection phase. Once again, we report the successfulness of the cross-validation procedure in estimating the right range of values where the out-of-sample performance would fall. With respect to the LBWI-V

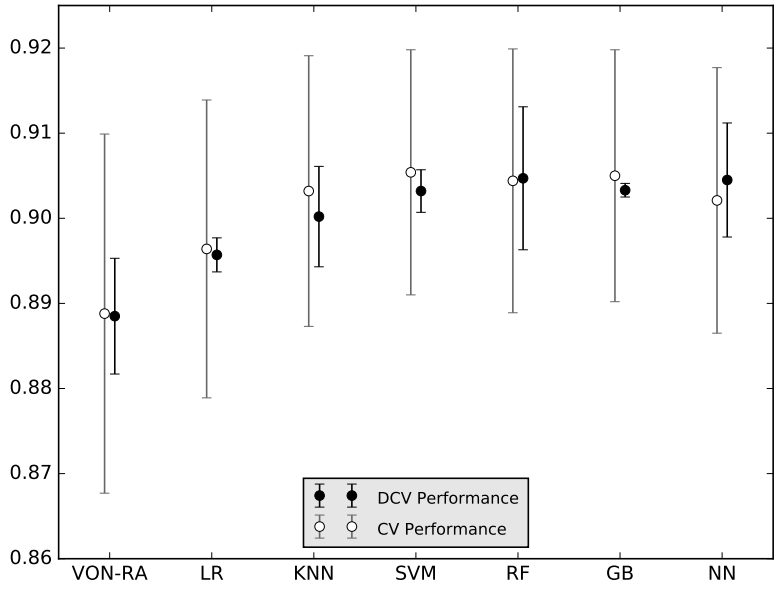


Figure 4.20: Plot of the double cross-validation AUC-ROC range against the cross-validation score (LBWI-G dataset).

dataset, we report that the estimates are this time a bit over-optimistic, but the out-of-sample performance is again inside the estimated variability obtained by

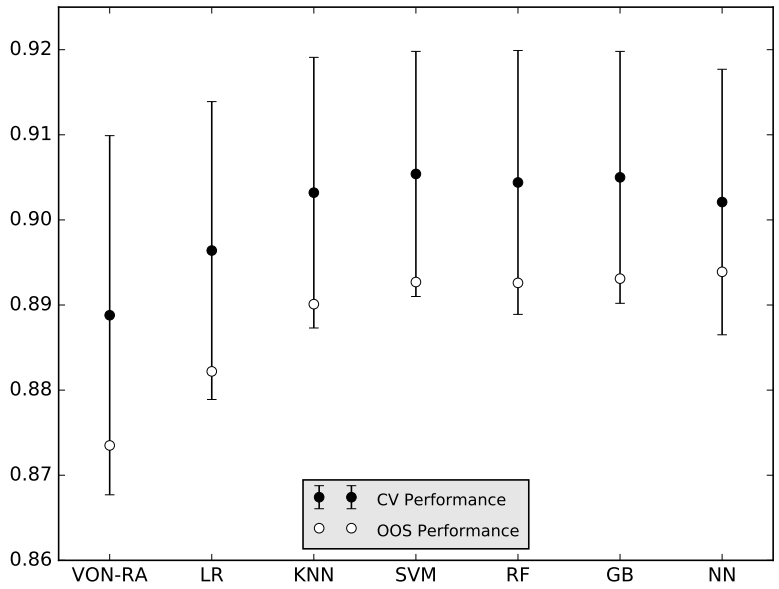


Figure 4.21: Plot of the cross-validation AUC-ROC estimation against the out-of-sample performance (LBWI-G dataset).

averaging the various cross-validation folds, so the results remain valid.

The results of the statistical hypothesis testing are shown in Table **4.18**. All the models achieved significant differences in AUC-ROC with respect to the VON-RA model, with the exception of Logistic Regression. Furthermore, all the "complex" models (Support Vector Machines, Random Forests, Gradient Boosting Machines and Neural Networks) have reached significance in comparison with the Logistic Regression models, and even k-Nearest Neighbors fell slightly above the significance level. This result emphasizes the fact that this dataset seem to be better suited for models that are able to represent complex hypothesis spaces, and not very favorable for linear models.

	LR	KNN	SVM	RF	GB	NN
VON-RA	0.1066	0.0013	< 0.0001	< 0.0001	< 0.0001	< 0.0001
LR		0.0573	0.0113	0.0294	0.0132	0.0020
KNN			0.2830	0.3608	0.2874	0.1195
SVM				0.9358	0.8291	0.5382
RF					0.7362	0.6324
GB						0.7633

Table 4.18: p-Values of DeLong's AUC-ROC test of significance between correlated ROC curves (LBWI-G test dataset).

