EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Industrial Engineering & Innovation Sciences
Information Systems Research Group

Septic Shock Mortality Prediction Using Deep Learning and Probabilistic Fuzzy Systems

Master's Thesis

K.R.C. (Kevin) Reijnders

Supervisors:
prof. dr. ir. Uzay Kaymak (TU/e)
dr. Florian Böing-Messing (TiU)

Eindhoven, June 2020

Abstract

Sepsis is one of the leading causes of death in intensive care units all over the world. The condition involves organ dysfunction in response to an infection, and can escalate into septic shock, in which the blood circulation of the patient is so severely disturbed that it starts to cause organ failure and poses a high mortality risk. In this work, we demonstrate how the mortality risk of septic shock patients can be predicted in a novel, near real-time fashion using information from vital signs and laboratory tests. We do so by first delineating and testing several strategies to leverage both types of information into a single prediction model. The first set of tests demonstrates how to overcome the extreme sparsity of the laboratory testing variables, while the second demonstrates that caution is required when splitting such data into training and testing sets - we find that standard splitting strategies result in severe leakage between the training and testing sets. Afterwards, we compare probabilistic fuzzy systems, densely connected neural networks and recurrent neural networks for septic shock mortality prediction on the resulting dataset. We first compare the performance of the models quantitatively using various model assessment metrics. Afterwards, we compare the models in a more qualitative fashion using techniques from the field of interpretable machine learning. We find that the densely connected neural network achieves the highest performance on the assessment metrics, mainly because of its ability to leverage a large number of features effectively at moderate computational cost. Perhaps surprisingly, our model inquiry reveals that the learned global behavior of this neural network is quite in line with traditional criteria used to assess sepsis severity. Our inquiry also reveals that recurrent neural networks are the most robust against feature permutation and unreliable feature values among the tested models.

Preface

This thesis was written as the final requirement for graduation of the Master of Science degree in Data Science & Entrepreneurship at the Jheronimus Academy of Data Science. I would like to thank a number of people for their support throughout this project and during the rest of the degree.

Firstly, I would like to thank my supervisor, Uzay Kaymak, for his guidance during this project from the very start until the very end. Though we were short on time during many of our meetings, every one of them was just as effective in finding the right answers to many tough problems. I would also like to thank my second supervisor, Florian Böing-Messing, for his feedback on the proposal, the intermediate presentation and the final work. It has certainly been valuable to receive feedback from a person specialized in a somewhat different field - it brings in new perspectives that would have not been seen otherwise.

Several other people also contributed to small, yet essential parts of this thesis. In that vein, I would like to thank Caro for helping me with her expertise on fuzzy clustering, Jasper for his explanations regarding Tensorflow, and finally Juul for assisting me in the interpretation of the many mysterious medical variables in this work.

Completing the Master's degree at JADS is as much of a team effort as it is an individual one, as nearly every course involves team projects that form a substantial part of the final grade. Therefore, I would like to thank my friends and team members at JADS for their support and for the vivid conversations in the breaks. To name but a handful of the people with whom I had the honour to collaborate many times, I would like to thank Martin, Bart, Mehul, Erik, Dora and Dimos. The ride would have been much less fun without you. It is painful that the Coronavirus has brought our pleasant conversations at JADS to such an abrupt end.

Finally, the better part of this work was written at home, with the exceptional circumstances due to the Coronavirus forcing all of us to work from home almost exclusively. This has made my gratitude for the support of my family even greater than it already was. Special thanks to my mom, whom we would jokingly yet justly refer to as "the manager" in here during these homeridden times.

Kevin Reijnders, June 2020

Contents

Chapter 1

Introduction

1.1 Problem Definition

For many decades, sepsis has been one of the leading causes of death in intensive care units (ICUs). Sepsis is defined as "a life-threatening organ dysfunction caused by a dysregulated host response to infection" (Singer et al., 2016, p.804), and can eventually progress into septic shock, which has a very high mortality rate. As with many diseases, early treatment is paramount to treat septic patients effectively. For this reason, various contemporary research efforts focus on the ahead-of-time prediction of the onset of sepsis (Desautels et al., 2016; Reyna et al., 2019). However, an assessment of the severity of the disease (i.e. mortality rate) is valuable in addition to disease onset prediction because it aids in (1) making better treatment decisions, (2) performing clinical trials and (3) monitoring the performance of the treatment that a patient is receiving (Bouch & Thompson, 2008). To give a more concrete example, near real-time individual mortality risk predictions can be used to check whether antibiotic treatment is having the desired effect, facilitating timely intervention when treatment is ineffective. The conventional approach to assessing mortality risk is through the usage of severity scoring systems (Bouch & Thompson, 2008). Although such systems are simple and well-tested, they are limited in various aspects: they are calibrated for general patient populations (rather than for sepsis patients specifically), only consist of a one-time prediction at ICU admission (no predictions after or during treatment) and are unable to capture complex, non-linear relationships that possibly involve temporal dynamics. For these reasons, sepsis treatment could probably be improved further if there were a system that predicts mortality for individual patient in a near real-time fashion, leveraging information from vital signs and lab tests.

As one would expect, various previous works have already investigated mortality prediction of sepsis patients based on individual patient data. On the one hand, previous works used logistic regression (Le Gall et al., 1995) or fuzzy models (Fialho et al., 2010), which are simple and interpretable but unable to model complex, non-linear patterns with multiple interacting variables. On the other hand, previous works include the usage of neural networks (Hanisch et al., 2011; Jaimes et al., 2005), which reside on the other end of the modeling spectrum as they are typically able to achieve high performance at the cost of low model interpretability. To incorporate elements of both type of systems, Fialho et al. (2016) use probabilistic fuzzy systems (PFS) which are both interpretable and able to model non-linear relationships (i.e. universal function approximators). What is missing in literature is a direct empirical comparison between PFS and neural networks, particularly for septic shock mortality prediction. In response to the massive increase in popularity of deep learning (LeCun et al., 2015), also in the healthcare domain (Esteva et al., 2019), various literature reviews request additional research into the usage of deep learning techniques on electronic health records (EHR) data (Norgeot et al., 2019; Shickel et al., 2017; Xiao et al.,

2018). Several papers in top journals of medical fields also recently acknowledged the potential of machine learning in healthcare (Burki, 2016; Esteva et al., 2019; Rajkomar et al., 2019; Yu et al., 2018).

In summary, our research is centered on the following problem:

> *It is unknown how deep learning performs compared to probabilistic fuzzy systems on septic shock mortality prediction.*

1.2 Research Questions

Next, we formulate a sequence of subquestions that, upon answered, inform us of how deep learning and probabilistic fuzzy systems compare on septic shock mortality prediction.

Constructing a predictive model for septic shock mortality first requires a thorough understanding of septic shock and the traditional, clinical methods of assessing mortality risk. This leads to our first subquestion:

1. *How is sepsis and septic shock defined, and how is a patient's mortality rate predicted traditionally?*

Once an adequate understanding of septic shock and traditional mortality prediction methods has been acquired, we must develop an understanding of deep learning and probabilistic fuzzy systems as to be able to implement these techniques. This leads to our second subquestion:

2. *What are the foundations of probabilistic fuzzy systems and deep learning?*

The severity of a septic shock condition is dependent on a range of physiological variables. For this work, physiological variables are of interested. Particularly, measurements originating from (1) vital signs and (2) laboratory testing. As these variables are measured irregularly and at extremely varying time scales (e.g. once per second to once per day), incorporating both types into one model is a challenge. Another challenging aspect is creating a fair model assessment as multiple facets of the time series can induce leakage of the training data into the testing data (e.g. leakage due to time-order violation). Thus, our third sub-question is as follows:

3. *How can events recorded at uneven and misaligned timescales be transformed as to allow predictive modeling, and how should the resulting dataset be split to avoid leakage between the training and testing set?*

Upon answering the questions above, we have become capable to implement deep learning and probabilistic fuzzy systems as to compare their effectiveness. Therefore, the final sub-question is:

4. *How do deep learning and probabilistic fuzzy systems compare on predicting septic shock mortality?*

1.3 Scope

The focus of this research is on comparing deep learning and probabilistic fuzzy systems for septic shock mortality prediction in a manner that is similar to the work by the Fialho et al. (2016). Mortality is predicted as a probability and the variables taken into account are restricted to those

in Fialho et al. (2016), i.e. to a set of variables spanning vital signs and laboratory testing. Furthermore, it was carefully decided to execute the comparison between probabilistic fuzzy systems and deep learning exclusively on the freely accessible MIMIC-III database (Johnson et al., 2016) and with open source tools, thereby aiding future reproduction, reuse and extension. Finally, the emphasis of this work is on data processing, modeling and analysis, i.e. on a demonstration of how contemporary techniques can be applied in a clinical setting. A comprehensive evaluation of the models in terms of clinical/medical value and correctness is beyond the scope of this work as no access to medical experts was available.

1.4 Research Methodology

In order to answer research question 1 and 2, we perform a literature study. For answering research question 3, we start by critically examining how related previous works address the problem of converting events recorded at uneven and misaligned timescales as to allow predictive modeling. Afterwards, we formulate three strategies for this transformation, and test the effectiveness of each on our dataset to find which is most effective. The effectiveness is tested both by assessing measures for feature informativeness of the target label and by evaluating the performance predictive models that require little tuning (i.e. gradient boosting machines). Continuing with the strategy that is found to be the most effective, we again critically examine previous related works to formulate two ways of splitting the data into training and testing sets. Then, we test whether leakage between the training and testing sets occurs by evaluating the performance of gradient boosting machines with a large number of trees that are able to exploit leakage to obtain unrealistic levels of performance. As clear evidence of leakage was found in one of the splitting strategies, the other strategy was used for the remainder of the research. Lastly, to answer research question 4, we fit probabilistic fuzzy systems and two types of deep learning models (densely connected neural networks and recurrent neural networks) to our data, and subsequently compare their performance quantitatively using two carefully chosen metrics. Finally, we inquire the behavior of the models using various techniques from the field of explainable machine learning, enabling us to assess whether model behavior is in line with general intuitions regarding increasing or decreasing mortality risk and whether some of the models show suspicious, noisy behavior.

1.5 Contributions

The main contribution of this work is an in-depth comparison of probabilistic fuzzy systems and deep learning for septic shock mortality prediction, involving both an assessment on performance metrics and a model inquiry to inspect the inner mechanisms of the models. To the best of our knowledge, such comparison has not been made yet, particularly in the specific context of predicting septic shock mortality. A second contribution is that we demonstrate how to leverage events sampled at irregularly and strongly varying timescales into a single prediction model that generates predictions in near real-time, i.e. every time new information becomes available. Again, to the best of our knowledge, this is a different approach than in most related works: such works usually transform the time series using a template (e.g. hourly predictions), and thereby lose information and predict less frequently. We also contribute by implementing the above in a transparent, reproducible manner on an open clinical database, thereby putting future works at a head start that also aim to perform similar data conversions, aim to use the same open clinical database, or aim to use probabilistic fuzzy systems.

1.6 Outline

The remainder of this research is organized as follows. Chapter 2 includes the preliminaries to our work, which contain general information about sepsis (definitions, causes, incidence, costs, etc.) and severity scoring systems (the conventional way of predicting mortality). Afterwards,

we continue with Chapter 3 where we discuss the theoretical background of our work, with a main focus on the two types of prediction models in this research: probabilistic fuzzy systems and deep learning. Additionally, various model assessment metrics and interpretable machine learning techniques are discussed. In Chapter 4, we describe the data processing methods used to transform the irregular events into a modeling-friendly format and to compare probabilistic fuzzy systems and deep learning. As our experiments are conducted on a large dataset that raises the need for a considerable amount of preprocessing, we dedicate Chapter 5 to the details of our experimental setup. This Chapter also includes additional information on the model tuning procedures. Subsequently, the results of our experiments are discussed in Chapter 6, followed by the conclusions of this work in Chapter 7.

Chapter 2

Preliminaries: Sepsis and Mortality Prediction

Building and testing models for predicting septic shock mortality requires an understanding of the sepsis and septic shock conditions as well as previous approaches to predicting mortality risk. General information about the disease includes aspects such as the formal definitions of sepsis and septic shock, main causes for these conditions, general treatment procedures, disease incidence and associated costs. Regarding the prediction of mortality, we investigate standard approaches to this task that have found wide adoption in clinical practice, namely severity scoring systems.

2.1 Sepsis and Septic Shock

2.1.1 Definitions

The latest definitions of sepsis and septic shock were created in 2016 by a task force of the European Society of Intensive Care Medicine and the Society of Critical Care Medicine (Singer et al., 2016). Put simply, sepsis is a synonym for blood poisoning, but is officially defined as "a life-threatening organ dysfunction caused by a dysregulated host response to infection" (Singer et al., 2016, p. 804). Septic shock is defined as "a subset of sepsis in which underlying circulatory and cellular metabolism abnormalities are profound enough to substantially increase mortality" (Singer et al., 2016, p. 806). Formerly, severe sepsis was also an acknowledged condition that was in between these two extremes of sepsis diagnoses. Because the severe sepsis diagnosis was dropped in 2016, various research that we will discuss in this chapter still utilizes this definition. With the new definitions, severe sepsis falls partially under sepsis and partially under septic shock. Concretely, a patient with a suspected infection that develops an increase of 2 on the Sequential Organ Failure Assessment Score (SOFA) after the onset of the infection is defined to have sepsis. Table 2.1 shows which quantitative criteria constitute the SOFA score and Figure 2.1 shows a schematic representation of the formal diagnosis process of sepsis and septic shock. Note that sepsis assessment with the SOFA score is usually preceded by a quick SOFA (qSOFA) assessment, which is a simplified test to find signs of possible sepsis without the use of any resource-consuming laboratory tests. Furthermore, septic shock is a subset of sepsis which involves more severe symptoms and a substantially higher mortality rate than the 'weak' sepsis condition. A patient is defined to have septic shock when vasopressors are required to maintain a mean arterial pressure (MAP) over 65mm Hg and when his or her serum lactate level is above 2 mmol/L (see Figure 2.1).

Table 2.1: Overview of the Sequential Organ Failure Assessment Score criteria (Singer et al., 2016)

	Score				
System	0	1	2	3	4
Respiration					
Pao_2/Fio_2, mm Hg (kPa)	≥400 (53.3)	<400 (53.3)	<300 (40)	<200 (26.7) with respiratory support	<100 (13.3) with respiratory support
Coagulation					
Platelets, $\times 10^3/\mu L$	≥150	<150	<100	<50	<20
Liver					
Bilirubin, mg/dL (μmol/L)	<1.2 (20)	1.2-1.9 (20-32)	2.0-5.9 (33-101)	6.0-11.9 (102-204)	>12.0 (204)
Cardiovascular	MAP ≥70 mm Hg	MAP <70 mm Hg	Dopamine <5 or dobutamine (any dose)	Dopamine 5.1-15 or epinephrine ≤0.1 or norepinephrine ≤0.1	Dopamine >15 or epinephrine >0.1 or norepinephrine >0.1
Central nervous system					
Glasgow Coma Scale score	15	13-14	10-12	6-9	<6
Renal					
Creatinine, mg/dL (μmol/L)	<1.2 (110)	1.2-1.9 (110-170)	2.0-3.4 (171-299)	3.5-4.9 (300-440)	>5.0 (440)
Urine output, mL/d				<500	<200

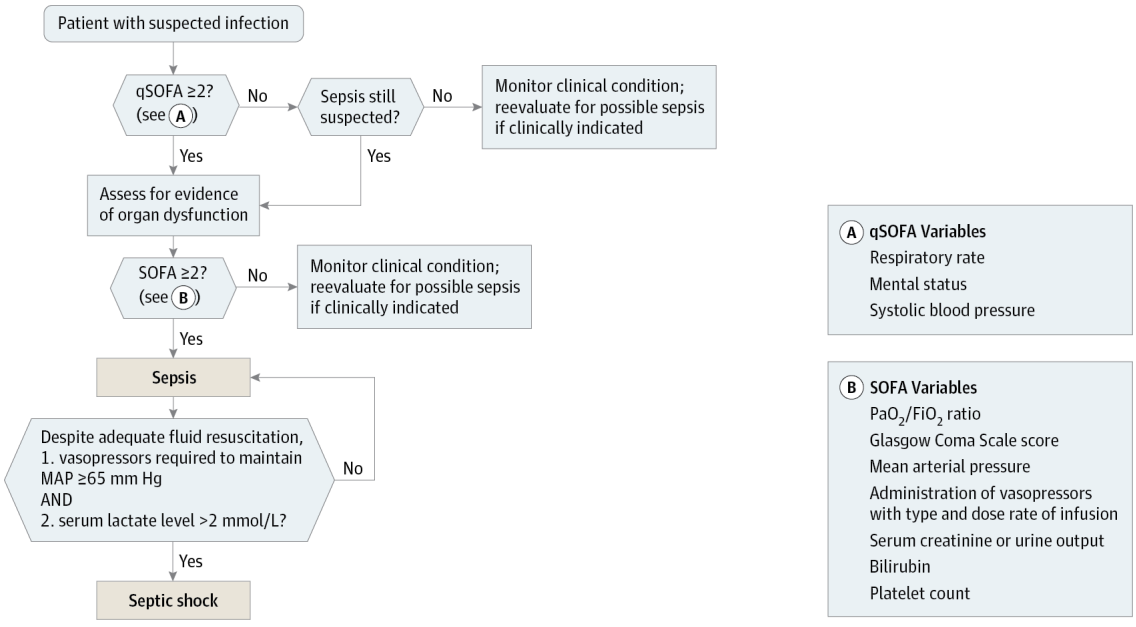

Figure 2.1: Schematic representation of sepsis and septic shock operationalization (Singer et al., 2016)

2.1.2 Causes, Prevention and Treatment

As Figure 2.1 indicates, the manifestation of sepsis always starts with an infection. Although the majority of infections that eventually lead to sepsis are caused by bacteria, sepsis can also be caused by a fungal infection, viruses or parasites (Martin, 2012). Sepsis is thus one of the main complications that can follow an infection with SARS-CoV-2 (Gorbalenya et al., 2020), the virus that causes the coronavirus disease that emerged in 2019. Furthermore, respiratory infections are the most common types of infections that eventually cause sepsis because infections in the lungs are especially likely to disseminate toxins in the blood, leading to a sepsis diagnosis. For similar reasons, the second and third most common types of sepsis-causing infections are of a genitourinary and abdominal nature. Patients with a weakened immune system are at extra risk of developing sepsis from an infection. The most effective way to prevent sepsis is through preventing the initial infection: careful hygiene control by for instance washing hands and cleaning environments has been found to be highly effective for sepsis prevention (World Health Organization, 2018). Other preventive measures include vaccines and preventive antibiotic treatment. If the patient has already been infected, then sepsis is usually treated through fighting the infection with antibiotics, fluid resuscitation, nutritional support and finally organ-supporting treatments in case of organ failure (Evans, 2018; Polat et al., 2017). We refer to Schmidt & Mandel (2019) for a comprehensive, up-to-date review on treatment guidelines in response to sepsis and septic shock. Moreover, early detection of sepsis is crucial for effective treatment: multiple studies show evidence that patients with a delayed treatment experience higher mortality rates (Liu et al., 2017; Seymour et al., 2017), with one study even finding a 3.6% to 9.9% increase in mortality per hour of treatment delay (Kumar et al., 2006). It is thus of utmost importance to administer antibiotics bundles as soon as possible, preferably immediately after the very first suspicion of sepsis. As the type of infection of a patient can be hard or time consuming to identify, it is often difficult to administer the right type of antibiotics - highlighting the value of mortality prediction which can assist medical personnel in detecting whether treatment is working.

2.1.3 Incidence and Mortality

Sepsis is a condition with a high incidence and substantial mortality rates. In the US, 1 in 3 patients who die in a hospital in the US have sepsis (Centers for Disease Control and Prevention, 2016), resulting in sepsis being the disease with the largest inpatient hospital expense in the US (Torio & Moore, 2013). Worldwide, sepsis incidence is estimated to be 437 per 100,000 person-years from 2003 to 2015, while the severe sepsis incidence was estimated to be 270 per 100,000 person-years (Fleischmann et al., 2016). In that meta-analysis, it was estimated that 31.5 and 19.4 million cases of sepsis and severe sepsis are treated in hospitals each year, respectively. The global mortality rates of sepsis and severe sepsis were found to be 17% and 28%. It must be noted that these numbers are a rather rough extrapolation as data on sepsis in low- and mid-income countries was very sparse in this study. A recent study by Rudd et al. (2020) that did include data from low- to mid-income countries even found that 48.9 million cases of sepsis occurred in 2017, with 11 million sepsis-related deaths that represent 19.7% of all global deaths. Note that the studies by Fleischmann et al. (2016) and Rudd et al. (2020) still (partially) rely on the old definitions of sepsis. With the new definitions, sepsis should have a mortality rate in excess of 10%, while septic shock should have a mortality rate in excess of 40%, but it is relevant to note here that these estimates rely on a substantially smaller amount of empirical analysis than in Fleischmann et al. (2016) and Rudd et al. (2020). In the Netherlands, sepsis is the fourth most frequent ICU admission diagnosis in the country: approximately 5,000 patients were admitted to the ICU in 2018, of which 27% died due to the consequences of sepsis (Stichting NICE, 2018). At first sight, this seems to indicate that sepsis incidence is rather low in the Netherlands compared to other countries. This can be motivated by the relatively proficient healthcare industry in the Netherlands. However, the study was restricted to sepsis cases in which the patient was admitted to the ICU and in which the manifestation was relatively severe (high APACHE IV score at diagnosis). It thus seems likely that the actual incidence of sepsis in the Netherlands is higher

Table 2.2: Overview of sepsis, severe sepsis and septic costs per person in the US. Also includes length of stay statistics. Adapted from Paoli et al. (2018).

Type	Severity	Time (days) / costs
Mean cost of sepsis hospitalization (median ± std)	Sepsis	\$16.324 (\$9,266 ± \$33,925)
	Severe sepsis	\$24,638 (\$13,832 ± \$37,710)
	Septic shock	\$38,298 (\$22,510 ± \$55,052)
Mean hospital length of stay (median ± std)	Sepsis	7.7 (5 ± 14.2)
	Severe sepsis	10 (7 ± 12.4)
	Septic shock	12.6 (9 ± 15.5)
Mean ICU length of stay (median ± std)	Sepsis	5.1 (3 ± 7)
	Severe sepsis	6.2 (3 ± 8.1)
	Septic shock	7.2 (4 ± 9.2)

than in the study, albeit still lower than worldwide estimates. Obviously, the incidence of sepsis is likely to be higher at the time of writing due to the Coronavirus pandemic.

2.1.4 Associated Costs

As the high incidence and mortality rates of sepsis-related conditions already foreshadow, the disease is accompanied by substantial healthcare costs. Table 2.2 shows an overview of the costs for sepsis, severe sepsis and septic shock patients in the US between 2010 and 2016. Sepsis-related conditions go accompanied by hospital stays of approximately one to two weeks, depending on the severity of the condition (shown in more detail in Table 2.2). Paoli et al. (2018) also found that both the length of stay and the associated costs of a sepsis-related condition are significantly higher when the diagnosis for sepsis was not present at the moment of admission. Without diagnosis at admission, both the costs and length of stay approximately double the numbers in Table 2.2. This again demonstrates the importance of detecting sepsis and treating infections early: the failure to do so is associated with significant costs and increased mortality rates. A recent study by Koster-Brouwer et al. (2016) estimated the costs of severe sepsis and septic shock in two leading hospitals in the Netherlands, and found the mean costs per hospital admission to be €24,520 and €32,875, respectively (95% confidence intervals: €21,288 to €27,973 and €29,615 to €36,281). The mean amounts roughly correspond to those in the US (shown in Table 2.2), but the variance in the treatment costs is much higher in the US. The fact that healthcare in the Netherlands is more heavily standardized and regulated by the government than in the US is probably a main cause for these differences.

2.2 Mortality Prediction using Severity Scoring Systems

Severity scoring systems are relatively simple tools to assess the severity of disease in patients in the ICU (Bouch & Thompson, 2008). The main idea behind these systems is that a clinician collects information about a patient (e.g. age, heart rate, etc.) and consequently applies the rule-based calculation prescribed by the specific scoring system to obtain an estimation of the severity of illness. Table 2.3 shows an example of the process of severity scoring a patient. As we can see, scoring systems provide a direct estimate for the mortality rate of the patient in question. The mortality rates corresponding to different scores are typically determined through an analysis of a diverse, large population of ICU patients from different hospitals. It is these underlying calculations that aid severity scoring systems in achieving strong calibration and generalization power. The simplicity and strong generalization power of such models is also their main weakness: they are (1) not customizable to fit specific diseases or patient groups, and (2) unable to capture temporal dynamics and complex, non-linear relationships. An example of the former is that severity scoring systems do not include various laboratory testing-related variables indicative of septic shock mortality (e.g. compare Appendix A.1 and B.2), and such variables cannot be added

Table 2.3: A fictitious example to demonstrate the process of severity scoring a patient. The system used for this example is APACHE II (Knaus et al., 1985). A specific total score corresponds to a fixed nonoperative and postoperative mortality rate. For a score of 19, this is 25% and 12%, respectively.

Criterion	**Measurement Value**	**Corresponding points**
History of severe organ failure or immunocompromise	No	2
Age	65 years	5
Temperature	38.5 °C	1
Mean arterial pressure	75 mm Hg	0
pH	7.39	0
Heart rate	70 beats/min	0
Respiratory rate	19 breaths/min	0
Sodium	140 mmol/L	0
Potassium	4 mmol/L	0
Creatinine	98 μmol/L	0
Acute renal failure	No	0
Hematocrit	36 %	0
White blood cell count	3.7 x 10^9 cells/L	2
Glasgow Coma Scale	7 points	8
FiO_2	<50%	0
PaO_2	>70 mmHg	1
Total		19

to severity scoring systems without re-doing the expensive, large scale data analysis to calibrate the scores and mortality probabilities. To give an example of the inability to capture temporal dynamics, a wildly varying heart rate over time may indicate that the patient is unstable and that mortality is imminent. However, by entering a single, average value for heart rate in a severity scoring system, this sign is invisible for those kinds of models and hence the high mortality risk will not be detected. Additionally, scoring systems are only valid when used for a single assessment immediately after the patient enters the ICU, i.e. only a single mortality prediction can be made for a single patient. This is in sharp contrast with the tested models for mortality prediction in this thesis: they involve predicting mortality multiple times for a single patient during his or her ICU stay. The capability to re-assess patient mortality over time makes our mortality prediction models useful for additional purposes such as monitoring treatment effectiveness.

As severity scoring systems have found widespread adoption in day-to-day clinical practice, many different systems have been proposed, used and tested over the years (Bouch & Thompson, 2008). One the most widely used scoring systems in clinical settings is the Acute Physiology and Chronic Health Evaluation (APACHE) score (Zimmerman et al., 2006). This system uses variables that should be measured within 24 hours of the ICU admission, and the worst values within these first 24 hours should be entered in the scoring system to estimate the final score. Although the APACHE scoring system has seen four major revisions over the years, the second version is most widely used as APACHE III and IV complicate the score calculation considerably while only slightly increasing predictive accuracy and reliability (Ayazoglu, 2011). A scoring system similar to APACHE is the Simplified Acute Physiology Score (SAPS), which was initially proposed as a simplification to the Acute Physiology Score (APS) (Le et al., 1984). The SAPS scoring system also uses the worst values of various variables measured within the first 24 hours after ICU admission. SAPS has been revised three times over the years (Moreno et al., 2005), and like APACHE the performance differences between different versions are relatively minor, although the latest version (SAPS 3) has a slightly better calibration. A third mortality scoring system that can be used upon ICU admission is the Mortality Prediction Model (MPM) (Lemeshow et al., 1985), which should be

easier to use than APACHE because it involves information from fewer variables. This system has also seen various minor revisions over time (Higgins et al., 2005; Lemeshow et al., 1993), but has not accomplished a popularity even remotely close to e.g. APACHE. Another recent scoring system is the Oxford Acute Severity of Illness Score (OASIS) (Johnson et al., 2013), which uses six physiological variables and a combination of machine learning and particle swarm optimization to combine these variables into a severity score. It was designed with the particular objective in mind to create a severity scoring system that uses fewer variables than APACHE, but that achieves a similar accuracy level. The idea is that such a severity scoring system would be easier to use in retrospective studies as it reduces the number of variables to be extracted from the data considerably. Appendix A shows a comparison of the variables used in the most relevant previously mentioned severity scoring systems, providing us with an overview of variables that generally relate to patient mortality. Finally, it must be noted that two other scoring models for determining the severity of sepsis specifically are the SOFA and qSOFA models. These two models already received extensive coverage in section 2.1.1.

Chapter 3

Background: Fuzzy Systems, Deep Learning, Model Assessment and Interpretability

In this chapter, we cover the theory on probablistic fuzzy systems and deep learning that is relevant with respect to our research purposes. Additionally, we discuss possible model assessment metrics and recently proposed techniques to inquire the behavior of predictive (black box) models. This chapter forms the theoretical basis to implement both types of prediction models in later chapters.

3.1 Fuzzy Systems

This research encompasses two types of supervised learning systems: probabilistic fuzzy systems and deep learning. In this chapter, we discuss the former, which is built upon the foundations of fuzzy sets as first proposed by Zadeh (1965). The choice for the usage of fuzzy systems in this thesis is not arbitrary: modeling based on fuzzy sets and fuzzy logic has a long academic history. The seminal paper by Zadeh (1965) has currently been cited over 90,000 times according to Google Scholar and over 33 journals and conferences currently exist that have fuzzy in their title. Many influential journals (e.g. Information Sciences and IEEE Transactions on Systems, Man, and Cybernetics) also publish articles about the topic (Mendel, 2017). The scope of this section is theory relevant for our main research purpose: predicting septic shock mortality using probabilistic fuzzy systems. To this end, we describe the basic intuitions behind fuzzy sets and rule-based fuzzy systems, as well as the probabilistic fuzzy systems as used in Fialho et al. (2016) to predict septic shock mortality.

3.1.1 Fuzzy Sets

The main aspect in which fuzzy sets differ from conventional or *crisp* sets is that they allow for statements to be partially true (Zadeh, 1965): elements can have a partial membership to a set. This is in contrast with traditional set theory where an element must either be a member of the set or not. As an example, consider the situation where we try to model whether a person is tall. A person with a length of 1.60 meter is most certainly not considered tall in the Netherlands, while a person of 2 meters is clearly tall. Obviously, various degrees of tallness remain in between: a person of 1.85 meter is somewhat tall, but substantially less tall than a person of 2 meters. In conventional set theory, we would have to model this concept by specifying a crisp boundary from

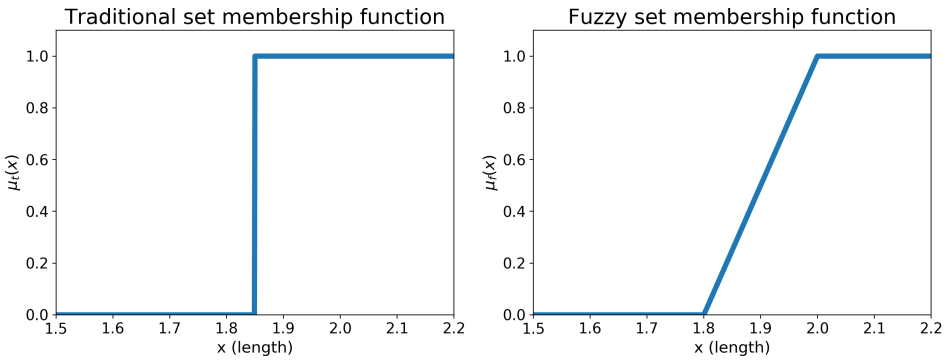

Figure 3.1: Graphical representation of the traditional or crisp set membership function of Equation 3.1 (*left*) and the fuzzy set membership function of Equation 3.2 (*right*).

where on a person must be considered tall. To this end, we could for instance define a conventional set membership function as follows:

$$\mu_c(x) = \begin{cases} 1 & \text{if } x \geq 1.85 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

Here, $\mu_c(x)$ denotes a traditional or *crisp* (c) membership function (μ) over a person's length (x). Now, a corresponding fuzzy membership function expressing whether a person is tall is:

$$\mu_f(x) = \begin{cases} 5 \cdot (x - 1.8) & \text{if } 1.8 \leq x < 2 \\ 1 & \text{if } x \geq 2 \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Figure 3.1 shows a graphical representation of the membership functions described in Equation 3.1 and Equation 3.2. Notice how the membership function based on fuzzy sets matches the human definition of being 'tall' much more neatly than the membership function based on crisp sets: a person is tall to a certain *degree* (number between 0 and 1), rather than strictly tall or not tall (either 0 or 1, respectively). In medical contexts, fuzziness is inherent in many situations, for instance in statements such as "blood pressure is high", "the patient is comatose" and "breathing rate is slow". In fact, techniques built on fuzzy sets have been widely used in computer-aided medical diagnosis systems (Yanase & Triantaphyllou, 2019).

3.1.2 Rule-based Fuzzy Systems

In order to leverage fuzzy sets for our classification problem (mortality prediction), we must first make the extension to *rule-based fuzzy systems* (often abbreviated as *fuzzy systems*). Figure 3.2 shows the general contents of a fuzzy system. As the name suggests, fuzzy rule-based systems are composed of rules. A rule has the format "IF p THEN q" (Mendel, 2017). The part of the rule that expresses the conditional, p, is referred to as the *antecedent* of the rule, while the statement after the conditional, q, is referred to as the *consequent*.

$$\begin{cases} \text{IF blood toxicity is high, THEN mortality risk is high} \\ \text{IF blood toxicity is low THEN mortality risk is low} \end{cases} \tag{3.3}$$

Consider the example of a set of fuzzy rules in equation 3.3. In this equation, the antecedent involves a statement about a linguistic linguistic variable, in this case blood toxicity. Now, the

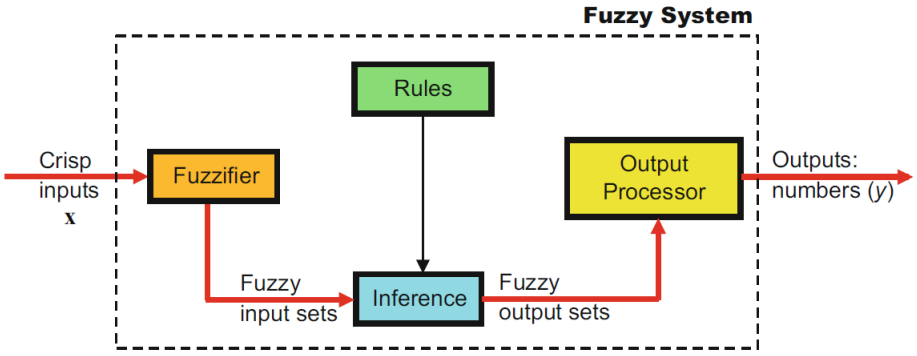

Figure 3.2: The main steps in a rule-based fuzzy system. Adapted from Mendel (2017).

concrete meaning of "high" and "low" of this linguistic variable is defined by fuzzy sets over a crisp input (e.g. numerically stored lab test results). The conversion of a crisp input into a linguistic variable is also commonly referred to as *fuzzification* (the orange box in Figure 3.2). The output of this fuzzification step is a fuzzy input, which is then used in the rules as in our example in Equation 3.3. The rules are deployed in a so-called *inference system*, in which the rules are used to convert the fuzzy input into a fuzzy output (a statement about the mortality risk in our example). The result of this step is a fuzzy output, which is finally converted into a crisp output via the output processor. For our mortality prediction problem, the output is a probability.

A relevant aspect to realize here is that we can distinct two approaches to build a fuzzy system. The first approach is to specify the fuzzifier, inference engine and output processor manually. In this case, domain knowledge is used to manually specify the behavior of the fuzzy system. The second approach is to build the system in a data-driven manner, meaning that the fuzzy system (or usually just *parts of* the fuzzy system) is inferred from a dataset. One way to build a fuzzy system in a data-driven manner is through supervised learning. The next section describes the fuzzy system used in this research, which is a combination of a manually specified and data-driven fuzzy system.

3.1.3 Probabilistic Fuzzy Systems

In probabilistic fuzzy systems (PFS) (Van Den Berg et al., 2002), the antecedents are fuzzy conditions which are similar to regular rule-based fuzzy systems. The consequents, however, are probability distributions (see Equation 3.4). If we now infer the contents of the rules from a dataset, we have effectively achieved a supervised learning system for classification problems. As there are no mandatory constraints on the shape of the antecedents and the number of antecedents, PFS are universal function approximators: they can fit complex, non-linear patterns in the data. Moreover, a PFS consist of a set of linguistically interpretable rules (e.g. "IF x is high, then the output probability y equals 0.7"), meaning that the model can be interpreted fairly well as long as the number of variables and rules are not too high, and the antecedent membership functions shapes are not too complex. PFS are thus powerful prediction models that strike a balance between on the one hand consistency, transparency and interpretability, and on the other hand the ability to model complex relationships and decision boundaries.

We will now specify PFS in more depth in order to be able to reason about their contents and to implement them in our experiments. We follow the specification of PFS as in Fialho et al. (2016). The general form of a PFS is as follows:

$$\begin{aligned} \text{Rule } R_j\text{: IF } \mathbf{x} \text{ is } A_j \text{ THEN } & y = \omega_1 \text{ with probability } Pr(\omega_1|A_j) \\ & y = \omega_2 \text{ with probability } Pr(\omega_2|A_j) \\ & ... \\ & y = \omega_c \text{ with probability } Pr(\omega_c|A_j). \end{aligned} \tag{3.4}$$

Like in a regular rule-based fuzzy system, a PFS can have multiple (i.e. j) fuzzy rules. Each rule has an antecedent A_j, which consists of a fuzzy membership function $\mu_{A_j}(\mathbf{x})$. The consequents of the rules transform the fuzzified input into a probability $Pr(\omega_c|A_j)$ for each class ω_c. As we are performing binary prediction (mortality or survival), $c \in \{0, 1\}$. Additionally, we wish to predict a class probability based on an input vector $\mathbf{x}$, thus we must obtain estimates for $Pr(\omega_c|\mathbf{x})$, which is the probability that an arbitrary data point $\mathbf{x}$ belongs to class ω_c. We obtain these estimates with the following equation:

$$\hat{Pr}(\omega_c|\mathbf{x}) = \sum_{j=1}^{J} \beta_{A_j}(\mathbf{x}) \cdot Pr(\omega_c|A_j). \tag{3.5}$$

In Equation 3.5, $\beta_{A_j}(\mathbf{x})$ is the normalized rule activation, which is defined as $\beta_{A_j}(\mathbf{x}) = \mu_{A_j}(\mathbf{x}) / \sum_{j=1}^{J} \mu_{A_j}(\mathbf{x})$. In less mathematical terms, this operation converts the raw rule membership values of instances into relative membership values that allow one to compare how much more an instance triggers the activation of one rule compared to another. It leads to the desirable property that $\sum_{j=1}^{J} \beta_{A_j}(\mathbf{x}) = 1$, i.e. that the normalized rule activations for all rules together are equal to 1 for every individual instance. In order to implement Equation 3.5, we must determine two sets of parameters: those regarding the antecedent membership functions $\mu_{A_j}(\mathbf{x})$ and those regarding the rule consequents $Pr(\omega_c|A_j)$. Fialho et al. (2016) do this using the sequential method. In this method, the antecedent membership functions are assumed to be the product of N univariate Gaussian membership functions (one for each dimension n). Following the definition of a Gaussian distribution, we get the following general specification of the antecedent rule membership functions:

$$\mu_{A_j}(\mathbf{x}) = \exp(-\sum_{n=1}^{N} \frac{(x_n - v_{jn})^2}{\sigma_{jn}^2}) = \prod_{n=1}^{N} \exp(-\frac{(x_n - v_{jn})^2}{\sigma_{jn}^2}) = \prod_{n=1}^{N} \mu_{A_j n}(x_n), \tag{3.6}$$

meaning that the membership of an instance $\mathbf{x}$ to rule A_j can be computed directly over all dimensions, but also by multiplying its membership to the rule over each individual dimension n. In turn, we require estimates for the centers $\mathbf{v}_j = \{v_{j1}, ..., v_{jn}\}$ and the widths $\Sigma_j = \{\sigma_{j1}, ..., \sigma_{jn}\}$ of the j antecedent membership functions in each dimension n of the input space. These centers are usually determined through a clustering method, and the widths are computed using the definition:

$$\sigma_{jn} = \min_{j' \neq j} ||\mathbf{v}_j - \mathbf{v}'_j||, \text{ for } (n = 1, ..., N), \tag{3.7}$$

wherein $||\mathbf{v}_j - \mathbf{v}'_j||$ is the Euclidian distance between two centers $\mathbf{v}_j$ and $\mathbf{v}'_j$. This implies that within a single rule, the widths are equal in every dimension with this method. This is usually sub-optimal, but as we will see later these widths can be tuned for every individual dimension later. Finally, we must estimate the intial values for the rule consequents, which can conveniently be performed using conditional probability estimation, resulting in:

$$Pr(\omega_c|A_j) = \frac{\sum_{i=1}^{I} \beta_{A_j}(\mathbf{x}_i) \cdot \chi_{\omega_c}(y_i)}{\sum_{i=1}^{I} \beta_{A_j}(\mathbf{x}_i)}, \tag{3.8}$$

with $\chi_{\omega_c} = 1$ if $y = \omega_c$ and 0 otherwise.

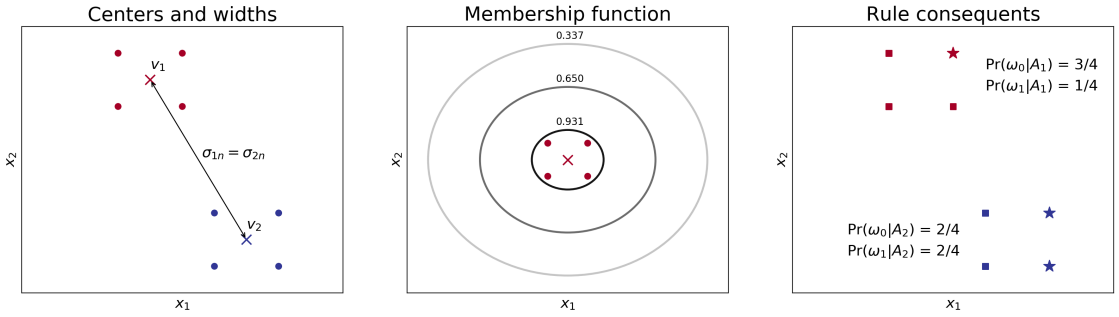

Figure 3.3: Simplified, two-dimensional illustration of the main concepts of a probabilistic fuzzy system. *Left*: observations (points), cluster centers (crosses) and the corresponding membership function widths. *Middle*: 3 examples of antecedent membership function values corresponding to cluster 1. The membership value is the same on all locations of the circle. *Right*: the rule consequent values corresponding to the two clusters (shapes indicate classes here).

Figure 3.3 shows an example where the main concepts of a PFS (centers, widths, membership functions and rule consequents) are visualized. Note, however, that the example PFS in Figure 3.3 is somewhat simplistic. Firstly, it is restricted to two dimensions and two clusters and rules. In reality, visualization may be more complicated due to a higher number of dimensions and rules. One could, in principle, also specify membership functions with different shapes than the Gaussian as in Figure 3.3. A PFS is thus very flexible with respect to its decision boundaries. However, a PFS is by no means unconditionally interpretable - as the number of rules and number of features grow, they become less and less interpretable. This is particularly true if the antecedents are allowed to have complex shapes. Secondly, the calculation of the rule consequents in the example in Figure 3.3 is simplified, as also the membership function is taken into account in reality when estimating the rule consequents (see Equation 3.8).

A notable hurdle that emerges when applying PFS in practice is that finding the correct number of clusters and the correct center positions for the antecedent rules can be challenging. In our simplified example in Figure 3.3, the number of clusters are clear, and the clusters are well separated. In reality, cluster separation is generally not so clear, making it difficult to identify the correct number of clusters. Clustering also inevitably suffers from the curse of dimensionality: the difficulty of separating data into clusters increases rapidly as the number of dimensions increases. The main reason for this issue is that distance functions used when clustering (usually the Euclidean distance) render observations increasingly similar as the number of dimensions increases. Hence, it is generally advised to reduce the number of dimensions (or, equivalently, features) before clustering.

Another limitation of the previously described PFS is that the clustering procedure at the start of the process is unsupervised: only the estimation of the rule antecedents, $Pr(\omega_c|A_j)$, involves usage of the target label. In order to tune all parameters of the PFS (the widths σ_{jn}, the centers v_{jn} and the rule antecedents $Pr(\omega_c|A_j)$) in a supervised manner, Fialho et al. (2016) use stochastic gradient descent to minimize the binary cross-entropy loss between the target labels and the predictions of the PFS. A further elaboration on gradient descent will follow in Section 3.3.2. Binary cross-entropy also goes by the name of negative log-likelihood, and is defined as follows:

$$L = -\sum_{i=1}^{I} y_i \ln(\hat{y}_i) = -\sum_{i=1}^{I} y_i \ln(\hat{Pr}(y_i|\mathbf{x}_i)). \tag{3.9}$$

Note that without any changes, the problem of optimizing the binary cross-entropy with respect to

the parameters of the PFS would be a constrained optimization problem because $0 \leq Pr(\omega_c|A_j) \leq 1$, which would be unsuitable for regular gradient descent algorithms. To transform the problem into an unconstrained optimization problem, auxiliary variables of $Pr(\omega_c|A_j)$ are optimized. The relationship between $Pr(\omega_c|A_j)$ and the auxiliary variables u_{j1} is described by the logit function:

$$u_{j1} = \log\left(\frac{Pr(\omega_1|A_j)}{1 - Pr(\omega_1|A_j)}\right). \tag{3.10}$$

Once the auxiliary variables are optimized, $Pr(\omega_1|A_j)$ can be obtained through the reverse function of Equation 3.10, i.e. the sigmoid function. Because we are dealing with binary classification, the probabilities for the other class can easily be obtained: $Pr(\omega_0|A_j) = 1 - Pr(\omega_1|A_j)$. Hereinafter, we will refer to the variant of PFS without gradient descent optimization as "PFS-CP" and to the variant with optimization as "PFS-ML", following Fialho et al. (2016).

3.2 Supervised Learning

Both probabilistic fuzzy systems and deep learning are within the wider realm of *supervised learning* (also called *machine learning*), which has a long academic history. A central concept in this field is that an output variable can be predicted given a set of input variables (James et al., 2013). This idea can be applied to a wide variety problems from many different domains. Examples are predicting credit default of bank customers given their financial transaction history, predicting whether a customer will churn based on his or her demographic information, and in our case: predicting whether a patient will die from septic shock given medical data about the patient. Mathematically, supervised learning can be specified using a simple equation:

$$\hat{y}_i = f(\mathbf{x}_i). \tag{3.11}$$

More concretely, we predict an output (also called *target label* or *class*) $\hat{y}$ for a single *instance* (data point or observation) i by computing a function f over one or more input *features* from instance i, stored in a vector $\mathbf{x}_i$. Note that the output can be a category (in which case we are performing *classification*) or a real number (in which case we are performing *regression*). To give an example of classification, one could try to predict whether a particular student i will pass ($y = 1$) or will not pass ($y = 0$) a course given some numerical information about the student. This numerical information should be stored as a list of numbers (i.e. as a vector $\mathbf{x}_i$). In this hypothetical example, we are interested in using the number of hours spent studying for the course and the average GPA of the student in other courses. If the student in question would have studied for 80 hours and would have an average GPA of 7.2, then $\mathbf{x}_i = (80, 7.2)$. Consequently, the probability of the student passing the course can be predicted by aggregating this information through computing a function f over these values. This naturally leads to the question what form that the function f should take, and how this form should be determined. What distinguishes the field of supervised learning is that f is learned in an automated fashion from a set of examples: an algorithm is fed many examples of previous generations of students of which the features (study time and average GPA) and target labels (course grade) are already known. Using this information, the supervised learning algorithm determines the contents of f by running the examples through a heavily automated, large set of computations. This process is called *training*, and is typically performed by comparing the actual target labels y with the predicted target labels $\hat{y}$ according to a loss function $L(y, \hat{y})$. The algorithm minimizes this loss by optimizing the parameters of f. Once the training process is finished, f has been estimated and the supervised learning algorithm is able to make predictions for new students upon given their study time for the course and their average GPA. In order to evaluate the performance of f, it is highly relevant to test f on a set of instances that were not present in the training data - only in this way we can assess whether f generalizes to unseen test data. For a more elaborate explanation of the general mechanisms that

underly supervised learning, we refer the reader to the excellent introductory works by Provost & Fawcett (2013) and James et al. (2013).

