UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MATTYWS FERREIRA GRAWE

Heterogeneous Ensemble Models for In-Hospital Mortality Prediction

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof^{a}. Dr^{a}. Viviane Moreira

Porto Alegre
November 2021

CIP — CATALOGING-IN-PUBLICATION

Grawe, Mattyws Ferreira

Heterogeneous Ensemble Models for In-Hospital Mortality Prediction / Mattyws Ferreira Grawe. – Porto Alegre: PPGC da UFRGS, 2021.

62 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2021. Advisor: Viviane Moreira.

1. Mortality prediction. 2. Data types. 3. Machine learning. 4. Ensemble. 5. Time-series. I. Moreira, Viviane. II. Título.

ABSTRACT

The use of Electronic Health Records data have extensively grown as they become more accessible. In machine learning, they are used as input for a large array of problems, as the records are rich and contain different types of variables, including structured data (*e.g.,* demographics), free text (*e.g.,* medical notes), and time series data. In this work, we explore the use of these different types of data for the task of in-hospital mortality prediction, which seeks to predict the outcome of death for patients admitted at the hospital. We built several machine learning models, - such as LSTM, TCN, and Logistic Regression for each data type, and combine them into a heterogeneous ensemble model using the stacking strategy. By applying deep learning algorithms of the state-of-the-art in classification tasks and using their predictions as a new representation for our data we could assess whether the classifier ensemble can leverage information extracted from models trained with different data types. Our experiments on a set of 20K ICU stays from the MIMIC-III dataset have shown that the ensemble method brings an increase of three percentage points, achieving an AUROC of 0.853 (95% CI [0.846,0.861]), a *TP Rate* of 0.800, and a weighted *F-Score* of 0.795.

Keywords: Mortality prediction. data types. machine learning. ensemble. time-series.

Predição de Mortalidade na UTI com Modelos Ensemble Heterogêneos

RESUMO

Com o crescimento da adoção de prontuários eletrônicos, e da acessibilidade da comunidade a esses dados, a área de aprendizado de máquina está fazendo o uso desses dados para a solução de uma vasta gama de problemas. Esses dados são ricos e complexos, e contam com uma diversidade grande de tipos de dados, como dados estruturados (*e.g.,* dados demográficos), texto livre (*e.g.,* exames e prontuário médico) e dados temporais (*e.g.,* medições de sinais vitais). Neste trabalho, buscamos explorar essa diversidade de tipos de dados para a tarefa de predição de mortalidade durante a estadia no hospital. Mais especificamente, usando apenas a janela das primeiras 48h de estadía do paciente. Contruímos diversos modelos de classificação para essa tarefa - incluindo LSTM, TCN e Logistic Regression - para cada tipo de dado existente na nossa base de dados, aplicando algoritmos do estado-da-arte da área de *deep learning*. Usando o resultado da classificação obtido por esses modelos, modelos *ensemble* foram treinados. Com isso, é possível avaliar se esses modelos conseguem tentar melhorar qualidade da classificação. Nossos experimentos usaram um conjunto de mais de 20mil estadias em UTIs presente na base de dados MIMIC-III, e mostramos que o uso de *ensemble* melhora a performance final em 3 pontos percentuais, conseguindo um melhor resultado de AUROC de 0,853 (95% IC [0,846; 0,861]), um *TP Rate* de 0.800, e um *weighted F-Score* de 0.795.

Palavras-chave: Predição de mortalidade, aprendizado de máquina, mineração de dados, algoritmos de classificação.

LIST OF ABBREVIATIONS AND ACRONYMS

EHR	Eletronic Health Records
ICU	Intensive Care Unit
MIMIC-III	Medical Information Mart for Intensive Care
STS	Structured Time Series
TTS	Textual Time Series
SD	Structured Data
PV-DM	Paragraph Vector - Distributed Memory
PV-DBOW	Paragraph Vector - Distributed Bag of Words
NB	Naive Bayes
LR	Logistic Regression
SVM	Support Vector Machines
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
CNN	Convolutional Neural Network
TCN	Temporal Convolutional Network
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
SOFA	Sequential Organ Failure Assesment
AUROC	Area Under Reciever Operating Curve
CUI	Concept Unique Identifier
WBC	White Blood Cell count
CCU	Coronary Care Unit

CSRU	Cardiac Surgery Care Unit
MICU	Medical Intensive Care Unit
SICU	Surgical Intensive Care Unit
TSICU	Trauma/Surgical Intensive Care Unit

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

In an Intensive Care Unit (ICU) environment, the duration of a patient's stay is associated with long-term sequelae, high financial cost, and places strain on the healthcare delivery system. To mitigate this strain in the medical system, the early detection of the deterioration in a patient's condition can reduce the outcome of mortality. This is confirmed by Kane et al. (2008), who observed that an increased number of registered nurse staffing is associated to lower mortality and adverse patient events. One way to improve patient monitoring and help early detection is by using automated systems and machine learning models. The work by Shimabukuro et al. (2017) found a decrease in both ICU length of stay and mortality rate after the adoption of a machine learning model for sepsis detection.

With the increasing adoption of Electronic Health Records (EHR), the volume of healthcare data is rapidly growing in the last few years. EHR data have extensively been in the development of machine learning models for several tasks within the medical field and poses a set of challenges for the machine learning area. These challenges stem from various reasons, including the temporal and sparse nature of the data. Data about a patient can be structured text (*e.g.,* demographics), free text (*e.g.,* a description of the patient's evolution), numerical (*e.g.,* vital signs), or even sound, image, and video. The volume and complexity of the data increase even more if the patient needs to be admitted into an Intensive Care Unit (ICU), as these patients need more care. These different types of data can be used to help understand the patient's current condition and create means to foresee the most likely outcomes for the patient.

The focus of this work is on in-hospital mortality prediction, which is the task of predicting the outcome of death for patients admitted at the hospital. More specifically, our analysis is on *mortality prediction of ICU patients*, based on data available at *the first 48 hours* since the patient is admitted into the ICU. We model this problem as a binary classification task and the solution relies on supervised learning algorithms.

Most of the existing work on in-hospital mortality prediction focus on a single type of data - *e.g.,* textual or structured - (HARUTYUNYAN et al., 2019; PIRRACCHIO et al., 2015; SUSHIL et al., 2018; JOHNSON; POLLARD; MARK, 2017; LEHMAN et al., 2012; MARAFINO et al., 2018; REDFERN et al., 2018; HSIEH et al., 2014), or combine two data types (WEISSMAN et al., 2018; HASHIR; SAWHNEY, 2020; DAVOODI; MORADI, 2018). Conversely, we leveraged the richness and variety of data in EHR systems and employed the array of data types in an ensemble model. By training models on top of specific approaches to handle structured and unstructured data associated with patients, we define a methodology that applies state-of-the-art machine learning techniques for each specific data, and use the strength of ensemble methods to combine them.

Our methodology was applied to 20K ICU stays from the Medical Information Mart for Intensive Care (MIMIC-III) (JOHNSON et al., 2016) database. We explored the diversity of patient data in MIMIC-III, including medical notes and vitals, to generate accurate predictions. We grouped the patient's features into structured data, structured time series data, and textual time series data. Our analysis considered each data both separately and combined using ensemble learning with the stacking algorithm. The results obtained by the ensemble were 0.853 (95% CI [0.846, 0.861]) in terms of the area under the receiver operating curve and 80% of true positive rate. This represents an increase of three percentage points compared to the best experimental run using a single data type.

The main contributions of this work are:

- evaluate each of the three data types to determine which provides the best performance;
- analyze if an ensemble model could leverage information extracted from models trained with different data types;
- leverage the relationship between each information in the ensemble prediction.

In addition, our experiments evaluated the effects of approaches for tackling the class imbalance problem. In our data, the positive class (*i.e.,* the death outcome) accounts for only 13.5% of the cases. This is typically problematic for classification algorithms as they can become biased towards the most prevalent class. Our results showed that addressing the class imbalance problem brings performance improvements for the classifiers.

The remainder of this work is organized as follows. In Chapter 2, we introduce the foundations of the machine learning algorithms used in this work, along with the metrics and methods used to evaluate them. Then, in Chapter 3 we discuss existing approaches for in-hospital mortality prediction. In Chapter 4.1 we discuss the methodology applied for data extraction, and in Chapter 4.2 the methodology applied for model training and evaluation. Chapter 5 we go through the results of the experimental evaluation.

2 BACKGROUND

In this section, we introduce the underlying concepts and techniques related to our methodology for predicting in-hospital mortality. In Section 2.1 it will be discussed data representation and how it affects the machine learning algorithms. In Section 2.2 is the discussion of classical machine learning algorithms employed in our methodology. After, in Section 2.3 we explain the deep learning algorithms used.

In Section 2.4 it will be talked about the ensemble learning algorithm that was used to perform the aggregation of our data. Then in Section 2.5 we will talk about Hyperband, the hyperparameter tunning algorithm used in our tests. Finally, we will finish this chapter with the evaluation methods used in our work, in Section 2.6 we explain the metrics used to evaluate our methodology and in Section 2.7 we exaplain the Shapley values.

2.1 Text Representations

As free text does not have a structure, machine learning methods cannot directly extract information from its raw format. As a result, we need to find a way to represent this type of data so it can be fed into a machine learning algorithm.

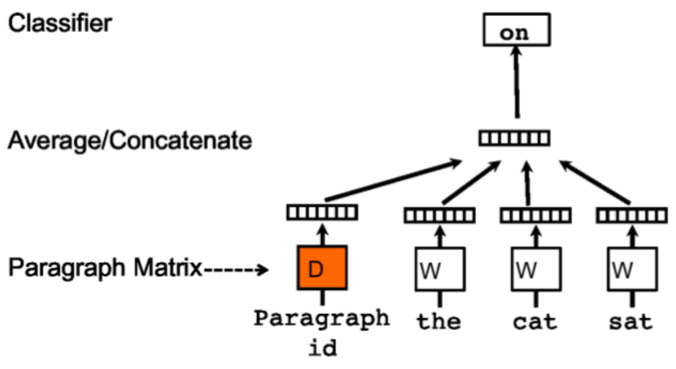
Representation Learning is the field of machine learning that allows systems to learn and transform data into a representation that differs from its original format (GOODFELLOW; BENGIO; COURVILLE, 2016). The performance of machine learning algorithms is directly affected by the data representation, and the process of hand engineering a representation is laborious and requires domain knowledge. As stated by Goodfellow, Bengio and Courville (2016), most machine learning problems could be solved by finding a representation that contains features that are representative to solve the problem, and then feed them to a machine learning algorithm. Another use for transforming representation is to transform data that could not be directly fed to a classification algorithm into a representation that could be used. This is a common step for unstructured data such as texts, audio, and images.

Distributed Representations, which are generated based on patterns found in data, are widely used for text representation because of their power of embodying the semantics of the words. Within this class, we find word and sentence embeddings (LE; MIKOLOV, 2014; MIKOLOV et al., 2013), which assign continuous space representations to words and sentences. This class of representation exploits the fact that words share attributes based on semantic context, and that a generalization can be induced by creating a space in which words with similar meanings are close to each other (GOODFELLOW; BENGIO; COURVILLE, 2016).

A popular method for generating a continuous representation of a sentence is Doc2Vec (LE; MIKOLOV, 2014). Doc2Vec is a set of two neural network architectures, each trained on a specific task, and the representation generated by both is a real-valued vector of the input sentence. The first architecture is called Distributed Memory (PV-DM), and it is trained on the task of outputting the next token, giving a sequence of tokens, and an internal identification of the current sentence that is being used for training. The second architecture is the Distributed Bag-of-Words (PV-DBOW). In this architecture, the training task is to output random words within the sentence.

Doc2Vec functions with the use of two matrices, D and W. D stores the vectors that represents the different paragraphs in the database, and W stores the representation of each word in the vocabulary. In Figure 2.1 we see a graphic representation of the PV-DM architecture, and it is shown the task of output the next word given the sentence. There D represents the *"Paragraph id"*, that acts as a memory for the present context. During training, D is unique for each paragraph fed to the network, while W is shared across paragraphs.

Figure 2.1: Doc2Vec's PV-DM architecure

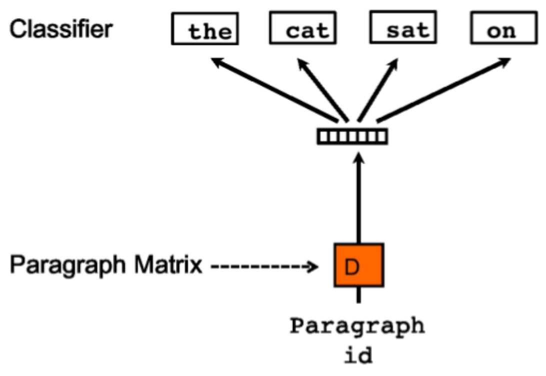


Source: (LE; MIKOLOV, 2014)

Figure 2.2 is the illustration of the PV-DBOW architecture. The figure show the task of output words given only a context, the *"Paragraph id"* D. After trained, either architecture can be used to infer a representation for a sentence that was not previous fed to the network.

We used Doc2Vec to generate a representation of the medical notes in our database.

Figure 2.2: Doc2Vec's PV-DBOW architecture



Source: (LE; MIKOLOV, 2014)

The real-valued vector representation generated can be used as an input to a neural network directly, which facilitates the use of textual information to train models.

It is important when dealing with machine learning algorithms that we first express all data types to a numerical representation. In this section we explained the algorithms used in the experiments in this work, now we will explain the machine learning algorithms used in our methodology.

2.2 Classical Machine Learning Algorithms

In this section, we will focus only on the classical machine learning algorithms used in this work, providing a brief introduction to each of them. For a more detailed discussion, please refer to Harrington (2012) and Russell and Norvig (2009). It is well established in machine learning that no single classification algorithm is able to perform well in all tasks. In this sense, we selected three algorithms from different families to apply to mortality prediction.

The first algorithm is the Naive Bayes (NB) classifier, which is based on the Bayesian theorem with the assumption of feature independence. It computes conditional probabilities for each feature in relation to the class to give the likelihood of that instance to belong to that class.

The second classifier is the Logistic Regression (LR) classifier, and it functions by multiplying each feature by a weight and adding them up. The result is given as input to a sigmoid function, which outputs a value between 0 and 1 that reflects the probability of belonging to a specific class.

The third classifier is the Support Vector Machines (SVM) (CORTES; VAPNIK, 1995). SVM creates a hyperplane (or a set of hyperplanes) which best separates the classes. These hyperplanes are used to classify new instances to a class using the boundaries of each plane.

One last classical algorithm that was not directly used but is the base for the following architectures is the Multilayer Perceptron (MLP). MLPs are composed of a set of neurons, where each neuron is a non-linear function. These neurons are organized in layers, with one layer for input, one layer for output, and one or more hidden layers. Each neuron receives multiple inputs from previous neurons, and associate a weight for each of these inputs. Each weight is corrected by the training algorithm (HAYKIN, 2001).

The algorithms described in this section have their limitations, especially when dealing with temporal data – and most of the data in our work is temporal. To deal with temporal data, we evaluated two deep learning algorithms, which are presented in the following sections.

2.3 Deep Learning Algorithms

Deep learning algorithms have shown impressive results in many classification tasks and quickly became the state-of-the-art. A key aspect is that these algorithms do not require feature engineering, which is a time-consuming step in the classical machine learning background. Next, we briefly introduce recurrent and convolutional neural networks which are the main types of deep learning algorithms and are employed in this work.

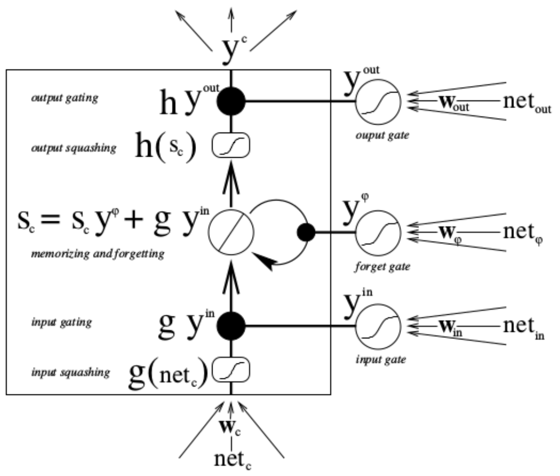
2.3.1 Long Short Term Memory

Recurrent Neural Networks (RNN) are Neural Networks that have the ability to process sequential data, learning a probabilistic distribution over the sequence. The major problem with RNNs is that they fail to capture long-term dependencies, as their gradient either explodes or vanishes, resulting in the bad performance of gradient optimization algorithms (BENGIO; SIMARD; FRASCONI, 1994; HOCHREITER et al., 2001). A solution to this exploding and vanishing problem can be seen in the Long Short Term Memory (LSTM) network architecture created by Hochreiter and Schmidhuber (1997).