Table **4.19** shows the improvement in AUC-ROC that was achieved out-of-sample by our models with respect to VON-RA. By looking at the table, it is evident that models that utilize complex hypothesis spaces provide a higher improvement with respect to linear models, same as what was observed in Table **4.11**. This time the gap seems to be even wider, with Random Forests, Support Machines, Gradient Boosting and Neural Networks that more than double the performance of the Logistic Regression. k- Nearest Neighbors also performs well, scoring an improvement that is almost twice the one obtained by the Logistic Regression.

LR	KNN	SVM	RF	GB	NN
1.00%	1.90%	2.21%	2.18%	2.25%	2.33%

Table 4.19: Improvement (in percentage) in out-of-sample AUC-ROC with respect to the VON-RA (LBWI-G test dataset).

With respect to mortality predictions, we report that every one of the models in the pool produced a better estimations of the true mortality in the out-of-sample data with respect to VON-RA. Such improvement is shown in percentage in Table **4.20**. With respect to Table **4.12**, the improvements are smaller; this however was somehow expected, since the gap between the VON-RA and the true mortality is narrower (0.0290 vs. 0.0209). We also remark that while the difference in estimation of k-Nearest Neighbors was substantially unchanged (0.0101 with the LBWI-V dataset, 0.0099 with the LBWI-G dataset), all the other models obtained closer estimations with this dataset in comparison with the LBWI-V.

LR	KNN	SVM	RF	GB	NN
20.08%	13.66%	20.75%	20.64%	19.68%	19.42%

Table 4.20: Improvement (in percentage) of the out-of-sample mortality estimation with respect to the VON-RA (LBWI-G test dataset).

Table **4.21** provides, the calculated F1 scores that the final models obtained on the out-of-sample data. Again, we remind that no proper optimization was performed to calculate the F1 metric, hence the results are not to be considerate definitive. We nonetheless note an increase in performance even in this case, with k-Nearest Neighbors scoring an increase of 0.0066, Logistic Regression of 0.0084, Random Forests of 0.109, Neural Networks of 0.0112, Support Vector Machines of 0.0146, and finally Gradient Boosting Machines with 0.0153.

VON-RA	LR	KNN	SVM	RF	GB	NN
0.8918	0.9002	0.8984	0.9064	0.9027	0.9071	0.9030

Table 4.21: Out-of-sample F1 scores (LBWI-G test dataset).

4.3.5 The big picture

So far, we have reported and discussed the performances of our pool of models with respect to the VON-RA on the two datasets that were produced for the study. In this section, we will focus on the comparison of results in both datasets. Fig. **4.22** shows the out-of-sample results we obtained on the two datasets together. While performing a statistically sound comparison between the two results is difficult, mainly because of the very wide difference in size between the two datasets, this picture offers some useful cues:

- we successfully managed to obtain models that perform significantly better than the VON-RA using both the datasets. We consider this result a strong one, since the VON-RA is a well-established model that has being perfected year after year, while our work is relatively young. Furthermore, it confirms that the rigorous application of model selection and validation techniques leads to state-of-the-art performances, even in a difficult task like this one has proven to be;
- the gap between the lowest and the highest scoring model constructed on the LBWI-V dataset is very narrow. Much higher increases in performances appear in the LBWI-G dataset, with the VON-RA and the Logistic Regression models scoring lower than what they did on the LBWI-V dataset. This suggests that the second set of features might is better exploited by models that are able to represent more complex hypothesis spaces, while not very beneficial for linear models;

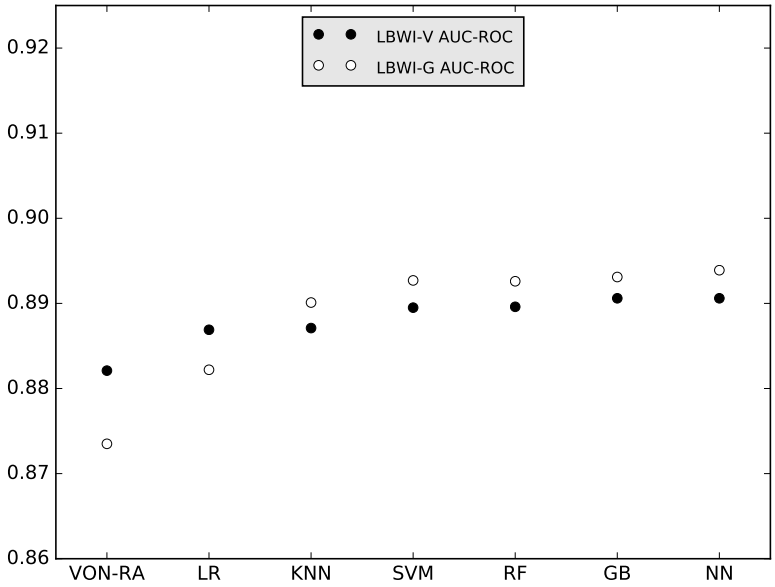


Figure 4.22: Comparison of out-of-sample AUC-ROC in the two test datasets (LBWI-V and LBWI-G).