3.3 Deep Learning and Neural Networks

Deep learning is a specific type of supervised learning which has the distinctive capability to infer useful representations for the prediction task at hand through statistical learning (Goodfellow et al., 2016). Figure 3.4 illustrates the main idea behind this capability of deep learning. When using deep learning on images to recognize whether a cat is in the image, the algorithm would be shown a collection of pictures in which some contain cats and some do not. Thus, the features would be the raw pixels of the image and the label of each image would be whether it contains a cat or not (binary classification). In more traditional forms of supervised learning (such as logistic regression), this would result in an inaccurate classifier because a single pixel in the picture (i.e. one feature) has a negligible correlation with the target label - it is not a meaningful feature for the prediction task at hand. Specifying meaningful features from the raw features by hand, however, is an impossible task because of the many different poses and shapes that cats and dogs have on images. Deep learning can find these meaningful features automatically through a supervised (or even unsupervised) learning process, thereby eliminating this issue of manual feature engineering to a great extent. These features can be of a hierarchical nature: in the example, the first layer of abstraction contains low-level concepts such as lines and edges, while higher layers contain higher-level concepts such as eyes, ears and noses. Subsequently, the network is able to use this information to make a prediction, in this case "cat" or "no cat". Note that this is a somewhat hypothetical example as it is generally quite hard to discover what concepts that the units and layers in the network correspond to precisely. This is also why neural networks are frequently referred to as 'black-box models' (more information on this issue wil follow in Section 3.5). Regardless, deep learning has proven to be an effective way to avoid this need for excessive manual feature engineering, thereby making various prediction tasks (such as image recognition) considerably easier: higher scores on performance metrics can be achieved with a significantly lower amount of hand-coding (LeCun et al., 2015).

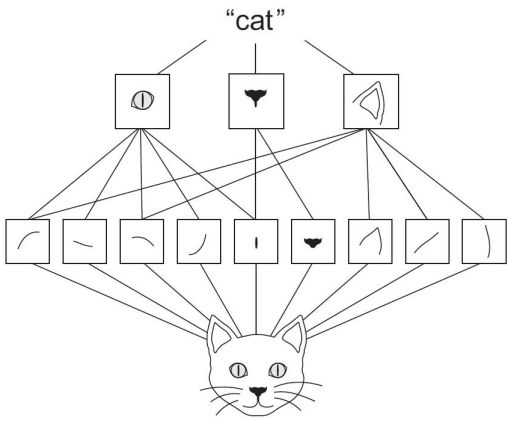

Figure 3.4: Demonstration of the learned hierarchical representation of a concept that deep learning is able to extract from statistical learning on a training dataset (Chollet, 2017).

3.3.1 Internal Representations

Though the automatic feature engineering of deep learning may seem somewhat mysterious at first sight, the whole process simply consists of a large amount of elementary calculations that transform the input step-by-step such that classification can be performed with a simple decision boundary in the final layer (shown in Figure 3.5). The fundamental building blocks of a deep

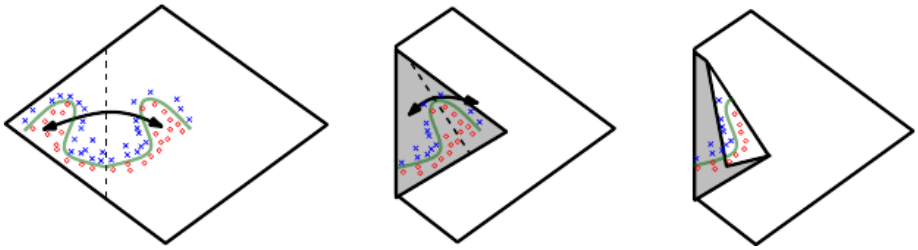

Figure 3.5: Illustration of how an interconnected set of perceptrons in a neural network is able to sequentially transform input space such that classification can be performed by means of a simple decision boundary in the last layer (Goodfellow et al., 2016).

learning model are *perceptrons*, also referred to as *neurons* (Goodfellow et al., 2016). Inside a perceptron, an input vector $\mathbf{x}$ is multiplied by a vector with weights, $\mathbf{w}$, and subsequently a scalar bias b is added to obtain the output o. More precisely, we get:

$$o = \mathbf{x}^T\mathbf{w} + b. \tag{3.12}$$

In the equation above, we follow the notation by Goodfellow et al. (2016) and transpose $\mathbf{x}$ such that it becomes a column vector rather than a row vector, enabling the matrix multiplication with the weight vector $\mathbf{w}$. Now, with a single perceptron we can only define a linear decision boundary. The power of deep learning is in stacking a large number of perceptrons in multiple layers, such that they can together form complex, non-linear decision boundaries in high dimensional spaces (see Figure 3.5). This is also where the "deep" in deep learning originates from: it refers to the many layers that such a connected network of perceptrons may have. In order to enable non-linear decision boundaries, a so-called *activation function* is applied over the perceptrons, which is typically the relu function. The relu function is defined as taking the maximum of 0 and the input. Together with the definition of the perceptron from Equation 3.12, this results in:

$$o = \max(0, \mathbf{x}^T\mathbf{w} + b). \tag{3.13}$$

Equation 3.13 is also visualized in the left part of Figure 3.6. As mentioned earlier, deep learning refers to the stacking of a large set of neurons in multiple layers, which is shown in the right part Figure 3.6. Such a stacked collection of perceptrons is also referred to as a *neural network* or a *multilayer perceptron* (MLP). In the right part of Figure 3.6, each circle represents a separate neuron, and each neuron has its own weights and bias, which are tuned using supervised learning (more detail on this learning process will follow in Section 3.3.2). The input layer takes the input data $\mathbf{x}$ as input, while subsequent layers each take all outputs of the previous layer as input. Layers that are neither the input nor the output layers are referred to as hidden layers: they are "hidden" in the model and only have an indirect relationship to the input or output. The number of units in the output layer is dictated by the type of output required. As we are interested in binary classification, the output for a single instance is a single number between 0 and 1, namely the probability of the instance being of the positive class (the probability for the negative class is then simply 1 minus this number). The output layer should thus have 1 unit for binary classification. If we would have a classification problem that involves three classes, then this would involve having three output units as shown in the right part of Figure 3.6. Technically, we could also have two output units and infer the probability for the third class, but in practice three output units are normally used to avoid confusion - the general rule is that the number of classes equals the number of output units. A final detail is that having a relu activation in the output layer would likely result in outputs that are not between 0 and 1. In order to achieve this, the sigmoid activation is applied over the perceptron in the last layer rather than the relu activation. With sigmoid activation, the calculation in the final layer will be:

$$o = \frac{1}{1 + e^{-(\mathbf{h}^T\mathbf{w}+b)}}, \tag{3.14}$$

where h refers to the output vector containing all outputs of the semi-final layer. Note that other outputs demand other activations: classification with three or more classes usually involves softmax activation while regression (i.e. a real number as output) involves linear activation (Goodfellow et al., 2016).

In this section, we have only covered the internal representations of a specific, basic type of deep learning, namely densely connected neural networks or multilayer perceptrons. We first continue to explain general aspects of deep learning (their training and regularization) in the upcoming sections. Only after understanding these basic aspects, we turn to other model architectures in Section 3.3.4 and 3.3.5.

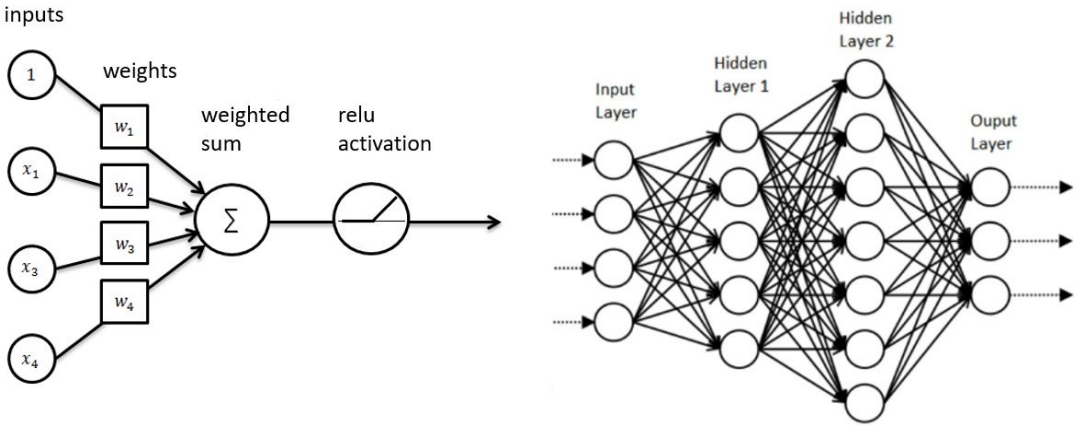

Figure 3.6: Left: The main contents of a perceptron, an individual unit in a neural network (Song, 2018). Right: A neural network or multilayer perceptron (Manero-Bastin, 2019). Each hidden layer contains a collection of perceptrons.

3.3.2 Training a Neural Network

In order to make predictions, multilayer perceptrons require estimates for the weights $\mathbf{w}$ and biases b in each perceptron, i.e. the parameters of the network. These parameters are estimated through a statistical learning process. Before the model has seen any data, the parameters are initialized in a randomized fashion. Glorot uniform initialization (Glorot & Bengio, 2010) is generally appropriate for neurons with relu activation and therefore also the standard initialization strategy in contemporary software such as Tensorflow (Abadi et al., 2015). After their initialization, the parameters are optimized through a gradient descent algorithm, which incrementally minimizes the loss function such that the predictions of the neural network resemble the actual target labels as closely as possible. In the context of binary classification, the loss function is the negative log-likelihood, which was previously shown in Equation 3.9. It is thus the same for the PFS and deep learning models in this thesis. In order to make the training process succeed, the loss function must match the activation function of the final layer. Otherwise, gradient saturation may occur, which impedes the gradient descent optimization of the parameters. Chollet (2017) provides practical advice on how to avoid this issue. When performing binary classification, the output layer should contain one unit with a sigmoid activation and binary cross-entropy loss. For classification involving three or more classes, the output layer should contain a number of units equal to the number of classes, paired with the softmax activation function and categorical cross-entropy loss. For regression (i.e. the prediction of a real-valued target label), the final layer should contain a single unit with linear activation, paired with the mean squared error loss function (Chollet, 2017). The reasons behind these choices are somewhat technical and will not be discussed here. Regardless, it is good to keep in mind that the recommendations by Chollet (2017) are but general practical advice and should thus be taken with a grain of salt. Upon closer inspection of the

interaction between output activation and loss functions, various modifications to the previously described pairs turn out to be possible. In the vast majority of practical situations, however, the previously described combinations are used.

But how exactly are the parameters of the neural network optimized? The precise dynamics of this process are rather technical and highly automated in modern software, thus we will provide only a high-level overview here. Figure 3.7 shows a schematic overview of the process. First, a *forward pass* is executed in which the equations in the neural network are evaluated on a batch of training instances. This results in a set of predictions for the target labels of the instances in the batch. Taking as input the loss function, the newly made predictions, the true target labels and the equations that generated the predictions, the neural network now performs *backward propagation*. This means that all the previously described inputs are used to derive how the parameters should be changed in order to decrease the loss. More concretely, this so-called *error signal* is the direction in which the parameters should be altered, which is also referred to as *the gradient*. In mathematical terms, the gradient is nothing more than the derivative of the loss with respect to the parameters. Calculating the derivative by hand would be tedious due to the large amount of parameters across the network. Luckily, obtaining the derivative can be formulated in a recursive fashion because of the stacking of the perceptrons: the loss function can be progressively 'unfolded' from output to input in order to obtain the gradient of the loss function with respect to the parameters. Thus, although the process of computing the gradient involves many steps that would be practically impossible to perform by hand, it can be automated to a great extent. Contemporary software such as Tensorflow (Abadi et al., 2015) automate most of this process. Once the gradient is known, the parameters are updated in its direction. The magnitude of this change (i.e. *how much* should the parameters be changed in the direction of the gradient?) is determined by the gradient descent algorithm, which will be described in more detail shortly. After the alteration of the parameters through the forward and backward pass, the next training iteration can start using the newly updated parameters, which are then again updated after a new forward and backward pass, etc.. This process of repeated forward and backward passes is continued until the training process is halted by the modeler. Typically, the training process involves passing over the full training data multiple times in a batch-wise fashion. One full pass over the training data is also referred to as an *epoch* (Chollet, 2017).

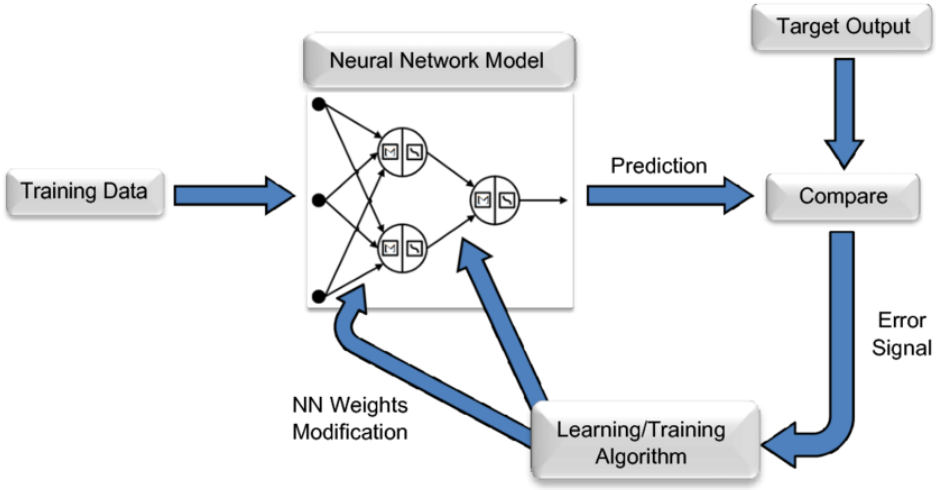

Figure 3.7: Schematic overview of the steps in a single training iteration of a neural network. Adapted from Shamsudin (2013).

Another relevant concept regarding neural network training and machine learning algorithms in general is gradient descent. Gradient descent is a family of algorithms with the main goal of optimizing a continuous, differentiable objective function by changing its parameters (Ruder, 2016).

When applied to machine learning, the objective function is the loss function of the model. This objective function is optimized in an iterative, incremental process that involves determining the gradient, computing the step size and altering the parameters using the step size and the gradient. Figure 3.8 visualizes the idea. The found minimum of the objective function need not be the global minimum - particularly in deep learning, the objective function is high-dimensional and non-linear due to the large number of parameters, rendering the landscape in which gradient descent algorithm operates extremely high-dimensional and irregular. There are usually no guarantees that gradient descent ends in a global minimum. This has led to many different forms of gradient descent that deploy various tricks to avoid getting stuck into a local minimum (Ruder, 2016). The most basic form of gradient descent has a fixed step size, which is specified through a parameter called the *learning rate*. A fixed step size, however, can lead to various issues, among which repeated overshooting of the minimum of the objective function (i.e. endlessly bouncing back-and-forth between points a little above the minimum). One way of addressing this is issue is by including *momentum*, which involves having a variable step size that changes in a manner that resembles a ball rolling down a hill. It involves taking into account the magnitude of the gradient in the past few steps, similar to how the speed of a ball rolling down a hill changes smoothly depending on whether the last few meters were steep or flat. Another way to overcome overshooting minima is to deploy an adaptive learning rate, which involves taking into account gradients in the area surrounding the current parameter settings when setting the current step size. Other, more recent gradient descent algorithms even use a combination of momentum and an adaptive learning rate (Ruder, 2016). In most situations, the eventual outcome of the optimization procedure is very similar for the variants with momentum, adaptive learning rates or both (Goodfellow et al., 2016). Some experiments seem to indicate that the Adam gradient descent algorithm developed by Kingma & Ba (2014), which uses a combination of momentum and an adaptive learning rate, converges slightly faster to a reasonably low local minimum, but there is by no means any universally superior gradient descent algorithm. Only empirical testing on the specific optimization problem at hand can be used to definitively determine which gradient descent algorithm is best for which task. Finally, an effective way to decide when to halt the gradient descent optimization is through monitoring the improvement in the objective function over the last few steps. If this improvement has been small for a number a steps, then this is an indication that the algorithm is converging and that training can be stopped. This way of halting gradient descent algorithms is called *early stopping* (Goodfellow et al., 2016).

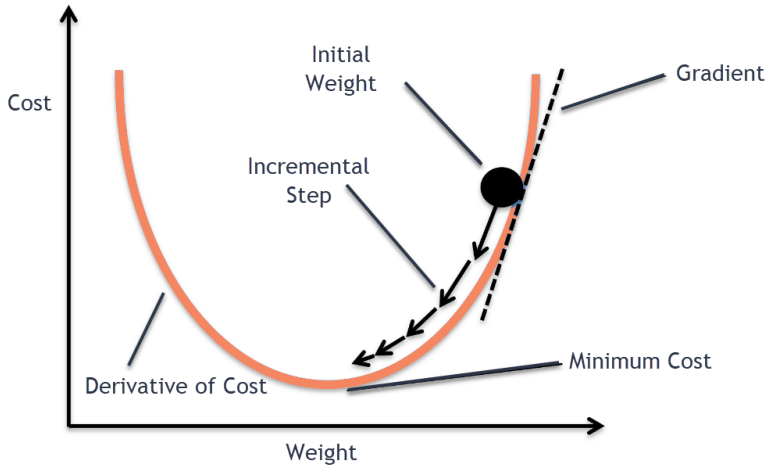

Figure 3.8: Visualization of gradient descent. Adapted from Lanham (2020).

In order to make the error landscape in which gradient descent has to operate less "shaky", it is common practice to normalize the input data before passing it to a deep learning model (Chollet, 2017). Normalization (i.e. subtracting the mean and dividing by the standard devation) evens

the scale of different features and thereby reduces the need for the model to have coefficients with strongly varying magnitudes, which in turn makes the error landscape less "shaky". The output of individual units, however, can still have strongly varying magnitudes because of the relu activation function. This can render the scales of coefficients corresponding to layers that are deeper in the model strongly uneven, which then still results in a "shaky" error landscape. A recent innovation in deep learning that resolves this issue is batch normalization (Ioffe & Szegedy, 2015), which uses clever tricks to normalize the outputs from intermediate layers to make the gradient descent more stable. This technique is found to be particularly effective in large networks, which sometimes even turn out to be untrainable without batch normalization layers (Chollet, 2017). The positive effect of batch normalization on the learning process, however, is mainly noticeable in large, deep networks such as pre-trained neural networks for computer vision problems. This thesis does not involve a model of such extremely large size - the number of layers we use is small compared to for instance the Xception and Inception networks, which each have hundreds of layers and tens of millions of parameters.

3.3.3 Regularization

Because deep learning models usually involve a large amount of parameters, they have a strong natural tendency to overfit. To counteract this tendency, many methods are available to regularize deep learning models.

A first possible way of regularizing machine learning models is L1- and/or L2-regularization (Goodfellow et al., 2016). These methods add a penalty term to the loss function that increases when the magnitude of the coefficients increases. This effectively pushes the learning procedure to choose smaller coefficients, thereby reducing model capacity and overfitting. The main distinction between L1- and L2-regularization is that the former induces sparsity, meaning it has a tendency to keep parameters either non-zero or to completely push them to zero. This has proven to be a valuable property when applied to linear or logistic regression models, as L1-regularization in such regression models corresponds to feature selection (James et al., 2013). L1-regularization then becomes an efficient way to let the model decide which features should be selected. Whether this sparsity-inducing effect is a desirable property in a deep learning model is not so straightforward because a weight with a value of 0 does not implicitly correspond to feature selection in a neural network (as opposed to linear or logistic regression). This is because other weights of other units that involve the same feature can still be non-zero. Once again, only empirical testing of the specific model on the specific application at hand can definitively establish which form of regularization results in the best predictive performance. As the extent of regularization can tuned with a parameter, and because L1- and L2-regularization can be combined and applied over any set of weights or units to the liking of the modeler, the number of possible configurations of this type regularization quickly grow huge in a deep learning model.

Other forms of regularization that are specific to neural networks exist as well. Simply choosing a smaller number of perceptrons and/or layers in the network can be seen as such a form of regularization as it reduces the number of parameters and therefore the capacity of the model. Another deep learning-specific form of regularization is *dropout*, which is a type of layer that can be put between other layers in the network, and that randomly sets some of the outputs of the preceding layer to 0 during training (Goodfellow et al., 2016). This might seem a somewhat odd and arbitrary way of regularization, but it is actually an efficient, deep-learning specific way of ensembling multiple models, which has long been known to reduce overfitting. Dropout corresponds to ensembling because it 'alters' the model randomly at training time, resulting in a network that is more robust to noise and therefore less prone to overfitting. The number of outputs that are set to 0 during training is a tunable parameter determined by the modeler, and is referred to as the *dropout rate*. Dropout is empirically demonstrated to be effective for many different types of deep learning models (Goodfellow et al., 2016), both as a replacement of L1- and L2-regularization and in addition to them. A major advantage of dropout regularization in comparison to L1- and L2-regularization is that dropout has a considerably smaller number of

tunable hyperparameters: one merely needs to determine where to put dropout layers and what the dropout layers should be, which is involves much fewer degrees of freedom than determining how much L1- and how much L2-regularization to apply in every individual layer, unit or weight of the network.

Over time, even more methods of regularization have been devised for deep learning models. Examples of popular methods are multitask learning (training the model on other learning tasks jointly with the main task), data augmentation (creating variations of the input data such as mirroring and resizing images), parameter sharing (forcing the same or similar parameters in distinct neurons) and adversarial training (feeding reverse-engineered examples to the model that are wrongly classified) (Goodfellow et al., 2016). In fact, even the previously described early stopping method can be seen as a form of regularization as it involves halting model training and thereby reducing overfitting, although the reduction in model capacity will usually be minor due the small parameter modifications near convergence. The scope of this research is restricted to the most widely used and computationally inexpensive forms of regularization.

3.3.4 Recurrent Neural Networks

A type of architecture that has been found particularly effective for predictive modeling on data that involves temporal structures is the recurrent neural network (RNN) (Goodfellow et al., 2016). RNNs have significantly advanced the state-of-the-art in tasks such as speech recognition and natural language processing - two tasks in which pattern recognition over temporal sequences is a crucial aspect. The internal representations that recurrent neural networks rely on differ considerably from the previously covered densely connected neural networks. The remainder of section therefore unveils the inner mechanisms of RNNs.

A first major difference between RNNs as opposed to a densely connected neural network (DNN) is that they require a differently structured input (see Table 3.1 and Figure 3.9). If we use a densely connected neural network on a time series, then a single instance corresponds to the feature values at a single timestep. This is problematic if the past of the time series contains relevant information for the prediction at the current timestep: the past observations are simply different instances, and can therefore not be taken into account directly when predicting for the current instance. This is in contrast with RNNs, where a single instance is a full time series. In this way, the past of the time series is available to the model, making it possible to learn temporal structures from a dataset. Note that the example in Table 3.1 and Figure 3.9 has but one feature, but we could equally well have multiple features. Furthermore, it is relevant to realize that an RNN is not restricted to having a single, very long instance as its input as illustrated in the right part of Figure 3.9: it can also take multiple instances as input while resetting its state upon receiving a new instance. In our scenario, one instance corresponds to one patient, with each patient having a time series of clinical variables measured over time. Likewise, an RNN can be configured to predict one label per patient, but it can also be configured to predict multiple times for each patient, e.g. each time that an observation is made (at each timestep). This flexibility is one of the main reasons why RNNs have been found to be so effective for many prediction tasks that involve temporal dynamics.

Next, we turn to the internal representations that RNNs rely on to learn temporal patterns from data, and use the notation by Goodfellow et al. (2016). An RNN unit maintains a hidden state $h^{(t)}$ at each timestep that is formed by computing a function over the input $x^{(t)}$ at the current timestep and the previous hidden state using a learnable set of parameters θ. More concisely, we can represent the (hidden) state of the RNN at timestep t, $h^{(t)}$, as:

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta). \tag{3.15}$$

The hidden states thus have a recurrent relationship, hence the name recurrent neural networks. This property allows for unfolding the network from output to input in order to train the model

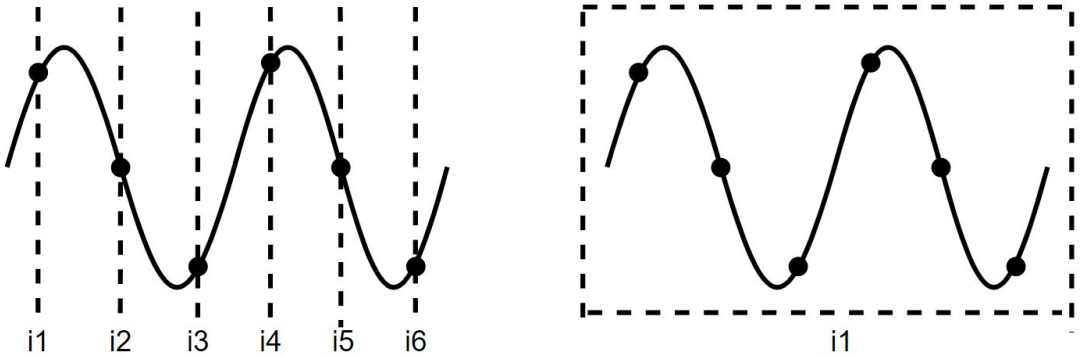

Figure 3.9: Visualization of the difference between instances of a time series fed as input to a DNN as opposed to an RNN. The filled dots represents observations in the time series. This figure corresponds to Table 3.1.

Table 3.1: Demonstration of the difference between instances of a time series fed as input to a DNN as opposed to an RNN. This table corresponds to Figure 3.9.

X	**Y**	**Instance in DNN input**	**Instance in RNN input**
0.5	0.93	1	1
1.0	0.45	2	1
1.5	0.05	3	1
2.0	0.97	4	1
2.5	0.47	5	1
3.0	0.09	6	1

via backpropagation in a similar manner as was previously described in Section 3.3.2.

Figure 3.10 shows the architecture of an RNN unit more in-depth. This particular RNN has an output at each timestep, though a single output after all timesteps is also possible with a slightly different architecture. The learnable parameters θ of the model are the connections U, V and W. U is the so-called *input-to-hidden connection*, which is a set of weights that are multiplied by the input in order to obtain the hidden state. Like in the neurons of an DNN, the hidden state $h^{(t)}$ is transformed using an activation function before further processing. A difference with a DNN is that the hidden states are activated using the tanh function instead of the relu function, which is necessary because with relu, the values in the hidden state can only increase, which is undesired. V is the *hidden-to-output connection*, in which the hidden state is similarly multiplied with the weights in V to obtain the output. Finally, W is the *hidden-to-hidden connection*. Mathematically, we can describe the operations of an RNN unit at each timestep as follows:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)}, \tag{3.16}$$

$$h^{(t)} = \tanh(a^{(t)}), \tag{3.17}$$

$$o^{(t)} = c + Vh^{(t)}. \tag{3.18}$$

$o^{(t)}$ usually receives some kind of output activation as well, which is the sigmoid function in our context as we are performing binary classification. b and c are biases.

Unfortunately, the basic RNN unit is of little practical use because it is extremely difficult to train - its gradients tend to vanish or explode such that learning long-term dependencies becomes

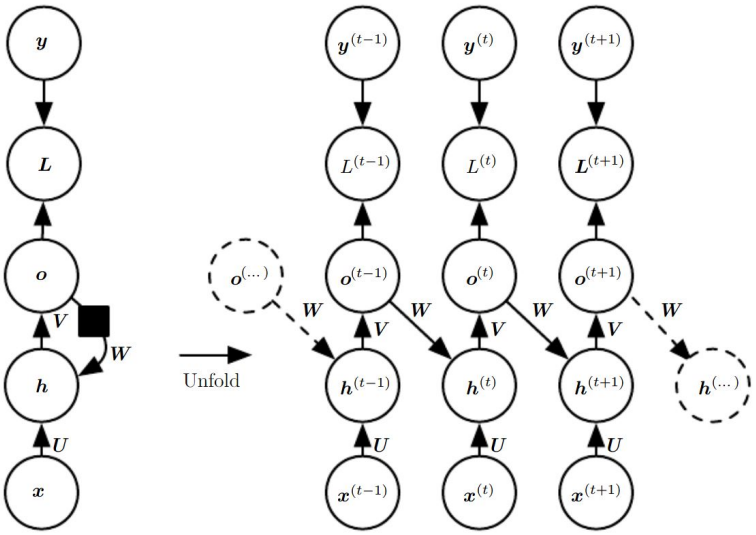

Figure 3.10: Illustration of the internal representations of an RNN that has an output at each timestep. The left shows the concise representation, in which the timesteps are 'folded'. The right shows the unfolded representation. y denotes the target label, L the loss and o the output. In the training phase, W connects the actual target label y with the hidden state h to make it easier to learn the network - a clever trick which is called *teacher forcing* (Goodfellow et al., 2016). At testing time, W connects subsequent hidden units, as also specified in the equations but left out in the figure. Figure adapted from Goodfellow et al. (2016).

practically impossible (Goodfellow et al., 2016). A modified version of the RNN unit, the Long-Short Term Memory (LSTM) unit, overcomes this issue. Figure 3.10 shows a graphical overview of the additional operations in an LSTM unit compared to an RNN unit. The LSTM adds three gates to the RNN, namely an input, forget and output gate. All gates have separate parameters and take the features $x^{(t)}$ and the previous hidden state $h^{(t-1)}$ as input - meaning they are controlled by the 'context'. The input gate $f_i^{(t)}$ of LSTM unit i enables the network to learn which inputs are important for the hidden state, the forget gate $g_i^{(t)}$ enables the network to independently learn the time scale of temporal patterns, and the output gate $q_i^{(t)}$ enables the network to learn when to shut off the output (such that irrelevant parts in the sequence can be ignored). Mathematically, the gates are extremely similar:

$$f_i^{(t)} = \sigma(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}), \tag{3.19}$$

$$g_i^{(t)} = \sigma(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}), \tag{3.20}$$

$$q_i^{(t)} = \sigma(b_i^q + \sum_j U_{i,j}^q x_j^{(t)} + \sum_j W_{i,j}^q h_j^{(t-1)}). \tag{3.21}$$

Each gate is activated with the sigmoid function, σ, such that their values are squeezed between 0 and 1. Each gate also has its own bias b_i, own input-to-hidden connection $U_{i,j}$ and recurrent connection $W_{i,j}$, all of which are indexed by the particular unit i and feature j. Note that we technically cannot call $W_{i,j}$ the output-to-hidden connection anymore in the LSTM unit, because only the output gate $q_i^{(t)}$ is directly connected to the output. Nevertheless, the relationship between

the inputs and the hidden states and is as follows:

$$s_i^{(t)} = f_i^{(t)} h_i^{(t-1)} + g_i^{(t)} \sigma(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}), \tag{3.22}$$

$$h_i^{(t)} = \tanh(s_i^{(t)}) q_i^{(t)}. \tag{3.23}$$

Here, $h_i^{(t)}$ is the output hidden state of the LSTM unit, while $s_i^{(t)}$ is the internal hidden state before recurrect activation. The internal hidden state, $s_i^{(t)}$, is again activated with the tanh function, similarly as in a regular RNN unit.

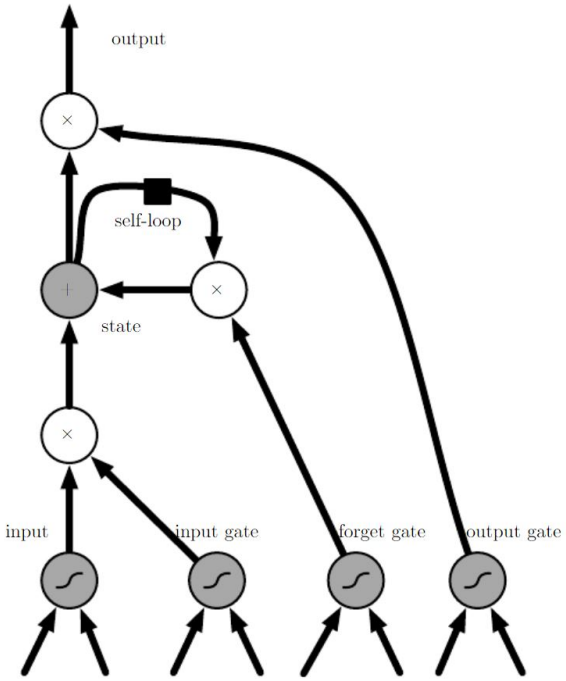

Figure 3.11: Visualization of the operations in an LSTM unit. The inputs to each operation at the bottom are $x^{(t)}$ and $o^{(t-1)}$ (the output at the previous timestep). Figure adapted from Goodfellow et al. (2016).

A subsequent innovation of the LSTM unit is the Gated Recurrent Unit (GRU), which is very similar but has only two gates, namely an "update" gate u and a "reset" gate r (Goodfellow et al., 2016). Both gates can be individually controlled by the statistical learning process to amplify or ignore information from previous states through time. Having two gates instead of three makes the GRU cheaper to train (fewer trainable parameters) while its representational capacity is almost indistinguishable from the LSTM. Extensive empirical works have shown that both the LSTM and GRU have similar performance, and that various other modified versions of the LSTM deliver no consistent performance increase across many different statistical learning tasks (Greff et al., 2016; Jozefowicz et al., 2015). Therefore, the GRU unit is the recurrent unit of choice for this work. The definitions of gates and the forward propagation of the GRU are as follows:

$$s_i^{(t)} = u_i^{(t-1)} h_i^{(t-1)} + (1 - u_i^{(t-1)}) \sigma(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} r_j^{(t-1)} h_j^{(t-1)}), \tag{3.24}$$

$$u_i^{(t)} = \sigma(b_i^u + \sum_j U_{i,j}^u x_j^{(t)} + \sum_j W_{i,j}^u h_j^{(t)}), \tag{3.25}$$

$$r_i^{(t)} = \sigma(b_i^r + \sum_j U_{i,j}^r x_j^{(t)} + \sum_j W_{i,j}^r h_j^{(t)}). \tag{3.26}$$

Again, $s_i^{(t)}$ is activated with the tanh function to obtain $h_i^{(t)}$. Figure 3.12 shows a visualization of a GRU unit. $\tilde{h}$ represents the right part of Equation 3.24 (after σ). Concretely, the update gate u controls the balance between (1) directly taking into account information from the previous state and (2) taking into account the processed input of the current timestep. The reset gate r determines how much information of the current state should be used when processing the input data of a timestep.

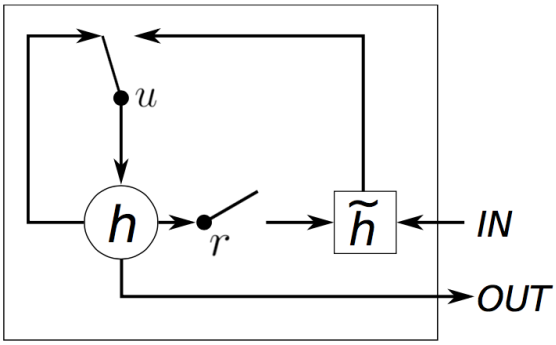

Figure 3.12: Visualization of the operations in a GRU unit. Figure adapted from Cho et al. (2014).

3.3.5 Convolutional Neural Networks

Another type of deep learning model architecture that deserves consideration is the convolutional neural network. These models have significantly advanced the state of the art in computer vision in recent years (Goodfellow et al., 2016). Though computer vision is their main field of application, they have also been proven effective for modeling temporal data - audio data in particular. The convolution operations that they rely on, however, are only effective on image-related data and time series that are evenly sampled and not too sparse. Unfortunately, the research in this thesis involves sparse and irregular multivariate time series (more detail in Section 4.1.1), which renders the convolutional neural network architecture ineffective. Other related works that test deep learning models on multivariate clinical time series also do not consider convolutional neural networks (Harutyunyan et al., 2019; Purushotham et al., 2018). We will therefore not discuss these types of neural networks in any more detail here and refer the interested reader to the chapter on convolutional neural networks in the work by Goodfellow et al. (2016).

3.4 Model Assessment Metrics

Assessing the performance of a supervised learning model in a quantitative fashion can be non-trivial because of the many different metrics that are available. Therefore, we dedicate this section to a description of the intuitions behind the model assessment metrics that were considered in this research. Our scope is restricted to metrics for assessing binary classification, as our experiments solely involve this type of predictive modeling.

Perhaps the most commonly used and most basic evaluation metrics are those that are based on the confusion matrix (Provost & Fawcett, 2013). Table 3.2 shows a confusion matrix and a selection of straightforward metrics based upon this matrix and that are commonly used in predictive modeling. We also provide a set of additional definitions based on the confusion matrix for later reference:

Table 3.2: Confusion matrix including the most widely used corresponding model performance evaluation metrics. Adapted from Fawcett (2006).

		True Class		
		Positive (patient death)	**Negative (patient survival)**	
Predicted Class	**Positive (patient death)**	True Positive (TP)	False Positive (FP)	Precision: $\frac{TP}{TP+FP}$
	Negative (patient survival)	False Negative (FN)	True Negative (TN)	
		True Positive Rate (=Recall) $\frac{TP}{TP+FN}$	False Positive Rate $\frac{FP}{FP+TN}$	Accuracy $\frac{TP+FP}{TP+FP+TN+FN}$

- condition positive = p = TP + FN
- condition negative = n = FP + TN
- prediction positive = $\hat{p}$ = TP + FP
- prediction negative = $\hat{n}$ = FN + TN

Computing any of the metrics based on the confusion matrix first requires the model to predict either the positive class or the negative class for each instance, which at first sight seems to be not in line with the PFS and deep learning models described in Section 3.1.3 and 3.3: the outputs of these algorithms are class *probabilities* (i.e. a number between 0 and 1) rather than binary classes (i.e. either 0 or 1). This raises the question how to convert the class probabilities into binary predictions for class membership. One way to do so is to define a simple cutoff at a probability of 0.5: if the prediction is below 0.5, the model predicts the negative class, and if the prediction is higher than or equal to 0.5, the model predicts the positive class. However, we are free to choose any other threshold, and the threshold clearly affects the TPs, FPs, FNs and TNs. The choice that we make for the threshold thereby also affects the model assessment metrics. But how then, do we choose what threshold to use? Generally, the context and application of the model must be taken into account to determine an optimal threshold, for instance through attaching costs to TPs, FPs, FNs and TNs (Provost & Fawcett, 2013). Determining such costs, however, is far from trivial in many situations. For instance, when predicting septic shock mortality, the costs of an erroneously high mortality prediction would be difficult to determine as it would vary strongly across different patients and different situations.

One way to avoid having to determine a class prediction threshold is to assess model performance with metrics that do not require manually choosing a class prediction threshold. Perhaps the most widely used metric with this property is the area under the receiver operating characteristic curve (ROC-AUC) (Fawcett, 2006; Provost & Fawcett, 2013). The ROC curve of a classifier is a plot of the true positive rate versus the false positive rate (definitions of both can be seen in Table 3.2). An example of an ROC curve can also be seen in the left part of Figure 3.13. Each point in an ROC plot corresponds to a particular choice for the threshold and hence a particular TPR and FPR. Randomly guessing the class label results in TPR=FPR, thus all points above the corresponding diagonal from (0,0) to (1,1) indicate that the choice for the threshold corresponding to the particular point was superior to random guessing et vice versa. Furthermore, each prediction model has an individual line in an ROC plot, making it somewhat difficult to compare model performance, particularly when the ROC curves of two models cross multiple times. A solution to this problem is aggregating the ROC curve of a model by computing the area under the ROC curve (ROC-AUC), which results in a scalar value for model performance that can easily be (statistically) compared.

The ROC curve, and thereby also the ROC-AUC metric, have the often desirable property that they are independent of the underlying distribution of the classes. More concretely, it does not

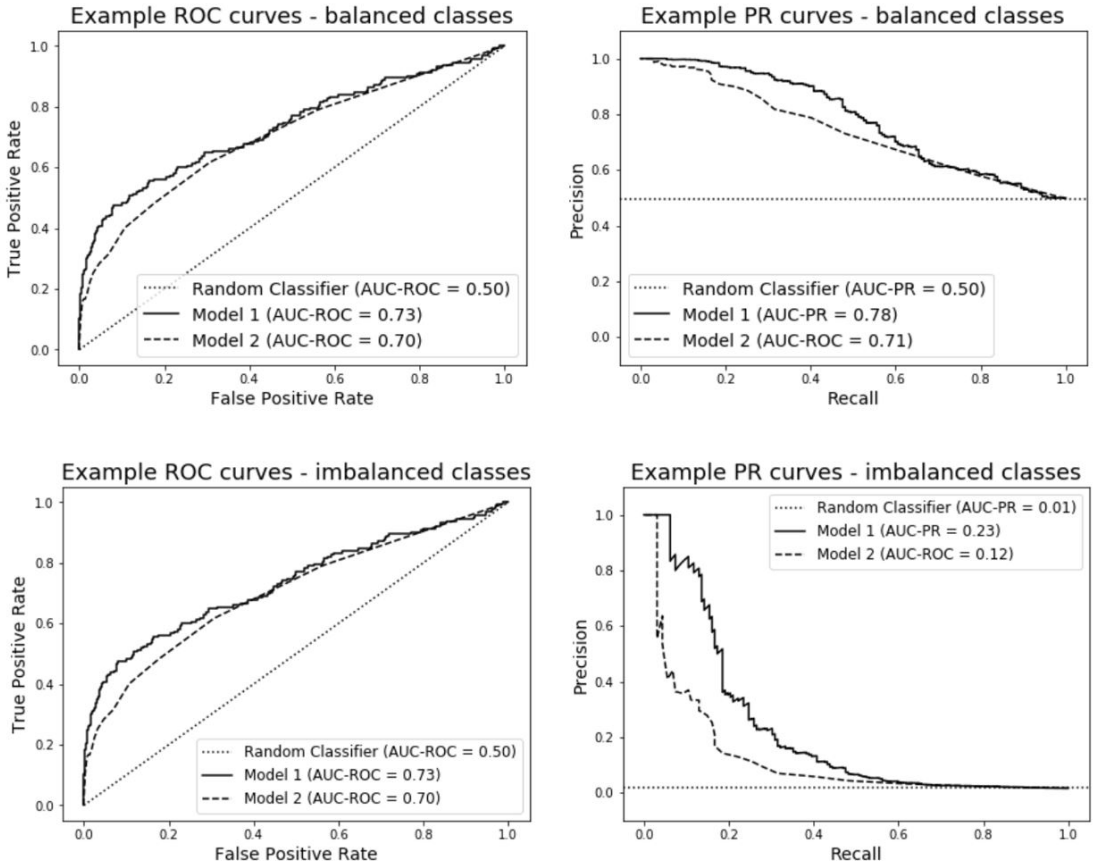

Figure 3.13: Examples of ROC and PR curves to demonstrate the difference between the two when the class distribution changes. In a scenario where classes are balanced (top), model performance is similar on both the PR and ROC curves. When the class distribution becomes strongly imbalanced (bottom), the ROC curves do not change while the PR curves show that performance has decreased strongly and that much room for improvement remains. This example is inspired by Fawcett (2006), and we refer the reader to Appendix C for the code implementation to obtain these plots.

matter whether 70% or 1% of the instances belongs to the positive class: the ROC-AUC remains the same. However, it is sometimes argued that this property makes the ROC-curve and the ROC-AUC inappropriate for situations with a strong class imbalance, because the ROC-AUC then paints a rather optimistic picture of model performance (Davis & Goadrich, 2006; Saito & Rehmsmeier, 2015). A frequently posed solution is to use precision-recall (PR) curves and the area under these curves (PR-AUC) instead. PR curves are similar to ROC curves, but the difference is that the PR curves put recall on the x-axis as opposed to the false positive rate (the true positive rate is a synonym for precision, i.e. the metrics on the y-axis are the same between in both plots). This difference makes PR curves sensitive to changes in the underlying class distribution, accurately reflecting possible low model performance compared to the baseline of randomly guessing class membership in this scenario. Figure 3.13 illustrates how the ROC-curve and PR-curve differ in the light of class imbalance. Note that in a PR-curve, the performance of the baseline classifier that randomly guesses class membership is a horizontal line with a precision equal to the fraction of instances belonging to the positive class. The baseline performance in a PR-curve thus differs between different datasets with different underlying class distributions. This in contrast with the ROC-curve, in which random guessing always corresponds to the diagonal from (0,0) to (1,1), no matter the target label distribution.

A recently proposed and less commonly used threshold-averaging model assessment metric is the Area Under the Kappa Curve (AUK) (Kaymak et al., 2012). The Kappa curve involves a plot of the false positive rate versus Cohen's kappa. Figure 3.14 shows this curve for the same scenario as in Figure 3.13. Cohen's kappa is defined as

$$\kappa = \frac{a - p_c}{1 - p_c}, \tag{3.27}$$

in which a is the accuracy as in the confusion matrix in Table 3.2 and p_c the probability of predicting the correct class due to chance, i.e. $p_c = p\hat{p} + n\hat{n}$. Cohen's kappa can have values between -1 and 1, where values below 0 indicate the performance is worse than random guessing. Like in the other two curves, each value for the positive/negative class prediction threshold has its own kappa value, and we can compute the area under the Kappa curve to summarize the plot into a single number. The Kappa curve is sensitive to the underlying class distribution much like the PR-AUC. However, it possesses the desirable additional property that it has a unique maximum that can be used to select the optimal threshold, i.e. the threshold that provides the biggest improvement over randomly guessing class membership. This is an interesting, alternative method of determining an optimal threshold without the need for numerical (mis-)classification costs and benefits. Another convenient property of the Kappa curve compared to the PR-curve is that the baseline performance of randomly guessing class membership is always equal to 0, no matter the class distribution. This makes comparing the performance of a prediction model to a random classifier easier than in a PR-curve, where the performance of a random classifier is dependent on the class distribution.