This network uses a sophisticated, non-linear network activation function, based on specialized units called *gates*.

An LSTM model is composed of multiple gates that act on the gradient, controlling its value inside each memory unit that builds the network. The network is usually built using: an input gate, that controls the gradient that is incoming to the unit; an output gate, which acts on the gradient that the unit is outputting to the network; and a forget gate introduced by Gers, Schmidhuber and Cummins (2000), which controls the effect of past information in the sequence. Figure 2.3 illustrates a memory unit from LSTM, where y^{in} is the input gate activation, computed by net_{in} (the network error for the input gate), and w_{in} (the memory cell's input gate weight). The same is true for the output gate activation y^{out}, and the forget gate activation y^{φ}, computed by their respective input error and weight. Next, the network input net_c to this cell, which passes through an *input squashing* that is a centered logistic sigmoid function and put the network input into a range of $|-2, 2|$, and then multiplied by the input gate activation y^{in}. This is used to update the unit's internal state s_c for the current timestep. The forgetting step is done by multiplying the forget gate activation with the current cell unit state. Then the updated internal state s_c is squashed by another centered sigmoid function, but with a range between $|-1, 1|$, and then multiplied by the output gate activation y_{out}, generating the cell unit output y^c.

Figure 2.3: Memory unit of a LSTM network architecture with forget gate.



Source: (GERS; SCHMIDHUBER; CUMMINS, 2000)

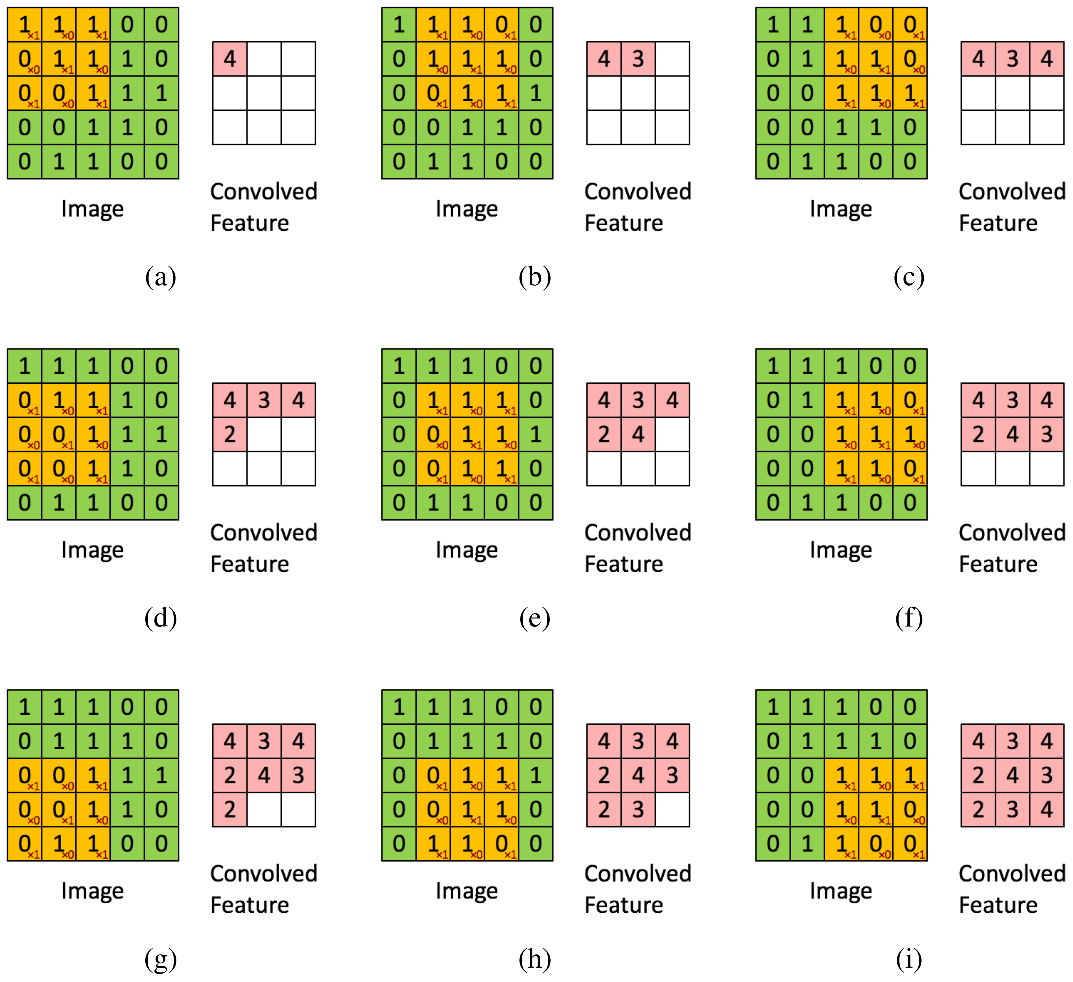
LSTMs can deal with time-series data and are widely used in a diversity of classification problems for this capability. Here we use it as one of our base-models, because of the temporal aspect of our data.

2.3.2 Temporal Convolutional Network

Convolutional Neural Networks (CNN) (LECUN et al., 1998) are capable of extracting spatial or temporal features, using two different operations, *convolution*, and *pooling*. With these operations, CNNs output a smaller, more meaningful representation that is fed to a highly connected network.

A convolution is a linear transformation over the input data, it is composed of a Kernel, that sequentially moves over the data dimensions, producing an output for each time the *kernel* move. Each moving is done by a stride, which is the number of sequential units that the *kernel* will move. Figure 2.4 is the graphic illustration of a convolution into a matrix. In the figure, we can see the *kernel* highlighted in yellow moving through the input matrix in green, and creating a convolved feature at each movement.

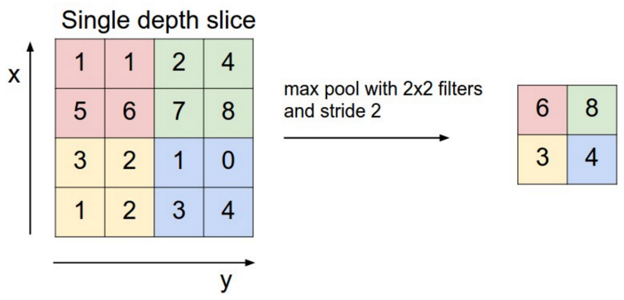
Figure 2.4: Illustration of a convolution.



Source: (BRITZ, 2015)

The pooling operation has a similar structure as the convolution, as it operates with Kernels and strides, but its objective is to reduce the dimension of its input using a simple filtering operation, usually min, max and average. Figure 2.5 illustrates the result of a pooling over a matrix.

Figure 2.5: A pooling operation over a matrix.



Source: (BRITZ, 2015)

However, there is a limitation in the CNNs when working with sequential data. The convolution operation, because of its nature it leaks information from future time steps while progressing through the sequence (BAI; KOLTER; KOLTUN, 2018).

To solve this problem, Lea et al. (2016) designed the Temporal Convolutional Network (TCN). The leakage of information is solved by the use of a modification named causal convolutions, which adapts convolution to output at time t using only data until time $t-1$. Although it solves the problem of leakage, causal convolution alone cannot examine historical information with a size more than linear kernel, which difficult analysis of tasks requiring longer history.

To mitigate this problem, the solution is to apply dilated convolution, which lets the convolution *kernel* activate distant points in the input. Formally, for a 1-D sequence input $x \in \mathbb{R}$ and a *kernel* $f : \{0, ..., k-1\} \rightarrow \mathbb{R}$, the dilated convolution operation F on element s of the sequence is defined as

$$F(s) = (x *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot x_{s-d \cdot i} \tag{2.1}$$

In Equation 2.1, d is the dilation factor, k is the filter size, and $s - d \cdot i$ accounts for the direction of the past. The result is the same as inserting fixed steps between adjacents kernel units, enabling the output to represent a wider range of inputs.

As summarized by Bai, Kolter and Koltun (2018), TCNs have a few advantages in relation to RNNs. The first one is that the convolution operation can be parallelized, reflecting on a shorter training time with smaller memory requirements. However, when performing predictions on an already trained model, it uses more memory than an RNN. We chose to use a TCN because it can deal with temporal data, generating a set of different models to create our final ensemble model.

Now that we presented all the base machine learning algorithms used in this work, in the next section we will explain the ensemble algorithm used to glue all the parts together.

2.4 Ensemble Learning

An ensemble learning algorithm is a method of combining multiple models or generalizers into a single model, with the end to increase the predictive performance. Usually, the models that compose an ensemble reflect subspaces of the entire hypothesis space based on the data used. One method to create and combine multiple models is Stacked Generalization, or *Stacking*. As stated in the original work (WOLPERT, 1992), stacking works by deducing the biases of the generalizers with respect to a provided learning set, by generalizing in a second space whose inputs are.

Given a set $D = \{(x_n, y_n), n = 1, ..., N\}$, with x_n being the values of the features for the data instance n and y_n being the class of this instance, let us denote D_j and $D^{-j} = D - D_j$ as the test and training set, respectively, of a j-fold cross-validation over the dataset D. Now let us define a set of k learning algorithms, which we call *base-models*, chosen to train over this data to generate a set of base-models M_k^{-j}. For each instance x_n in D_j, we use the predictions z_{kn} generated by each model M_k^{-j} as the new representation for x_n. In the end of the cross validation process, data is represented by $D_{CV} = \{(y_n, z_{1n}, ..., z_{kn}), n = 1, ..., N\}$ and is used to train a new model, which is called *meta-model* (TING; WITTEN, 1999).

Polikar (2006) argues that ensemble methods should be used to reduce the risk of an unfortunate selection of a poorly performing classifier. One of the applications of an ensemble is as a data fusion method, so that we could combine heterogeneous features into a single classifier. We use stacking as a data fusion method to combine the array of data types that we have in our dataset. Providing an appropriate method to work with these different data types, generating a new feature space with the classes probabilities generated by the base-models as data to a new single classifier (TING; WITTEN, 1999).

Until now we described the set of machine learning algorithms that we used in our work. However, each of them has a set of parameters that are tied to how well they can learn patterns in data. In the next section, we describe which techniques we used to automatically identify a set of optimal values for these parameters for each algorithm.

2.5 Hyperparameter Optimization

A typical machine learning algorithm has the final objective of minimizing a loss function over a finite set of instances from a natural distribution. This is done by mapping these instances to a function f through the optimization of a set of parameters Θ. This optimization process is controlled by a set of hyper-parameters λ and the quality of the machine learning predictions depends on the configuration of these hyper-parameters (BERGSTRA; CA; CA, 2012). However, the understanding of how they interact with each other and how these interactions affect the resulting model is lacking.

Practitioners often use brute-force methods for hyperparameter optimization, such as random search and grid search. Li et al. (2017) developed Hyperband, a method to approach the optimization problem addressing it as a pure-exploration adaptive resource allocation problem. Hyperband uses another optimization algorithm called Successive Halving as a subroutine (JAMIESON; TALWALKAR, 2016). The Successive Halving optimization method performs an evaluation over all defined configurations, discards half of the configurations with the worse performances, and repeats it until only one configuration remains. It works with resource allocation to converge to an optimal configuration, where it exponentially gives more resources to the most promising configurations. Based on the number n of defined configurations and a finite budget B (training time, training iterations, dataset subsampling, feature subsampling), it allocates B/n resources on average across all the configurations.

Feurer and Hutter (2019) states that automated hyperparameter optimization has several important use cases, such as reducing human effort for applying machine learning; improving the performance of the machine learning algorithms by relating them; and improving reproducibility and fairness of scientific studies. Hyperband and grid search algorithms are used in this work because of these aforementioned benefits.

With all the methods described that are the building blocks for our methodology, now in the final two sections, we have to describe how do we evaluate the performance of these machine learning algorithms.

2.6 Evaluation Metrics

The evaluation of the models generated in this work was done using traditional evaluation metrics that analyze the predictions made by the model in comparison to the ground truth. First, the predictions are assigned into one of four categories:
True Positive (TP) – patients who died in-hospital and were correctly predicted by the model;
True Negative (TN) – patients who did not die in-hospital and were correctly predicted by the model;
False Positive (FP) – patients who survived but who were predicted by the model as dying during the stay;
False Negative (FN) – patients who died during the stay but who were predicted by the model as surviving.

With these four categories, it is possible to measure the performance of a machine learning model using the following metrics:
TP Rate is the ratio $TP/(TP + FN)$ that measures the classifier's ability to classify positive instances correctly.
TN Rate is the ratio $TN/(TN + FP)$ that measures the classifier's ability to classify negative instances correctly.
Precision is the ratio $TP/(TP + FP)$ and it measures the percentage of instances that were correctly assigned to the class out of all instances that the model classified as belonging to that class. Here we use the weighted version, *i.e.,* we compute the metric for both the positive and the negative class and then calculate a weighted average in which the weight is given by the number of instances in each class.
Recall is similar to the TP Rate; however, here we denote Recall as the weighted average for both classes (and not just the positive class).
F1-Score is the harmonic mean between Precision and Recall and is defined as:

$$F1 = \frac{2 * (Precision * Recall)}{(Precision + Recall)} \tag{2.2}$$

Since we have an unbalanced dataset, we report on both the weighted (W F-score) and the unweighted (W F-score) macro F1. In the latter, we assign equal weights to the classes.
AUROC or Area Under Receiver Operating Curve is a measurement of how the model is capable of distinguishing between the classes. The ROC curve shows the trade-off

between True Positive Rate and False Positive Rate across different thresholds.

2.7 Shapley Values

Shapley values were introduced by Shapley (2016) and is derived by the game theory. In this theory, every player can evaluate and choose their actions to the end of increasing their payout. For the field of machine learning, we could see players as the values of our input features, and the payout as the prediction for this instance. This way, we can distribute the contribution of each feature to the instance model's prediction.

Let us define F as the set of all features and $S \subseteq F$ as a subset of features in F. To compute the importance of a feature to the prediction of a model, we need a model $f_{S\cup\{i\}}$, trained with a feature i, and a second model f_S trained with i withheld. Then, the predictions of both models are compared, this comparison is done for all subsets $S \subseteq F$ $\{i\}$. This can be expressed as ((LUNDBERG; LEE, 2017; SUNDARARAJAN; NAJMI, 2019)):

$$\phi_i = \sum_{S \subseteq F\{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} [f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S)] \tag{2.3}$$

The work from Lundberg and Lee (2017) introduces the SHAP (SHapley Additive exPlanation) values, as a method of computing Shapley values using additive features attribution. The solution of an additive features attribution problem is desirable to have three properties:

- **Local accuracy**: the explanation model should match the output of the original model f for the input x;
- **Missingness**: missing features in the original input cannot have impact;
- **Consistency**: input's attribution should not decrease if the input's contribution increases or stays the same.

In their work, Lundberg and Lee (2017) describes different approaches to compute SHAP values, two are model-agnostic, and the other four are model-specific. In our work, we are interested in the Linear SHAP method, used to compute the SHAP values for our best model. The intuition of Linear SHAP is that, if we assume input feature independence, SHAP values can be approximated directly from the model's weights. So, given a linear model $f(x) = \sum_{j=1} M w_j x_j + b$ and $\phi_0(f, x) = b$, we have

$$\phi_i = w_j(x_j - E[x_j]) \tag{2.4}$$

With this equation is computed the SHAP values for each feature. These SHAP values will be used to evaluate the contribution of the features used in our ensemble models trained.

3 RELATED WORK

In the realm of medical sciences, an early prediction of the patient's outcome is beneficial to improve the quality of treatment and reduce costs. As a result, there is a significant body of research aimed at early detection which relies on medical scores and/or machine learning.