- on the LBWI-G dataset, we used only about 30% of the available data, because of the huge number of missing values. Nonetheless, we managed to obtain both a better absolute AUC-ROC with respect to the LBWI-V dataset and higher increases in performance in general. All the learning algorithms, especially the ones that are able to represent complex hypothesis spaces, perform better with large quantities of data. This means that ultimately, if these models prove to be valid, the use of more data with the new set of features could be beneficial and perhaps lead to further improvements;
- every model we constructed and examined provides a better estimate of the mortality than the VON-RA. Even though we could not prove this result locally on actual NICUs (since the dataset we were provided was opportunely anonymized for privacy purposes), this result constitutes evidence that our models can lead to better risk-adjusted scores with respect to infant mortality, eventually resulting in better decisions concerning quality of health care across hospitals.

One last consideration regards the state of the task, and provides further justification to results that a short-sighted analysis could consider minimal. It seems evident that we are reaching the upper bound on the performances that one could obtain in this particular task. We cannot explain infant mortality completely just by looking at a very partial and short term history of the mother and some measurements on the newborn right after birth. Several factors that might happen in

the days following birth, which influence infant mortality at 28 days, are not represented by these features; others might still be unknown to the clinicians. The results indicate that we are moving towards closing the gap between the performance that more traditional predictive models can achieve and the limit imposed by both the nature of the problem and factors that remain unaccounted for. In light of this claim, even small improvements have to be considered meaningful.

Feature ranking

Stepping aside for one moment from the pure supervised objective, as anticipated in Section **4.1**, we wanted to obtain a more precise ranking of the features as calculated by more complex learning algorithms such as Random Forest. In particular, we focus on the ranking produced by the training after the final model selection phase, such that the comparison with the one produced by the Decision Tree (which was itself trained with all the available data) is as fair as possible. We will not focus on the numbers obtained, since the calculations come from two different libraries and are performed very differently: what is important is the order that is assigned to the features. The results are shown in Table **4.22**.

Feature	Mean Decrease in Impurity
bwgt	0.2661
gadays	0.2411
bheadcir	0.1594
ap1	0.1291
ap5	0.1213
chorio	0.0143
nbirths	0.0134
vagdel	0.0131
sex	0.0091
aster	0.0086
mhypertens	0.0085
locate	0.0069
pcare	0.0050
amagsulf	0.0039

Table 4.22: Feature importances calculated by the Random Forest algorithm.

The ranking seems to confirm some of the insights that emerged from previous analysis (with Decision Tree and SOM). We see that the first five features are the strongest predictors in the set, since they retain once again the majority of the discriminative power. Immediately after, though their contribution is smaller, we see three features that were indicated by the unsupervised analysis as potentially helpful: `chorio`, `nbirths` and `vagdel`, which are not included in the LBWI-V dataset and could thus be among the features responsible of the improvement we reached using the LBWI-G dataset. Another consistent result with respect to the Decision Tree analysis is that feature `amagsulf` is ranked last in both cases. This is somehow in contrast with what the medical intuition is on the role of magnesium sulfate in

LBWI mortality, and might constitute a reason to investigate in depth the relevance of this feature by clinicians.

4.3.6 Considerations prior to deployment

The models built and examined in this study were mainly thought for survey purposes, but there is nothing to prevent their deployment for producing risk-adjusted scores for NICUs. With this perspective in mind, however, we advice that two measures should necessarily be taken before going into production. These are:

- perform a more intensive model selection phase. During our work, our model selection algorithms generally employed 100 random samples from each learning algorithm hyper-parameter distributions. While this number is more than enough from an academic point of view, in a production setting there is the need to obtain the higher score possible before deployment. This implies that the space of hyper-parameters must be explored more thoroughly, perhaps even using grid-search, to find better and better configurations. Furthermore, for reproducibility reasons, we fixed a random number generator seed. In a production setting, however, reproducibility is not a concern, therefore the random seed value becomes itself a hyper-parameter;
- take care of the missing values. Before, we illustrated how our LBWI-G dataset was smaller in size than the LBWI-V, and that this was mainly due to missing values. In our study, we did not worry too much about this issue, since in the future those features will appear more frequently. When used in production, however, the models need to be used on as much of the data as possible to ensure that no valuable information is lost in the preprocessing.