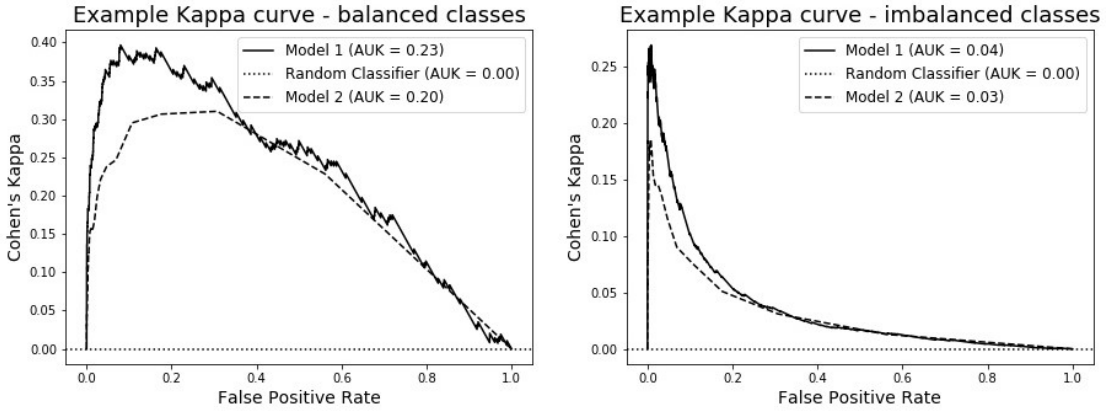

Figure 3.14: Example of a Cohen's kappa curve as used for computing the AUK metric. This figure corresponds to the imbalanced class scenario from the bottom row of Figure 3.13.

3.5 Explainable Machine Learning

In some domains, it may not be important *how* a machine learning model generates its predictions as long as they are correct for the vast majority of instances. An example is predicting which e-mails are spam and filtering them - as long as false positives and false negatives are quite rare, it is not disastrous if they occur and they can easily be rectified (e.g. by moving the e-mail out of the spam folder). In such cases, a quantitative assessment of model performance on particular metrics (such as those in the previous section) is sufficient to conclude whether a machine learning model will be accurate enough for deployment. Obviously, this is much different in our application domain as erroneously predicting survival while the patient dies in reality could have disastrous consequences. Therefore, it is a requirement in this research that we can interpret the reasons for particular predictions as to validate them in light of clinical inuitions about increased mortality

risk. In this section, we take a deeper dive into model interpretability, which is also generally referred to as *eXplainable Artificial Intelligence (XAI)* or *Interpretable Machine Learning* (Adadi & Berrada, 2018; Arrieta] et al., 2020; Molnar, 2019). Concretely, we examine what it means for a prediction model to be interpretable, how this translates to the models used in this research, and a handful of popular techniques to achieve interpretability.

A general definition of model interpretability is *the degree to which a human can understand the cause of a decision* (Miller, 2019). In our context, the cause of the decision is how the model output (mortality probability) relates to the inputs (clinical variables of a patient, e.g. blood pressure). To give an example, if a prediction by a PFS involves the activation of 500 rules at once, then a human is clearly unable to oversee the dynamics of the model, and it will be unclear how the model output relates to the input. In this case, the human clearly cannot understand the cause of the decision, thus the model is not interpretable. This example also demonstrates that in general, human-friendly explanations should be *selected* (Miller, 2019): one is usually not interested in a complete attribution of all factors that influenced the decision, but one is interested in the few main determinants that caused the decision. For instance, it suffices to explain that the stock prices recently went down because of the Coronavirus, while in fact, the complex interplay of the virus with governmental measures, the economy and human behavior is what caused the stock prices to decline. In addition to being *selected*, human-friendly explanations are *contrastive*: the explanation should clearly reveal as to why the stock prices did *not* stay at the same level. Applied to machine learning predictions, contrastiveness can also include providing information about so-called *counterfactual instances* that demonstrate what changes in an instance are required to flip the prediction around (e.g. when would the model predict mortality instead of survival for this patient at this time?). Finally, human-friendly explanations are *social*, which in our interpretation means that they should be formed with awareness about the audience of the explanation and the application domain. Explanations should thus clearly link to the clinical intuitions about increased mortality risk in this work.

Another fundamental aspect regarding prediction model interpretability is that usually, it is at odds with the objective of maximizing accuracy or another model performance metric (Gacto et al., 2011). If one wishes to simplify a model in order to make it easier to interpret by a human, then the most straightforward remedy is to remove elements from the model. This increases interpretability at the cost of a reduction of the model performance on the evaluation metric. In most situations, model performance and model interpretability is a trade-off, and the application domain should dictate which balance between the two is appropriate.

3.5.1 Interpretability for Fuzzy Systems

Having established a conceptualization of interpretability, we now turn to the implications for probabilistic fuzzy systems. Gacto et al. (2011) and Alonso et al. (2015) provide two excellent, comprehensive reviews on the various facets of fuzzy systems that are relevant with respect to interpretability. Here, we will discuss those facets that are most relevant for our particular type of fuzzy systems, namely PFS. As the definition of interpretability already indicates, an interpretable PFS has a number of rules that allows a regular mortal to keep its contents into memory without an excessive long-term effort to memorize the system. This means that the PFS should not have more than a handful of rules. The number of rules can be limited manually when specifying the model up-front, or it can be reduced by merging similar rules after fitting the model. Furthermore, an individual rule should have a readable description length, which requires a rule to involve not more than a handful of dimensions and membership functions. Two other relevant criteria are the *redundance* and *irrelevance* of rules. A rule is redundant if it has large overlap with another rule (i.e. fired at the same time), while a rule is irrelevant if it is either nearly always activated (for a majority of instances) or hardly ever activated (only for a negligible fraction of instances). The Jaccard similarity coefficient S between two rules A and B is a quantitative criterion that can be

used to assess these two aspects, and is defined as follows (Setnes et al., 1998):

$$S(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\sum_{i=1}^{I} \mu_A(x_i) \wedge \mu_B(x_i)}{\sum_{i=1}^{I} \mu_A(x_i) \vee \mu_B(x_i)}. \tag{3.28}$$

It is straightforward how the equation above can be used to assess rule redundancy (simply plug in two rules of the PFS for A and B). For assessing irrelevance, we must set A or B to be the universal set (i.e. the set in which all observations i have an activation μ of 1), in which case $S = 0$ implies that a rule is never activated and $S = 1$ implies that a rule is always activated. Thus, values of S close to 0 or 1 will indicate rule irrelevance in this case.

3.5.2 Interpretability for Other Models

The methods described above are exclusively intended for fuzzy systems. However, this research also involves deep learning, with at its core the objective to compare both types of models. Interpretability methods specifically for deep learning unfortunately seem to be destined almost exclusively for computer vision, and the libraries to implement these techniques are still very immature and thus challenging to use (see e.g. Olah et al. (2017), Olah et al. (2018) and Sicara (2019)). For these reasons, we leave deep learning-specific interpretability methods for future research. Luckily, various model-agnostic interpretability methods have been devised in prior works that are more directly applicable to our context and more mature. The fact that these methods can also be applied to any prediction model conveniently enables a side-by-side comparison of PFS and deep learning, thereby directly addressing interpretability with respect to the main research problem of this work.

Global Interpretability

A crucial question when interpreting prediction models is how the input features generally relate to the output (i.e. global effects). More concretely, is it useful to be able to inspect whether the relationship between a feature and the target label is positive, negative, U-shaped, etcetera. In his work, Molnar (2019) states various ways to obtain such insights in a model-agnostic manner. The first is through partial dependence plots (PDPs), originally proposed by Friedman (2001). Figure 3.15 shows two examples of PDPs while Figure 3.16 visualizes how the values on a PDP are calculated (the graphical intuition behind Equation 3.29). Partial dependence plots show a self-chosen feature on the x-axis and the target label on the y-axis. To compute the value of $\hat{f}_{x_S}(x_S)$ at a particular value of feature x_S, we first set all instances in the data to have that particular value for the chosen feature S ($x_1 = 0.75$ in Figure 3.16). Then, we generate predictions over all these 'new' instances with the frozen value for x_S, and average all predictions. This process is repeated for all values of x_S that the modeler desires to have on the x-axis of the PDP. This method of calculating the values on the y-axis of a PDP is also known as the Monte Carlo method (Molnar, 2019). More formally, we can define this process as follows:

$$\hat{f}_{x_S}(x_S) = \frac{1}{n} \sum_{i=1}^{n} \hat{f}(x_S, x_C^{(i)}), \tag{3.29}$$

in which x_C denotes the complement features that are not on the x-axis of the PDP (other features than x_S), and where i refers to an instance in the data. This process can be repeated for multiple features in the data to gauge how particular values of a feature generally influence the predictions of a supervised learning model. In brief, the interpretation of a point along the y-axis in a PDP is: 'the average value of the target label at this feature value'.

Though PDPs provide some insight into global model behavior, it is relevant to realize that they paint but a limited picture of feature influence: feature interactions can push the actual predictions at a particular value for a particular feature up and down from the effect shown in

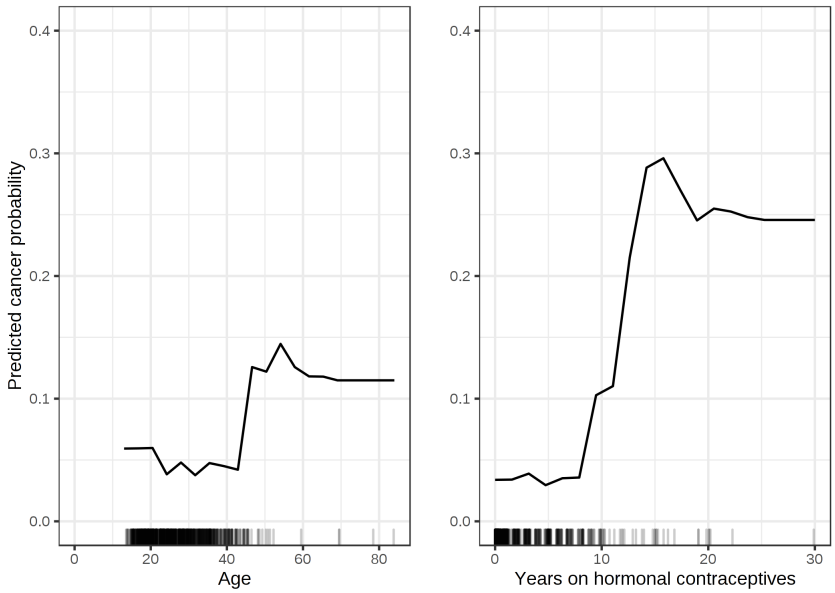

Figure 3.15: Two examples of partial dependence plots (PDPs). Figure adapted from Molnar (2019).

the PDP. Thereby, PDPs involve the assumption that features are independent, which is often violated. For example, forget the ticks on the axes of Figure 3.16 for a moment, and suppose x_1 denotes the horsepower of a car while x_2 denotes its maximum speed. Clearly, these two factors are correlated. If both a Ford Fiesta and Bugatti Veyron are in the data, then while calculating the PDP for horsepower, we would inevitably include a prediction for a car with the horsepower of a Ford Fiesta and the maximum speed of a Bugatti Veyron. Clearly, this is highly unrealistic, and because such points are not in the training data, model behavior can be odd and unstable in such regions. Apley & Zhu (2016) propose the accumulated local effects (ALE) plot, which answers the same question as the PDP ("how do particular values for a feature generally influence model predictions?") yet overcome this issue.

Figure 3.17 shows the intuition behind ALE plots. Now, only instances within an interval surrounding the chosen value of x_1 are used to estimate the effect of the feature value on the predictions. Thereby, the ALE avoids having to make a prediction for the Ford Fiesta that was able to reach speeds of 400 km/h. Furthermore, the predictions at the end of the interval are compared to those at the start of the interval, i.e. we average ove *differences* in predictions rather than over predictions themselves. Then, we accumulate these averaged differences from the starting interval up to the current interval in order to estimate the general effect of a particular feature value on the predictions. The interpretation of the y-axis in ALE plot is thus slightly different than for a PDP plot - it is centered on 0 and shows the difference to the mean prediction. The whole process of computing an ALE is quite technical, so we refer the reader to the section on ALEs in Molnar (2019) and the original paper by Apley & Zhu (2016) for more detail. For now, it suffices to know that ALEs avoid unrealistic instances at the expense of a more conceptually complex calculation and a choosable parameter that could influence how the plot looks like (i.e. the number of intervals). We will demonstrate in the results of this thesis that this influence seems to be quite limited, probably due to the accumulation of the effects. Albeit the calculations underlying the construction of an ALE can be considered more conceptually complex, they are generally cheaper than those for a PDP, mainly because we need not pass over all instances in the data to estimate a value on the y-axis in an ALE.

Another intuitive way to asses how a particular feature influences the predictions in a model-

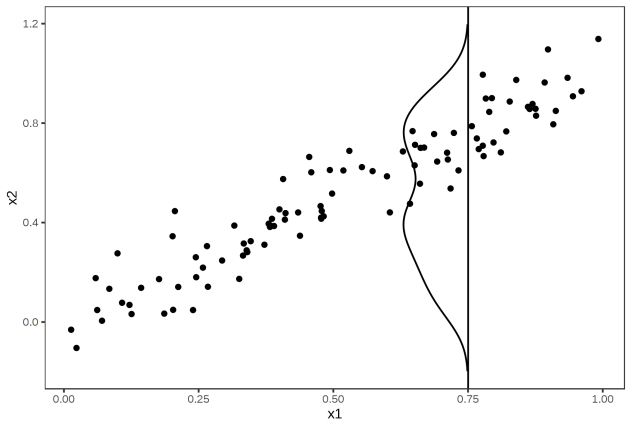

Figure 3.16: Example demonstrating how the values on a PDP are calculated, along with the here misguided assumption that the features are independent and that the distribution of x_2 at $x_1 = 0.75$ is equal to its distribution overall. Note that two features are shown here (x_1 and x_2), so the y-axis is *not* $\hat{f}_{x_s}(x_s)$, i.e. this figure is not a PDP itself! Figure adapted from Molnar (2019).

agnostic manner is through permutation feature importance (Molnar, 2019). This method provides a different perspective on the effect of a feature than PDPs and ALEs: it provides insight into how the model error increases if we 'destroy' a feature, i.e. artificially permute its values such that the feature has no relationship with the target label anymore. Calculating permutation feature importance is very straightforward: we first train a model and asses its error on the testing data, as usual. Then, we permute the values of a feature on all instances in the test set by randomizing them artificially, thereby breaking the relationship between the feature and the target label. Subsequently, we generate predictions on the modified test set and asses how the model error has increased due to the 'destruction' of the feature. This allows one to gauge the influence of a particular feature on the predictions of a model, and whether that influence affected model performance positively or negatively. Unfortunately, permutation feature importance does not reveal anything regarding how *a particular value for a particular feature* generally influences the *direction* of the prediction, in contrast to PDPs and ALEs. Therefore, permutation feature importance is a complementary global model interpretability method with respect to PDPs and ALEs.

In this section, we have discussed but the tip of the iceberg of global, model-agnostic interpretability methods - our overview is restricted to relatively intuitive and popular methods that fit in this research. For more methods, we refer the reader to the excellent, accessible work by Molnar (2019) and the many papers that underlie his work.

Local Interpretability

The global interpretability methods described in the previous section suffer from the limitation that they provide little information about individual predictions for individual instances - they only provide insight into global model behavior. In many application domains of machine learning, it is valuable to have a human-friendly explanation of how a particular prediction for a particular instance arose: "Why does the model predict such a high probability for this instance?". Such explanations provide deeper, qualitative insights that can be used to validate model behavior, test hypotheses and potentially generate new scientific knowledge. Therefore, we briefly discuss two popular methods to achieve human-friendly explanations of predictions from black-box models for particular instances: Local Interpretable Model-Agnostic Explanations (LIME) and Shapley Additive Explanations (SHAP).

Recently proposed by Ribeiro et al. (2016) and having gathered impressive popularity in a short

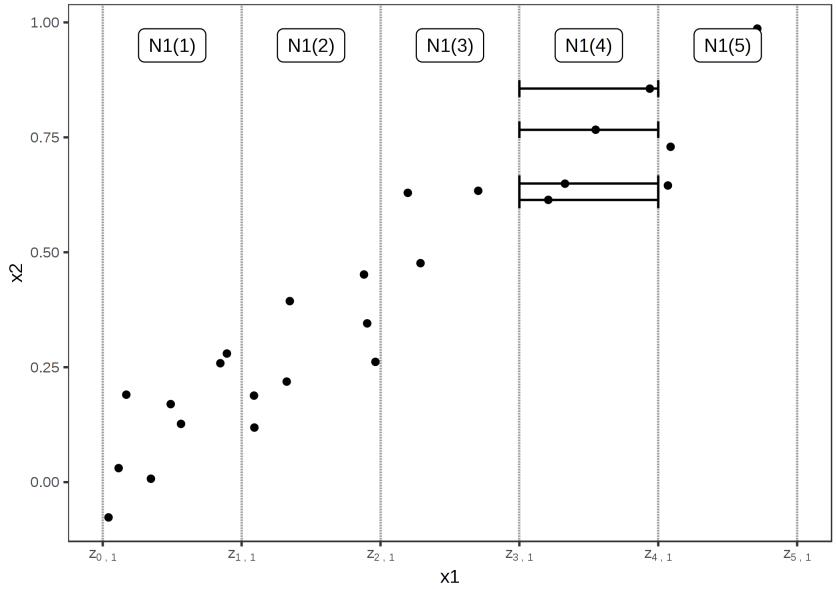

Figure 3.17: Demonstration of the computations that underly accumulated local effects (ALE) plots. This way, we avoid unrealistic instances. Figure adapted from Molnar (2019).

amount of time, LIME is a simple yet effective way to achieve human-friendly explanations for predictions of individual instances. Figure 3.18 shows the intuition behind this method. Briefly, we choose an instance, randomly generate instances near the selected instance, assign weights to the generated instances based on a distance function with respect to the to-be explained instance, and then fit a linear model using the feature values of the generated instances, their weights, and the predictions of the to-be explained model for these instances. We can then interpret the coefficients of the linear model as the 'explanation' for the prediction. Formally, we can describe this process as follows:

$$\text{explanation}(x) = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g), \tag{3.30}$$

in which g is the 'explanation' model, L is the loss function, f the black box model to be explained, π_x the neighbourhood function of the to-be explained instance x, and Ω a regularization function of g. Regularization is included because we desire a small model that provides a *selected*, human-friendly explanation.

As one would expect, strong objections can be made to the appropriateness of explanations generated by LIME. For instance, choosing a linear regression model for g, as in the original work by Ribeiro et al. (2016), is debatable if the to-be-explained model is highly non-linear in the local area. Translating the issue to Figure 3.18, if we would have an instance at the top of the inverted, V-shaped decision boundary, then we can generate two lines in opposite directions which are both equally correct (or incorrect) explanations! The wider issue here is that the choices for g and π_x can exert strong influence on the explanation, and that many different and possibly contradicting explanations can emerge when changing these parameters of LIME. In fact, impactful setup and the possibility of having multiple contradicting explanations is a key issue for many interpretability methods. To get a grasp of how well a particular explanation fits the black box model, the so-called *fidelity* (Molnar, 2019) of an explanation can be inspected. Fidelity can be any metric of choice that expresses how much the predictions of an explanation model match those from the black box model. In the implementation of LIME by Ribeiro et al. (2016), this is the R^2 between the explanation model predictions and the black box model predictions. Lastly, generating random

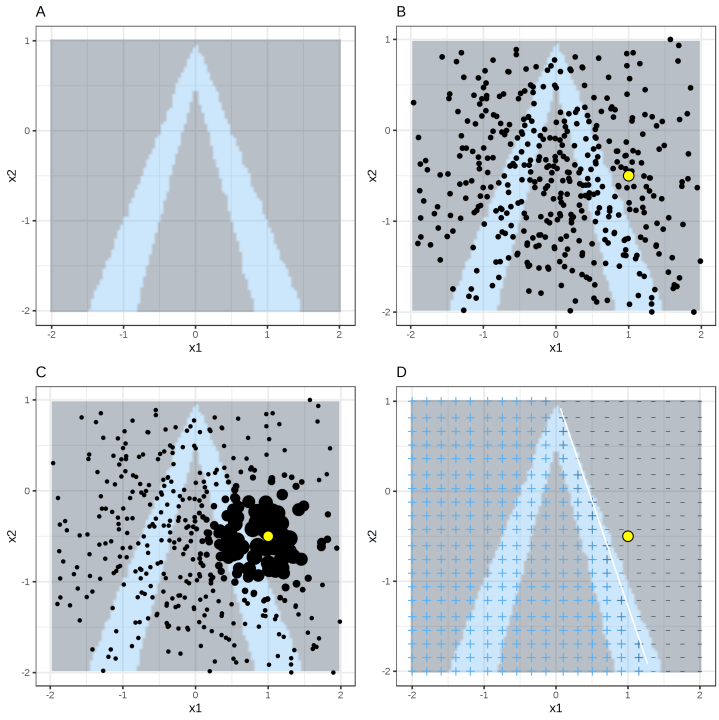

Figure 3.18: Example of how LIME generates explanations for instances. The background shows the decision boundary of the to-be-explained model, the yellow dot shows the instance to be explained, the black dots in B and E show instances generated by LIME for creating the explanation (size indicates the weight of each), and the white line in D demonstrates the 'explanation' model. Figure adapted from Molnar (2019).

instances near the to-be explained instance can also result in unrealistic instances, though this problem is less severe for LIME than for PDPs because the permutation happens on local rather than a global level and because generated instances are weighted by their distance to the to-be explained instance.

Another popular local interpretability method is SHAP, which is based on the application of the classic Shapley values (Shapley, 1953) to machine learning. Again, we visually demonstrate the intuition, which can be seen in Figure 3.19. Like with LIME, the method starts with the selection of an instance for which the prediction should be explained, for example the prediction of the value of an apartment given that it has a garden (indicated by a tree in the figure), an area of 50m^2 and a cat. In our hypothetical example, the value of such as an apartment is €320,000 for this instance. Now, how could we estimate the influence of the cat on the value of the apartment? The "shapley"-way of performing this assessment is to generate all instances with possible combinations excluding the cat (the right of Figure 3.19), make predictions for these instances, compute the differences between the prediction of each of these instances with the original instance, and then finally average these differences. The manufactured instances with the combinations of features are also referred to as *coalitions*, reflecting how Shapley (1953) applied this idea in a game-theoretic setting. In that setting, shapley values answer the question: "How much does each individual player contribute to the final payoff of the game?". For machine learning problems, this changes to: "How does each individual feature value contribute the final prediction of black-box model?".

The implementation of shapley values for machine learning problems raises various challenging issues. To start, the number of coalitions grows exponentially in the number of features, which quickly renders the computations intractable as the number of features increases. Furthermore,

one cannot just "remove" a feature and make a new prediction - this creates a missing value in the instance which requires imputation for most supervised learning models. The question then becomes how to impute - do we use an indicator value, or do we use some kind of sampling? The issues surrounding the implementation of shapley values gives rise to approximation methods for estimating them. A recent set of approximation methods are SHapley Additive exPlanations, also referred to as SHAP (Lundberg & Lee, 2017). Because these methods get quite sophisticated rather quickly, we omit technical or mathematical details regarding these approximation methods and refer the technically inclined reader to the chapter on SHAP in Molnar (2019) and the underlying papers for a more in-depth explanation. The bottom line is that contemporary approximation methods seem to suffer from similar problems as LIME: they are plagued by impactful algorithm parameters that can give rise to many different, possibly contradicting explanations for a single instance, and may involve generating predictions for unrealistic instances in the estimation procedure. Given its impressive popularity in a short amount of time, however, SHAP seems to be a promising local interpretability method. Lastly, Figure 3.20 shows an example of the output produced by SHAP. In contrast to LIME, SHAP typically includes all features used by the model, and does not (or rather, *not really*) use a surrogate model. This can be advantageous in particular contexts, but as of right now, neither SHAP nor LIME are without caveats when applying them in practice.

Figure 3.19: Visualization of the intuition behind Shapley values. The left shows how a cat influences the predictions of two coalitions, while the right shows the possible coalitions for an instance with a garden (indicated by the tree), 50m^2 area, and a location on a 2nd floor. Figure adapted from Molnar (2019).

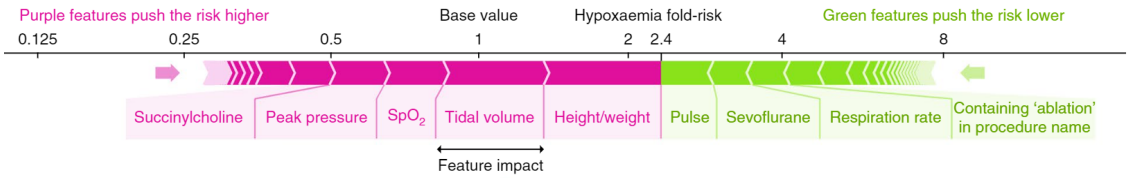

Figure 3.20: Example of a SHAP explanation for predicting hypoxaemia fold-risk, a potentially life-threatening physiological condition that involves a low arterial blood oxygen. Figure adapted from Lundberg et al. (2018).

Again, we have discussed but the tip of the iceberg of local interpretability methods in this section. Other methods include counterfactual examples, prototypes/criticisms, adversarial examples and influential instances (Molnar, 2019). Another interesting alternative are the so-called *anchors* Ribeiro et al. (2018), which is a continuation of LIME that intends to address some of its shortcomings. However, anchors add additional complexity on top of LIME and still suffer from a highly

configurable and impactful setup (Molnar, 2019), which is why we do not consider this method in-depth in this research. Regardless, future research with different local interpretability methods than LIME or SHAP is encouraged.

Chapter 4

Method: Data Processing and Modeling

This chapter delineates the methodology used in this research. Firstly, we discuss our methods to transform the vital sign- and laboratory testing events into a format that enables the prediction of mortality with supervised learning models, and delineate how we will test three alternatives for this data transformation. Secondly, we elaborate on possible methods to split the data in training and testing sets, followed by the formulation of a test to empirically compare two alternatives. Finally, we describe our approach to fit, tune and compare probabilistic fuzzy systems and deep learning models for septic shock mortality prediction.

As the medical database used in this research is somewhat complex, it may not be straightforward to implement the methods in this chapter without further detail on the specifics of the data and the experimental setup. These details will follow later in Chapter 5.

4.1 Data Processing

A crucial element in predictive modeling is the data used for model fitting and assessment. As is becoming increasingly common in the current era of ubiquitous data storage and reuse, the process of transforming stored data into a format suitable for predictive modeling requires careful consideration and experimentation in this research. To this end, we first cover several aspects regarding this issue here, and formulate several tests along the way to empirically establish which approach is the most suitable.

4.1.1 Predicting using Misaligned and Unevenly Sampled Events

In this work, our interest is in the binary prediction of within-72 hour mortality from septic shock, similar as in the work by Fialho et al. (2016). The features for making these predictions are various physiologic variables, i.e. measurements originating from both bedside monitoring (e.g. heart rate) and laboratory testing events (e.g. blood glucose level). Naturally, data from these events become available in the ICU at wildly varying timescales: the heart rate of a patient might be measured every minute during an ICU stay, while a highly specific laboratory test might be performed only once per day. Cismondi et al. (2013) refer to such data as *misaligned, unevenly sampled data*. Figure 4.1 graphically demonstrates the idea. Misaligned, unevenly sampled data can pop up in any prediction scenario involving clinical variables (e.g. predicting diagnoses or length-of-stay), but are also likely occur in many other contexts such as predicting health-related outcomes based on data from wearables, predicting whether a customer will exceed his credit card

limit based on historical purchase transactions, or predicting whether a player will die in a video game based on his previous actions.

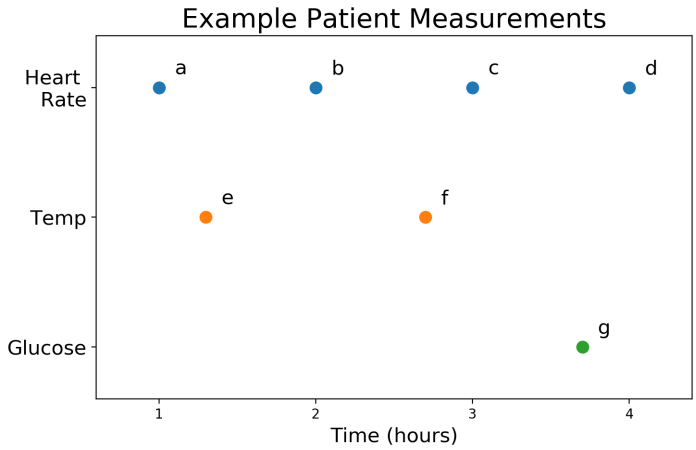

Figure 4.1: Simple example of a misaligned, unevenly sampled time series. When templating with the heart rate as the template, we would predict only at the timestamps of a, b, c and d. In this work, we predict every time new information becomes available, which is at a, e, b, f, c, g and d for this example, respectively.

One way to convert data from misaligned, unevenly sampled events into a format suitable for supervised learning (i.e. with an instance in each row that contains both features and labels) is through *templating* (Cismondi et al., 2013), which is visualized in Figure 4.2. Templating involves choosing the timestamps of the events of one variable as 'the template', and matching the events of all other variables to the timestamps of this variable. Alternatively, the user can specify the timestamps on the template himself, e.g. one timestamp per hour. The matching of events to the template happens by moving the events in the non-template variable(s) to the closest timestamp in the template. Some prior works choose the most frequently measured variable as the template (Fialho et al., 2016), while others simply define a template with timestamps at hourly intervals (Harutyunyan et al., 2019; Wang et al., 2020). Though templating can considerably reduce the number of instances in the data and thereby also reduce the computational demands of fitting supervised learning models, the stance in this work is that we want to predict *every time new information becomes available*, i.e. every time an event of any variable occurs. Despite the fact that this increases the number of instances in the data, it matches clinical intuition (new information means a new mortality risk), results in more frequent predictions that enable timely intervention based on the predictions, and eliminates various complicated questions that arise from templating. Such questions include aspects such as which variable to choose as the template, how to aggregate values that occur more often than the template, how to implement the templating process in code, and whether carrying values back in time is allowed - for example, in Figure 4.2, a measurement at 21 hours is shifted back to 18 hours, at which point it was not yet available in the clinical reality.

A relevant issue that frequently occurs when processing misaligned and unevenly sampled events for supervised learning is severe sparsity of the variables in the resulting time series. To illustrate this issue, consider the situation where the heart rate of a patient is measured every minute for 3 days, while only one blood glucose level test is performed during this time. This would result in 3x24x60+1=4.321 entries in the time series, of which only 1 entry contains a value for the blood glucose level, corresponding to missing values for that variable in 1-1/4.321=99.98% of the instances available for the model. Obviously, this makes it very difficult for a supervised learning model to learn anything regarding the blood glucose variable. This strategy corresponds to the

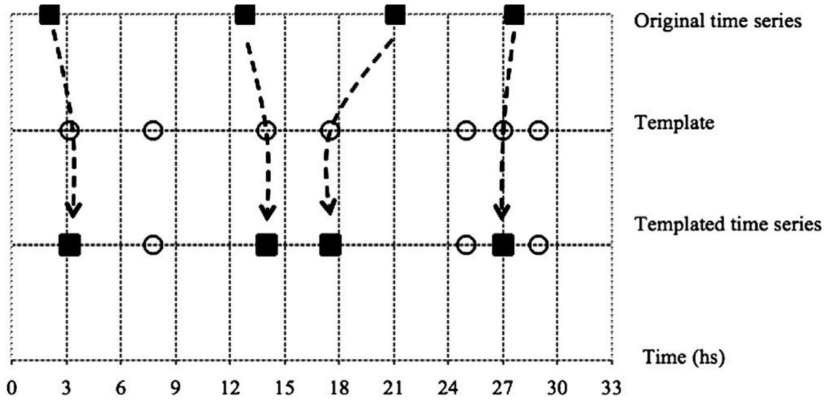

Figure 4.2: The regular templating approach to align the measurements in misaligned and unevenly sampled events for predictive modeling. This involves matching the measurement in the original time series (top) to the closest moment in the template (middle). Figure adapted from Cismondi et al. (2013).

top part of Figure 4.3. A different approach that enables one to fill in a substantial amount of missing values is to carry the value of a measurement forward in time for a while. More concretely, this means that we assume that the measured blood glucose level is constant and valid for some time after the measurement became available. In the middle pane of Figure 4.3, we demonstrate this strategy with a maximum carry forward time (named a "forward-filling window size" in the figure). In this research, we use the median time between two consecutive events of the same variable as the window size: if glucose tests are usually performed once every 6 hours, then we assume the outcome of such a test is valid for approximately 6 hours. Note that the choice for the median is somewhat arbitrary - we could equally well take the 60% quantile, though we do not use the mean as there seem to be outliers in the time between events that pull the mean beyond the average. After the window ends, the value of the variable 'expires', and we assume the variable to be missing again. This is similar to the approach for filling missing values proposed by Cismondi et al. (2013). A third approach is to set a forward-filling window of inifinite size, and simply impute missing values forward in time until the next measurement of the particular value is available or until the end of the time series (illustrated in the bottom pane in Figure 4.3). On the one hand, this may lead to biased data, as it would for instance not be correct to assume that the heart rate of a patient stays the same for hours after the last measurement. On the other hand, this approximation might be better than handling points in time beyond a small forward-filling window as if they were missing values, and imputing e.g. the global mean for these missing values.

In this research, we put all three strategies shown in Figure 4.3 to the test in order to find which strategy results in the best features on our data. Afterwards, we continue the other tests in this research with the best imputation scheme. Now, how can we conveniently and simply assess which strategy leads to the best features for the model? We test the best strategy in two ways. The first is to assess the mutual information (Kraskov et al., 2004) that each variable provides about the target label under each imputation scheme. This tells us whether the features are individually informative without the burden of fitting and tuning any supervised learning models on the data. Furthermore, mutual information informs us about both linear and non-linear variable informativeness of the target label, which is in contrast to e.g. the Pearson correlation coefficient (Guyon & Elisseeff, 2003), which is only able to capture linear relationships. Mutual information, however, does not capture feature interactions as it can only assess the mutual information of one feature simultaneously. To compute mutual information, we use the method proposed by Kraskov et al. (2004) and use the default number of neighbors in the scikit-learn package (Pedregosa et al.,

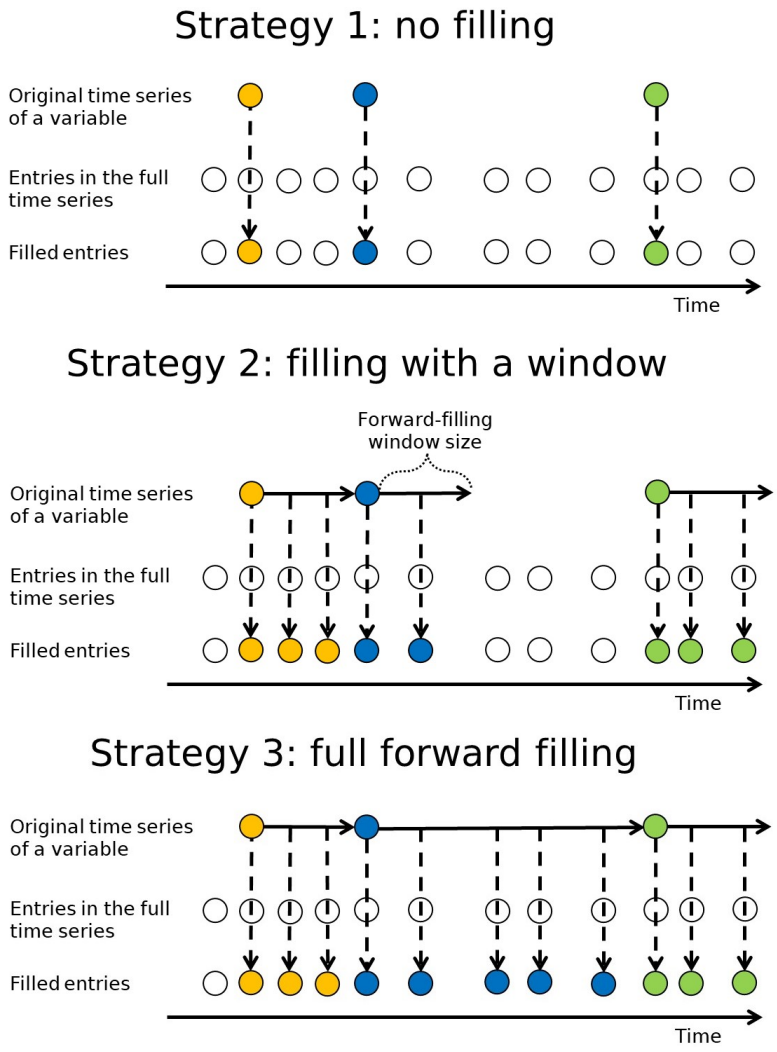

Figure 4.3: Three strategies for filling missing values in a time series originating from misaligned and unevenly sampled events.

2011), i.e. 3 neighbors. Secondly, we split the data resulting from each imputation scheme into a training and testing set and fit a gradient boosting decision tree (Ke et al., 2017) that we assess on the ROC-AUC and AUK metrics on the validation set. This provides some degree of information about the performance of supervised learning models that can capture feature interactions, in contrast to mutual information. For the gradient boosting model, we use a sufficiently high number of trees (1000) and the default parameters in the LightGBM package (Ke et al., 2017). For more details on the implementation and parameters, we refer the reader to Appendix C. After completing the procedure, we deem the imputation scheme that results in the highest model performance to be the best. A first side note is that for filling strategy 2 (filling with a window), we estimate the median time between the events of each variable on the training set rather than on the full dataset to avoid leakage of the training data into the testing data. A second note is that we impute values that are still missing after applying the imputation schemes (i.e. 'truly' missing values) with the global mean of the corresponding feature. This approach avoids having to remove a large amount of the instances in the data, which would leave an unfair comparison between the different filling schemes.

4.1.2 Train- and Test Splitting Strategies

To assess predictive performance, a model should be built on data that is distinct from the data it is assessed or tested on (Provost & Fawcett, 2013). As this research involves temporal data, particular caution is required to avoid leakage between the model building (training) and model assessment (validation or test) data sets. In non-temporal datasets, observations for training and testing sets are typically chosen at random (Provost & Fawcett, 2013). In temporal datasets, however, this is inappropriate because in reality, we can only use data from the past and the present to predict the future - a dynamic that is not preserved when selecting instances fully at random. As Figure 4.4 demonstrates, various ways of splitting the data into a training and testing set are possible on our data. One way is to treat each instance or observation in the data as an independent data point, without taking time or cases into account (observation-based random splitting). Though this is perhaps the most standard and simplest to split the data, and though it nicely preserves the same distribution in the training and testing sets, this is clearly not in line with the clinical situation. In case 1, for instance, instances at the start and end of the sequence were in the training set, and in the testing set we try to predict the target label value in between. Another way of splitting is time-based splitting (top right in Figure 4.4), in which all instances past some point in time are put into the test set. This would be more in line with the clinical situation, as this involves only using information from the past and present to predict the future. Yet another way is to split based on cases (in our context, cases correspond to patients), meaning that each case is either in the training set or in the test set. Again, this way may not preserve the fact that we cannot use information from the feature. Furthermore, the data could be split based both on cases and on time, such that each patient belongs to either the training or the testing set and such that we only use information from the past and present to predict the feature (and thus cut off some instances). Still, even more ways of splitting the data can be though of, such as for instance splitting the sequence of each patient into a training- and testing part.

So, how do earlier related works split the data for predictive modeling, and which strategy would be the most appropriate in this work? In our understanding, the work by Fialho et al. (2016) seems to involve the observation-based random splitting strategy. Perhaps under the arguments that this is the standard way of splitting data for model assessment, that it results in similar training and testing distributions, and that a single instance only contains information at a single point in time anyway (there is no way to directly exploit information from the future at prediction time), this can be deemed an appropriate strategy. Our hypothesis, however, is that this is probably not an appropriate strategy for two reasons: (1) it is not in line with clinical reality as it implies the use of various instances in the future to learn to predict for instances in the past, and (2) this strategy will result in leakage between the training and testing datasets that can be exploited by powerful supervised learning models which are able to capture high levels of complexity in the data. To see why this could be the case, consider case 1 in the top-left pane of Figure 4.4. Here, both the start and end of the case is in the training set, and we are inferring the label for the observations in between (these are in the test set). Now, although a single instance only contains information of an observation at a single point in time, our hypothesis is that powerful supervised learning models will learn to recognize the case from its features, and thereby exploit the knowledge of the end of the case to predict the correct target label. As for the other splitting strategies, those directly based on time are mostly not possible on our data as the start time of each patient stay is randomized for anonymization purposes. This only leaves the case-based strategy from Figure 4.4, which may still involve usage of information from the future, but cannot result in the previously described leakage due to patient recognition. In fact, this is also the strategy used in the related works by Harutyunyan et al. (2019), Purushotham et al. (2018) and Desautels et al. (2016). Now, to test whether the envisioned leakage occurs in observation-based splitting, and to test whether that leakage is absent in case-based splitting, we test the same model twice with both splitting strategies. Because we require a model with sufficient representational power to recognize patients from many features, we again use a gradient boosting decision tree (Ke et al., 2017) with a sufficiently large number of trees (i.e. 1000 trees). We also assess the ROC-AUC and

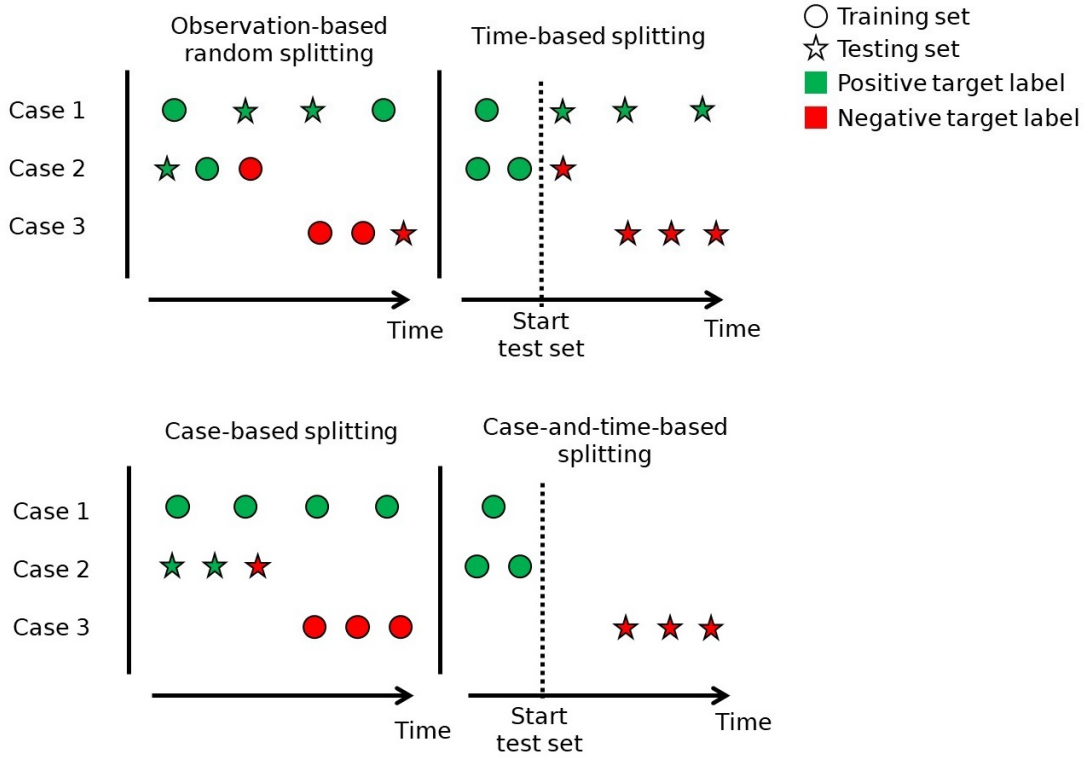

Figure 4.4: Four possibilities for splitting the data into a training and testing set. In our context, an observation is a point in time during the hospital admission at which we make a mortality prediction. A case corresponds to a patient. For clarity of presentation, a negative target label value means death with 72 hours (intuitively a negative outcome), while a positive target label value means survival in this figure.

AUK metrics of both models on the testing set again. If the observation-based splitting method results in unrealistically high performance, then this provides evidence that the envisioned form of leakage due to patient recognition may occur, and that the rest of the tests in this research should use the case-based splitting strategy.

4.2 Model Fitting

Evidently, only two types of models are tested in this research, namely PFS and deep learning. Though it could be interesting for medical professionals to compare these models to the severity scoring systems mentioned in Section 2.2, this is refrained from in this research for several reasons. The first is that severity scoring systems are designed for a single prediction of patient mortality, usually within 24 hours after ICU admission. Using severity scoring systems in our near real-time mortality prediction scenario (with multiple predictions per patient) would thus be inappropriate use. Severity scoring systems are also intended for and calibrated on diverse sets of patients with varying conditions, while our interest is only in patients with septic shock. Restricting the application of severity scoring systems to this subgroup of patients results in yet another violated assumption, rendering the usage of these systems even more inappropriate. Furthermore, previous works have already demonstrated that supervised learning models, and deep learning in particular, generally outperform SAPS-II and OASIS in accurately predicting mortality upon or near ICU admission (Harutyunyan et al., 2019; Purushotham et al., 2018). Finally, severity scoring systems require many different variables (as shown in Appendix A.1), whose data extraction would

complicate the experimental setup considerably.

4.2.1 Probabilistic Fuzzy Systems

Firstly, we need to determine how many features and how many rules to use in the PFS. In our research, it is a requirement that the PFS is interpretable such that its rules can be validated by medical professionals. More concretely, the PFS should be simple enough as to be kept in short-term memory for an individual that received only minimal training in using the PFS. Following the conventional wisdom that an average human cannot take more than approximately 4 new elements in memory at a given time (Cowan, 2001), it is our preference to have this number of rules and features. However, various circumstances are imaginable where it would be wise to deviate somewhat from this number: if we can achieve the same performance with e.g. 3 features and 2 rules as with 4 features and 4 rules, then the former is clearly preferable. The opposite is true as well: if a substantial performance gain can be achieved by using 6 features and 5 rules, then we are willing to accept the added complexity and decreased interpretability. Thus, the design space is set at a margin around the optimum of four. We (somewhat arbitrarily) test 1 to 7 features and 2 to 7 rules. As the number of available features strongly exceeds 7 in this research, all acceptable configurations w.r.t. the number of features must involve feature selection.

Many different feature selection methods are possible (Guyon & Elisseeff, 2003), thus we show an overview of the advantages and disadvantages of the considered methods in Table 4.1. Clearly, different methods have different advantages and disadvantages, rendering no choice clearly superior. While variable ranking methods such as the Pearson correlation coefficient and mutual information have the advantages of being relatively simple to implement and computationally cheap, they do not take into account how the actual prediction model performs on the features and do not take into account feature interactions. A random or grid search over the features and model hyper-parameters does directly incorporate model performance on the selected features, but may leave noise features in the final model. The method of choice in Fialho et al. (2016) is forward feature selection, which is a hill climbing method that conveniently avoids having redundant and/or noise features in the model. However, it raises several challenging issues in our research: as our dataset is quite large, forward feature selection is rather computationally expensive, especially because it is desired to tune the number of rules in the PFS jointly with the feature selection. Though backward elimination is a similar method which is computationally cheaper than forward feature selection in our situation, it raises the issue of how to determine feature importances for the PFS, of which no standard, quantitative definition is available for probabilistic fuzzy systems to the best of our knowledge. The full overview of considerations regarding different feature selection methods is provided in Table 4.1.