3.1 Mortality Prediction using SOFA

The Sequential Organ Failure Assessment (SOFA) score was first introduced by Vincent et al. (1996) with the aim of providing means for the identification of multiple organ failure. SOFA uses a numeric scale from 0 to 4, in which higher values mean a more significant organ dysfunction. Although it was not designed for mortality prediction, some studies established a correlation between the SOFA score (measured at the patient's admission) and mortality (JENTZER et al., 2018; MBONGO et al., 2009; HO et al., 2007).

A systematic review by Minne, Abu-Hanna and Jonge (2008) surveyed 18 studies that evaluated the performance of SOFA for mortality prediction of ICU patients. The AUROC scores reported in these studies range from 0.61 to 0.87 considering the SOFA score measured at the patient's admission.

Subsequently, Mbongo et al. (2009) analyzed 864 patients, with a mortality rate of 8.2%, and found that SOFA achieved an AUROC of 0.846 (95% CI [0.796,0.897]) when discriminating survivor vs. non-survivor. More recently, Jentzer et al. (2018) analysed 9,961 ICU patients with 893 (9%) of those having died in-hospital. The SOFA score calculated on the first day predicted mortality with an AUROC of 0.828 (95% CI [0.813,0.843]). In our work, we used the SOFA score to perform a baseline test.

3.2 Mortality prediction using machine learning

Harutyunyan et al. (2019) presented a benchmark of several tasks in the medical field using the MIMIC-III database (JOHNSON et al., 2016). In their work, the authors presented several LSTM architectures and trained them in several tasks in the medical domain. The tasks include predicting in-hospital mortality, decompensation, length-of-stay, and phenotyping. The goal of the authors was to use multitask learning to extract certain useful information from the input sequence that single-task models could not leverage. Instead, they created a single model that was trained using all of the four mentioned tasks. They compared models trained on individual tasks only and used medical scores such as SAPS with their proposed architecture. Their results achieved a higher AUROC in all the tasks in comparison with the metrics and the individual models. For in-hospital mortality prediction, the AUROC was 0.87 (95% CI [0.852,0.887]). They used the same features across their tests, these are: capillary refill rate, diastolic blood pressure, fraction inspired oxygen, Glasgow coma scale eye opening, Glasgow coma scale motor response, Glasgow coma scale total, Glasgow coma scale verbal response, glucose, heart rate, height, mean blood pressure, oxygen saturation, respiratory rate, systolic blood pressure, temperature, weight, pH.

Pirracchio et al. (2015) compared medical scores for severity assessment with machine learning algorithms. The authors trained a *Super Learner* algorithm on data from 24,508 patients from the MIMIC-II database (SAEED et al., 2011) for in-hospital mortality using the first 24h of data following the ICU admission. The *Super Learner* is an ensemble algorithm that selects the optimal set of regression algorithms that minimizes a chosen error, using a set of weighted candidate algorithms. Using the AUROC to score the performance of the methods, the results obtained by the *Super Learner* were 0.88 (95% CI [0.87, 0.89]) outperforming APACHE-II, SAPS-II, and SOFA scores.

Sushil et al. (2018) used clinical notes from the MIMIC-III dataset to train classification algorithms into four classification tasks: mortality prediction (in-hospital, post 30 days discharge mortality, post-one-year discharge mortality), primary diagnostic category prediction, primary procedural category, and gender prediction. In their work, they made use of two representation techniques for the texts, *Stacked denoising autoencoder* (SDAE) and Doc2Vec. They used the concatenation of all notes associated with each patient as the source data. After preprocessing, for the SDAE method, they transformed each patient's text into two separate representations – bag-of-words (BoW) and bag-of-medical-concepts, which used Concept Unique Identifiers (CUIs). For Doc2Vec, they used the textual representation of each patient. They achieved the highest AUROC of 0.9457 by feeding only the BoW representation into a feedfoward neural network for the in-hospital mortality task.

Silva et al. (2021) also used clinical notes from MIMIC-III database to determine their predictive value as prognostic markers for 1-year all-cause mortality among people with diabetes. They used only the first admission from patients with type 1 and type 2 diabetes, who have stayed over one day in an ICU. Using the words in the texts as features and the Least Absolute Shrinkage and Selection Operator as the classification algorithm, they achieved an AUROC of 0.922 (95% CI [0.920, 0,924]) on a dataset with 5,942 patients.

Davoodi and Moradi (2018) proposed a fuzzy deep model for ICU mortality prediction. Their features are demographics and vital signs and the model was evaluated on 11K patients from MIMIC-III. The best performing configuration achieved an AUROC of 0.739 (95% CI [0.738, 0,740]).

Specifically on the topic of ICU mortality prediction using different types of data, there are two closely related works by Weissman et al. (2018) and Hashir and Sawhney (2020). Both of them also used the MIMIC-III database. Weissman et al. (2018) uses structured data from lab results and bedside measurements and unstructured data from the medical notes. They employed two approaches to add unstructured data to the machine learning model. The first one was to extract 21 keywords from all the medical notes, and the second one was to use the 500 most predictive phrases. The phrases were constructed by creating a document-term matrix of one, two, and three-word n-grams, using penalized logistic regression as the classifier. They found that adding unstructured data yields an improvement of 0.06 to 0.09 points in AUROC, reaching 0.89 (95% CI [0.88, 0.90]). Hashir and Sawhney (2020) uses hierarchical CTS-RNN to model multiple notes. Their experiments with around 35K ICU stays achieved the best results when a joint model of notes and structured clinical time series was applied. This configuration scored 0.902 (95%CI [0.898, 0.906]) in terms of AUROC using data from the first 48h of the patient's stay.

Table 3.1 summarizes the characteristics of the existing work on mortality prediction, such as the technique used, dataset size, and AUROC result. All works in this table use the MIMIC-III dataset, either as the only dataset or to evaluate the methodology. These works were obtained by a search of the terms *"MIMIC"* and *"In-hospital mortality"*.

The main differences between this work and these closely related approaches are as follows. While the work by Weissman et al. (2018) does use medical notes, they ignore the time-changing information. In our work, we maintain the time-varying aspect of data, and make use of Deep Learning and representation learning techniques that can handle time-series as input. By maintaining the time aspect of the data, we make use of the information present in the patient progression during the ICU stay, which could be

Table 3.1: In-hospital mortality predictions articles.

Article	Data Origin	Features	Data Window	Data Size (Mortality Rate %)	Technique	AUROC
Pirracchio et al. (2015)	Vital signs and lab results	12 struct. features	24h	24,508 (12.2%)	Super learner	0.88 (95% CI [0.87, 0.89])
Song et al. (2018)	Vital signs and lab results	-	24h	21,139 (13.23%)	Transformer with attention mechanism	0.857
Harutyunyan et al. (2019)	Vital Signs and lab Results	16 struct. features	48h	21,139 (13.23%)	Multitask channel-wise LSTM	0.870 (95% CI [0.852, 0.887])
Sushil et al. (2018)	Clinical notes	-	-	27,731 (9%)	Feedfoward Neural Network	0.945
Lehman et al. (2012)	Clinical notes	-	24h	14,739 (14.6%)	Hierarchical Dirichlet Processes	0.82
Marafino et al. (2018)	Vital signs, lab results, and clinical notes	21 struct. features + notes TF-IDF	24h	101,196 (10.4%)	Logistic Regression	0.922 (95% CI [0.916, 0.924])
Weissman et al. (2018)	Vital signs, lab results, and clinical notes	14 struct. features	48h	25,947 (21.2%)	Penalized Logistic Regression	0.89 (95% CI [0.88, 0.90])
Redfern et al. (2018)	Vital signs and lab results	14 struct. features	24h	97,933 (4.8%)	LDTEWS and NEWS scores	0.916 (95% CI [0.912, 0.921])
Hashir and Sawhney (2020)	Vital signs, lab results, and clinical notes	12 struct. features + medical notes	48h	36,561 (6.5%)	CNN-RNN	0.902 (95%CI [0.898, 0.906])
Davoodi and Moradi (2018)	Vital signs and lab results	27 struct. features	48h	10,972 (9.31%)	Deep Rule-Based Fuzzy System	0.739

favorable to our final predictor. In relation to the work by Hashir and Sawhney (2020), there are two main differences: (i) besides considering time-series data, we also consider static structured data, and (ii) while their multimodal approach uses joint training, ours relies on an ensemble of independent base-models.

Both Chapter 2 and Chapter 3 lays the foundation of our problem and methodology for in-hospital mortality. Before talking about the methodology, we should discuss how we extracted data from MIMIC-III that will be used to train and evaluate this methodology.

4 MATERIALS AND METHODS

In this chapter, we present the methodology used to select our ICU stays and extract the data used, as well as the design of the experiments done to evaluate the machine learning algorithms. First in Section 4.1, we explain how the set of ICU stays and how the data for each of the data types were extracted. Each data type required a specific process. Next, in Section 4.2, we describe how we dealt with each data type to feed them as an input to the machine learning algorithms. Then, we detail the approaches used to deal with unbalanced data. The chapter concludes with a description of the methodology used to evaluate the final ensemble algorithm.

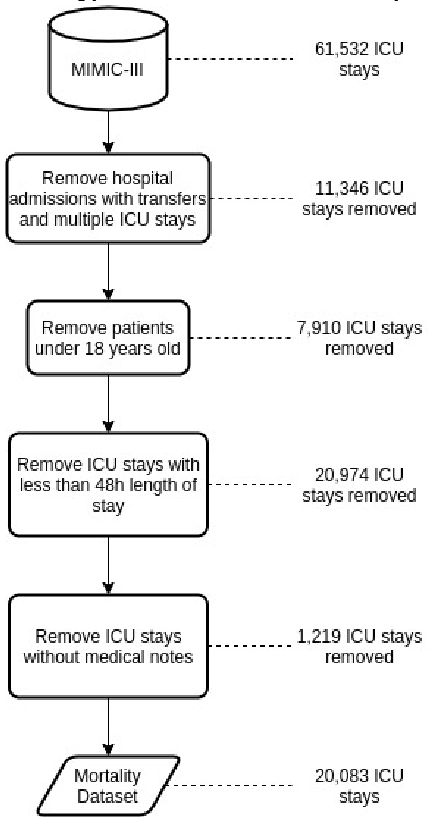
4.1 Data preparation

Our data comes from the MIMIC-III database (JOHNSON et al., 2016). MIMIC-III has data for patients admitted at the Israel Deaconess Medical Center ICU, in Boston. Each patient has one or more hospital admissions in the database, and each hospital admission could have one or more ICU stays. During a stay, medical staff do several visits to the patient to perform observations, take measurements, or administer treatments. Each of these is called an *event*, *i.e.,* a categorical or numerical variable value measured at a specific point in time. Several events could be created for a patient in a single visit.

The MIMIC-III database has a total 61,532 ICU admissions. It is composed of 25 tables, ranging from administrative (patient demographic, medical staff information, ICD9 codes) to laboratory and bedside data amounting to around 13,000 different variables related to a patient. It is relevant to say that not all 13,000 variables are distinct. Because of the nature of the database, with data being collected from two different medical systems fed by several staff members, some events with different names are duplicates. For example, "*H Rate*" and "*heart rate*" refer to the same variable and yet would be recorded as different events. We applied a simple rule-based approach to identify and merge these cases. A deeper analysis of this issue was done by Barcelos, Mendoza and Moreira (2020).

Our data selection for the task of in-hospital mortality followed the same procedure adopted by Harutyunyan et al. (2019). The extraction of the ICU stays for our dataset followed the process illustrated in Figure 4.1. The following stays were discarded:

Figure 4.1: Methodology used to select ICU stays from MIMIC-III.

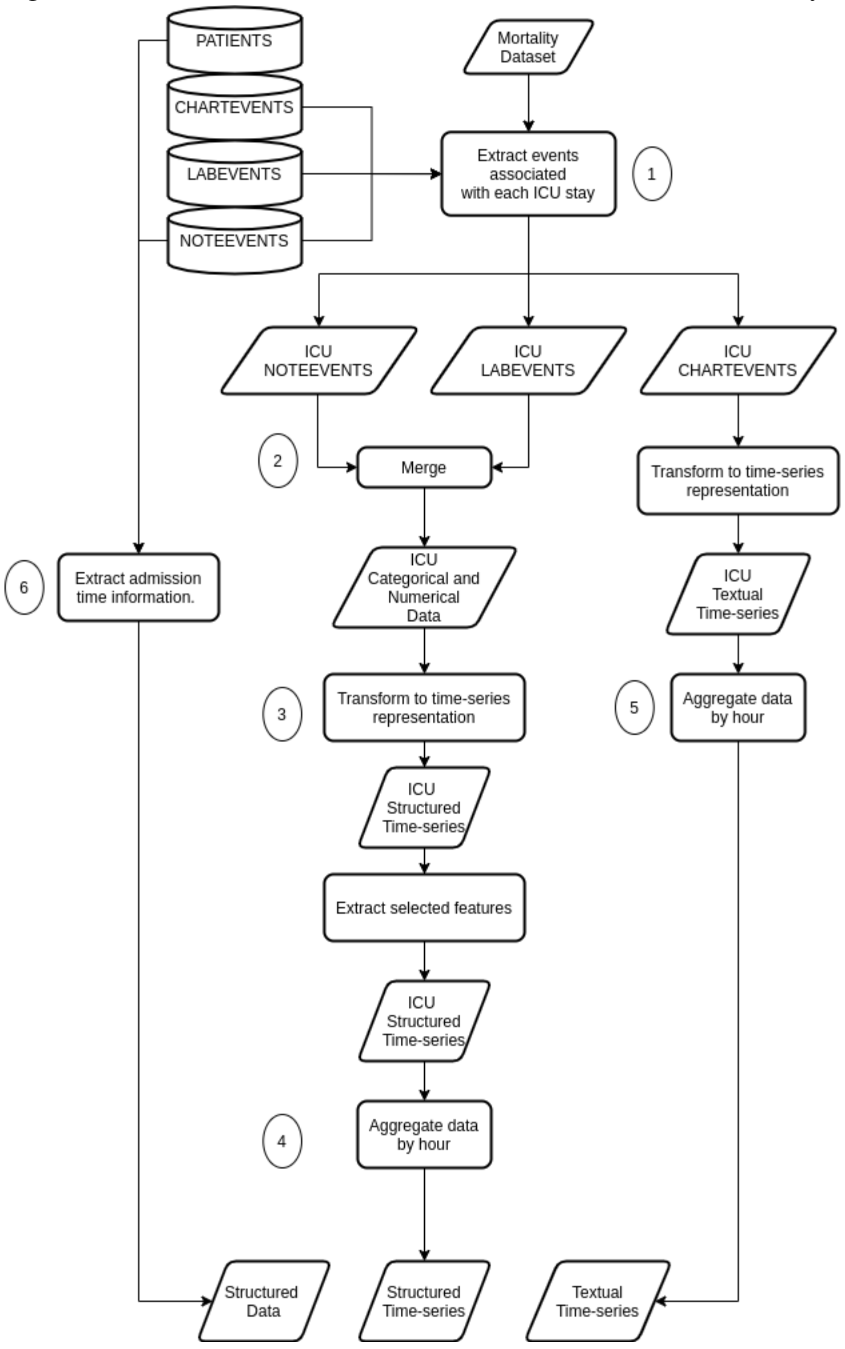

MIMIC-III
61,532 ICU stays
Remove hospital admissions with transfers and multiple ICU stays
11,346 ICU stays removed
Remove patients under 18 years old
7,910 ICU stays removed
Remove ICU stays with less than 48h length of stay
20,974 ICU stays removed
Remove ICU stays without medical notes
1,219 ICU stays removed
Mortality Dataset
20,083 ICU stays

Source: Author

- *Multiple stays*. Hospital admissions that had multiple ICU stays or had any transfers between ICU units or wards during the period of hospitalization. The reason is that the multiple ICU stays would be correlated as they belong to the same individual and a survival outcome may be followed by a death adding noise to the model. These admissions correspond to 11,346 records.
- *Patients under 18 years old*. All patients younger than 18 years old were also removed due to differences between adult and pediatric physiology. This step removed 7,910 ICU stays.
- *Short stays*. ICU stays of less than 48 hours are not relevant to this study as our goal is to use the first 48 hours to predict the outcome. This filter removed 20,974 ICU stays.
- *Stays without medical notes*. All ICU stays that had no recorded medical notes in the first 48h of stay were also removed. The reason for this is that it would create an inconsistency in the comparison between the models generated with structured temporal data and textual temporal data since missing values would affect the final ensemble model.