With respect to the second point, we tested the performance of our models on an imputed version of the test dataset, where missing values were substituted respectively with the mean (for `bwgt`, `bheadcir`, `gadays`, `ap1`, `ap5`, `nbirths`) and the most frequent value (for the remaining features). With this imputation scheme, we saved an additional 277 observations, changing the size of the test data from 2792 to 3069 records, which dropped the percentage of missing data from almost 14% to an acceptable rate of 5%.

VON-RA	LR	KNN	SVM	RF	GB	NN
0.8755	0.8822	0.8897	0.8930	0.8929	0.8937	**0.8939**

Table 4.23: Out-of-sample AUC-ROC scores (LBWI-G test dataset with imputation).

Despite the imputation scheme was pretty simple, the results, shown in Table **4.23**, are encouraging. Indeed, we observe that the models appear to be robust to imputation, as they behave consistently with what observed in the out-of-sample test with the smaller dataset, and in some cases even account for a tiny improvement.

Chapter 5

Conclusions and further works

The objective of this study was the investigation of the problem of predicting mortality of low birth-weight infants from a machine learning perspective. We conducted thorough experiments using state-of-the-art machine learning models, which involved handling the data to make it usable by such models in order to express their full potential, as well as training and evaluating the models rigorously following consolidated model selection and validation practices. Furthermore, we tried to characterize the above-mentioned problem by exploring the data with a combination of supervised and unsupervised models, to provide field experts with possible additional knowledge that could result, in the future, in an even better understanding of the problem.

When we looked at the data that we were given for the first time, together with our clinicians support team we formulated a series of simple questions, specifically:

- Can we train one or more predictive learning models that have better predictive performance and that generalize their results better than the VON-RA model?
- Can learning algorithms that are able to represent more complex hypothesis spaces provide the sought-after performance increase?
- Is the new set of features capable of improving anticipated prediction of LBWI mortality risk?
- Can we exploit machine learning based exploratory analysis of the data to identify clinically-relevant patterns?

We let data answer to these questions, finding responses we believe to be impacting and that provide new insight into the characterization and assessment of LBWI mortality. In the following, we will sum up our work by discussing the answers that emerged from the data.

Better predictive models

The answer to the first question is yes: by using state-of-the-art machine learning models, dedicated data manipulation and specific model selection and evaluation techniques, we successfully managed to create better performing models than the VON-RA, in both the datasets that were examined in this study. In the LBWI-V dataset, the resulting improvement was of 0.94% for the best performing model; in the LBWI-G dataset, which is constituted by a set of features selected by our neonatologists support team, the improvement more than doubled, summing up to 2.36% for the best performing model. We consider these results positive in light of:

- the fact that the VON-RA model is a reliable and consolidated model that has been perfected by neonatologists for over 25 years;
- the fact that these results refer to out-of-sample data consisting of patients whose features were recorded in a completely unseen year with respect to the training data;
- the fact that the problem is probably approaching saturation, meaning that further significant improvements are less likely to be possible. With respect to this point, both improvements, which appear relatively narrow at first sight, characterize as a huge step ahead in performance;
- the fact that all the comparisons we performed between the VON-RA and our models through hypothesis testing have shown that the differences in our performance of choice (AUC-ROC) were significant, meaning that the improvements are justifiable not only on a numerical basis, but also from a statistical point of view.

Furthermore, all our models provide more accurate mortality estimations than the VON-RA, which means that these models are likely to be more accurate to compare NICUs. Therefore, their adoption in comparison studies could lead to improvements in the quality of health care of hospitals.

At last, one word of caution: we remind that these results are strictly dependent on the data at our disposal, which regards a restricted cohort composed only by Italian infants. Nonetheless, since we obtained strong results and proved to have constructed a valid approach to tackle this problem, we believe that our models and methodologies are sound independently of the size and variety of the data and demand to be tested on a worldwide scale.

The impact of complex models

When we started our work, we decided to select a pool of promising state-of-the-art learning algorithm to train on the task, setting up the model selection and evaluation framework in order to be independent on the particular model choice. Given this agnostic set-up, what we observed in both datasets is that models that can represent complex hypothesis spaces obtained systematically better performances than linear models such as Logistic Regression. This was less evident in the LBWI-V dataset, where it seems that the relationship among the features is only slightly non-linear and traditional models like Logistic Regression still achieve good performances. Another reason of the small improvement might be the fact that we are possibly reaching the upper bound of performances for this learning problem given that choice of features: in this context, even a narrow improvement is likely to be significant. On the other hand, in the LBWI-G dataset, models like Random Forests, Support Vector Machines, Neural Networks or Gradient Boosting Machines seem to better capture the actual relationships underlying the data. Our belief in this sense is that the new features add a degree of non-linearity, which can be fully captured and exploited with the use of complex and powerful machine learning models.