In this research, we use a combination of a variable ranking method and a grid search. Firstly, we rank the features based on mutual information (Guyon & Elisseeff, 2003). We use mutual information rather than Pearson correlation because this criterion is able to take into account non-linear relationships. Again, we compute mutual information using the method proposed by Kraskov et al. (2004), implemented in the scikit-learn package (Pedregosa et al., 2011). With the feature ranking, we perform a grid search over the number of features and the number of rules used in the PFS. The features are picked from top to bottom in the ranking based on mutual information. More detail on the tuning process of the PFS will follow in Section 5.5 of the experimental setup.

Another set of decisions regarding the PFS is the shape of the fuzzy membership functions, and how is dealt with rules that are redundant or irrelevant. Regarding the first issue, we take the same approach as Fialho et al. (2016): each membership function is a product of univariate Gaussians, i.e. one Gaussian for each rule and each feature. Regarding the second issue, the redundancy of rules and/or features is inspected through both inquiring the model parameters (i.e. centers, widths and conditional probabilities) and applying the Jaccard similarity from Equation 3.28. In this work, we only investigate the degree of redundancy and irrelevance in the model, and do not

Table 4.1: Overview of considerations regarding possible feature selection methods. Note that our data involves 26 features available for the model and a maximum of 7 selected features.

Feature Selection Method	**Advantages**	**Disadvantages**
Variable ranking	Simple to implement	Can result in correlated features in the model (redundant features)
	Computationally cheap	Ignores possibly valuable feature interactions
		Does not directly take into account the performance of the actual prediction model on the features, i.e. indirect assessment of feature impact on performance
Random/grid search	Wide, diverse exploration of solution space	Possibly miss out on valuable features in the final, selected model
	Simple to implement	Possibly include noise features or redundant features in the final, selected model
	Easily tune other hyperparameters jointly with the feature selection	No hill climbing method, thus unlikely to end exactly at a (global or local) optimum
	Computational demands can be set at a desired level (i.e. setting the number of tested configurations)	
Forward selection	Hill climbing method, converges to an optimum (either local or global)	Computationally expensive: requires $\sum_{i=20}^{26} i = 161$ model fits to select 7 features (without tuning other hyperparameters)
	Avoids redundant and/or noise features in the final model	Greedy approach that may get stuck in a local optimum
		Too expensive to tune the number of rules jointly with the feature selection
Backward elimination	Hill climbing method, converges to an optimum (either local or global)	Requires model feature importance of which no general, quantitative definition is available for the PFS
	Cheaper than forward feature selection: would involve 26 model fits without tuning the number of rules	Greedy approach that may get stuck in a local optimum
	Avoids redundant and/or noise features in the final model	

iteratively correct high redundancy and irrelevance. Therefore, usage of the method proposed by Setnes et al. (1998) to address redundancy and irrelevance would be an interesting avenue for future research.

Next, various decisions are to be made regarding the way in which the model parameters are fit to the data. Fialho et al. (2016) initialize the centers and widths of the rules using fuzzy c-means clustering, likely at the argument that this clustering method results in a fuzzy partitioning of the data that matches the PFS which is fitted using the cluster centers afterwards. We caution that the objective function of fuzzy c-means can lead to cluster centers that are extremely close to one another, resulting subsequently into: small membership function widths arising from Equation 3.7, large values in the exponent of Equation 3.6, numerical underflow of the membership function, an undefined gradient in the loss function (Equation 3.9), a forced stop of the gradient descent algorithm and early termination of model training before convergence. As rectifying this issue with a manual override of the (nowadays) highly automated gradient descent procedure is technically challenging to implement, we test both the use of fuzzy c-means clustering as well as k-means clustering to avoid having to interfere with the gradient descent. We deem it likely that the objective function of the latter pushes cluster centers away from one another more strongly (because it is a crisp clustering method), thereby reducing the odds of initializing the PFS with overlapping cluster centers (and an error in the gradient descent optimization as a result). Furthermore, we make initial estimates for the rule consequents in the same manner as Fialho et al. (2016), namely by using conditional probability estimation. Gradient descent is used to optimize the centers, widths and rule consequents after their initialization. We use the Adam gradient descent algorithm that

was discussed earlier in Section 3.3.2. Recall that this method is one of the most recent gradient descent algorithms, and that it incorporates elements from both gradient descent algorithms with adaptive learning rates and with momentum. Because of these mechanisms, we do not see a need to manually tune the learning rate and leave it at its default - we deem it rather unlikely that the model would not converge under the default setting. The Adam optimizer was also empirically found to converge somewhat faster than the older gradient descent algorithms that served as its inspiration (Kingma & Ba, 2014), and thus should conveniently reduce the computational demands of running the optimization procedure. Regardless, we remind the reader that no universally superior gradient descent algorithm exists (as mentioned earlier in section 3.3.2), thus future researchers that aim for a more exhaustive model tuning procedure should probably experiment with different gradient descent algorithms with varying hyperparameters as well.

4.2.2 Deep Learning

Like a PFS, deep learning involves various design decisions. Regarding the features for deep learning, we take two positions. The first is that deep learning should be able to extract meaningful features automatically, and that it thereby should be fit on all available features. The second is that it is our research objective to compare PFS and deep learning, thus it could be considered unfair to provide deep learning with more information than the PFS. To take both perspectives into account, deep learning models are considered on both the final selection of features for the PFS and on all available features. We assess both a densely connected neural network and a recurrent neural network on both feature sets, resulting in four combinations. For the recurrent neural network, we use the GRU units described in Section 3.3.4 as these are less demanding on computational resources than LSTM units while maintaining the ability to model both short-term and long-term temporal patterns. As discussed in section 3.3, tuning the hyperparameters of a deep learning model is challenging because they are numerous and because fitting a deep learning model is computationally expensive - particularly on our sizable dataset. We perform the hyperparameter tuning through a random search. As mentioned in the influential work by Goodfellow et al. (2016), a random search is preferable over a grid search when tuning the numerous hyperparameters in a deep learning network as it results in a quicker, more diverse exploration of the solution space. Because the computational resources available in this research are limited, only the main hyperparameters of the neural networks are tuned: the number of layers, the number of hidden units per layer, L1-/L2-regularization and the dropout rates between the layers. Additional detail regarding the hyperparameter tuning can be found in Section 5.5.

In this research, an extensive tuning of deep learning hyperparameters is refrained from to restrict the complexity of the experimental setup and the computational demands on our computing resources. Firstly, we refrain from experimenting with different types of gradient descent algorithms because they generally converge to optima with similar performance (Goodfellow et al., 2016), meaning that the impact of the choice of gradient descent algorithm is likely to have but a limited effect on the final model performance. As mentioned earlier, however, no gradient descent algorithm is universally superior to every other over every task, thus for future research with the orientation to squeeze out as much model performance as possible, it would be sensible to experiment with different types of gradient descent algorithms. We, however, do not. Like with the PFS, we stick to the Adam gradient descent algorithm as proposed by Kingma & Ba (2014) and leave the learning rate at its default, which was also the chosen approach in Harutyunyan et al. (2019). Various possibilities regarding regularization are left unexplored as well, including but not limited to multitask learning, data augmentation, adversarial training and parameter sharing. The first three methods are relatively labor-intensive ways of model regularization as they involve considerable additional manual data processing. Creating correct and meaningful data augmentation, for instance, is rather straightforward for images (one can just mirror, flip or resize the image) but complicated in our scenario (what clinical variables can be safely and correctly altered during training, and in which situations?). Parameter sharing is also challenging to deploy effectively in our scenario because of the sparse and irregular multivariate time series with clinical data. This is

unlike parameter sharing in image recognition problems, where one can reasonably assume various properties about the image data and thereby restrict the number of parameters in the network through for instance a convolutional neural network architecture. Lastly, we do not perform a so-called "broad to narrow" hyperparameter search (repeating the searching process in promising areas originating from earlier searches) to reduce computational demands and implementation complexity.

4.2.3 Data Splitting, Patient-Instance Hierarchy and Class Imbalance

Once the data is split in a training and testing set (using the best splitting strategy from earlier), the training set is used for fitting the models, feature selection and hyperparameter tuning, while the test set is only touched once, namely for the final model assessment. Note that this implies that the 'full' training dataset is split in a 'sub' training set (for fitting the models) and a validation set (for early stopping and assessing model performance during feature selection and hyperparameter tuning). This practice of leaving a test set completely untouched until the final model assessment avoids optimistic performance claims due to test set overfitting and is in parallel to model assessment in data science competitions such as Kaggle (Kaggle, 2020) and computer vision benchmarks such as KITTI (Alhaija et al., 2018; Fritsch et al., 2013; Geiger et al., 2012). This method of model assessment seems to have become the de facto standard within the wider data science community, which is another reason as to why we follow this practice. Paralleling Harutyunyan et al. (2019), we take 70% of our data as the training set, 15% as the validation set and 15% as the test set. We also take care to split the data in a case-based (in our context patient-based), fashion.

In this research, a single instance corresponds to a single point in time for a single patient. This means that a hierarchy is present in our data: an instance belongs to one patient, and one patient has multiple instances. The number of instances per patient is highly variable (ranging from 3 to approximately 43,000), thus ignoring the patient-instance hierarchy in the data is bound to result in a biased picture of algorithm performance on the evaluation metrics: the supervised learning algorithms would likely only learn to predict correctly for the frequently occurring patients, resulting in an unwanted model that is only effective for the small subset of patients that have many instances in the data. To solve this issue while preserving both the data of all patients and the instances at every event, each instance is weighted at training time such that each patient is of equal importance in the loss function of the algorithm. This implies that each instance of a patient with 3 instances will have a weight in the loss function of 1/3, each instance of a patient with 100 instances will have a weight of 1/100, etcetera. At validation- and testing time, such weights are unneeded as our metrics of interest are the ROC-AUC and the AUK, on which the sample weights hardly exert influence because they rely on a ranking of model metrics across all possible thresholds.

Not to be confused with the instance weights described above are class weights, which serve the purpose of addressing class imbalance by weighing the loss produced by each instance with respect to the relative occurrence of its class. For instance, in a binary classification problem where only 1% of the instances in the data belong to the positive class, most supervised learning algorithms would fail to learn classifying the positive instances. A solution is then to penalize mistakes in classifying a positive instance 100 times more than mistakes in a negative instance, such that the algorithm pays equal attention to both classes regardless of the strong minority of the positive class. As our prediction problem involves imbalanced binary classification as well (though less severe imbalance than in the example), we also apply such class weights at training time in order to avoid the undesired situation where the algorithms would pay excessive attention to the majority class (i.e. only predicting survival). Note that such class weights are *not* desired at validation and testing time as we have chosen the ROC-AUC and AUK as our evaluation metrics (we elaborate on the choice for these metrics in Section 4.3). The former has the desirable behavior of being insensitive to class imbalance, while the latter is (desirably) sensitive to class imbalance by default. Correcting for class imbalance at validation and testing time would thus make the AUK rather

redundant as this metric would then not be conditioned on class imbalance anymore (imbalance would be equalized by the class weights). In conclusion, we apply both instance and sample weights at training time, but no weights at validation and testing time.

4.3 Model Assessment

4.3.1 Quantitative Assessment

Our research method involves the prediction of probabilities for whether a patient will die or survive. This is a binary classification problem for which multiple metrics are possible. Various possible metrics were discussed in-depth in Section 3.4. Firstly, our situation does not enable us to attach numeric costs to correct and wrong classifications, thus we are unfortunately unable to tune the prediction thresholds based on the application of the model. Hence, model performance is assessed in this research using the ROC-AUC and the AUK. Both metrics provide distinct yet useful information as the ROC-AUC is not affected by the class distribution, while the AUK is sensitive to this aspect of the data. Furthermore, we choose the AUK over the PR-AUC because the Kappa curve shows performance in a very similar way as the PR-AUC (namely, how the model performs across multiple thresholds dependent on the underlying class distribution), but has additional desirable properties. This is somewhat at odds with earlier related works: Fialho et al. (2016) only asses model performance using the ROC-AUC, while Purushotham et al. (2018) and Harutyunyan et al. (2019) use the ROC-AUC and the PR-AUC.

Rather than assessing the final model performance on the selected metrics through cross-validation (as in Fialho et al. (2016)), we train our best models from the tuning procedures on the training set (while using the validation set for early stopping) and then sample 1000 observations from the test set 500 times with replacement to obtain 500 different ROC-AUC and AUK estimates. This approach resembles the work on MIMIC prediction model benchmarking by Harutyunyan et al. (2019) and substantially reduces computational demands as our models do not need to be re-trained many times as would have been the case with cross-validation. This bootstrapping approach also enables us to assess the variability in model performance and to use the nonparametric Mann-Whitney U test (Sheskin, 2007) to determine whether the models have significantly different levels of performance. Finally, we show the ROC curve and Kappa curve over all observations in the test set in order to investigate how the model performs across different prediction thresholds.

4.3.2 Model Inquiry

We complement our quantitative assessment of the models with a more qualitative assessment in which we inquire aspects regarding their internal representations and behavior. This qualitative inspection adds an extra dimension to the comparison of the PFS and deep learning models. More concretely, we are not only interested in which model performs best on the previously described metrics, but also into what particular patterns in the data cause a model to predict a high or low mortality risk. Particularly for deep learning models, which possess strong representational power and are prone to overfitting as a consequence, it is relevant to assess whether model behavior does not exhibit odd artifacts and whether it is in line with general clinical intuitions. For instance, if the mortality probability predictions of a deep learning model are rather unstable (i.e. jump up and down) when the value of the oxygen saturation feature (SpO2) decreases, this would be evidence that the model is acting on spurious correlations - it is obvious from a clinical perspective that decreasing peripheral oxygen saturation should go accompanied by an increasing mortality risk. Note that a full, in-depth interpretation of the model behavior with respect to physiology and the chemistry of the human body is beyond the scope of this work - this work is limited to basic clinical inuitions. To put our objectives of the model inquiry in brief, they are twofold: (1) inspect if and to what degree suspicious behavior occurs in the models which is not in line with basic clinical intuitions, and (2) mutually compare the models regarding their behavior in response

to patterns in the data.

Firstly, we perform a PFS-specific inquiry. We start by inspecting the learned cluster centres, widths and conditional probabilities. Afterwards, we create several plots to visualize the learned rules, similar as in Fialho et al. (2016). Inspecting the above provides insights regarding whether the PFS matches clinical intuitions about mortality risk and to which extent particular rules can be considered redundant or irrelevant. For a more formal assessment of redundancy and irrelevance, we resort to the Jaccard similarity coefficient (Equation 3.28).

As our main interest is in an in-depth comparison of PFS and deep learning, we also perform paired inquiries into both models using various 'explainable machine learning' methods from Section 3.5.2. Paralleling Section 3.5.2, we inquire models on both a *global* and a *local* level. The main considerations regarding the priorly discussed methods for both are shown in Table 4.2 and 4.3, respectively. Regarding the global level, we follow the advice by Molnar (2019) and use the ALE instead of the PDP to avoid the issue of having unrealistic instances in the underlying calculation of global feature effects. We use the implementation of ALEs from the Alibi package (Klaise et al., 2020). As earlier described in Table 4.2, implementing ALE plots on RNNs presents considerable challenges, and Alibi does not provide any default functionality to solve these issues. Thereby, we refrain from the challenges surrounding the application of ALE plots to RNNs in this research - we omit ALE plots for our GRU networks. We thereby leave an investigation of how to conveniently and meaningfully implement ALE plots for RNNs up to future research. To validate whether the choice for the bin size in ALE indeed exerts limited influence on the curve, we select one ALE plot and vary the bin size from small to large to assess whether and how the ALE plot changes. In addition to ALE plots, we compute permutation feature importance for all features of all models, as this provides additional insight into global model behavior with respect to ALE plots (refer to Section 3.5.2 as to why). In contrast to ALE plots, feature permutation importance can easily be computed for RNNs, such that we conveniently get at least *some* insight into global RNN model behavior. Note that we restrict our assessment of feature permutation importance to individual features only, as testing the effects of permuting multiple features simultaneously presents the challenge of which combinations to test, particularly because testing all combinations is very computationally intensive. Unfortunately, feature interaction effects regarding permutation feature importance are thus left unexplored in this research.

Table 4.2: Overview of advantages and disadvantages regarding the considered methods to interpret the model on a global level, answering the question: "How does a feature influence the predictions globally?".

Method	**Advantages**	**Disadvantages**
Partial dependence plots	Ease of implementation	Highly likely to involve extremely implausible instances in the dependence calculation: biased picture of feature influence
	Simple to interpret	Unclear how to use with RNNs: should an estimation for a single instance only include the same timestep, similar timesteps or all possible timesteps? How to deal with past values of a feature?
Accumulated local effects plots	Does not or hardly involve predictions for implausible instances	Curve is somewhat sensitive to the interval parameter
	Scales well to big datasets	Slightly harder to interpret than other methods
	Software package available to abstract away from complex underlying calculations required	Unclear how to use with RNNs: should an interval only include the same timestep, similar timesteps or all possible timesteps? How to deal with past values of a feature?
Permutation feature importance	Ease of implementation	Does not show relationship between feature values and prediction values (e.g. "How exactly does a low value for this feature relate to the predicted value?")
	Simple to interpret	Can be variant due to random permuting, but can be tackled by averaging over multiple iterations
	Scales well to big datasets	
	Clear how to use with RNNs: we can simply permute a feature over all timesteps	

For an in-depth comparison of the behavior of the PFS and deep learning models on a local level, we perform a case study in this work. We select a patient, and explore the predictions of each model for this patients over time. Then, we select a handful of interesting instances for both patients and use LIME to obtain explanations for the predictions. The full considerations regarding the choice for LIME or SHAP to this end are shown in Table 4.3. Our take is that the approximation methods required in SHAP make it difficult to use it in a reliable, reproducible manner for this research. Furthermore, the explanations generated by SHAP have a somewhat counterintuitive interpretation due to the marginalization across the coalitions that underlies the computation of SHAP values. Thus, our method of choice for generating explanations is LIME, though SHAP is a worthy competitor and is encouraged for future research. More in-depth, we use LIME with its default parameters as implemented in Ribeiro et al. (2016). The emphasis is on generating *some* explanations for the predictions, as fully explaining black box models using multiple possible explanations with different configurations of LIME is a daunting task. To get a sense of *how well* a particular explanation fits the model, we inspect the fidelity (Molnar, 2019) of each LIME explanation. In the LIME package (Ribeiro et al., 2016), fidelity is operationalized as the R^2 of the LIME model with respect to the black box model on the generated local instances used to create the explanation. This fidelity metric is conveniently implemented in the supplemental LIME package from Ribeiro et al. (2016). Finally, we briefly investigate the effect of changing the kernel weight in a LIME explanation to get a grasp of how impactful this aspect of the LIME setup is on the predictions. This is performed through inspecting a series of explanations for a single instance of a single model while varying the kernel width.

Table 4.3: Overview of advantages and disadvantages regarding the considered methods to interpret model predictions for individual instances, answering the question: "What features contribute to the prediction for a specific instance and how?".

Method	**Advantages**	**Disadvantages**
Local interpretable model-agnostic explanations (LIME)	Software packages available - no need to implement calculations manually	Many different and contradicting explanations are possible and likely for one instance: explanations are very sensitive to the neighbourhood function, choice for the surrogate model and other LIME hyperparameters.
	Default implementation provides "selective" explanations that only include the most influential features	Unclear how to use with RNNs: should the surrogate model only include feature values at the current timestep, or also across past timesteps? What about perturbations?
	The fidelity metric makes explicit how well explanations match the to-be-explained model in the local area	Random sampling of data points in neighbourhood can lead to unrealistic instances.
Shapley additive explanations (SHAP)	Software packages available - no need to implement calculations manually	Highly likely to involve extremely implausible instances in the coalitions: biased picture of feature influence
	Takes into account all features simultaneously in explanations	Exact calculation of shapley values is intractable - must use approximation methods, leading to variable feature effect estimates
		Approximation methods (KernelSHAP and Deep SHAP) are complex to implement and interpret, while still computationally expensive
		Easily misinterpreted - somewhat unintuitive interpretation (feature importance is the average marginal contribution among all sampled coalitions)
		Unclear how to use with RNNs: should a coalition only include feature values at the same timestep, similar timesteps or all possible timesteps? How to deal with past values of a feature? How to deal with the explosion of possible coalitions when considering feature values across multiple timesteps?

Chapter 5

Experimental Setup

This chapter contains a detailed description of the experimental setup in this research, including a description of the database, which information is required from this database, the required preprocessing and finally the details of the model- and hyperparameter tuning.

5.1 The Database: MIMIC-III

To put probabilistic fuzzy systems and deep learning to the test, we use version 1.4 of the freely accessible Medical Information Mart for Intensive Care (MIMIC-III) (Johnson et al., 2016). This database and its previous releases have found widespread adoption in observational, medical studies over the years. The data in MIMIC-III was recorded at the Beth Israel Deaconess Medical Center in Boston between 2008 and 2014, but was carefully anonymized as to allow for usage in research without ad-hoc elaborate consent procedures for each researcher wishing to use the data. This anonymization process roughly consists of two sets of operations: (1) removing patient information such as names and addresses, and (2) obscuring the true dates of the patient-related events by binning the birth dates of patients with an age over 89 and shifting the start date of the dataset to the future. Although it is made impossible to identify the true dates of the events in the data, the order of and time between events for each patient individually are kept intact as to allow for analyses that involve temporal dynamics. The relative order of events occurring between different patients, however, is not preserved, meaning that if two patients receive a treatment at the same time in the data, they may actually still be treated at different points in time in reality. This means that most time-based splitting strategies are not possible, as was already mentioned in Section 4.1.2.

The MIMIC-III database consists of 26 tables containing information such as patient demographics, diagnoses, medical interventions, transfers, lab tests and bedside measurements. An entity-relationship diagram of the tables in MIMIC-III and their mutual links can be found in Appendix C.1. We describe how to obtain access to MIMIC-III and how to build the database for local use in Appendix C.2.

5.2 Variables and Target Label

In this research, we predict mortality based on data from electronic health records. To make more concrete what data that we use and what form that our mortality predictions take, we provide more detail on the chosen variables (features) and target label here.

The death of a patient is the result of a complex interaction of numerous factors. To name but a few factors, patient mortality can depend on patient physiology, demographics, medication, nutrition

and executed surgical procedures. To avoid an excessive amount of features, we follow Fialho et al. (2016) and restrict this research to physiological variables from two categories: bedside monitoring (also referred to as vital signs in this work) and laboratory testing. Table 5.1 shows the variables considered in this work. Obviously, more accurate predictions of patient mortality are likely to be possible when more variables are considered. Although we refrain from this challenge, we consider future research that considers more variables invaluable.

Regarding the target label, we also follow Fialho et al. (2016) and predict mortality within 72 hours as a binary variable. According to Fialho et al. (2016), 72 hours strikes a balance between being early enough (such that predictions are actionable) and not being too early (which could result in too much uncertainty about mortality and wildly inaccurate predictions as a consequence). Again, other choices are possible here, and we encourage future research into the usage of different time frames.

Table 5.1: Overview of the variables considered in this research.

Category	**Variable**
Bedside monitoring	CVP
	Diastolic Blood Pressure
	FiO2
	Heart Rate
	SpO2
	Systolic Blood Pressure
	Temperature
Laboratory Testing	Arterial Base Excess
	Arterial PCO2
	Arterial pH
	Arterial po2
	AT3
	Bicarbonate
	Bilirubin
	Calcium
	Creatinine
	CRP
	Glucose
	GOT(ASAT)
	GPT(ALAT)
	Hematocrit
	Platelets
	Potassium
	PTT
	Sodium
	Urea
	White Blood Cells

5.3 Preprocessing for Septic Shock Mortality Prediction

As elaborated upon earlier (mainly in Section 4.1), the objective in this research is to predict septic shock mortality in a close-to-real-time fashion that involves making a prediction every time new information becomes available, i.e. at each event of each patient. This implies that we need a dataset with contents in the format of Table 5.2, with a single instance corresponding to a single point in time of a single patient. Although most of the variables in Fialho et al. (2016) are present in the MIMIC-III database, they are spread over various tables and not in the required format,

raising the need for a data conversion procedure. As this procedure gets considerably complicated due to the large size of the tables involved (some involve hundreds of millions of rows) and the fact that many variables have multiple different designations in the data, it would be desirable to have a standardized, tried-and-tested tool or library to perform this extraction.

Table 5.2: The desired format of a dataset for fitting PFS and deep learning models. A patient appears in multiple instances, though each instance is on a different moment in time. Features include bedside monitoring and laboratory testing, while the target label is mortality within 72 hours.

		Bedside Monitoring			**Laboratory Testing**			
Date / Time	**Patient ID**	**CVP**	**Diastolic Blood Pressure**	**...**	**Arterial Base Excess**	**Arterial PCO2**	**...**	**Death within 72 hours**
10-02-2020 16:02	1	92	120	...	NA	1.1	...	0
10-02-2020 16:42	1	100	110	...	138	1.1	...	1
...	...	...	...	...	...	...	...	...

Two earlier works that provide code to perform an extraction on MIMIC-III similar to the one required in this research are those by Purushotham et al. (2018) and Harutyunyan et al. (2019). As Appendix B.2 shows, the overlap in features of these extracted datasets with Fialho et al. (2016), and thus with our features, is unfortunately fairly minimal: the work by Purushotham et al. (2018) only contains 11 of the 27 variables and the work by Harutyunyan et al. (2019) only 8. Hence, these extracted datasets miss a significant amount of invaluable information for predicting septic shock mortality, particularly information from laboratory testing. In addition, the data extractions in both works do not match the practice of generating an instance at every event, which is desired in our research as motivated earlier in Chapter 4. In the work by Purushotham et al. (2018), mortality is only predicted two times for a single patient, namely at 24 and 48 hours after admission. In the work by Harutyunyan et al. (2019), one of their extraction procedures results in a dataset that involves predicting mortality only once per patient (namely 48 hours after admission), and another procedure results in a dataset that involves predicting within-24-hour mortality at every hour during the ICU stay (see Appendix B.1 for a more elaborate description of the relevant extracted dataset in Harutyunyan et al. (2019)). The last benchmark dataset is somewhat close to our desired format, but due to the minimal overlap in features and the template with hourly predictions, this extracted dataset is still inadequate for our research purposes. Though understandable given the research purpose of Purushotham et al. (2018) and Harutyunyan et al. (2019) to create and test reusable, standardized benchmark datasets, their code is clearly designed for performing exactly their extraction rather than a custom extraction: customizing the variables and the prediction moments is a tedious job as the code is hardly modular, parameterized and reusable regarding this aspect. For these reasons, we do not use their works to realize our desired data extraction from MIMIC-III.

Over the course of this research project (April 2020), a work by Wang et al. (2020) was published with as its main objective to provide a reusable and customizable data extraction pipeline for MIMIC-III. Though this work is an interesting initiative that could have been used to realize parts of our extraction procedure if it had been published earlier, it unfortunately does not provide functionality to extract instances at every event with our filling strategies and thereby cannot eliminate most of the preprocessing required in this work. Though a promising initial attempt, the work by Wang et al. (2020) is clearly only in its earliest phase - it has no user or reference guide and hardly contains any comments in its source code. Furthermore, performing an extraction will take approximately 5-10 hours while requiring a machine with at least 50GB of RAM according to the Github repository of the work. For all of these reasons, and because the other two works by Purushotham et al. (2018) and Harutyunyan et al. (2019) are also inadequate for our research purposes, we design a custom data extraction procedure that we will describe in the remainder of this chapter.

5.3.1 Relevant Tables and Fields in MIMIC-III

Now, knowing both the contents of MIMIC-III and the required output data format for fitting models, we can start to identify which parts of the database are relevant. Figure 5.1 shows the tables and fields from the MIMIC-III database that are relevant with respect to our features and target label. A patient is identified with a `SUBJECT_ID` and a single hospital admission with a `HADM_ID`. The main table with hospital admissions, `ADMISSIONS`, contains a `DEATHTIME` field which records whether an admission ended with patient death, and if so, at what time. `DIAGNOSES_ICD` contains all diagnoses for patients - this table is used to find patients with septic shock, a process that we describe in more detail in section 5.3.2. The required information about bedside monitoring and laboratory testing is recorded in the `CHARTEVENTS` and `LABEVENTS` tables, respectively. As the name of the tables suggest, these tables store this information as *events*, implying that a single row in these tables denotes a single time that a particular action was performed on a patient. Each row thus refers to a particular patient (`SUBJECT_ID`), hospital admission (`SUBJECT_ID`) and event type (`ITEMID`), while also containing the time that the event occurred (`CHARTTIME`), the measured value if applicable (`VALUENUM`) and the corresponding unit of measurement (`VALUEUOM`). Now, `ITEMID` is a numeric indicator for the type of events, but to retrieve the type it stands for in text format we need to consult the `LABEL` columns of the item definitions tables. These definitions tables are (`D_ITEMS` and `D_LABITEMS`) for bedside monitoring events and laboratory testing events, respectively. A single row in these tables corresponds to a single `ITEMID`. The two definitions tables deviate somewhat. `D_ITEMS` contains a `CATEGORY` field that denotes the general category of the `ITEMID` (for example "medications" or "surgical procedures"). In addition, `UNITNAME` denotes the standard unit of measurement for an `ITEMID`, but may be overridden by `VALUEUOM` in `CHARTEVENTS`. In contrast, `D_LABITEMS` does not contain a field with a default unit of measurement. It also contains a substitute for `CATEGORY`, namely `FLUID`, which denotes the fluid that the laboratory test originated from (e.g. urine or blood). For a full description of the entire MIMIC-III database we refer the reader to Appendix C.1, and for a full description of the meaning of every column we refer the reader to the official MIMIC-III documentation[1].

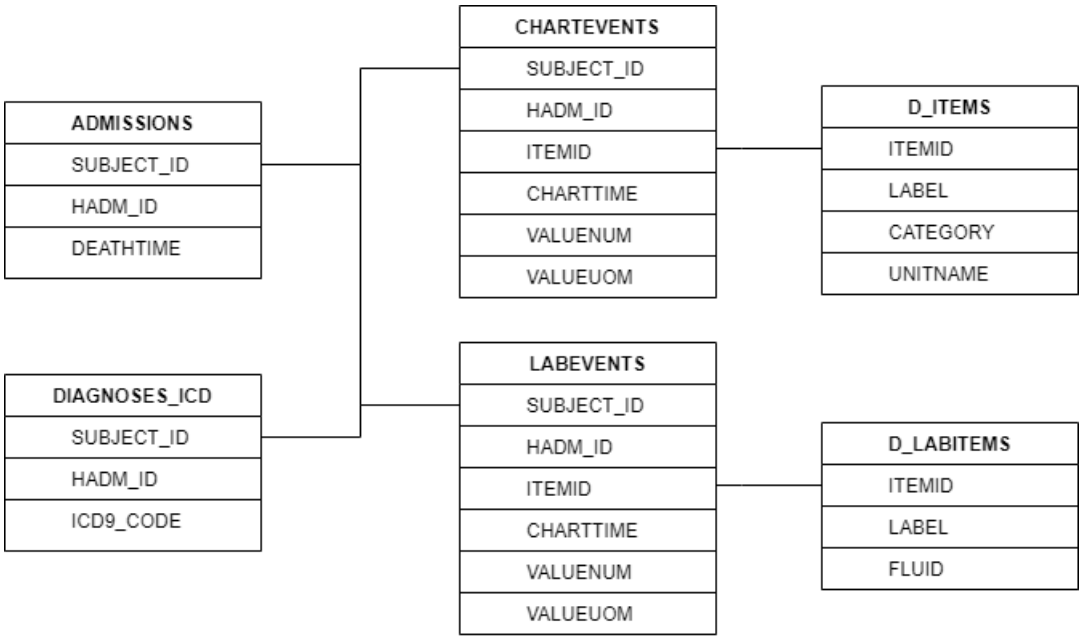

Figure 5.1: Diagram of the relevant tables and fields in MIMIC-III

[1]The official MIMIC-III documentation can be found at `https://mimic.physionet.org/about/mimic/`

5.3.2 Filtering the Events on Septic Shock

CHARTEVENTS and LABEVENTS are enormous tables: they contain approximately 331 and 28 million events, respectively. This means that these tables must be filtered in order to be workable. One way to do so is to filter them on only events of interest: those that relate to hospital admissions involving septic shock.

The preferred way of identifying patients with septic shock is to use the most recent official septic shock definition that was shown earlier in Figure 2.1. Now, according to this definition, a patient has septic shock if he or she has SOFA > 2 and a MAP $\geq$ 65mm Hg with a serum lactate level >2 mmol/L. Identifying at what time which patients conform to these criteria is extremely challenging. The 7 SOFA variables include clinical observations, bedside monitoring and lab events. In addition, MAP is hardly entered directly into CHARTEVENTS (see Appendix E.1 for details), meaning that it would have to be estimated from the more frequently measured systolic and diastolic blood pressure. This estimation, however, is only valid during normal resting heart rates. Combining all the previous, it is thus clear that identifying septic shock patients based on the most up-to-date septic shock definition by Singer et al. (2016) requires very complex processing over a very large amount of data. This is challenging to implement, both in terms of conceptual complexity and computational feasibility. We therefore refrain from this way of septic shock identification in this research.

Another way to identify septic shock in MIMIC-III is via *diagnosis codes*, i.e. based on whether a clinician diagnosed a patient to have septic shock. Now, finding all hospital admissions of patients diagnosed with septic shock is relatively easy - we can look up diagnoses in the DIAGNOSES_ICD table, filter on the ICD9_CODE code for septic shock (785.52) and then retrieve all corresponding SUBJECT_IDs and HADM_IDs. Subsequently, we can filter CHARTEVENTS and LABEVENTS on these SUBJECT_IDs and HADM_IDs to retrieve only those events that relate to hospital admissions and patients that involved septic shock at some point in time. This approach is much simpler and easier to execute than the approach based on the sepsis-3 definition. It does, however, present several limitations. The first is that we rely on a subjective clinician judgement rather than objective criteria, meaning that we may include patients that did not objectively have septic shock or exclude patients that objectively did have septic shock. The second is that the approach results in the inclusion of all events of all patients that were diagnosed with septic shock at some point, rather than only including events from the time during which patients were in septic shock. As the time of a diagnosis is not stored, this information is also non-recoverable with this approach. Finally, patient filtering via diagnosis codes inevitably results in the usage of older definitions of septic shock because MIMIC-III was recorded between 2001 and 2012, which is before the announcement of the sepsis-3 definition. We are aware that these limitations can have a strong impact on the results of this research. We therefore consider future research that addresses these limitations invaluable.

For executing the filtering on septic shock patients, we use the implementation by Johnson et al. (2018) in the official MIMIC-III code repository (Johnson, Stone et al., 2017). For more details regarding the implementation, we refer the reader to Appendix C.

5.3.3 Selecting Laboratory Testing Events

Knowing which tables are relevant in MIMIC-III and having filtered the event tables on septic shock patients such that they are workable, we can now extract the appropriate laboratory testing events from LABEVENTS. A high-level overview of this procedure is shown in Figure 5.2. First, we perform a search on the D_LABITEMS table in order to identify which measurement types (hereinafter referred to by their identifier, "ITEMID") relate to our laboratory testing variables. More specifically, we perform a lower-case keyword search on the LABEL column while only considering measurement types that are drawn from blood. Which keywords were searched for and how they correspond to the 20 laboratory testing variables can be seen in Appendix D.1.2. We discard measurement types from fluids such as urine and bone marrow because we cannot consider a measurement

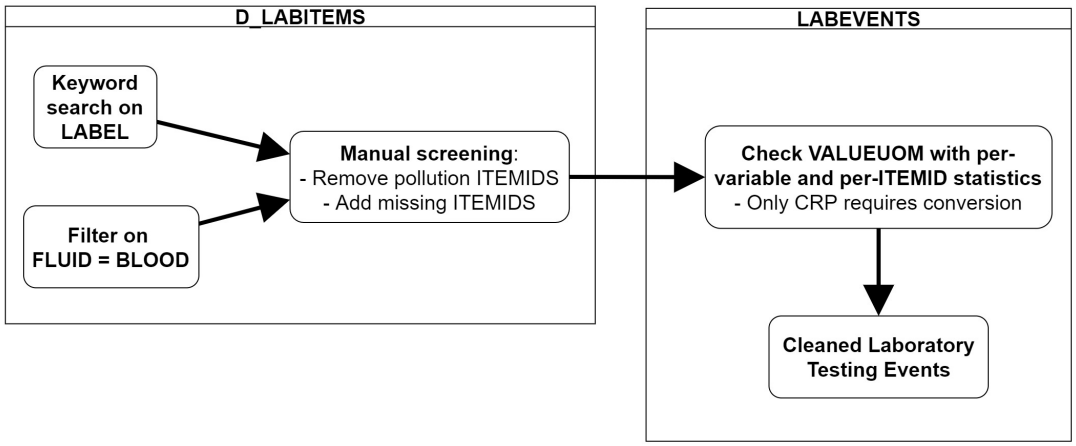

Figure 5.2: Diagram of the data conversion process for obtaining the relevant laboratory testing events. Operations on `D_LABITEMS` are shown in the left box and operations on `LABEVENTS` are shown in the right box. Note that `LABEVENTS` is already filtered on septic shock patients here.

the same variable when it is measured in different fluids: for instance, a glucose measurement in urine will have tremendously different values than a glucose measurement in blood. Incorporating measurements from all the different types of fluids in our models would require seeing them as distinct variables, which would lead to an unmanageable large amount of variables. To avoid this issue and to parallel Fialho et al. (2016), we thus restrict ourselves to measurements originating from blood tests. After the keyword search, various pollution `ITEMID`s pop up that do not clearly correspond to the variable that is searched for. We manually delete these pollution terms. Next, we manually add various `ITEMID`s that are difficult to add via a keyword search (for example the `ITEMID` corresponding to "pH", which would result in an excessive amount of pollution `ITEMID`s when keyword-searched).

Having identified a set of `ITEMID`s that potentially correspond to our desired laboratory testing variables, the next step is to check whether the units of measurement are uniform within each `ITEMID` and variable. We firstly check whether all events have the same units of measurement by inspecting the distinct values for `VALUEUOM` in the events. As it turns out, hardly any of the lab events require a conversion (only CRP). Now, we also inspect basic statistics of each `ITEMID` in each variable in order to see whether we can truly consider them as the same type of measurement. If the difference between the means of two `ITEMID`s that correspond to the same variable is small compared to their standard deviations, we consider it acceptable to consider both `ITEMID`s the same variable. If some of the `ITEMID`s within a single variable do not conform to this requirement, then we drop them. Afterwards, we have obtained a cleaned set of laboratory testing events. Note that in contrast to the bedside monitoring variables, we choose not to perform a hard range check as it is not so straightforward to definitively conclude whether a lab measurement was incorrect, particularly when compared to bedside monitoring variables.

5.3.4 Selecting Bedside Monitoring Events

In addition to variables originating from laboratory testing, we require variables from bedside monitoring. These measurements are stored in the `CHARTEVENTS` table, which also hosts information from other events such as clinical observations and medication administration. Notably, `CHARTEVENTS` also contains measurements regarding laboratory testing, but the MIMIC-III documentation informs us that these are copies from `LABEVENTS`, as it is often convenient for clinicians to enter lab results into bedside monitoring systems. We thus do not extract any information

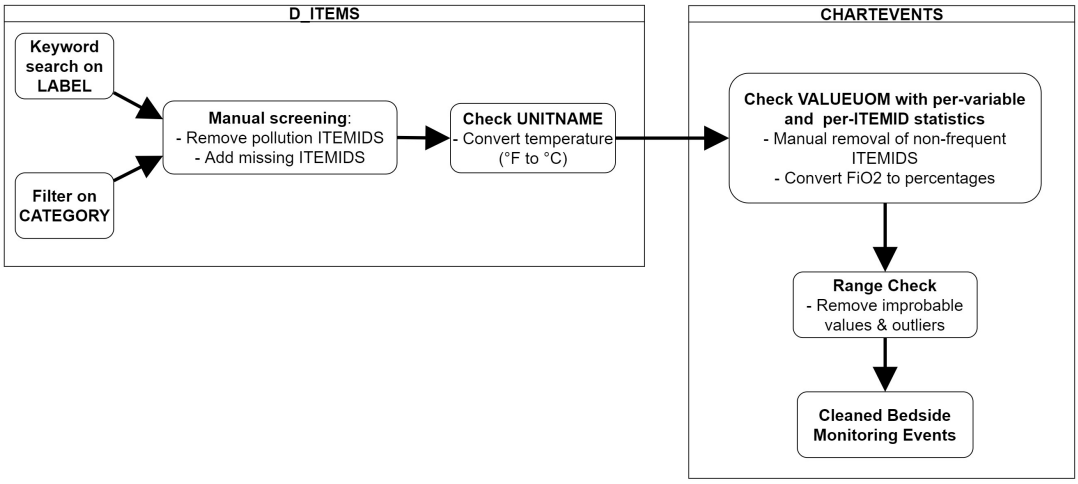

Figure 5.3: Diagram of the data conversion process for obtaining the relevant bedside monitoring events. Operations on `D_ITEMS` are shown in the left box and operations on `CHARTEVENTS` are shown in the right box. Note that `CHARTEVENTS` is already filtered on septic shock patients here.

regarding laboratory tests from `CHARTEVENTS`.

For the bedside monitoring variables, we specify a similar albeit slightly more complex extraction procedure as for the laboratory testing variables. Figure 5.3 shows a high-level overview of the steps that constitute this process. Again, we perform a lower-case keyword search on the `LABEL` column of the measurement definitions table (`D_ITEMS`) in order to find potential `ITEMID`s that correspond to variables of interest. The keywords used in this search can be found in Appendix D.1.3. For the bedside monitoring variables, we simultaneously filter on `CATEGORY` because `CHARTEVENTS` contains many other types of events that we are not interested in with regard to the bedside monitoring variables (e.g. medications, surgical procedures and nutrition). The selected categories can also be found in Appendix D.1.3. Like with laboratory testing, we again manually remove pollution `ITEMID`s and add missing `ITEMID`s. A difference with laboratory testing, however, is that the definitions table contains a unit designation as well (namely the `UNITNAME` field). Only temperature contains conflicting units as some `ITEMID`s are in Fahrenheit and some in Celsius. We therefore convert all measurement to Celsius.

Afterwards, we perform the same check as on the laboratory testing variables. We first inspect the distinct values for `VALUEUOM` for each `ITEMID` with the same variable, and convert units if required (which turns out only to be necessary for FiO2, which is sometimes a fraction and sometimes a percentage). Next, we check the per-variable and per-`ITEMID` statistics again, which led to the exclusion of various `ITEMID`s (we will provide more detail in Chapter 6). Finally, the remaining `ITEMID`s of the bedside monitoring variables seem to be plagued by unrealistic measurement values much more severely than the laboratory testing variables. For instance, various entries with blood pressures over 1000 can be found, which are not physically possible and thus measurement errors. We address this issue by removing all events with unrealistic values through checking that each measurement is within the allowed range for the specific variable. The used variable ranges are specified in Appendix D.1.3. Afterwards, we have obtained a cleaned set of events corresponding to our bedside monitoring variables of interest.

5.3.5 Converting the Events to the Modeling Data Format

Having obtained two sets of filtered events (for laboratory testing and bedside monitoring), the data can now be converted to the target format as shown in Table 5.2. A non-trivial step in this

process (in terms of implementation complexity) is to extract the median time between events for each variable, as required for testing filling strategy 2 from Figure 4.3. Recall that this information is used as the "maximum carry-forward time" to fill missing values. When determing the time between events, it is of utmost importance to take into account that the data contains different patients, and that a single time between two events must thus be between two events of the same patient. This is perhaps obvious, but easily overseen in the implementation. For more detail on this data transformation process, we refer the reader to the details on our code and implementation in Appendix C. The final step to be performed is the addition of the target label: patient mortality within 72-hours. This is a rather straightforward merge between the dataset with the features and the `ADMISSIONS` table, followed by a computation to translate the time of death into a binary label that indicates mortality within 72 hours at each timestep. Afterwards, we have obtained the desired data format as shown in Table 5.2.

5.4 Imputation and Normalization

Even after executing a filling strategy, many variables in the dataset still contain a significant amount of missing values (more detail wil follow in Chapter 6). Note that even full forward filling (strategy 3 in Figure 4.3) still results in missing values, e.g. before a particular variable is measured for a patient. In order to fit PFS and deep learning models, these missing values must be dealt with as they cannot deal with null values natively. Omitting all instances that contain one or more missing values would result in dropping a substantial amount of instances and thereby creating a significant bias in the data, distancing it from clinical reality. Therefore, we resort to the standard practice of imputing all missing values with the means of the corresponding variable. The mean is computed over the training set to avoid leakage into the validation and test sets.

Afterwards, the variables are normalized by subtracting their means and dividing by their standard deviations. Such normalization sets all variables on an equal scale, which aids the clustering (for the PFS) to pay attention to all features evenly as our clustering methods use the euclidean distance measure. Normalization is also advised before fitting deep learning models as it aids the backpropagation, which was earlier discussed in 3.3.2. Again, to avoid leakage between the training, validation and test sets, the mean and standard deviation of the variables are computed on the training dataset (after the imputation). Note that the imputation and normalization happens before our tests regarding the imputation strategy, splitting strategy and PFS/deep learning models, but after our exploratory analysis.

5.5 Model Tuning

In the effort to obtain an appropriate model fit for the PFS and deep learning models, we tune several aspects. The main aspects of these model tuning processes were already discussed in Section 4.2, but in this section we provide additional details. Table 5.3 shows the main aspects of each model that were tuned in this research.

As mentioned earlier, we test 1 to 7 features for the PFS (selected from the top of the variable ranking based on mutual information) and 2 to 7 rules. In addition, we test both fuzzy c-means clustering and K-Means clustering because of the earlier described issue with multiple cluster centers in the same position and a halting gradient descent as a result. We perform a grid search over all possible combinations involving these 3 factors, resulting in the fitting of 7x6x2=84 models. Fitting these models took 157 minutes on our hardware (a high-end laptop, more detail regarding our hardware can be found in Appendix C).

Tuning the deep learning models is more challenging because of the many hyperparameters in such models and the relatively expensive training procedure. As mentioned in the influential work by Goodfellow et al. (2016), a random search is preferable over a grid search when tuning the numerous hyperparameters in a deep learning network because it results in a quicker, more

Table 5.3: The design choices/hyperparameters explored in this research, including their respective value ranges in the random search.

Model	**Design choice / hyperparameter**	**Value Range / Search type**
PFS	Number of features	[1,7]
	Number of rules	[2,7]
	Clustering method	FCM, K-Means
	Search type	Grid
DNN	Number of layers	[1,10]
	Number of units per layer	[10,500]
	Dropout rate (between layers)	[0,0.3]
	Search type	Random
RNN (GRU)	Number of layers	[1,2]
	Number of units per layer	[2,20]
	Dropout rate (input-to-hidden)	[0,0.1]
	Search type	Random

diverse exploration of the solution space. The hyperparameter settings for the deep learning models can be found in Table 5.3. The rationale for tuning these particular hyperparameters was discussed in Section 4.2.2. The value range for the number of units and the number of layers was determined through manual tests with the objective of keeping the running time of a single model training reasonable. The number of units in the densely connected neural network (DNN) starts at 10 in order to avoid severe bottlenecks in the network that usually hamper performance. Likewise, the dropout rate is kept relatively small, i.e. at a maximum of 0.3. The dropout layers are put between any two layers except before the final layer in similar vain: to avoid bottlenecks because of concatenated dropout layers. The last layer is frozen in every configuration: it contains a single unit and sigmoid activation, which is a requirement as our output is a probability of binary class membership. The value range of the number of layers in Table 5.3 is excluding this static final layer. For the GRU, fewer layers and fewer units per layer were tested because a single GRU unit is more expensive to train due to the backpropagation through time in this network. Furthermore, an input-to-hidden dropout with a maximum of 0.1 was used, because higher dropout rates up to and including 0.3 led to unsuccessful model training in preliminary tests. We refrain from using hidden-to-hidden dropout because this usually seems to have harmful effects (Goodfellow et al., 2016) due to a loss of information over the input data. As the computational resources in our research are limited, a time budget was set for the hyperparameter tuning of the deep learning models. More specifically, each random search was given a time budget of 1 hour on our local machine, leading to a total of 4 hours (the options include a DNN with all features, a DNN with the same features as the best PFS, a GRU network with all features and a GRU network with the same features as the best PFS).