Our final dataset has a total of 20,083 stays. From those, 17,359 patients were discharged alive from the ICU (negative instances) and 2,724 patients died (positive instances), making that an unbalanced dataset.

Figure 4.2: Extraction of the events associated with the ICU stays.


PATIENTS
CHARTEVENTS
LABEVENTS
NOTEEVENTS
Mortality Dataset
Extract events associated with each ICU stay
1
ICU NOTEEVENTS
ICU LABEVENTS
ICU CHARTEVENTS
2
Merge
Transform to time-series representation
6
Extract admission time information.
ICU Categorical and Numerical Data
ICU Textual Time-series
3
Transform to time-series representation
5
Aggregate data by hour
ICU Structured Time-series
Extract selected features
ICU Structured Time-series
4
Aggregate data by hour
Structured Data
Structured Time-series
Textual Time-series

Source: Author

From the selected ICU stays, we extract the events that are relevant to our classification problem. Each event associated with an ICU stay in MIMIC-III is divided into different tables. The data of interest for us comes from the following tables:

- `PATIENTS`, which stores metadata for each patients;

- `ICUSTAYS`, which stores metadata for each ICU stay;
- `NOTEEVENTS`, which stores the medical notes associated with an ICU stay;
- `LABEVENTS`, which stores data related to laboratory exams; and
- `CHARTEVENTS`, which stores all measurements of physiologic functions.

The extraction process is depicted in Figure 4.2. In (1), we extract events from each of the aforementioned tables, and separate data for each ICU stay into separate files, named by the id of the ICU stay. After that, each data type goes through specific pre-processing tasks. Then, in (2), data from `LABEVENTS` and `CHARTEVENTS` for the same stay were merged. With that, we obtain the numerical and categorical data for each ICU stay and then feed it to the next step (3) that consists in transforming these data into a time-series representation, *i.e.,* the features. In (4), the values of each feature are aggregated by hour, starting at the time of admission in the ICU, until the 48^{th} hour. For each hour bucket, we get the mean value for each feature. To handle missing data, we fill each value with the last measurement made for that feature, and in case that no measurement was taken, we fill it with the mean value extracted from the training set for each loop of the k-Fold. For the medical notes, the process (5) is almost identical. The notes are aggregated by hour, concatenating each textual event. In (6), we extract the weight and height from the admission note, and age and gender from from the `PATIENTS` table.

In Table 4.1, we have the characteristics and variables values for our dataset, along with the values separated by outcome. Looking at the table, we see that the patients who died during their stay were older, and had a length of stay higher than the patients who were discharged alive from the ICU. Most of the patients are male (55.65%), and most of them were admitted and cared for at MICU (40.17%). Features from Systolic Blood Pressure to Pulse Pressure are time-series features, and we see a difference in values between patients' outcomes. The Blood Oxygen Saturation has 0's for their value because that was a value that is highly common.

With the foundation of the methods used and with a broader view about our data, in the next chapter, we move on to the description of our methodology.

4.2 Predicting In-hospital Mortality

In-hospital mortality prediction is the task of assessing whether a patient is likely to die during the course of the hospital stay. Such a prediction should be made preferably

Table 4.1: Baseline characteristics and outcome measures. Continuous variables are presented as median [InterQuartile Range]; Binary or categorical variables as Count (%).

	Overall population	**Dead at Hospital Discharge**	**Alive at Hospital Discharge**
Age	66 [53-78]	73 [60-82]	65 [52-77]
ICU LOS (days)	4 [3-6]	5 [3-10]	4 [3-6]
Height (cm)	170 [162.5-177.8]	167.64 [160-177.8]	170.18 [162.56-177.8]
Weight (kg)	77.4 [65-92]	74 [61-88]	78 [65.77-93]
Sex:			
Male	11136 (55.45%)	1472 (54.04%)	9664 (55.67%)
Female	8947 (44.55%)	1252 (45.96%)	7695 (44.33%)
Site[1] :			
CCU	2723 (13.56%)	347 (12.74%)	2376 (13.69%)
CSRU	3562 (17.74%)	148 (5.43%)	3414 (19.67%)
MICU	8068 (40.17%)	1476 (54.19%)	6592 (37.97%)
SICU	3400 (16.93%)	473 (17.36%)	2927 (16.86%)
TSICU	2330 (11.60%)	280 (10.28%)	2050 (11.81%)
Systolic Blood Pressure (mmHg)	117 [103.5-134]	114 [100-132]	117.5 [104-134]
Diastolic Blood Pressure (mmHg)	58.25 [50-68]	57 [49-66]	59 [51-68]
Mean Blood Pressure (mmHg)	75.83 [66.5-86.67]	74 [65-85]	76 [67-87]
Central Venous Pressure (mmHg)	0 [0-6]	0 [0-8]	0 [0-5]
Temperature (^{o}C)	37 [36.5-37.56]	36.89 [36.28-37.56]	37.1 [36.56-37.56]
Respiratory Rate (insp/min)	18.34 [15-22]	20 [16-24]	18 [15-22]
PaO2 (mmHg)	119 [89-170]	109 [82-154]	121 [90-173]
FiO2 (mmHg)	0.6 [0.45-40]	0.6 [0.5-40]	0.6 [0.4-40]
Bilirubin (mg/dL)	0.9 [0.5-2.5]	1.4 [0.6-5.5]	0.8 [0.4-2.1]
Platelets ($10^9/L$)	177 [119-247]	150 [85-235]	180 [124-248]
Creatinine (mg/dL)	1 [0.7-1.7]	1.3 [0.8-2.4]	1 [0.7-1.5]
Lactate (mmol/L)	1.9 [1.3-3.2]	2.6 [1.6-4.7]	1.8 [1.2-2.9]
BUN (mg/dL)	21 [13-36]	31 [19-51]	19.5 [13-33]
Arterial pH	7.38 [7.33-7.43]	7.37 [7.3-7.43]	7.38 [7.34-7.43]
WBC ($10^9/L$)	11 [7.9-15]	12.1 [7.9-17.2]	10.9 [7.9-14.7]
PaCO2 (mmHg)	36 [31-42]	33 [28-39.5]	36 [31.5-42]
Hemoglobin (g/dl)	10.3 [9.2-11.5]	10.2 [9.1-11.5]	10.3 [9.3-11.6]
Hematocrit (%)	30 [27-33.4]	30.2 [27.1-33.9]	30 [27-33.3]
Potassium (mmol/L)	4.1 [3.7-4.5]	4.1 [3.7-4.6]	4.1 [3.7-4.5]
Glucose (mg/dL)	129 [106-161]	134.75 [107-172]	128 [106-160]
Heart Rate (bpm)	85 [74-98]	88 [75-103]	85 [74-97]
Blood Oxygen Saturation	0 [0-0]	0 [0-0]	0 [0-0]
GCS	14 [9-15]	8 [6-13]	14 [10-15]
Pulse Pressure (mmHg)	58 [47-71]	57 [45-71]	58 [47-71]

[1] Coronary care unit (CCU); Cardiac surgery recovery unit (CSRU); Medical intensive care unit (MICU); Surgical intensive care unit (SICU); Trauma/surgical intensive care unit (TSICU).

using the data from the first hours of the patient's admission into the hospital or ICU. Predicting a possible outcome of a patient in the early stages of admission is important to give the health professionals time to take an adequate course of action to treat the patient properly.

Our approach considers the use of different types of data that are generated by a patient's ICU stay and use them to produce a machine learning model to predict the patient's chance of in-hospital mortality. We model this as a binary classification problem in which the positive class is the death outcome. We rely solely on data generated during the *first 48 hours* of the ICU stay. Data about a patient can be divided into three types: (i) structured data (*e.g.,* weight, height, and sex); (ii) structured time-series data (*e.g.,* results of exams and vital signs); and (iii) textual time-series (medical notes taken through the

ICU stay).

We approached model creation using a stacking algorithm to create an ensemble using multiple models trained on the different types of data. In addition, solutions for tackling the class imbalance problem were employed both in the base- and meta-models. On top of our methodology, we computed the hourly SOFA score to perform a baseline test, and used the SOFA values for the first 48h for each patient. We extracted time-series features and trained a machine learning model using the extracted features.

4.2.1 Data Types

In the following subsections, we further explain the approach employed for each type of data. Each data type necessitates specific methods to tackle its use as data to train a machine learning algorithm. In each subsection, we explain how do we extract the features, how do we use this feature to train the algorithms, and present some general statistics for each feature.

4.2.1.1 Structured Data (SD)

Our structured data consists of the information that does not change over time or data for which changes are not substantial. The purpose of the structured data is to give contextualized information about the patient. The features in this category are weight and height at admission time, age, and sex. The weight and height variables were extracted from the admission notes for each ICU stay, and the other values were extracted from the admission data recorded for each stay.

4.2.1.2 Structured Time-Series (STS)

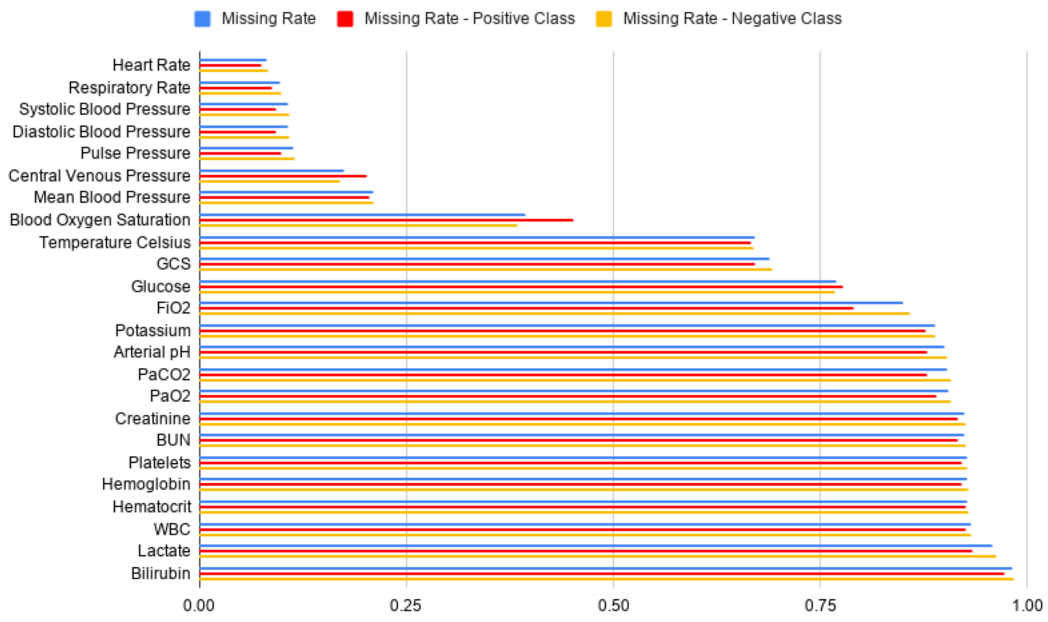
Structured time-series (STS) data are generated by measuring patients' vital signs and recording results of their laboratory exams performed during the patient's ICU stay. It consists of numerical and categorical data measured at a specific point in time by a healthcare provider.

Our STS features are Glasgow Comma Score, Systolic Blood Pressure, Diastolic Blood Pressure, Mean Blood Pressure, Central Venous Pressure, Heart Rate, Respiratory Rate, Blood Oxygen Saturation, Body Temperature, Hemoglobin, Hematocrit, White Blood Cell count (WBC), Platelets, Arterial pH, PaO2, FiO2, PaCO2, Lactate, Creatinine, BUN, Bilirubin, Potassium, and Glucose. These features were selected by a physician and reflect the health state in which the patients' physiological systems are at the moment.

Data from an ICU poses a diversity of challenges since their generation is a product of measurements made by a team composed of different professionals and the nature of the patient's disease and condition. As a consequence, measurements are spread through time at a variable rate, sometimes having hours between measurements and sometimes minutes or days.

One of the problems created by the irregular measurement of features is the missingness of values. For each feature, we have a total of 960,219 events; from those, 126,987 events come from patients who died at the ICU, and 833,232 from patients who survived. The rate of missing events can be seen in Figure 4.3, and we see that most of our features have a missing rate higher than 60%. With the exception of Central Venous Pressure, Blood Oxygen Saturation, and Glucose, the missing rate for the patients in the negative class is higher than in the positive class. During our experiments, the missing values were replaced by the mean values, obtained by adding all events for each feature.

Figure 4.3: Missing events rate for the STS data, total, and for each patient outcome.



Source: Author

To understand if these variables have differences among patients with different outcomes, we extracted the mean value for each hour across all patients grouped by class and created a chart that can be seen in Figure 4.4. The chart shows the values by class for the four variables with the lowest missing rates. This difference between classes is more prominent for Heart Rate and Systolic Blood Pressure, as seen in Figures 4.4a and 4.4c, respectively. For the Respiratory Rate in Figure 4.4b and Diastolic Blood Pressure in Figure 4.4d the values are closer, however a difference still exists. Another important piece of information that we find in these charts is the presence of outliers, represented by the abrupt changes in values in Figures 4.4b and 4.4d. The approach used to reduce the impact of these outliers is the standardization of the values by applying the z-score, that is, subtract each instance by the population mean, and divide the result by the population standard deviation, using only training set to extract both mean and standard deviation.

To deal with the STS data, the method needs to be able to process and extract meaningful patterns through time and to deal with large volumes of high-dimensional data. For the purpose of classifying STS data, we train LSTM and TCN models as these architectures have good performances dealing with temporal data, without the need for dimensionality reduction or feature extraction. The training process for the model follows a process of hyperparameter tuning on a subset separated from the training data, and a k-fold cross-validation as training process.

Hyperparameters play an important part in the performance, but the decision of which hyperparameters to use it is not a simple task. To this end, we used the Hyperband algorithm to search and test a subspace of hyperparameters. The best configuration was used. This process was done onto a separated subset with only 15% of the population size, extracted randomly from the dataset, to avoid using a piece of information both in tuning and in training.

To better analyze the performance of both models, we performed a k-fold cross-validation, analyzing each model performance through several metrics. At the end of the k-fold, we have each model performance measured by predicting the testing fold and their respective class probabilities to be used as input for the ensemble model.

4.2.1.3 Textual Time-Series (TTS)

As measuring vitals and performing exams on patients generates structured temporal data, a visit of a healthcare provider or a health exam can also produce textual data describing the information that is important to understand the patient's history and condition. Along with the challenges of dealing with textual data, these texts are distributed through time, which adds another layer of complexity.

Here, the problem of missing data is yet more prevalent than on STS data. Some patients do not have any textual event in the first 48h of stay. To be precise, the total number of events in the extracted dataset is 143,970, with positive instances having 21,552 and negative instances 122,418. When we compare the total of 960,219 events for the STS data type, we get almost eight times more events for STS data type than the TTS data type.

To be able to use textual data as input to a machine learning model, we adopted the sentence embedding method using Doc2Vec (discussed in Section 2.1) to transform the raw text into a real-valued vector. By transforming each text into different real-valued vectors, we preserve the temporal aspect of the data differently from the approach adopted by Weissman et al. (2018) discussed in Chapter 3.

To train the Doc2Vec model, we used all text available in the MIMIC-III database, and not just the texts in our sample. Having large volumes of text is important to yield good quality representations. Since Doc2Vec is an unsupervised method, it does not take the class we wish to predict into consideration. Thus, it does not introduce bias over our classification model.

After generating the representation model and transforming the texts of our dataset to the new representation. It is important to notice that both models trained on STS and TTS data used the same distribution of patients and folds.

4.2.2 Balancing the Training Data

As discussed in Section 4.1, our dataset is very unbalanced, with the positive instances corresponding to 13.5% of the total. This distribution of data between classes can be harmful to the final performance of the classification model since it could be biased towards the negative class, which is more prevalent. To deal with this issue, we apply two techniques to balance the training data. The evaluation is carried on the test data, which maintains the unbalanced distribution. This is important because the classification models should be able to handle unbalanced data since this is how most real datasets are formed.

The first approach is to use *cost-sensitive classification* to penalize the misclassification of the minority class (false negatives) by a higher degree than the misclassification of the majority class (false positives). Using this method, we do not have to deal with the unbalancing by altering the distribution in the dataset.