The impact of the new set of features

Our results show that models trained with the new set of features, that were suggested to us by field experts and constitute one of the key original contributions of this work, performed systematically better than the LBWI-V features, using less than a third of the data. We believe that this result represents a strong argument in favor of this new set of predictors. The main hindrance we had to cope with during our work was the high percentage of missing data. We are firmly convinced that with more data, we could have obtained even stronger results, since machine learning algorithms give their best when they are trained with large data collections. The results we have produced suggest that the contribution of these new predictors ultimately lead us to a reach an improvement at the task. Therefore, we invite the medical community to consider these highly missing features as much important as the other more traditional predictors, and strongly encourage their steady collection in neonatal protocols. We also think that the medical community should investigate further the clinical relationship between these features and the mortality of LBWI, since there might be hidden but more impacting information that need to be discovered, which could help reach further improvements.

A better characterization of the problem

In the unsupervised part of our research, we provided clues that the dataset has inherent structure that could be exploited. The SOM analysis showed us that infants coming from multiple gestations are "seen" differently by unsupervised learning algorithms, producing two clearly separated mortality clusters, which we believe need to be explored more thoroughly by clinicians to see if these two populations are characterized by differences in factors that lead to mortality. The various SOM visualization techniques and a feature ranking allow us to discover that a diagnosis of chorioamnionitis and a cesarean delivery seem to be, immediately after the more clinically obvious, the strongest mortality predictors, and might have some form of clinically relevant dependence with a higher mortality risk that could be worth to investigate in-depth. The analysis of the Decision Tree learning process suggests that Apgar scores might be features that "trick" the learning algorithm to produce misclassification errors. Our work is not to draw conclusions from these facts: we simply expose them to the medical community, hoping to drive the research in the field towards the study of factors that were previously perhaps unanalyzed or not taken into proper consideration.

5.1 Future work

This study opens up to a series of areas that need further investigation. A logical next step to our analysis is to incorporate the additional knowledge that was obtained in the unsupervised part of this work into the model building process, for example by converting the information that clinicians could derive from the unsupervised analysis into new and more predictive features. Another interesting follow-up is to include the unsupervised models themselves inside the modeling process: for example, one could use the SOM to preprocess the training data before feeding it to a predictive learning model, leveraging the clustering that this model produces to obtain better predictive performances.

The more intriguing possibility, however, comes with the availability of more data. Being able to test non-linear models with a huge amount of data from all over the world is what will ultimately prove that our approach is sound or maybe drive us towards an alternative and more prolific direction.

Appendix A

Pseudo-code for double cross-validation

This exemplifying code in pseudo-Python explains how double cross-validation was implemented for the purposes of the experiments presented in Chapter 4.

Listing A.1: Double cross-validation

```
N_OUTER_FOLDS = 3
N_INNER_FOLDS = 5
N_HYPERPARS_TRIALS = 100

def model_selection(algorithm, data, labels, hyperpars):
   pred = select_column_from(data, "pred")
   data = remove_column_from(data, "pred")
   died = select_column_from(data, "died")
   data = remove_column_from(data, "died")

   inner_folds = split(data, labels, N_INNER_FOLDS)

   test_scores = []

   for hpar in hyperpars:
      model = create_model(algorithm, hpar)
      cv_scores = []
      for [train_idx, test_idx] in inner_folds:
         data_train = subset(data, train_idx)
         data_test = subset(data, test_idx)

         labels_train = subset(labels, train_idx)
         labels_test = subset(labels, test_idx)

         model = train(model, data_train, labels_train)
         score = test(model, data_test, labels_test)
         append(cv_scores, score)

      append(test_scores, average(cv_scores))

   best_model_index = argmin(test_scores)
```

```
   best_model_hyperpars = hyperpars[best_model_index]

   best_model = create_model(algorithm, best_model_hyperpars)
   best_model = train(best_model, data, labels)

   return best_model


def double_cv(trainfile, algorithm):

   data = load_data(trainfile)
   pred = select_column_from(data, "pred")
   data = remove_column_from(data, "pred")
   died = select_column_from(data, "died")
   data = remove_column_from(data, "died")

   outer_folds = split(data, died, N_OUTER_FOLDS)
   for [train_idx, test_idx] in outer_folds:
      data_train = subset(data, train_idx)
      data_test = subset(data, test_idx)

      died_train = subset(data, train_idx)
      died_test = subset(data, test_idx)

      pred_train = subset(pred, train_idx)
      pred_test = subset(pred, test_idx)

      hyperpars = sample_configurations(algorithm, N_HYPERPARS_TRIALS)
      best_model = model_selection(algorithm, data_train, died_train,
          hyperpars)

      auc_test = auc_score(best_model, data_test, died_test)
      auc_vonra = auc_score(pred_test)

      compare(auc_test, auc_vonra)
```

Appendix B

Explanation of hyper-parameters values

As we mentioned in Section 3.8.3, we describe hereby the values of the hyper-parameters that were tuned during the final model selection phase. The type of feature scaling is a global hyper-parameter that affects all the learning algorithms. All the other parameters are intend to be specific to each algorithm. We also report that the random seed for each of our experiments was 123.