Various other, static aspects in model design are involved in our experiments as well. As mentioned earlier, we use the Adam optimizer (Kingma & Ba, 2014) in the gradient descent optimization of the PFS and the deep learning models, and keep the learning rate at its default value of 0.01. As we are attempting binary classification, binary cross entropy is utilized as the loss function. The intermediate layers in the DNN use the default rectified linear unit activation (Chollet, 2017), while the GRU layers use the default tanh activation (Abadi et al., 2015). Furthermore, the batch size is set to 512 for the PFS and DNNs to achieve a balance between speed of convergence and accurate gradients (more observations in a single batch yields better gradient estimates, but more expensive

gradient estimation). For the GRU networks, we are forced to use a batch size of 1 (i.e. a sequence of 1 patient) for technical reasons: due to the high variability in the lengths of the sequences of the patients, and because a single batch must contain sequences of equal length during training, using a larger batch size than 1 generally leads to excessive padding (e.g. padding a sequence of length 10 to length 1000 because 1 patient in the batch has a long sequence) and exploding computational demands as a result. Furthermore, to balance training time and certainty of convergence, early stopping is used jointly with a maximum number of epochs (100 epochs). Early stopping for the DNN is conditioned on 5 epochs without improvement in validation ROC-AUC, where an epoch must improve at least 0.5% to be considered an improvement. We thereby assume an improvement of less than 0.5% in ROC-AUC is not meaningful during training. Because of its computational expensiveness, a stricter early stopping criterion was used for the GRU networks, namely early stopping after 1 epoch without an improvement of at least 5%. We realize that this is quite a strict requirement and may lead to incomplete model training, but in practice it seems that our GRU networks converge before the end of the first epoch, and training a GRU for a single epoch can take 5 to 15 minutes depending on the hyperparameters. It thus seems that a reasonable model fit can be achieved under a relatively strict early stopping criterion. All other design choices and hyperparameters that were not mentioned in this section were left at their defaults.

We would like to emphasize, once again, that our model tuning is not exhaustive. A first limitation that comes to mind is that a separate feature selection for deep learning was not executed in the light of the theoretical capability of deep learning to engineer features automatically through its stacked layers and the gradient descent optimization procedure. In practice, however, randomness and bias in the data can prevent deep learning from the ideal automatic feature engineering (Chollet, 2017) through supervised learning, thus it would be better to also perform feature selection for deep learning externally if the highest level of performance is desired. As deep learning already involves many hyperparameters and computationally expensive model fitting procedures, such efforts were not executed in this research (apart from fitting deep learning models on the features selected for the best PFS). Likewise, the diversity of the explored architectures for the GRU networks has been limited due to the high computational demands of training such networks. Future works that can afford the usage of heavier hardware should investigate using more layers, more units, and hybrid models with both GRU- and densely connected layers. Furthermore, brief initial experimentation with using both dropout and L1-/L2-regularization seemed to indicate that using both at the same time was not effective, even when setting the dropout rates and regularization parameters rather low (i.e. at a maximum of 0.05). A cause for this phenomenon could be that additional constraints on the regularization norms are required (Goodfellow et al., 2016) in order to stabilize the learning process. Implementing such constraints would make the implementation of having both dropout rates and regularization parameters rather complex - a challenge which we refrain from in this research. Using bigger models with more regularization methods is therefore also an interesting avenue for future research. Finally, many more different directions could be explored when tuning the models, among which testing multiple learning rates, testing multiple gradient descent algorithms, using other feature selection methods, using different initialization methods of the PFS, using different membership functions in the PFS, using an automated form of hyperparameter tuning (Hutter et al., 2019) and so forth. It is clear that many avenues for future research remain regarding these aspects.

A final aspect that deserves mention is that we use a static validation set, implying that validation set overfitting may occur due to frequently repeated model assessment on the exact same validation set. Obviously, it would be desirable to generate random training and validation splits during model tuning, such that each assessment is on a different training and validation set (i.e. as in cross-validation). However, this practice is difficult and expensive to execute on our data due to the large size, the required variable filling strategies, the patient-instance hierarchy and the highly variable number of instances per patient in the data. Solving all these issues would require a considerable amount of additional computational resources and would also significantly increase the conceptual complexity of the model assessment procedure.

Chapter 6

Results

This chapter starts with the results from the preprocessing that was discussed in the better part of the previous chapter. Afterwards, we test the filling and splitting strategies to determine the best way to fill missing values and to split the data in a training, validation and testing set. Lastly, we perform an exploratory analysis on the dataset that arises from the best filling and splitting strategy, followed by the tuning and assessment of the PFS and deep learning models.

6.1 Preprocessing

6.1.1 Input Data Size and Filtering on Septic Shock Patients

The first set of results emerges from the execution of the preprocessing. In order to get a feel for the size of the raw input data from MIMIC-III, the number of records (rows) per relevant table is shown in Table 6.1. The data includes information from 46,520 different patients.

Table 6.1: Size of the (raw) relevant tables in MIMIC-III.

Table Name	ADMISSIONS	CHARTEVENTS	D_ITEMS	D_LABITEMS	DIAGNOSES_ICD	LABEVENTS
Number of Rows	58,976	330,712,483	12,487	753	651,047	27,854,055

By filtering on hospital admissions that include a septic shock diagnosis, the number of patients drops to 2,416, the number of chartevents drops to 32,303,461 and the number of labevents drops to 2,066,973. This filtering step thus reduces the data considerably: the number of patients is reduced approximately 20 times while the number of chart- and labevents is reduced approximately 10 times. This makes the data much more manageable for the preprocessing in the remainder of the pipeline.

6.1.2 Selecting Laboratory Testing Events

Table 6.2 shows the eventual selection of `ITEMID`s and variables as well as the statistics originating from their events in `LABEVENTS`. Obtaining this table involves several steps (which were shown in Figure 5.2). Details on the removed pollution `ITEMID`s, manually included additional `ITEMID`s and unit conversions can be found in Appendices D.1.2 and D.1.1, respectively. Inspecting Table 6.2 in more detail, we see that most laboratory testing variables only contain a single `ITEMID`. For the variables that do contain multiple `ITEMID`s (bicarbonate, glucose, hematocrit, potassium, sodium and white blood cells), we see that the means are relatively close and the difference between these means is well within the standard deviations of the `ITEMID`s. Thus, we conclude it is reasonable to see these `ITEMID`s as a single variable. It must also be noted that various `ITEMID`s seem to have rather high maxima compared to their means. This is particularly the case for glucose,

GOT(ASAT), GPT(ALAT) and white blood cells. As it is not so obvious to define minimum and maximum values for the laboratory testing variables in similar vain as for the bedside monitoring variables, these outliers were left in the data during the preprocessing. The outliers can, if desired, be handled later on during the modeling stage as well. Furthermore, leaving them in is not too harmful for the performance of the rest of the preprocessing because the number of outliers in the filtered `LABEVENTS` table is relatively small.

The resulting filtered laboratory events table has 865,988 rows, meaning that this filtering process approximately halves its size. This is interesting as we initially had 753 different `ITEMID`s among the laboratory events, while we now only use a total of 26 different `ITEMID`s for the laboratory testing variables. It thus appears that we are interested in relatively common laboratory tests rather than very rare, specific tests. AT3 and CRP seem to be exceptions to this rule as they only appear in 4 and 484 events, respectively. We undertook a brief manual investigation into whether we missed any `ITEMID`s for these variables, but this does not seem to be the case. Due to the extremely low occurrence of AT3, it was dropped from further consideration after this point. Another observation is that the number of distinct patients is 2,414 in the filtered labevents, meaning that two patients do not have any laboratory events at all.

Table 6.2: The eventual selection of `ITEMIDs` for the laboratory testing variables and their statistics. Note that all statistics are after unit conversion.

Variable Name	ITEMID	LABEL	Number of Times Measured	Mean	Standard Deviation	Min	Median	Max	Units (Distinct Units in VALUEUOM)
Arterial PCO2	50818	pco2	51957	42.13	11.72	11.00	40.00	176.00	mm Hg
Arterial base excess	50802	base excess	51959	-1.61	6.19	-50.00	-1.00	31.00	mEq/L
Arterial pH	50820	ph	54494	7.36	0.10	6.35	7.37	7.75	units
Arterial PO2	50821	po2	51968	111.93	58.04	0.00	100.00	649.00	mm Hg
AT3	51140	antithrombin	4	35.38	25.04	4.50	36.00	65.00	%
Bicarbonate	50803	calculated bicarbonate. whole blood	548	21.92	6.59	5.00	21.00	49.00	mEq/L
	50882	bicarbonate	55478	23.89	5.80	5.00	24.00	50.00	mEq/L
Bilirubin	50885	bilirubin, total	18932	4.62	7.69	0.00	1.40	82.20	mg/dL
Calcium	50893	calcium, total	48965	8.22	0.89	1.80	8.20	27.40	mg/dL
Creatinine	50912	creatinine	55026	1.85	1.60	0.00	1.30	22.00	mg/dL
CRP	50889	c-reactive protein	484	107.37	77.92	1.00	87.15	299.90	mg/L, mg/dL
Glucose	50809	glucose	9960	133.02	60.27	10.00	121.00	891.00	mg/dL
	50931	glucose	54964	133.99	65.80	4.00	121.00	3565.00	mg/dL
GOT(ASAT)	50878	asparate aminotransferase (ast)	18570	244.76	1005.12	3.00	50.00	23830.00	IU/L
GPT(ALAT)	50861	alanine aminotransferase (alt)	18586	150.12	481.07	0.00	38.00	8405.00	IU/L
Hematocrit	50810	hematocrit, calculated	3464	30.46	5.99	0.00	30.00	56.00	%
	51221	hematocrit	54449	28.97	4.68	2.10	28.50	61.00	%
Platelets	51265	platelet count	49207	214.87	165.50	5.00	180.00	1479.00	K/uL
Potassium	50822	potassium, whole blood	9610	4.16	0.82	1.40	4.00	12.90	mEq/L
	50971	potassium	58837	4.09	0.66	1.80	4.00	12.70	mEq/L
PTT	51275	ptt	35642	48.04	26.18	16.50	38.80	158.40	sec
Sodium	50824	sodium, whole blood	3033	136.33	6.88	14.00	136.00	181.00	mEq/L
	50983	sodium	57211	138.81	5.67	102.00	139.00	182.00	mEq/L
Urea	51006	urea nitrogen	54869	38.29	27.61	1.00	30.00	242.00	mg/dL
White	51300	wbc count	63	10.90	8.68	0.10	8.40	38.90	K/uL
blood cells	51301	white blood cells	47362	13.01	9.61	0.00	11.10	462.60	K/uL

6.1.3 Selecting Bedside Monitoring Events

Table 6.3 shows the eventual selection of `ITEMID`s and variables as well as the statistics originating from their events in `CHARTEVENTS`. Again, obtaining this table involved various steps (which were shown in Figure 5.3). Details on the removed pollution `ITEMID`s, manually included additional `ITEMID`s, unit conversion and value range bounds can be found in Appendices D.1.3, D.1.1, D.1.1 and D.1.3, respectively. At a glance, we can see in Table 6.3 that bedside monitoring variables generally contain a larger amount of distinct `ITEMID`s than laboratory testing variables. It is our suspicion that this is the case because different types of hardware are used to measure vital signs, and these different types of hardware probably log their measurements under distinct `ITEMID`s.

Inspecting the official MIMIC-III documentation reveals that the Beth Israel Deaconess Medical Center used a different clinical information system in the period 2001-2008 than from 2008 onward, and that ITEMIDs from the latter system have a value >220,000. This also explains why we see both a large amount of events with low ITEMIDs and a large amount of events with high ITEMIDs within each variable. Another key difference with respect to the laboratory testing events is that a significant amount of the ITEMIDs had to be removed because they had a unclear units in VALUEUOM or means that strongly differed from the other ITEMIDs within the particular variable. The full list of exclusions after the statistics check in this step can be found in Table D.8. All these exclusions involved ITEMIDs that appear in only up to 100 events as well, and are thus highly unlikely to exert a strong influence on the final results of this research.

The removal of measurements outside the range bounds resulted in the removal of 6,330 events, which is a small amount compared to the 3,294,177 events that remain after the filtering. CHARTEVENTS was thus reduced by approximately a factor 10 through the filtering on relevant ITEMIDs. This is a considerably larger factor than for the laboratory testing events, which is expected knowing that CHARTEVENTS contains many events that are outside our scope (e.g. nutrition, medications, surgical procedures, etc.). The number of patients in the filtered bedside monitoring events table is 2,392, implying that more patients do not have a presence in this table compared to the filtered laboratory events table. This is somewhat surprising as intuition would lead one to conclude that not measuring vital signs for septic shock patients should be extremely rare. Perhaps the bedside monitoring data of these patients was corrupted, and was subsequently excluded by the creators of the MIMIC-III database.

Table 6.3: The eventual selection of `ITEMIDs` for the bedside monitoring variables and their statistics. Note that all statistics are after unit conversion.

Variable Name	ITEMID	LABEL	Number of Times Measured	Mean	Standard Deviation	Min	Median	Max	Units (Distinct Units in VALUEUOM)
CVP	113	cvp	117810	12.65	5.72	1.00	12.00	35.00	None, mmHg
Diastolic blood pressure	8368	arterial bp [diastolic]	185373	58.01	13.27	20.00	56.00	150.00	mmHg
	8441	nbp [diastolic]	107206	55.14	14.57	20.00	54.00	150.00	mmHg
	8555	arterial bp #2 [diastolic]	1046	57.55	15.61	25.00	55.00	143.00	mmHg
	220051	arterial blood pressure diastolic	202090	57.85	12.50	20.00	56.00	150.00	mmHg
	220180	non invasive blood pressure diastolic	151942	59.17	14.50	20.00	58.00	150.00	mmHg
	225310	art bp diastolic	13031	56.94	12.64	20.00	56.00	149.00	mmHg
FiO2	190	fio2 set	56048	50.01	15.68	21.00	50.00	100.00	None, torr
	223835	inspired o2 fraction	127336	47.28	14.08	0.00	40.00	100.00	None
Heart rate	211	heart rate	278190	89.43	19.24	0.00	88.00	214.00	BPM, bpm
	220045	heart rate	438947	94.25	18.92	0.00	94.00	216.00	bpm
SpO2	646	spo2	273708	97.05	3.93	30.00	98.00	100.00	%, None
	220277	o2 saturation pulseoxymetry	426691	96.46	3.95	30.00	97.00	100.00	%
Systolic blood pressure	51	arterial bp [systolic]	185272	115.65	23.56	20.00	113.00	200.00	None, mmHg
	455	nbp [systolic]	107640	112.30	21.52	20.00	110.00	200.00	None, mmHg
	6701	arterial bp #2 [systolic]	1048	109.35	20.54	55.00	108.00	176.00	None, mmHg
	220050	arterial blood pressure systolic	202152	113.53	21.61	20.00	111.00	200.00	mmHg
	220179	non invasive blood pressure systolic	152249	112.33	20.85	20.00	110.00	200.00	mmHg
	225309	art bp systolic	13046	110.05	21.66	22.00	108.00	199.00	mmHg
Temperature	676	temperature c	13677	37.06	0.90	30.70	37.00	40.30	Deg. C, None
	677	temperature c (calc)	65468	36.90	0.90	30.28	36.83	42.17	Deg. C, None
	678	temperature f	65456	36.90	0.90	30.28	36.83	42.17	Deg. F, None
	679	temperature f (calc)	13679	37.06	0.90	30.70	37.00	40.30	Deg. F, None
	223761	temperature fahrenheit	78187	36.87	0.87	30.56	36.83	41.00	?F
	223762	temperature celsius	9468	36.83	1.16	31.20	36.90	42.50	?C
	226329	blood temperature cco (c)	2802	37.17	1.00	32.10	37.20	40.70	?C

6.2 Filling Strategies

Having obtained the cleaned laboratory testing and bedside monitoring events, we can now turn to testing the 3 strategies for filling missing values in the time series constructed from the misaligned and unevenly sampled events. For this test, we split the data in a case-based fashion such that 70% of the patients is in the training set and 30% of the patients is in the testing set. The decision for this way of splitting is a bit of a spoiler of the test in the next section, but as it turns out the hypothesized leaking occurs in observation-based random splitting, thus we must split in a case-based fashion. Regardless, this split results in having 637,860 instances in the training set and 286,983 instances in the test set, which approximately preserves the 70%-30% balance between the training and testing sets in terms of the number of instances.

For strategy 2 from Figure 4.4, we must first estimate the size of the forward filling window. Recall that the median time between two events of each variable was chosen as the window size, and that the the window size is estimated on the training set to avoid leakage between the training and testing datasets. Table 6.4 shows basic statistics regarding the time between events for each variable on the training set. As expected, the median time between bedside monitoring events is generally smaller than the time between laboratory testing events. Whereas the median time between bedside monitoring events is 1 or 2 hours, the median time between laboratory testing events is much more variable, ranging from 4 hours and 49 minutes for arterial pH to 4 days and

15 minutes for CRP. Note that CRP is a large outlier when compared to the other laboratory testing variables, which is probably caused by its rare occurrence (CRP only contains 484 events according to Table 6.2). Another noteworthy aspect of Table 6.4 is that all times between events are left skewed, i.e. have mean > median. The maxima are also very high compared to the medians. As we carefully defined a time to event to be for a single variable and a single patient, this must mean that some patients have very long hospital admissions and are tested only very infrequently on some of the variables, perhaps at admission and discharge only.

Table 6.4: The time between events for the variables. The format of the values is "days : hours : minutes : seconds".

Category	Variable	Mean	Median	Standard Deviation	Minimum	Maximum
Bedside Monitoring	CVP	0:01:24:50	**0:01:00:00**	0:08:36:10	0:00:01:00	43:10:00:00
	Diastolic blood pressure	0:00:52:12	**0:01:00:00**	0:06:26:19	0:00:00:00	139:16:08:00
	FiO2	0:02:22:34	**0:02:00:00**	0:09:22:25	0:00:01:00	46:12:06:00
	Heart Rate	0:00:48:14	**0:01:00:00**	0:06:10:59	0:00:00:00	139:16:57:00
	SpO2	0:00:49:07	**0:01:00:00**	0:06:16:29	0:00:00:00	139:15:55:00
	Systolic blood pressure	0:00:52:09	**0:01:00:00**	0:06:26:06	0:00:00:00	139:16:08:00
	Temperature	0:02:17:48	**0:02:00:00**	0:10:41:54	0:00:00:00	139:16:57:00
Laboratory Testing	Arterial base excess	0:09:16:07	**0:04:52:00**	1:04:17:51	0:00:01:00	89:00:16:00
	Arterial PCO2	0:09:16:08	**0:04:52:00**	1:04:18:06	0:00:01:00	89:00:16:00
	Arterial pH	0:09:14:45	**0:04:49:00**	1:05:39:31	0:00:01:00	137:01:45:00
	Arterial PO2	0:09:16:03	**0:04:52:00**	1:04:17:56	0:00:01:00	89:00:16:00
	Bicarbonate	0:15:22:23	**0:13:01:00**	0:10:14:28	0:00:00:00	25:03:29:00
	Bilirubin	1:09:03:54	**0:23:50:00**	1:22:51:02	0:00:03:00	38:07:25:00
	Calcium	0:17:06:12	**0:14:23:00**	0:12:59:27	0:00:01:00	21:06:25:00
	Creatinine	0:15:41:23	**0:13:19:00**	0:10:02:06	0:00:01:00	25:03:29:00
	CRP	6:08:34:45	**4:00:15:00**	7:16:33:01	0:03:25:00	50:01:16:00
	Glucose	0:13:12:16	**0:10:59:00**	0:10:37:46	0:00:00:00	25:03:29:00
	GOT(ASAT)	1:09:29:59	**0:23:52:00**	1:23:49:30	0:00:03:00	38:07:25:00
	GPT(ALAT)	1:09:25:20	**0:23:52:00**	1:23:21:14	0:00:03:00	35:01:53:00
	Hematocrit	0:14:51:53	**0:12:47:30**	0:10:41:41	0:00:00:00	12:00:13:00
	Platelets	0:17:34:36	**0:21:17:00**	0:10:35:36	0:00:01:00	12:00:13:00
	Potassium	0:12:32:17	**0:10:12:00**	0:10:02:50	0:00:00:00	21:20:26:00
	PTT	0:20:42:10	**0:14:02:00**	1:04:02:10	0:00:01:00	34:20:37:00
	Sodium	0:14:16:51	**0:12:02:00**	0:10:14:41	0:00:00:00	25:03:29:00
	Urea	0:15:43:55	**0:13:23:00**	0:10:02:15	0:00:01:00	25:03:29:00
	White blood cells	0:18:16:36	**0:22:20:00**	0:10:36:43	0:00:00:00	12:00:13:00

Table 6.5 displays how the different filling strategies differ in terms of the resulting missing values and mutual information w.r.t. the target label. The first strategy clearly results in severe sparsity of most variables: particularly laboratory testing variables have many missing values, as most of them are only present in approximately 5% of instances or even less. It is also interesting to see that most bedside monitoring variables are still non-missing in over 65% of the instances, with heart rate even being present in 79%. As we have seen previously, the number of bedside monitoring events is substantially larger than the number of laboratory testing events, and bedside monitoring events also turn out to be reported at largely at the exact same timestamp. From a clinical perspective, this makes sense as most of the bedside monitoring variables are probably measured at one go and thus reported simultaneously. The filling strategy with the median time between events for each variable as the maximum carry-forward window sizes greatly reduces the

missing values already, but the full forward filling strategy naturally results in a much higher percentage of non-missing values, namely one of over 90% for most variables. Regarding mutual information, we observe that all variables have quite a low value regardless of the filling strategy. Still, strategy 3 clearly results in the highest mutual information of each feature for every variable, making it the best choice according to this assessment. The difference with strategy 2, however, is quite small for the majority of variables.

Figure 6.1 shows the performance of the three gradient boosting decision trees on the three filling strategies. Once again, we observe that strategy 2 and 3 are clearly superior to strategy 1, both when assessing model performance on the ROC curve and on the Cohen's kappa curve. The difference between strategy 2 and 3, however, is small, but upon closer examination strategy 3 results the highest ROC-AUC and AUK. This is in line with the assessment based on mutual information, thus we conclude that strategy 3 is the best strategy for filling missing values among our considered alternatives. We thereby continue the remaining analyses in this research with this filling strategy.

Table 6.5: Percentages of present (non-missing) values and mutual information w.r.t the target label with the 3 filling strategies in the testing sets. Strategies 2 and 3 greatly decrease the number of missing values and increase the mutual information. Strategy 3 seems to result in the features with the largest mutual information.

		Percentage of non-missing values			**Mutual information w.r.t. target label**		
Category	**Variable**	**Strategy 1 (No Filling)**	**Strategy 2 (Window)**	**Strategy 3 (Full Forward)**	**Strategy 1 (No Filling)**	**Strategy 2 (Window)**	**Strategy 3 (Full Forward)**
Bedside Monitoring	CVP	14.01%	20.14%	36.98%	0.001	0.003	0.005
	Diastolic blood pressure	66.82%	95.56%	98.58%	0.006	0.010	0.011
	FiO2	22.90%	55.32%	85.17%	0.006	0.011	0.016
	Heart rate	79.08%	96.49%	98.67%	0.006	0.010	0.010
	SpO2	76.64%	94.37%	98.52%	0.012	0.014	0.016
	Systolic blood pressure	66.89%	95.60%	98.58%	0.013	0.020	0.021
	Temperature	17.78%	66.31%	98.37%	0.002	0.009	0.015
Laboratory Testing	Arterial base excess	5.57%	34.14%	90.78%	0.002	0.012	0.018
	Arterial PCO2	5.57%	38.85%	93.25%	0.001	0.005	0.010
	Arterial pH	5.87%	40.09%	93.88%	0.002	0.014	0.023
	Arterial PO2	5.57%	38.86%	93.25%	0.001	0.012	0.021
	Bicarbonate	5.81%	77.97%	99.79%	0.001	0.012	0.014
	Bilirubin	1.92%	45.77%	94.39%	0.000	0.026	0.034
	Calcium	5.06%	76.98%	98.91%	0.000	0.007	0.010
	Creatinine	5.68%	78.73%	99.79%	0.001	0.012	0.015
	CRP	0.05%	3.93%	9.96%	0.000	0.008	0.015
	Glucose	6.67%	72.60%	99.81%	0.000	0.015	0.021
	GOT(ASAT)	1.87%	45.04%	94.66%	0.001	0.049	0.064
	GPT(ALAT)	1.87%	45.07%	94.38%	0.000	0.039	0.048
	Hematocrit	5.97%	73.88%	99.83%	0.000	0.012	0.018
	Platelets	5.10%	92.88%	99.79%	0.001	0.040	0.045
	Potassium	7.03%	72.61%	99.83%	0.001	0.010	0.013
	PTT	3.76%	56.62%	97.61%	0.001	0.039	0.059
	Sodium	6.20%	76.25%	99.82%	0.001	0.006	0.009
	Urea	5.66%	78.80%	99.79%	0.001	0.013	0.016
	White blood cells	4.91%	94.75%	99.79%	0.001	0.035	0.036

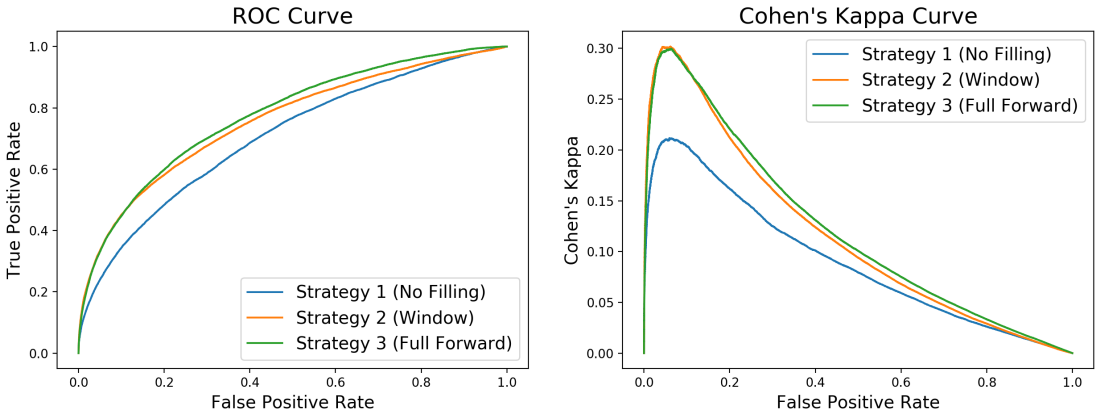

Figure 6.1: ROC and Cohen's kappa curves for the gradient boosting trees on the testing sets with the 3 filling strategies. The ROC-AUCs for strategy 1, 2 and 3 are 0.705, 0.757 and 0.775, respectively. The AUK values are 0.089, 0.116 and 0.121.

6.3 Splitting Strategies

To test the hypothesis that leakage would occur between the training and testing sets in case of the observation-based random splitting strategy from Figure 4.4, gradient boosting decision trees were trained and assessed on both strategies. General statistics regarding the training and testing sets for both strategies are shown in Table 6.6. Clearly, in the observation-based random splitting strategy, the vast majority of patients appears in both the training and the test set, which is not the case with the other splitting strategy (case-based splitting). The eventual performance of the gradient boosting decision trees can be seen in Figure 6.2. Particularly when inspecting the ROC curve and the ROC-AUC, we can see that the algorithm is able to exploit a leakage between the training and testing set as the ROC-AUC on the testing set is nearly 1. In the Cohen's kappa curve, we also observe that a threshold exists in which the algorithm achieves a Cohen's kappa near 1, indicating near-perfect classification. Both of this is not the case when performing case-base splitting - the performance of the prediction algorithm is much more realistic in this scenario. Thus, we conclude that leakage occurs in the random, observation-based splitting method, and continue the remainder of this research with the case-based splitting strategy.

Table 6.6: General statistics for the training and testing datasets with the splitting strategies.

	Observation-based Random Splitting		**Case-based Splitting**	
	Train	**Test**	**Train**	**Test**
Number of Patients	2416	2414	1691	725
Number of Instances	647,390	277,453	627,862	296,981
Percentage of Total Instances	70.0%	30.0%	67.9%	32.1%
Percentage of Instances with y=1	10.0%	9.95%	10%	9.7%

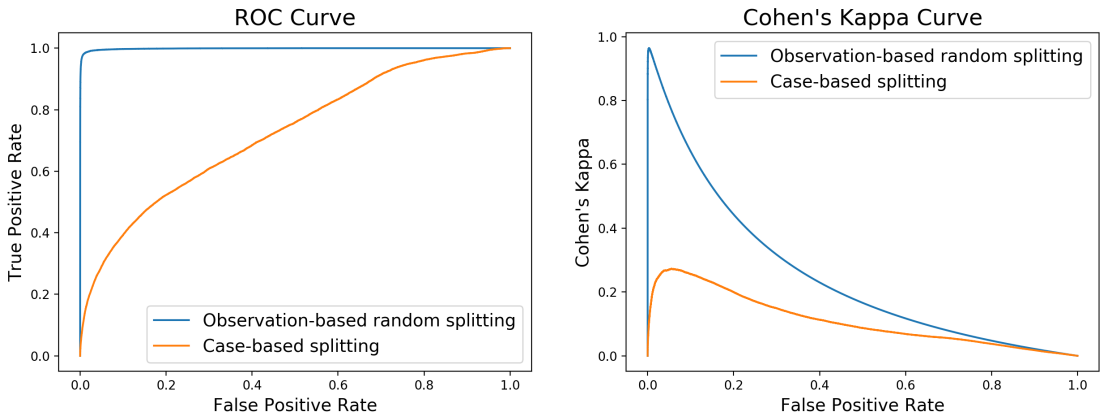

Figure 6.2: ROC and Cohen's kappa curves for the gradient boosting trees on the testing sets with the 2 splitting strategies. The ROC-AUCs for observation-based and case-based splitting are 0.999 and 0.725, respectively. The AUK values are 0.252 and 0.110.

6.4 The Final Dataset: Descriptive Statistics

Having determined the best filling strategy (full forward filling) and a splitting strategy without detrimental leakages (case-based splitting), we can now construct the final dataset used for comparing the PFS and deep learning models. In this section, we inquire this dataset in-depth to achieve an adequate understanding of its contents such that we can more deeply interpret the results from subsequent analyses regarding the PFS and deep learning models.

Descriptive statistics of the features in the full dataset are displayed in Table 6.7, and visualized as histograms in Appendix D.2.1. Note that the statistics in this table are, by definition, different from those in Table 6.2 and 6.3 because of the processing on the events that happens afterwards. Note that the statistics in Table 6.7 are also before imputing the still-missing values of each variable with their global means. Although most of the bedside monitoring variables are only missing in a small percentage of the instances in the data, CVP is still missing in the majority of instances (66.67%) and FiO2 also still has a substantial missingness rate (13.15%), probably because not all septic patients are attached to a mechanical ventilator. Regarding the laboratory testing variables, we observe that a majority has either less than 1% missing values or approximately 5%. Again, some exceptions are present. Particularly CRP has an exceptionally high amount of missing values, which is only natural considering that we already found earlier that CRP events are relatively rare. Furthermore, it appears that the feature distributions seem rather similar when grouping the data by the two target label values (see Appendix D.2.2), indicating that in isolation, most features do not seem to possess a clear linear relationship with the target label.

An interesting question one may ask after inspecting Table 6.7 is whether *informative missingness* (Rubin, 1976) is present in the dataset, i.e. whether the missingness of a variable is informative with respect to the target label. A way to informally assess the informative missingness of a feature is to compare the fraction of instances with a target label value of 1 between the two groups with a missing and non-missing value, respectively. This method, however, is a rather subjective exercise. In order to formally establish informative missingness of a variable, we should perform some statistical test to assess whether the group with missing values has a different fraction of instances with a target label value of 1 (i.e. mortality within 72 hours) than the group with non-missing values. However, the semantics of our dataset make it difficult to find a suitable statistical test because (1) the samples have different sizes (missing vs non-missing), (2) the variables in each test (missingness and the target label) are both dichotomous and (3) the samples are certainly not dependent (no clear pairing between the patients in the instances of both samples) but also

Table 6.7: Descriptive statistics of the features in the dataset used for modeling.

Category	**Variable**	**Mean**	**Std.**	**Min.**	**Median**	**Max.**	**Missing values (%)**
Bedside monitoring	Diastolic blood pressure	58.59	13.11	20.00	57.00	150.00	1.45%
	CVP	12.44	5.72	1.00	12.00	35.00	66.97%
	FiO2	50.21	17.33	0.00	50.00	100.00	13.15%
	Heart rate	92.16	19.02	0.00	91.00	215.00	1.34%
	SpO2	96.61	4.33	30.00	97.67	100.00	1.53%
	Systolic blood pressure	114.78	21.02	20.00	112.00	200.00	1.45%
	Temperature	36.84	0.83	30.28	36.83	42.50	1.68%
Laboratory Testing	Arterial base excess	-0.16	6.11	-50.00	0.00	31.00	8.12%
	Arterial PCO2	42.63	12.35	11.00	40.00	176.00	6.60%
	Arterial pH	7.38	0.08	6.35	7.39	7.75	5.67%
	Arterial PO2	103.81	52.54	0.00	96.00	649.00	6.57%
	Bicarbonate	24.05	6.27	5.00	23.54	50.00	0.33%
	Bilirubin	2.70	5.79	0.00	0.70	82.20	5.25%
	Calcium	8.23	0.86	1.80	8.20	27.40	1.21%
	Creatinine	1.96	1.63	0.00	1.40	22.00	0.32%
	CRP	148.65	85.14	1.00	171.60	299.90	85.04%
	Glucose	134.44	55.66	4.00	124.00	3565.00	0.30%
	GOT(ASAT)	130.39	640.72	3.00	35.00	23830.00	5.64%
	GPT(ALAT)	87.92	324.93	0.00	26.00	8405.00	5.70%
	Hematocrit	28.91	4.36	0.00	28.50	61.00	0.28%
	Platelets	224.81	158.35	5.00	193.00	1470.00	0.31%
	Potassium	4.10	0.60	1.40	4.00	12.70	0.28%
	PTT	41.33	20.52	16.5	34.70	158.40	1.59%
	Sodium	139.32	5.21	14.00	139.00	182.00	0.29%
	Urea	44.80	34.24	1.00	35.00	242.00	0.33%
	White blood cells	13.10	9.35	0.00	11.20	462.60	0.32%

certainly not independent (patients are highly likely to occur in both samples). For these reasons, we do not attempt to perform any formal statistical test for informative missingness and simply show the previously mentioned informal assessment of informative missingness in Table 6.8. We also show the mutual information of both categories with respect to the target label, similar as in Che et al. (2018) (though they show correlation, which does not reflect non-linear relationships). Although the differences between the fractions of missing and not missing seem substantial for some variables (e.g. Arterial pH), they are clearly smaller than the standard deviations of the fractions. Furthermore, the mutual information of the 'missingness' of a variable with respect to the target label is rather small (i.e. below 0.001 for all variables), clearly smaller than the mutual information found for Strategy 3 in Table 6.5. In conclusion, we observe that it is not the case that the missingness of a variable is very informative of the target label value in our data. Recall that the way of determining this, however, is subjective as it does not involve a formal statistical test.

Another relevant set of descriptive statistics are those regarding the target label. Overall, 832,672 instances have a value of 0 for the target label, while the remaining 92,171 instances have a value of 1. Thus, 10% of the instances belong to the positive class, meaning that some class imbalance

Table 6.8: Statistics regarding informative missingness for subjective assessment.

Category	Variable	Fraction y=1, not missing (Std.)	Fraction y=1, missing (Std.)	Mutual information of missingness with y
Bedside Monitoring	CVP	0.1060 (0.3079)	0.0965 (0.2953)	0.0001
	Diastolic blood pressure	0.0997 (0.2996)	0.0946 (0.2926)	0.0000
	FiO2	0.1037 (0.3049)	0.0728 (0.2599)	0.0007
	Heart Rate	0.0998 (0.2997)	0.0930 (0.2904)	0.0000
	SpO2	0.0995 (0.2994)	0.1084 (0.3110)	0.0000
	Systolic blood pressure	0.0997 (0.2997)	0.0938 (0.2915)	0.0000
	Temperature	0.0994 (0.2992)	0.1149 (0.3189)	0.0000
Laboratory Testing	Arterial base excess	0.1030 (0.3039)	0.0621 (0.2413)	0.0008
	Arterial PCO2	0.1024 (0.3031)	0.0612 (0.2396)	0.0007
	Arterial pH	0.1020 (0.3027)	0.0604 (0.2383)	0.0006
	Arterial PO2	0.1024 (0.3031)	0.0614 (0.2400)	0.0007
	Bicarbonate	0.0997 (0.2996)	0.0863 (0.2808)	0.0000
	Bilirubin	0.1012 (0.3016)	0.0730 (0.2601)	0.0003
	Calcium	0.0997 (0.2996)	0.0963 (0.2950)	0.0000
	Creatinine	0.0997 (0.2996)	0.0773 (0.2671)	0.0000
	CRP	0.0973 (0.2964)	0.1001 (0.3001)	0.0000
	Glucose	0.0997 (0.2996)	0.0765 (0.2659)	0.0000
	GOT(ASAT)	0.1015 (0.3019)	0.0694 (0.2541)	0.0003
	GPT(ALAT)	0.1016 (0.3021)	0.0673 (0.2505)	0.0004
	Hematocrit	0.0997 (0.2996)	0.0825 (0.2752)	0.0000
	Platelets	0.0997 (0.2996)	0.0843 (0.2779)	0.0000
	Potassium	0.0997 (0.2996)	0.0835 (0.2766)	0.0000
	PTT	0.1002 (0.3003)	0.0656 (0.2476)	0.0001
	Sodium	0.0997 (0.2996)	0.0879 (0.2833)	0.0000
	Urea	0.0997 (0.2997)	0.0751 (0.2636)	0.0000
	White blood cells	0.0997 (0.2996)	0.0848 (0.2786)	0.0000

is present but the imbalance is not severe. Inspecting the data on patient-level, we observe that 1473 patients survive while 943 patients die, implying a mortality of 39%. This mortality rate corresponds to the findings by Singer et al. (2016), who estimated the mortality rate for septic shock to be approximately 40%.

Furthermore, it is important to realize that the number of instances per patient varies - statistics can be found in Table 6.9. On average, a single patient contains 383 instances in the data, but the number of instances per patient has strong right skew and a long right tail. While the median is 191 instances per patient, 155 patients can be found that each generate over 1,000 instances, with 5 patients even generating over 10,000 instances. This intricacy emerges from the fact that for some patients, heart rate seems to be measured once per minute for an extended period of time (e.g. >10 days), while for most patients, heart rate is measured only hourly. Recall from Section 4.2.3 that if left unaddressed, this feature of the data is likely to lead to supervised learning models that put a disproportionate focus on frequently-occurring patients. Also recall that we counter this issue by applying instance weights when training the model.

A final aspect that deserves mention is the realized split of the data in a training, validation and test set. Table 6.10 shows the main statistics regarding this split. Because we want to split in a patient-based manner and because the number of instances per patient is highly variable, achieving the desired split where exactly 70% of the instances is in the training set with 15% in both the validation and test sets would be a tedious process. Through trial and error, a split was achieved that approximately satisfies these percentages. To ensure that the label distribution

Table 6.9: Statistics of the number of generated instances per patient. Observe the rather high maximum compared to the rest of the data.

	Mean	**Standard Deviation**	**Minimum**	**First Quantile**	**Median**	**Third Quantile**	**Maximum**
Number of instances per patient	383	1,368	3	94	161	430	44,018

remains intact between the training, validation and test sets, this statistic is also shown in Table 6.10.

Table 6.10: Statistics regarding the splitting of the data in a training, validation and test set.

	Training	**Validation**	**Test**
Patients (% of total)	1,691 (70%)	362 (15%)	363 (15%)
Instances (% of total)	688,457 (74%)	117,419 (13%)	118,967 (13%)
% of instances in positive class	9.9%	10.3%	9.9%

6.5 Models

6.5.1 Model Tuning

Recall that for the PFS, the variables are first ranked based on mutual information with respect to the target label on the training set. This ranking is shown in Table 6.11. This table reveals that the mutual information gradually decreases when moving down the ranking, without any large gaps in mutual information between two consecutive features. The top 7 features are GOT(ASAT), PTT, GPT(ALAT), platelets, bilirubin, white blood cells and FiO2. Interestingly, the top 6 of these features originate from laboratory testing, and have quite some overlap with the SOFA criteria from Table 2.1 (bilirubin, FiO2 and platelets). The top 7 features are used in the remainder of the model tuning of the PFS.

Figure 6.3 and Table 6.12 jointly show the gist of the results from tuning the clustering algorithm, number of features and number of rules of the PFS. As the AUK is related to the ROC-AUC, the heatmaps with the AUK metric look extremely similar and are omitted for conciseness. Fitting one PFS to the data takes approximately 2 minutes on average, and fitting all models took 156 minutes in total. As was explained in Section 4.2.1, using fuzzy c-means clustering to initialize the PFS frequently results in an error in the gradient descent algorithm used to tune the centers, widths and conditional probabilities. These errors are shown as empty cells in the left heatmap of Figure 6.3. Recall that the cause of this error is usually two or more extremely close cluster centers. We observe that this occurs most frequently near the bottom-left corner of the left pane in Figure 6.3, i.e. with a relatively low number of rules and a relatively high number of features. We do not see a clear reason as to why this is the case. The error appeared in 19 of the 42 configurations with the fuzzy c-means clustering algorithm, corresponding to 45% of the models with this algorithm. Another observation is that, naturally, adding more features results in better model performance. However, changing the number of rules with the same number of features (moving horizontally in the heatmaps), generally does not make much of a difference with respect to the ROC-AUC. Finally, the best performing model is the configuration with K-Means clustering, 7 rules and 7 features, which is non-surprisingly the model with maximum representational power. Note, however, that the number of rules can be decreased to 4 with a negligible drop in performance (see Table 6.12). As this makes the model considerably simpler and easier to interpret while maintaining an extremely similar level of model performance, we consider the model with K-Means clustering, 7 features and 4 rules to be the best model and continue with this model in the remainder of the analyses.

Table 6.11: Ranking of the features based on mutual information with respect to the target label.

Feature	**Mutual Information**
GOT(ASAT)	0.0564
PTT	0.0546
GPT(ALAT)	0.0512
Platelets	0.0474
Bilirubin	0.0397
White blood cells	0.0359
FiO2	0.0293
Arterial pH	0.0281
Glucose	0.0281
Urea	0.0280
Arterial PO2	0.0259
Arterial base excess	0.0259
Bicarbonate	0.0242
Hematocrit	0.0214
SpO2	0.0211
Arterial PCO2	0.0206
Systolic blood pressure	0.0199
Potassium	0.0182
CRP	0.0181
Creatinine	0.0179
Calcium	0.0172
Sodium	0.0146
Temperature	0.0127
Heart rate	0.0113
Diastolic blood pressure	0.0083
CVP	0.0039

For the tuning of the densely connected neural networks, the full results of the model tuning (with a computation time budget of one hour) are shown in Appendix D.3. Within the fitted DNNs on all 26 available features, no clear trends between hyperparameter settings and performance seem to be visible. The ROC-AUC of the tested models on the validation set varies from 0.706 for the worst model to 0.7851 for the best model. The best model has 5 layers, with 35, 77, 221, 417 and 113 units in each layer, respectively (excluding the final output layer with 1 unit). The dropout rates between the layers are somewhat high: they are 0.2934, 0.1615, 0.1503 and 0.0216, respectively. When tuning the DNN on the feature set of the best PFS, a clear pattern is visible: the top of the ranking is filled with models with only 1 layer, indicating that the DNNs with stacked layers overfit and are unable to effectively engineer meaningful features in their hidden units. The best model has 127 units in its first layer, and achieves an ROC-AUC of 0.711 on the validation set, which is only very slightly higher than the best PFS. Fitting a DNN on all features takes 3.6 minutes on average, while fitting a DNN on the 7 features from the best PFS takes 2.2 minutes on average.

As for the tuning of the neural networks with GRU units, we found that fitting these models is rather expensive compared to DNNs: for GRU networks with only 2 layers it takes approximately 25 minutes to train the network with our early stopping criteria. Every network is only trained

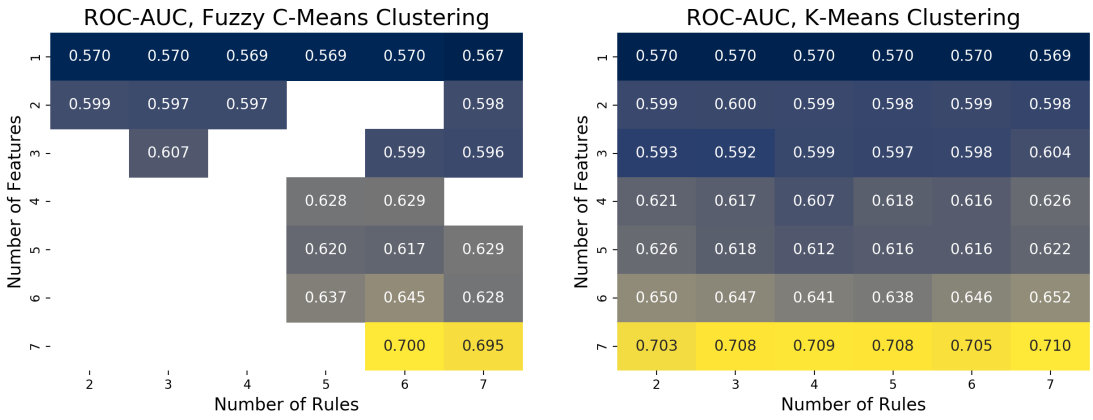

Figure 6.3: Heatmaps showing the ROC-AUC of the PFS with varying hyperparameters. In 45% of the cells for the variant with fuzzy c-means clustering (left), fitting the PFS results an error (these cells were left empty).

Table 6.12: Snippet of the top models from the tuning of the PFS.

Number of Features	Clustering Type	Number of Rules	ROC-AUC	AUK
7	K-Means	7	0.7095	0.1004
7	K-Means	4	0.7086	0.0989
7	K-Means	3	0.7081	0.0984
7	K-Means	5	0.7078	0.0981
7	K-Means	6	0.7054	0.0988
7	K-Means	2	0.7028	0.0974

for 2 epochs under these criteria, but does seem to converge as the training and validation loss hardly decrease between the end of the first epoch and the end of the second epoch. Fitting a GRU with a single layer is more than twice as fast, with a time of approximately 10 minutes to fit such a model to the data. Due to the computation budget of one hour, only a handful of GRU networks could be fit to the data. Appendix D.3 shows how each configuration performs. On all available features, the best model has a single layer with 16 units, achieving an ROC-AUC of 0.782 on the validation set. This is extremely close to the ROC-AUC of the DNN that uses all features, which is interesting because the GRU network has much fewer parameters and therefore much less representational power. On the feature set of the best PFS, the best GRU network achieves an ROC-AUC of 0.710 on the validation set with two layers containing 2 and 15 hidden units, respectively. The dropout rate between these layers is 0.03. This is, again, on par with the DNN on the same features.