The second approach is to random undersample the training data, and consequently the optimization data, by a rate of 1:1, while maintaining the same distribution between classes in the test set. This undersampling is done in the K-Fold, by previously selecting a balanced set for each fold, and using this balanced version as the fold is used for training the machine learning algorithm.

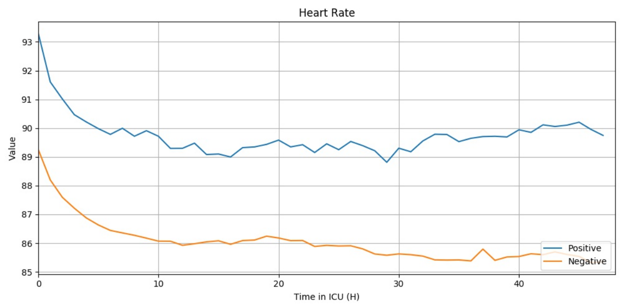
4.2.3 Ensemble Model Creation

As discussed in Section 2.4, classifier ensembles tend to have better predictive performance since they combine more sources of evidence from the data. Our meta-models combine the predictions of the base-models that were trained using data from a single data type. There are four base-models (i) LSTM trained on the STS data, (ii) TCN trained on the STS data, (iii) LSTM trained on TTS, and (iv) TCN trained on TTS. The classification results yielded by the models trained solely using SD were unsatisfactory (*i.e.,* around 0.6 in terms of AUROC). Thus, the class predictions made using SD were not directly fed into the ensemble model. Instead, we concatenated the raw SD into the predictions generated with STS and TTS data to compose the meta-model training and test data. The rationale was to test whether, despite not yielding good predictions on its own, SD could still be useful when combined with other patient data that could potentially provide more context.

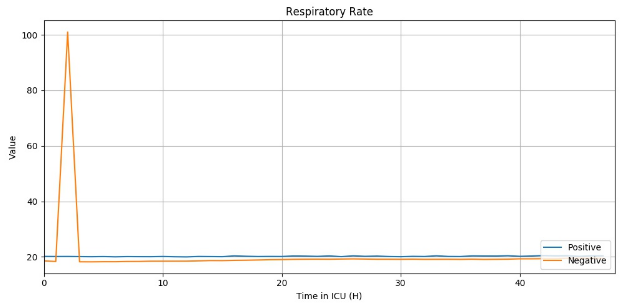
Figure 4.5 illustrates the pipeline used to train our classification models. In (1), the dataset is split into hyperparameter tuning, and model evaluation with each subset accounting for 15/85% of the instances, respectively. The split was done using stratification to maintain the same class distribution in each partition. Hyperparameter tuning (2) takes the hyperparameter tuning dataset and runs the hyperparameter search algorithm for each of the base-models. Then, with these selected hyperparameters, in (3), the classification algorithms are executed on the model evaluation dataset using k-fold cross-validation. In (4), the predictions (*i.e.,* class probabilities) from the trained base-models are used as training data for the meta-model. To avoid data leakage into our meta-model, the training folds is split into p-1 *training folds* to generate the probabilities for the remaining fold. This process is repeated p times, and at the final of this process we have the predictions for all instances for the first training folds, generating our meta-model training data without the leakage problem. In (5) we generate the test data for the meta-models by training the base-algorithms on the training folds and generating the predictions on the test fold. In (6) we evaluate the base-models, and (7) we first generate the hyperparameters by using the grid-search tuning algorithm and evaluate the performance metrics on the test-folds.

The ensemble creation finishes the description of our methodology. Next, we start to evaluate the performance of the algorithms in our data. We will describe the materials used for our experiments, and the result obtained for each algorithm, for each data type, up until the ensembles.

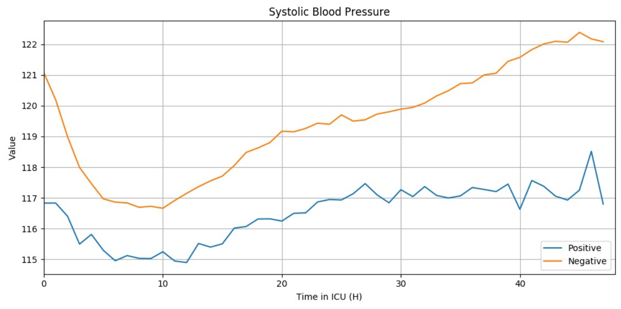
Figure 4.4: Mean values for each hour of a patient's stay for the four variables with the fewest missing values. The differences in values between the positive and the negative class are visible in (a) Heart Rate and (c) Systolic Blood Pressure. Different patterns can also be seen in (d) Diastolic Blood Pressure. However, for (b) Respiratory Rate, the values for both classes are quite close.



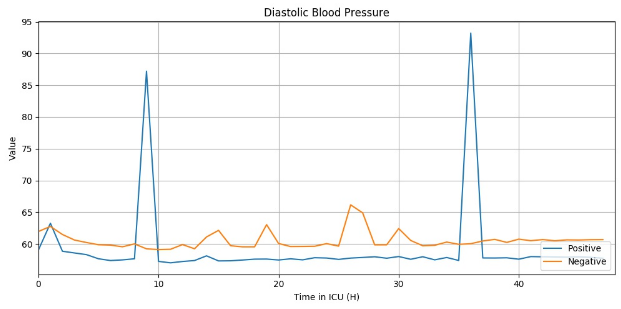
(a) Heart Rate



(b) Respiratory Rate



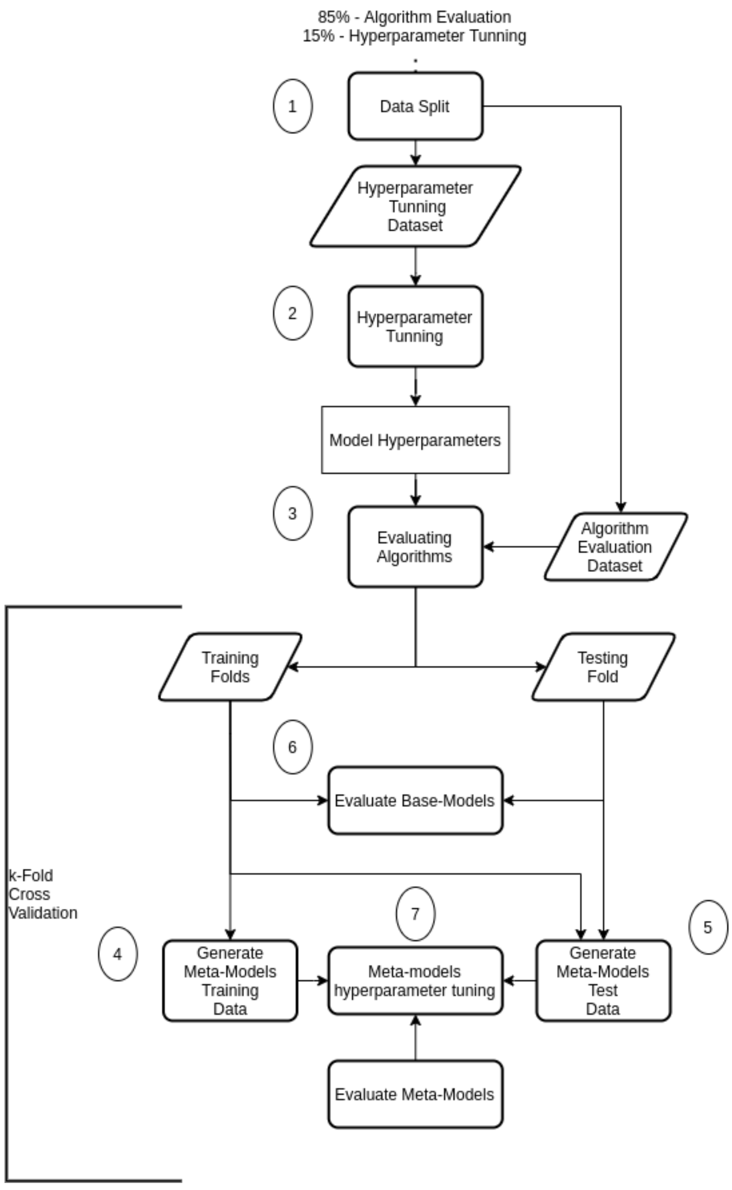
(c) Systolic Blood Pressure



(d) Diastolic Blood Pressure

Source: Author

Figure 4.5: The model training and evaluation pipeline.


85% - Algorithm Evaluation
15% - Hyperparameter Tunning
1
Data Split
Hyperparameter Tunning Dataset
2
Hyperparameter Tunning
Model Hyperparameters
3
Evaluating Algorithms
Algorithm Evaluation Dataset
Training Folds
Testing Fold
6
Evaluate Base-Models
k-Fold Cross Validation
7
5
4
Generate Meta-Models Training Data
Meta-models hyperparameter tuning
Generate Meta-Models Test Data
Evaluate Meta-Models

Source: Author

5 EXPERIMENTS

In this section, we describe the experimental evaluation that we performed aiming at answering the following questions:

1. What type of data provides the best classification performance?
2. Does balancing the classes improve model performance for this task and which data balancing method is best?
3. Does combining different types of data in a heterogeneous ensemble model improve the results of in-hospital mortality prediction?

We start by detailing the tools and resources used, and then the results are presented and discussed.

5.1 Libraries and Frameworks

Our experiments were done in Python. The library Keras was used for training the models, and Pandas was used to manipulate and process the dataset. Scikit Learn was used to compute the evaluation metrics and to generate the meta-models.

Hyperparameter tuning was done using the Hyperband method in the Keras Tuner library. The Keras Tuner searches over random samples of defined values, *e.g.,* we define the hidden units of a layer as a hyperparameter, set a min and a max value for it, and it returns a value in the defined range. The library keeps track of the combinations of hyperparameters used. The min and max values for the hyperparameters were the number of hidden layers between one and four and the units for each layer between eight and 256. For TCN specifically, we varied the kernel size between three and five, the number of dilations between one and four. The use of a dropout was conditioned randomly for each layer and varied between 0.2 and 0.5. We set the loss function to Binary Crossentropy, the hidden layer activation as LeakyRelu, the training optimizer as the Adam optimization algorithm, and the network activation as the Sigmoid function.

The training process follows the Stacking algorithm. The base-models were trained using 5-fold cross-validation and, for each loop, the network output for each instance in each testing fold was concatenated with its respective structured features and fed to the meta-model. After the training data generation process, we create the data that will be used to evaluate the meta-models by training each classification algorithm used in the

Table 5.1: Results for the base-models using 5-fold cross-validation. Best scores in bold.

	Algorithm	Balancing	Data Type	Precision	Recall	W F-Score	U F-Score	TP Rate	AUROC
1	NB	Unbalanced	SD	0.747	0.864	0.801	0.464	0	0.604
2	SVM	Unbalanced	SD	0.809	0.217	0.185	0.211	**0.946**	0.614
3	LR	Unbalanced	SD	0.747	0.864	0.802	0.464	0	0.614
4	LSTM	Unbalanced	STS	0.854	0.872	**0.859**	**0.672**	0.338	0.830
5	TCN	Unbalanced	STS	0.856	**0.876**	0.858	0.661	0.294	**0.831**
6	LSTM	Unbalanced	TTS	0.817	0.844	0.827	0.598	0.231	0.740
7	TCN	Unbalanced	TTS	0.763	0.864	0.802	0.465	0.002	0.659
8	SVM	Cost-Sensitive	SD	0.819	0.413	0.383	0.307	0.689	0.613
9	LR	Cost-Sensitive	SD	0.803	0.554	0.623	0.476	0.622	0.615
10	LSTM	Cost-Sensitive	STS	0.862	0.740	0.777	0.633	0.738	0.813
11	TCN	Cost-Sensitive	STS	0.863	0.743	0.779	0.637	0.744	0.816
12	LSTM	Cost-Sensitive	TTS	0.834	0.713	0.753	0.591	0.615	0.744
13	TCN	Cost-Sensitive	TTS	0.825	0.593	0.649	0.512	0.693	0.689
14	NB	Undersampling	SD	0.802	0.533	0.604	0.464	0.638	0.608
15	SVM	Undersampling	SD	0.807	0.314	0.296	0.276	0.830	0.582
16	LR	Undersampling	SD	0.801	0.551	0.620	0.473	0.616	0.612
17	LSTM	Undersampling	STS	0.854	0.730	0.769	0.621	0.710	0.790
18	TCN	Undersampling	STS	**0.869**	0.728	0.769	0.631	0.794	0.824
19	LSTM	Undersampling	TTS	0.831	0.639	0.693	0.542	0.670	0.712
20	TCN	Undersampling	TTS	0.840	0.578	0.639	0.508	0.757	0.714

base-models in the entire training dataset, generating their predictions and concatenating with the structured features for the ensemble evaluation dataset. For the meta-models, we tested three machine learning algorithms, Support Vector Machine with linear kernel, Logistic Regression, Naive Bayes. In the next section, we discuss our results for 20 base-models, 20 meta-models, and a baseline run.

5.2 Results

In this section, we discuss the results obtained by the models trained, and analyze the impact of the strategies to solve data imbalance, the predictive power of each data type in the meta-models.

5.2.1 Base-models

We start our analysis by looking into the results of the base-models. Table 5.1 shows the results for the evaluation metrics of interest calculated using 5-fold cross-validation for all base-models. The best score for each metric is in bold.

Analyzing the models trained on unbalanced data (lines 1-7 in Table 5.1), we can see that all algorithms, with the exception of SVM (line 2), had difficulties in classifying positive instances (*i.e.,* low *TP rate*). At the same time, using the unbalanced data yields the highest *Recall* (line 5) and *W F-Score* (line 4), indicating an underfitting of the positive class for both. This behavior is expected since the majority population has a bigger influence on the overall error and is thus benefited (CHOULDECHOVA; ROTH, 2018). However, this process was the opposite for SVM, as it had almost a perfect *TP Rate*, with their low *Recall*, indicating the underfitting on the negative class. The STS data type had a better performance with LSTM (line 4), achieving the best score for *W F-Score* and *U F-Score*. This algorithm had a better *TP Rate* than the TCN (line 5) and a difference of 0.001 in *AUROC*. Regarding the TTS data type, the best-trained model was using LSTM (line 6), since the TCN (line 7) had only a 0.2% in *TP Rate*. Comparing LSTM trained on STS (line 4) and on TTS (line 6) data, we notice that the STS model overperformed the TTS model in all metrics, actually the STS using unbalanced data generated the best models.

The models generated using cost-sensitive classification are shown in lines 8 to 13. The NB algorithm could not be used in conjunction with cost-sensitive classification in the libraries used in this work. Here we already see the impact of addressing the data imbalance problem. With the cost-sensitive method, the models trained on SD data type had a better performance compared with models generated with unbalanced data. For LR (line 9), the model was capable of identifying positive instances. Here we see an example of how evaluating AUROC alone is limited – the differences between SVM in line 2 and line 8, and LR in line 3 and line 9 are very small in terms of AUROC, but looking at the other metrics, we see a much wider variation. For the STS and TTS data types, we verify that those models display an increase in *TP Rate* and decrease in *Recall*, indicating that the ability to classify positive instances comes at the cost of misclassifying the negative instance, this is further confirmed analyzing the performance of the undersampling method. On the STS data type, we see that LSTM (line 10) and TCN (line 11) had similar performances, with TCN being slightly better.

For the TTS data type, we see a higher *TP Rate* in favor of TCN (line 13), and a higher *Recall* for LSTM (line 12). The high *TP Rate* and low *Recall* for TCN (line 13) indicate that this model is producing more False Positives, while LSTM (line 12) is returning more False Negatives, given its high *Recall* and low *TP Rate*. For the algorithms trained with the cost-sensitive method, we found that the STS generated better models than the TTS and SD data types.