Feature scaling

- `scaler`: specifies the feature scaling method among min-max scaling, de-mean transformation, standardization or no scaling at all.

Logistic Regression

- `C`: inverse value of the regularization coefficient. Smaller values mean a more regularized model.
- `penalty`: type of regularization penalty. Besides the usual L2 penalty, there is also an L1 version, based on Manhattan distance;
- `warm_start`: enables or disables the *warm-start* optimization technique, which is basically a trick that allows the classifier to reuse previous results in order to speed the search for the regularization coefficient [11].

k-Nearest Neighbor

- `n_neighbors`: the number of examples constituting the neighborhood.
- `p`: power parameter for the Minkowski metric. When p = 1, this is equivalent to using Manhattan distance, and Euclidean distance for p = 2.

- `weights`: weight function used in prediction. If equal to `uniform`, all examples in the neighborhood are weighted equally; if equal to `distance`, the examples are weighted with the inverse of their distance to the example.

Support Vector Machines (RBF Kernel)

- `C`: inverse value of the regularization coefficient. Smaller values mean a more regularized model.
- `gamma`: a value that is used to scale the kernel matrix.

Random Forest

- `bootstrap`: whether to use bootstrap samples to construct the dataset each tree is given;
- `criterion`: the criterion function to determine the best split. Choices are Entropy or the Gini index;
- `max_depth`: maximum depth of each tree in the forest. This parameter acts as a regularizer: a deeper tree means less regularization;
- `max_features`: maximum number of features to consider when looking for the best split. Can be either `None` (all the features), `log2` (base 2 logarithm of the number of features), `sqrt` (square root of the number of features);
- `min_samples_split`: minimum number of examples required to split a node. This parameter acts as a regularizer: smaller values mean less regularization;
- `n_estimators`: number of trees composing the forest.

Gradient Boosting Trees

- `colsample_bytree`: subsample ratio of features when constructing each tree;
- `learning_rate`: value to weigh the contribution of each tree;
- `max_depth`: maximum depth of each tree in the forest;
- `min_samples_split`: minimum number of examples required to split a node. This parameter acts as a regularizer: smaller values mean less regularization;
- `n_estimators`: number of boosting stages to be performed;
- `gamma`: regularization parameter;

- `subsample`: subsample ratio of training examples when constructing each tree.

Neural Networks

- `activation`: activation function of the hidden layer;
- `batch_size`: mini-batch dimension (the choices are powers of 2 to facilitate the work of the GPU);
- `dropout`: dropout regularization parameter;
- `epochs`: maximum number of epochs to train the net. If this number is reached, the training is interrupted;
- `hidden_units`: number of units in the hidden layer;
- `init`: type of weights initialization routine. Can be sampled uniformly between 0.1 and -0.1, or normally with mean 0 and standard deviation 0.1. The Lecun initialization method is best described in [32];
- `learning_rate`: learning rate of the gradient descent optimizer;
- `momentum`: momentum is a fraction of the previous values of the weights that is applied to the update of the weights to allow faster learning. Such fraction of update is regulated by this parameter;
- `optimizer`: the optimized gradient descent routine to update the weights. Choices are *stochastic gradient descent*, *RMSprop* and *Adam*, which are descrived for example in [49].

Some of the neural network parameters were fixed beforehand to speed up the training process: the back propagation algorithm used mini-batches and the number of hidden layers was fixed to be 1. Note that in principle the latter assumption does not pose a particular issue, since as long as they have at least one hidden layer, neural networks are *universal approximators* [26]. In general, hyper-parameters that were set beforehand do not influence learning directly (meaning that the algorithm "learns" regardless of which hyper-parameter is set), but they do have an impact in the amount of time needed to complete the learning.