6.5.2 Quantitative Assessment

Table 6.13 shows the performance of the final prediction models on the test set using the bootstrap sampling procedure described in Section 4.3. The PFS-ML is only slightly better than the PFS-CP - the difference in ROC-AUC between the models is nearly equal to the standard deviation of this metric. The gradient descent optimization thus only improved the model to a limited extent. This is also the case when this is assessed using the AUK (which also holds for all other observations in this section). Though both metrics illustrate that all trained models are clearly better than randomly guessing a class, they also show that considerable room for improvement remains. Regarding the neural networks, we find that, in line with intuition, using all available

features leads to a higher performance on the metrics than when only using the 7 features that were selected during the tuning of the PFS. On all features, the DNN and GRU network are about on par, though the performance of the DNN is about one standard deviation higher than that of the GRU. This makes the DNN the best-performing model, perhaps also because only a small GRU network could be trained with limited tuning. When using the 7 features that were eventually selected for the PFS, the performance of the GRU is on par with the PFS-CP, while the PFS-ML and the DNN seem on par at a level with a slightly higher performance. However, the DNN scores slightly better compared to the PFS-ML, but within one standard deviation of the performance of the PFS-ML. In general, the results presented in Table 6.13 thus point to the densely connected neural network using all available features as the winner with regard to the model assessment metrics. For conciseness, we will hereinafter refer to the DNN on all features as DNN-all, to the DNN on the 7 features of the PFS as DNN-7, to the GRU on all features as GRU-all and to the GRU on the 7 features of the PFS as GRU-7.

Table 6.13: Main results of the quantitative model assessment: the means and standard deviations of each model on the 500 bootstrapped test sets, each with 1000 samples.

	ROC-AUC (std)	**AUK (std)**
PFS-CP	0.6540 (0.0309)	0.0714 (0.0150)
PFS-ML	0.6910 (0.0315)	0.0937 (0.0156)
DNN (all features)	0.7838 (0.0260)	0.1394 (0.0152)
DNN (7 features)	0.7096 (0.0311)	0.1023 (0.0156)
GRU (all features)	0.7547 (0.0274)	0.1226 (0.0153)
GRU (7 features)	0.6505 (0.0320)	0.0712 (0.0153)

A natural follow-up question is to what extent that we can consider the performance levels of the various prediction models to be different from one another. Figure 6.4 shows boxplots of the ROC-AUC of all models in the bootstrapping procedure. The same plot with the AUK rather than the ROC-AUC is extremely similar, and is therefore left out for conciseness again. In Figure 6.4, the variability of model performance is clearly visible, though the previous conclusions regarding the order of model performances remains the same. To form a more objective comparison between the performance levels of the various models, we show the results of a Mann-Whitney U-test for each pair in Table 6.14. The vast majority of ROC-AUCs are different at an extremely high significance level, something which may not be directly clear from Figure 6.4 (compare e.g. the PFS-CP and PFS-ML). Only the GRU-7 and the PFS-ML seem to be rather similar, with a p-value of merely 0.0368. Regardless, it is relevant to keep in mind that a difference can be extremely significant yet very small - the magnitude of a difference and its statistical significance are two distinct concepts. Finally, we display the ROC and Cohen's kappa curves in Figure 6.5 to assess how the models perform across different thresholds. Apart from the GRU-7 and the PFS-CP, crossings of the curves are rare, indicating that the performance order of the models remains roughly intact across the majority of thresholds. The optimal threshold of the PFS-CP (i.e. threshold correspond to the maximum value of Cohen's Kappa) is remarkably low compared to the other models, which occurs because the PFS-CP does not predict any high mortality probabilities at all due to the absence of gradient descent optimization of the predictions. For more intuitive interpretation, we advise considering to rescale the outputs to a scale of 0 to 1 in model re-use.

Table 6.14: P-values of the Mann-whitney U test, allowing us to assess whether the significance level of the difference between the ROC-AUC scores of the models in the bootstrapping procedure.

	PFS-ML	**DNN (all features)**	**DNN (7 features)**	**GRU (all features)**	**GRU (7 features)**
PFS-CP	1.12×10^{-61}	4.98×10^{-164}	1.48×10^{-106}	6.99×10^{-161}	3.68×10^{-2}
PFS-ML	-	1.54×10^{-157}	1.08×10^{-20}	1.21×10^{-128}	1.42×10^{-69}
DNN (all features)	-	-	2.04×10^{-145}	1.11×10^{-53}	2.55×10^{-164}
DNN (7 features)	-	-	-	7.40×10^{-90}	1.53×10^{-111}
GRU (all features)	-	-	-	-	1.21×10^{-160}

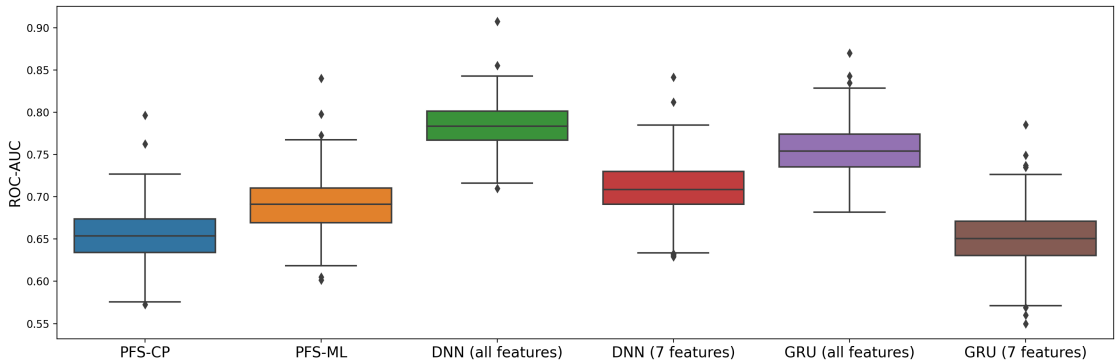

Figure 6.4: Boxplots showing the variability of the ROC-AUCs of the models side-by-side.

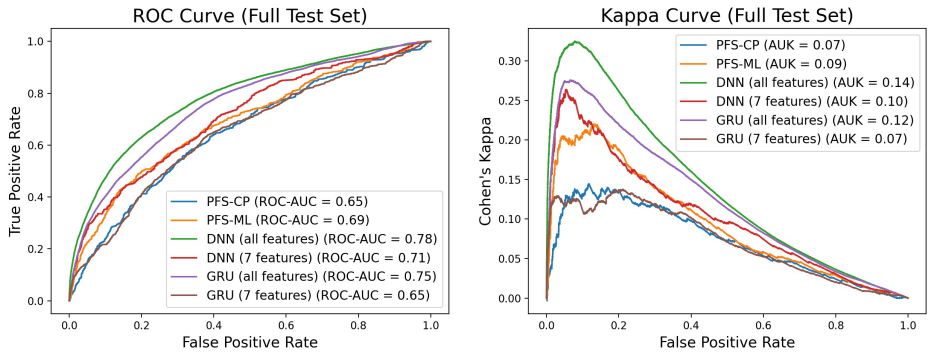

Figure 6.5: ROC and Cohen's kappa curves of the models on the full test set. The thresholds at the optimal operating point of the Cohen's kappa curves are 0.11, 0.81, 0.84, 0.90, 0.95 and 0.92, respectively (same order of models as in the figure).

6.5.3 Model Inquiry

Continuing the model assessment on the quantitative, performance-related criteria, we now turn to an in-depth investigation of the dynamics behind the predictions of the models (i.e. a more qualitative analysis). This includes a look into the internal representations of the PFS and a side-by-side comparison of the PFS and deep learning using global and local interpretability methods.

PFS

Recall that the predictions of a PFS are generated through an interplay of equations that ultimately rely on the centers, widths and conditional probabilities. These elements are visualized in Table 6.15 and 6.16. Note that these elements and those in the remainder of this section are exclusively for the PFS-ML. In Table 6.15, it can be seen that the centers v_j of most rules in most dimensions are somewhere between approximately 2 standard deviations below the mean to 3 standard deviations above the mean. Compared to Fialho et al. (2016), the centers are thus quite far from the means of the variables, indicating that the gaussian shape of the membership functions might be a troublesome to fit to our data - an aspect that could have been foreseen when inspecting Figure D.1 and D.2, as these tables already indicate that most of the features selected for the PFS are clearly non-normally distributed. Inspecting the rightmost column of Table 6.15, we observe that most centers lie on the outer percentiles of the features, once again indicating that the PFS is 'abusing' the gaussian shapes by only having one side of the gaussian in the range of the truly occurring values in the data. Furthermore, rule 3 has some distinctive characteristics compared to the other

rules. Its centers are extremely far from the means of GOT(ASAT) and GOT(ALAT), while its widths are very wide for all features (approximately 15 in normalized units). This indicates that this rule could be irrelevant because it is nearly always activated (for a majority of feature values). Regarding the conditional probabilities in Table 6.16, rule 2 is somewhat uninteresting because its conditional probability of mortality is very close to the prior probability of mortality over all instances in the test set (which, recall, was 0.1). Rule 1 indicates a lowered risk of mortality, while rule 3 indicates an extremely high mortality risk. Membership to rule 4 also points to a substantially increased mortality risk, though much less severe than rule 3.

Table 6.15: The centers and widths of each rule of the PFS-ML.

Feature	**Rule Number**	v_j	σ_j	v_j, **Original Scale**	σ_j, **Original Scale**	**Percentile corresponding to** v_j
Bilirubin	Rule 1	-0.58	2.77	-0.62	17.17	0.00
Bilirubin	Rule 2	-1.03	1.34	-3.02	9.58	0.00
Bilirubin	Rule 3	1.65	15.80	11.21	86.41	0.95
Bilirubin	Rule 4	1.91	3.35	12.58	20.27	0.95
FiO2	Rule 1	-1.14	1.40	31.66	73.28	0.03
FiO2	Rule 2	-1.10	3.48	32.31	107.24	0.03
FiO2	Rule 3	2.88	14.31	97.50	284.59	0.95
FiO2	Rule 4	-1.77	2.96	21.34	98.80	0.01
GOT(ASAT)	Rule 1	-0.92	1.05	-445.08	772.60	0.00
GOT(ASAT)	Rule 2	-1.00	1.15	-500.63	834.57	0.00
GOT(ASAT)	Rule 3	13.85	15.89	8,685.38	9,945.53	1.00
GOT(ASAT)	Rule 4	0.73	3.53	570.91	2,303.47	0.97
GPT(ALAT)	Rule 1	-0.43	2.79	-44.73	889.09	0.00
GPT(ALAT)	Rule 2	-0.60	2.31	-92.62	750.56	0.00
GPT(ALAT)	Rule 3	11.86	16.09	3,519.15	4,744.67	1.00
GPT(ALAT)	Rule 4	3.19	4.71	1,006.15	1,445.76	0.99
PTT	Rule 1	-1.24	1.04	15.99	62.34	0.00
PTT	Rule 2	1.13	3.62	64.24	114.85	0.90
PTT	Rule 3	2.08	14.92	83.58	344.79	0.96
PTT	Rule 4	-1.59	2.00	8.87	81.96	0.00
Platelets	Rule 1	0.56	3.25	317.78	747.69	0.75
Platelets	Rule 2	1.86	2.05	525.05	556.58	0.95
Platelets	Rule 3	-1.96	14.58	-84.43	2,558.35	0.00
Platelets	Rule 4	0.15	3.06	252.35	718.25	0.63
White blood cells	Rule 1	-0.32	2.29	9.88	33.29	0.44
White blood cells	Rule 2	-0.12	0.98	11.66	21.50	0.55
White blood cells	Rule 3	1.00	15.66	21.74	152.85	0.87
White blood cells	Rule 4	1.45	4.55	25.72	53.48	0.92

To complement the numeric representation of the parameters of the PFS, we show plots of the membership functions of all rules for each feature in Figure 6.6. This immediately reveals various relevant aspects of the PFS. Firstly, we observe that the widths of all rules are rather wide, as hardly any rule ever touches low membership values (e.g. below 0.2) in the plots. Secondly, the plots confirm our previously formed hypothesis that rule 3 could be irrelevant because it has a membership value near 1 at the majority of feature values for all features except GOT(ASAT) and GPT(ALAT). Speaking of these last two features, we observe that the centers of the membership functions of rule 1 and 2 for these features is below 0, which does not occur in any instance in the

Table 6.16: Conditional probabilities of mortality (given a rule) in the final PFS.

	Class 0 (Survives next 72h)	**Class 1 (Mortality within 72h)**
Rule 1	0.95	0.05
Rule 2	0.89	0.11
Rule 3	0.04	0.96
Rule 4	0.54	0.46

data at all: recall from Table 6.7 that the minima for these features are 3.00 and 0.00, respectively. Inspecting the distributions of these features in Figure D.1, it appears these features typically have low values and possess a long right-tail. The PFS seems to have attempted to model this using the right tail of the gaussian membership functions. Similar behavior is present for white blood cells. Finally, rule 1 and 2 show strong resemblance on all features except PTT, indicating that these rules could be redundant, especially because they have quite similar conditional probabilities.

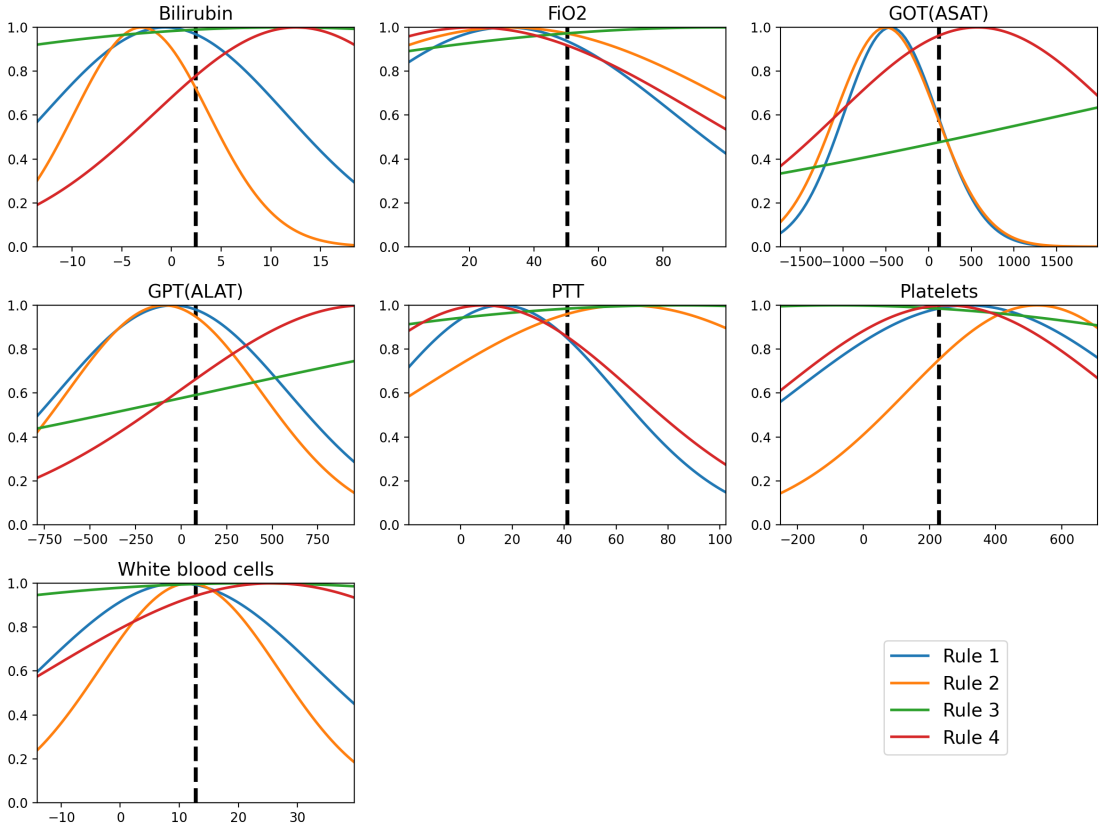

Figure 6.6: Visualization of the rules of the PFS for each feature. The dotted line indicates the mean value of the feature, and the x-axis ranges up to 3 standard deviations away from the mean.

To assess redundancy and irrelevance in a more formal fashion, Table 6.17 shows the values for the similarity criterion from Equation 3.28 for each pair of rules, and for each rule with the universal set. Interestingly, this table contradicts our previous intuition that rule 1 and 2 might be redundant - rule 1 and 4, and rule 3 and 4, show more similarity with one another than rule 1 and 2. Explanations for this finding could be that most features have a long right tail in their distribution and that rule 4 is usually highly activated (which by definition generates strong overlap

with many other rules). As Fialho et al. (2016) consider rules redundant when their similarity is greater than 0.4, rule 1, 3 and 4 would have been merged if we would have taken their approach to fitting a PFS. Furthermore, all rules have relatively low similarity with the universal set, indicating they can be considered somewhat irrelevant because the activation of their membership functions is relatively sparse. Rule 2 could be considered on the verge of irrelevant because is seems to be hardly ever activated. To get an even more in-depth view of how the activation of the rules varies over the test set, Table 6.18 shows statistics regarding the rule activation over the instances in the test set. It reveals that in contrast to our intuition about rule 3, it is still quite rare for this rule to be highly activated as activations of over 0.25 only appear beyond the 75-percentile. Also, the median activation of rule 1 and 2 are only 0.06 and 0.04, respectively, thus it is quite rare for these rules to be strongly activated.

Table 6.17: Jaccard similarities between the rules of the PFS-ML and similarity of each rule with the universal set.

	Rule 2	**Rule 3**	**Rule 4**	**Universal Set**
Rule 1	0.38	0.33	0.54	0.10
Rule 2	-	0.27	0.33	0.07
Rule 3	-	-	0.51	0.25
Rule 4	-	-	-	0.13

Table 6.18: Univariate statistics on the rule activations of the PFS-ML in the test set.

	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**	**Universal Set**
Mean	0.10	0.07	0.25	0.13	1.00
Std	0.10	0.08	0.06	0.08	0.00
Minimum	0.00	0.00	0.14	0.00	1.00
25% Percentile	0.00	0.01	0.23	0.07	1.00
50% Percentile	0.06	0.04	0.24	0.14	1.00
75% Percentile	0.16	0.11	0.25	0.19	1.00
Maximum	0.54	0.43	0.96	0.54	1.00

It is also relevant to relate the internal mechanisms of the PFS to the sepsis-3 definition - is it in line with what we know about sepsis? The SOFA criteria from Table 2.1 indicate a worsening patient condition from left to right, and some variables occurring in this score also occur in the PFS. Firstly, an increasing fraction of inspired oxygen (FiO2) relates to a higher SOFA score, and should thus relate to a higher predicted mortality. Inspecting the rule plot of FiO2 (Figure 6.6), high fractions of FiO2 result in high membership to rule 3, which (by Table 6.16) means an increased mortality risk. This is in line with clinical intuition, as patients requiring the inspiration of highly oxygenated air are likely to be in critical condition. We observe a similar effect in the plot of bilirubin, although in this case membership to rule 4 increases at high values of bilirubin as well. Increasing bilirubin thereby increases the mortality risk to a lesser extent than FiO2, as the conditional probability of mortality is lower for rule 4 than for rule 3. Regardless, increased mortality risk with higher bilirubin compared to the baseline risk is again in line with the SOFA criteria. Finally, platelets are also a common factor in the SOFA criteria and the PFS. The increased mortality rate with decreasing platelets applies to the rule plot of platelets to a limited extent. Though indeed, membership to rule 3 increases with decreasing platelets, and membership to the other rules decreases, the membership to rule 1 and 4 is still quite high at low values of platelets considering that the minimum possible value of platelets is 0. When platelets are at their lowest levels, the membership to rule 1 and 4 is still near 0.8. Thus, the rule plots of bilirubin and FiO2 are quite well in line with the SOFA criteria, while the rule plot of platelets is less conforming.

In conclusion, the final fit of the PFS to the data contains both desirable and non-desirable be-

havior. Desirably, the PFS does reflect various elements of the SOFA criteria, thus its internal representations are in line with clinical intuitions to a considerable extent. Undesirably, the gaussian membership function shapes are not in line with the distributions of the selected features, most rules are only seldomly activated and some rules have a high similarity, making them somewhat redundant.

Comparative Global Inquiry

Having performed a model inspection specifically for the PFS, we now turn to a side-by-side comparison of PFS and deep learning, including an inspection of both global model behavior (using ALE plots and permutation feature importance) and local model behavior (via a case study of model predictions over time with LIME explanations for two selected instances).

Firstly, the general relationship between feature values and model predictions is inspected through a series of ALE plots which can be seen in Figure 6.7, 6.8 and 6.9. Recall that the GRU networks are not included in these plots because of the difficulties they raise due to the explicit inclusion of feature values at different timesteps in these models. Furthermore, the ALEs for the outermost bins were excluded in the plots to avoid them from having many very close points in normal variable ranges and one or two distant points with large interpolations in between. Comparing the models side-by-side on the features selected for the PFS (Figure 6.7), we firstly observe that the curve of the PFS-CP is remarkably flat across all features. As mentioned earlier, the PFS-CP always predicts low mortality probabilities (between 0.093 and 0.129), which is caused by the absence gradient descent in the training - the output probabilities are not calibrated to be in a range between 0 and 1. This then leads to extremely small ALEs for the PFS-CP and fairly horizontal curves in Figure 6.7. As the PFS-ML does not suffer from this issue and was the better model in the quantitative assessment (compared to the PFS-CP), we do not consider the PFS-CP any further in our discussion of the ALE plots. For PTT and FiO2, the 3 models (PFS-ML, DNN-7 and DNN-all) behave quite similar and in line with general intuition: a higher FiO2 generally means that the patient has more difficulty breathing and thus a higher mortality risk, while a higher PTT (partial thromboplastin time) implies slower coagulation and a more severely ill patient (parallelling that decreasing platelets increase SOFA score according to Table 2.1). Across these 2 features, the predictions of the DNN-7 and PFS-ML are generally more strongly influenced by changing values (the lines of these models are generally at the bottom or top in Figure 6.7), an aspect that also seems quite well preserved at the other features. This matches intuition as many other features also contribute to predictions for the DNN-all, meaning that it should generally be affected less heavily by the value of a single feature. Another general observation regarding the DNN-7 in Figure 6.7 is that it possesses some behavior which is not in line with the other models, and which has no clear explanation from a clinical perspective to the best of our (limited) medical knowledge. Firstly, mortality risk seems to decrease significantly for the DNN-7 with increasing GPT(ALAT), while in fact, increasing GPT(ALAT) is a somewhat vague, non-specific indication of possible liver malfunction. The behavior of the other models does thus make more sense. Secondly, increasing bilirubin weirdly first increases mortality risk slightly, then decreases risk, and finally increases risk again - this is odd behavior given that we know from the SOFA criteria in Table 2.1 that increasing bilirubin should indicate increased mortality risk. The DNN-all does not seem to suffer as much from such extreme, quirky and non-justifiable behavior for any of the features. The PFS-ML generally shows stable, reasonable behavior except for platelets, where the predicted mortality risk weirdly starts to increase strongly again when the value of this feature increases past high levels. This clearly contradicts the SOFA criteria, where decreasing platelets indicates increased mortality risk. To conclude the observations in Figure 6.7, the behavior of the PFS-CP cannot be assessed with the figure due to its ubiquitously low predictions, the PFS-ML generally acts quite in line with clinical intuitions though not for one feature, the DNN-7 shows quirky global behavior for two features and the DNN-all possesses stable, clinically justifiable global behavior.

For further validation of the DNN-all, the ALE plots of the remaining features (i.e. those excluding

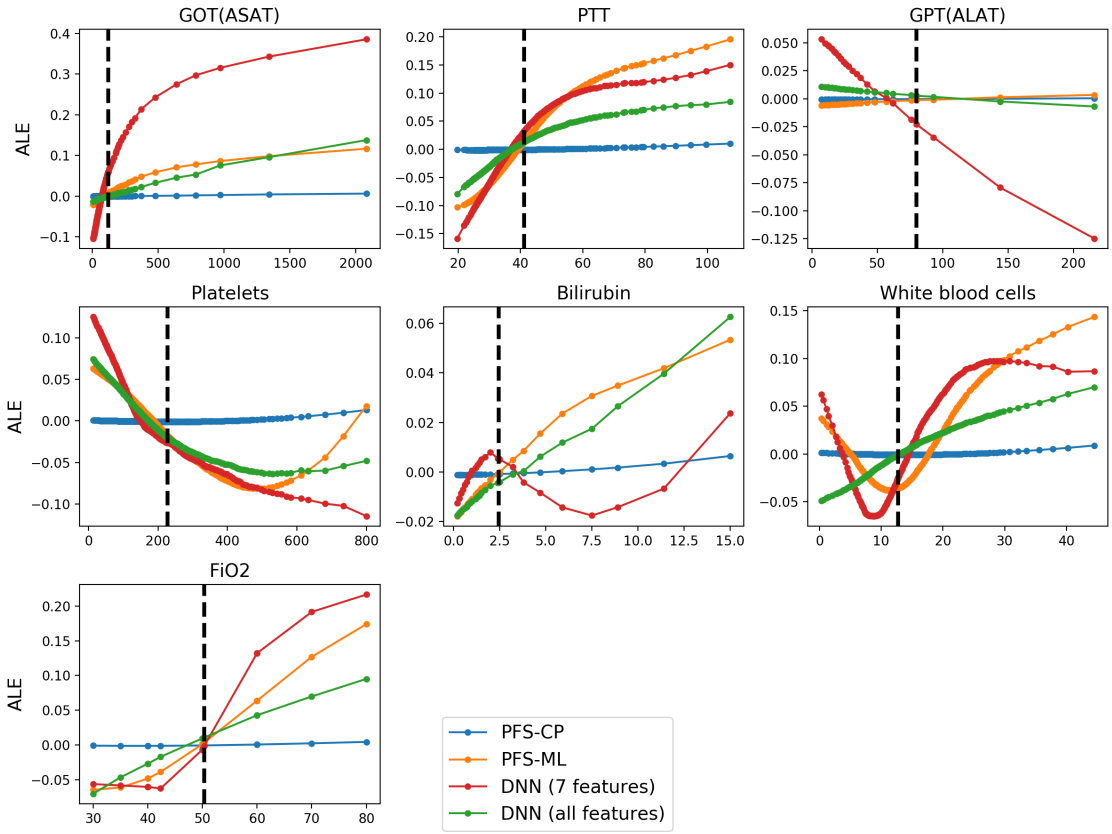

Figure 6.7: Accumulated local effects plots of the PFS-ML, DNN-7 and the DNN-all. Like in the plot of the rules of the PFS, the dotted line indicates the mean of the feature. Recall that the interpretation of an ALE is the difference to the mean prediction, i.e. how much a prediction is higher or lower than normal for that particular feature value.

the 7 features selected for the PFS) are shown in Figure 6.8 and 6.9. Because various variables have no clear direct relationship to sepsis severity and/or mortality risk when examined in the light of SIRS, SOFA and the sepsis-3 definition, our discussion will be restricted to a subset of the plots shown in the figures. Firstly, we know from qSOFA (Singer et al., 2016) that decreased blood pressure is an indicator of possible sepsis, and that therefore, lower blood pressure should generally indicate increased sepsis severity and thus increased mortality risk. This is reflected in the ALE plots of diastolic and systolic blood pressure, as both variables clearly involve a negative relationship with mortality risk. The global behavior of creatinine, on the other hand, largely contradicts the SOFA criteria, in which higher creatinine levels indicate a higher score and greater mortality risk. In the ALE plot of creatinine, we only observe a slight increase in mortality risk at moderate levels, while the predicted mortality risk strangely decreases when creatinine moves to higher levels. For more means to validate model behavior, we can also contrast the ALE plots to the SIRS criteria (Singer et al., 2016). In SIRS, a heart rate above 90 beats per minute is a criterion for possible sepsis, and the ALE plot of heart rate clearly reflects this statement as the predicted mortality probability ramps up greatly when heart rate increases to levels above 90. As for temperature, values below 36 °C or above 38 °C are a criterion for SIRS, and the ALE plot of temperature does indicate an increased mortality risk with low temperature but not with high temperature - meaning that this aspect of SIRS is only reflected partially. A final straightforward intuition that we can use to validate model behavior is that decreased peripheral oxygen saturation generally means that the body is fighting to deliver sufficient oxygen to its organs, meaning that

a decreasing SpO2 should indicate an increasing mortality risk. This behavior is reflected in the ALE plot of SpO2, where the predicted mortality probability shows a strong increase as SpO2 moves down. In conclusion, it seems that the global behavior of the DNN-all is largely in line with SIRS, SOFA and qSOFA, with suspicious behavior on but a few of its variables.

To test whether the ALE plots look differently when the size of the bins for the estimation of the curve is altered, a full grid with various values for this parameter was evaluated. This was done for the ALE plot of the DNN-7 on the white blood cells feature. The result can be seen in Appendix D.4. The parameter only seems to have effect on the resolution of the ALE plot, and the curves look extremely similar with all tested values. This is probably because of the accumulation of the local effects from left to right in the plots. It thus seems unlikely that this parameter could make the ALE plots look extremely different when altered. Furthermore, we would like to emphasize that our approach has the limitation that it does not reveal anything regarding interaction effects between features, which is a serious limitation as it is highly likely that such effects do occur in reality. Considering the fairly sizable amount of available variables in the models of this research, an inspection of interaction effects would be challenging because many feature combinations are possible and because three-or-more-way interactions could be present as well. We thus leave an exploration of model behavior with regard to feature interaction effects for future research. The same holds for interaction effects with regard to feature permutation importance. We will discuss our analysis of single variable feature permutation importances hereafter.

To investigate how model performance is affected by the 'destruction' of individual features, bar plots are shown of the feature permutation importances of all models in Figure 6.10 and 6.11. Recall that in this research, feature permutation importance denotes the decrease in model ROC-AUC when a particular feature is randomly permuted on the test set. To highlight the main results in Figure 6.10, we first observe that all models suffer strongly from permuting FiO2. As the inspiration of oxygen in a patient is, intuitionally, a strong indicator of increased mortality risk, it is not surprising that the predictions of all models are relatively strongly influenced by this feature. Furthermore, the DNN-7 shows a remarkably strong sensitivity to the destruction of GOT(ASAT) and GPT(ALAT), which is in line with the previously observed large magnitude of the values in the ALE plots in Figure 6.7. In general, the permutation feature importances for the DNN-7 are on the higher end of the spectrum across the 7 features, indicating that this model seems less robust to inaccurate or 'broken' feature values than the other models. Thus, once again, the DNN-7 shows signs of overfitting as it seems to capture some noise in the data, leading to turbulent predictions. Another noteworthy observation when inspecting Figure 6.10 is that the feature permutation importance of platelets for the GRU-7 is negative, meaning that the permutation of this feature actually increases the ROC-AUC of this model! It thus seems that the GRU-7 has not successfully picked up patterns from platelets during training that generalize to the test set. Regardless, the GRU-7 has the lowest feature permutation importance for the vast majority of features, implying that it interestingly seems most robust to inaccurate or 'destroyed' feature values. Likely, this is because the GRU-7 is aware of the 'history' of a specific patient (i.e. previous instances) via its internal state, which is a capability that the PFS and DNN models do not posses because they only take the patient attributes at the current timestep directly into account when predicting.

Another interesting global aspect of the models is the number of learned parameters. The PFS-CP and PFS-ML each have only 64 learned parameters (7x4=28 parameters for the widths, also 28 for the centers, and 4x2=8 for the conditional probabilities). 64 learned parameters is extremely small compared to the deep learning models: the DNN-7 has 1,144 learned parameters, the DNN-all has 160,877, the GRU-7 has 937 and the GRU-all has 2,129. From this point of view, it is thus impressive that the PFS-ML manages to achieve performance levels on the ROC-AUC similar to the DNN-7 while it has much, much fewer parameters. In that regard, the PFS models can be considered more efficient than the deep learning models.

As for the feature permutation importances for the DNN-all and GRU-all in Figure 6.11, we firstly observe that the magnitudes are noticeably smaller than in Figure 6.10: all effects in the

former have magnitudes smaller than approximately 0.017, while this is 0.057 in the latter. This effect likely originates from the fact that the DNN-all and GRU-all base their predictions on more features, which goes accompanied by a reduced influence of a single feature on the predictions. Another observation is that again, we observe that the feature permutation importances of the GRU are generally lower than the DNN, reflecting that the internal state of the GRU and the inclusion of past timesteps in the predictions make this model more stable to unreliable feature values. Zooming in on particular features, we see that in both models, SpO2, systolic blood pressure and urea have high permutation importances. It is interesting that these features prove to be among the most important ones for the GRU-all and DNN-all, while they were not selected through the variable ranking based on mutual information. Regardless, the high permutation importance of these features is justifiable from a clinical perspective as a low SpO2 is a clear hint of a 'fighting' body and systolic blood pressure is in the qSOFA criteria. Urea is particularly interesting, as this variable is a general indicator for kidney dysfunction, and its effect was apparently sizable in the ALE plot already. Finally, various negative permutation feature importances can be spotted in Figure 6.11, once again indicating that the neural networks probably did not succeed to learn generalizable patterns from various features. Particularly for the GRU-all, a sizable number of features (7 of the 26) have a negative permutation importance. Regardless, we emphasize once again that feature interactions are not included in our analysis, and that it could be the case that the features with negative permutation feature importances are still valuable in combination with particular other features. We leave an investigation into such interaction effects for future research.

Local Inquiry

Having inquired global model behavior, we now turn to local model behavior. Recall that to this end, we select one patient and inspect the mortality risk predictions for this patient over time, accompanied by two LIME explanations for two instances. This approach resembles that of a case study. Figure 6.12 shows the mortality probability predictions over time for the 6 models for a selected patient. Note that we hand-picked one particular prototypical patient here, and that many other patient could have been selected as the test set includes 363 patients. We refer the reader to Appendix D.5 for a figure of another patient and for which the models perform less well. Also, we encourage the reader to interact with our code to explore model behavior for other patients. More details regarding the implementation can be found in Appendix C. Returning to the predictions shown in Figure 6.12, we firstly observe that all models predict a high mortality probability at the start (i.e. at ICU admission), and quite successfully predict a high mortality prediction towards the end as well, which is correct as this patient eventually died. Furthermore, we can once again observe that the PFS-CP generally predicts quite low probabilities again, but starts to predict above its 'optimal' threshold near the end. Thus, the small predicted probabilities need not be an issue and can be rescaled to have a minimum of 0 and a maximum of 1 for more convenient interpretation. Another interesting pattern in Figure 6.12 is that the predictions are somewhat 'shaky' in general, particularly for the DNN-all. On the models with 7 features, various plateaus are visible, which is because no features change for various instances with respect to their predecessor. This is because predictions are made at every timestep regardless of which variable was changed at the event and regardless of whether the event corresponded to a variable of one of the 7 features. Also, it seems that the GRU networks are a bit more 'smooth' over time, probably caused by the hidden state of the GRU units upon which their predictions are based. This is probably desirable behavior, as shaky predictions would likely spark mistrust in a model when used in a real ICU setting.

Now, to investigate what factors contributed to a particular prediction for a selection of instances, we inspect explanations generated by LIME. The two selected instances are indicated by black circles in Figure 6.12. Recall that generating LIME explanations for the GRU networks is extremely challenging due to the fact that these networks take the feature values at particular timesteps as input explicitly, rather than just the feature values at a particular point in time (as

the other models do). Thus, LIME explanations are only provided for the PFS-CP, PFS-ML, DNN-all and DNN-7. For the leftmost and rightmost instance, the LIME explanations are shown in Figure 6.13 and 6.14, respectively. Firstly, it can be seen that the fidelity of the DNN-7 is quite reasonable, but that the fidelity of the other models is moderate (PFS-ML) to low (PFS-CP and DNN-all). We must thus keep in mind that most explanations match the actual model behavior in the local area only to a limited extent. Regardless, Figure 6.13 reveals that GOT(ASAT) is an important variable for the predictions of the PFS-CP, PFS-ML and DNN-7, matching the global behavior as earlier evaluated with the ALE plots. White blood cells and platelets, on the other hand, do not seem to play an important role in the predictions of any model. Also, note how the explanation for the prediction of the DNN-all is quite different from the other models (urea, systolic blood pressure and heart rate play an important role in the prediction), while still showing some similarities (e.g. the high importance of PTT). Finally, the DNN-7 once again shows some behavior that contradicts clinical intuition, as GPT(ALAT) has a negative coefficient in the explanation, implying that increasing GPT(ALAT) would weirdly mean decreasing mortality risk. Regardless, it seems that overall, the local explanations for the models match global model behavior quite well, rendering the first instance a prototypical example. For the second instance, the explanation stays roughly the same, although GPT(ALAT) now has a large positive coefficient for the DNN-7 and although GOT(ASAT) and FiO2 have now become important determinants for the predictions of the DNN-all. Also, the fidelity of the explanation for the PFS-CP is now high, while the other models have moderate fidelities. The explanation for the second inspected instance is thus generally more in line with actual local model behavior than the explanation for the first instance.

Finally, we briefly investigate the influence of the kernel width of LIME on an explanation to get a grasp of how sensitive the explanation could be to this parameter. To this end, the explanation of the DNN-all on the first instance in the case study was chosen because this explanation has quite low fidelity, and we deem it likely that the explanation could vary greatly because of the many features of this model. The results of this experiment are shown in Figure 6.15. A very small kernel width leads to all points having the same prediction of the DNN-all, which then leads to a LIME explanation with only coefficients of 0 and a perfect fidelity. With bigger although still small kernel widths, all generated points are very close to the explained instance, leading to all of them being assigned similar weights through the euclidean distance metric, and subsequently to a linear model with extremely low fidelity. It seems to become impossible to approximate local model behavior properly if we only sample instances very close the chosen instance for the explanation. With kernel widths bigger than 1.5, the explanation stabilizes and has similar, acceptable levels of fidelity, again probably because of the euclidean distance function used to determine the weights. If only a handful of instances has a high weight, then it becomes possible to approximate the prediction model with a linear model for the explanation. In conclusion, it seems that explanations are relatively stable with regard to the kernel width as long as it is sufficiently large, and the default (5.099) is more than wide enough for our data.

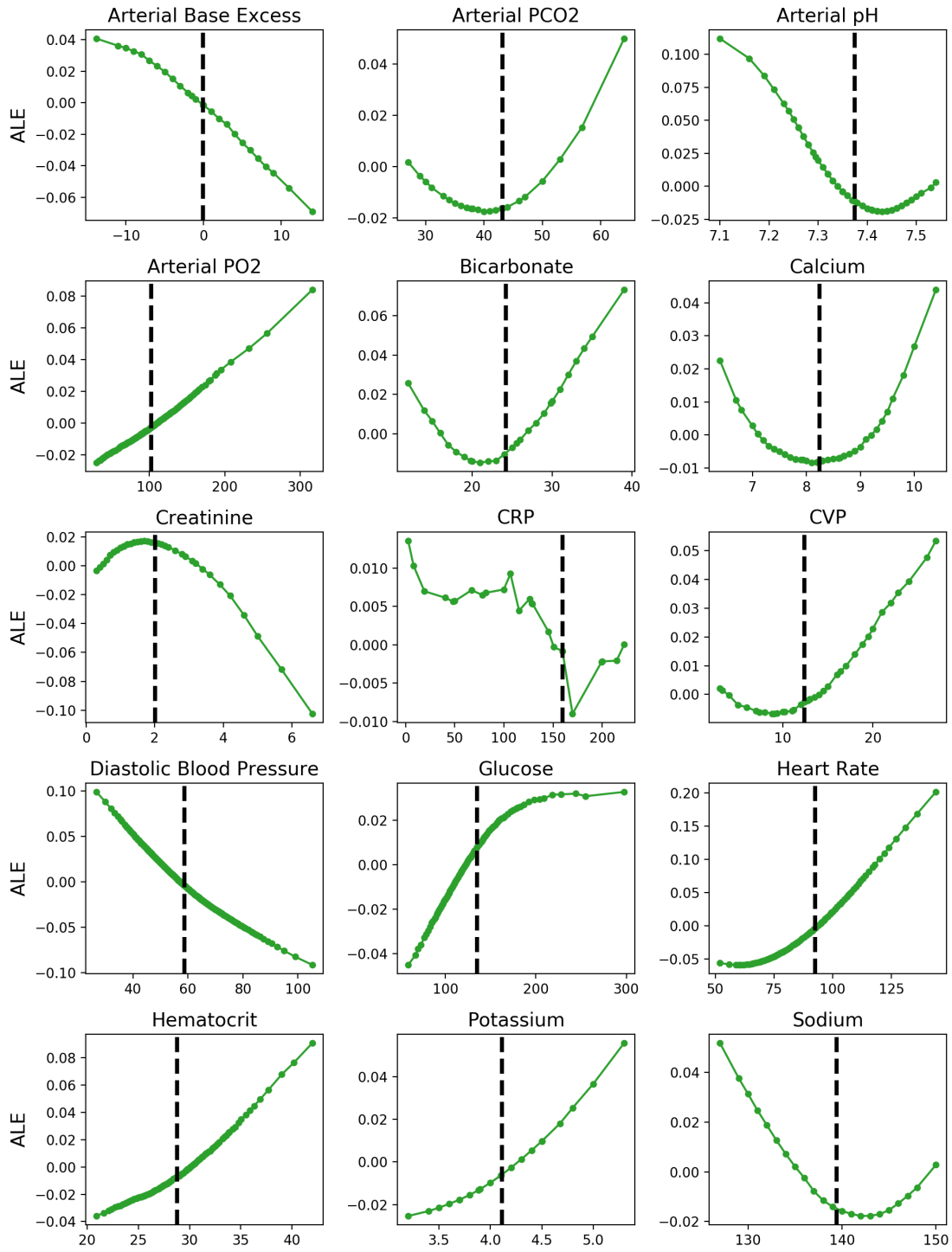

Figure 6.8: First part of the accumulated local effects plots of the remaining feature in the DNN on all features.

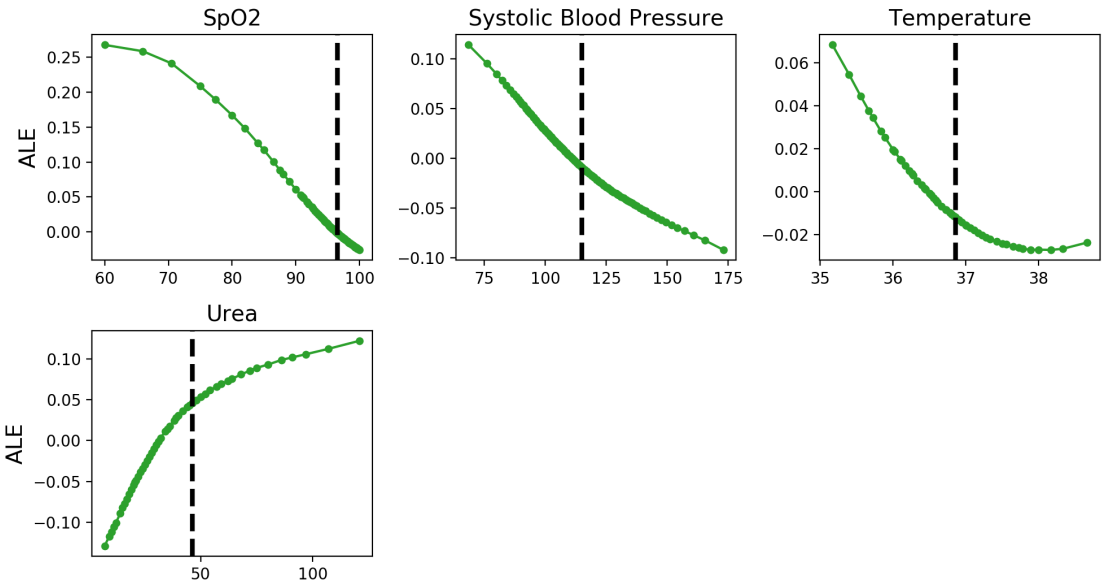

Figure 6.9: Second part of the accumulated local effects plots of the remaining feature in the DNN on all features.

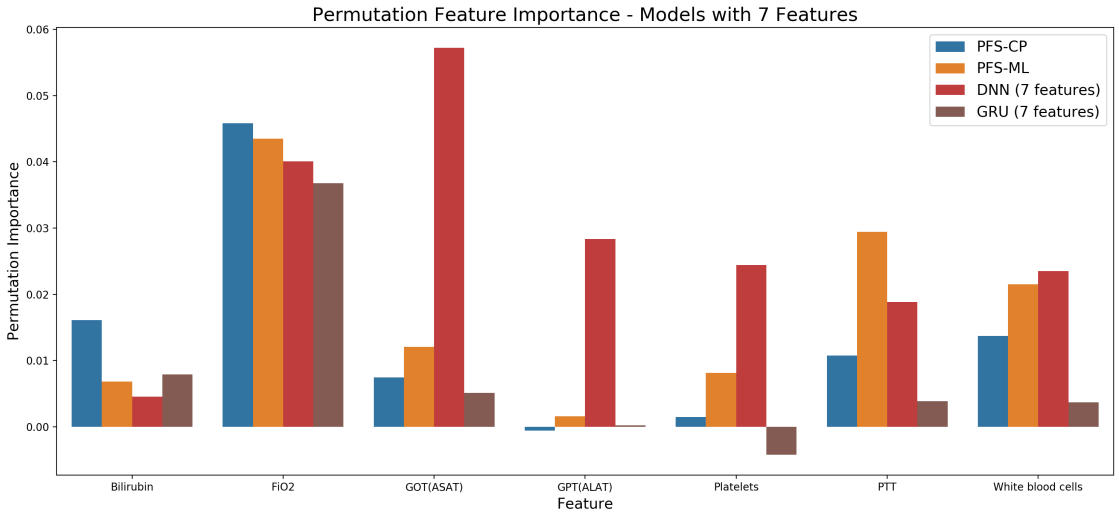

Figure 6.10: Permutation feature importances in the models with the 7 features of the PFS. The permutation feature importance is the decrease in ROC-AUC after permutation of the particular feature.