The last method for addressing class imbalance was undersampling (lines 14-20) the majority class (negative class). It is important to clarify that at every k-fold loop, we used the set of training folds with balanced classes while maintaining the unbalanced distribution in the test fold. Here the SD data (lines 14-16) generated models that were able to learn to distinguish positive instances and classify them correctly. Judging by the higher *TP Rate* in comparison to *Recall*, we see that all methods sacrificed the ability to classify negative instances to be able to classify positive instances. Among the three algorithms, the highest *TP Rate* was obtained by SVM (line 15). SVM is the best performing model for classifying positive instances, however, its low *Recall* is a result of a high number of false positives. For the STS data type, we see a higher performance for TCN (line 18) in comparison to LSTM (line 17). For the TTS data type, we see a similar pattern than the one in the cost-sensitive models (line 12 and 13). However, their similar values in *AUROC* show a similar performance between these models, each favoring a different class.

By analyzing all results obtained using each data type, STS clearly had a better classification performance compared to the other data types. STS achieved higher scores in all class balancing methods. Thus, we conclude that the answer to our first proposed question is that *STS is the data type with the best predictive power*. We could assign this best performance because of the difference between the number of events between STS and TTS. Another fact that can be inferred from the results is that solving the class imbalance problem is very important. This finding is in line with existing works on this topic Monteiro et al. (2020), Hashir and Sawhney (2020), Steinmeyer and Wiese (2020), Caicedo-Torres and Gutierrez (2019). In our experiments, both undersampling and cost-sensitive classification brought noticeable benefits.

Finally, our results show that no algorithm, data type, or strategy for balancing the classes were able to achieve good results in all evaluation metrics. In this sense, the choice of the best model depends on the goals of the task at hand. For in-hospital mortality prediction, the base-model which achieved a good balance across all metrics was TCN trained on STS data using undersampling classification (line 18).

5.2.2 Meta-Models

In this section, we analyze the results obtained for the meta-models. Because of the superior results achieved by the approaches for mitigating the class imbalance problem (cost-sensitive classification and undersampling), we generated ensembles for these two approaches and not for the unbalanced data. Here we performed training using a

Table 5.2: Results for the heterogeneous ensemble models. Best scores in bold.

	Algorithm	Balancing	Data Types	Precision	Recall	W F-Score	U F-Score	TP Rate	AUROC
1	SVM	Cost-Sensitive	STS + TTS	0.870	**0.761**	0.794	0.655	0.764	0.847
2	LR	Cost-Sensitive	STS + TTS	0.871	0.755	0.790	0.652	0.775	0.848
3	SVM	Cost-Sensitive	SD + STS	0.875	0.622	0.678	0.558	0.889	0.842
4	LR	Cost-Sensitive	SD + STS	0.869	0.751	0.786	0.647	0.771	0.841
5	SVM	Cost-Sensitive	SD + TTS	0.861	0.504	0.554	0.462	0.892	0.776
6	LR	Cost-Sensitive	SD + TTS	0.851	0.703	0.748	0.599	0.717	0.776
7	SVM	Cost-Sensitive	SD + STS + TTS	0.878	0.661	0.712	0.588	0.885	**0.855**
8	LR	Cost-Sensitive	SD + STS + TTS	0.874	0.760	0.793	0.657	0.786	**0.855**
9	NB	Undersampling	STS + TTS	0.872	0.752	0.788	0.651	0.788	0.843
10	SVM	Undersampling	STS + TTS	0.874	0.739	0.778	0.643	0.813	0.844
11	LR	Undersampling	STS + TTS	0.873	0.755	0.790	0.654	0.788	0.844
12	NB	Undersampling	SD + STS	0.871	0.759	0.793	0.656	0.775	0.840
13	SVM	Undersampling	SD + STS	0.875	0.666	0.718	0.591	0.869	0.841
14	LR	Undersampling	SD + STS	0.872	**0.761**	**0.795**	0.657	0.775	0.841
15	NB	Undersampling	SD + TTS	0.849	0.627	0.685	0.548	0.771	0.756
16	SVM	Undersampling	SD + TTS	0.864	0.466	0.522	0.436	0.911	0.759
17	LR	Undersampling	SD + TTS	0.850	0.650	0.704	0.564	0.759	0.761
18	NB	Undersampling	SD + STS + TTS	0.873	0.754	0.790	0.653	0.790	0.850
19	SVM	Undersampling	SD + STS + TTS	**0.881**	0.573	0.615	0.518	**0.927**	0.852
20	LR	Undersampling	SD + STS + TTS	0.876	**0.761**	**0.795**	**0.661**	0.800	0.853

combination of the outputs to analyze which one had the best performance. As described in Section 4.2.3, for STS and TTS data, the predictions from both LSTM and TCN models were used. For the SD data, the raw data was used. The results are in Table 5.2.

The first information that we can extract from the meta-models results is that every combination produced a higher *AUROC* when compared with the models trained in individual data. Analyzing the results by combination, we that for both balancing methods, combining SD + STS + TTS produced the best results. For the SVM algorithm using cost-sensitive, the SD + STS + TTS (line 7) had a worse *TP Rate* than the models with SD + STS (line 3) and SD + TTS (line 5). However, its performance in all other metrics was higher, resulting in a better performance on the negative instances. For the LR, SD + STS + TTS (line 8) had the best performance in all metrics compared with the other combinations with the same algorithm. With the undersampling method, the SVM model decayed its *Recall* after adding the SD data in the combinations. So, the best result obtained by this specific model was using STS + TTS. For the NB algorithm, the SD + STS + TTS had the best values in almost all metrics, with the exception of *W F-Score* and *U F-Score*. For LR, the SD + STS + TTS had the best results in all metrics and is the best model generated in all our tests, achieving the highest *Recall*, *W F-Score*, *U F-Score*, and high performance on *TP Rate*, *AUROC*, and *Precision*.

Comparing the results of the different classification algorithms on the ensembles, we see that SVM is still associated with the highest *TP Rate* (line 19) and the lowest *Recall*, *U F-Score*, and *W F-Score* (lines 3, 5, 7, 13, 16 and 19). Both LR and NB had similar results when comparing both using the same combination of data, with the LR having the higher values. By analyzing the performances, we conclude that if the end goal is to prioritize the identification of the most positive instances, the SVM algorithm is the best, but at cost of performance negative instances, which produce a high volume of false positives. However, if the goal is to maintain a good balance between the performance on positive and negative instances, the LR algorithm is the best option.

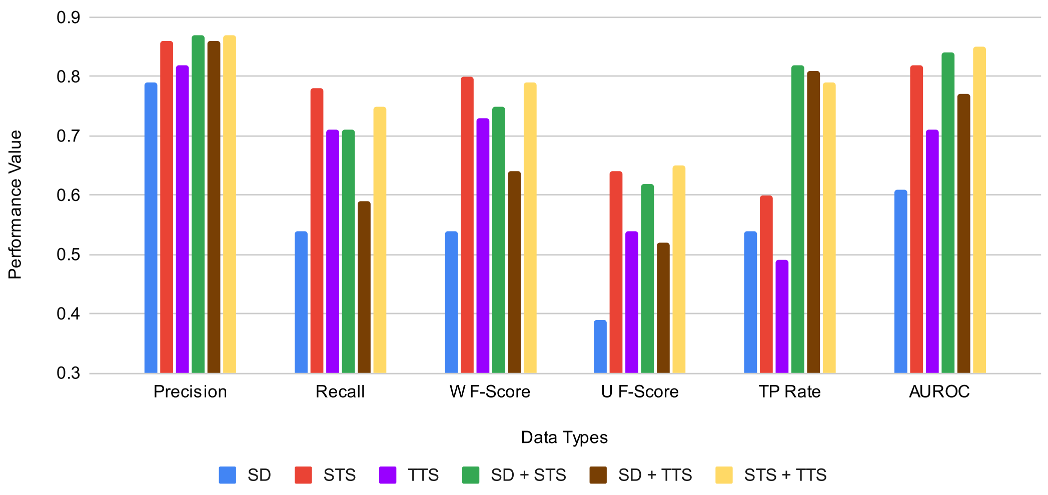
Looking at the strategies for dealing with the class imbalance problem, both strategies have similar results for *Precision* and *AUROC*, when analyzing the SD + STS + TTS. Undersampling yielded better results in all metrics, except for *AUROC*, but the difference was as low as 0.003.

Figure 5.1 shows the results for each data type used on their own and in heterogeneous combinations. The scores of each evaluation metric were averaged for all experimental runs in which the data type (or combination) was applied (the runs with unbalanced data were discarded from this evaluation to allow a fir comparison). The combination of all three types of data yielded the best *Precision*, *TP Rate*, and *AUROC*. In terms of the other metrics, the scores obtained with this heterogeneous ensemble were always close to the top performers. The metric that most benefited from the combination of data types was *TP rate*. This can be clearly seen in Figure 5.1 as the four bars that correspond to the ensembles are much higher than the three bars that correspond to the types used in isolation. Overall, we can see that using a single data type tended to produce the poorest results, with SD on its own being clearly the worst performer. Even STS, which had good results in most metrics, yielded a low *TP Rate* compared to the ensembles. Despite the poor performance of SD, the use of this type of data brought improvements in terms of *Precision*, *TP Rate*, and *AUROC* when used in combination with STS and TTS.

With the findings from our analyzes, we can answer the remaining research questions. Regarding our second question – *Does balancing the classes improve model performance for this task and which data balancing method is best?* Our results showed that addressing the class imbalance problem played a big role in model performance, noticeably improving the quality metrics. While cost-sensitive classification had the best results in the base-models, in the meta-models the performances obtained on the undersampled dataset were similar. Thus, we conclude that both strategies for balancing the classes could be used in the heterogeneous ensembles.

The answer to our third question – *Does combining different types of data in a heterogeneous ensemble model improve the results of in-hospital mortality prediction?* is *yes*. Our results showed that the models combining the three types of data achieved the

Figure 5.1: Evaluation Metrics for the different types of data and heterogeneous combinations



Source: Author

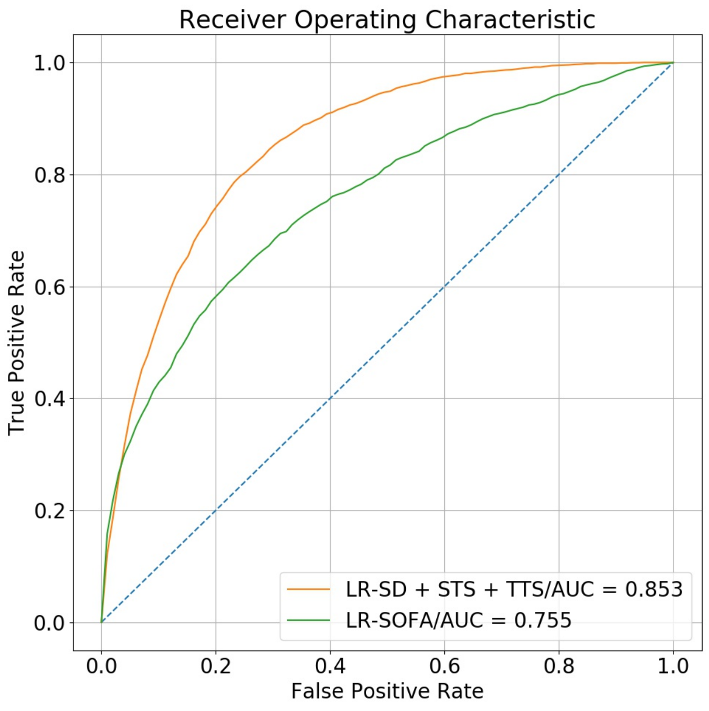
best *Precision*, *U F-Score*, and *AUROC* while maintaining a good performance across the other metrics.

5.2.3 Comparison with a Baseline

In this section, we compare the performance of our heterogeneous ensemble for in-hospital mortality prediction against a baseline classifier that relies on the SOFA score. The intuition for using SOFA as a baseline is to obtain a classification model based on a popular score that is widely applied in predicting the clinical outcomes of critically ill patients. To be comparable with our experimental runs, in which only the first 48 hours of data are considered, our baseline used the hourly SOFA scores for this time window. Also, this run was done using the undersampling method. With the extracted hourly SOFA scores for each patient, we extracted time-series features for SOFA values for each ICU stay using the Python package *tsfresh* (CHRIST et al., 2018; CHRIST; KEMPA-LIEHR; FEINDT, 2016). Then, the same package was used to select only the relevant features for our classification problem. Next, we ran a Logistic Regression classifier on these features.

The heterogeneous ensemble used in the comparisons in this section combines all three data types using a LR classifier. Its details are in line 20 in Table 5.2. Figure 5.2 shows the ROC curves for our heterogeneous ensemble and for the SOFA baseline. We can see that the baseline had a lower performance achieving an *AUROC* of 0.755 (95% CI [0.744 0.767]) and was outperformed by our ensemble model which had an *AUROC*

Figure 5.2: ROC curves for a heterogeneous ensemble that combines all data three types and for the baseline SOFA model.



Source: Author

of 0.853 (95% CI [0.846, 0.861]).

As reported in Section 3.1, previous works using SOFA for mortality prediction have identified AUROC results ranging from 0.61 to 0.87. Our scores are within that range. In relation to the work by Jentzer et al. (2018), the differences that may explain the lower scores found here are our larger number of instances (we use twice as many patients) and the fact that the patients in the aforementioned work were from a single cardiac ICU and here the patients come from five different ICUs and from a wider spectrum.

Finally, in Table 5.3, we inspect the confusion matrices generated for the SOFA model and for the same heterogeneous ensemble (LR-SD+STS+TTS). The heterogeneous ensemble was able to correctly classify an additional 368 patients into the positive class (an increase of 25%) and it was also better than SOFA at identifying the negative class – with 169 additional patients being correctly classified.

5.3 Model Analysis

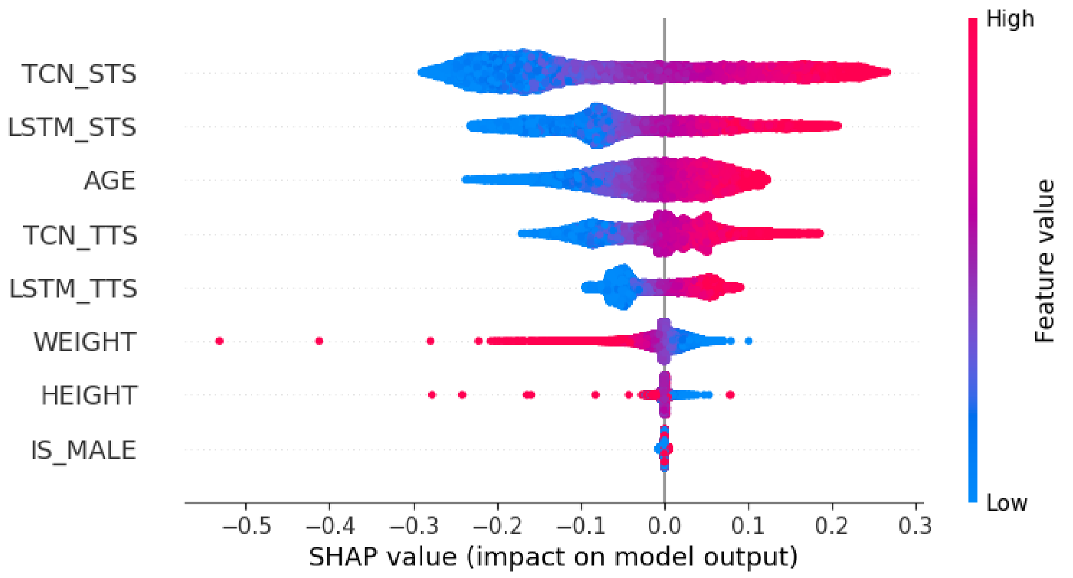
In this section, we further analyze the results obtained by the LR model trained with undersampling, our best generated model. We start with the analysis of the features

Table 5.3: Confusion matrices for the baseline SOFA run and the heterogeneous ensemble

SOFA				Heterogeneous Ensemble			
		Predicted				Predicted	
		1	0			1	0
Actual	1	1484	831	Actual	1	1852	463
	0	3786	10970		0	3617	11139

used by the model. For this analysis, we use the SHAP values to understand the impact of each feature in the model output. The values are in Figure 5.3. These values were generated by the predictions for each instance in the dataset, respecting the k-fold process. The chart is made with dots, where each dot represents an instance. The width of the line represents the distribution of SHAP value in respect of feature value, and the colors reflect the values of the features.