Appendix C

Final hyper-parameter values

LBWI-V dataset

Learning algorithm	Hyper-parameter	Value
Logistic Regression	C	0.077700
	penalty	l2
	warm_start	True
	scaler	demean
k-Nearest Neighbor	n_neighbors	194
	p	1
	weights	uniform
	scaler	std
Support Vector Machine	C	0.198076
	gamma	4.684829
	scaler	std
Random Forest	bootstrap	True
	criterion	entropy
	max_depth	7
	max_features	log2
	min_samples_split	100
	n_estimators	55
	scaler	None
Gradient Boosting Trees	colsample_bytree	0.547000
	learning_rate	0.050567
	max_depth	3
	n_estimators	391
	reg_gamma	0.000010
	subsample	0.208225
	scaler	None
Neural Network	activation	relu
	batch_size	64
	dropout	0.298524
	epochs	25
	hidden_units	196
	init	lecun
	learning_rate	0.044112
	optimizer	sgd
	scaler	std

LBWI-G dataset

Learning algorithm	Hyper-parameter	Value
Logistic Regression	C	1485.514868
	penalty	l1
	warm_start	False
	scaler	std
k-Nearest Neighbor	n_neighbors	201
	p	1
	weights	distance
	scaler	std
Support Vector Machine	C	942.959026
	gamma	0.000600
	scaler	std
Random Forest	bootstrap	True
	criterion	entropy
	max_depth	9
	max_features	log2
	min_samples_split	36
	n_estimators	191
	scaler	None
Gradient Boosting Trees	colsample_bytree	0.456373
	learning_rate	0.087289
	max_depth	3
	n_estimators	90
	reg_gamma	0.000876
	subsample	0.320172
	scaler	None
Neural Network	activation	relu
	batch_size	64
	dropout	0.501592
	epochs	25
	hidden_units	43
	init	lecun
	learning_rate	0.061641
	optimizer	sgd
	scaler	std

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, et al. "TensorFlow: A System for Large-Scale Machine Learning". In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, 2016, pp. 265–283.

[2] Sylvain Arlot and Alain Celisse. "A Survey of Cross-Validation Procedures for Model Selection". In: *Statistics Survey* 4 (2010), pp. 40–79.

[3] James S. Bergstra and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization". In: *Journal of Machine Learning Research* 13.1 (2012), pp. 281–305.

[4] James S. Bergstra et al. "Algorithms for Hyper-Parameter Optimization". In: *Advances in Neural Information Processing Systems 24*. NIPS, 2011, pp. 2546–2554.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006.

[6] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32.

[7] Leo Breiman, Jerome H. Friedman, and Charles J. Stone. *Classification and regression trees*. Chapman & Hall, 1993.

[8] Gavin C. Cawley and Nicola L. C. Talbot. "On Over-Fitting in Model Selection and Subsequent Selection Bias in Performance Estimation". In: *Journal of Machine Learning Research* 11 (2010), pp. 2079–2107.

[9] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *CoRR* abs/1603.02754 (2016).

[10] François Chollet. *Keras*. 2015. URL: `http://bit.ly/2atJMxE`.

[11] Bo-Yu Chu et al. "Warm Start for Parameter Selection of Linear Classifiers". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2015, pp. 149–158.

[12] Thomas M. Cover and Peter E. Hart. "Nearest Neighbor Pattern Classification". In: *IEEE Transactions on Information Theory* 13.1 (Sept. 1967), pp. 21–27.

[13] Elizabeth R. DeLong, David M. DeLong, and Daniel L. Clarke-Pearson. "Comparing the Areas under Two or More Correlated Receiver Operating Characteristic Curves: A Nonparametric Approach". In: *Biometrics* 44.3 (1988), pp. 837–845.

[14] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. 2nd Edition. Wiley, 2000.

[15] Tom Fawcett. "An Introduction to ROC Analysis". In: *Pattern Recognition Letters* 26 (2006), pp. 861–874.

[16] Peter Flach. *Machine Learning: the Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012.

[17] Jerome H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *Annals of Statistics* 29.5 (2001), pp. 1189–1232.

[18] Luigi Gagliardi et al. "Assessing mortality risk in very low birthweight infants: a comparison of CRIB, CRIB-II, and SNAPPE-II." In: *Arch Dis Child Fetal Neonatal* 89.5 (2004), pp. 419–422.

[19] Larry C. Gilstrap, Robert Christensen, William H. Clewell, et al. "Effect of corticosteroids for fetal maturation on perinatal outcomes: Nih consensus development panel on the effect of corticosteroids for fetal maturation on perinatal outcomes". In: *JAMA* 273.5 (1995), pp. 413–418.

[20] Trevor Hastie and Robert Tibshirani. *The Elements of Statistical Learning*. 2nd Edition. Springer, 2009.

[21] Trevor Hastie et al. *An Introduction to Statistical Learning with Applications in R*. 4th Edition. Springer, 2014.

[22] Simon Haykin. *Neural Networks and Learning Machines*. 3rd Edition. Pearson, 2009.

[23] Donald O. Hebb. *The Organization of Behavior*. Wiley, 1949.