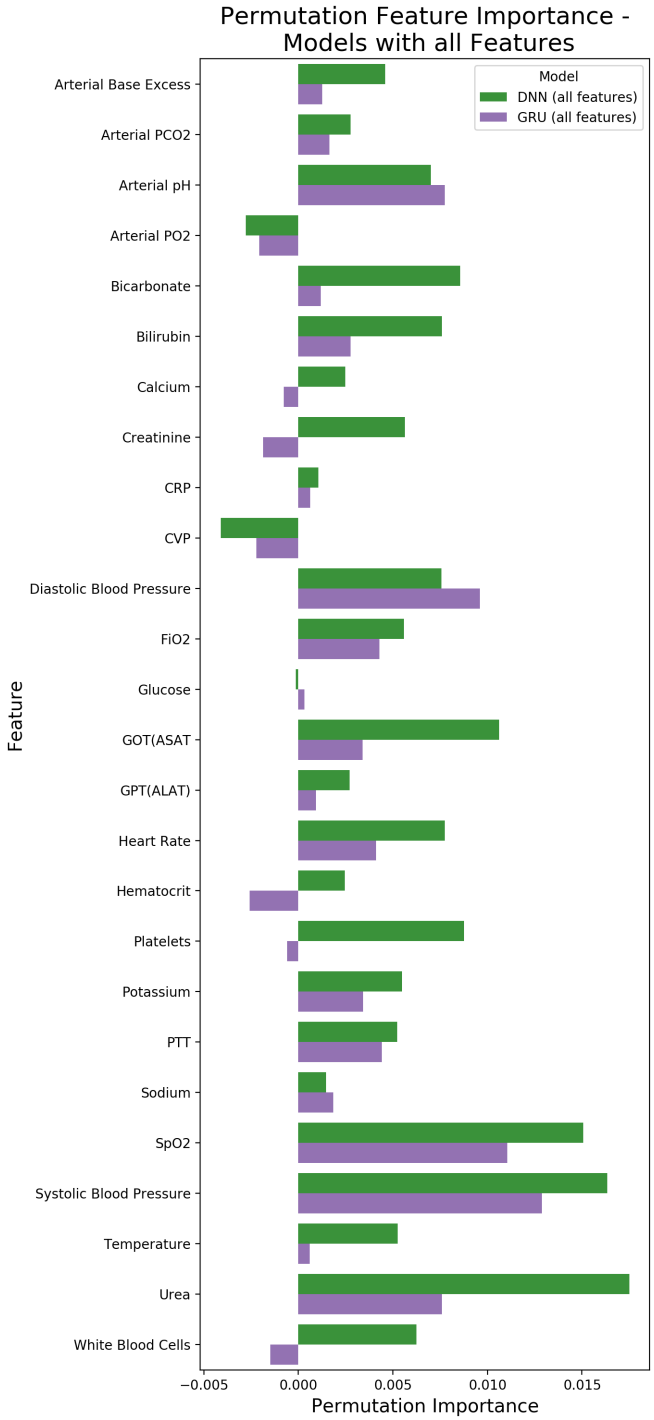

Figure 6.11: Permutation feature importances in the models with all available features.

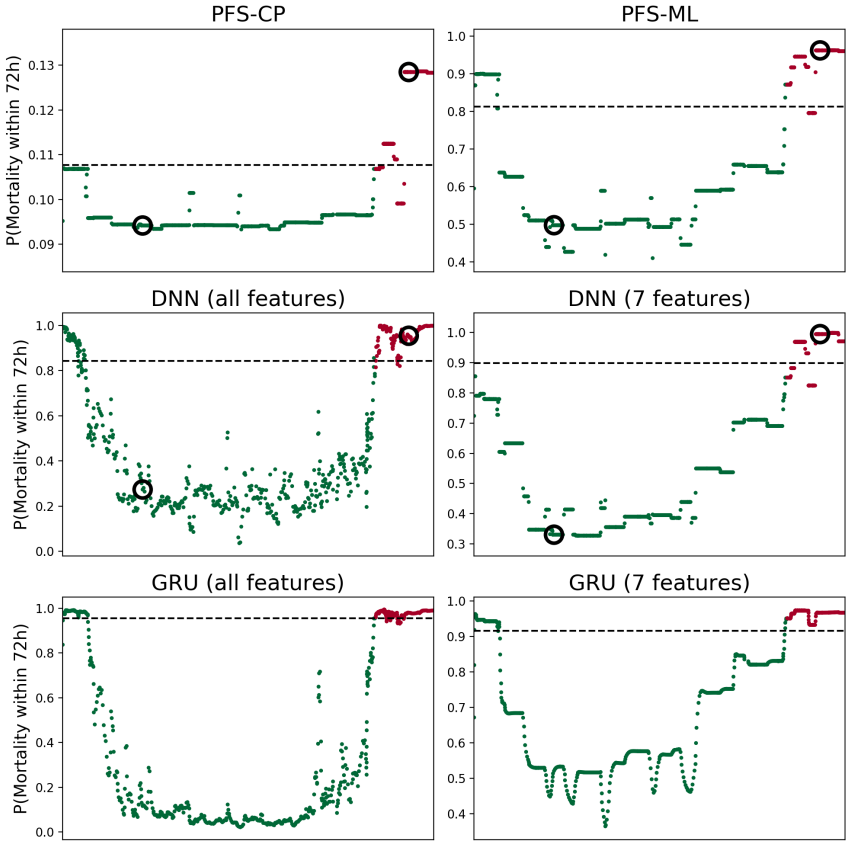

Figure 6.12: Predictions of the models over time for a selected patient. A red color indicates patient mortality within 72 hours, and the dotted line indicates the optimal threshold from the cohen's kappa curves. The circled instances are explained later using LIME in Figure 6.14 and 6.14.

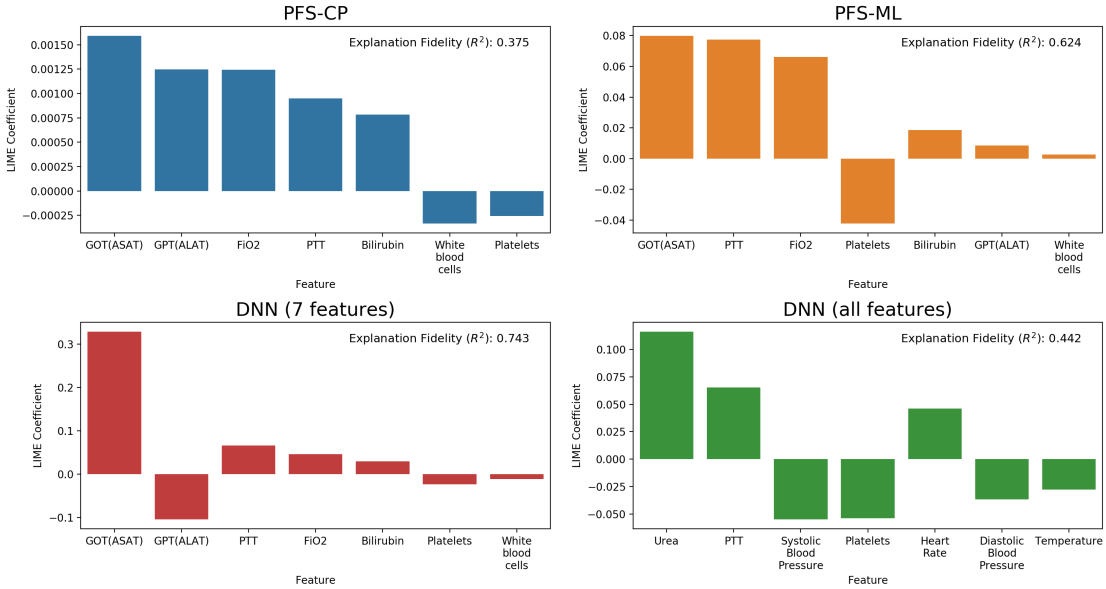

Figure 6.13: LIME explanations for the left instance from Figure 6.12.

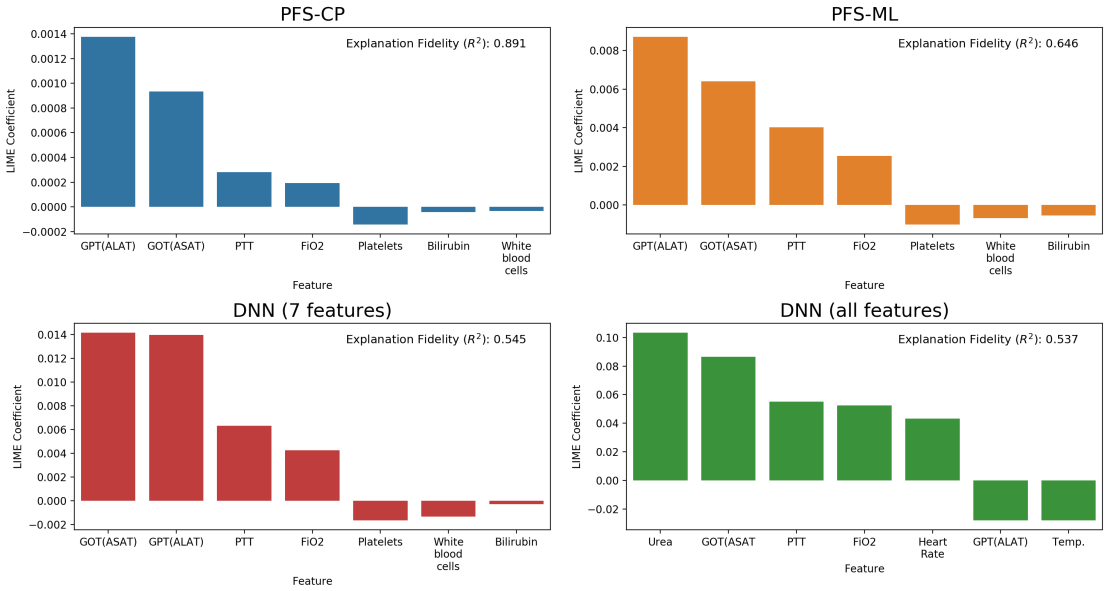

Figure 6.14: LIME explanations for the right instance from Figure 6.12.

Figure 6.15: Investigation of the behavior of LIME explanations with varying kernel widths.

Chapter 7

Conclusions

The key objective of this research was to investigate how deep learning compares to probabilistic fuzzy systems for septic shock mortality prediction. To this end, we have set out to answer four research questions:

1. *How is sepsis and septic shock defined, and how is a patient's mortality rate predicted traditionally?*

Sepsis is the formalization of a condition called blood poisoning and is officially defined as a life-threatening organ dysfunction caused by a dysregulated host response to infection. Septic shock is the aggravation of sepsis, in which circulatory and cellular metabolism dysfunctions are so severe that they pose a very high mortality risk to the patient. Traditionally, patient mortality risk is assessed with severity scoring systems such as APACHE and SAPS, which involve measuring several basic physiological variables upon ICU admission, looking up the corresponding number of points per variable value in a table, and summing the points into a score which is linked to a particular mortality risk. These systems cannot leverage non-linear or temporal patterns in physiological variables and can only be used for a one-time mortality risk assessment upon ICU admission.

2. *What are the foundations of probabilistic fuzzy systems and deep learning?*

A probabilistic fuzzy system converts crisp inputs into fuzzy input sets, applies fuzzy rules to infer a fuzzy output set, and consequently processes the output into a probability. Deep learning leverages layers of stacked perceptrons, or other units such as the gated recurrent unit, to learn internal representations of a dataset that enable it to predict one or more variables. Deep learning is characterized by high representational power, many parameters and hyperparameters, high computational cost, high flexibility to deal with different types of data, and finally a strong tendency to overfit the training data.

3. *How can events recorded at uneven and misaligned timescales be transformed as to allow predictive modeling, and how should the resulting dataset be split to avoid leakage between the training and testing set?*

Events recorded at uneven and misaligned timescales can be converted through templating or by simply creating an instance at each event occurrence. Generally, both approaches result in irregular and sporadic multivariate time series, which require some filling strategy to address the extreme sparsity of the variables. In our experiments, full forward filling was the best strategy to address this sparsity. For model assessment without leakage, events recorded over time that belong to a case must be split using either a time-based or case-based splitting strategy. According to our experiments, following the standard practice of observation-based random splitting on such

data leads to severe leakage between the training and test sets, with severe overfitting of powerful supervised learning models as a result.

4. *How do deep learning and probabilistic fuzzy systems compare on predicting septic shock mortality?*

When fed the same feature set, our densely connected neural network predicts slightly better than the probabilistic fuzzy system optimized with gradient descent. At a somewhat lower level of performance reside the probabilistic fuzzy system without gradient descent tuning and a recurrent neural network with gated recurrent units. The performance of these last two models is very similar. When fed all available features, both the densely connected neural network and the recurrent neural network outperform the probabilistic fuzzy system with gradient descent optimization, with the densely connected neural network having the highest performance among the two networks, though the difference is slight. Upon inquiring the probabilistic fuzzy system, we find that its internal representations are in line with clinical intuition to a sizable extent, but the chosen membership functions used in this research do not seem fully appropriate for the data while various rules also show considerable redundancy and irrelevancy. Upon performing a comparative inquiry into the models on a global level, we find that the densely connected neural network trained on the same 7 features as in the probabilistic fuzzy system shows suspicious behavior for several variables that does not match clinical intuitions, indicating some overfitting. The densely connected neural network on all available features hardly shows such suspicious behavior and seems to match clinical intuitions regarding its many features and their relationships to mortality risk rather well. Through comparing the feature permutation importance for all models, we find that recurrent neural networks with gated recurrent units are most resilient to feature permutation, hinting that these models seem well-suited to tackle sparse and irregular multivariate time series with potentially unreliable feature values at particular timesteps. In our comparative local inquiry of the models, which includes a case study of predictions over time, we again find evidence that the GRU delivers more stable predictions over time than the other models, likely because of its hidden internal state and explicit modeling of different timesteps. Furthermore, we find that all models correctly predict a high mortality risk near the end of the ICU stay for the selected patient, but various other patients can be found in the data for which all models struggle to predict correctly.

7.1 Contributions

Bringing new technologies into practice within the healthcare domain is a long and challenging process. Though this work is not directly applicable in a real ICU setting without further research, it is one of the first steps of the many required to improve healthcare for septic shock patients. As septic shock is a large, costly and challenging worldwide problem (as discussed in Section 2.1.3 and 2.1.4), even small steps towards improved healthcare for people with this condition are valuable.

Firstly, this work contributes with a direct empirical comparison between probabilistic fuzzy systems and deep learning for septic shock mortality prediction, which, to the best of our knowledge, has not been addressed in previous works. In this empirical comparison, we have included several relatively new techniques from the field of interpretable machine learning, thereby contributing to the recent stream of research that aims to increase trust in and transparency of complex black-box models as well. Secondly, this work contributes with a demonstration of how to leverage misaligned and unevenly sampled events for predictive modeling on MIMIC-III. We have empirically evaluated the effectiveness of various filling and splitting strategies, thereby paving the way for future research that aims to leverage misaligned and unevenly sampled events for predictive modeling purposes. As MIMIC-III is also freely accessible for research purposes, and because our implementation code contains elaborate comments and various reusable functions, the bar to re-use parts of this work is low. The fact that all required data transformations in this work can

be executed on a local machine in a small amount of time is also key advantage compared to the work by Wang et al. (2020): the extraction pipeline in that work takes many hours and requires an enormous amount of RAM which is not available on regular high-end laptops. Thirdly, the developed Python modules that contain the implementation of probabilistic fuzzy systems and the conversion process from misaligned and unevenly sampled events into a time series with self-chosen filling strategies is a key contribution. The classes and functions in these modules are not restricted to MIMIC-III, but can be applied in any domain. The implementation of the PFS-CP closely resembles the models in the popular scikit-learn package (Pedregosa et al., 2011), making it easy to use for the many data scientists that are familiar with this package. The PFS-ML is implemented as a Tensorflow Keras layer (Abadi et al., 2015), which similarly generates ease of use for the many data scientists familiar with Tensorflow. In addition, this enables easy customization via the Tensorflow ecosystem. For instance, it is relatively simple to substitute other gradient descent algorithms with user-specified hyperparameters for training the algorithm. Through the default capability of Tensorflow to distribute computations over many cores and machines, the PFS-ML also scales well to even larger datasets than the one used in this work.

Considering the three key contributions mentioned above, it is our view that this work is more than an incremental model improvement despite the fact that our best deep learning model only improves the performance of the probabilistic fuzzy system by approximately 10% on our evaluation metric. Transferring the probabilistic fuzzy system for septic shock mortality prediction to MIMIC-III in a transparent, reproducible and reusable manner is far from trivial, as demonstrated by the length and complexity of this work. Finally, we would like to emphasize that a synthesis of probabilistic fuzzy systems, deep learning, model evaluation metrics and machine learning interpretability, combined with a well-documented, transparent and reusable implementation on MIMIC-III, together delivers a valuable package that puts future efforts that aim to leverage medical data for predictive modeling at a considerable head start.

7.2 Economic Value

Though a full quantitative assessment of the economic value of our work is beyond our scope and very difficult to execute, we still provide a very basic estimation in this section as an indication. Recall that in Section 2.1.3 and 2.1.4, we explored the incidence, mortality rates and costs related to sepsis, severe sepsis and septic shock. In an extensive and recent study by Rudd et al. (2020), it was found that in 2017, 11 million sepsis-related deaths occurred worldwide. Let us now assume that probably, a majority of these deaths was due to septic shock, say 75%. This would imply that worldwide, approximately 0.75 x 11 = 8.25 million deaths would have occurred due to septic shock. Another relevant fact from Section 2.1.4 is that in the Netherlands, the cost of a septic shock hospitalization is €32,875 on average (Koster-Brouwer et al., 2016), which is quite close to the costs of such a hospitalization in the US, which is $38,298 (Paoli et al., 2018). If we somewhat crudely assume that the cost of a septic shock hospitalization is approximately €30,000 worldwide, then the worldwide cost of septic shock hospitalizations that resulted in death would be €30,000 x 8.25 million ≈ €250 billion billion per year. This is an unfathomable amount of money which demonstrates that even incredibly small contributions towards better healthcare for septic shock patients can have substantial economic value. For instance, if the developed analyses and software in this thesis constitute 0.001% of the work required to reduce the costs of each septic shock hospitalization by 1%, then the economic value of this work would be 0.00001 * 0.01 * €250 billion = €25,000 per year. As elaborated upon in Section 7.1, this work is such a small step towards what will hopefully eventually become better care for septic shock patients.

7.3 Limitations and Recommendations for Future Research

Implementing the analyses in this work has involved a long trail of decisions, many of which could have been taken differently. Unfortunately, the results of the comparison between probabilistic

fuzzy systems and deep learning are likely to depend strongly on many of these decisions. Throughout the description of the analyses in this work, we have already mentioned a sizable amount of considerations, limitations and alternatives. In this section, we highlight the main limitations of our work and use these to form recommendations for future research.

The decisions made in the preprocessing of the data from MIMIC-III are probably the most impactful on the results. One of the first steps in the preprocessing was finding the events that correspond to the variables of interest in this research. Though we have performed a fairly extensive search for such events, our search was somewhat ad-hoc and some relevant events could have been missed. In addition, the merging of different types of events into one variable (e.g. the multiple types of events for diastolic blood pressure) has been challenging because it is difficult to decide whether two types of events belong to the same concept or not. A related issue is that the data quality of some measurement events is questionable due to them having unrealistically large values. For bedside monitoring, we have attempted to fix this through a somewhat provisional value range check, but for laboratory testing, it is difficult to decide on such ranges and such a check was thus omitted. Needless to say, the in- or exclusion of outliers can exert great influence on model behavior. Another set of potentially impactful preprocessing decisions is that we have not excluded pediatric patients, have defined patients as the 'cases' rather than e.g. ICU stays, and have used diagnosis codes to identify septic shock patients rather than the physiological sepsis-3 criteria. Furthermore, MIMIC-III contains an incredible wealth of information, and much information has remained unleveraged in our models because of our decision to restrict our scope to the variables in the work by Fialho et al. (2016). Naturally, many other features that are predictive of mortality could be investigated in future works. To name but a handful, one could consider a wider range of laboratory testing variables, medication, nutrition and demographics. Also, the choice to use mortality within 72 hours as the target label is somewhat arbitrary - perhaps it is better to predict mortality probability within multiple time frames, or even time to mortality. Possibly, this would lead to more informative labels and better models as a result.

In our inquiry of the probabilistic fuzzy system, it appeared that its success in fitting the data was limited. In particular, we would firstly advise that future works which aim to use probabilistic fuzzy systems investigate different initialization strategies. For instance, one could consider to use the fuzzy clustering method with particle swarm optimization as proposed by Fuchs et al. (2019), or one could use a supervised learning method to generate partitions. In similar vein, different membership function for the rules in the probabilistic fuzzy system (i.e. other functions than gaussians) could be explored as well, perhaps accompanied by the rule base simplification strategies as proposed by Setnes et al. (1998) or Fuchs et al. (2020) to avoid irrelevant and redundant rules. Also, probabilistic fuzzy systems offer the flexibility to use completely different features for individual rules, which is another aspect that has been left unexplored in this work.

Though we already discussed the limitations of our model tuning and feature selection methods in-depth earlier, it would be inappropriate to refrain from mentioning them here. It has been our experience that selecting which feature selection and model tuning methods to use is a decision that leaves one stuck between a rock and a very hard place: each method has strong drawbacks, be it high computational cost, greedyness, or the exclusion of feature interactions. A related objection to our chosen approach could be that we use a static training and validation set for model tuning, which could have led to validation set overfitting. In light of the current rise of cheap cloud technology on which computations can be distributed with minimal refactoring, we are likely but a few years away from the possibility to use cross-validation on the large amount of data in this work to counteract validation set overfitting. Likewise, greater computational power could be leveraged effectively by experimenting with bigger neural networks, more feature selection methods and a wider hyperparameter search. Perhaps, this process can soon be automated to a great extent as well because the field of automated machine learning seems to be making rapid progress (Hutter et al., 2019).

Though the interpretable machine learning techniques used in this work have resulted in valuable insights, we must note that our local inquiry has been limited. Upon closer inspection, it seems that

the application of LIME and SHAP present considerable challenges, among which the impactful algorithm setup and the many possible explanations for a single instance are perhaps the most prominent. This makes it all the more dangerous that in this work, we could not afford to dive deeply into the mechanics behind these techniques, and had to rely on implementations that automate parts of the computations with possibly incorrect assumptions. In our view, local explanation methods currently possess such high a level of complexity that they should be studied in a more focused and isolated manner, e.g. on simpler datasets that do not raise the question whether the model or the feature engineering is the cause of quirky model behavior.

Finally, it is relevant to realize that this work was written by (an aspiring) data scientist without access to certified medical professionals. Despite the best of our efforts, it could thus well be the case that upon reading this document, a medical professional would frown his or her eyebrows when reading certain descriptions, decisions or results. To indicate some areas where this could be happen, it could be that our selection and merging of event types (`ITEMID`s) is inappropriate in particular areas, that we have included results from laboratory testing that were indisputably measurement errors or that our validation of model behavior in the model inquiry is partially wrong or too simplistic. The emphasis in this work has been on data science rather than medical science.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Retrieved from `http://tensorflow.org/` (Software available from tensorflow.org)

Adadi, A. & Berrada, M. (2018). Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI). *IEEE Access*, *6*, 52138–52160.

Alhaija, H., Mustikovela, S., Mescheder, L., Geiger, A. & Rother, C. (2018). Augmented reality meets computer vision: Efficient data generation for urban driving scenes. *International Journal of Computer Vision (IJCV).*

Alonso, J. M., Castiello, C. & Mencar, C. (2015). Interpretability of Fuzzy Systems: Current Research Trends and Prospects. In *Springer Handbook of Computational Intelligence* (pp. 219–237). Springer.

Apley, D. W. & Zhu, J. (2016). *Visualizing the effects of predictor variables in black box supervised learning models.* Retrieved from `https://arxiv.org/abs/1612.08468`

Arrieta], A. B., Díaz-Rodríguez, N., Ser], J. D., Bennetot, A., Tabik, S., Barbado, A., ... Herrera, F. (2020). Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, *58*, 82–115.

Ayazoglu, T. A. (2011). A comparison of APACHE II and APACHE IV scoring systems in predicting outcome in patients admitted with stroke to an intensive care unit. *Anaesthesia, Pain & Intensive Care*, *15*(1), 7–12.

Bouch, D. C. & Thompson, J. P. (2008). Severity scoring systems in the critically ill. *Continuing Education in Anaesthesia, Critical Care & Pain*, *8*(5), 181-185.

Burki, T. K. (2016). Predicting lung cancer prognosis using machine learning. *The Lancet Oncology*, *17*(10), e421.

Centers for Disease Control and Prevention. (2016). *Sepsis.* Retrieved 7 January 2020, from `https://www.cdc.gov/sepsis/datareports/index.html`

Che, Z., Purushotham, S., Cho, K., Sontag, D. & Liu, Y. (2018). Recurrent neural networks for multivariate time series with missing values. *Scientific Reports*, *8*(1), 1–12.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. & Bengio, Y. (2014). *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.* Retrieved from `https://arxiv.org/abs/1406.1078`

Chollet, F. (2017). *Deep learning with python* (1st ed.). Greenwich, CT, USA: Manning Publications Co.

Cismondi, F., Fialho, A. S., Vieira, S. M., Reti, S. R., Sousa, J. M. & Finkelstein, S. N. (2013). Missing data in medical databases: Impute, delete or classify? *Artificial Intelligence in Medicine*, *58*(1), 63–72.

Cowan, N. (2001). The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, *24*(1), 87–114.

Davis, J. & Goadrich, M. (2006). The relationship between precision-recall and roc curves. In *Proceedings of the 23rd International Conference on Machine learning* (pp. 233–240).

Desautels, T., Calvert, J., Hoffman, J., Jay, M., Kerem, Y., Shieh, L., ... Das, R. (2016). Prediction of Sepsis in the Intensive Care Unit With Minimal Electronic Health Record Data: A Machine Learning Approach. *JMIR Med Inform*, *4*(3), e28.

Esteva, A., Robicquet, A., Ramsundar, B., Kuleshov, V., DePristo, M., Chou, K., ... Dean, J. (2019). A guide to deep learning in healthcare. *Nature Medicine*, *25*, 24-29.

Evans, T. (2018). Diagnosis and management of sepsis. *Clinical Medicine*, *18*(2), 146–149.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, *27*(8), 861–874.

Fialho, A. S., Cismondi, F., Vieira, S. M., Sousa, J. M., Reti, S. R., Howell, M. D. & Finkelstein, S. N. (2010). Predicting outcomes of septic shock patients using feature selection based on soft computing techniques. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems* (pp. 65–74).

Fialho, A. S., Vieira, S. M., Kaymak, U., Almeida, R. J., Cismondi, F., Reti, S. R., ... Sousa, J. M. (2016). Mortality prediction of septic shock patients using probabilistic fuzzy systems. *Applied Soft Computing*, *42*, 194–203.

Fleischmann, C., Scherag, A., Adhikari, N. K., Hartog, C. S., Tsaganos, T., Schlattmann, P., ... Reinhart, K. (2016). Assessment of global incidence and mortality of hospital-treated sepsis. Current estimates and limitations. *American Journal of Respiratory and Critical Care Medicine*, *193*(3), 259–272.

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 1189–1232.

Fritsch, J., Kuehnl, T. & Geiger, A. (2013). A New Performance Measure and Evaluation Benchmark for Road Detection Algorithms. In *International Conference on Intelligent Transportation Systems (ITSC)* (pp. 1693–1700).

Fuchs, C., Spolaor, S., Nobile, M. S. & Kaymak, U. (2019). A Swarm Intelligence Approach to Avoid Local Optima in Fuzzy C-Means Clustering. In *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)* (pp. 1–6).

Fuchs, C., Spolaor, S., Nobile, M. S. & Kaymak, U. (2020). A Graph Theory Approach to Fuzzy Rule Base Simplification. In *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems* (pp. 387–401).

Gacto, M. J., Alcalá, R. & Herrera, F. (2011). Interpretability of linguistic fuzzy rule-based systems: An overview of interpretability measures. *Information Sciences*, *181*(20), 4340–4360.

Geiger, A., Lenz, P. & Urtasun, R. (2012). Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 3354–3361).

Glorot, X. & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (pp. 249–256).

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep learning*. MIT Press. (`http://www.deeplearningbook.org`)

Gorbalenya, A., Baker, S., Baric, R. et al. (2020). The species Severe acute respiratory syndrome-related coronavirus: classifying 2019-nCoV and naming it SARS-CoV-2. *Nature Microbiology*, *5*(4), 536.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R. & Schmidhuber, J. (2016). LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, *28*(10), 2222–2232.

Guyon, I. & Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, *3*(Mar), 1157–1182.

Hanisch, E., Brause, R., Paetz, J. & Arlt, B. (2011). Review of a large clinical series: Predicting death for patients with abdominal septic shock. *Journal of Intensive Care Medicine*, *26*(1), 27–33.

Harutyunyan, H., Khachatrian, H., Kale, D. C., Ver Steeg, G. & Galstyan, A. (2019). Multitask learning and benchmarking with clinical time series data. *Scientific Data*, *6*(1), 96.

Higgins, T. L., Teres, D., Copes, W., Nathanson, B., Stark, M. & Kramer, A. (2005). Updated Mortality Probability Model (MPM-III). *Chest*, *128*(4), 348S.

Hutter, F., Kotthoff, L. & Vanschoren, J. (2019). *Automated Machine Learning*. Springer.

Ioffe, S. & Szegedy, C. (2015). *Batch normalization: Accelerating deep network training by reducing internal covariate shift.* Retrieved from `https://arxiv.org/abs/1502.03167`

Jaimes, F., Farbiarz, J., Alvarez, D. & Martínez, C. (2005). Comparison between logistic regression and neural networks to predict death in patients with suspected sepsis in the emergency room. *Critical Care*, *9*(2), R150.

James, G., Witten, D., Hastie, T. & Tibshirani, R. (2013). *An Introduction to Statistical Learning* (Vol. 112). Springer.

Johnson, A. E., Aboab, J., Raffa, J. D., Pollard, T. J., Deliberato, R. O., Celi, L. A. & Stone, D. J. (2018). A Comparative Analysis of Sepsis Identification Methods in an Electronic Database. *Critical Care Medicine*, *46*(4), 494–499.

Johnson, A. E., Kramer, A. A. & Clifford, G. D. (2013). A new severity of illness scale using a subset of acute physiology and chronic health evaluation data elements shows comparable predictive accuracy. *Critical Care Medicine*, *41*(7), 1711–1718.

Johnson, A. E., Pollard, T. J. & Mark, R. G. (2017). Reproducibility in critical care: a mortality prediction case study. In *Machine Learning for Healthcare Conference* (pp. 361–376).

Johnson, A. E., Pollard, T. J., Shen, L., Li-wei, H. L., Feng, M., Ghassemi, M., ... Mark, R. G. (2016). MIMIC-III, a freely accessible critical care database. *Scientific data*, *3*, 160035.

Johnson, A. E., Stone, D. J., Celi, L. A. & Pollard, T. J. (2017). The MIMIC Code Repository: enabling reproducibility in critical care research. *Journal of the American Medical Informatics Association*, *25*(1), 32–39.

Jozefowicz, R., Zaremba, W. & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning* (pp. 2342–2350).

Kaggle. (2020). *Competitions.* Retrieved 12 February 2020, from https://www.kaggle.com/competitions

Kaymak, U., Ben-David, A. & Potharst, R. (2012). The AUK: A simple alternative to the AUC. *Engineering Applications of Artificial Intelligence*, *25*(5), 1082–1089.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems 30* (pp. 3146–3154).

Kingma, D. P. & Ba, J. (2014). *Adam: A method for stochastic optimization.* Retrieved from https://arxiv.org/abs/1412.6980

Klaise, J., Van Looveren, A., Vacanti, G. & Coca, A. (2020). *Alibi: Algorithms for monitoring and explaining machine learning models.* Retrieved from https://github.com/SeldonIO/alibi

Knaus, W. A., Draper, E. A., Wagner, D. P. & Zimmerman, J. E. (1985). APACHE II: a severity of disease classification system. *Critical Care Medicine*, *13*(10), 818–829.

Koster-Brouwer, M. E., Klouwenberg, K., Peter, M., Pasma, W., Bosmans, J. E., van der Poll, T., ... Cremer, O. L. (2016). Critical care management of severe sepsis and septic shock: a cost-analysis. *Netherlands Journal of Critical Care*, *24*(3), 12–18.

Kraskov, A., Stögbauer, H. & Grassberger, P. (2004). Estimating mutual information. *Physical Review E*, *69*(6), 066138.

Kumar, A., Roberts, D., Wood, K. E., Light, B., Parrillo, J. E., Sharma, S., ... others (2006). Duration of hypotension before initiation of effective antimicrobial therapy is the critical determinant of survival in human septic shock. *Critical Care Medicine*, *34*(6), 1589–1596.

Lanham, J. (2020). *Gradient descent explained.* Retrieved 24 April 2020, from https://www.oreilly.com/library/view/learn-arcore-/9781788830409/e24a657a-a5c6-4ff2-b9ea-9418a7a5d24c.xhtml

Le, J. G., Loirat, P., Alperovitch, A., Glaser, P., Granthil, C., Mathieu, D., ... Villers, D. (1984). A simplified acute physiology score for ICU patients. *Critical Care Medicine*, *12*(11), 975–977.

LeCun, Y., Bengio, Y. & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

Le Gall, J.-R., Lemeshow, S., Leleu, G., Klar, J., Huillard, J., Rué, M., ... Artigas, A. (1995). Customized probability models for early severe sepsis in adult intensive care patients. *JAMA*, *273*(8), 644–650.

Lemeshow, S., Teres, D., Klar, J., Avrunin, J. S., Gehlbach, S. H. & Rapoport, J. (1993). Mortality Probability Models (MPM II) Based on an International Cohort of Intensive Care Unit Patients. *JAMA*, *270*(20), 2478–2486.

Lemeshow, S., Teres, D., Pastides, H., Avrunin, J. S. & Steingrub, J. S. (1985). A Method for Predicting Survival and Mortality of ICU Patients Using Objectively Derived Weights. *Critical Care Medicine*, *13*(7), 519–525.

Liu, V. X., Fielding-Singh, V., Greene, J. D., Baker, J. M., Iwashyna, T. J., Bhattacharya, J. & Escobar, G. J. (2017). The timing of early antibiotics and hospital mortality in sepsis. *American Journal of Respiratory and Critical Care Medicine*, *196*(7), 856–863.

Lundberg, S. M. & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems 30* (pp. 4765–4774). Curran Associates, Inc.

Lundberg, S. M., Nair, B., Vavilala, M. S., Horibe, M., Eisses, M. J., Adams, T., ... Lee, S.-I. (2018). Explainable machine-learning predictions for the prevention of hypoxaemia during surgery. *Nature Biomedical Engineering*, *2*(10), 749–760.

Manero-Bastin, A. (2019). *How to Configure the Number of Layers and Nodes in a Neural Network.* Retrieved 17 February 2020, from https://www.datasciencecentral.com/profiles/blogs/how-to-configure-the-number-of-layers-and-nodes-in-a-neural

Martin, G. S. (2012). Sepsis, severe sepsis and septic shock: changes in incidence, pathogens and outcomes. *Expert Review of Anti-infective Therapy*, *10*(6), 701-706.

Mendel, J. M. (2017). *Uncertain Rule-Based Fuzzy Systems.* Springer. Retrieved from https://link.springer.com/book/10.1007/978-3-319-51370-6

Miller, T. (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, *267*, 1–38.

Molnar, C. (2019). *Interpretable Machine Learning.* Retrieved from https://christophm.github.io/interpretable-ml-book/

Moreno, R. P., Metnitz, P. G. H., Almeida, E., Jordan, B., Bauer, P., Campos, R. A., ... on behalf of the SAPS 3 Investigators (2005). SAPS 3—From evaluation of the patient to evaluation of the intensive care unit. Part 2: Development of a prognostic model for hospital mortality at ICU admission. *Intensive Care Medicine*, *31*(10), 1345–1355.

Norgeot, B., Glicksberg, B. S. & Butte, A. J. (2019). A call for deep-learning healthcare. *Nature Medicine*, *25*(1), 14-15.

Olah, C., Mordvintsev, A. & Schubert, L. (2017). Feature visualization. *Distill*. Retrieved from https://distill.pub/2017/feature-visualization

Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K. & Mordvintsev, A. (2018). The Building Blocks of Interpretability. *Distill*. Retrieved from https://distill.pub/2018/building-blocks

Paoli, C. J., Reynolds, M. A., Sinha, M., Gitlin, M. & Crouser, E. (2018). Epidemiology and costs of sepsis in the United States—An analysis based on timing of diagnosis and severity Level. *Critical Care Medicine*, *46*(12), 1889-1897.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Polat, G., Ugan, R. A., Cadirci, E. & Halici, Z. (2017). Sepsis and septic shock: current treatment strategies and new approaches. *The Eurasian Journal of Medicine*, *49*(1), 53–58.

Provost, F. & Fawcett, T. (2013). *Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking* (1st ed.). O'Reilly Media, Inc.

Purushotham, S., Meng, C., Che, Z. & Liu, Y. (2018). Benchmarking deep learning models on large healthcare datasets. *Journal of Biomedical Informatics*, *83*, 112–134.

Rajkomar, A., Dean, J. & Kohane, I. (2019). Machine learning in medicine. *New England Journal of Medicine*, *380*(14), 1347–1358.

Reyna, M., Josef, C., Jeter, R., Shashikumar, S., Westover, M., Nemati, S., ... Sharma, A. (2019). Early prediction of sepsis from clinical data: the PhysioNet/Computing in Cardiology Challenge 2019. *Critical Care Medicine*.

Ribeiro, M. T., Singh, S. & Guestrin, C. (2016). "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (p. 1135–1144).

Ribeiro, M. T., Singh, S. & Guestrin, C. (2018). Anchors: High-precision model-agnostic explanations. In *Thirty-Second AAAI Conference on Artificial Intelligence.*

Rubin, D. B. (1976). Inference and missing data. *Biometrika*, *63*(3), 581–592.

Rudd, K. E., Johnson, S. C., Agesa, K. M., Shackelford, K. A., Tsoi, D., Kievlan, D. R., ... others (2020). Global, regional, and national sepsis incidence and mortality, 1990–2017: analysis for the global burden of disease study. *The Lancet*, *395*(10219), 200–211.

Ruder, S. (2016). *An overview of gradient descent optimization algorithms.* Retrieved from https://arxiv.org/abs/1609.04747

Saito, T. & Rehmsmeier, M. (2015). The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PloS One*, *10*(3).

SchemaSpy. (2019). *SchemaSpy Analysis of mimic.mimiciii - All Relationships.* Retrieved 15 January 2020, from https://mit-lcp.github.io/mimic-schema-spy/relationships.html

Schmidt, G. A. & Mandel, J. (2019). *Evaluation and Management of Suspected Sepsis and Septic Shock in Adults".* Retrieved 9 January 2020, from https://stichting-nice.nl/doc/jaarboek-2018-web.pdf

Setnes, M., Babuska, R., Kaymak, U. & van Nauta Lemke, H. R. (1998). Similarity measures in fuzzy rule base simplification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *28*(3), 376–386.

Seymour, C. W., Gesten, F., Prescott, H. C., Friedrich, M. E., Iwashyna, T. J., Phillips, G. S., ... Levy, M. M. (2017). Time to treatment and mortality during mandated emergency care for sepsis. *New England Journal of Medicine*, *376*(23), 2235–2244.

Shamsudin, S. (2013). *The Development of Neural Network Based System Identification and Adaptive Flight Control for an Autonomous Helicopter System* (Unpublished doctoral dissertation).

Shapley, L. S. (1953). A value for n-person games. *Contributions to the Theory of Games*, *2*(28), 307–317.

Sheskin, D. J. (2007). *Handbook of Parametric and Nonparametric Statistical Procedures* (4th ed.). Chapman and Hall/CRC.

Shickel, B., Tighe, P. J., Bihorac, A. & Rashidi, P. (2017). Deep EHR: a survey of recent advances in deep learning techniques for electronic health record (EHR) analysis. *IEEE Journal of Biomedical and Health Informatics*, *22*(5), 1589–1604.

Sicara. (2019). *tf-explain.* Retrieved from https://github.com/sicara/tf-explain

Singer, M., Deutschman, C. S., Seymour, C. W., Shankar-Hari, M., Annane, D., Bauer, M., ... Angus, D. C. (2016). The Third International Consensus Definitions for Sepsis and Septic Shock (Sepsis-3). *JAMA*, *315*(8), 801-810.

Song, T. (2018). *Implementing the Perceptron algorithm from scratch with Python.* Retrieved 17 February 2020, from https://tianyusong.com/2018/02/23/implementing-the-perceptron-algorithm-from-scratch-with-python/

Stichting NICE. (2018). *Jaarboek 2018.* Retrieved from https://stichting-nice.nl/doc/jaarboek-2018-web.pdf

Torio, C. & Moore, B. (2013). *National Inpatient Hospital Costs: The Most Expensive Conditions by Payer.* Retrieved 7 January 2020, from https://europepmc.org/article/med/27359025

Van Den Berg, J., Kaymak, U. & Van Den Bergh, W.-M. (2002). Fuzzy classification using probability-based rule weighting. In *2002 IEEE World Congress on Computational Intelligence. 2002 IEEE International Conference on Fuzzy Systems* (Vol. 2, pp. 991–996).

Wang, S., McDermott, M. B., Chauhan, G., Ghassemi, M., Hughes, M. C. & Naumann, T. (2020). Mimic-extract: A data extraction, preprocessing, and representation pipeline for MIMIC-III. In *Proceedings of the ACM Conference on Health, Inference, and Learning* (pp. 222–235).

World Health Organization. (2018). *Sepsis.* Retrieved 28 December 2019, from https://www.who.int/news-room/fact-sheets/detail/sepsis

Xiao, C., Choi, E. & Sun, J. (2018). Opportunities and challenges in developing deep learning models using electronic health records data: a systematic review. *Journal of the American Medical Informatics Association*, *25*(10), 1419–1428.

Yanase, J. & Triantaphyllou, E. (2019). A systematic survey of computer-aided diagnosis in medicine: Past and present developments. *Expert Systems with Applications*, *138*, 112821.

Yu, K.-H., Beam, A. L. & Kohane, I. S. (2018). Artificial intelligence in healthcare. *Nature Biomedical Engineering*, *2*(10), 719–731.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*(5), 338–353.

Zimmerman, J. E., Kramer, A. A., McNair, D. S. & Malila, F. M. (2006). Acute Physiology and Chronic Health Evaluation (APACHE) IV: hospital mortality assessment for today's critically ill patients. *Critical Care Medicine*, *34*(5), 1297–1310.

Appendix A

Variables in Severity Scoring Systems

A.1 Overview of Variables Used in Popular Severity Scoring Systems

Table A.1: Comparison between the variables used in the most relevant severity scoring systems. The rightmost column indicates in how many of the systems that the variable occurs.

Variable	APACHE II	APACHE IV	SAPS 3	MPM II	OASIS	Freq.
Age	✓	✓	✓	✓	✓	5
Glasgow coma scale (GCS)	✓	✓	✓	✓	✓	5
Heart rate	✓	✓	✓	✓	✓	5
Temperature	✓	✓	✓	✗	✓	4
Creatinine	✓	✓	✓	✗	✗	3
Mean arterial pressure	✓	✓	✗	✗	✓	3
Respiratory rate	✓	✓	✗	✗	✓	3
Ventilation status	✗	✓	✗	✓	✓	3
Acute renal failure	✓	✗	✗	✓	✗	2
Bilirubin	✗	✓	✓	✗	✗	2
Chronic renal failure	✗	✓	✓	✗	✗	2
Cirrhosis	✗	✓	✓	✗	✗	2
FiO2	✓	✓	✗	✗	✗	2
Hematocrit	✓	✓	✗	✗	✗	2
ICU admission: planned or unplanned	✗	✗	✓	✓	✗	2
Intra-hospital location before ICU admission	✗	✓	✓	✗	✗	2
PaO2	✓	✓	✗	✗	✗	2
pH	✓	✓	✗	✗	✗	2
Sodium	✓	✓	✗	✗	✗	2
Systolic blood pressure	✗	✗	✓	✓	✗	2
Urine output	✗	✓	✗	✗	✓	2
White blood cell count	✓	✓	✗	✗	✗	2
A-a gradient	✓	✗	✗	✗	✗	1

Table A.1: Comparison between the variables used in the most relevant severity scoring systems. The rightmost column indicates in how many of the systems that the variable occurs.

Variable	APACHE II	APACHE IV	SAPS 3	MPM II	OASIS	Freq.
Acute infection at ICU admission	✗	✗	✓	✗	✗	1
AIDS	✗	✓	✗	✗	✗	1
Albumin	✗	✓	✗	✗	✗	1
Anatomical site of surgery	✗	✗	✓	✗	✗	1
Biosafety level	✗	✓	✗	✗	✗	1
Cardiac dysrhythmia	✗	✗	✗	✓	✗	1
Cardiopulmonary resuscitation before admission	✗	✗	✗	✓	✗	1
Cerebrovascular incident	✗	✗	✗	✓	✗	1
Co-morbidities	✗	✗	✓	✗	✗	1
Elective surgery	✗	✗	✗	✗	✓	1
Emergency Surgery	✗	✓	✗	✗	✗	1
Gastrointestinal bleeding	✗	✗	✗	✓	✗	1
Geographic area	✗	✗	✓	✗	✗	1
Hepatic failure	✗	✓	✗	✗	✗	1
History of severe organ failure or immunocompromise	✓	✗	✗	✗	✗	1
Hydrogen ion concentration	✗	✗	✓	✗	✗	1
Immunosupression	✗	✓	✗	✗	✗	1
Intracranial mass effect	✗	✗	✗	✓	✗	1
Leukemia / Myeloma	✗	✓	✗	✗	✗	1
Leukocytes	✗	✗	✓	✗	✗	1
Lymphoma	✗	✓	✗	✗	✗	1
Metastatic carcinoma	✗	✓	✗	✗	✗	1
Metastatic neoplasm	✗	✗	✗	✓	✗	1
Oxygenation	✗	✗	✓	✗	✗	1
PaCO2	✗	✓	✗	✗	✗	1
Platelets	✗	✗	✓	✗	✗	1
Potassium	✓	✗	✗	✗	✗	1
Pre-ICU in-hospital length of stay	✗	✗	✗	✗	✓	1
Readmission	✗	✓	✗	✗	✗	1
Reasons(s) for ICU admission	✗	✗	✓	✗	✗	1
Surgical status at ICU admission	✗	✗	✓	✗	✗	1
Urea	✗	✓	✗	✗	✗	1
Use of major therapeutic options before ICU admission	✗	✗	✓	✗	✗	1

Appendix B

Details on Previous MIMIC-III Benchmark Datasets

B.1 Conceptual Description of the Benchmark Datasets

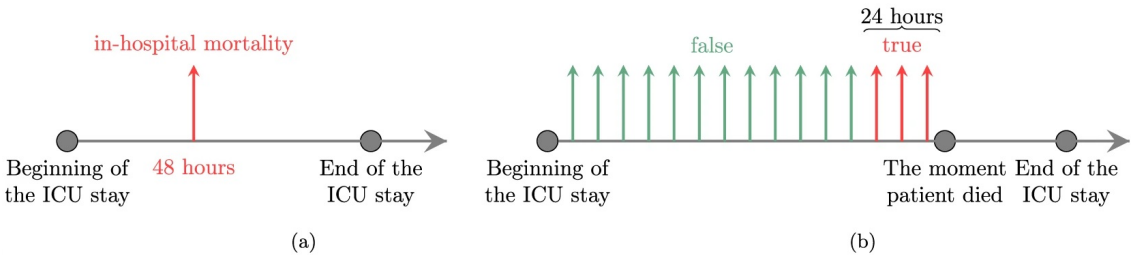

Figure B.1: Intuition behind two relevant benchmark datasets on MIMIC-III from Harutyunyan et al. (2019). (a) involves predicting in-hospital mortality 48 hours after admission, which corresponds to the "in-hospital mortality" benchmark in Harutyunyan et al. (2019) and Purushotham et al. (2018). (b) involves predicting mortality within 24 hours during every of the ICU stay, which corresponds to the "decompensation" benchmark in Harutyunyan et al. (2019). Figure adapted from Harutyunyan et al. (2019).

B.2 Variable Comparison between Fialho and the Benchmark Datasets

Table B.1: Comparison between the types of patient measurements (variables) used in the original work on predicting mortality using probabilistic fuzzy systems by Fialho et al. (2016) and the benchmark datasets by Harutyunyan et al. (2019) and Purushotham et al. (2018). For the last, we only consider the manually curated variable set (Feature Set A). Clearly, the overlap of both with Fialho et al. (2016) is limited.