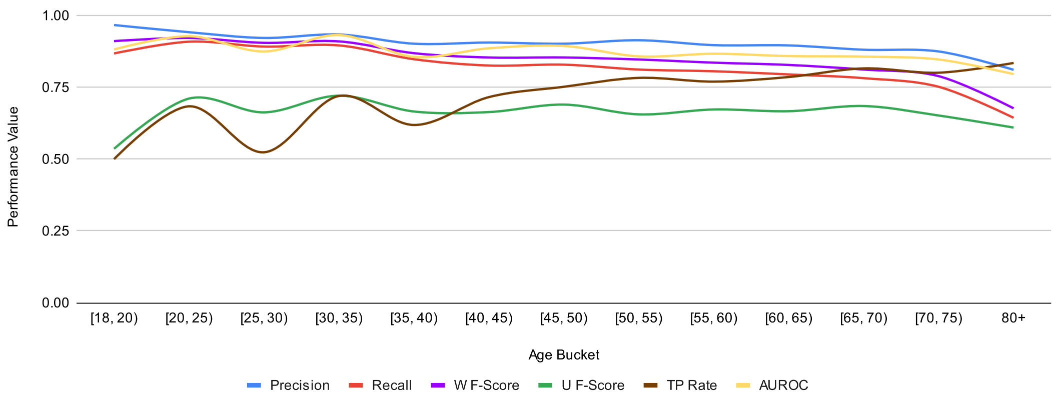
Figure 5.3: Shap values for the predictions generated by the algorithm.



Source: Author

Looking at the figure, we see that the predictions generated by models trained with the STS data have the highest impact on the ensemble output. The TCN predictions were the highest placed feature and the best base-model with undersampled data. The third feature with the highest impact is *age*. With this fact, we measure the predictive performance of our model in relation to this feature, by aggregating patients into age groups [18, 20), [20, 25), [25, 30), [30, 35), [35, 40), [40, 45), [45, 50), [50, 55), [55, 60), [60, 65), [65, 70), [70, 75), 80+. The group of 80+ is because, for the sake of anonymity, MIMIC-III groups all ages above 80. The results can be seen in Figure 5.4. We can see that, as age increases, there is a decrease in *Precision*, *Recall*, *W F-Score*, and *AUROC*. However, this is inverse for the *TP Rate*. For our model, a higher age is indicative of a higher in-hospital mortality risk. This is compatible with the chart in Figure 5.3 as higher values in age had a positive SHAP value.

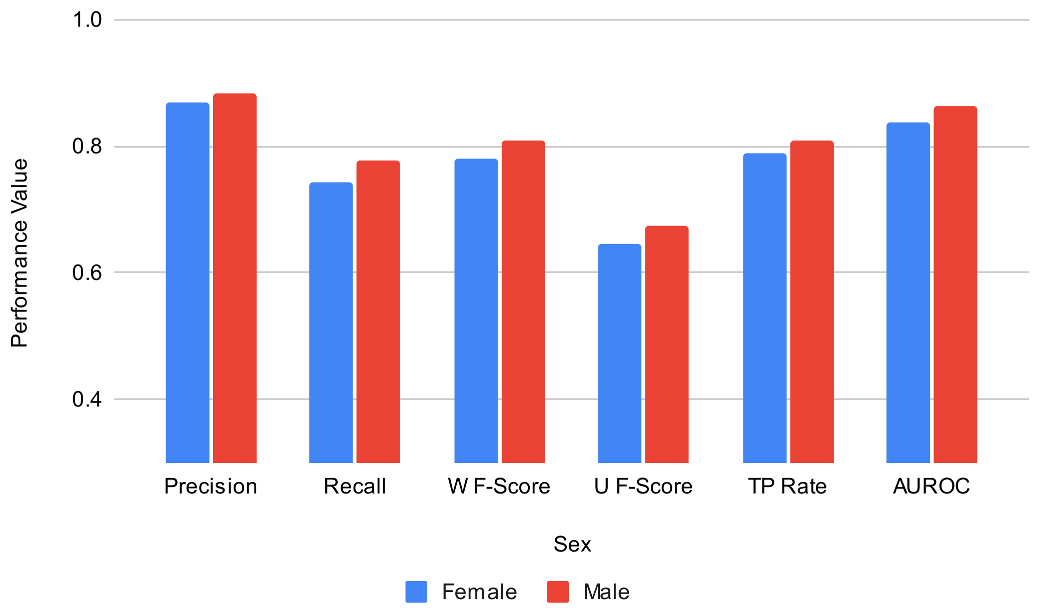
Figure 5.4: Model performance for age feature.



Source: Author

Another feature that we looked into is the patient's sex ("IS_MALE"). This feature did not have any impact on the model output. Figure 5.5 shows model performance for both male and female patients. It shows very similar performances between these groups.

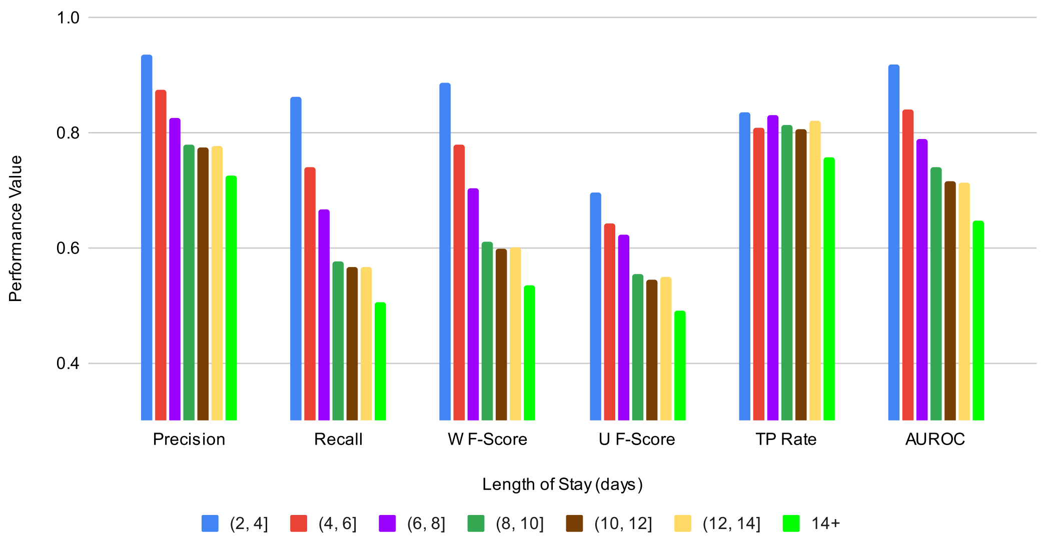
Figure 5.5: Model performance by sex.



Source: Author

Other variables that we could analyze to understand the performance of our model are: length of stay (LOS), and care unit. LOS is a variable that should affect the performance of our model, as patients who died had a higher LOS (see Table 4.1). Since we only use data from the first 48h of stay, a longer duration of a stay could implicate in a variation of the patient's condition. Figure 5.6 shows the metrics for LOS ranges. These ranges were the same used in the analysis made by (HARUTYUNYAN et al., 2019). In the chart, we see a similar behavior that appeared for our age analysis, in which every metric, with the exception of *TP Rate*, presented a decrease when the LOS increases. Here, the *TP Rate* oscillated for every range. However, it presented the lowest value with 14+ days stay.

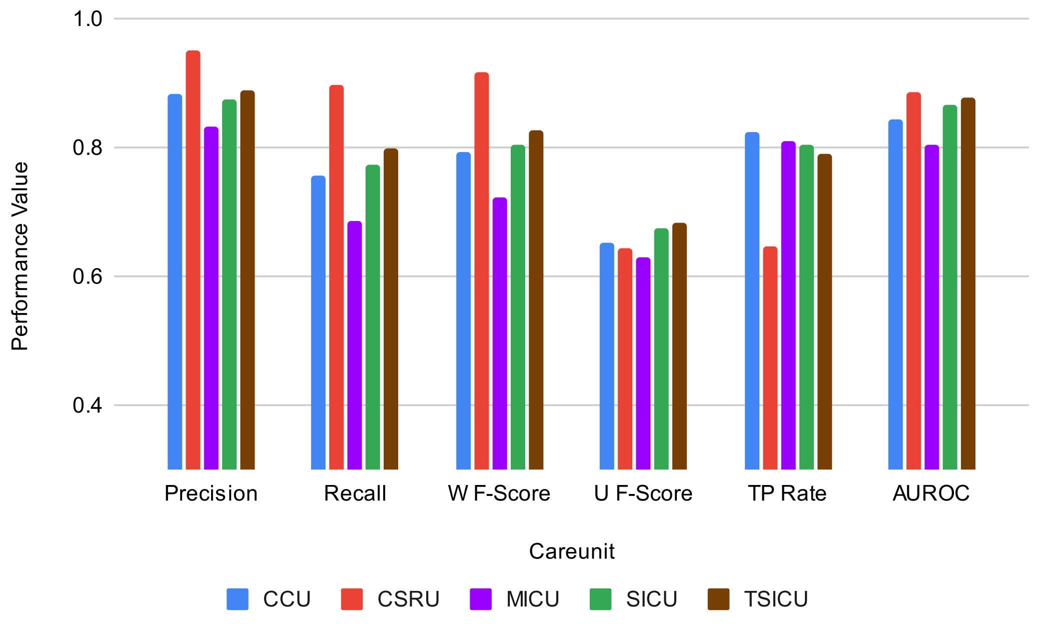
Figure 5.6: Model performance by length of stay, in days.



Source: Author

Our data comes from patients that were admitted to different care units, namely the Coronary care unit (CCU), Cardiac surgery recovery unit (CSRU), Medical intensive care unit (MICU), Surgical intensive care unit (SICU), and Trauma/surgical intensive care unit (TSICU). Figure 5.7 shows the performance of the models for each of those units. We can see that mortality prediction on the CSRU had the lowest *TP Rate* over all other care units. This could be explained by the fact that only 5.43% of diseased patients were from this unit. CCU and MICU had similar performance results for *TP Rate*. Even though MICU corresponds to 40.17% of the total patient population, and 54.19% of patients who died during their stay, it had a similar performance as the CCU. That could be a consequence of the more diverse set of patient conditions found in a general-purpose unit like MICU, which could make the prediction task harder. While the CCU has a smaller population than MICU, it had a similar performance, which could be explained by the specificity of patient condition in this care unit.

Figure 5.7: Model performance by care unit – Coronary care unit (CCU), Cardiac surgery recovery unit (CSRU), Medical intensive care unit (MICU); Surgical intensivecare unit (SICU), and Trauma/surgical intensive care unit (TSICU).



Source: Author

To sum up these analyzes, we saw that the base-model predictions had a high influence on the meta-model prediction. Age was the third most relevant feature and had a positive impact in identifying in-hospital mortality as the patients are older. Patient's sex had no impact on the prediction, and we can see both in Figure 5.3 with the SHAP values and Figure 5.4 that show the model performance by sex. For the care unit, the size of the population in the unit does not yield better performance, as we can see by comparing MICU vs CCU.

With these analyzes, we wanted to get a deeper understanding of our model predictions and identify the influences of the variables in our proposed task. By identifying these factors, we can fine-tune our model in the future to improve its performance, or simply identify its weaknesses when using it in a real-world scenario.

6 CONCLUSION

This work investigated the use of ensemble models built using heterogeneous types of data (structured, text, and time series) for the task of in-hospital mortality prediction. We carried out a series of experiments using data from MIMIC-III. Our dataset had 20K hospital stays and was very unbalanced – the positive class corresponds to 13.5% of the instances.

We designed a methodology to process each type of data. First, base-models were created for each type on its own, and then the different data types were combined in meta-models using ensemble learning with the stacking strategy. We experimented with different classification algorithms and methods for dealing with the class imbalance problem. Our results were evaluated using six classification quality metrics.

Looking at the results of the base-models, we concluded that, among the individual data types, structured time series provided the best classification models. When the heterogeneous meta-models were considered, we verified that the use of different types of data brings important gains in classification quality. We also found that addressing the class imbalance problem by either using cost-sensitive classification or undersampling the majority class noticeably improves the classification models.

The results obtained in this work can be used as a base for future studies using heterogeneous data types. Here we applied an ensemble methodology, in which models could be trained with specific types of data without influencing each other, and achieved an increase in performance with the meta-model. This leaves an open question as to whether it is possible to obtain performance improvements by combining the heterogeneous data types in some other manner, *e.g.,* in a joint training method such as the one by Hashir and Sawhney (2020). With an end-to-end approach for constructing our models, we could remove the workload that the Stacking algorithm incurs by training a large number of models to create the data for the meta-model. Another consequence of this approach is that data types would share information between them during model training, which could be beneficial to enhance the utility of each data type. Another future work could involve testing whether applying other methodologies to handle data types (or even a combination of methodologies) could mitigate the inability of some models in handling handle specific cases. Finally, our base-models relied on deep learning algorithms that typically achieve state-of-the-art results in many tasks but they behave as black-boxes. In future work, we plan to explore explainable models following, for example, the approach by Caicedo-Torres and Gutierrez (2019).

An article describing the results of our research is currently under review in the Journal of Biomedical Informatics.

REFERENCES

BAI, S.; KOLTER, J. Z.; KOLTUN, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. **arXiv preprint arXiv:1803.01271**, 2018.

BARCELOS, H.; MENDOZA, M.; MOREIRA, V. Identifying and fusing duplicate features for data mining. In: **Anais do XXXV Simpósio Brasileiro de Bancos de Dados**. Porto Alegre, RS, Brasil: SBC, 2020. p. 133–144. ISSN 0000-0000. Available from Internet: <https://sol.sbc.org.br/index.php/sbbd/article/view/13631>.

BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. **IEEE Transactions on Neural Networks**, v. 5, n. 2, p. 157–166, Mar 1994. ISSN 1045-9227.

BERGSTRA, J.; CA, J. B.; CA, Y. B. **Random Search for Hyper-Parameter Optimization**. [S.l.], 2012. v. 13, 281–305 p. Available from Internet: <http://scikit-learn.sourceforge.net.>

BRITZ, D. **Understanding Convolutional Neural Networks for NLP**. 2015. Available from Internet: <http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/>.

CAICEDO-TORRES, W.; GUTIERREZ, J. ISeeU: Visually interpretable deep learning for mortality prediction inside the ICU. **Journal of Biomedical Informatics**, Elsevier, v. 98, p. 103269, 2019.

CHOULDECHOVA, A.; ROTH, A. The frontiers of fairness in machine learning. **arXiv preprint arXiv:1810.08810**, 2018.

CHRIST, M. et al. Time series feature extraction on basis of scalable hypothesis tests (tsfresh – a python package). **Neurocomputing**, v. 307, p. 72–77, 2018. ISSN 0925-2312.

CHRIST, M.; KEMPA-LIEHR, A. W.; FEINDT, M. Distributed and parallel time series feature extraction for industrial big data applications. **arXiv preprint arXiv:1610.07717**, 2016.

CORTES, C.; VAPNIK, V. Support-vector networks. **Mach. Learn.**, Kluwer Academic Publishers, USA, v. 20, n. 3, p. 273–297, sep. 1995. ISSN 0885-6125. Available from Internet: <https://doi.org/10.1023/A:1022627411411>.

DAVOODI, R.; MORADI, M. H. Mortality prediction in intensive care units (ICUs) using a deep rule-based fuzzy classifier. **Journal of Biomedical Informatics**, Elsevier, v. 79, p. 48–59, 2018.

FEURER, M.; HUTTER, F. Hyperparameter optimization. In: **Automated machine learning**. [S.l.]: Springer, Cham, 2019. p. 3–33.

GERS, F. A.; SCHMIDHUBER, J. A.; CUMMINS, F. A. Learning to forget: Continual prediction with lstm. **Neural Comput.**, MIT Press, Cambridge, MA, USA, v. 12, n. 10, p. 2451–2471, oct. 2000. ISSN 0899-7667. Available from Internet: <http://dx.doi.org/10.1162/089976600300015015>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016.

HARRINGTON, P. **Machine Learning in Action**. USA: Manning Publications Co., 2012. ISBN 1617290181.

HARUTYUNYAN, H. et al. Multitask learning and benchmarking with clinical time series data. **Scientific Data**, Springer Science and Business Media LLC, v. 6, n. 1, Jun 2019. ISSN 2052-4463.