[24] Jeffrey D. Horbar. "The Vermont Oxford Network: Evidence-Based Quality Improvement for Neonatology". In: *Pediatrics* 103.Supplement E1 (1999), pp. 350–359.

[25] Jeffrey D. Horbar et al. "Hospital and Patient Characteristics Associated With Variation in 28-Day Mortality Rates for Very Low Birth Weight Infants". In: *Pediatrics* 99.2 (1997), pp. 149–156.

[26] Kurt Hornik, Maxwell Stinchcombe, and Halber White. "Multilayer Feedforward Networks Are Universal Approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366.

[27] David W. Hosmer Jr., Stanley Lemeshow, and Rodney X. Sturdivant. *Applied Logistic Regression*. 3rd Edition. Wiley, 2013.

[28] John D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95.

[29] Nathalie Japkowicz and Mohak Shah. *Evaluating Learning Algorithms: a Classification Perspective*. Cambridge University Press, 2011.

[30] Teuvo Kohonen. *Self-Organizing Maps*. 3rd Edition. Springer, 2001.

[31] Max Kuhn and Kjell Johnson. *Applied Predictive Modeling*. Springer, 2013.

[32] Yann LeCun, Leon Bottou, et al. "Efficient BackProp". In: *Neural Networks: Tricks of the trade*. Springer, 1998.

[33] Erich L. Lehmann and Joseph P. Romano. *Testing statistical hypotheses*. 3rd Edition. Springer, 2005.

[34] Stuart P. Lloyd. "Least Squares Quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137.

[35] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[36] Wes McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. 2010, pp. 51–56.

[37] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[38] The Vermont Oxford Network. *Manual of Operations: Part 2. Data Definitions & Infant Data Forms. Release 18.0*. 2014. URL: `http://bit.ly/2lg5J79`.

[39] Travis E. Oliphant. "Python for Scientific Computing". In: *omputing in Science & Engineering* 9 (2007), pp. 10–20.

[40] Stephen W. Patrick, Robert E. Schumacher, and Matthew M. Davis. "Methods of Mortality Risk Adjustment in the NICU: A 20-Year Review". In: *Pediatrics* 131.1 (2013), pp. 68–74.

[41] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[42] John C. Platt. "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods". In: *Advances in Large Margin Classifiers*. MIT Press, 1999, pp. 61–74.

[43] Murray M. Pollack et al. "A Comparison of Neonatal Mortality Risk Prediction Models in Very Low Birth Weight Infants". In: *Pediatrics* 105.5 (2000), pp. 1051–1057.

[44] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. 2016. URL: `http://bit.ly/1gm1uk2`.

[45] Brian Ripley. *tree: Classification and Regression Trees*. R package version 1.0-37. 2016. URL: `http://bit.ly/2ktDttN`.

[46] Frank Rosenblatt. "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain". In: *Psychological Review* 65.6 (1958), pp. 65–386.

[47] Fabrice Rossi. *Yasomi is Yet Another Self-Organising Map Implementation (in R)*. 2013. URL: `http://bit.ly/2lghtGM`.

[48] Guido van Rossum. *Python Reference Manual*. Tech. rep. 1995. URL: `http://bit.ly/1oDM6iq`.

[49] Sebastian Ruder. *An overview of Gradient Descent Optimization Algorithms*. Tech. rep. 2016. URL: `http://bit.ly/2ktPgbj`.

[50] David E. Rumelhart, James L. McClelland, and Geoffrey E. Hinton. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*. MIT Press, 1986, pp. 77–109.

[51] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[52] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[53] Marina Sokolova and Guy Lapalme. "A Systematic Analysis of Performance Measures for Classification Tasks". In: *Information Processing & Management* 45.4 (2009), pp. 427–437.

[54] Nitish Srivastava et al. "Dropout: a Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[55] Theano Development Team. "Theano: A Python framework for fast computation of mathematical expressions". In: *arXiv e-prints* abs/1605.02688 (2016).

[56] Vladimir N. Vapnik. *The Nature of Statistical Learning*. Springer, 2000.

[57] Juha Vesanto. "SOM-Based Data Visualization Methods". In: *Intelligent Data Analysis* 3.2 (1999), pp. 111–126.

[58] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag, 2009.

[59] Linda L. Wright, Jeffrey D. Horbar, Harry Gunkel, et al. "Evidence from Multicenter Networks on the current use of antenatal corticosteroids in very low birthweight infants". In: *American Journal of Obstetrics and Gynecology* 173.1 (1995), pp. 263–269.

[60] John A. F. Zupancic, Douglas K. Richardson, Jeffrey D. Horbar, et al. "Revalidation of the Score for Neonatal Acute Physiology in the Vermont Oxford Network". In: *Pediatrics* 119.1 (2007), pp. 156–163.