Feature	**Present in Fialho et al.**	**Present in Harutyunyan et al.**	**Present in Purushotham et al.**
Fraction of inspired oxygen	✓	✓	✓
Heart Rate	✓	✓	✓
Arterial pO2	✓	✓	✓
Systolic blood pressure	✓	✓	✓
Temperature	✓	✓	✓
Diastolic blood pressure	✓	✓	✗
Glucose	✓	✓	✗
Arterial pH	✓	✓	✗
Bicarbonate	✓	✗	✓
White blood cells	✓	✗	✓
Sodium	✓	✗	✓
Potassium	✓	✗	✓
Urea	✓	✗	✓
Bilirubin	✓	✗	✓
Arterial PCO2	✓	✗	✗
Arterial base excess	✓	✗	✗
SpO2	✓	✗	✗
Hematocrit	✓	✗	✗
Platelets	✓	✗	✗
PTT	✓	✗	✗
AT3	✓	✗	✗
Calcium	✓	✗	✗
Creatinine	✓	✗	✗
GOT(ASAT)	✓	✗	✗
GPT(ALAT)	✓	✗	✗
CRP	✓	✗	✗
CVP	✓	✗	✗

B.3 Diagram of the Conversion Process from the Native MIMIC-III Database to the Benchmark Datasets

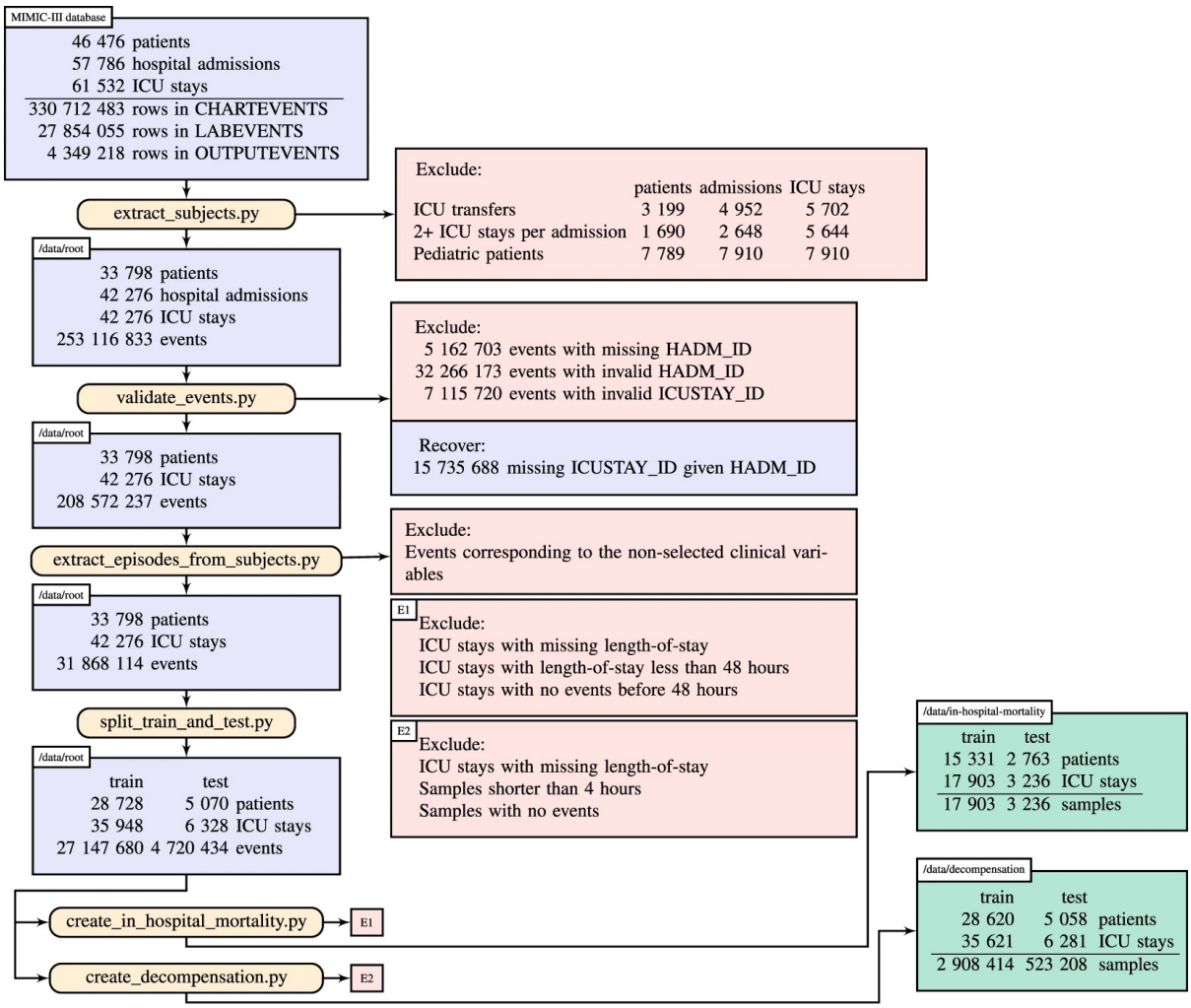

Figure B.2: Main steps, assumptions, statistics and scripts regarding the computation of the mortality and decompensation benchmark datasets from the MIMIC-III database. Adapted from Harutyunyan et al. (2019).

Appendix C

Database and Implementation Code Details

C.1 Database Schema

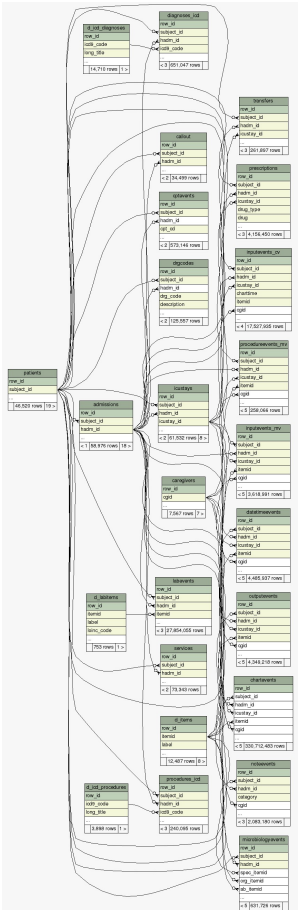

Figure C.1: Visualization of the MIMIC-III Schema (SchemaSpy, 2019)

C.2 Accessing and Building the MIMIC-III Database

Although MIMIC-III is an open database, it cannot be directly downloaded from a URL. The steps to obtain access to MIMIC-III are described in-detail on the "Requesting access" page in the official MIMIC-III documentation. In brief, one first has to complete the CITI "Data or Specimens Only Research" online course, which mainly serves the purpose of informing the user of the conditions for using MIMIC-III (e.g. one should not attempt to identify any individual in the dataset). Consequently, access can be requested to the dataset, which involves providing a description with the reason for access, obtaining a validation of your request by your supervisor (he or she is sent an e-mail with a link to validate your request), and getting consent from the managers of the PhysioNet platform. Afterwards, compressed versions of the tables in the database can be downloaded on the MIMIC-III page (an individual .csv.gz file for each table). Note that the compressed tables take approximately 6.2GB of disk space.

To access the tables conveniently without having to load them fully into RAM, the official MIMIC-III documentation provides a description of how to build a PostgreSQL database of MIMIC-III on your local machine, both for Windows and Unix/Mac. These pages delineate the various commands that have to be run in order to build the database. To sum up some additional relevant details:

- Our research was executed on a Windows machine with an Intel i7-8705G CPU, 8GB RAM and a PM981 NVMe Samsung 512GB SSD.
- To solve some errors that occurred when running the commands to build the PostgreSQ database, we had to untick the box "Beta: Use Unicode UTF-8 for worldwide language support" in Control Panel - Clock and Region - Region - Administrative - Change system locale.
- Filepaths in the PostgreSQL shell (PSQL) must be specified with forward path slashes and surrounded by single quotes.
- As the process of building the database seemed to freeze with the compressed tables on our machine (.csv.gz files), we built the database using the extracted tables (.csv files). Keep in mind that the extracted tables take 43GB of disk space, and that the PostgreSQL database of these files takes an additional 62GB including table indexes. This means that you will need at least 105GB of disk space to prepare the MIMIC-III database for use.
- On our machine, loading the (uncompressed) tables into the database took approximately 3 hours, while building the indexes (for faster query performance) took approximately 15 minutes.

As described on the page of the Windows tutorial, running SQL queries on MIMIC-III in the PSQL shell always requires one to run "\c mimic;" and "set search_path to mimiciii;" at the start of the session in order to query the tables in the PostgreSQL database. Before running these prompts, the PSQL shell asks the user to specify a server, database, port, username and password. For the first 4, the user can just hit enter without typing anything, while the password needs to typed in manually before hitting enter. Note that the window stays black while typing the password (i.e. the user does not see whether the typing actually happens), but the user needs to type the password anyway and then hit enter.

C.3 Tooling

The implementation of this research was executed using a combination of various tools. The notebooks were developed using Jupyter with the Anaconda environment manager. Needless to say, the MIMIC-III database is built using PostgreSQL. Modules with re-usable Python functions were developed using Visual Studio Code. A specification of the Python packages used in this

research, including their versions, can be found in the "package dependencies.txt" file in the top level of the supplementary folder. All materials are open source.

C.4 Implementation Materials

As emphasized by Johnson, Pollard & Mark (2017), reproducibility and reusability in machine learning research within the healthcare domain is extremely beneficial for reproducibility and building future works on top of a earlier works. Following the philosophy of Papers With Code, which are now also the official guidelines for NeurIPS 2020, we provide elaborate materials for reproducing this research and building on top of it in future works. In the supplementary folder to this thesis, you will find the following materials:

- 11 ordered Jupyter notebooks containing the actual implementation of the analyses with extensive comments, documentation and explanations. All lines that write files to disk are commented out for safety - this avoids you from directly overwriting files if you re-execute the notebooks. In notebook cells that took more than a few minutes to run, we put a comment that indicates how long the code ran on our machine as an indication.
- A text file with a specification of the (direct) package dependencies and a dump (.yml) of the full conda environment that can be used to dissect all additional, lower-level dependencies of the (direct) package dependencies.
- A set of Excel files with various results of the preprocessing and analysis that were produced along the way when executing the notebooks.
- A folder named "data" which contains material such as processed datasets and results from the model tuning. Note that the files of the MIMIC-III database itself are not included here, as access to the raw data must be acquired through the official access requesting process described in Section C.2.
- A folder named "exploratory data analysis" with various results that were produced in the corresponding Jupyter notebook.
- A folder named "modules" containing various re-usable Python modules with functionality used across the Jupyter notebooks. This functionality is particularly interesting for future research. It includes materials such as classes for probabilistic fuzzy systems, functions for computing the AUK, functions for executing templating efficiently on large datasets with a degree of customization, and a preamble with various common package imports used across all notebooks. We have included extensive docstrings to ease future (re-)use and extension.
- A folder named "notebook_figures" with various figures from this thesis that are also shown in various Jupyter notebooks for clarity of presentation.
- A folder named "queries", with two SQL queries that generate materialized views (processed datasets) on the MIMIC-III database for processing data out-of-memory. One of these ("Explicit_sepsis.sql") originates from the official MIMIC code repository (Johnson, Stone et al., 2017).

Appendix D

Details on the Results

D.1 Preprocessing

D.1.1 Joint Details on Laboratory Testing and Bedside Monitoring

Manually Included Additional ITEMIDs

Table D.1: Manually included `ITEMID`s (overriding the keyword search). We cannot keyword match for "ph" and "temp" because that would result in a tremendous amount of pollution `ITEMIDs`.

Variable type	**Variable**	**ITEMID**	**LABEL**
bedside monitoring	temperature	3654	temp skin [c]
bedside monitoring	temperature	3655	temp rectal [f]
laboratory testing	arterial base excess	50802	base excess
laboratory testing	arterial ph	50820	ph
laboratory testing	got(asat)	50878	asparate aminotransferase (ast)

Unit Conversion

Table D.2: `ITEMID`s that require unit conversion and their corresponding variables.

ITEMID	Variable Type	Explanation
50889	laboratory testing	Some of the events with this ITEMID have unit mg/dL, while the others have mg/L. We convert mg/dL entries to mg/L.
678	bedside monitoring	All entries of these ITEMIDs are in Fahrenheit. We convert them to Celsius.
679		
3654		
223761		
189	bedside monitoring	All entries of these ITEMIDs are fractions. We convert them to percentages.
190		
1206		
1863		
5955		
7041		
7570		

D.1.2 Details on Laboratory Testing Variables

Keywords for the Search on D_LABITEMS

Table D.3: Keywords used for the laboratory testing variables in the search over the `LABEL` column in `D_LABITEMS`. Recall that we also scoped to search solely entries with `FLUID` = blood.

Variable	Keywords
arterial base excess	arterial base excess
arterial pco2	arterial pco2, pco2
arterial ph	arterial ph
arterial po2	arterial oxygen saturation, sao2, o2sat, o2 sat, arterial po2, po2
at3	at3, antithrombin
bicarbonate	bicarbonate, hco3
bilirubin	bilirubin
calcium	calcium
creatinine	creatinine
crp	crp, c-reactive protein, c reactive protein
glucose	glucose
got(asat)	aspartate transaminase, aspartate aminotransferase, glutamic oxaloacetic transaminase, got
gpt(alat)	alanine transaminase, alanine aminotransferase, gpt, sgpt, serum glutamate-pyruvate transaminase, serum gluatmic-pyruvic transaminase
hematocrit	hematocrit, hct
platelets	platelets, thrombocytes, platelet count
potassium	potassium
ptt	ptt, partial thromboplastin time, kcct, kaolin-cephalin clotting time
sodium	sodium
urea	blood urea nitrogen, bun
white blood cells	white blood cells, wbc

Manually Excluded Pollution ITEMIDs

Table D.4: Manually excluded pollution `ITEMIDs` that pop up during the keyword search over `D_LABITEMS` for the laboratory testing variables.

Variable	ITEMID	LABEL	Reason
calcium	50808	free calcium	Assume we are only interested in total calcium
bilirubin	50883	bilirubin, direct	Assume we are interested in total bilirubin, which is direct + indirect
bilirubin	50884	bilirubin, indirect	Assume we are interested in total bilirubin, which is direct + indirect
platelets	51240	large platelets	This is (probably) a different concept than regular platelets
glucose	51529	estimated actual glucose	Assume we are only interested in regular glucose
white blood cells	51533	wbcp	Various different meanings for this abbrevation. Unclear which applicable, thus could be a different concept.

D.1.3 Details on Bedside Monitoring Variables

Keywords and categories for the Search on D_ITEMS

Table D.5: Keywords used for the bedside monitoring variables in the search over the `LABEL` column in `D_ITEMS`.

Variable	Keywords
cvp	cvp, central venous pressure
diastolic blood pressure	diastolic blood pressure, dbp, diastolic
fio2	fraction of inspired oxygen, fio2, fo2, inspired o2 fraction
heart rate	heart rate
spo2	spo2, peripheral oxygen saturation, o2 saturation pulseoxymetry
systolic blood pressure	systolic blood pressure, sbp, diastolic
temperature	temperature

Table D.6: Selected categories used for searching the relevant bedside monitoring `ITEMIDs` in `D_ITEMS`.

CATEGORY
cardiovascular (pulses)
general
output
pulmonary
respiratory
routine vital signs
None

Manually Excluded Pollution ITEMIDs

Table D.7: Initially manually excluded pollution `ITEMIDs` that pop up during the keyword search over `D_ITEMS` for the bedside monitoring variables.

Variable	ITEMID	LABEL	Reason
diastolic blood pressure	153	diastolic unloading	Different concept
fio2	191	fio2/o2 delivered	Different concept
systolic blood pressure	480	orthostat bp sitting [systolic]	Different concept - not a regular SBP
systolic blood pressure	482	orthostatbp standing [systolic]	Different concept - not a regular SBP
systolic blood pressure	484	orthostatic bp lying [systolic]	Different concept - not a regular SBP
systolic blood pressure	492	pap [systolic]	Different concept - pulmonary artery, which has strongly different pressure values then elsewhere
temperature	591	rle [temperature]	Unclear unit - celsius or fahrenheit? Also no unit of these in chartevents
temperature	597	rue [temperature]	Unclear unit - celsius or fahrenheit? Also no unit of these in chartevents
temperature	645	skin [temperature]	Unclear unit - celsius or fahrenheit? Also no unit of these in chartevents
systolic blood pressure	666	systolic unloading	Different concept
heart rate	3494	lowest heart rate	Different concept
diastolic blood pressure	8444	orthostat bp sitting [diastolic]	Different concept - not a regular SBP
diastolic blood pressure	8445	orthostatbp standing [diastolic]	Different concept - not a regular SBP
diastolic blood pressure	8446	orthostatic bp lying [diastolic]	Different concept - not a regular SBP
diastolic blood pressure	8448	pap [diastolic]	Different concept - pulmonary artery, which has strongly different pressure values then elsewhere
temperature	224642	temperature site	Different concept

Table D.8: Manually excluded pollution `ITEMIDs` after the per-`ITEMID` and per-variable value statistics check for the bedside monitoring variables.

Variable	**ITEMID**	**LABEL**	**Reason**
systolic blood pressure	6	abp [systolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
fio2	189	fio2 (analyzed)	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
systolic blood pressure	442	manual bp [systolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
fio2	727	vision fio2	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
systolic blood pressure	3313	bp cuff [systolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
systolic blood pressure	3315	bp left arm [Systolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
systolic blood pressure	3317	bp left leg [Systolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
systolic blood pressure	3321	bp right arm [systolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
systolic blood pressure	3323	bp right leg [systolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
systolic blood pressure	3325	bp uac [systolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
fio2	3420	fio2	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
temperature	3654	temp rectal [f]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
temperature	3655	temp skin [c]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
diastolic blood pressure	8364	abp [diastolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
diastolic blood pressure	8440	manual bp [diastolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
diastolic blood pressure	8502	bp cuff [diastolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics

Table D.8: Manually excluded pollution `ITEMIDs` after the per-`ITEMID` and per-variable value statistics check for the bedside monitoring variables.

Variable	**ITEMID**	**LABEL**	**Reason**
diastolic blood pressure	8503	bp left arm [diastolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
diastolic blood pressure	8504	bp left leg [diastolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
diastolic blood pressure	8506	bp right arm [diastolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
diastolic blood pressure	8507	bp right leg [diastolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
diastolic blood pressure	8508	bp uac [diastolic]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
temperature	8537	temp/iso/warmer [temperature, degrees c]	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
systolic blood pressure	224167	manual blood pressure systolic left	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
diastolic blood pressure	224643	manual blood pressure diastolic left	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
diastolic blood pressure	227242	manual blood pressure diastolic right	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics
systolic blood pressure	227243	manual blood pressure systolic right	Events of this type hardly occur, have an unclear unit or strongly deviating value statistics

Measurement Value Range Bounds

Table D.9: Range bounds for the bedside monitoring variables in order to exclude measurements that were clearly errors.

Variable	**Lower bound**	**Upper bound**
cvp	1	35
diastolic blood pressure	20	150
fio2	0	100
heart rate	0	220
spo2	30	100
systolic blood pressure	20	200
temperature	30	45

D.2 Descriptive Statistics

D.2.1 Feature Distributions

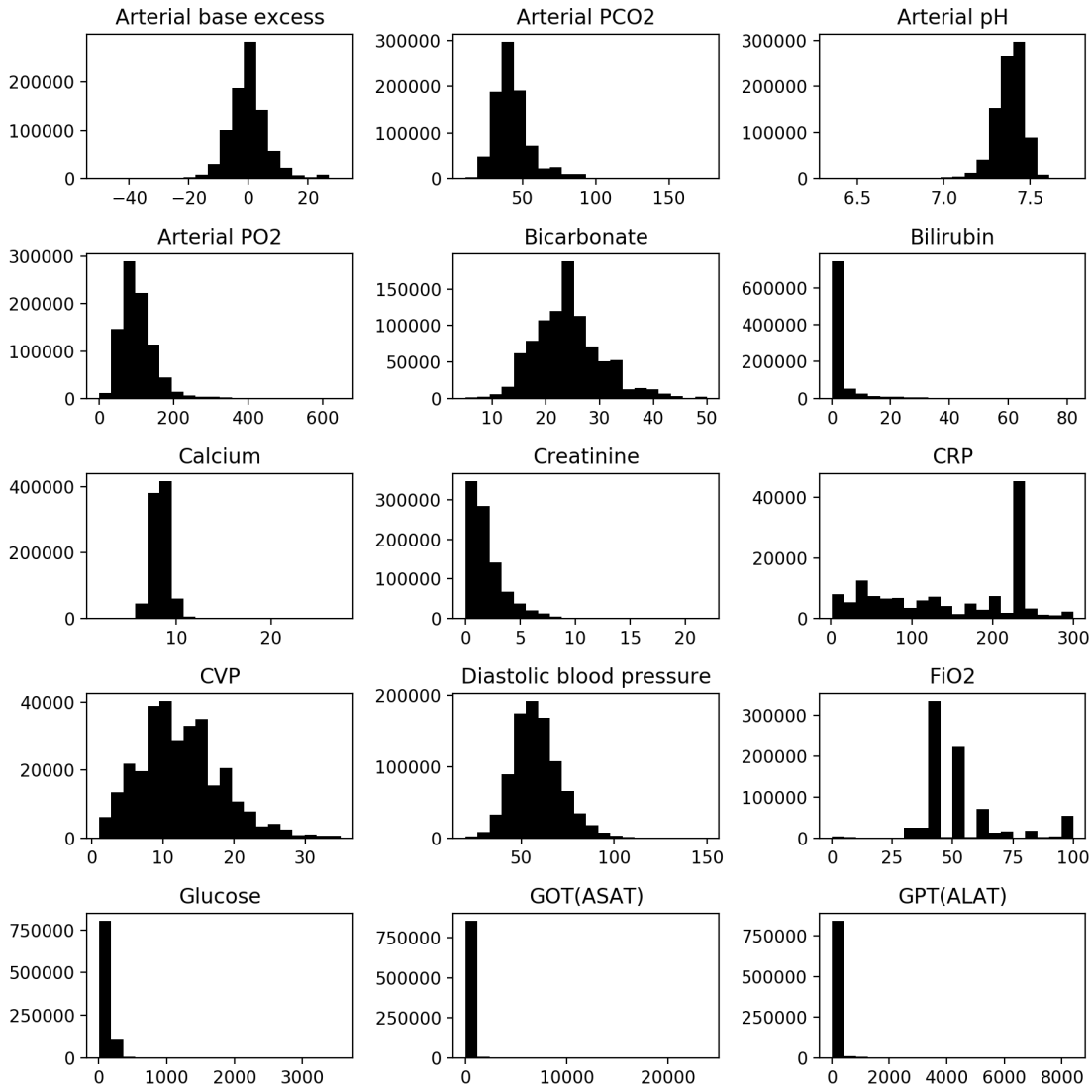

Figure D.1: First part of the histograms of feature values in the dataset used for modeling.

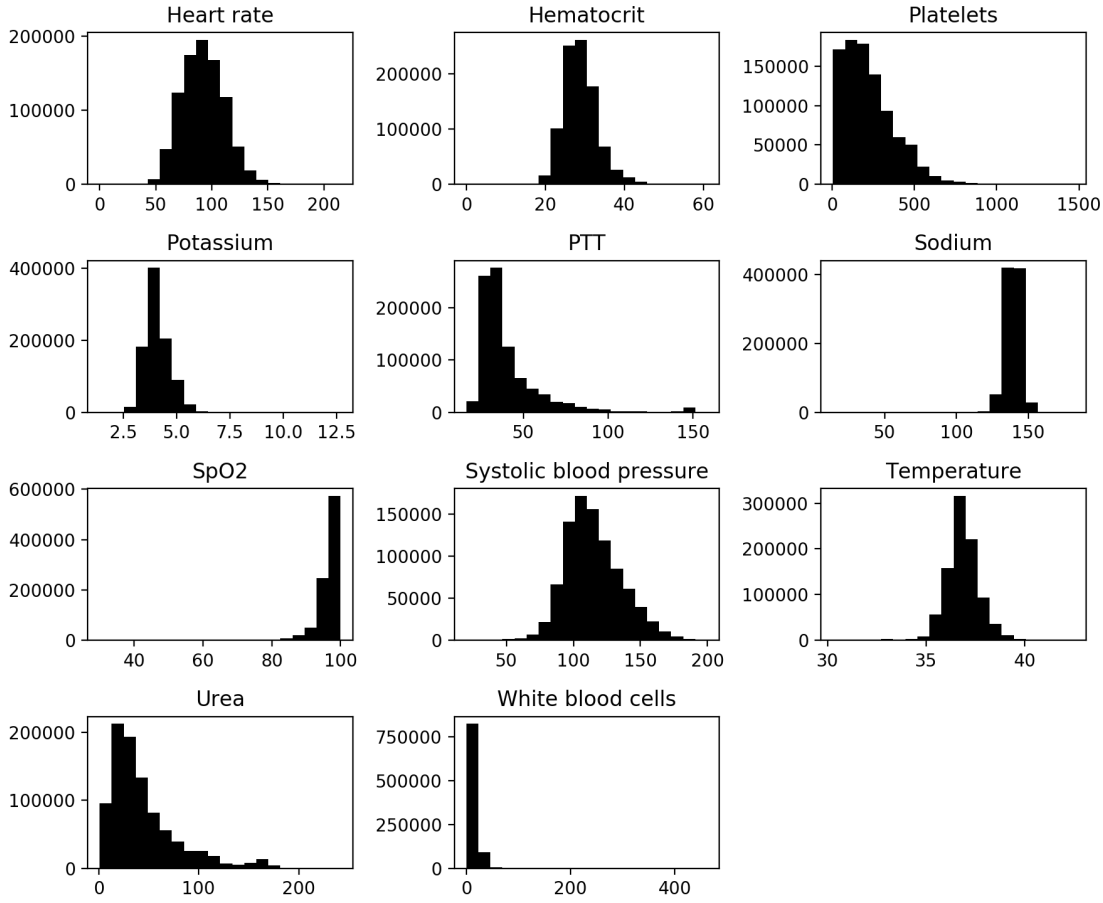

Figure D.2: Second part of the histograms of feature values in the dataset used for modeling.

D.2.2 Feature Distributions by Target Label Value

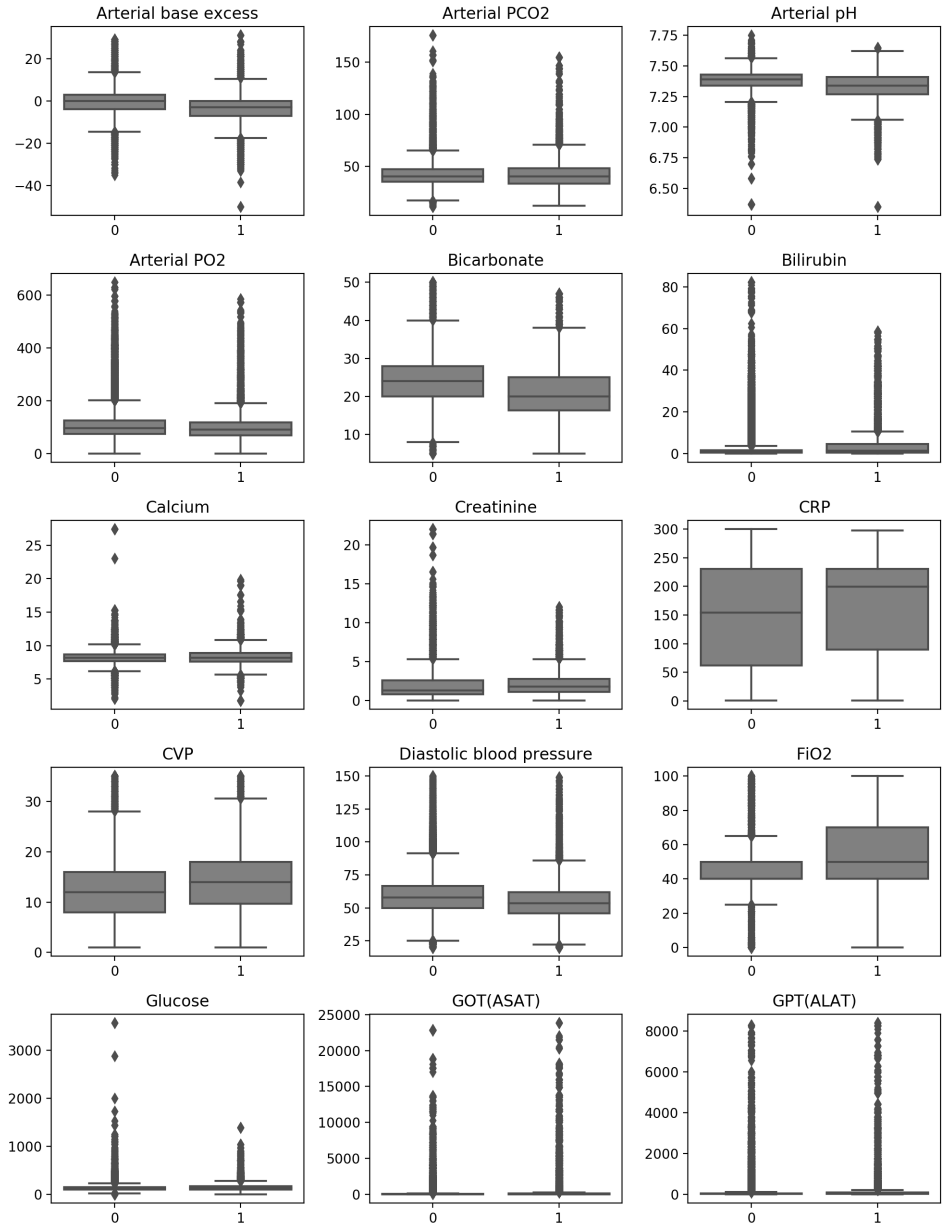

Figure D.3: First part of the boxplots of feature values per target label value in the dataset used for modeling.

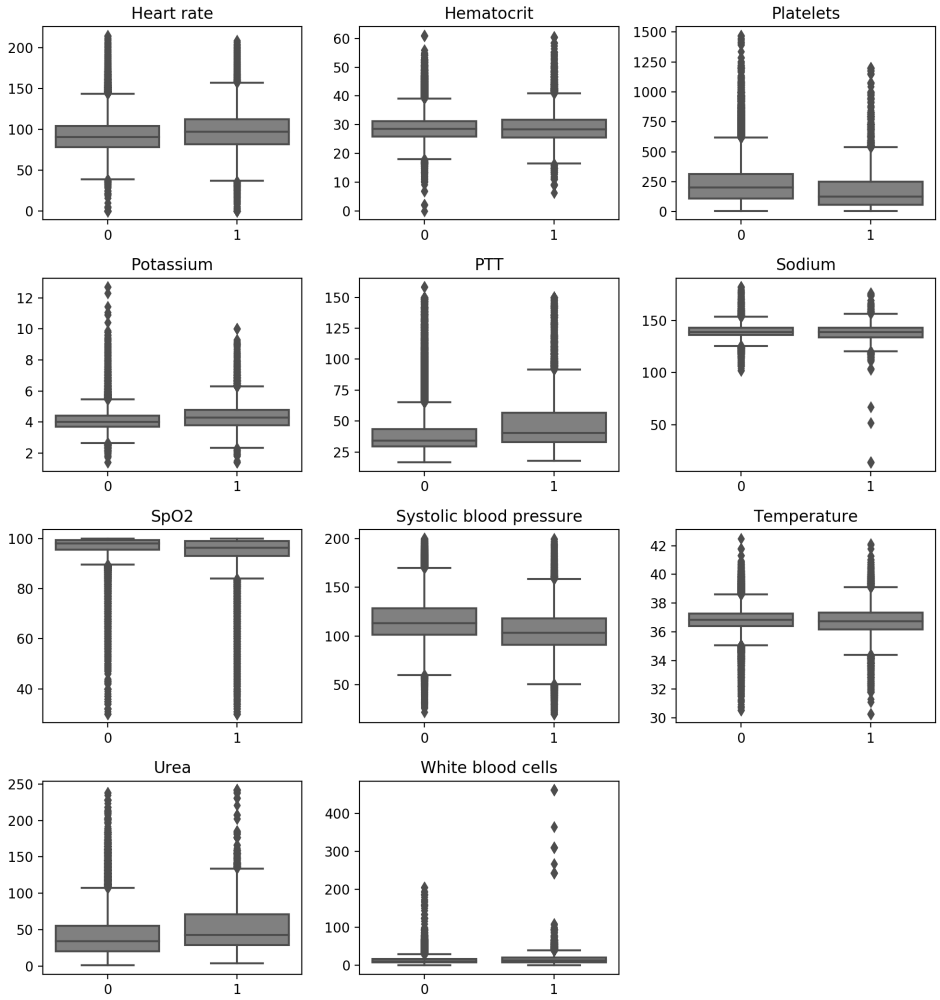

Figure D.4: Second part of the boxplots of feature values per target label value in the dataset used for modeling.

D.3 Model Tuning Details

This section includes the full results from the tested configurations during the model tuning procedures for the densely connected neural networks and the GRU networks. Recall that each combination of a network with a feature set was given a time budget of approximately 1 hour.

Table D.10: Evaluated models during the tuning of the DNN on all features.

Number of layers	Number of units in each layer	Dropout rates between layers	ROC-AUC	AUK	Computation time of fitting the model in minutes
5	[35, 77, 221, 417, 113]	[0.2934, 0.1615, 0.1503, 0.0216]	0.785132	0.142987	1.984094
9	[25, 82, 32, 309, 476, 85, 370, 273, 172]	[0.0463, 0.2097, 0.0360, 0.1455, 0.1898, 0.2455, 0.2049, 0.1496]	0.776684	0.139136	3.418462
3	[186, 84, 26]	[0.2473, 0.2897]	0.768721	0.133804	1.291333
4	[216, 199, 496, 128]	[0.2756, 0.1465, 0.1835]	0.749018	0.124496	2.111435
10	[201, 90, 101, 347, 449, 279, 342, 171, 81, 348]	[0.0677, 0.2689, 0.2479, 0.0278, 0.2669, 0.2141, 0.1554, 0.2356, 0.1763]	0.746166	0.122380	4.149024
6	[448, 66, 261, 416, 135, 75]	[0.0656, 0.1256, 0.0744, 0.0252, 0.1036]	0.743208	0.118986	2.652830
10	[237, 478, 116, 375, 345, 372, 346, 328, 291, 11]	[0.1589, 0.1256, 0.1006, 0.1868, 0.1314, 0.2208, 0.1554, 0.1737, 0.1936]	0.742828	0.120700	5.616453
7	[263, 251, 140, 269, 461, 342, 442]	[0.2756, 0.2702, 0.0100, 0.2871, 0.0412, 0.0851]	0.740079	0.120719	4.704954
10	[135, 494, 25, 330, 379, 133, 166, 483, 231, 423]	[0.0594, 0.2282, 0.0507, 0.0265, 0.2056, 0.2860, 0.0012, 0.1537, 0.2438]	0.731199	0.114089	4.792279
6	[245, 406, 82, 265, 403, 213]	[0.2997, 0.0708, 0.1190, 0.1164, 0.2010]	0.729088	0.114820	2.939404
9	[278, 416, 337, 259, 368, 244, 166, 148, 218]	[0.0666, 0.0958, 0.1092, 0.1617, 0.2044, 0.1605, 0.0742, 0.2899]	0.721967	0.110192	4.532112
9	[255, 406, 143, 385, 394, 166, 165, 209, 213]	[0.1590, 0.0918, 0.0913, 0.0335, 0.0750, 0.2753, 0.0792, 0.2153]	0.717824	0.108856	4.482829
8	[207, 395, 370, 97, 466, 316, 403, 68]	[0.0424, 0.0674, 0.2095, 0.2710, 0.1899, 0.0015, 0.1717]	0.717056	0.109554	4.498152
9	[259, 141, 450, 466, 266, 287, 157, 212, 373]	[0.0377, 0.0622, 0.0154, 0.1322, 0.0090, 0.1370, 0.1947, 0.0835]	0.711616	0.106138	4.424826
4	[350, 371, 143, 484]	[0.1593, 0.070, 0.0034]	0.706470	0.099214	2.446836

Table D.11: Evaluated models during the tuning of the DNN on the 7 features from the best PFS model.

Number of layers	Number of units in each layer	Dropout rates between layers	ROC-AUC	AUK	Computation time of fitting the model in minutes
1	[127]	N/A	0.711040	0.102817	0.982926
1	[57]	N/A	0.706301	0.100045	0.964883
1	[253]	N/A	0.704983	0.100378	1.040051
1	[90]	N/A	0.700566	0.098030	0.941535
1	[483]	N/A	0.698401	0.094935	1.030251
1	[237]	N/A	0.697966	0.096886	1.003557
7	[342, 328, 282, 374, 281, 197, 236]	[0.2802, 0.2762, 0.0750, 0.1770, 0.2682, 0.2845]	0.683472	0.086693	3.846633
6	[230, 172, 84, 374, 268, 160]	[0.0728, 0.1215, 0.1669, 0.0889, 0.2346]	0.682469	0.084337	2.779639
8	[419, 457, 30, 62, 212, 413, 477, 127]	[0.2991, 0.0039, 0.1980, 0.0540, 0.2021, 0.2219, 0.1159]	0.680697	0.087436	4.278874
4	[304, 92, 390, 348]	[0.2296, 0.1085, 0.0400]	0.676070	0.082734	1.959259
2	[336, 73]	[0.2953]	0.674813	0.083582	1.275262
2	[145, 460]	[0.1771]	0.673302	0.086950	1.264685
7	[121, 212, 267, 141, 346, 206, 87]	[0.2243, 0.1149, 0.1740, 0.0150, 0.1296, 0.1506]	0.672723	0.078645	2.414978
6	[453, 390, 59, 56, 396, 436]	[0.0007, 0.2351, 0.0125, 0.1434, 0.1085]	0.670311	0.082883	2.989748
5	[374, 478, 222, 96, 285]	[0.2308, 0.0886, 0.0447, 0.0067]	0.669988	0.081196	2.581334
6	[203, 227, 261, 152, 176, 137]	[0.1184, 0.2997, 0.0279, 0.1786, 0.1187]	0.668195	0.080573	2.091529
8	[83, 285, 357, 323, 208, 251, 294, 287]	[0.1443, 0.0266, 0.0244, 0.2650, 0.1136, 0.2045, 0.1051]	0.662324	0.077898	3.849983
3	[428, 428, 36]	[0.0612, 0.2257]	0.662237	0.076618	2.010038
5	[437, 242, 484, 331, 207]	[0.1430, 0.1827, 0.0758, 0.2826]	0.661021	0.076036	3.522748
3	[353, 265, 333]	[0.0398, 0.0393]	0.659981	0.079668	1.811388
3	[486, 108, 240]	[0.2157, 0.1473]	0.658561	0.073141	1.704547
7	[255, 255, 151, 409, 85, 299, 270]	[0.0266, 0.1024, 0.2959, 0.0413, 0.0858, 0.1681]	0.657248	0.075793	3.145350
8	[266, 168, 233, 392, 20, 57, 322, 94]	[0.1601, 0.2878, 0.2857, 0.2585, 0.0477, 0.1487, 0.2780]	0.652750	0.072764	2.545823
3	[426, 492, 400]	[0.1675, 0.0008]	0.651923	0.073974	2.582120
4	[289, 155, 469, 376]	[0.1836, 0.0591, 0.0967]	0.649377	0.071111	2.299387
3	[479, 274, 105]	[0.0699, 0.0456]	0.648371	0.073537	1.948092

Table D.12: Evaluated models during the tuning of the GRU network on all available features.

Number of layers	Number of units in each layer	Dropout rates between layers	ROC-AUC	AUK	Computation time of fitting the model in minutes
1	[16]	N/A	0.781861	0.141310	10.055467
1	[17]	N/A	0.778956	0.139485	8.099269
1	[5]	N/A	0.767229	0.133890	9.181591
2	[13, 14]	[0.0933]	0.758658	0.129112	24.553569

Table D.13: Evaluated models during the tuning of the GRU network on the 7 features from the best PFS.

Number of layers	Number of units in each layer	Dropout rates between layers	ROC-AUC	AUK	Computation time of fitting the model in minutes
2	[2, 15]	[0.0348]	0.709630	0.100522	27.584601
2	[9, 17]	[0.0059]	0.668772	0.081022	25.244659
2	[8, 9]	[0.0904]	0.661141	0.079320	26.086430

D.4 Robustness Check for ALE Plots

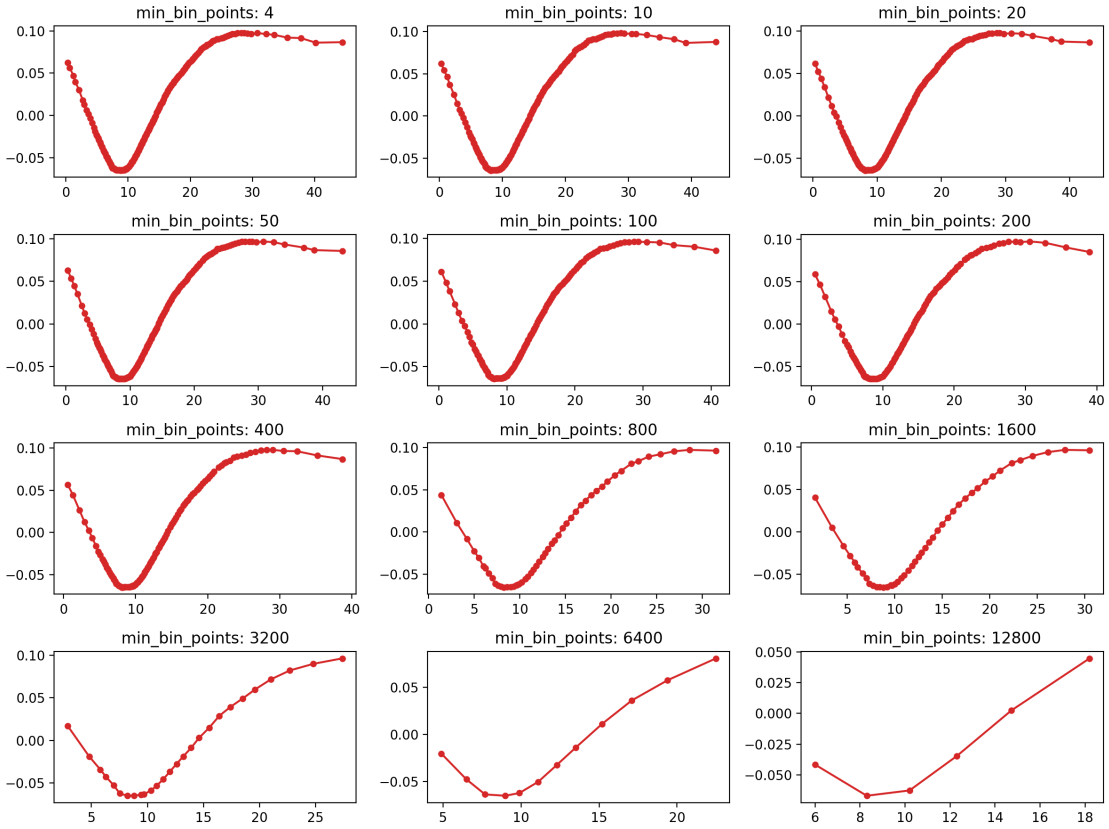

Figure D.5: Robustness check regarding the bin size of the ALE plots (this plot is for the white blood cells feature and the DNN model with 7 features). Probably because of the accumulation of the effects in the bins, the effect of this parameter on the appearance of the plot is small. It mainly affects the resolution of the figure, i.e. how many points will be on the plot. A too small bin size leads to shaky plots, while a too large bin size leads to large interpolations between points and possibly a biased picture of the actual accumulated local effects. Note that the values of x-axis vary somewhat bcause we leave out some of the outer bins to avoid large interpolations in the plots (as mentioned earlier).

D.5 Predictions for Another Patient over Time

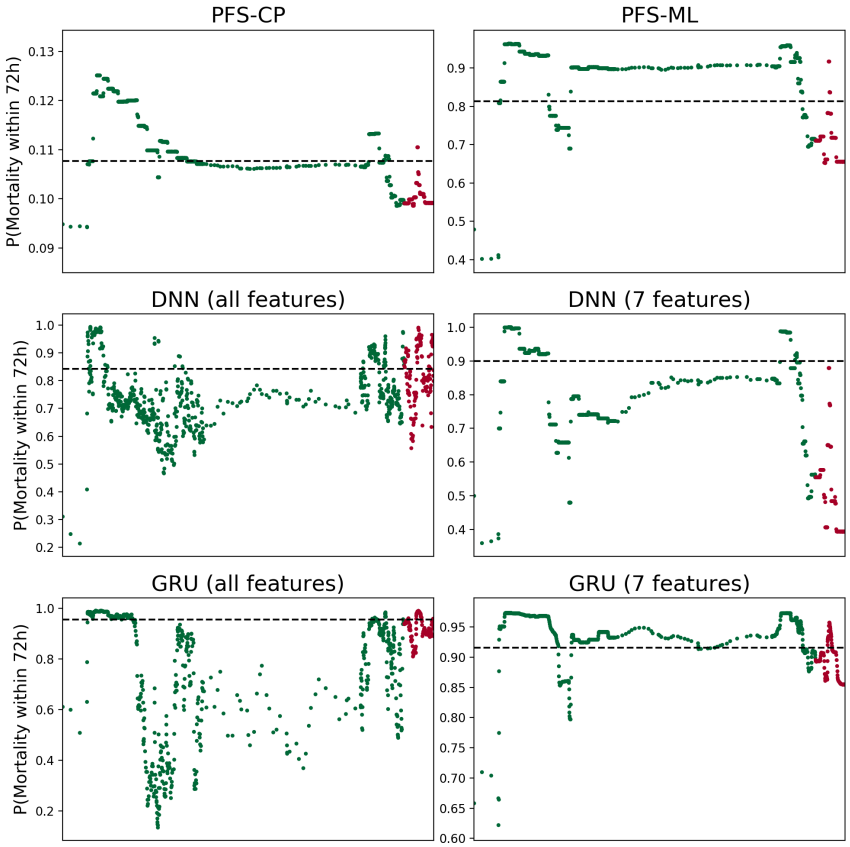

Figure D.6: Predictions of the models over time for a selected patient. A red color indicates patient mortality within 72 hours, and the dotted line indicates the optimal threshold from the Cohen's kappa curves. Clearly, the predictions of the models are not very correct for this patient.

Appendix E

Additional Details

E.1 Occurence of MAP events

Table E.1: Presence of `ITEMID`s related to mean arterial pressure (MAP) in `CHARTEVENTS`.

ITEMID	LABEL	Number of events in CHARTEVENTS
438	MAP	10
1199	HFO MAP	65
1200	High MAP Alarm	7
1201	Low MAP alarm	7
1321	map	10
2309	FEM ALINE MAP	23
2353	R FEM MAP	4
2369	fem map	43
2544	FEM ART MAP	40
2770	left radial MAP	41
2974	FEM LINE MAP	1
3067	FEM MAP	54
5680	RADIAL MAP	16
5804	sheath MAP	12
6399	ART MAP	14
6579	IR Sheath MAP	3
6605	Fem sheath MAP	27