HASHIR, M.; SAWHNEY, R. Towards unstructured mortality prediction with free-text clinical notes. **Journal of Biomedical Informatics**, Elsevier, v. 108, p. 103489, 2020.

HAYKIN, S. **Redes Neurais - 2ed.** BOOKMAN COMPANHIA ED, 2001. ISBN 9788573077186. Available from Internet: <https://books.google.com.br/books?id=lBp0X5qfyjUC>.

HO, K. et al. Comparison of acute physiology and chronic health evaluation (APACHE) II score with organ failure scores to predict hospital mortality. **Anaesthesia**, v. 62, p. 466–73, 06 2007.

HOCHREITER, S. et al. **Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies**. 2001.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Comput.**, v. 9, n. 8, p. 1735–1780, nov. 1997. ISSN 0899-7667.

HSIEH, Y.-Z. et al. Prediction of survival of icu patients using computational intelligence. **Computers in biology and medicine**, Elsevier, v. 47, p. 13–19, 2014.

JAMIESON, K.; TALWALKAR, A. Non-stochastic best arm identification and hyperparameter optimization. In: PMLR. **Artificial Intelligence and Statistics**. [S.l.], 2016. p. 240–248.

JENTZER, J. et al. Predictive value of the sequential organ failure assessment score for mortality in a contemporary cardiac intensive care unit population. **Journal of the American Heart Association**, v. 7, p. e008169, 03 2018.

JOHNSON, A. E.; POLLARD, T. J.; MARK, R. G. Reproducibility in critical care: a mortality prediction case study. In: PMLR. **Machine Learning for Healthcare Conference**. [S.l.], 2017. p. 361–376.

JOHNSON, A. E. et al. MIMIC-III, a freely accessible critical care database. **Scientific data**, Nature Publishing Group, v. 3, p. 160035, 2016.

KANE, R. et al. The association of registered nurse staffing levels and patient outcomes: Systematic review and meta-analysis. **Medical care**, v. 45, p. 1195–204, 01 2008.

LE, Q.; MIKOLOV, T. Distributed representations of sentences and documents. In: PMLR. **International conference on machine learning**. [S.l.], 2014. p. 1188–1196.

LEA, C. et al. Temporal convolutional networks: A unified approach to action segmentation. In: SPRINGER. **European Conference on Computer Vision**. [S.l.], 2016. p. 47–54.

LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, Nov 1998. ISSN 0018-9219.

LEHMAN, L.-w. et al. Risk stratification of ICU patients using topic models inferred from unstructured progress notes. In: AMERICAN MEDICAL INFORMATICS ASSOCIATION. **AMIA annual symposium proceedings**. [S.l.], 2012. v. 2012, p. 505.

LI, L. et al. Hyperband: A novel bandit-based approach to hyperparameter optimization. **The Journal of Machine Learning Research**, JMLR. org, v. 18, n. 1, p. 6765–6816, 2017.

LUNDBERG, S. M.; LEE, S.-I. A unified approach to interpreting model predictions. In: GUYON, I. et al. (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2017. v. 30. Available from Internet: <https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf>.

MARAFINO, B. J. et al. Validation of prediction models for critical care outcomes using natural language processing of electronic health record data. **JAMA network open**, American Medical Association, v. 1, n. 8, p. e185097–e185097, 2018.

MBONGO, C.-L. et al. Performance of saps3, compared with APACHE II and SOFA, to predict hospital mortality in a general ICU in southern europe. **European journal of anaesthesiology**, v. 26, p. 940–5, 08 2009.

MIKOLOV, T. et al. **Efficient Estimation of Word Representations in Vector Space**. 2013. Available from Internet: <http://arxiv.org/abs/1301.3781>.

MINNE, L.; ABU-HANNA, A.; JONGE, E. de. Evaluation of SOFA-based models for predicting mortality in the ICU: A systematic review. **Critical care**, BioMed Central, v. 12, n. 6, p. 1–13, 2008.

MONTEIRO, F. et al. Prediction of mortality in intensive care units: a multivariate feature selection. **Journal of Biomedical Informatics**, Elsevier, v. 107, p. 103456, 2020.

PIRRACCHIO, R. et al. Mortality prediction in the ICU: can we do better? Results from the Super ICU Learner Algorithm (SICULA) project, a population-based study. 2015.

POLIKAR, R. Ensemble based systems in decision making. **IEEE Circuits and systems magazine**, IEEE, v. 6, n. 3, p. 21–45, 2006.

REDFERN, O. et al. Predicting in-hospital mortality and unanticipated admissions to the intensive care unit using routinely collected blood tests and vital signs: Development and validation of a multivariable model. **Resuscitation**, v. 133, 09 2018.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3rd. ed. USA: Prentice Hall Press, 2009. ISBN 0136042597.

SAEED, M. et al. Multiparameter intelligent monitoring in intensive care II MIMIC-II: A public-access intensive care unit database. **Critical care medicine**, v. 39, p. 952–60, 05 2011.

SHAPLEY, L. S. **A value for n-person games**. [S.l.]: Princeton University Press, 2016.

SHIMABUKURO, D. et al. Effect of a machine learning-based severe sepsis prediction algorithm on patient survival and hospital length of stay: A randomised clinical trial. **BMJ Open Respiratory Research**, v. 4, p. e000234, 11 2017.

SILVA, K. D. et al. Clinical notes as prognostic markers of mortality associated with diabetes mellitus following critical care: A retrospective cohort analysis using machine learning and unstructured big data. **Computers in Biology and Medicine**, Elsevier, v. 132, p. 104305, 2021.

SONG, H. et al. Attend and diagnose: Clinical time series analysis using attention models. In: **Thirty-second AAAI conference on artificial intelligence**. [S.l.: s.n.], 2018.

STEINMEYER, C.; WIESE, L. Sampling methods and feature selection for mortality prediction with neural networks. **Journal of Biomedical Informatics**, Elsevier, v. 111, p. 103580, 2020.

SUNDARARAJAN, M.; NAJMI, A. The many shapley values for model explanation. **CoRR**, abs/1908.08474, 2019. Available from Internet: <http://arxiv.org/abs/1908.08474>.

SUSHIL, M. et al. Patient representation learning and interpretable evaluation using clinical notes. **Journal of Biomedical Informatics**, v. 84, p. 103–113, 2018. ISSN 1532-0464.

TING, K. M.; WITTEN, I. H. Issues in stacked generalization. **Journal of Artificial Intelligence Research**, Morgan Kaufmann Publ Inc, v. 10, p. 271–289, jan 1999. ISSN 10769757.

VINCENT, J.-L. et al. The SOFA (sepsis-related organ failure assessment) score to describe organ dysfunction/failure. **Intensive care medicine**, v. 22, p. 707–10, 08 1996.

WEISSMAN, G. E. et al. Inclusion of unstructured clinical text improves early prediction of death or prolonged ICU stay. **Critical Care Medicine**, Lippincott Williams and Wilkins, v. 46, n. 7, p. 1125–1132, 2018. ISSN 15300293.

WOLPERT, D. Stacked generalization. **Neural Networks**, v. 5, p. 241–259, 12 1992.

APPENDIX A — RESUMO EXPANDIDO EM PORTUGUÊS: PREDIÇÃO DE MORTES DE INTERNADOS EM UTIS COM MODELOS ENSEMBLE HETEROGÊNEOS

Para um paciente, um maior tempo de internação em uma UTI resulta em uma maior probabilidade de sequelas e até maior mortalidade, assim como pode pressionar o sistema de saúde já que faz o uso de recursos limitados por maior tempo. Uma forma de mitigar esses problemas é a detecção antecipada da deterioração do estado de saúde do paciente durante sua internação. Isso pode ser feito através de sistemas automatizados de monitoramento e modelos de aprendizado de máquina. Com o crescimento da adoção de prontuários eletrônicos e da acessibilidade da comunidade a esses dados, a área de aprendizado de máquina está fazendo o uso desses dados para a solução de uma vasta gama de problemas. Esses dados são ricos e complexos, e contam com uma diversidade grande de tipos de dados, como dados estruturados (*e.g.,* dados demográficos), texto livre (*e.g.,* exames e prontuário médico) e dados temporais (*e.g.,* medições de sinais vitais).

Neste trabalho, buscamos explorar essa diversidade de tipos de dados para a tarefa de predição de mortalidade durante a estadia no hospital. Mais especificamente, usando apenas a janela de dados das primeiras 48h de estadía do paciente. Contruímos modelos de classificação para essa tarefa para cada tipo de dado existente na nossa base de dados, aplicando algoritmos do estado-da-arte da área de *deep learning*. Usando o resultado da classificação obtido por esses modelos, modelos *ensemble* foram treinados. Com isso, é possível avaliar se esses modelos conseguem tentar melhorar qualidade da classificação.

A base de dados analisada foi extraída da base de dados aberta MIMIC-III, que possui um total de 61.532 pacientes. Após o processo de extração, o conjunto de pacientes final ficou com um total de 20.083 pacientes. Desses, 17.359 pacientes saíram da UTI com vida (instâncias negativas) e 2.724 pacientes morreram (instâncias positivas). Essa distribuíção de pacientes entre classes forma uma base de dados desbalanceada.

Para os tipos de dados, temos: dados estruturados (DE), dados estruturados temporais (DET), dados textuais temporais (DTT). Para o DE, usamos o sexo biológico, a altura, o peso e a idade. Para os DET, usamos 24 variáveis diferentes. Tanto os DET como os DTT foram agregados por hora de medição, no qual o textos foram concatenados e dos dados estruturados foi extraída a média. Para o pré-processamento dos DET, os valores faltantes foram preenchidos usando a média global dos dados usados para treinamento e para o DTT, foi usado um modelo Doc2Vec, treinado em cima de todas a notas médicas dos pacientes do MIMIC-III, de forma a poder transformar os textos para um representação vetorial, mantendo-se a dimensão temporal dos dados.

O algoritmo de ensemble usado pela nossa metodologia é o *Stacking*. Nesse algoritmo, os modelos são treinados em camadas, de modo que na primeira camada temos os modelos chamados de *base-models*, e segunda camada é chamada de *meta-models*. Nesse algoritmo, são treinados os *base-models* para que sejam usados para gerar predições em cima das instâncias na base de dados e gerar os dados para os *meta-models*. Os algoritmos de aprendizado de máquina usados para os *base-models* foram *Long Short Term Memory* (LSTM) e *Temporal Convolutional Network* (TCN) para ambos os tipos de dados temporais. Para os *meta-models*, foram usados os algoritmos de aprendizado Naive Bayes, Regressão Logística e *Support Vector Machines* (SVM), esses algoritmos também foram avaliados usando apenas os dados estruturados, porém não obtiveram bons resultados e não foram utilizados como *base-models*.

Um dos problemas do treinamento utilizando *Stacking* é evitar o vazamento de informações dos dados, ou seja, evitar que durante a criação dos dados de treinamento dos *meta-models*, na hora de gerar a predição de uma instância, ela não tenha sido usada durante o treinamento do *base-models*. Para isso foi utilizado dois k-fold aninhados, de forma que o primeiro é utilizado para avaliação dos modelos, e o segundo é feito apenas em cima dos folds de treinamento do primeiro. No segundo k-fold, o fold de teste é transformado pelo modelo treinado com os folds de treinamento, formando ao final os dados para treinamento dos *meta-models*.

Nosso modelos foram avaliados por seis métricas, e o resultado final dos testes mostrou uma melhora de performance final em três pontos percentuais usando o ensemble, obtendo o melhor resultado de AUROC de 0,853 (95% IC [0,846; 0,861]).

APPENDIX B — BASE MODELS STANDARD DEVIATION

In Table B.1 we have the standard deviation values obtained for each metric evaluated in our tests, for each base-model generated and balancing method tested.

Table B.1: Standard deviation from the results for the base-models using 5-fold cross-validation.

	Algorithm	Balancing	Data Type	Precision	Recall	W F-Score	U F-Score	TP Rate	AUROC
1	NB	Unbalanced	SD	0	0	0	0	0	0.006
2	SVM	Unbalanced	SD	0.025	0.07	0.127	0.077	0.045	0.006
3	LR	Unbalanced	SD	0	0	0	0	0	0.006
4	LSTM	Unbalanced	STS	0.004	0.004	0.003	0.006	0.015	0.009
5	TCN	Unbalanced	STS	0.004	0.004	0.003	0.014	0.041	0.008
6	LSTM	Unbalanced	TTS	0.009	0.007	0.007	0.021	0.048	0.011
7	TCN	Unbalanced	TTS	0.035	0	0.001	0.004	0.004	0.096
8	SVM	Cost-Sensitive	SD	0.05	0.314	0.336	0.169	0.431	0.009
9	LR	Cost-Sensitive	SD	0.003	0.003	0.002	0.003	0.014	0.005
10	LSTM	Cost-Sensitive	STS	0.007	0.035	0.026	0.023	0.066	0.01
11	TCN	Cost-Sensitive	STS	0.006	0.038	0.028	0.023	0.07	0.028
12	LSTM	Cost-Sensitive	TTS	0.008	0.03	0.022	0.016	0.066	0.018
13	TCN	Cost-Sensitive	TTS	0.021	0.127	0.119	0.088	0.075	0.078
14	NB	Undersampling	SD	0.007	0.006	0.006	0.004	0.036	0.01
15	SVM	Undersampling	SD	0.052	0.22	0.286	0.169	0.237	0.072
16	LR	Undersampling	SD	0.006	0.009	0.008	0.008	0.026	0.011
17	LSTM	Undersampling	STS	0.011	0.032	0.024	0.023	0.072	0.03
18	TCN	Undersampling	STS	0.003	0.025	0.02	0.02	0.018	0.01
19	LSTM	Undersampling	TTS	0.012	0.053	0.046	0.026	0.101	0.012
20	TCN	Undersampling	TTS	0.013	0.077	0.064	0.038	0.135	0.017

APPENDIX C — META MODELS STANDARD DEVIATION

In Table C.1 we have the standard deviation values obtained for each metric evaluated in our tests, for each meta-model generated and balancing method tested.

Table C.1: Standard deviation from the results for the meta-models using 5-fold cross-validation.

	Algorithm	Balancing	Data Type	Precision	Recall	W F-Score	U F-Score	TP Rate	AUROC
1	SVM	Cost-Sensitive	STS + TTS	0.005	0.032	0.024	0.02	0.058	0.006
2	LR	Cost-Sensitive	STS + TTS	0.005	0.029	0.021	0.017	0.058	0.006
3	SVM	Cost-Sensitive	SD + STS	0.003	0.078	0.071	0.056	0.047	0.003
4	LR	Cost-Sensitive	SD + STS	0.005	0.025	0.019	0.016	0.05	0.003
5	SVM	Cost-Sensitive	SD + TTS	0.007	0.148	0.173	0.12	0.05	0.021
6	LR	Cost-Sensitive	SD + TTS	0.009	0.029	0.022	0.016	0.068	0.022
7	SVM	Cost-Sensitive	SD + STS + TTS	0.002	0.076	0.069	0.056	0.046	0.006
8	LR	Cost-Sensitive	SD + STS + TTS	0.006	0.027	0.02	0.017	0.06	0.006
9	NB	Undersampling	STS + TTS	0.001	0.008	0.006	0.006	0.01	0.005
10	SVM	Undersampling	STS + TTS	0.003	0.005	0.004	0.004	0.015	0.007
11	LR	Undersampling	STS + TTS	0.002	0.011	0.008	0.007	0.02	0.006
12	NB	Undersampling	SD + STS	0.004	0.02	0.015	0.015	0.028	0.006
13	SVM	Undersampling	SD + STS	0.003	0.058	0.051	0.042	0.036	0.008
14	LR	Undersampling	SD + STS	0.004	0.019	0.015	0.015	0.027	0.008
15	NB	Undersampling	SD + TTS	0.009	0.036	0.031	0.02	0.067	0.011
16	SVM	Undersampling	SD + TTS	0.008	0.09	0.109	0.073	0.048	0.012
17	LR	Undersampling	SD + TTS	0.01	0.038	0.031	0.02	0.077	0.012
18	NB	Undersampling	SD + STS + TTS	0.002	0.007	0.006	0.006	0.012	0.005
19	SVM	Undersampling	SD + STS + TTS	0.003	0.18	0.208	0.149	0.041	0.008
20	LR	Undersampling	SD + STS + TTS	0.003	0.013	0.01	0.009	0.023	0.007